VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÁ VIZUALIZACE DAT Z CHYTRÝCH ZAŘÍZENÍ V IOT

WEB VISUALIZATION OF SMART DEVICES DATA IN IOT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. PETR JOHN**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JIŘÍ HYNEK, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání diplomové práce

Student: **John Petr, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Informační systémy a databáze
Název: **Webová vizualizace dat z chytrých zařízení v IoT**
Web Visualization of Smart Devices Data in IoT
Kategorie: Uživatelská rozhraní
Zadání:

1. Prostudujte oblast internetu věcí (Internet of Things, IoT), typy chytrých zařízení a způsobu komunikace a zpracování dat z těchto zařízení.
2. Prostudujte principy tvorby přehledových obrazovek typu dashboard a existující typy vizualizací vhodných pro jejich návrh.
3. Prostudujte technologie pro tvorbu webových uživatelských rozhraní a vizualizací dat (např. Angular, D3.js, Leaflet, apod.). Proveďte průzkum existujících vizualizačních knihoven.
4. Proveďte analýzu požadavků na vizualizaci dat získaných ze senzorů chytrých zařízení. Zaměřte se na problematiku vizualizaci jejich stavu (klíčové ukazatele výkonnosti - KPI, detekce anomálií apod.).
5. Navrhněte webové komponenty a sadu pohledů využívající tyto komponenty, které dokáží zpracovat a přívětivě graficky vizualizovat data ze senzorů chytrých zařízení.
6. Navržené rozšíření implementujte.
7. Výsledné řešení otestujte z hlediska jeho použitelnosti. Navrhněte možná rozšíření.

Literatura:

- Greengard, S.: The Internet of Things. MIT Press, 2015, ISBN 978-026-2527-736.
- Few, S.: Information Dashboard Design: The Effective Visual Communication of Data. Sebastopol, USA: O'Reilly, 2006, ISBN 978-059-6100-162.
- Angular: Introduction to the Angular Docs [online]. 2021 [cit. 2021-09-03]. Dostupné z: https://angular.io/docs
- Interní dokumentace firmy Logimic

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Hynek Jiří, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 18. května 2022
Datum schválení: 11. října 2021

Abstrakt

Se stále rostoucím počtem chytrých zařízení roste i potřeba z nich data ukládat, agregovat a zpracovávat. Získaná data je poté nutné uživateli zpřístupnit efektivním způsobem, díky kterému bude schopný rychle a efektivně reagovat na stav sledovaného zařízení. Tato diplomová práce se zabývá jejich ukládáním a zobrazováním za pomocí dashboardů. Cílem práce bylo vytvořit knihovnu, která bude obsahovat komponenty určené k zobrazování těchto dat a jednoduchou backendovou aplikaci poskytující API schopnou efektivně ukládat a dotazovat se na data ze senzorů. Knihovna je představena v jednoduché aplikaci, která využívá vytvořenou backendovou aplikaci. Některé části tohoto řešení byly nasazeny ve firmě Logimic, kde díky nasazení API vrstvy došlo mimo jiné ke snížení nákladů na provoz databázového systému na 15 % průměrné ceny dosavadního systému za poslední 2 měsíce. Zbylé části a přístupy jsou nyní postupně integrovány do existujících systémů firmy.

Abstract

Due to the constant growth in the count of smart devices the need to store, aggregate and process the data effectively rises. The gathered data needs to be displayed effectively to the user, who will be able to react to the current state of the monitored device. This master's thesis deals with storing the data and displaying them using dashboards. The goal of this thesis was the creation of a library composed of components intended for data visualization and a simple API application, that can store and query the data effectively. The library is introduced in a simple application that uses the API application. Some parts of this solution are currently used in the company Logimic, where the costs of the database system reduced to 15 % of the average cost of the previous database system in the last 2 months. The rest of the solution is currently being integrated into the existing systems of the company.

Klíčová slova

dashboard, vizualizace, IoT, Internet věcí, chytrá zařízení, zobrazení dat, časové série

Keywords

dashboard, visualization, IoT, Internet of Things, smart devices, data visualisation, time series

Citace

JOHN, Petr. *Webová vizualizace dat z chytrých zařízení v IoT*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Hynek, Ph.D.

Webová vizualizace dat z chytrých zařízení v IoT

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jiřího Hynka, Ph.D. Další informace mi poskytl pan Ing. Michal Valný, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Petr John
9. května 2022

Poděkování

Rád bych poděkoval panu Ing. Jiřímu Hynkovi, Ph.D. a panu Ing. Michalu Valnému, Ph.D. za poskytnutou pomoc a konzultace při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Se stále rostoucím počtem chytrých zařízení roste i množství dat, která je zapotřebí zobrazit. Díky tomuto nárůstu se neustále zvyšují nároky na použité vizualizace, které musí svému uživateli zobrazit data takovým způsobem, aby mohl rychle a aktivně reagovat na zobrazené informace. V dnešní době bohužel nejsou časté platformy, které by poskytovaly možnost tohoto zobrazení pokud možno pro uživatele jednoduchým a interaktivním způsobem.

Příkladem oblasti, kde je této platformy zapotřebí může být vytápění, kde v minulosti každé domácnosti stačil klasický kotel na tuhá paliva, v dnešní době má většina z nich kotel na jiná paliva (plyn, elektrický) nebo automatický kotel na tuhá paliva. Mnoho z těchto zařízení poskytuje možnost nastavit způsob vytápění bezdrátově za pomocí mobilní aplikace z pohodlí obývákového gauče. Tímto se řadí mezi takzvaná IoT zařízení, tedy zařízení, která plní nějakou specifickou funkci (například vytápění), je připojeno k internetu a dokáže přes něj komunikovat s dalším zařízením ve stejné síti. Počet a zájem o tato zařízení neustále roste, nejsou zaměřená pouze na vytápění ale také na další aspekty našeho každodenního života. Příkladem dalších zařízení může být nositelná elektronika jako chytré hodinky nebo fitness náramky, solární panely, a to jak fotovoltaické, tak panely pro ohřev vody, automatické brány, 3D tiskárny a další. Výrobci těchto zařízení často poskytují i zmíněné aplikace k jejich ovládání, ale ty často zobrazují pouze část aktuálního stavu.

Mnoho IoT zařízení dovoluje získávat přes internet svůj stav a tím zpřístupňuje možnost zkontrolování správné funkce nebo stavu zařízení na dálku. Otázkou ale zůstává jak data z jednotlivých zařízení získávat, ukládat a zobrazovat uživateli tak, aby je byl schopen bezpracně a bezstarostně vyhodnotit.

