ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Khairullin** Jméno: **Artur** Osobní číslo: **509198**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

Specializace: **Technologie internetu věcí**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Zpracování a management dat v IoT a bezdrátových sítích senzorů

Název bakalářské práce anglicky:

Data Processing and Management in IoT and Wireless Sensor Network

Pokyny pro vypracování:

Cílem projektu je seznámit se s metodami a přístupy ke zpracování a správě dat z rozsáhlých senzorových sítí. V rámci projektu vznikne aplikace (backend + frontend), která umožní základní komunikaci se senzory, uložení, zpracování a management dat (vizualizace, filtrace, data mining).
Při vypracování se řiďte následujícími pokyny:
1) vypracujte přehled dostupných systému a to jako z oblasti open-source, tak komerčních
2) po dohodě se vedoucím implementujte aplikaci pro práci s dostupnými daty;
3) aplikaci otestuje v reálném provozu a diskutujte případné nedostatky.

Seznam doporučené literatury:

[1] BANSAL, Sharu; KUMAR, Dilip. IoT ecosystem: A survey on devices, gateways, operating systems, middleware and communication. International Journal of Wireless Information Networks, 2020, 27: 340-364.
[2] JOSHVA DEVADAS, T.; THAYAMMAL, S.; RAMPRAKASH, A. IoT data management, data aggregation and dissemination. Principles of internet of things (IoT) ecosystem: insight paradigm, 2020, 385-411.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Stanislav Vítek, Ph.D. katedra radioelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **15.02.2024** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **21.09.2025**

doc. Ing. Stanislav Vítek, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

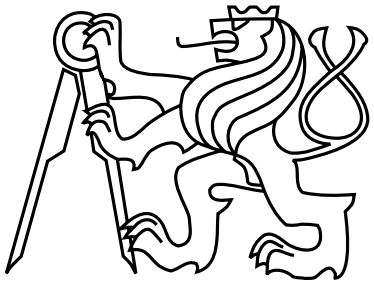
III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Podpis studenta

Datum převzetí zadání

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Zpracování a management dat v IoT a bezdrátových sítích senzorů

Artur Khairullin

Vedoucí práce: doc. Ing. Stanislav Vítek, Ph.D.

Studijní program: Softwarové inženýrství a technologie, Bakalářský

Obor: Technologie internetu věcí

19. května 2024

Poděkování

Chtěl bych vyjádřit své upřímné poděkování doc. Ing. Stanislavu Vítkovi, Ph.D., za jeho cenné rady a odborné vedení, které mi byly neocenitelnou oporou při přípravě této bakalářské práce. Dále bych rád poděkoval své rodině za podporu v mém rozhodnutí studovat v zahraničí a za poskytnutí lásky, povzbuzení a pochopení, které byly zásadní pro mé vzdělávání a osobní rozvoj. Jsem taky vděčný za všechny příležitosti, které se mi naskytly, a za všechny, kteří mě na mé cestě podporovali a pomáhali mi.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 24. 5. 2024 .....

Abstract

This bachelor's thesis deals with the creation of a modular and easily expandable Dashboard for the visualization and management of data from IoT sensors. The application enables effective collection, analysis and presentation of data locally, with the possibility of deployment using Docker. The dashboard provides users with a comprehensive view of real-time data and features messaging and alerts. The result is a customizable, extensible and intuitive application that maximizes the value of the IoT infrastructure.

Abstrakt

Tato bakalářská práce se zabývá vytvořením modulárního a snadno rozšiřitelného Dashboardu pro vizualizaci a správu dat z IoT senzorů. Aplikace umožňuje efektivní sběr, analýzu a prezentaci dat lokálně, s možností nasazení pomocí Dockeru. Dashboard poskytuje uživatelům komplexní pohled na data v reálném čase a nabízí funkci odesílání zpráv a upozornění. Výsledkem je přizpůsobitelná, rozšiřitelná a intuitivní aplikace, která maximalizuje hodnotu IoT infrastruktury.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

1.1 Motivace

V současné době jsme svědky nebývalého rozmachu technologií Internetu věcí (IoT), které transformují jak běžný život, tak různá průmyslová odvětví. S růstem počtu zařízení IoT a bezdrátových senzorových sítí (WSN) se zvyšuje i objem generovaných dat, což přináší nové výzvy v oblasti zpracování a managementu těchto dat [1]. V této souvislosti je vývoj efektivních nástrojů pro sběr, monitorování a analýzu dat z IoT zařízení nezbytný pro zajištění jejich užitečnosti a hodnoty.

Bezdrátové senzorové sítě jsou klíčovou součástí IoT, poskytují cenné informace o fyzickém světě a umožňují uživatelům získávat hlubší porozumění a kontrolu nad svým prostředím. Tyto sítě se skládají z velkého počtu senzorů, které mohou být rozmístěny v rozsáhlých geografických oblastech a v různých prostředích, od domácností po průmyslové komplexní systémy [2]. Senzory shromažďují data o různých parametrech, jako je teplota, vlhkost, tlak, pohyb, a mnoho dalších. Tyto informace pak mohou být využity pro automatizaci procesů, optimalizaci operací, předvídání údržby a zlepšení bezpečnosti a komfortu.

Avšak, s rostoucím množstvím dat se zvyšují i požadavky na jejich zpracování. Aby bylo možné data efektivně využívat, je nezbytné je správně shromažďovat, ukládat, analyzovat a vizualizovat. Právě zde se ukazuje potřeba vývoje softwarového řešení – Dashboardu, který by uživatelům poskytoval přehledné a snadno interpretovatelné informace ze senzorů v reálném čase. Takový Dashboard by měl být schopen nejen zobrazovat data, ale také umožňovat jejich analýzu a poskytovat užitečné insighty, které by mohly vést k lepšímu rozhodování a efektivnějšímu využívání zdrojů.

S přihlédnutím k těmto potřebám je motivací pro tuto bakalářskou práci vytvoření robustního, flexibilního a uživatelsky přívětivého Dashboardu, který by byl schopen zpracovávat a vizualizovat data ze senzorů v reálném čase. Záměrem je poskytnout nástroj, který by uživatelům usnadnil interpretaci velkých objemů dat a umožnil jim rychle reagovat na změny a události zaznamenané senzorovými sítěmi. Dashboard by měl být navržen s možností integrace dat z různých typů senzorů a nabízet modulární architekturu pro jednoduché rozšíření a adaptaci na konkrétní požadavky uživatelů.

1.2 Cíle

Cílem této bakalářské práce je navrhnout a implementovat Dashboard pro zpracování a management dat z IoT a bezdrátových sítí senzorů, který umožní uživatelům efektivní monitorování a analýzu dat v reálném čase. Práce se nejprve zaměří na současné trendy v oblasti IoT platforem, jejich výzvy a možnosti, které přinášejí v oblasti sběru a analýzy dat. Následně specifikujeme požadavky na Dashboard, které budou reflektovat potřeby uživatelů a technické možnosti současných IoT zařízení.

Specifikace požadavků na Dashboard bude reflektovat nejen aktuální potřeby uživatelů, ale také současné možnosti softwarového inženýrství a nejnovější postupy v oblasti vývoje webových aplikací. Hlavní důraz bude kladen na modulární design, uživatelskou přívětivost a škálovatelnost řešení.

Návrh a implementace Dashboardu bude zahrnovat vývoj uživatelského rozhraní a zároveň vývoj backendových služeb, které zajistí správnou funkcionalitu a výkon aplikace.

Po dokončení vývoje bude Dashboard podroben důkladnému testování, aby bylo zajištěno, že splňuje všechny požadavky. V závěru práce budou zhodnoceny dosažené výsledky a bude diskutován potenciál pro další rozvoj aplikace.

Kapitola 2

Analýza

2.1 Stávající IoT platformy

V rámci dynamicky se rozvíjejícího ekosystému Internetu věcí se objevuje stále více platforem, které usnadňují vývoj, nasazení a správu IoT aplikací a zařízení. Tyto platformy představují různorodé funkcionality a služby, které reflektují potřeby rozličných uživatelů a podniků. V následujících podkapitolách se podrobněji zaměříme na deset vybraných IoT platforem, které se vyznačují svým přínosem pro komunitu vývojářů a uživatelů.

2.1.1 Arduino IoT Cloud

Arduino Cloud [4] je komplexní IoT platforma, která uživatelům umožňuje snadno a rychle vytvářet, ovládat a monitorovat jejich připojené projekty. Nabízí bezproblémová IoT řešení přizpůsobená jednotlivcům, školám a podnikům. Platforma poskytuje intuitivní rozhraní, předpřipravené šablony a širokou škálu podporovaného hardwaru, což uživatelům umožňuje rychle realizovat jejich IoT nápady.

S Arduino Cloud mohou uživatelé vizualizovat a ovládat data ze senzorů odkudkoli pomocí přizpůsobitelných dashboardů. Platforma také nabízí bezpečnou funkci Over-the-Air (OTA) pro vzdálené aktualizace zařízení a poskytuje upozornění v reálném čase, aby uživatelé byli informováni o stavu a událostech svých zařízení. Uživatelé mohou vytvářet a sdílet neomezený počet dashboardů s ostatními, což usnadňuje spolupráci a prezentaci jejich projektů.

Klíčové vlastnosti a možnosti Arduino Cloud:

1. **Rychlá tvorba projektů.** Uživatelé mohou rychle vytvářet IoT projekty pomocí intuitivního rozhraní platformy a předpřipravených šablon.

2. **Vizualizace a ovládání dat ze senzorů.** Arduino Cloud umožňuje uživatelům vizualizovat a ovládat data ze senzorů odkudkoli pomocí přizpůsobitelných dashboardů.

3. **Široká škála podporovaného hardwaru.** Platforma je kompatibilní s různými deskami Arduino, deskami založenými na ESP a zařízeními třetích stran, která podporují Python, MicroPython, JavaScript nebo Node-RED.

4. **Aktualizace Over-the-Air.** Uživatelé mohou vzdáleně aktualizovat svá zařízení pomocí zabezpečené funkce OTA platformy.

5. **Upozornění v reálném čase.** Platforma poskytuje upozornění v reálném čase, aby uživatelé byli informováni o stavu a událostech svých zařízení.

6. **Sdílené dashboardy.** Uživatelé mohou vytvářet a sdílet neomezený počet dashboardů s ostatními, což usnadňuje spolupráci a prezentaci jejich projektů.

7. **Integrace API.** Arduino Cloud nabízí API, která umožňují integraci s dalšími službami a platformami, což rozšiřuje možnosti IoT projektů.

8. **Mobilní aplikace.** Aplikace IoT Remote umožňuje uživatelům ovládat a monitorovat jejich projekty na cestách, poskytuje plnou kontrolu odkudkoli na světě.

9. **Řešení pro různé skupiny uživatelů.** Arduino Cloud nabízí přizpůsobené plány pro jednotlivce (plán Maker), školy (plán School) a podniky (plán Enterprise), které vyhovují jejich specifickým potřebám a požadavkům.

Omezení a limity:

1. **Kompatibilita hardwaru.** I když Arduino Cloud podporuje širokou škálu hardwaru, uživatelé se mohou potřebovat ujistit, že jejich konkrétní zařízení jsou s platformou kompatibilní.

2. **Doba uchovávání dat.** Doba uchovávání dat se může lišit v závislosti na zvoleném plánu, což může mít dopad na historická data dostupná pro analýzu.

3. **Ceny.** Ačkoli Arduino Cloud nabízí bezplatný plán, přístup k pokročilým funkcím a možnostem může vyžadovat upgrade na placené plány, což by mohlo být pro některé uživatele omezením.

4. **Křivka učení.** I když se Arduino Cloud snaží být uživatelsky přívětivý, uživatelé s omezenými zkušenostmi v oblasti IoT a programování mohou čelit křivce učení při nastavování a konfiguraci svých projektů.

2.1.2 Particle

Particle [13] je komplexní platforma IoT typu Platform-as-a-Service (PaaS), která umožňuje společnostem snadno vytvářet a spravovat připojené produkty. Particle poskytuje integrovanou infrastrukturu, která zahrnuje software, konektivitu a hardware, aby společnosti mohly urychlit svůj růst, optimalizovat provoz a měnit celá odvětví k lepšímu.

Klíčové vlastnosti a možnosti Particle:

1. **Plně integrovaná infrastruktura.** Particle zajišťuje bezproblémovou spolupráci softwaru, konektivity a hardwaru IoT v jedné infrastruktuře od edge po cloud.

2. **Nekonečná přizpůsobitelnost.** Vše v Particle je přeprogramovatelné a rekonfigurovatelné, takže můžete vyvíjet vlastní aplikace pro váš přesný případ použití.

3. **Důvěryhodnost.** Více než 270 000 vývojářů a polovina společností z žebříčku Fortune 500 vytváří připojená zařízení s Particle.

4. **Rozsáhlá dokumentace a podpora pro vývojáře.** Particle poskytuje dokumentaci, návody, referenční materiály a nástroje pro snadný vývoj IoT zařízení.

Omezení a limity:

1. **Závislost na platformě Particle.** Při použití platformy Particle jste do určité míry závislí na jejich infrastruktuře a ekosystému.

2. **Náklady.** I když Particle nabízí různé cenové plány, náklady na používání jejich platformy mohou být neúnosné, zejména při škálování.

3. **Omezení hardwaru.** Přestože je Particle vysoce přizpůsobitelný, stále může existovat omezení týkající se podporovaného hardwaru a integrace s některými specifickými zařízeními nebo senzory.

2.1.3 ThingsBoard

ThingsBoard [17] je open-source platforma IoT, která umožňuje sběr, zpracování a vizualizaci dat a správu zařízení. Nabízí konektivitu zařízení prostřednictvím standardních protokolů IoT, jako jsou MQTT, CoAP a HTTP, a podporuje nasazení v cloudu i on-premises. ThingsBoard kombinuje škálovatelnost, odolnost proti chybám a výkon, takže nikdy nedojde ke ztrátě dat.

Klíčové vlastnosti a možnosti ThingsBoard:

1. **Zřizování a správa zařízení a aktiv.** Bezpečné zřizování, monitorování a ovládání IoT entit pomocí bohatých serverových API. Definování vztahů mezi zařízeními, aktivy, zákazníky nebo jakýmikoli jinými entitami.

2. **Sběr a vizualizace dat.** Shromažďování a ukládání telemetrických dat škálovatelným a odolným způsobem. Vizualizace dat pomocí vestavěných nebo vlastních widgetů a flexibilních dashboardů. Sdílení dashboardů se zákazníky.

3. **Zpracování a reakce.** Definování řetězců pravidel zpracování dat. Transformace a normalizace dat ze zařízení. Vyvolávání alarmů na příchozí telemetrické události, aktualizace atributů, nečinnost zařízení a akce uživatelů.

4. **Mikroslužby.** Vytvoření clusteru ThingsBoard pro získání maximální škálovatelnosti a odolnosti proti chybám s architekturou mikroslužeb. Podpora nasazení v cloudu i on-premises.

5. **Dashboardy IoT v reálném čase.** Vytváření bohatých dashboardů IoT pro vizualizaci dat a vzdálené ovládání zařízení v reálném čase. Více než 30 přizpůsobitelných widgetů umožňuje vytvářet vlastní dashboardy pro koncové uživatele pro většinu případů použití IoT.

6. **IoT Rule Engine.** Zpracování příchozích dat ze zařízení pomocí flexibilních řetězců pravidel založených na atributech entit nebo obsahu zpráv. Předávání dat do externích systémů nebo spouštění alarmů pomocí vlastní logiky. Konfigurace komplexních řetězců oznámení pro alarmy. Rozšíření funkčnosti na straně serveru nebo manipulace se zařízeními pomocí vysoce přizpůsobitelných pravidel. Definování logiky aplikace pomocí návrháře řetězců pravidel drag-n-drop.

7. **Správa alarmů.** Poskytování možnosti vytvářet a spravovat alarmy související s entitami: zařízeními, aktivy, zákazníky atd. Monitorování alarmů v reálném čase a šíření alarmů do hierarchie souvisejících entit. Vyvolávání alarmů při odpojení zařízení nebo událostech nečinnosti.

8. **Přizpůsobení a integrace.** Rozšíření výchozí funkčnosti platformy pomocí přizpůsobitelných řetězců pravidel, widgetů a implementací transportu. Kromě podpory MQTT, CoAP a HTTP mohou uživatelé ThingsBoard používat vlastní implementace transportu nebo přizpůsobit chování stávajících protokolů.