Tato situace se ale netýká pouze osobních zařízení a domácích spotřebičů, ale také z velké části průmyslu, který se postupem času stále více automatizuje a operace, které dříve bylo nutné vykonávat ruční prací jsou vykonávány strojově. Tyto stroje ale nejsou bezchybné a je potřeba je monitorovat. Část monitorování je také možné automatizovat, ale nakonec je vždy zapotřebí lidská kontrola.

Tento úkol se na první pohled může jevit jako jednoduchý, bohužel každé zařízení často vyžaduje unikátní způsob zobrazení, a to nejen například kvůli povaze daných dat, ale také například kvůli tomu, že nás nezajímají pouze holá data, ale například extrémy hodnot nebo relace mezi hodnotami samotnými. Práce s daty také přináší problémy, se kterými je nutné se vypořádat. Data mohou obsahovat špatné nebo zkreslené záznamy, nebo jich může být příliš velké množství pro smysluplné zpracování. Problematikou efektivního zobrazování a ukládání se zabývá poměrně velké množství firem. Jednou z nich je i firma Logimic, se kterou jsem při řešení této diplomové práce spolupracoval.

Tato diplomová práce se snaží vyřešit některé ze zmíněných problémů, a to hlavně způsoby, jakými je možné data efektivně zobrazit uživateli v takovém formátu, aby pro něj byla jednoduše interpretovatelná a efektivní způsoby uložení velkého množství dat ze senzorů. Za těmito účely budou vytvořeny dvě aplikace. První aplikace založená na aplikačním rámci Angular a bude sloužit k zobrazení dat. Druhá aplikace bude sloužit k demonstraci efektivních způsobů ukládání potřebných dat.

Kapitola 2 poskytuje přesnější popis Internetu věcí, přehled senzorů, které je v této oblasti možné využít a způsoby komunikace. Způsoby efektivního uložení dat z IoT zařízení jsou obsaženy v kapitole 3. Kapitola 4 obsahuje popis pohledů dashboard, vizualizace v nich použitelné, popis KPI a přístupy k detekci anomálií. Kapitola 5 obsahuje popis technologií, které je možné využít k vytváření uživatelských rozhraní. V kapitole 6 je obsažena analýza aktuálního stavu ve firmě Logimic. Návrh řešení je obsažen v kapitole 7. V následující kapitole 8 je obsažena implementace tohoto řešení následovaná kapitolou 9.3 obsahující testování vytvořeného řešení.

Kapitola 2

Internet věcí

Internet věcí, IoT z anglického *Internet of Things*, je možné obecně definovat jako věci nebo objekty, které jsou připojeny nějakou technologií k internetu, nebo k dalším IoT zařízením. Zájem o zařízení tohoto typu rychle roste, nejen v soukromém prostoru, ale i v průmyslu. Důkazem může být například i to, že odhadovaný počet IoT zařízení v roce 2021 podle některých zdrojů je 10 – 12 miliard [1, 24] a stále se odhaduje další nárůst. Z praktického pohledu musí být každému zařízení přidělen unikátní identifikátor (UID) a IP adresa, aby s ním bylo možné komunikovat přes některou z vybraných technologií, mezi které patří například WiFi nebo tradiční ethernet.

Kromě tradičního chápání IoT existují ještě další způsoby [19], které tuto problematiku popisují jinak. Některé zdroje používají pojem „propojená zařízení“ pro jakákoli zařízení, která jsou spolu spojena. Tato zařízení nutně nemusí být připojena do internetu, ale stále častěji se k němu připojují. Další zdroje používají pouze podkategorii IoT, kterou nazývají Industriální internet. Tento Industriální internet se specializuje pouze na zařízení, která je možné použít v průmyslu a často dělí zařízení v něm podle toho, jestli komunikují s uživatelem nebo pouze s ostatními zařízeními v síti. Posledním typem, který stojí za zmínění, je takzvaný internet všeho, Internet of Everything, který se více vrací k původní definici IoT a byl navržen firmou Cisco. Tato práce používá původní definici IoT a to z toho důvodu, že ostatní definice buď vychází z ní buď vychází, nebo se zaměřují na její podmnožinu.

2.1 Typy sensorů

Ve světe IoT se nachází velký počet typů sensorů a způsobů, jak je možné na ně nahlížet při jejich dělení. Díky těmto dělením je často možné vybrat odpovídající způsob nejen uložení ale i zobrazení uživateli. Například data z GPS je nutné uživateli zobrazit jiným způsobem než naměřenou teplotu za poslední hodinu.

Prvním z možných způsobů dělení je dělení na základě měřené fyzikální veličiny [37], například na světelné senzory, které měří intenzitu světla, senzory teploty, vlhkosti, vzdálenosti a další. Toto rozdělení je ale zaměřené na měřenou veličinu, kterou je možné reprezentovat číselně a kvůli tomu nám nerozdělí velkou skupinu senzorů.

Druhým způsobem je dělení na základě toho, k čemu jsou jejich data využita při řešení různých problémů, například na základě toho, jak je možné je využít pro zlepšení využití elektrické energie [6], kde senzory rozdělují následovně:

- senzory obsazenosti – řeší, zda-li je daný objekt aktuálně obsazený či obývaný, například pomocí pohybových senzorů, senzorů reagujících na vibrace a dalších,

- senzory prostředí – měří podmínky, které jsou aktuálně v daném objektu, tedy například teplotu, kvalitu vzdudu, množství CO_2, za pomocí senzorů k měření těchto fyzikálních veličin,
- monitory napájení – specializované senzory, které měří aktuální využití elektrické energie v daném objektu,
- ostatní senzory – jakékoli další senzory, které je možné využít ke zlepšení využití elektrické energie jako například elektronické zásuvky, chytré žárovky a další.

Třetí způsob dělí senzory podle způsobu komunikace, kterou využívají pro přenos dat [2]. K tomuto účelu se většinou používají bezdrátové technologie a to hlavně z toho důvodu, že přivedení klasického internetového spojení může být nepraktické, například protože je zařízení na nedostupném místě, nebo je pohyblivé.

Bezdrátové technologie se často dělí podle dosahu technologie na nízkonapěťové rozlehlé sítě z anglického *Low Power Wide Area Network* a sítě krátkého dosahu. Nejznámějšími zástupci nízko napěťových rozlehlých sítí jsou následující technologie:

- SigFox – technologie vhodná pro přenášení malého množství dat mezi IoT zařízeními do maximální vzdálenosti 50 kilometrů za pomocí velmi krátkých vln [2].
- Celulární sítě – tradiční technologie většinou používaná pouze k připojení telefonních zařízení, ale je možné je využít i k připojení IoT zařízení. Oproti technologii SigFox poskytují celulární sítě dostupnost na větší vzdálenosti, ale jsou více vhodné k ovládání uživatelem než k přenosu dat mezi zařízeními [2].
- LoRaWAN – technologie založená na rádiové komunikaci. Zařízení v ní spadají do dvou typů a to brány a koncová zařízení. Komunikace probíhá v topologii hvězda, kdy mnoho koncových zařízení komunikuje s právě jednou bránou. [21]

Pro komunikaci v malých sítích, nebo sítích, kde je zapotřebí přenášet velké množství dat mezi zařízeními, je častější využití sítí s kratším dosahem. K těmto technologiím patří následující:

- 6LoWPAN – technologie využívající protokol IPv6 pro přenos dat přes bezdrátové osobní sítě s nízkou spotřebou. Jak již napovídá název, tato technologie využívá standardizovaný protokol IPv6, a je tedy možné jednoduše komunikovat s tradičními zařízeními. Narozdíl od tradiční WiFi se tato technologie zaměřuje na nízkou spotřebu, cenu, dosah a cenu [33].
- Bluetooth – technologie pro přenos dat v osobních sítích, kvůli nedostatkům z pohledu použití v IoT, konkrétně vysoké spotřeby, pomalé inicializace a omezeného počtu zařízení je nyní nahrazován technologií BLE, bluetooth low energy, která není s původní technologií kompatibilní, ale snaží se odstranit její nevýhody, jako je například například vyšší spotřeba [9].
- RFID – technologie založená na unikátních ID tagů, které je možné číst použitím rádiových antén, často využívané pro kontrolu přístupu, celní systémy nebo v logistice [42]. V dnešní době je spíše používaná varianta takzvaného pasivního RFID, která nepotřebuje napájení tagu.

- NFC – technologie NFC je založená na technologii RFID a je také určena na přenos dat na velmi krátké vzdálenosti. Narozdíl od RFID poskytuje možnost pouze pasivního tagu, každé spojení je omezeno pouze na dvě zařízení. Technologie je často využívána pro platby mobilním telefonem [11].

Kromě těchto rozdělení existují i zařízení, která na první pohled nejsou tradičními senzory a často je tak nechápeme. Příkladem mohou být například kamerové systémy, které mohou nejen plnit funkce jiných senzorů. Záznam z kamery může být například využít k detekci pohybu, ale také poskytují unikátní možnosti. Příkladem těchto možností u již zmíněných kamerových systémů může být například detekce hustoty provozu a přizpůsobení časování semaforů za účelem zvýšení plynulosti provozu [41].

Všechny tyto senzory ale mají společné vlastnosti. Hlavní podobnou vlastností je fakt, že senzory sledují závislost fyzické veličiny nebo stavu v aktuálním čase. Díky tomu se dají tato data chápat jako takzvané časové řady nebo časové série (anglicky *time series*). Při práci s nimi často rozdělujeme[1] časové série na série událostí a měření. Typičtější jsou měření, kdy je ve stanoveném čase měřena hodnota senzoru. Tento přístup nám dovoluje zachytávat měřenou veličinu, která se postupně a neustále vyvíjí, jako například teplota. Na druhou stranu série událostí vyjadřují pouze diskrétní změny stavu, kdy není zapotřebí několikrát ukládat stejnou hodnotu, ale jsou důležité pouze její změny.

Tato diplomová práce se specializuje na jakékoli senzory, jejiž hodnoty je možné prakticky reprezentovat číselně, nebo za pomocí textových řetězců obou typů časových sérií. Ostatní senzory vyžadují radikálně odlišné přístupy tak, jak je tomu například u již zmíněných kamerových systémů.

2.2 Existující cloudová řešení

Chytrá zařízení je často velmi praktické spojovat do větších sítí. Jednotlivé senzory totiž nemusí poskytovat všechny potřebné informace a centralizace získaných informací sníží požadavky na koncová zařízení. Za tímto účelem je nutné zvolit strategii použitou ke vzájemnému propojení zařízení, kdy je možné vytvořit aplikaci na již existujícím cloudovém řešení nebo na vlastním zařízení. Cloudová řešení nám přináší možnost vytvořit aplikaci, která bude vysoce dostupná, bude možné ji v případě nutnosti jednoduše škálovat bez potřeby vlastního zařízení, které by bylo nutné spravovat.

Samotná cloudová řešení v dnešní době nabízí velké množství společností a často se jednotlivé nabídky liší pouze dodatečnými službami, které jsou nabízeny. Někteří poskytovatelé například nabízí vlastní databázové systémy nebo analytické nástroje, které mohou být při vývoji nebo vyhodnocování finálního řešení velmi nápomocné. Mezi 3 nejpoužívanější [4] cloudové poskytovatele patří Amazon Web Servicies[2] Microsoft Azure[3] a Google Cloud[4].

Ať už zvolíme jakékoli řešení, musíme se vždy nějakým způsobem vypořádat se senzory, které využívají ke komunikaci technologie, které nejsou přímo připojené k internetu, jako například Bluetooth a se zařízeními, které vytváří vlastní *ad-hoc* sítě. Častým řešením obou problémů je vytvoření takzvaných bran [45] (častěji nazýváno anglickým názvem *gateway*), které slouží k agregaci dat z těchto zařízení a jejich zpřístupnění do tradičních interneto-

[1] Popis časových sérií podle společnosti infuxdata: https://www.influxdata.com/what-is-time-series-data/

[2] Amazon Web Servicies: https://aws.amazon.com/

[3] Microsoft Azure: https://azure.microsoft.com/

[4] Google Cloud: https://cloud.google.com/

vých sítí. Za tímto účelem musí být daná brána schopná komunikovat za použitím velkého množství technologií a protokolů a být dostatečně výkonná na případnou konverzi mezi jednotlivými typy protokolů.

V dnešní době už existují i cloudová řešení nebo ekosystémy, které spolu přináší i jiné, netradiční výhody. Představitelem tohoto ekosystému může být The Things Netwrok[5], který zjednodušuje připojení a předzpracování zařízení, která komunikují přes LoRaWAN sítě.

2.3 Technologie přenosů dat ze senzorů

Pro přenos samotných dat ať už od senzorů k jednotlivým bránám, nebo od bran k serveru, který bude data ukládat jsou často využívány 2 přístupy. První z nich je využití tradiční protokolu HTTP, který přináší velkou volnost ve způsobu, jakým je možné data přenášet a dobře známé způsoby zabezpečení v podobě TLS. Druhou možností je protokol MQTT, který poskytuje možnost přenášet data přes síť s menšími požadavky na šířku přenosového pásma a výkon zařízení [44] než HTTP.