Omezení a limity:

1. **Škálovatelnost.** I když ThingsBoard podporuje horizontální škálování pomocí architektury mikroslužeb, stále mohou existovat určitá omezení v závislosti na konkrétním nasazení a infrastruktuře.

2. **Výkon.** Výkon platformy ThingsBoard může být ovlivněn různými faktory, jako je počet připojených zařízení, frekvence příchozích dat a složitost pravidel zpracování.

3. **Zabezpečení.** Přestože ThingsBoard poskytuje různé funkce zabezpečení, jako je šifrování transportu a správa přihlašovacích údajů zařízení, implementace komplexní bezpečnostní strategie může vyžadovat další úsilí a odborné znalosti.

4. **Křivka učení.** Pro efektivní využití všech funkcí a možností přizpůsobení platformy ThingsBoard mohou být nutné určité technické znalosti a zkušenosti s IoT.

2.1.4 thethings.iO

thethings.iO [16] je komplexní platforma IoT, která poskytuje bezproblémová řešení pro různá odvětví, včetně chytrých měst, chytrého průmyslu, chytrých budov a digitalizace produktů. Platforma se zaměřuje na zvýšení efektivity, konektivity a inovací v ekosystému IoT.

Klíčové vlastnosti a možnosti thethings.iO:

1. **Vlastní dashboardy IoT.** Vytváření personalizovaných dashboardů pro získání přehledu o ekosystému IoT v reálném čase.

2. **Neomezené ukládání dat.** Bezproblémové ukládání a přístup k neomezenému množství dat pro informované rozhodování.

3. **Zřizování a licencování zařízení.** Snadná správa zřizování a licencí zařízení pro zjednodušení zkušeností s IoT.

4. **Výkonné integrace.** Bezproblémová integrace se stávajícími systémy a maximalizace potenciálu ekosystému IoT.

5. **Aplikace pro různá odvětví.** Platforma thethings.iO nabízí řešení pro různá odvětví, jako je chladírenský řetězec, sledování aktiv, inženýrství, zemědělství, chytré domácnosti, chytrá výstavba, chytré kapaliny a maloobchod.

6. **Moduly a konektivita.** Platforma poskytuje moduly, jako jsou Blinders pro maloobchodníky s IoT a EasyGateway, a podporuje integrovaná zařízení a integrované technologie LPWAN pro zajištění konektivity.

7. **Cenové možnosti.** thethings.iO nabízí cenové plány SaaS a možnosti on-premise IoT platformy, které vyhovují různým potřebám zákazníků.

8. **Rozsáhlé zdroje a podpora.** Platforma poskytuje rychlý start, cloudový kód, návody k používání panelu, zdroje na GitHubu, protokoly a serializaci a mediální zdroje pro podporu uživatelů a vývojářů.

Omezení a limity:

1. **Závislost na platformě.** Při používání platformy thethings.iO může existovat určitá závislost na jejich infrastruktuře a ekosystému.

2. **Škálovatelnost.** I když platforma podporuje škálování, stále mohou existovat určitá omezení v závislosti na konkrétním nasazení a infrastruktuře.

3. **Integrace s proprietárními systémy.** Integrace platformy thethings.iO s některými proprietárními nebo staršími systémy může vyžadovat další úsilí a přizpůsobení.

2.1.5 SiteWhere

SiteWhere [15] je open-source platforma IoT, která poskytuje systém pro sběr, ukládání, zpracování a integraci dat ze zařízení. Platforma je navržena tak, aby byla škálovatelná a mohla zpracovávat miliardy událostí zařízení denně.

Klíčové vlastnosti a možnosti SiteWhere:

1. **Platforma IoT serveru.** Poskytuje serverovou platformu založenou na osvědčených technologiích, která funguje jako kontroler pro zpracování dat ze zařízení. Server lze nainstalovat na místní počítač nebo spustit v cloudu.

2. **Dlouhodobé uchovávání dat.** Zajišťuje dlouhodobé uchovávání dat odeslaných ze zařízení. Historická data událostí zařízení jsou cenná a SiteWhere nabízí platformu, kde data nejsou nikdy smazána, bez ohledu na objem událostí.

3. **Pokročilý systém komunikace se zařízeními.** Poskytuje pokročilý systém komunikace se zařízeními, který umožňuje řízení celého životního cyklu registrace zařízení, odesílání příkazů na základě typu hardwaru, příjem datových odpovědí a jejich agregaci.

4. **Správa zařízení.** Poskytuje kompletní model správy zařízení. Specifikace zařízení umožňují deklarovat třídy zařízení spolu s metadaty, která poskytují rozšířený kontext pro zařízení. Zařízení jsou vytvářena na základě specifikací spolu s jedinečnými informacemi, jako je hardwarové ID a metadata specifická pro zařízení.

5. **Integrace.** Integruje se s frameworky integrace třetích stran, jako je Mule AnyPoint, což umožňuje, aby data událostí spouštěla komplexní interakce, jako je přidávání dat do Salesforce nebo upozorňování na poplachové stavy pomocí textových zpráv generovaných Twilio.

6. **Zabezpečení.** Chrání informace omezením přístupu k datům na základě osvědčeného systému správy uživatelů. Systém lze nakonfigurovat tak, aby používal externí zdroje dat identit, jako jsou úložiště LDAP.

7. **Základní technologie platformy.** SiteWhere závisí na mnoha podpůrných open-source technologiích, jako jsou Apache Tomcat, Spring Framework, Spring Security, Hazelcast a různé technologie ukládání dat (MongoDB, Apache HBase, InfluxDB).

Omezení a limity:

1. **Složitost.** SiteWhere je komplexní platforma s mnoha funkcemi a možnostmi integrace. Pochopení a efektivní využití všech funkcí může vyžadovat značné technické znalosti a zkušenosti.

2. **Závislost na externích technologiích.** SiteWhere závisí na různých externích technologiích a frameworcích. Změny nebo problémy v těchto základních technologiích mohou mít dopad na funkčnost a výkon SiteWhere.

3. **Křivka učení.** Vzhledem k rozsáhlým funkcím a flexibilitě SiteWhere může existovat významná křivka učení pro efektivní využití platformy, zejména pro uživatele s omezenými zkušenostmi s IoT a souvisejícími technologiemi.

4. **Zastavený vývoj.** Zdá se, že vývoj platformy SiteWhere byl zastaven a platforma již není aktivně vyvíjena. To může vést k problémům s kompatibilitou, zabezpečením a podporou v budoucnu.

2.1.6 Ubidots

Ubidots [19] je výkonná a uživatelsky přívětivá platforma Industrial IoT, která společnostem umožňuje maximalizovat provozuschopnost strojů, zlepšit efektivitu a řídit digitální transformaci. Platforma poskytuje komplexní sadu nástrojů a funkcí pro připojení, monitorování a řízení průmyslových aktiv, což organizacím usnadňuje přijetí Průmyslu 4.0.

Klíčové vlastnosti a možnosti Ubidots:

1. **Rychlé a snadné připojení.** Rychlé připojení strojů k Ubidots pomocí HTTP nebo MQTT konektorů nebo více než 60 IoT integrací. Intuitivní nástroje umožňují rychle odemknout praktické poznatky a překonat datové sily.

2. **Vizualizace v reálném čase.** Vylepšení SCADA systémů na dílně o cloudové vizualizace v kombinaci s více než 20 widgety pro pokročilé monitorování stavu, správu energie nebo inteligentní údržbu.

3. **Interoperabilita a automatizace.** Spouštění vlastních akcí na základě dat ze strojů, jako je vytváření návštěv údržby prostřednictvím CMMS softwaru, aktualizace úrovní zásob v MES nebo jednoduché zasílání e-mailových, hlasových nebo SMS upozornění.

4. **Přizpůsobení a branding.** Přizpůsobení vzhledu a fungování IoT aplikace. Segmentace zařízení, strojů a dashboardů do více uživatelských skupin s podrobnými oprávněními - vše pod vlastní značkou.

5. **Škálovatelnost a flexibilita.** Ubidots je navržen tak, aby byl škálovatelný a přizpůsobitelný rostoucím potřebám organizací. Platforma je nezávislá na odvětví a může být snadno integrována do stávajících systémů a infrastruktury.

6. **Pokročilá analýza dat.** Využití vestavěných nástrojů pro analýzu dat k získání hodnotných poznatků z průmyslových dat. Identifikace trendů, anomálií a oblastí pro zlepšení.

7. **Zabezpečení a dodržování předpisů.** Ubidots klade velký důraz na zabezpečení dat a dodržování předpisů. Platforma je v souladu s průmyslovými standardy a poskytuje robustní bezpečnostní funkce pro ochranu citlivých dat.

Omezení a limity:

1. **Omezení vlastního vývoje.** I když Ubidots poskytuje rozsáhlé možnosti přizpůsobení, některé vysoce specializované případy použití mohou vyžadovat vlastní vývoj nebo integraci, což může být časově a zdrojově náročné.

2. **Náklady.** Náklady na používání platformy Ubidots se mohou lišit v závislosti na počtu připojených zařízení, objemu dat a požadovaných funkcích.

2.1.7 ThingSpeak

ThingSpeak [18] je analytická platforma IoT, která umožňuje shromažďovat, vizualizovat a analyzovat živé datové toky v cloudu. Platforma poskytuje nástroje pro snadné odesílání dat ze zařízení, vytváření okamžitých vizualizací živých dat a odesílání upozornění. ThingSpeak je navržen tak, aby urychlil vývoj IoT projektů a umožnil uživatelům rychle realizovat jejich nápady.

Klíčové vlastnosti a možnosti ThingSpeak:

1. **Sběr dat v soukromých kanálech.** Odesílání dat ze senzorů soukromě do cloudu pomocí zabezpečených kanálů.

2. **Sdílení dat pomocí veřejných kanálů.** Možnost sdílení dat s ostatními prostřednictvím veřejných kanálů pro spolupráci a prezentaci projektů.

3. **RESTful a MQTT API.** Snadná integrace zařízení a aplikací pomocí standardních komunikačních protokolů.
4. **Analytické nástroje MATLAB.** Pokročilá analýza a vizualizace dat pomocí integrovaných nástrojů MATLAB přímo na platformě ThingSpeak.
5. **Plánování událostí.** Možnost plánování akcí a úloh na základě příchozích dat nebo časových plánů.
6. **Upozornění.** Konfigurace upozornění a oznámení na základě definovaných prahových hodnot nebo podmínek.
7. **Integrace aplikací.** Snadná integrace s externími aplikacemi a službami pro rozšíření funkčnosti a automatizaci pracovních postupů.

Omezení a limity:

1. **Omezení bezplatného plánu.** Bezplatný plán ThingSpeak má omezení týkající se počtu zpráv, intervalu aktualizace a počtu kanálů. Pro pokročilé použití mohou být nutné placené plány.
2. **Omezení vlastního vývoje.** I když ThingSpeak poskytuje rozsáhlé možnosti přizpůsobení, některé vysoce specializované případy použití mohou vyžadovat vlastní vývoj nebo integraci, což může být časově a zdrojově náročné.

2.1.8 Blynk

Blynk [5] je nízko-kódová softwarová platforma IoT pro podniky a vývojáře, která umožňuje rychlý vývoj a poskytování výjimečných zážitků ze zařízení, od prototypu až po nasazení v podniku. Platforma poskytuje plně integrovanou sadu softwarových nástrojů IoT, včetně zřizování zařízení, vizualizace dat ze senzorů, vzdáleného ovládání pomocí mobilních a webových aplikací, aktualizací firmwaru Over-The-Air, privátního cloudu, analýzy dat, správy uživatelů a přístupu, upozornění a automatizace.

Klíčové vlastnosti a možnosti Blynk:

1. **No-code aplikace.** Vytváření aplikací IoT připravených pro zákazníky bez kódování pomocí drag-and-drop editorů pro iOS a Android.
2. **Zřizování zařízení.** Využití špičkového zřizování zařízení s asistencí BLE přímo z aplikací Blynk.
3. **Network Co-Processor (NCP).** Proměna produktů v chytrá zařízení pomocí Blynk.NCP, který pokrývá vše kromě vývoje hardwaru.
4. **Blueprints.** Návody, příklady kódu firmwaru a uživatelská rozhraní dashboardů pro urychlení vývoje.
5. **Podpora širokého hardwaru.** Kompatibilita s ESP32, Arduino, Raspberry Pi, Texas Instruments, Particle a více než 400 dalšími modely hardwaru.

6. **Zabezpečený cloud.** Připojení zařízení k zabezpečenému cloudu připravenému k okamžitému použití.
7. **Aktualizace Over-The-Air.** Odesílání aktualizací firmwaru na jedno zařízení nebo správa škálovaných nasazení.
8. **Flexibilní API.** Firmware a REST API pro výměnu dat mezi hardwarem a aplikacemi.

Omezení a limity:

1. **Závislost na platformě Blynk.** Při používání platformy Blynk existuje určitá závislost na jejich infrastruktuře a ekosystému.
2. **Omezení bezplatného plánu.** Bezplatný plán Blynk může mít omezení týkající se počtu zařízení, datových bodů a funkcí. Pro pokročilé použití mohou být nutné placené plány.
3. **Požadavky na odborné znalosti.** I když Blynk zjednodušuje vývoj IoT, stále mohou být nutné určité technické znalosti a zkušenosti s elektronikou a programováním.

2.1.9 Adafruit IO

Adafruit IO [3] je snadno použitelná platforma IoT, která umožňuje rychle a snadno připojit projekty k internetu věcí. Poskytuje intuitivní způsob interakce s hardwarovými komponentami přes internet, což z ní dělá atraktivní volbu pro nadšence, vývojáře a tvůrce, kteří chtějí experimentovat s IoT.

Klíčové vlastnosti a možnosti Adafruit IO:

1. **Jednoduchost použití.** Adafruit IO je navržen tak, aby byl snadno použitelný a přístupný i pro začátečníky. Poskytuje přímočarý způsob, jak začít s IoT projekty bez složitého nastavování nebo konfigurace.
2. **Datalogging.** Platforma umožňuje snadné zaznamenávání a ukládání dat ze senzorů a zařízení. Data lze vizualizovat pomocí grafů a dashboardů pro sledování a analýzu.
3. **Komunikace s mikrokontroléry.** Adafruit IO umožňuje bezproblémovou komunikaci s různými mikrokontroléry a hardwarovými platformami, což usnadňuje interakci se zařízeními přes web.
4. **Integrace s produkty Adafruit.** Platforma je navržena tak, aby bezproblémově fungovala s širokou škálou hardwarových komponent a produktů prodávaných společností Adafruit, což usnadňuje začlenění do IoT projektů.

Omezení a limity:

1. **Pokročilé funkce.** Ve srovnání s některými komplexnějšími platformami IoT může Adafruit IO postrádat některé pokročilé funkce a možnosti přizpůsobení.
2. **Závislost na ekosystému Adafruit.** Přestože Adafruit IO podporuje různý hardware, je optimalizován a nejlépe funguje s produkty a komponentami Adafruit. Použití hardwaru od jiných výrobců může vyžadovat další úsilí nebo přizpůsobení.

2.1.10 Cayenne IoT

Cayenne IoT [6] je výkonná a uživatelsky přívětivá platforma, která zjednodušuje proces vytváření a správy IoT projektů. Vyvinutá společností myDevices, Cayenne IoT funguje jako most mezi hardwarem a cloudem, poskytuje komplexní řešení pro začátečníky i zkušené vývojáře k vytváření IoT aplikací. Platforma je známá svou všestranností a podporuje širokou škálu zařízení, senzorů a akčních členů.

Klíčové vlastnosti a možnosti Cayenne IoT:

1. **Kompatibilita zařízení.** Cayenne IoT podporuje širokou škálu zařízení, od populárních mikrokontrolérů jako Arduino a Raspberry Pi až po různé senzory, akční členy a komunikační moduly.
2. **Rozhraní drag-and-drop.** Intuitivní rozhraní drag-and-drop umožňuje uživatelům snadno vytvářet vlastní dashboardy přidáváním widgetů pro různá zařízení a datové body. Tento vizuální přístup zjednodušuje proces vytváření komplexních IoT projektů.
3. **Vzdálené monitorování a ovládání.** Cayenne umožňuje uživatelům vzdáleně monitorovat a ovládat připojená zařízení. Platforma poskytuje centralizované rozhraní pro správu IoT aplikací.
4. **Přizpůsobitelné triggery a upozornění.** Uživatelé mohou nastavit triggery a upozornění na základě specifických podmínek nebo událostí. Tato funkce umožňuje automatizované reakce na změny v prostředí.