Protokol MQTT je konceptuálně rozdílný od HTTP. Zatímco HTTP pracuje na principu požadavek – odpověď, MQTT pracuje na principu publikování a odebírání. Zařízení můžeme v tomto principu rozdělit do dvou následující typů podle jejich zodpovědností:

- Klient – koncová zařízení, která mohou publikovat nebo odebírat informace. Za tímto účelem se klienti připojují k jednotlivým agentům.
- Agent (*Broker*) – zařízení zodpovědná za přijímání všech publikujících klientů, filtrování přijatých zpráv a jejich odesílání klientům, kteří tyto zprávy odebírají.

Další důležitou vlastností protokolu je i poskytnutí vestavěného mechanismu pro rezervaci a řízení *QoS*. Protokol poskytuje 3 úrovně [38], které se stupňují:

- *QoS0* – zpráva je zaslána maximálně jednou a její doručení není nijak potvrzeno.
- *QoS1* – zpráva je zaslána alespoň jednou a je možné, že bude doručena vícekrát.
- *QoS2* – zpráva je zaslána přesně jednou, doručení je zaručeno pomocí „čtyřcestného potřesení rukou“ častěji nazýváno anglickým názvem *4-way handshake*.

Z pohledu bezpečnosti je stejně jako v případě HTTP možné použít TLS nebo symetrické šifrování, ale obě tato řešení představují další značný nárůst požadavků [13] na samotná zařízení. Další výkonově přijatelnější možností je mapování hodnoty na HMAC haš (*Value-to-HMAC Mapping*), kdy se naměřená hodnota před odesláním zakóduje klíčovanou hašovací funkcí. Jelikož jsou hašovací funkce jednocestné, tedy není možné z vytvořeného haše získat původní hodnota, musí příjemce vytvořit tabulku, která bude obsahovat jednotlivé haše a k nim odpovídající hodnotu.

Z pohledu útočníka je útok velmi složitý, protože by pro něj bylo nutné vytvořit celou tabulku a to bez znalosti klíče. Právě tento klíč je ale zároveň nevýhodou, jelikož musí být přítomen na každém zařízení a pokud možno unikátní pro každé zařízení. I s tímto omezením ale metoda představuje možnost, která má nejmenší dopad na výkon a zachovává důvěrnost i integritu zaslaných zpráv.

[5] Domovská stránka https://www.thethingsnetwork.org/

Kapitola 3

Databázové systémy

Data ze senzorů je po jejich získání nutné vhodně uložit. Kvůli velkému množství dat je nutné vybrat vhodný typ úložiště, který dokáže data nejen optimálně ukládat, ale také poskytuje možnost provádět operace nad nimi. Tyto operace nejsou pouze jednoduché výběry, jak tomu často je u tradičních dat. Data ze senzorů mohou být například zašumněná nesprávnými hodnotami, které sensor produkuje při spuštění, mohou obsahovat výpadky v momentech, kdy vypadlo lokální internetové připojení nebo mohou být pro použití uživatelem velmi detailní. Výběr správného systému má velmi znatelný dopad i na výkon a responzivitu výsledné aplikace a nesprávný výběr může vést k aplikaci, která bude v praxi nepoužitelná.

Příkladem tohoto dotazu může být například zjištění maximální teploty každého dne za minulé léto. Uživatele nezajímá výběr všech dat za dané období, data si sám dohledávat nechce data chce již vybraná na základě jeho kritérií a to pokud možno v co nejmenším čase. K ukládání velkého množství dat, nad kterými je nutné provádět operace, se využívají databázové systémy, které je možné rozdělit na základě toho, jak vnímají data v nich uložená.

3.1 Relační databázové systémy

Relační databáze, často také SQL databáze jsou nejčastější a v dnešní době nejpoužívanější[1] typ databázového systému. Databází tohoto typu existuje velké množství jako například PostgreSQL, MySQL nebo SQLite. Tyto databáze vnímají data na základě relační algebry [10], pomocí které jsou definovány nám bližší pojmy tabulek, řádků, relací a dalších. Pro ovládání systému systémy poskytují standardizovaný jazyk SQL, ke kterému většina databázových systému přidává vlastní nestandardní rozšíření. U dat uložených v relačních databázích je nutné dbát na správné využití relačního algebry s použitím takvaných normálních forem [28].

Data se v relačních databázích ukládají po jednotlivých řádcích, což umožňuje rychlý přístup k jednotlivým záznamům. Další výhodou relačních databází je to, že zachovávají vlastnosti ACID [20], tedy poskytují atomicitu transakcí, konzistenci dat v databázi, izolaci jednotlivých transakcí a trvanlivost změn. Za jistou výhodu se dá považovat standardizovaný jazyk SQL, díky kterému je teoreticky možné vyměnit jeden databázový systém za

[1] Statistiky použití databázových systémů: https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/

jiný. V praxi je tato výměna složitější kvůli nestandardním rozšířením, které v některých případech poskytují výhody, které není jednoduché ignorovat.

Jazyk SQL ale neposkytuje žádnou optimalizaci pro ukládání dat z IoT senzorů a použití těchto databází zavádí několik problémů. Prvním z těchto problémů je fakt, že kvůli použití relační algebry je nutné zachovat datový typ senzoru stejný. Tento typ ale nemusí být správným řešením pro všechny typy použitých senzorů. Výstupem senzorů jsou často desetinná čísla, ale v některých případech může být typem hodnoty řetězec. Například senzor, který udává stav dveří může nabývat pouze hodnot "Otevřeno" a "Zavřeno". Toto při použití SQL databází znamená zavedení speciální tabulky pro tento typ senzoru. Za druhou nevýhodu je možné považovat to, že relační databáze delegují operace nad určitým typem dat na aplikační vrstvu. Příkladem operace s tímto problémem může být zjištění, zda-li se zadaná GPS pozice nachází ve vymezeném prostoru. Poslední nevýhodou je fakt, že kvůli zachování vlastností ACID je velmi složité škálovat tyto databáze do šířky (tedy přidáním více strojů), nicméně možné a zřídka prováděné [36].

3.2 NoSQL databázové systémy

NoSQL databáze jsou databáze, které nejsou založeny na relačním kalkulu popsaném výše, ale na jiných konceptech. Většina moderních NoSQL databází se snaží zjednodušit problém škálování do šířky rozdělením dat na různá fyzická zařízení (anglicky nazýváno *sharding*) [17]. Kvůli tomuto rozdělení není možné zajistit vlastnosti ACID popsané výše. Všechny databáze se tedy opírají o takzvaný CAP teorém [18], který definuje tyto 3 následující požadavky na distribuovaný systém:

- Konzistence *(Consistency)* – systém by měl při každém požadavku poskytovat správná data.
- Dostupnost *(Availability)* – systém musí dříve nebo později odpovědět na všechny požadavky.
- Odolnost k porušení *(Partition tolerance)* – systém je možné rozdělit do více skupin, které spolu nemusí nutně komunikovat.