Omezení a limity:

1. **Omezení bezplatného plánu.** Bezplatný plán Cayenne IoT má omezení týkající se počtu zařízení, datových bodů a některých pokročilých funkcí. Pro rozsáhlejší projekty mohou být nutné placené plány.
2. **Omezené možnosti vlastního vývoje.** I když Cayenne IoT poskytuje flexibilitu prostřednictvím rozhraní drag-and-drop a integrací, může být omezující pro uživatele, kteří vyžadují vysoce přizpůsobená nebo specializovaná řešení.

2.2 Funkční požadavky

1. **FR01: Získávání dat.** Systém musí být schopen automaticky shromažďovat data ze senzorů v reálném čase a importovat je do Dashboardu.
2. **FR02: Zpracování dat.** Dashboard bude zpracovávat přijatá data pro analýzu, což zahrnuje agregaci, filtrování a transformaci dat do použitelných informací.
3. **FR03: Vizualizace dat.** Systém musí poskytovat vizualizační nástroje, jako jsou grafy, tabulky a mapy, pro grafické zobrazení dat uživatelům.
4. **FR04: Výstrahy v reálném čase.** Dashboard musí mít schopnost konfigurovat a odesílat výstrahy uživatelům v případě, že data překročí definované prahové hodnoty. (messengery)

5. **FR05: Uživatelské přizpůsobení.** Uživatelé by měli mít možnost přizpůsobit si layout a jaké metriky jsou zobrazeny na jejich osobních dashboardových panelech.
6. **FR06: Integrace více senzorů.** Systém musí být kompatibilní s různými typy senzorů a umožňovat jejich snadnou integraci do dashboardu.
7. **FR07: Analýza historických dat.** Dashboard musí umožňovat uživatelům prohlížet a analyzovat historická data a trendy.
8. **FR08: Generování reportů.** Systém musí poskytovat funkci pro generování a export reportů v různých formátech, jako jsou PDF nebo Excel.
9. **FR09: Konfigurace systému.** Musí být možné konfigurovat a spravovat systémové nastavení, včetně přidávání nebo odstraňování senzorů, nastavení prahových hodnot a plánování údržby.
10. **FR10: Přístup přes API.** Systém musí poskytovat API pro umožnění integrace s externími aplikacemi a službami.
11. **FR11: Zálohování a obnova.** Musí být zajištěny mechanismy pro zálohování a obnovu systému a dat pro případ selhání systému nebo datových ztrát.
12. **FR12: Přístupnost.** Systém musí být přístupný přes standardní webové prohlížeče a kompatibilní s běžnými operačními systémy a zařízeními.

2.3 Kvalitativní požadavky

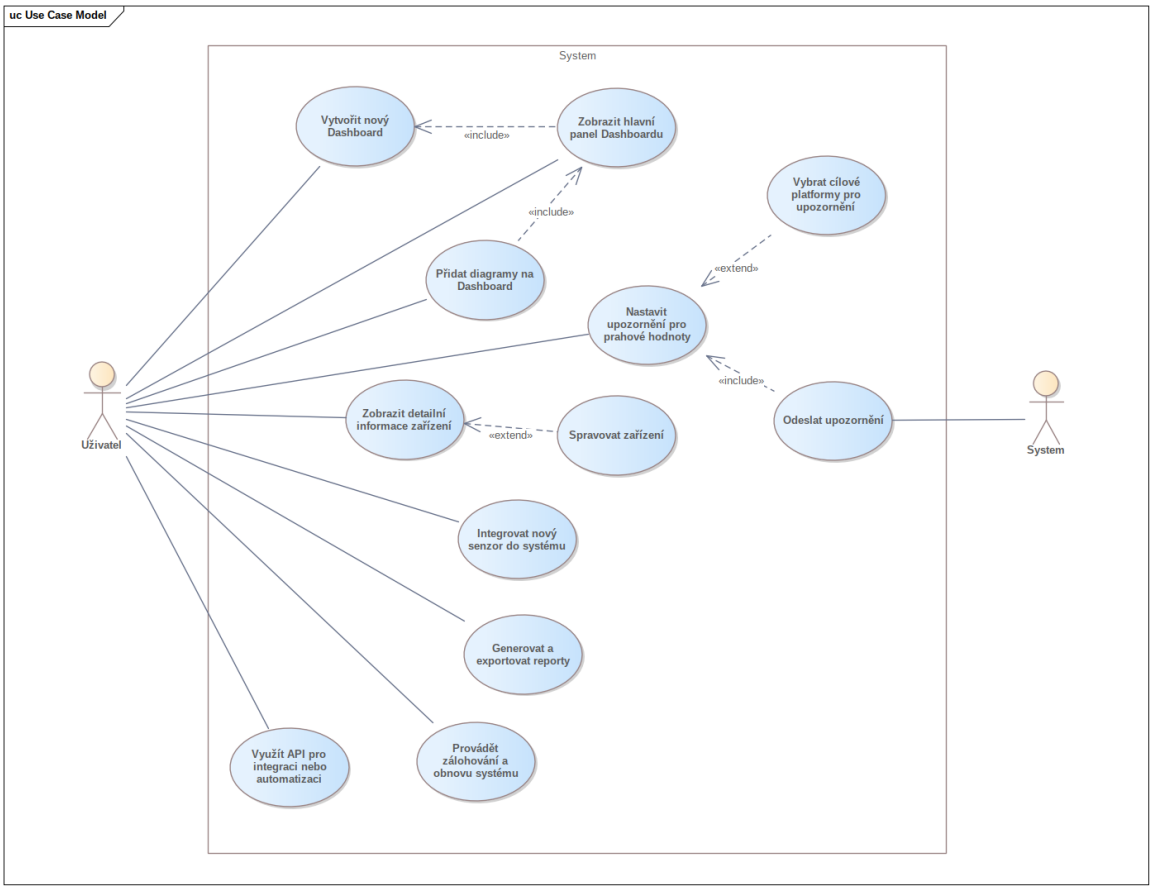
1. **QR01: Uživatelská přívětivost.** Dashboard musí být intuitivní a snadno použitelný pro osoby bez technického vzdělání, s jasně definovanými funkcemi a možností snadné navigace.
2. **QR02: Přizpůsobitelnost.** Dashboard musí umožňovat uživatelům přizpůsobení zobrazení, včetně výběru, které datové body a metriky jsou pro ně nejdůležitější a jak jsou prezentovány.
3. **QR03: Výkon a škálovatelnost.** Dashboard musí být schopen efektivně zpracovávat a zobrazovat velké objemy dat z více senzorů v reálném čase bez významného zpoždění nebo snížení výkonu.
4. **QR04: Aktualizace v reálném čase.** Dashboard musí poskytovat aktualizace dat v reálném čase, aby uživatelé mohli okamžitě vidět změny a reagovat na nové informace.
5. **QR05: Modulární architektura.** Systém by měl být navržen jako sada nezávislých modulů, které lze snadno aktualizovat nebo nahradit, aniž by to ovlivnilo ostatní části systému.

Kapitola 3

Navrh

3.1 Use Case Model

V této sekci prezentuji Use Case Model na obrázku 3.1, který slouží jako vizuální přehled interakcí uživatelů s navrhovaným Dashboardem IoT. Tento model mapuje různé funkce systému a poskytuje představu o tom, jak mohou uživatelé využívat dashboard pro nastavení upozornění, správu zařízení, konfiguraci datových zdrojů a další klíčové operace.



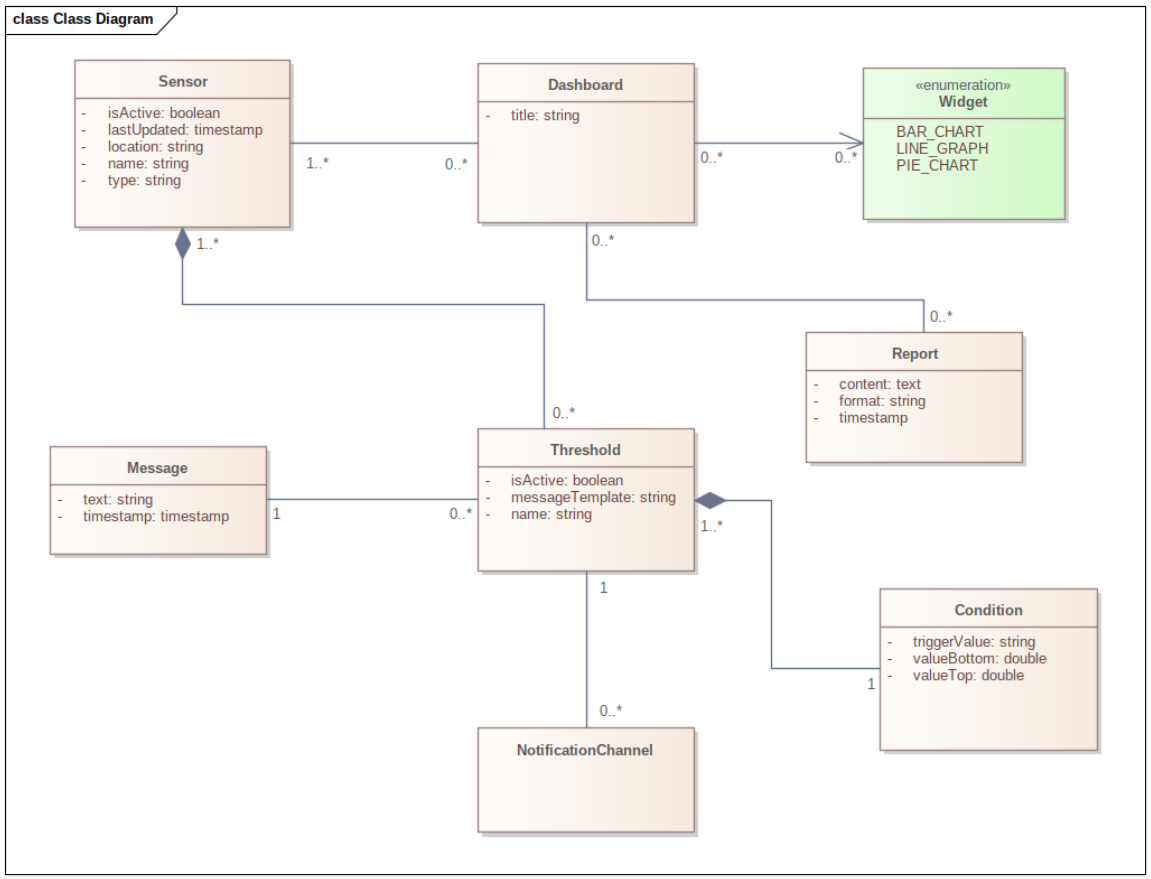
Obrázek 3.1: Use Case Model

3.2 Datový model

V návrhové fázi mého projektu Dashboardu IoT je klíčové podrobně specifikovat a organizovat různé komponenty, které jsou součástí systému. Diagram tříd na obrázku 3.2 poskytne vizuální reprezentaci těchto komponent ve formě tříd a bude ilustrovat jejich vzájemné vztahy a interakce.

V centru diagramu tříd je třída Dashboard, která představuje uživatelské rozhraní, kde lze agregovat a zobrazit informace z různých Widgetů. Každý Dashboard může být konfigurován s různými widgety, které umožňují vizualizaci dat v reálném čase, jako jsou grafy, seznamy a další interaktivní prvky. Tyto widgety budou odrážet aktuální stav připojených IoT zařízení a senzorů a poskytnou přehledné a přizpůsobitelné zobrazení klíčových metrik.

Třída Sensor je zásadní, protože obsahuje detailní informace o každém senzoru v síti, včetně typu senzoru, jeho umístění a shromážděných dat. Tato třída bude úzce spolupracovat s třídou Threshold, která bude zodpovědná za generování upozornění, když hodnoty senzoru překročí definované Condition.



Obrázek 3.2: Diagram tříd

Třída Threshold bude spojena s třídami Message a Condition, které umožní definování obsahu upozornění a podmínek, za kterých mají být upozornění aktivována. Uživatelé budou mít možnost konfigurovat preferované komunikační kanály, jako jsou Telegram, WhatsApp, Discord atd., pro příjem těchto upozornění.

Kromě toho umožní třída Report exportování dat z Dashboardu pro účely archivace nebo sdílení. Uživatelé budou moci generovat reporty, které obsahují přesné a relevantní informace vybrané z dostupných dat. Diagram tříd je navržen s ohledem na škálovatelnost a modulárnost, což zajistí snadné přidávání nových funkcí a integraci s různými systémy v budoucnu.

3.3 Návrh uživatelského rozhraní

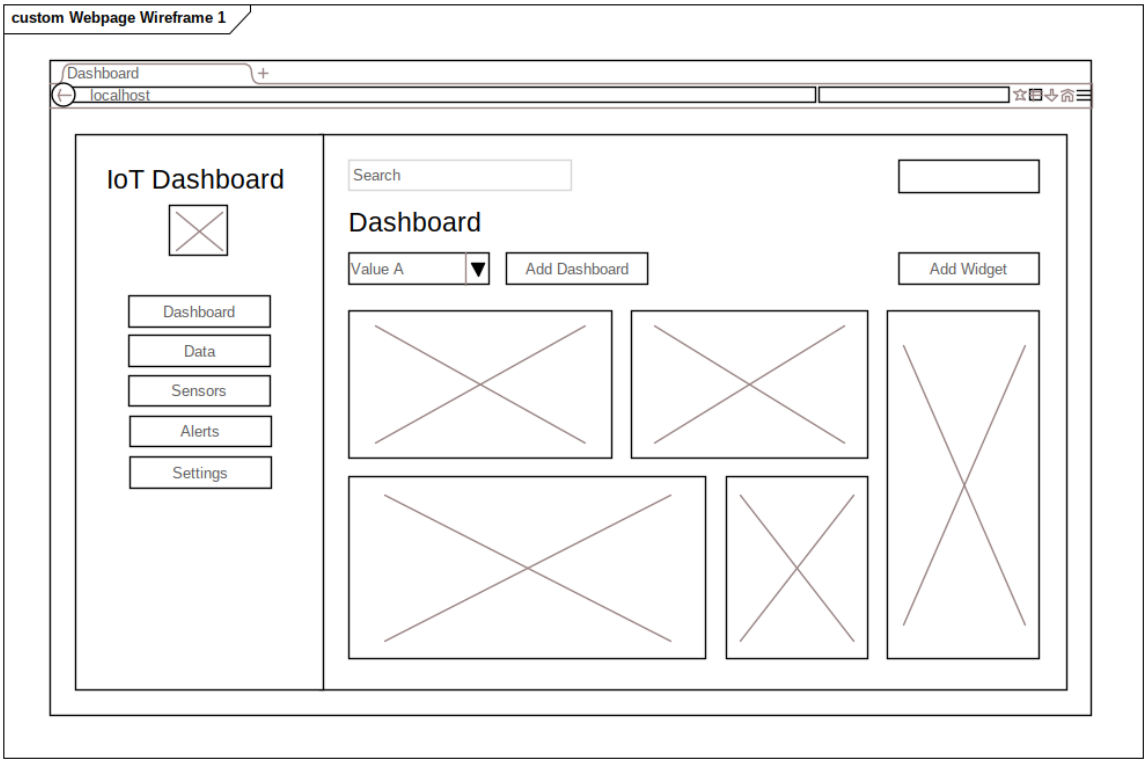
V procesu návrhu uživatelského rozhraní pro můj Dashboard IoT klade důraz na vytvoření intuitivního, efektivního a vizuálně přitažlivého rozhraní, které usnadní uživatelům interakci s komplexními daty a funkcemi systému. Cílem je poskytnout uživatelům jasný a logický tok informací, který jim umožní rychle a snadno navigovat, konfigurovat a analyzovat

data z jejich IoT zařízení.

Uživatelské rozhraní bude zahrnovat prvky, jako jsou přizpůsobitelné dashboardy, interaktivní widgety a jednoduché, ale mocné nástroje pro správu upozornění a reportů. Výsledné rozhraní bude sloužit jako most mezi složitými daty a koncovým uživatelem, přičemž klíčovým aspektem designu bude uživatelský komfort a celková uživatelská zkušenost.

3.3.1 Dasboard

První obrazovka 3.3, kterou uživatelé uvidí po otevření mého Dashboardu, je pečlivě navržena tak, aby poskytovala rychlý přehled a intuitivní přístup k nejdůležitějším funkcím. Na levém okraji obrazovky se nachází boční navigační menu, které je možné podle potřeby uživatele skrýt, aby se získal větší prostor pro práci. Toto menu zahrnuje klíčové sekce jako Zařízení, Historie dat a Nastavení, každá s vlastní ikonou a popiskem pro snadnou identifikaci.



Obrázek 3.3: Dashboard

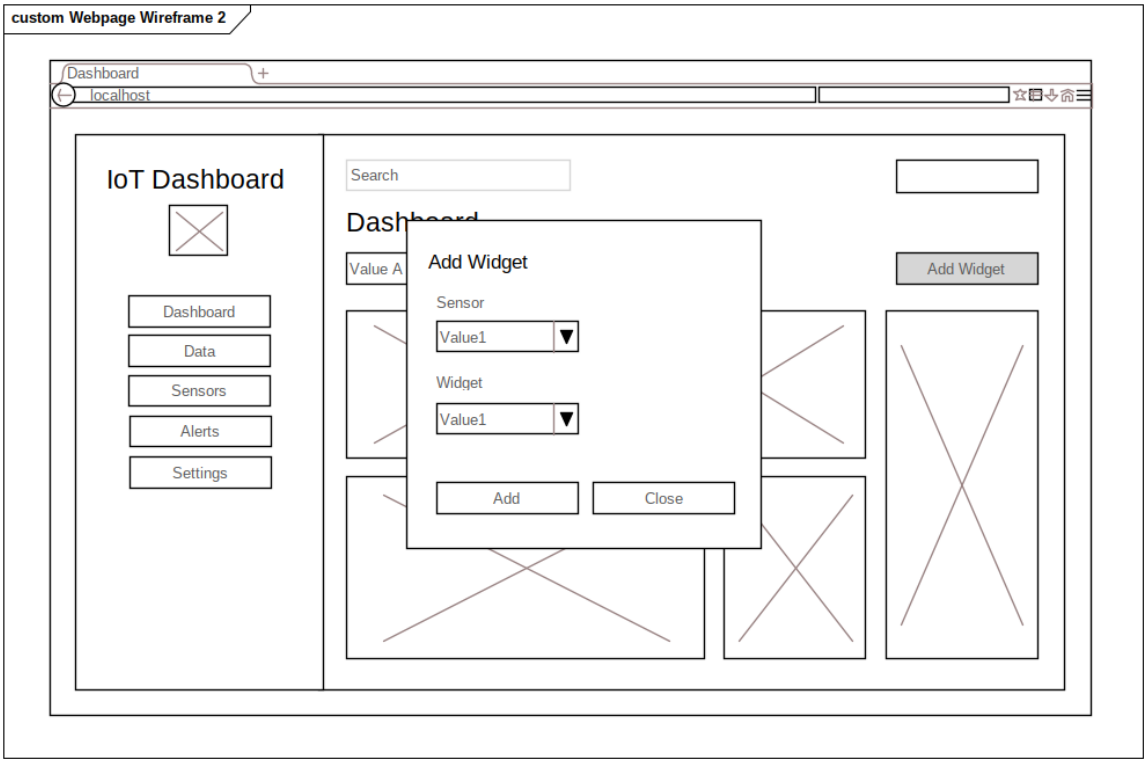
Horní část obrazovky je vyhrazena pro navigační lištu, která obsahuje vyhledávací pole umožňující rychlé vyhledávání a ikony pro přístup k nastavení a upozorněním. Tyto ikony jsou záměrně umístěny tak, aby byly hlavní funkce dostupné bez ohledu na to, zda je boční menu zobrazené.

Střed obrazovky tvoří hlavní obsahová sekce Dashboardu, kde jsou umístěny widgety. Tyto widgety mohou být různého typu – od grafů po seznamy – a každý z nich poskytuje uživatelům specifické informace a analýzy. Uživatelé mohou snadno přidávat nové widgety pomocí tlačítka Přidat widget a vytvářet nové Dashboardy pomocí tlačítka Přidat Dashboard. K dispozici je také dropdown menu, které umožňuje rychlý výběr mezi různými Dashboardy, které uživatel vytvořil.

3.3.2 Přidat widget

Po kliknutí na tlačítko Přidat widget se otevře modální okno 3.4, které uživatelům umožňuje konfigurovat nový widget pro jejich Dashboard. V tomto okně mohou uživatelé vybrat ze seznamu dostupných senzorů, které chtějí monitorovat. Následně si zvolí typ widgetu, který nejlépe odpovídá způsobu, jakým chtějí data zobrazit – ať už jde o grafy, seznamy nebo jiné vizualizační prvky. Po dokončení výběru mohou uživatelé potvrdit své nastavení a přidat widget na Dashboard pomocí tlačítka Přidat, nebo mohou celý proces zrušit a vrátit se zpět k Dashboardu pomocí tlačítka Zavřít. Celý proces je navržen tak, aby byl co nejintuitivnější

a nejpohodlnější, s cílem zjednodušit přidávání a konfiguraci widgetů a zároveň poskytnout uživatelům flexibilitu ve způsobu prezentace jejich dat.



Obrázek 3.4: Přidat widget

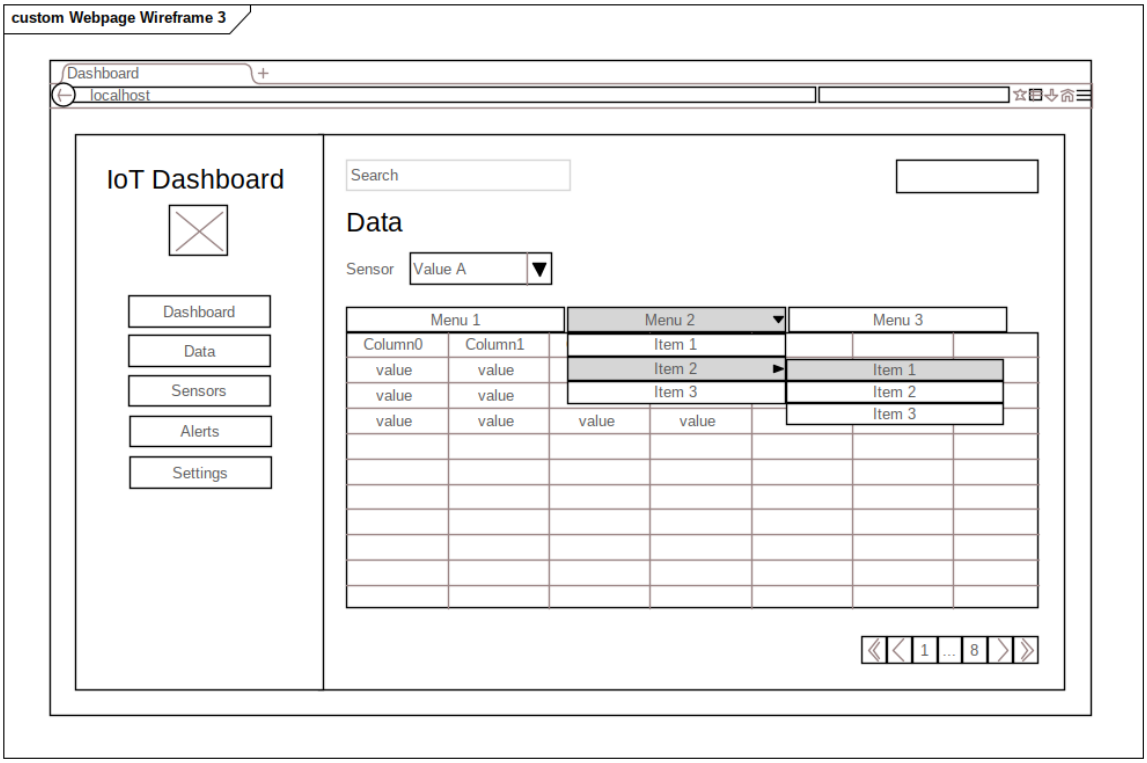
3.3.3 Data

Obrazovka Data 3.5 je navržena jako centrální místo pro prohlížení a manipulaci s daty shromážděnými z připojených senzorů. Tato obrazovka poskytuje uživatelům tabulkovou reprezentaci dat, která umožňuje hlubší analýzu a porozumění shromážděným informacím.

Na začátku obrazovky se nachází rozbalovací menu, kde uživatelé mohou vybrat specifický senzor, jehož data chtějí zobrazit. Po výběru se v hlavní části obrazovky zobrazí tabulka s daty odpovídajícími vybranému senzoru. Tabulka byla navržena s ohledem na uživatelskou přívětivost a efektivitu práce, nabízí proto funkce, jako je sortování dat podle libovolného sloupce, což uživatelům umožňuje rychle organizovat data podle potřeby.

Uživatelé mají také možnost skrýt nebo zobrazit určité sloupce v tabulce, což jim umožňuje přizpůsobit si zobrazení podle svých preferencí a soustředit se pouze na data, která jsou pro ně v daném okamžiku nejdůležitější.

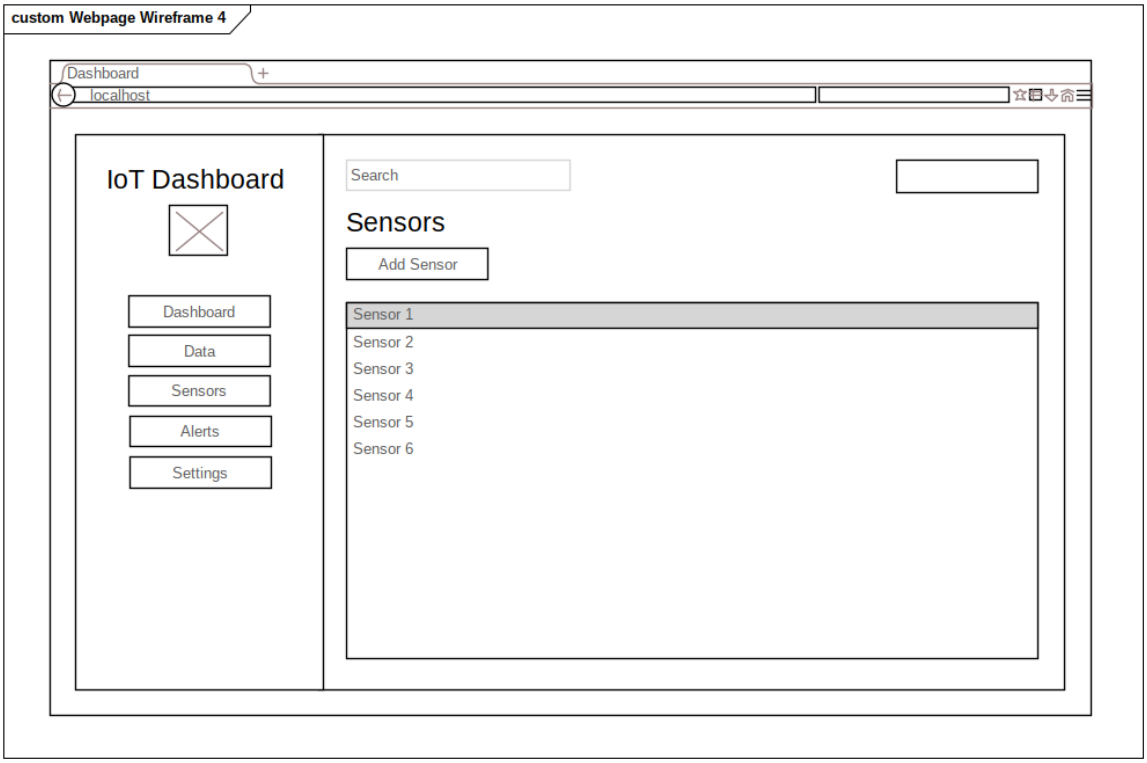
Další důležitou funkcí je možnost exportu dat do souboru, což uživatelům umožňuje snadno stáhnout a uložit data pro další zpracování nebo sdílení mimo Dashboard. Data mohou být exportována do různých formátů, jako jsou CSV nebo Excel, podle toho, co je pro uživatele nejpohodlnější.



Obrázek 3.5: Data

3.3.4 Sensors

Stránka Sensors 3.6 zobrazuje seznam dostupných senzorů v systému. Každý senzor je reprezentován svým názvem nebo označením (např. Senzor 1, Senzor 2 atd.). Tento seznam poskytuje uživatelům rychlý přehled o všech senzorech, které jsou součástí jejich IoT řešení. Pomocí vyhledávacího pole v horní části seznamu mohou uživatelé snadno najít konkrétní senzor, který je zajímá. Tlačítko Add Sensor v dolní části seznamu umožňuje uživatelům přidávat do systému nové senzory.

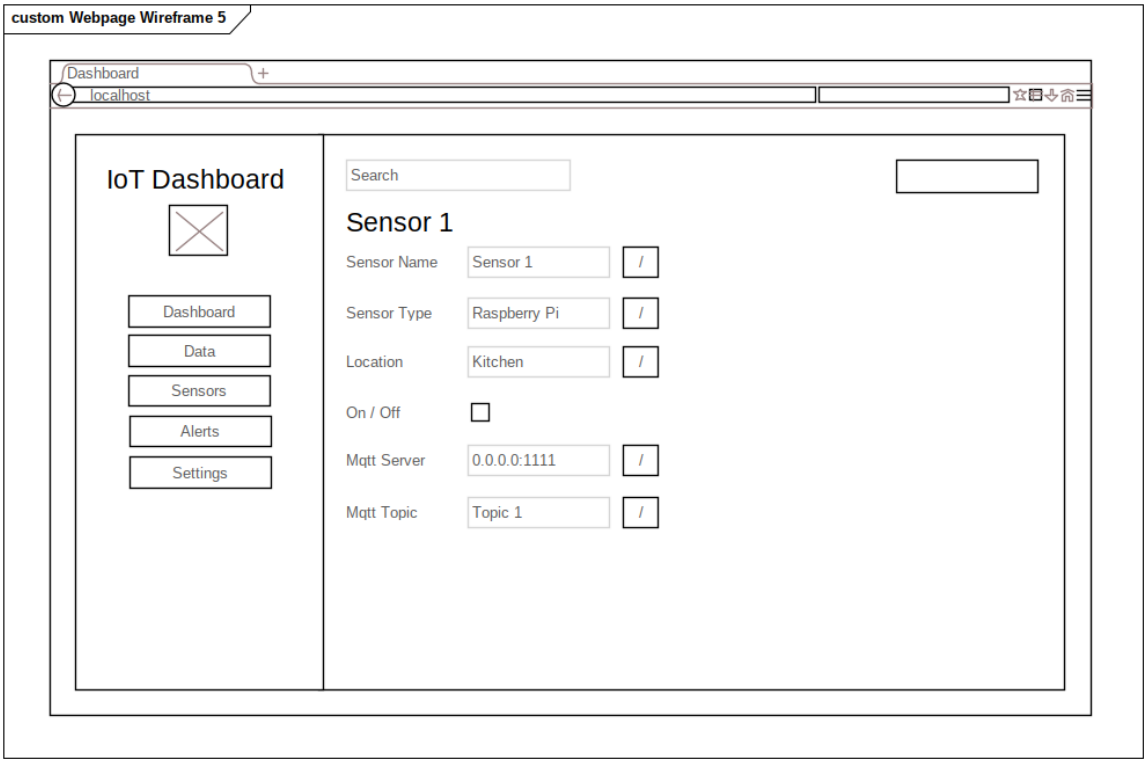


Obrázek 3.6: Sensors

3.3.5 Sensor

Stránka Sensor 3.7 poskytuje detailní informace o konkrétním senzoru a umožňuje uživatelům upravovat jeho nastavení. Po výběru senzoru ze seznamu na předchozí stránce se uživatel dostane na tuto stránku, kde může zobrazit a upravit různé atributy senzoru. Stránka obsahuje několik editovatelných polí, která umožňují uživatelům přizpůsobit senzor podle jejich potřeb. Uživatelé mohou zadat nebo upravit název senzoru, který jej jasně identifikuje v rámci jejich IoT řešení. Dále mohou specifikovat typ senzoru, například teplotní senzor, senzor vlhkosti nebo senzor pohybu. Pole pro umístění umožňuje uživatelům zadat fyzické umístění senzoru, což usnadňuje jeho identifikaci a správu v rámci větších IoT systémů.