V praxi je bohužel možné splnit vždy pouze dva ze tří požadavků. Všechny NoSQL databáze popsané níže splňují požadavky dostupnosti a odolnosti k porušení a obětují konzistenci a řadí se mezi systémy takzvané BASE [7] systémy *(Basically Available Soft state with Eventual consistency)* v překladu tedy *Prakticky vždy dostupné eventuálně konzistentní systémy bez záruky konzistence*. Tyto systémy tedy nezaručují, že uživatel dostane na požadavek aktuální odpověď, protože se její změna nemusela propagovat napříč všemi stroji, ale zaručuje, že k této změně eventuálně dojde. Mnohé NoSQL databáze povolují kvůli této vlastnosti sdílet určitá data mezi více stroji, čímž vytváří redundanci, ale zvyšují odolnost proti výpadkům systému. Tento přístup se také často nazývá *replikace*, nebo vytváření tzv. *Replica sets*.

Existuje velké množství typů těchto databází jako například databáze s širokými sloupci, dokumentové databáze, databáze klíč hodnota a mnohé další. Kvůli tomu, že tyto systémy často nepodporují relační kalkul, často neposkytují ani standardizovaný jazyk SQL, nebo poskytují jen jeho omezenou podporu. Ze stejného důvodu není nad těmito systémy často možné ani provádět operace, které jsou při použití relačních systémů velmi časté. A to například operace spojení, kterou NoSQL databáze vůbec nepodporují a nebo je jich alternativa velmi pomalá.

Na druhou stranu tyto systémy poskytují velkou volnost při jejich použití. Některé systémy například vůbec nevyžadují schéma databáze, nebo je jej možné kdykoli v průběhu používání aplikace změnit, aniž by bylo nutné provádět změny na již zapsaných záznamech nebo výrazně snížit rychlost operací nad daty. Některé databázové systémy poskytují jen velmi zjednodušený dotazovací jazyk a jakékoli složitější operace delegují na aplikační vrstvu, kde je možné je provést pomocí specializovaných knihoven jako je například Apache Spark [32] nebo poskytují vlastní knihovny, které umožňují provádět agregace nebo i libovolné operace a to většinou dokonce distribuovaným způsobem často založeným na operacích mapování a redukování (MapReduce) [12].

3.2.1 Amazon DynamoDB

DynamoDB je příkladem NoSQL databázového systému typu klíč-hodnota a byla vyvinuta firmou Amazon, díky čemuž je možné ji velmi jednoduše začít využívat na cloudovém řešení AWS, do kterého je integrována. Databázový systém je možné spustit i na vlastním zařízení, nebo u jiného poskytovatele, ale mezi verzí, která je dostupná na AWS cloud a verzí, která je spustitelná kdekoli jsou velké rozdíly a to například ve způsobu vyhodnocování dotazů, které je v stáhnutelné verzi sekvenční[2]. Databázový systém je plně proprietární. DynamoDB neobsahuje prakticky žádnou podporu pro agregace a velmi omezené možnosti pro složitější výběry, zato ale poskytuje možnost data velmi rychle zapisovat a také číst a to hlavně v případě, že se k položkám přistupuje za pomocí primárního klíče nebo indexovaného sloupce. K vytváření dotazů a celkového ovládání systému je nutné použít oficiální driver, přes který se dotazy posílají ve formátu velmi podobném formátu JSON.

DynamoDB požaduje základní definici tabulky, kdy je nutné při vytváření zadat sloupce, které budou použity pro primární klíče tabulky i s jejich typy. Zbylé položky nejsou závislé na tomto schématu a je možné je přidávat nebo nevyužívat libovolně a to vložením dokumentu, který obsahuje nebo neobsahuje klíče těchto položek.

Databáze neposkytuje žádnou podporu nebo optimalizaci pro ukládání datových sérii a hlavně také neobsahuje žádnou podporu složitějších dotazů, kvůli tomu není i přes možnost velmi rychle zapisovat příchozí data dobrou volbou pro ukládání časových sérii. Za určitou nevýhodu se dá považovat i přílišná integrace do ekosystému AWS, kvůli kterému je i při použití na lokálním systému nutné zadávat informace, které nejsou k lokálnímu běhu zapotřebí, jako například způsob, jakým bude placeno za použití databázového systému, region a další.

3.2.2 MongoDB

MongoDB je dokumentová NoSQL databáze, která je vyvíjena stejnojmennou společností jako otevřený software. V poslední době ale výrobce změnil licenci z GNU GPLv3 na vlastní licenci SSPL, která je zatím, až na určité změny, s původní licencí srovnatelná. Tato změna velmi zneklidnila komunitu uživatelů tohoto systému[3]. I nadále ale zůstává MongoDB jako častá první volba při využití NoSQL databázového systému. Podobně jako DynamoDB je i MongoDB možné provozovat robustním cloudovým řešením jako například MongoDB Atlas přímo od výrobce tohoto databázového systému.

[2]Rozdíly verzí Amazon DynamoDB: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBLocal.UsageNotes.html

[3]Reakce komunity ke změně licence: https://www.scylladb.com/2018/10/22/the-dark-side-of-mongodbs-new-license/

MongoDB nevyžaduje u vstupních dat prakticky žádnou stálou strukturu. Ke každému vstupnímu dokumentu automaticky přidává atribut `_id`, který využívají jako primární klíč. Díky předchozím vlastnostem umožňuje ukládat jakákoli data ve formátu JSON, která jsou interně reprezentována ve formátu BSON.

Oba formáty jsou si velmi podobné, formát BSON se dá považovat pouze za rozšíření formátu JSON, od kterého přebírá syntaxi. Formát BSON také přidává podporu pro nové datové typy jako například pro datový typ Date, který umožňuje reprezentovat datum a čas. Největším rozdílem a výhodou je finální reprezentace obou formátů. Zatímco JSON má pouze textovou reprezentaci, BSON reprezentuje data binárně, díky čemuž umožňuje data reprezentovat efektivněji, ale ne čitelně při otevření standardním textovým editorem.

Na rozdíl od předchozích systémů je v tomto databázovém systému možné využít optimalizaci pro časové série, která přidává určitě požadavky na vkládána data. Tato možnost byla do databázového systému přidána ve verzi 5, která vyšla v červenci 2021[4], takže se jedná o relativně novou vlastnost a dá se počítat se zásadními změnami. Vstupní soubor poté musí obsahovat položku s datem, které se využije jako primární klíč vkládaných záznamů a musí být vybrána takzvaná granularita času, která udává jak přesně je zapotřebí čas ukládat. Na druhou stranu tento výběr nijak neomezuje možnost ukládání dat, která díky tomu není nutná nijak předzpracovat. Další výhodou je také poskytnutí velmi silného agregačního systému, který dovoluje nejen využívat předdefinované operace ale také vytvoření operacích vlastních. Operace musí být založeny na principu map-reduce, díky čemuž je možné v případě distribuce systému mezi více strojů, vykonávat distribuovaně přes jednotlivé uzly. MongoDB také dovoluje využít kompresi přes celou tabulku. V základním nastavením je použita komprese *Snappy*, jejíž použití nemá v dnešní době znatelný negativní dopad na výkon systému a umožňuje efektivnější uložení.

Nevýhodou systému pro uložení časových sérií IoT je bohužel to, že kvůli povolení volného schématu vstupních dat není možné využít speciálních kompresí, jako například delta komprese popsané v následující části. I přes to, že poskytované komprese dokáží snížit výslednou velikost dat, nevyužívají za tímto účelem předem známé struktury časových sérií.

3.2.3 InfluxDB

Databázový systém InfluxDB je posledním systémem popsaným v této části. Jedná se o specializovaný NoSQL systém, který je optimalizován právě na ukládání a zpracování časových sérií. Tento systém je plně vyvíjen jako software s otevřeným zdrojovým kódem komunitou a společností InfluxData pod licencí MIT. Díky nejen kvůli tomuto faktu je tento systém velmi oblíbený a je jej možné provozovat prakticky u jakéhokoliv cloudového poskytovatele nebo velmi jednoduše na vlastním stoji.

Oproti předchozím dvěma NoSQL systémům neposkytuje takovou volnost při ukládání dat. Data ukládaná do systému musí vždy obsahovat časovou značku a název měření. Tato kombinace je poté použita jako primární klíč. K samotnému uložení je poté potřeba zadat název měřené veličiny a samotnou hodnotu. Tyto data se v terminologii používané v InfluxDB jmenují bod. Ke každému bodu je ještě možné přidat takzvané štítky, v angličtině „tag“. Tyto štítky mohou být použity pro ukládání metadat o daném bodu a je možné na základě nich efektivně vyhledávat. I přes tyto limitace poskytuje větší volnost než zmíněné relační databázové systémy. Oproti těmto systémům nevyžaduje InfluxDB žádnou inicializaci měření, polí ani štítků v databázi. Tyto informace stačí do databáze pouze zaspat

[4]Poznámky k verzi 5.0 databázového systému MongoDB: https://www.mongodb.com/docs/manual/release-notes/5.0/#time-series-collections

a systém se sám postará o jejich efektivní uložení. Podobnou volnost systém poskytuje i v typu samotných hodnot. Zatímco do klasických relačních systémů je nutné zapsat pouze předem definovaný datový typ, InfluxDB umožňuje zapsat více datových typů, například celé číslo, číslo s plovoucí desetinnou čárkou ale i řetězec.

InfluxDB stejně jako ostatní databázové systémy využívá k ukládání dat takzvanou rozdílovou (*delta*) kompresi (anglicky *delta compression*). Databázové systémy využívají tuto kompresi počítají s tím, že změny hodnot nejsou skokové ale mění se postupně s malým rozdílem mezi nimi. Díky tomuto předpokladu není nutné ani efektivní ukládat celé hodnoty, ale pouze rozdíl mezi předchozí a aktuální hodnotou. Takto vypočtená hodnota je poté nazývána rozdíl, anglicky *delta*. Pokud při zápisu hodnoty je předchozí hodnota již deltou, systém musí dopočítat předchozí hodnotu a spočítat rozdíl, který se poté uloží. Tato hodnota se poté nazývá jako rozdíl rozdílu (*delta of deltas*). Stejný způsob reprezentace je možné uložit i pro časovou značku. Díky tomuto způsobu uložení je možné velmi ušetřit na potřebné zapsané velikosti.

Stejně jako systém MongoDB poskytuje tento databázový systém efektivní způsob dotazování dat a to za pomocí vlastního jazyka Flux, který je také založený na zřetězeném zpracování dat. Narozdíl od obou předem zmíněných NoSQL databázových systémů umožňuje InfluxDB využít i podmnožinu standardního jazyka SQL a je tedy přívětivý i k vývojářům, který jsou zvyklí používat relační databázové systémy. Za stejně zajímavou vlastnost se dá považovat i poskytováná možnost integrace vybraných relačních databází, díky které je možné se z jazyka Flux připojit do relačního databázového systému, vyhodnotit nad ním určitý dotaz a výsledek tohoto dotazu, poté spojit s daty, která jsou uložena v databázovém systému InfluxDB.

Za nevýhodu se dá považovat restrikce vkládaných dat, ale v případě ukládání dat z IoT zařízení tento problém nenastává. Další vlastností, kterou se dá považovat za nevýhodu, je neexistence klasického řadiče, který často využívají relační databáze, místo kterého InfluxDB poskytuje RESTové API a možnost zapisovat pomocí nástroje Telegraf[5].

3.3 NewSQL databázové systémy

Databázové systémy NewSQL jsou nové relační systémy, které se snaží poskytovat některé výhody databázových systémů typu NoSQL, jako je například možnost systém jednoduše škálovat do šírky, bez porušení vlastností ACID. Dalším zásadním rozdílem oproti NoSQL databázovým systémům je fakt, že stejně jako relační databázové systémy poskytují standardizovaný jazyk SQL, nebo alespoň jeho významnou podmnožinu, místo jazyku nestandardního nebo proprietárního řadiče. Stejně jako NoSQL databázové systémy tedy musí NewSQL databázové systémy poskytovat možnost škálování do šířky a tedy se nějakým způsobem vypořádat s již zmíněným CAP teorémem. Tyto systémy se k němu staví velmi jednoduše a to tak, že obětují dostupnost systému a zachovají konzistenci a odolnost proti rozdělení.

Z praktického pohledu poté poskytují speciální způsoby aktualizace těchto redundantních záznamů, například NuoDB, které využívá tazkvaných atomů, které data nejen obsahují, ale také se starají o udržení konzistence všech kopií. Díky preferování konzistence se na první pohled podobají spíše relačním databázovým systémům než systémům NoSQL, které preferují dostupnost. Další významnou podobností je i fakt, že se stejně jako relační databáze zaměřují na zpracování relačních dat a stávají se tedy možnou alternativou k nim.

[5]Telegraf, agent pro kolekci dat: https://www.influxdata.com/time-series-platform/telegraf/

Z tohoto důvodu nejsou přizpůsobeny pro ukládání časových sérií a kvůli tomu zde není obsažen podrobný rozbor těchto systémů. I přes to, že nemají optimalizace pro časové série, je možné do nich tyto data uložit. V takovém případě je možné vytvořit velkou množinu požadovaných agregačních funkcí v jazyce SQL nebo využít již zmiňovaný Apache Spark. Srovnání jednotlivých databází tohoto typu je možné najít v mnoha publikacích, například [26, 27].