Po úpravě těchto polí mohou uživatelé kliknout na tlačítko Save pro uložení provedených změn a aktualizaci informací o senzoru v systému, nebo mohou použít tlačítko "Cancel"pro zrušení změn a návrat na předchozí stránku beze změn. Stránka Sensor poskytuje uživatelům flexibilitu při konfiguraci a správě jednotlivých senzorů, což jim umožňuje přizpůsobit IoT řešení jejich specifickým požadavkům a zajistit efektivní monitorování a sběr dat.



Obrázek 3.7: Sensor

Wireframy pro stránky Alerts a Settings nejsou v tomto návrhu zahrnuty, protože se předpokládá, že budou vypadat podobně jako stránka Sensors. Není tedy nutné vytvářet samostatné wireframy pro každou z těchto stránek, jelikož budou pravděpodobně sdílet podobné rozvržení a funkce.

Stránka Alerts by mohla obsahovat seznam upozornění a možnosti jejich konfigurace, podobně jako stránka Sensors obsahuje seznam senzorů a možnosti jejich správy. Uživatelé by mohli procházet existující upozornění, upravovat jejich nastavení nebo vytvářet nová upozornění podle svých potřeb.

Podobně by stránka Settings mohla poskytovat uživatelům možnost přizpůsobit obecná nastavení IoT řešení, jako jsou předvolby jednotek, frekvence aktualizací dat nebo integrace s externími systémy. Rozvržení a interakce by mohly být podobné jako na stránce Sensors, kde uživatelé mohou upravovat atributy a nastavení jednotlivých senzorů.

Kapitola 4

Implementace

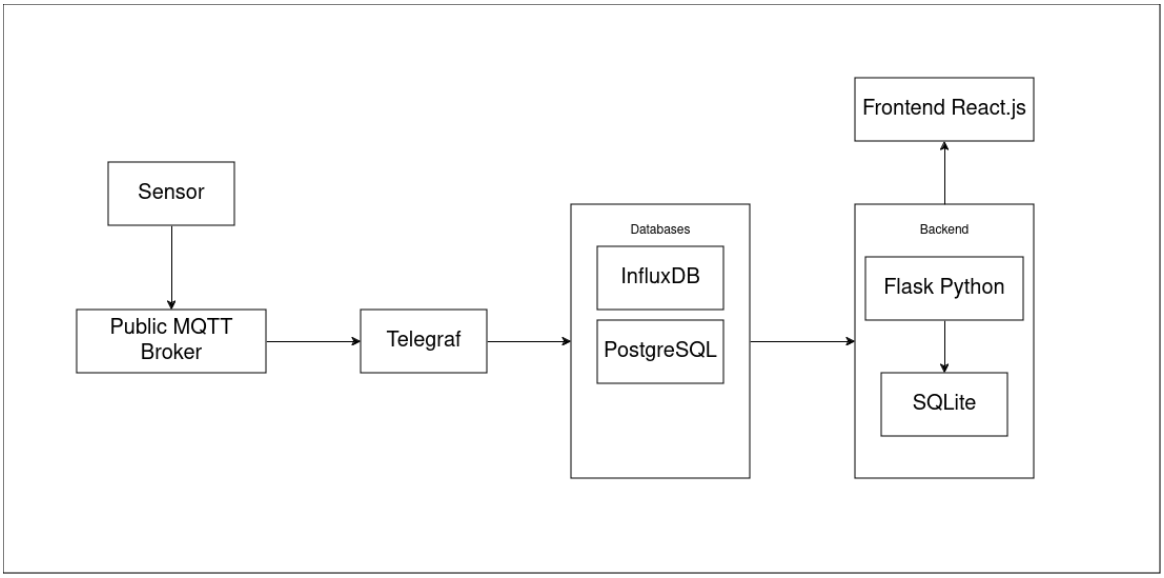
4.1 Úvod

Tato kapitola se zabývá implementací aplikace Dashboard, která kombinuje moderní webové technologie a techniky sběru dat s cílem vytvořit efektivní a škálovatelné řešení.

V následujících sekcích budou podrobně popsány jednotlivé aspekty implementace, včetně architektury, frontendu, backendu, sběru dat, integrace databáze a nasazení. Budou poskytnuty detailní vysvětlení a ukázky kódu, aby byl důkladně popsán proces implementace a zvolená řešení při vývoji této aplikace. Cílem je poskytnout ucelený pohled na implementaci a prezentovat použité technologie a postupy.

4.2 Přehled architektury

Na základě provedené analýzy a požadavků byla navržena architektura aplikace Dashboard, jejíž schéma je prezentováno na obrázku 4.1.



Obrázek 4.1: Architektura aplikace

Diagram znázorňuje tok dat v aplikaci, kde data jsou sbírána z MQTT senzorů a odesílána na veřejný MQTT server. Odtud jsou data přijímána agentem Telegraf, který je shromažďuje a předává dál. Data jsou následně uložena do databáze InfluxDB nebo alternativně do PostgreSQL. Frontend aplikace komunikuje s backendem prostřednictvím API volání pro získání a zobrazení dat uživatelům. Všechny komponenty aplikace jsou provozovány v Docker kontejnerech, což zajišťuje snadnou přenositelnost a škálovatelnost celého řešení.

4.3 Implementace frontendu

4.3.1 Úvod

Frontend aplikace Dashboard je implementována pomocí populární JavaScriptové knihovny React. React je výkonná a flexibilní knihovna pro tvorbu uživatelských rozhraní, která umožňuje efektivní vývoj interaktivních a škálovatelných webových aplikací. Díky své komponentově orientované architektuře a deklarativnímu přístupu k renderování UI je React ideální volbou pro vývoj moderních a responzivních aplikací. Při implementaci frontendu aplikace byly využity osvědčené postupy a doporučení z oficiální dokumentace Reactu [14].

Pro zajištění navigace mezi jednotlivými stránkami aplikace Dashboard byl použit balíček react-router-dom. React Router je de facto standardní řešení pro routování v React aplikacích, které umožňuje definovat a spravovat cesty (routes) a jejich mapování na příslušné komponenty.

V aplikaci Dashboard jsou routy definovány v souboru `App.js`, který představuje hlavní komponentu aplikace. Ukázka kódu s implementací routování:

```
1 function App() {
2   const [theme, colorMode] = useMode();
3
```

```
  return (
    <ColorModeContext.Provider value={colorMode}>
      <ThemeProvider theme={theme}>
        <CssBaseline />
        <div className="app">
          <Sidebar />
          <main className="content">
            <Topbar />
            <Routes>
              <Route path="/" element={<Dashboard />} />
              <Route path="/data" element={<Data />} />
              <Route path="/sensors" element={<Sensors />} />
              <Route path="/sensors/:sensorId" element={<Sensor />} />
              <Route path="/thresholds" element={<Thresholds />} />
              <Route path="/settings" element={<Settings />} />
              <Route path="/about" element={<About />} />
            </Routes>
          </main>
        </div>
      </ThemeProvider>
    </ColorModeContext.Provider>
  );
}
```

Pomocí komponenty BrowserRouter z balíčku react-router-dom je zajištěno, že aplikace používá HTML5 History API [8] pro navigaci. Pro navigaci mezi stránkami jsou použity komponenty `Link` a `NavLink`, které umožňují vytvářet odkazy na definované routy. Tyto komponenty zajišťují, že při kliknutí na odkaz nedojde k obnovení stránky, ale pouze k aktualizaci URL a vykreslení příslušné komponenty.

React Router také podporuje dynamické routy s parametry, které umožňují předávat data mezi komponentami. Například, při kliknutí na konkrétní položku v seznamu může být ID položky předáno jako parametr do URL a následně použito pro načtení detailů položky.

4.3.2 Material-UI a témování aplikace

Pro vytvoření moderního a konzistentního uživatelského rozhraní byla v aplikaci Dashboard použita knihovna Material-UI (MUI) ve verzi 5. MUI je populární sada React komponent, která implementuje Material Design specifikaci od Google [11]. Nabízí širokou škálu předpřipravených komponent, jako jsou tlačítka, formulářové prvky, navigační panely a mnoho dalších, které usnadňují vývoj responzivního a přívětivého uživatelského rozhraní.

V aplikaci Dashboard je implementována podpora pro tmavý a světlý režim, což umožňuje uživatelům přizpůsobit vzhled aplikace podle svých preferencí. Pro správu témování je využita knihovna MUI a vlastní soubor `theme.js`, který definuje barevné palety pro oba režimy [10]. Příklad použití:

```
  const theme = useTheme();
  const colors = tokens(theme.palette.mode);

    <Box
      sx={{
        "& .pro-sidebar-inner": {
          background: `${colors.primary[400]} !important`,
```

```
8           },
9         }}
10      >
```

V souboru `theme.js` je vytvořena funkce `tokens`, která na základě zvoleného režimu (mode) vrací odpovídající barevnou paletu. Paleta obsahuje různé odstíny barev, například `grey` a `primary`, které jsou definovány pomocí hexadecimálních kódů. Tato funkce umožňuje snadno přistupovat k barvám v závislosti na aktuálním režimu aplikace.

Pro přepínání mezi tmavým a světlým režimem je použit kontext `ColorModeContext`, který uchovává aktuální režim a poskytuje funkci pro jeho změnu. Tento kontext je dostupný v celé aplikaci díky komponentě `ColorModeProvider`, která obaluje ostatní komponenty. V jednotlivých komponentách lze snadno přistupovat k aktuálnímu tématu pomocí hooku `useTheme` z MUI a k odpovídajícím barvám pomocí funkce `tokens`.

Pro vytvoření postranní navigace (sidebar) byla použita knihovna react-pro-sidebar, která poskytuje sadu komponent pro vytváření vysoce úrovňových a přizpůsobitelných bočních navigací. Tato knihovna nabízí flexibilní a snadno použitelné řešení pro implementaci responzivní a interaktivní navigace.

4.3.3 Vizualizace dat

Aplikace Dashboard poskytuje uživatelům přehledné a interaktivní vizualizace dat, které usnadňují analýzu a pochopení informací. Pro zobrazení dat jsou použity dva hlavní přístupy: tabulky a grafy.

Tabulky jsou implementovány pomocí komponenty `Table` z knihovny MUI. Tato komponenta nabízí výkonné a flexibilní řešení pro zobrazení tabulkových dat s možností stránkování, řazení, filtrování a přizpůsobení vzhledu.

Pro vizualizaci dat pomocí grafů je v aplikaci Dashboard využita knihovna Nivo. Nivo [12] je výkonná knihovna pro vytváření datových vizualizací v Reactu, která nabízí širokou škálu typů grafů a bohaté možnosti přizpůsobení. Grafy vytvořené pomocí Nivo jsou interaktivní, responzivní a snadno integrovatelné do React aplikací.

V aplikaci Dashboard jsou použity různé typy grafů z knihovny Nivo:

- Line - Spojnicový graf pro zobrazení trendů a vývoje hodnot v čase.
- Bar - Sloupcový graf pro porovnání kategorií a jejich hodnot.
- Pie - Koláčový graf pro zobrazení podílů a rozdělení celku na části.

4.3.4 Komunikace s backendem

V aplikaci Dashboard není použita žádná externí knihovna pro komunikaci s backendem. Místo toho je využito nativní rozhraní prohlížeče Fetch API [7], které umožňuje odesílat HTTP požadavky a zpracovávat odpovědi.

Následující ukázka kódu demonstruje použití Fetch API pro získání dat ze serveru:

```
const fetchDashboards = async () => {
  try {
    const response = await fetch(baseUrl + '/dashboards');
    const data = await response.json();
    setDashboards(data);
  } catch (error) {
    console.error('Error fetching dashboards:', error);
  }
};
```

4.4 Implementace backendu

4.4.1 Úvod

Pro implementaci backendu aplikace Dashboard byl zvolen framework Flask, který je napsán v jazyce Python. Flask je oblíbený a široce používaný webový framework, který poskytuje jednoduchou a intuitivní strukturu pro vývoj webových aplikací a API.

Hlavním důvodem pro výběr Flasku a Pythonu je schopnost efektivně pracovat s velkými objemy dat. Aplikace Dashboard vyžaduje zpracování a analýzu rozsáhlých datových sad, a Python je známý svou silnou podporou pro manipulaci s daty a vědecké výpočty. Flask také nabízí snadnou integraci s různými databázovými systémy, včetně InfluxDB, která je použita v aplikaci Dashboard pro ukládání a dotazování časových řad. S Flaskem lze snadno implementovat endpointy API pro interakci s databází a poskytovat data frontendové části aplikace.

4.4.2 Struktura projektu

Struktura backendového projektu vychází z doporučení uvedených v oficiální dokumentaci Flasku [9]. Cílem je vytvořit organizovanou a udržovatelnou strukturu, která usnadňuje vývoj, testování a nasazení aplikace.

Hlavní komponenty struktury projektu jsou následující:

- **Controller**: Adresář `controller` obsahuje jednu třídu kontroleru, která definuje všechny endpointy API aplikace. Tato třída obsahuje metody odpovídající jednotlivým endpointům. Každá metoda zpracovává příchozí požadavky, volá příslušné služby a vrací odpovědi klientovi.
- **Services**: Adresář `services` obsahuje moduly, které zapouzdřují logiku aplikace a komunikaci s databází. Každý modul služby se zaměřuje na konkrétní doménu, například `sensorservice.py` pro operace související se senzory, `dashboardservice.py` pro operace související s dashboardy atd. Služby obsahují funkce pro získávání, ukládání a manipulaci s daty a jsou volány z metod kontroleru.
- **Config**: Konfigurační soubory aplikace jsou umístěny v adresáři `config`. Tento adresář obsahuje soubory jako `config.py`, který definuje různé konfigurační proměnné a nastavení pro různá prostředí (např. vývoj, testování, produkce). Konfigurační soubory také načítají citlivé údaje, jako jsou přístupové údaje k databázi nebo klíče API, z externího souboru `.env`.

- **.env**: Soubor `.env` obsahuje citlivé konfigurační proměnné a přístupové údaje, které by neměly být uloženy v systému správy verzí. Tento soubor je načítán během inicializace aplikace a jeho hodnoty jsou zpřístupněny prostřednictvím konfiguračních souborů.
- **Requirements**: Soubor `requirements.txt` obsahuje seznam všech externích závislostí a knihoven požadovaných backendovou aplikací. Tento soubor se používá při nasazování aplikace k instalaci všech potřebných balíčků.
- **Tests**: Adresář `tests` obsahuje testovací skripty a testovací data pro aplikaci. Testy jsou organizovány podle funkcionalit a lze je spustit pomocí testovacích frameworků jako pytest.

Tato struktura projektu poskytuje jasné oddělení zodpovědností a usnadňuje údržbu a rozšiřování backendu.

4.4.3 Endpointy

Backend aplikace Dashboard poskytuje sadu endpointů API, které umožňují interakci s různými zdroji dat a funkcionalitami. Tyto endpointy jsou rozděleny do několika kategorií podle jejich účelu.

Následující tabulka poskytuje přehled dostupných endpointů:

Endpoint	**HTTP metody**	**Popis**
/sensors	POST, GET	Vytvoření a získání senzorů
/sensors/<id>	GET, PUT, DELETE	Operace s konkrétním senzorem
/data/<id>	GET	Získání dat pro konkrétní senzor
/dashboards	POST, GET	Vytvoření a získání dashboardů
/dashboards/<id>	GET, PUT, DELETE	Operace s konkrétním dashboardem
/widgets	POST, GET	Přidání a získání widgetů na dashboardu
/widgets/<id>	DELETE	Odebrání widgetu z dashboardu
/alerts	POST, GET	Vytvoření a získání prahových hodnot
/alerts/<id>	DELETE	Smazání prahové hodnoty
/channels	POST, GET	Aktualizace a získání komunikačních kanálů

Tabulka 4.1: Přehled endpointů

Všechny endpointy vrací data ve formátu JSON a používají příslušné HTTP stavové kódy pro indikaci úspěchu nebo chyby. Chybové zprávy jsou také vraceny ve formátu JSON s podrobnými informacemi o chybě.

4.4.4 Komunikace s databázemi

Backend aplikace Dashboard využívá dvě databáze pro různé účely: SQLite a InfluxDB.

SQLite je relační databáze používaná pro ukládání interních dat aplikace, jako jsou informace o senzorech, dashboardech a dalších entitách. SQLite je jednoduchá, lehká a snadno integrovatelná databáze, která nevyžaduje samostatný databázový server.

InfluxDB je časová databáze optimalizovaná pro rychlé a efektivní ukládání a dotazování časových řad. InfluxDB se používá pro ukládání a získávání dat ze senzorů, která mají časovou povahu. Každý datový bod v InfluxDB obsahuje časové razítko a odpovídající hodnotu naměřenou senzorem. InfluxDB umožňuje efektivní dotazování a agregaci dat na základě časových rozsahů a dalších kritérií.