3.4 Databázové indexy

Záznamy v databázi je možné uložit bez jakéhokoli indexu. Pokud by ale žádný index nebyl vytvořen, bylo by nutné při vyhledávání požadované položky projít celou databázi a hledat správný prvek. V relačních databázích by to díky jejich zarovnání znamenalo procházet jednotlivé řádky a porovnávat hledanou hodnotu s hodnotou v řádku. Díky definici tabulky, která je v relačních databázích nutná, je možné vypočítat adresu diskového bloku na kterém je informace uložena. V NoSQL databázích je tento problém ještě významnější a zejména kvůli možnosti systém distribuovat mezi více strojů a možnosti zapsat velmi různorodá data, která jsou na rozdíl od relačních databází nezarovnaná. Toto vyhledávání by bylo příliš neefektivní, a proto se nad sloupci, podle kterých se bude často vyhledávat, vytváří takzvaný index, který slouží ke zrychlení přístupu k datům. Tyto indexy mohou být několika typů, každý index má svoje vlastnosti. Nejčastější z nich podle [39] jsou popsány níže.

3.4.1 Stromové indexy

Stromový index je v současné době nejčastější [25] způsob indexace v moderních databázích. Stromových algoritmů je velké množství, ale často jsou používané algoritmy založeny na binárním stromu nebo B+ stromech. Tyto algoritmy poskytují dobrý výkon při vyhledávání, kde B+ strom má při vyhledávání v nejhorším případě $O(\log n + \log L)$, jejich největší nevýhodou je potřeba vyvažování stromu, které může být hlavně nad stromy velké hloubky příliš náročné.

Další vlastností stromových indexů je možnost vyhledávat na základě rozsahu hodnot, tedy provádět takzvaný range scan, díky kterému je možné efektivně získávat například hodnoty, které spadají do zadaného časového období. Tato vlastnost není samozřejmá pro všechny typy indexů.

Hlavní nevýhodou tradičních stromových indexů představuje nemožnost využít tyto stromové algoritmy využít při distribuování systému, kde by vyvážení stromu mohlo znamenat přenos záznamů napříč použitými zařízeními.

3.4.2 Index pomocí hašování

Hašování se nabízí jako další z možných přístupů při vytváření databázových indexů. Tento přístup je z praktického hlediska založen na stejném principu jako hašovací tabulky. Za pomoci vybrané hašovací funkce je ze vstupního klíče vytvořen index, který je využit při ukládání dané hodnoty. Více klíčů může nabývat stejné hodnoty indexu, tento jev se nazývá hašovací kolize a v případech, kdy tato kolize nastane, je nutné se s ní nějakým způsobem vypořádat. Častým řešením je ukládání dat do seznamů. Z pohledu vyhledávání i vkládání tedy může dojít ke dvěma situacím. První – index vygenerovaný na základě klíče není kolizní a operace mají konstantní časovou složitost nebo druhé – index je kolizní a je nutné prohledat seznam uložených dat, což znamená v nejhorším případě lineární složitost.

Další výhodou je možnost toto indexování použít i v distribuovaném systému, kde se často používá název konzistentní hashování. Tento typ hashování se typicky používá pro indexy primárních klíčů a to díky jednoduché modifikaci, kde výsledkem hashovací funkce je index zařízení, na kterém se daná data nachází.

Oproti stromovým indexům hašované indexy nedovolují provést vyhledávání na základě rozsahu dat. Důvod je jednoduchý, vzájemná pozice dvou klíčů není zachována po vytvoření jejich indexů.

3.4.3 Indexy pomocí LSM Stromů

Log-Structured merge-tree je stromová struktura, která se na rozdíl od předchozích skládá z dvou nebo více stromových komponent. První komponenta je často uložena v paměti a slouží jako cache, druhá a všechny další komponenty bývají uloženy na disku. Všechny nové záznamy jsou ukládány do první komponenty a jsou díky stromové struktuře seřazeny. Až tato komponenta dosáhne maximální velikosti je z první komponenty vyjmut seřazený segment a vložen do druhé komponenty algoritmem, který je podobný algoritmu merge-sort.

Díky tomuto přístupu je možné do LSM stromu rychle zapisovat a zároveň nemá hlavní nevýhodu tradičních stromů, tedy vyvažování. I při použití LSM stromů je vyvažování potřeba, ale díky použití cachovací komponenty není zapotřebí často provádět vyvažování nad stromem velké výšky. Narozdíl od hašovaných indexů LSM stromy zachovávají možnost vyhledávání na základě rozsahu.

Tento index je často používán v NoSQL databázových systémech, ale podporu začínají zavádět i některé relační databázové systémy. Z uvedených systému používá tuto nebo odvozenou metodu indexování MongoDB[6], InfluxDB[7] a SQLite[8].

[6] LSM v WiredTiger, který používá MongoDB: http://source.wiredtiger.com/3.0.0/index.html

[7] Více o používaných strukturách v InfluxDB: https://docs.influxdata.com/influxdb/v2.2/reference/internals/storage-engine/

[8] LSM v SQLite https://www.sqlite.org/src4/doc/trunk/www/lsmusr.wiki

Kapitola 4

Dashboard

Po přenesení a vhodném uložení dat vzniká problém jejich efektivního zobrazení uživateli. Nejpoužívanějším nástrojem v dnešním době jsou takzvané informační dashboardy, které představují typicky jednostránkový pohled, který uživatelům zobrazuje vše, co potřebují ke své práci. Pokud je tento pohled správně navržen, poskytuje uživatelům možnost monitorovat daný systém na první pohled.

Toto očekávání klade na dashboardy nároky. Kvůli velké rozmanitosti a množství dat, které je zapotřebí zobrazit na malém prostoru, je velmi složité tyto nároky splnit a většina vytvořených dashboardů tyto vlastnosti nesplňuje. K tvorbě odpovídajícího pohledu je tedy nutné pochopit nejen vlastnosti zobrazovaných dat, ale také to, jak budou využity. Za tímto účelem je nutné využít vlastnosti samotného lidského vnímání a důkladný návrh použitých komponent.

Požadavky na správně navržený dashboard definoval Stephen Few [15] následovně:

- Vysokoúrovňový pohled – nedoporučuje se zobrazovat samotná data, ale spíše jejich agregace, které lépe vyjadřují aktuální stav. Na první pohled je tedy nutné zjistit co se děje a ne proč se právě toto děje. Dashboard může poskytovat možnost stav blíže analyzovat ale není to nutné.
- Konzistentní, jednoduché a intuitivní zobrazení – je zapotřebí vyžít komponentů které dokáží data vhodně zobrazit aniž by na obrazovce zabíraly mnoho místa. Je také nutné zaměřit se na výběr samotných komponent, které musí primárně sloužit k zobrazení i na úkor jejich vzhledu.
- Přizpůsobené – informace musí být zobrazeny tak, aby je pochopili cíloví uživatelé.