Komunikace s oběma databázemi je implementována pomocí příslušných knihoven a modulů v Pythonu. Pro SQLite se používá standardní knihovna `sqlite3`, která poskytuje rozhraní pro práci s SQLite databázemi. Pro InfluxDB se používá oficiální knihovna `InfluxDB-Python`, která poskytuje jednoduché a intuitivní API pro interakci s databází InfluxDB.

V rámci backendové aplikace jsou definovány funkce pro zápis a čtení dat z obou databází. Tyto funkce abstrahují detaily komunikace s databázemi a poskytují jednotné rozhraní pro práci s daty v aplikaci. Příklad dotazu na data z InfluxDB pomocí knihovny InfluxDB-Python:

```
    client = InfluxDBClient(url=url, token=token, org=org)
    query = f'''
    from(bucket: "{bucket}")
      |> range(start: -1h)
      |> filter(fn: (r) => r._measurement == "{sensor_id}")
      |> limit(n: 100)
    '''
    query_api = client.query_api()
    result = query_api.query(org=org, query=query)
```

Další příklady kódu demonstrující komunikaci s databázemi:

```
def get_db():
    if "db" not in g:
        g.db = sqlite3.connect(
            current_app.config["DATABASE"],
            detect_types=sqlite3.PARSE_DECLTYPES
        )
        g.db.row_factory = sqlite3.Row
    return g.db

def get_influxdb_client():
    url = current_app.config["INFLUXDB_URL"]
    token = current_app.config["INFLUXDB_TOKEN"]
    org = current_app.config["INFLUXDB_ORG"]
    g.influx = InfluxDBClient(url=url, token=token, org=org)
    return g.influx
```

Listing 4.1: Inicializace SQLite

4.4.5 Notifikace a prahové hodnoty

Aplikace Dashboard poskytuje možnost vytvářet prahové hodnoty (thresholds) pro sledování a upozorňování na specifické hodnoty senzorů. Uživatel může definovat název hodnoty, horní a dolní mez pro danou hodnotu a také šablonu zprávy, která se odešle při překročení prahové hodnoty. Šablona zprávy může obsahovat tyto tři hodnoty, které jsou zpracovány jednoduchým parserem:

```
def parse_message(string, value_bottom=None, value=None, value_top=None):
    if value_bottom is not None:
        string = string.replace("value_bottom", str(value_bottom))
    if value is not None:
        string = string.replace("value", str(value))
    if value_top is not None:
        string = string.replace("value_top", str(value_top))
```

Pro pravidelnou kontrolu prahových hodnot se používá knihovna APScheduler. Každých 15 sekund se spustí úloha, která kontroluje databázi a porovnává hodnoty senzorů s definovanými prahovými hodnotami. Pokud se hodnota nachází mimo stanovený rozsah, vytvoří se notifikace a odešle se na povolené komunikační kanály, pokud jsou nastaveny v konfiguraci aplikace.

Jedním z požadavků na aplikaci byla schopnost odesílat notifikace na různé messengery, chaty a platformy. Pro implementaci byly zvoleny Telegram a Discord, protože nabízejí možnost integrace s Pythonem, jsou populární a uživatelsky přívětivé. Aplikace je navržena tak, aby bylo možné snadno přidat podporu pro další komunikační kanály. Ukázky kódu pro implementaci notifikací a komunikace s Telegramem a Discordem:

```
import asyncio
from telegram import Bot

def send_telegram_message(message, bot_token, channel_id):
    asyncio.run(send_message(message, bot_token, channel_id))

async def send_message(message, bot_token, channel_id):
    bot = Bot(token=bot_token)
    await bot.send_message(chat_id=channel_id, text=message)
    print("Message to the Telegram sent successfully!")

import discord

intents = discord.Intents.default()
intents.messages = True
intents.guilds = True

def send_discord_message(message, bot_token, channel_id):
    client = discord.Client(intents=intents)
    @client.event
    async def on_ready():
        print(f'Logged in as {client.user}')
        channel = client.get_channel(channel_id)
        if channel:
            await channel.send(message)
        else:
            print(f'Channel with ID {channel_id} not found.')
```

```
28          await client.close()
29      client.run(bot_token)
```

Pro komunikaci s Telegramem se používá knihovna `python-telegram-bot`, která poskytuje jednoduché rozhraní pro interakci s Telegram API. Proces vytvoření Telegram bota zahrnuje následující kroky:

1. Registrace nového bota pomocí BotFather na Telegramu.
2. Získání přístupového tokenu pro bota.
3. Inicializace bota v kódu aplikace pomocí přístupového tokenu.
4. Definice obslužných funkcí pro příjem a zpracování příchozích zpráv.
5. Spuštění bota a naslouchání příchozím zprávám.

Pro komunikaci s Discordem se používá knihovna `discord.py`. Proces vytvoření Discord bota je podobný jako u Telegramu:

1. Vytvoření nové aplikace a bota na Discord Developer Portalu.
2. Získání přístupového tokenu pro bota.
3. Inicializace bota v kódu aplikace pomocí přístupového tokenu.
4. Definice obslužných funkcí pro příjem a zpracování příchozích zpráv a událostí.
5. Spuštění bota a připojení k Discord serveru.

Jakmile jsou boti inicializováni a spuštěni, aplikace Dashboard může odesílat notifikace o překročení prahových hodnot na příslušné komunikační kanály. Uživatelé tak mohou být informováni o důležitých událostech a anomáliích v reálném čase.

4.5 Sběr dat pomocí Telegrafu

4.5.1 Obecný přehled Telegrafu

Telegraf je vysoce výkonný a flexibilní nástroj pro sběr dat. Podporuje širokou škálu vstupních pluginů, které umožňují shromažďovat metriky z různých systémů, služeb a protokolů. Mezi podporované vstupy patří například CPU, paměť, disky, databáze, API a mnoho dalších.

Nasbíraná data mohou být transformována, filtrována a obohacena pomocí procesorů a agregátorů. Telegraf také nabízí řadu výstupních pluginů pro odesílání dat do různých cílových úložišť, jako jsou databáze časových řad (např. InfluxDB), monitorovací systémy, fronty zpráv a další.

Konfigurace Telegrafu se provádí pomocí konfiguračního souboru ve formátu TOML. V tomto souboru se definují vstupní a výstupní pluginy, jejich nastavení a další parametry agenta.

4.5.2 Použití Telegrafu v aplikaci Dashboard

V aplikaci Dashboard je Telegraf použit jako agent pro sběr dat ze senzorů a jejich odesílání do různých databází.Konfigurace Telegrafu je uložena v souboru `telegraf.conf`:

```
[agent]
  interval = "10s"
  round_interval = true
  metric_batch_size = 1000
  metric_buffer_limit = 10000
  collection_jitter = "0s"
  flush_interval = "10s"
  flush_jitter = "0s"
  precision = ""
  debug = false
  quiet = false
  logfile = ""
  hostname = ""
  omit_hostname = false

[[outputs.influxdb_v2]]
  urls = ["http://influxdb:8086"]
  token = "${DOCKER_INFLUXDB_INIT_ADMIN_TOKEN}"
  organization = "${DOCKER_INFLUXDB_INIT_ORG}"
  bucket = "${DOCKER_INFLUXDB_INIT_BUCKET}"
  insecure_skip_verify = true

[[outputs.postgresql]]
  connection = "postgres://${POSTGRES_USER}:${POSTGRES_PASSWORD}@postgres
      :5432/${POSTGRES_DB}?sslmode=disable"

[[inputs.mqtt_consumer]]
  servers = ["tcp://broker.emqx.io:1883"]
  topics = [
    "electricity/sensor"
  ]
  data_format = "influx"
```

V sekci `agent` jsou definovány obecné parametry agenta, jako je interval sběru dat, velikost dávky metrik, limit bufferu a další.

Sekce `outputs.influxdbv2` definuje výstup do databáze InfluxDB. Jsou zde specifikovány URL databáze, přístupový token, organizace a bucket, do kterého se data ukládají. InfluxDB slouží jako hlavní úložiště časových řad pro aplikaci Dashboard.

Sekce `outputs.postgresql` definuje výstup do databáze PostgreSQL. Je zde uvedena connection string pro připojení k databázi, včetně přihlašovacích údajů. Ukládání dat do PostgreSQL demonstruje schopnost Telegrafu odesílat data do různých typů databází podle požadavků aplikace.

V sekci `inputs.mqttconsumer` je nakonfigurován vstupní plugin pro příjem dat ze senzorů přes protokol MQTT. Jsou zde specifikovány adresy MQTT brokerů a témata, ze kterých se data přijímají. Formát dat je nastaven na `influx`, což znamená, že data jsou očekávána ve formátu měření InfluxDB.

Díky této konfiguraci Telegraf naslouchá na specifikovaných MQTT tématech, přijímá data ze senzorů a odesílá je do nakonfigurovaných databází.

4.6 Nasazení pomocí Docker Compose

Docker je populární platforma pro kontejnerizaci aplikací, která umožňuje snadné vytváření, nasazení a spouštění aplikací v izolovaných kontejnerech. Docker Compose je nástroj, který slouží k definici a spouštění vícekontejnerových aplikací pomocí jediného konfiguračního souboru. V této práci jsem využil Docker Compose pro nasazení aplikace, která se skládá z několika služeb. Konfigurační soubor docker-compose.yml definuje jednotlivé služby a jejich nastavení. Například služba `telegraf` může vypadat takto:

```
telegraf:
  container_name: telegraf
  image: telegraf:1.25-alpine
  depends_on:
    - influxdb
    - postgres
  volumes:
    - ./telegraf/telegraf.conf:/etc/telegraf/telegraf.conf:ro
  env_file:
    - .env
```

Tato ukázka demonstruje, jak lze v souboru docker-compose.yml nakonfigurovat službu, včetně nastavení závislostí, portů, svazků a proměnných prostředí. Použití Docker Compose značně zjednodušuje nasazení a správu aplikace. Všechny potřebné služby jsou definovány v jednom konfiguračním souboru a mohou být spuštěny jediným příkazem. Docker Compose také usnadňuje škálování a správu aplikace v různých prostředích.

Kapitola 5

Testování

5.1 Úvod

Testování je nedílnou součástí procesu vývoje softwaru a hraje klíčovou roli při zajišťování kvality, spolehlivosti a bezpečnosti aplikací. Bez důkladného testování mohou být aplikace náchylné k chybám, bezpečnostním problémům a nespolehlivému chování.

Tato kapitola se zabývá testováním aplikace ze dvou hlavních hledisek. Zaprvé se zaměřuje na integrační testy endpointů API. Integrační testy ověřují, zda jednotlivé komponenty aplikace správně spolupracují a zda API vrací očekávané výsledky. Tyto testy pomáhají odhalit chyby v komunikaci mezi komponentami a zajistit, že API funguje podle očekávání.

Druhá část kapitoly je věnována testování celé aplikace s využitím Raspberry Pi Pico. Raspberry Pi Pico je cenově dostupný a všestranný mikrokontrolér, který umožňuje simulovat reálné prostředí a otestovat, jak aplikace funguje v praxi. Pomocí Raspberry Pi Pico lze ověřit, zda aplikace správně zpracovává data a reaguje na různé podněty.

V následujících sekcích se kapitola podrobně věnuje každé z těchto oblastí testování. Popisuje použité nástroje, testovací scénáře, výsledky testů a jejich dopad na kvalitu a spolehlivost aplikace. Cílem je poskytnout ucelený přehled o testovacím procesu a demonstrovat, jak testování přispívá k vývoji robustní a spolehlivé aplikace.

5.2 Integrační testy endpointů

5.2.1 Integrační testy a použitý testovací framework

Integrační testy jsou důležitou součástí procesu vývoje softwaru, které se zaměřují na ověření správné spolupráce a integrace mezi jednotlivými komponentami nebo moduly aplikace. Na rozdíl od jednotkových testů, které testují izolované části kódu, integrační testy se zabývají interakcí mezi různými částmi systému a jejich schopností pracovat společně pro dosažení požadovaných výsledků. Hlavním cílem integračních testů je odhalit chyby a problémy, které mohou vzniknout při propojení a komunikaci mezi různými komponentami, jako jsou nekompatibilní rozhraní, chybná výměna dat nebo nesprávné zpracování požadavků.

V této práci jsou integrační testy implementovány pomocí testovacího frameworku unittest, který je součástí standardní knihovny Pythonu. Unittest poskytuje sadu nástrojů a

konvencí pro psaní a spouštění testů. Hlavní výhodou použití unittest je jeho jednoduchost a přímočarost. Testy jsou organizovány do tříd, které dědí z `unittest.TestCase`, a každý test je definován jako metoda uvnitř této třídy. Unittest také poskytuje řadu assertačních metod pro ověřování výsledků testů, jako například `assertEqual()`, `assertIsInstance()` a další.

Při testování aplikace je využívána speciální konfigurace a databáze, která je vytvořena před každým testem a po jeho dokončení smazána. Toto zajišťuje izolaci testů a zabraňuje vzájemnému ovlivňování mezi jednotlivými testy. V ukázce kódu lze vidět, jak je vytvořena dočasná databáze pomocí `tempfile.mkstemp()` a jak je aplikace inicializována s testovací konfigurací:

```
def setUp(self):
    self.db_fd, self.db_path = tempfile.mkstemp()
    self.app = create_app({
        'TESTING': True
    })
    self.app_context = self.app.app_context()
    self.app_context.push()
    with self.app.app_context():
        db.init_db()

def tearDown(self):
    db.close_db()
    self.app_context.pop()
    os.remove(self.app.config['DATABASE'])
```

V metodě `setUp()` je vytvořen aplikační kontext a inicializována databáze. V metodě `tearDown()` je databáze uzavřena, aplikační kontext odstraněn a dočasná databáze smazána.

Testy jsou rozděleny do jednotlivých metod uvnitř testovací třídy `TestSensors`. Každá metoda testuje specifický aspekt aplikace, jako například získání seznamu senzorů, vytvoření nového senzoru, získání konkrétního senzoru, aktualizaci senzoru a další. Testy simulují HTTP požadavky na API endpointy pomocí `self.app.test_client()` a ověřují očekávané odpovědi a data pomocí assertačních metod.

Například test `test_get_sensors()` ověřuje, zda požadavek GET na endpoint `/sensors` vrací stavový kód 200 a zda jsou vrácená data ve formátu seznamu:

```
def test_get_sensors(self):
    response = self.app.test_client().get('/sensors')
    self.assertEqual(response.status_code, 200)
    data = json.loads(response.get_data(as_text=True))
    self.assertIsInstance(data, list)
```

Další test `test_create_sensor()` ověřuje vytvoření nového senzoru odesláním POST požadavku na endpoint `/sensors` s příslušnými daty:

```
def test_create_sensor(self):
    payload = {
        'type': 'temperature',
        'name': 'Sensor 1',
        'location': 'Room 101'
    }
    response = self.app.test_client().post('/sensors', json=payload)
    self.assertEqual(response.status_code, 201)
    data = json.loads(response.get_data(as_text=True))
    self.assertEqual(data['message'], 'success')
```

Tímto způsobem jsou testovány různé scénáře a funkcionality aplikace, aby se zajistila jejich správná integrace a funkčnost. Použití integračních testů a testovacího frameworku unittest v této práci pomáhá ověřit správnou funkčnost API endpointů a zajistit kvalitu a spolehlivost aplikace. Testy poskytují jistotu, že jednotlivé komponenty spolupracují správně a že aplikace se chová podle očekávání.

5.2.2 Simulace různých scénářů a okrajových případů při testování endpointů

Při testování endpointů API je důležité simulovat různé scénáře a okrajové případy, aby se ověřila robustnost a správná funkčnost aplikace. V této práci jsou testy navrženy tak, aby pokryly různé situace a ověřily chování API v různých podmínkách. Pro názornost budou scénáře vysvětleny na příkladu widgetů, přičemž ostatní komponenty, jako jsou senzory a dashboardy, byly testovány obdobným způsobem.

Jedním z testovaných scénářů je získávání widgetů přiřazených k dashboardu. Test `test_get_widgets()` nejprve vytvoří nový dashboard pomocí POST požadavku na endpoint `/dashboards`. Poté získá ID vytvořeného dashboardu z odpovědi. Následně se provede GET požadavek na endpoint `/dashboards/{dashboard_id}/widgets`, který by měl vrátit seznam widgetů přiřazených k danému dashboardu. Test ověřuje, zda je odpověď ve formátu seznamu a zda má stavový kód 200.

Dalším testovaným scénářem je přidávání widgetu na dashboard. Test `test_add_widget_to_dashboard()` začíná vytvořením nového dashboardu a senzoru. Poté se získají ID vytvořeného dashboardu a senzoru z odpovědí. Následně se připraví data pro widget, včetně ID senzoru a typu widgetu. Tato data se odešlou pomocí POST požadavku na endpoint `/dashboards/{dashboard_id}/widgets`. Test ověřuje, zda je odpověď úspěšná se stavovým kódem 201 a zda obsahuje očekávanou zprávu o úspěšném přidání widgetu na dashboard.

Třetím testovaným scénářem je odebírání widgetu z dashboardu. Test `test_remove_widget_from_dashboard()` začíná podobně jako předchozí testy vytvořením dashboardu, senzoru a přidáním widgetu na dashboard. Poté se získá ID přidaného widgetu pomocí GET požadavku na endpoint `/dashboards/{dashboard_id}/widgets`. Následně se provede DELETE požadavek na endpoint `/dashboards/{dashboard_id}/widgets/{widget_id}`, který by měl odebrat widget z dashboardu. Test ověřuje, zda je odpověď úspěšná se stavovým kódem 200 a zda obsahuje očekávanou zprávu o úspěšném odebrání widgetu z dashboardu.

Kromě těchto scénářů jsou v testech pokryty i další okrajové případy, jako například:

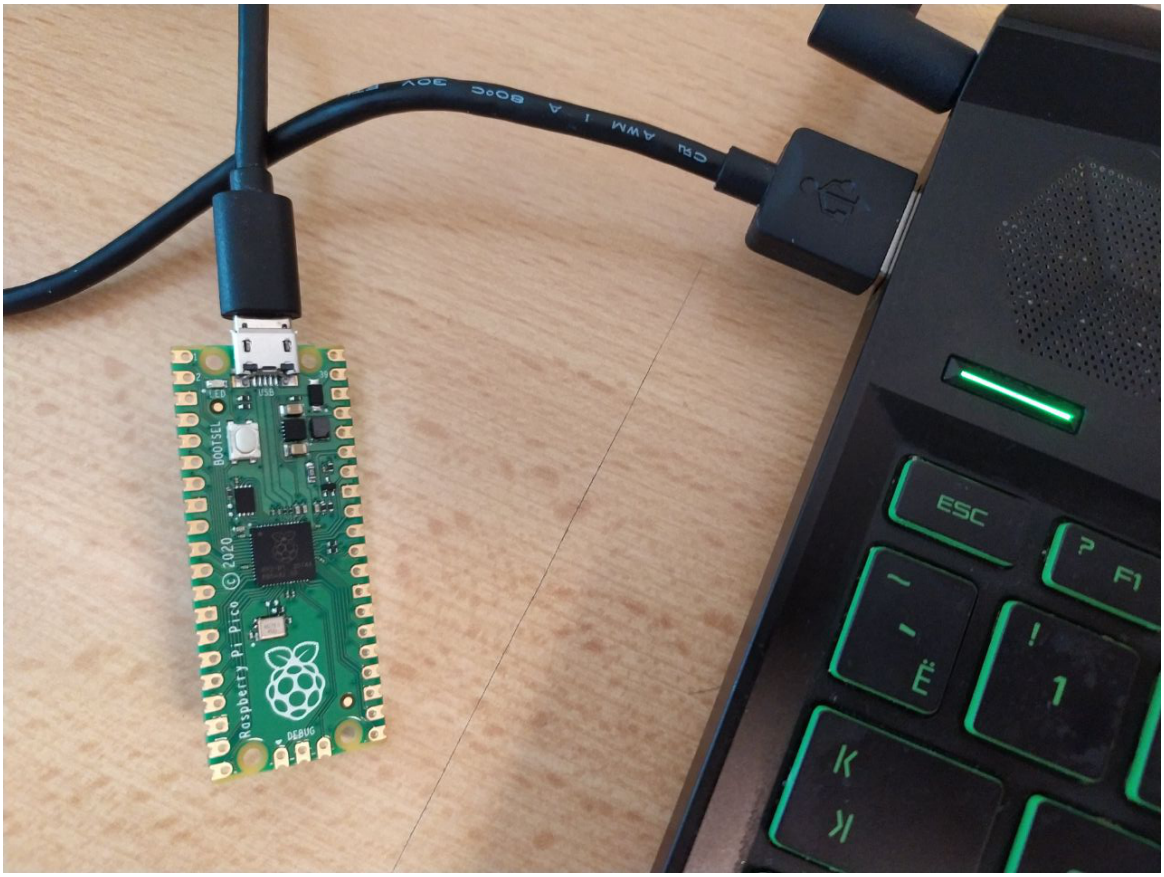
- Získání neexistujícího senzoru pomocí GET požadavku na endpoint `/sensors/{nonexistent_id}`, kde se očekává stavový kód 404 a chybová zpráva.
- Aktualizace senzoru bez zadaných polí pomocí PUT požadavku na endpoint `/sensors/{sensor_id}` s prázdným tělem požadavku, kde se očekává stavový kód 200 a zpráva o tom, že nejsou žádná pole k aktualizaci.

Tyto okrajové případy pomáhají ověřit, jak se API chová v neočekávaných situacích a zda poskytuje správné odpovědi a chybové zprávy.

5.3 Testování celé aplikace s Raspberry Pi Pico

5.3.1 Nastavení a konfigurace Raspberry Pi Pico pro testování aplikace

Pro testování celé aplikace bylo použito zařízení Raspberry Pi Pico. Prvním krokem bylo připojení Raspberry Pi Pico k počítači pomocí USB kabelu, který zajišťoval napájení a přenos dat. Na obrázku 5.1 je znázorněno připojení Raspberry Pi Pico k notebooku.



Obrázek 5.1: Raspberry Pi Pico

Po připojení Raspberry Pi Pico k počítači bylo potřeba nahrát program v jazyce MicroPython. K nahrání programu do Raspberry Pi Pico byl použit nástroj Thonny, který poskytuje integrované vývojové prostředí (IDE) pro MicroPython. Program byl napsán v Thonny a poté nahrán do Raspberry Pi Pico pomocí funkce "Save as"a výběrem cílového zařízení jako "Raspberry Pi Pico".

Jedním z klíčových aspektů konfigurace Raspberry Pi Pico bylo připojení k bezdrátové síti WiFi. Následující kód ukazuje, jak bylo provedeno připojení k WiFi síti:

```
1 import network
2
3 wlan = network.WLAN(network.STA_IF)
4 wlan.active(True)
```

```
wlan.connect("Redmi Note 7", "123456789")

while wlan.isconnected() == False:
  pass

print('is connected')
print(wlan.ifconfig())
```

V následujících sekcích bude popsáno, jak bylo Raspberry Pi Pico integrováno s aplikací a jak byly simulovány různé scénáře a případy použití pro testování celé aplikace.

5.3.2 Simulace senzorů a odesílání dat pomocí Raspberry Pi Pico

Po nastavení a konfiguraci Raspberry Pi Pico bylo dalším krokem simulovat data ze senzorů a odesílat je do aplikace prostřednictvím MQTT brokeru. K tomuto účelu byl použit veřejný MQTT broker "broker.emqx.io", který umožňuje snadné publikování a odběr zpráv. Následující kód ukazuje, jak bylo simulováno generování dat různých napětí a jejich odesílání na MQTT broker:

```
import time
import random
from umqtt.simple import MQTTClient

broker_address = "broker.emqx.io"
topic = "electricity/sensor"
sensor_id = "9bc8ac18-dcbf-4790-97f4-6f960a4aed97"

def simulate_data():
    voltage_1 = random.uniform(220, 240)
    voltage_2 = random.uniform(220, 240)
    voltage_3 = random.uniform(220, 240)
    voltage_4 = random.uniform(220, 240)
    voltage_5 = random.uniform(220, 240)
    return voltage_1, voltage_2, voltage_3, voltage_4, voltage_5

def connect_and_publish():
    client = MQTTClient("umqtt_client", broker_address)
    client.connect()
    print("Connected to MQTT broker")

    try:
        while True:
            timestamp_ns = int(time.time() * 1e9)
            voltage_1, voltage_2, voltage_3, voltage_4, voltage_5 =
                simulate_data()
            line_protocol_message = f"{sensor_id} voltage_1={voltage_1},
                voltage_2={voltage_2},voltage_3={voltage_3},voltage_4={
                voltage_4},voltage_5={voltage_5} {timestamp_ns}"
            print(line_protocol_message)
            client.publish(topic, line_protocol_message)
            time.sleep(10)
    except KeyboardInterrupt:
        print("Interrupted")
    finally:
        client.disconnect()
```

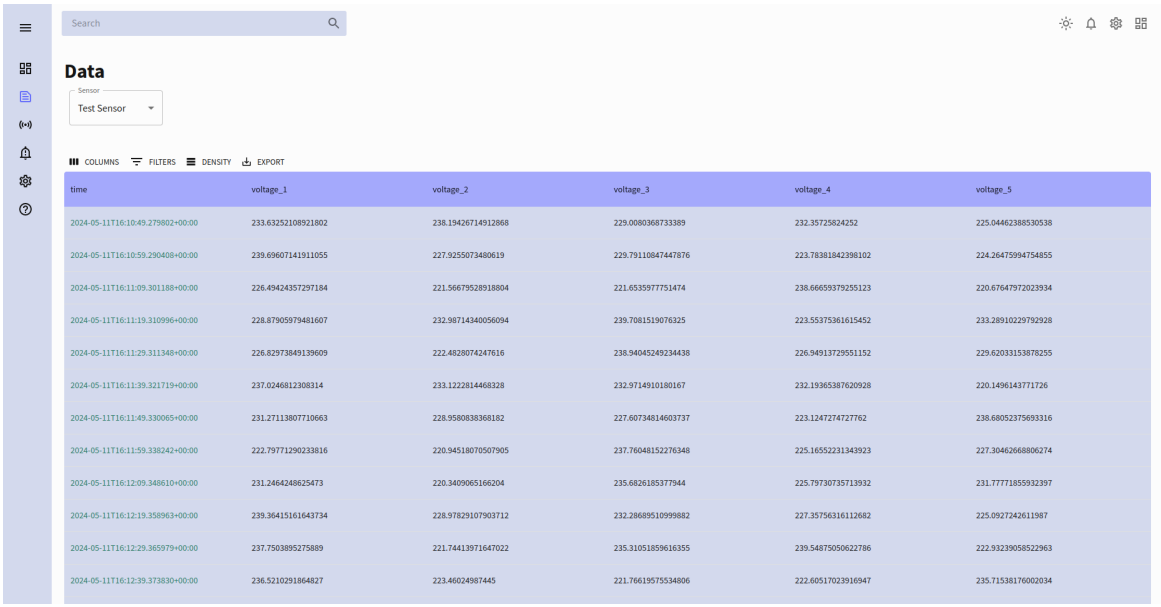
```
34
35 connect_and_publish()
```

V tomto kódu je definována funkce `simulate_data()`, která generuje náhodné hodnoty napětí v rozsahu od 220 do 240 voltů pro pět různých měření. Funkce `connect_and_publish()` se připojuje k MQTT brokeru pomocí klienta MQTTClient a poté v nekonečné smyčce generuje data, vytváří zprávu ve formátu Line Protocol a publikuje ji na téma `electricity/sensor`. Zpráva obsahuje ID senzoru, naměřené hodnoty napětí a časové razítko v nanosekundách.

Před spuštěním tohoto kódu na Raspberry Pi Pico bylo potřeba vytvořit odpovídající senzor v aplikaci. Obrázek 5.2 ukazuje stránku vytvořeného senzoru v aplikaci.

Obrázek 5.2: Nový senzor

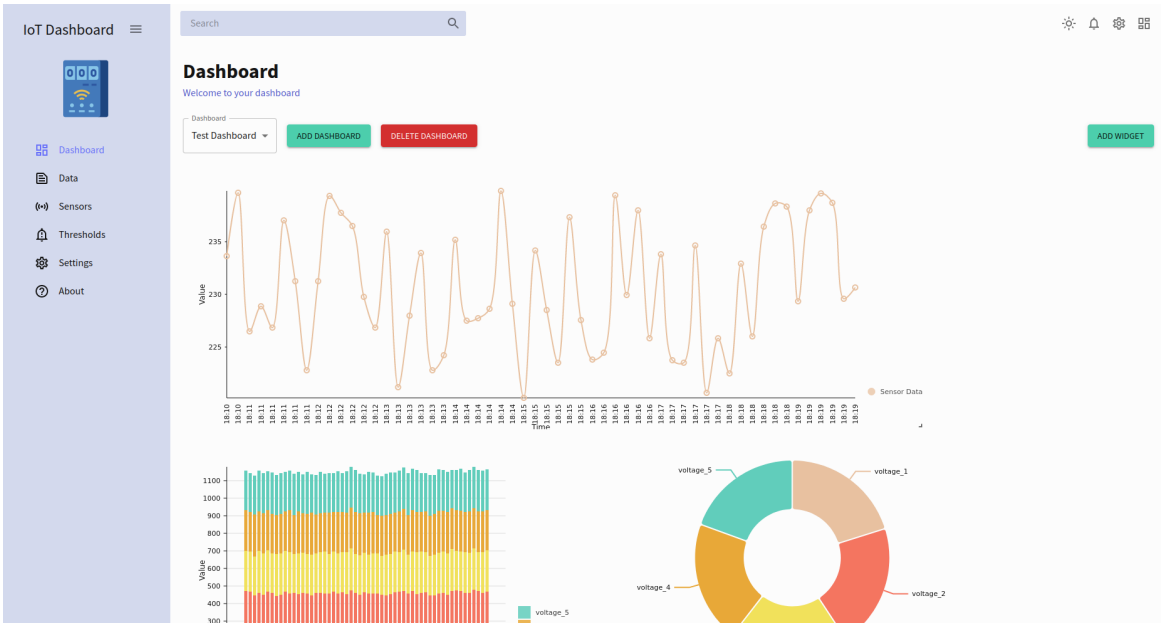
Po spuštění kódu na Raspberry Pi Pico se začala generovat simulovaná data po dobu přibližně 10 minut. Tato data byla odesílána na MQTT broker a následně sbírána pomocí Telegrafu, který je nakonfigurován pro příjem dat z MQTT a jejich ukládání do databáze InfluxDB. Výsledky simulace lze vidět na stránce `Data` v aplikaci, kde jsou data zobrazena v tabulce. Na obrázku 5.3 je ukázka tabulky s naměřenými hodnotami napětí.



time	voltage_1	voltage_2	voltage_3	voltage_4	voltage_5
2024-05-11T16:10:49.279802+00:00	233.63252108921802	238.19426714912868	229.0080368733389	232.35725824252	225.04462388530538
2024-05-11T16:10:59.290408+00:00	239.69607141911055	227.9255073480619	229.79110847447876	223.78381842398102	224.26475994754855
2024-05-11T16:11:09.301188+00:00	226.49424357297184	221.56679528918804	221.65359777751474	238.66659379255123	220.67647972023934
2024-05-11T16:11:19.310996+00:00	228.87905979481607	232.98714340056094	239.7081519076325	223.55375361615452	233.28910229792928
2024-05-11T16:11:29.311348+00:00	226.82973849139609	222.4828074247616	238.94045249234438	226.94913729551152	229.62033153878255
2024-05-11T16:11:39.321719+00:00	237.0246812308314	233.1222814468328	232.9714910180167	232.19365387620928	220.1496143771726
2024-05-11T16:11:49.330065+00:00	231.27113807710663	228.9580838368182	227.60734814603737	223.1247274727762	238.68052375693316
2024-05-11T16:11:59.338242+00:00	222.79771290233816	220.94518070507905	237.76048152276348	225.16552231343923	227.30462668806274
2024-05-11T16:12:09.348610+00:00	231.2464248625473	220.3409065166204	235.6826185377944	225.79730735713932	231.77771855932397
2024-05-11T16:12:19.358963+00:00	239.36415161643734	228.97829107903712	232.286895109999882	227.35756316112682	225.0927242611987
2024-05-11T16:12:29.365979+00:00	237.7503895275889	221.744139716470 22	235.31051859616355	239.54875050622786	222.93239058522963
2024-05-11T16:12:39.373830+00:00	236.5210291864827	223.46024987445	221.76619575534806	222.60517023916947	235.71538176002034

Obrázek 5.3: Data

Kromě zobrazení dat v tabulce byla také vytvořena stránka `Dashboard`, kde byly přidány grafy zobrazující průběh naměřených hodnot v čase. Na obrázku 5.4 je ukázka dashboardu s grafy pro jednotlivá napětí.



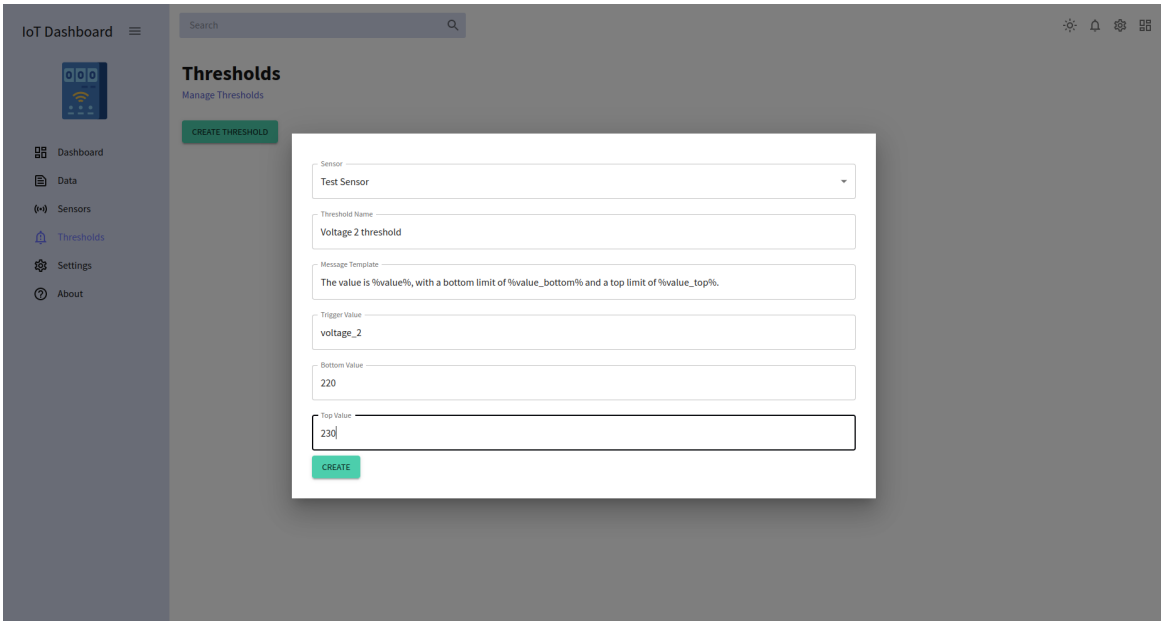
Obrázek 5.4: Grafy

Tímto způsobem bylo demonstrováno, jak lze pomocí Raspberry Pi Pico simulovat data ze senzorů, odesílat je na MQTT broker a následně je zobrazovat a vizualizovat v aplikaci. Tento proces umožnil otestovat celou aplikaci od sběru dat až po jejich prezentaci uživateli.

5.3.3 Testování prahových hodnot a upozornění

Důležitou součástí aplikace je možnost nastavení prahových hodnot pro jednotlivé vlastnosti senzorů a zasílání upozornění při překročení těchto hodnot. Pro otestování této funkcionality byl vytvořen nový práh pro vlastnost `voltage_2` senzoru.

Na obrázku 5.5 je znázorněno vytvoření nového prahu pro vlastnost `voltage_2`. Práh byl nastaven v rozmezí od 220 do 230 voltů. To znamená, že upozornění se odešle, pokud hodnota `voltage_2` překročí 230 voltů nebo klesne pod 220 voltů.



Obrázek 5.5: Vytvoření nového prahu

Po nastavení prahu bylo spuštěno generování dat pomocí Raspberry Pi Pico. Pico generovalo data s různými hodnotami napětí, včetně hodnot překračujících nastavený práh pro `voltage_2`.

Když hodnota `voltage_2` překročila horní hranici prahu 230 voltů nebo klesla pod spodní hranici 220 voltů, aplikace úspěšně odeslala upozornění na Discord. Na obrázku 5.6 je vidět, že zpráva o překročení prahu byla doručena na Discord kanál.

Obrázek 5.6: Upozornění na Discord

Pro ověření, že upozornění jsou odesílána pouze při překročení nastavených mezí prahu, bylo provedeno další testování. V nastavení aplikace byla funkce upozornění pro daný práh vypnuta. Po opětovném generování dat Raspberry Pi Pico s hodnotami překračujícími nastavené meze se upozornění na Discord již neodesílala. To potvrzuje, že funkce upozornění pracuje správně a respektuje nastavení v aplikaci.

Tímto testem byla ověřena funkčnost nastavení prahových hodnot a zasílání upozornění. Aplikace úspěšně detekuje překročení nastavených mezí prahů, ať už se jedná o překročení horní hranice nebo pokles pod spodní hranici. Upozornění jsou odesílána na specifikovaný Discord kanál. Zároveň je možné upozornění v nastavení vypnout, což bylo také úspěšně otestováno.

5.4 Výsledky testování

V průběhu vývoje a testování aplikace Dashboard bylo provedeno několik důležitých testů, které ověřily funkčnost a spolehlivost jednotlivých komponent a celého systému. Testy zahrnovaly ověření správnosti sběru dat ze senzorů, ukládání dat do databáze, zobrazování dat v uživatelském rozhraní, nastavení prahových hodnot a zasílání upozornění.

- Testování sběru dat ze senzorů:
  - Pomocí Raspberry Pi Pico byly simulovány různé scénáře a generována data z virtuálních senzorů.
  - Testy prokázaly, že aplikace dokáže spolehlivě přijímat data ze senzorů prostřednictvím MQTT protokolu a ukládat je do databáze InfluxDB.
  - Data byla úspěšně přijímána a ukládána v očekávaném formátu a intervalu.
- Testování ukládání dat do databáze:
  - Byla ověřena integrita a konzistence dat uložených v databázi InfluxDB.
  - Testy potvrdily, že data jsou ukládána ve správném formátu, s odpovídajícími časovými razítky a hodnotami.
  - Databáze byla schopna efektivně zpracovávat a ukládat velké objemy dat bez ztráty nebo poškození.
- Testování zobrazování dat v uživatelském rozhraní:
  - Uživatelské rozhraní aplikace bylo testováno z hlediska správného zobrazování dat ze senzorů.
  - Testy zahrnovaly ověření funkčnosti grafů, tabulek a dalších vizuálních prvků.
  - Data byla úspěšně načítána z databáze a zobrazována v reálném čase.
  - Uživatelské rozhraní poskytovalo přehledné a intuitivní zobrazení dat z různých senzorů.
- Testování nastavení prahových hodnot a zasílání upozornění:
  - Byla testována funkčnost nastavení prahových hodnot pro jednotlivé vlastnosti senzorů a zasílání upozornění při překročení těchto prahů.
  - Testy potvrdily, že aplikace správně detekuje překročení nastavených mezí a odesílá upozornění na specifikovaný komunikační kanál (Discord).

- Upozornění byla zasílána pouze v případě překročení prahových hodnot a respektovala nastavení v aplikaci.

Testování aplikace bylo klíčovým krokem v procesu vývoje a pomohlo ověřit, že aplikace splňuje očekávané požadavky a je připravena pro nasazení v reálném prostředí. Díky důkladnému testování lze s jistotou říci, že aplikace Dashboard poskytuje spolehlivé a efektivní řešení pro sběr, ukládání a vizualizaci dat ze senzorů a nabízí uživatelům hodnotné funkce pro sledování a analýzu těchto dat.

Kapitola 6

Závěr

Cílem této bakalářské práce bylo vytvořit webovou aplikaci pro monitorování a vizualizaci dat z IoT zařízení. Aplikace měla umožňovat sběr dat pomocí protokolu MQTT, jejich ukládání do databáze a následnou vizualizaci pomocí grafů a dalších prvků. Kromě toho měla aplikace poskytovat možnost definovat a odesílat upozornění na základě nastavených podmínek.

Během implementace se podařilo splnit většinu stanovených požadavků. Aplikace umožňuje sběr dat z IoT zařízení pomocí protokolu MQTT a jejich ukládání do databáze InfluxDB. Data jsou následně vizualizována pomocí interaktivních grafů a dalších prvků, které poskytují uživateli přehledný náhled na sledované metriky. Aplikace také umožňuje definovat upozornění na základě nastavených podmínek a odesílat je do komunikačního kanálu.

Přestože se podařilo implementovat základní funkcionalitu aplikace, existuje několik oblastí, ve kterých by bylo možné aplikaci dále vylepšit a rozšířit:

- Grafy v aplikaci momentálně neumožňují nastavení časových intervalů, což omezuje možnosti analýzy dat v různých časových obdobích.
- Aplikace neposkytuje možnost nastavit, jak často bude Telegraf číst data z MQTT a ukládat je do InfluxDB. Tato funkce by umožnila větší flexibilitu při sběru dat.
- Podpora více komunikačních protokolů a možnost komunikace s různými typy zařízení by rozšířila použitelnost aplikace v různých IoT scénářích.
- Upozornění jsou v současné verzi aplikace odesílána pouze do komunikačního kanálu a nejsou nikde ukládána. Možnost ukládat a spravovat upozornění přímo v aplikaci by poskytla uživatelům lepší přehled a kontrolu nad vzniklými událostmi.
- Aplikace by mohla být rozšířena o pokročilejší možnosti vizualizace dat, jako jsou například pokročilé typy grafů, filtry nebo možnost kombinovat data z různých zdrojů.
- Uživatelské rozhraní aplikace by mohlo být dále optimalizováno pro lepší použitelnost a intuitivnost ovládání.

Je důležité zmínit, že výše uvedené nedostatky a omezení aplikace mohou být vyřešeny v rámci budoucího vývoje a rozšíření aplikace. Další vývoj by se měl zaměřit na implementaci

chybějících funkcionalit, vylepšení uživatelského rozhraní a celkovou optimalizaci aplikace. Pravidelné testování s cílovou skupinou uživatelů by mělo být nedílnou součástí procesu dalšího vývoje, aby bylo zajištěno, že aplikace splňuje požadavky a očekávání uživatelů.

Věřím, že tato bakalářská práce položila základy pro Dashboard aplikaci pro monitorování a vizualizaci IoT dat a poskytla solidní základ pro další vývoj a vylepšení. S dalším rozšířením a optimalizací má tato aplikace potenciál stát se užitečným nástrojem pro správu a analýzu dat v IoT prostředí.

Literatura

[1] KRISHNAMURTHI, R. et al. An Overview of IoT Sensor Data Processing, Fusion, and Analysis Techniques. *Sensors*. 2020, 20, 21, s. 1–23.

[2] RAMALINGAM, H. – VENKATESAN, P. Analysis of Current Trends in Internet of Things Gateway and Edge Data Processing Characteristics. *International Journal of Engineering Research and Technology*. 2019, 8, 7, s. 34–40.

[3] web:adafruit. About Adafruit IO. `https://www.io.adafruit.com/`, stav z 2. 4. 2024.

[4] web:arduino. About Arduino. `https://www.cloud.arduino.cc/`, stav z 2. 4. 2024.

[5] web:blynk. About Blynk. `https://www.blynk.io/`, stav z 2. 4. 2024.

[6] web:cayenne. About Cayenne. `https://www.iies.in/blog/what-is-cayenne-iot-and-how-does-it-work/`, stav z 2. 4. 2024.

[7] web:developer.mozilla. Fetch API, . `https://www.developer.mozilla.org/en-US/docs/Web/API/Fetch_API/`, stav z 30. 4. 2024.

[8] web:developer.mozilla. History API, . `https://www.developer.mozilla.org/en-US/docs/Web/API/History-API/`, stav z 30. 4. 2024.

[9] web:flask. Flask's documentation. `https://www.flask.palletsprojects.com/en/2.3.x/`, stav z 24. 4. 2024.

[10] web:medium. MUI5 Color Themes with Dark/Light Modes + Theme color design. `https://www.medium.com/@facinick/reactjs-mui5-color-themes-with-dark-light-modes-theme-color-design-27039fd3d5de/`, stav z 30. 4. 2024.

[11] web:mui. MUI Core. `https://www.mui.com/core/`, stav z 24. 4. 2024.

[12] web:nivo. About Nivo. `https://www.nivo.rocks/`, stav z 24. 4. 2024.

[13] web:particle. About Particle.
`https://www.particle.io/`, stav z 2. 4. 2024.

[14] web:reactjs. React.js.
`https://www.reactjs.org/`, stav z 30. 4. 2024.

[15] web:sitewhere. About SiteWhere.
`https://www.sitewhere1.sitewhere.io/`, stav z 2. 4. 2024.

[16] web:thingsboard. About thethings.IO, .
`https://www.thethings.io/`, stav z 2. 4. 2024.

[17] web:thingsboard. About ThingsBoard, .
`https://www.thingsboard.io/`, stav z 2. 4. 2024.

[18] web:thingspeak. About ThingSpeak.
`https://www.thingspeak.com/`, stav z 2. 4. 2024.

[19] web:ubidots. About Ubidots.
`https://www.ubidots.com/`, stav z 2. 4. 2024.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

BLE Bluetooth Low Energy

CMMS Computerized Maintenance Management System

CoAP Constrained Application Protocol

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

IoT Internet of Things

JSON JavaScript Object Notation

LDAP Lightweight Directory Access Protocol

LPWAN Low-power wide-area network

MES Manufacturing Execution System

MQTT Message Queuing Telemetry Transport

MUI Material-UI

OTA Over-the-Air

PaaS Platform-as-a-Service

REST Representational State Transfer

SCADA Supervisory Control And Data Acquisition

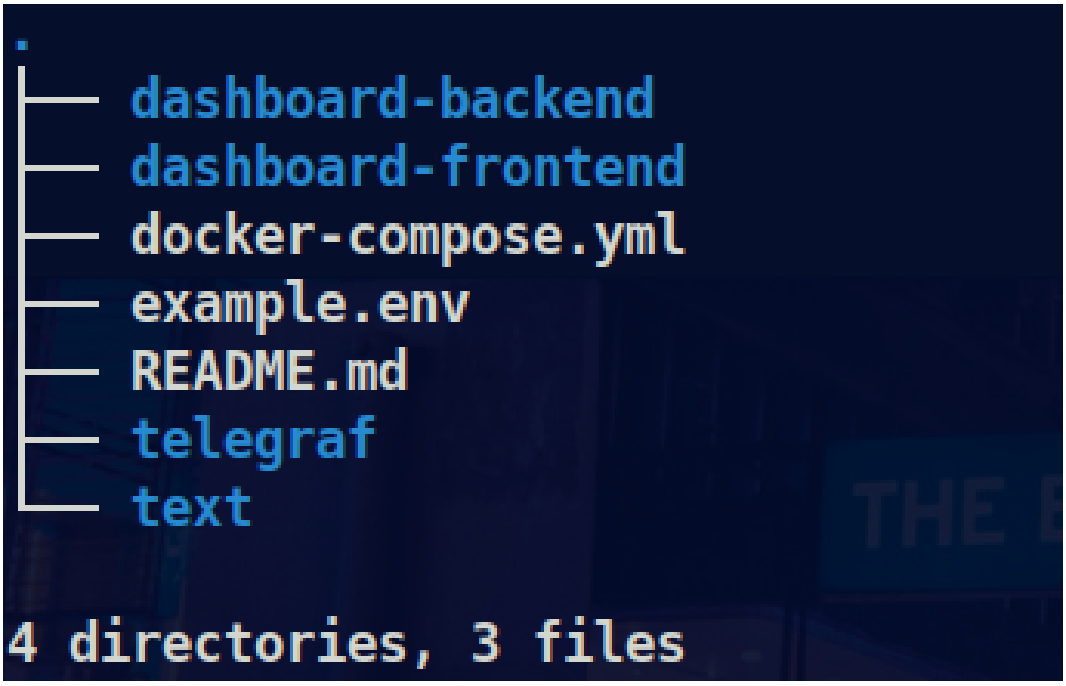
TOML Tom's Obvious, Minimal Language

WSN Wireless sensor network

Příloha B

Obsah přílohy

Obsah přílohy, která je nahrána na KOS.



Obrázek B.1: Seznam přílohy

Instalační návod je uveden v souboru `README.md`.