Kromě těchto kritérií je také nutné zamyslet se i nad dalšími vlastnostmi a to například, zda bude dashboard zobrazovat surová data, nebo jiné důležité odhozené metriky, které lépe popisují aktuální stav sledovaného systému.

4.1 Kategorie dashboardů

Stephen Few v jeho knize [16] prezentuje dělení dashboardů na základě role pro kterou je daný dashboard vytvořen jako nejpraktičtější způsob. Toto rozdělení zahrnuje 3 typy dashboardů a to: strategické, analytické a operační.

4.1.1 Strategické

Tyto dashboardy zobrazují informace, pomáhající uživatelům zhodnocovat stav společnosti a zobrazovat příležitosti, které se dané společnosti nabízí. K těmto účelům se většinou používají agregované a vysokoúrovňové pohledy a odhady vývoje v příštích měsících. Kromě těchto metrik často zobrazují i jednoduché srovnání očekávaných a reálných hodnot, například použití slov průměrné, dobré, špatné.

Pro tyto dashboardy jsou vhodné jednoduché komponenty pro zobrazení jednotlivých metrik. Data v dasboardech díky použití vysokoúrovňových dat není nutné zobrazovat data z reálného času, ale spíše agregace na základě časového údaje jako například prodeje v posledních 7 dnech.

4.1.2 Analytické

Analytické dashboardy ke svojí funkci často potřebují větší množství dat než strategické dashboardy a odpovídající nástroje, jako například možnosti porovnávat data mezi sebou nebo vyhledávat konkrétní historická data. Stejně jako předchozí typ i analytické dashboardy benefitují z agregovaných dat na základě časového údaje a to ze stejného důvodu.

Analytické dashboardy vyžadují podporu interakce s daty, jako například snížení úrovně aktuální agregace, aby bylo možné zjistit z jakého důvodu došlo k aktuálnímu stavu. Díky této vlastnosti má analytik, který využívá dashboard, možnost zobrazit dlouhé časové období nebo velké množství dat, které bude reprezentováno agregovanými daty a v případě potřeby přejít k přehledu, který bude více přesný. Příkladem může být situace, kdy se analytikovi zobrazí informace o určité třídě zařízení a v případě potřeby může přejít k stavu jednotlivých zařízení v této třídě.

4.1.3 Operační

Operační dashboardy jsou velmi rozdílné od předchozích dvou typů. Jelikož jsou užívané k monitorování provozu musí poskytovat možnost monitorovat i dynamická data, které je často nutné zobrazit ihned po jejich změně. Tyto změny mohou často vyžadovat okamžitou reakci od uživatele.

Stejně jako u strategických dashboardů je nutné data zobrazovat velmi jednoduše a to z následujícího důvodu. Pro uživatele je stav nouze velmi stresující a může od něj vyžadovat okamžitou reakci. Pokud by data a závěry z nich nebyly zobrazeny jednoduše mohl by uživatel zareagovat špatně díky jejich interpretaci. Narozdíl od předchozích typů je také nutné zvýraznit hodnoty, které nejsou v akceptovatelném rozmezí, takovým způsobem, aby uživatel hned zareagoval. Zobrazovaná data musí být konkrétní, tedy bez agreagací, nebo systém musí nabízet možnost zobrazit tato data na základě interakce uživatele se systémem.

4.2 Efektivní zobrazení

Kvalita jakéhokoliv dasboardu velmi závisí na efektivitě reprezentace vizualizačních komponent a dat v nich obsažené. Dashboardy reprezentují své komponenty zpravidla vizuálně, a proto je nutné se zaměřit na vlastnosti lidského vizuálního vnímání. Data jsou v dashboardech reprezentována hodnotami, které je nutné doplnit o jejich význam a účel. Například zobrazená hodnota 5 000 nemá pro uživatele žádnou vypovídající hodnotu. Na to, aby byla tato hodnota uživateli užitečná, musíme přidat popis „průměrná denní návštěvnost

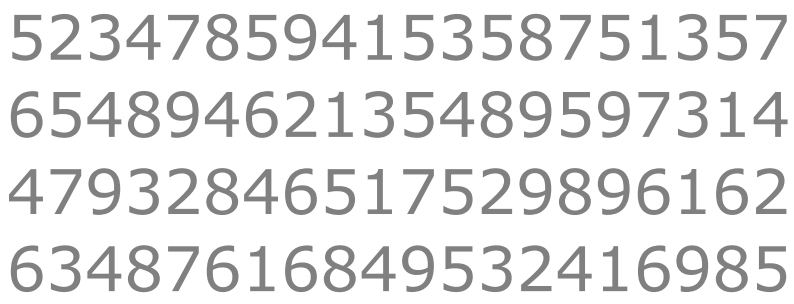
stránky“. Pokud je našim cílem rychlé pochopení uživatelem, musíme zajistit, aby byl proces získání informací z hodnot byl co nejrychlejší. Za tímto účelem je potřeba brát ohled při navrhování na způsob, jakým uživatel vnímá a zpracovává informace jemu prezentované.

Lidské vizuální vnímání [5] je možné chápat jako sérii po sobě následujících fází. V první fázi zpracování dochází k interpretaci viditelného světla pomocí očí. Paprsky viditelného světla prochází rohovkou, která je pomáhá směřovat dál k zornici a duhovce. Obě tyto části slouží ke kontrole množství světla, které prochází okem k čočce, která mění své vlastnosti na základě toho, na jaký objekt chceme zaostřit. Poslední částí oka je sítnice, na kterou se promítá výsledný obraz a je zpracován pomocí tyčinek a čípků. Takto zpracované paprsky již přechází do mozku, který z nich sestaví výsledný obraz.

Tento předzpracovaný obraz je ještě nutné interpretovat a udržet informace v něm zobrazené. Informace jsou nejdříve zpracovávány lidským vnímáním a uloženy v paměti. Lidské vnímání probíhá ve dvou fázích a to podvědomé a vědomé zpracování. Podvědomé zpracování je z těchto dvou rychlejší a dovoluje nám vnímat vjemy, které jsou nám známé, nebo jiným způsobem výrazné. Pozorné zpracování je oproti němu pomalejší a vyžaduje vyšší koncentraci. Oba případy jsou ovlivněny tím, jak funguje lidská paměť. Tu je možné rozdělit do 3 fází podle [3] a to:

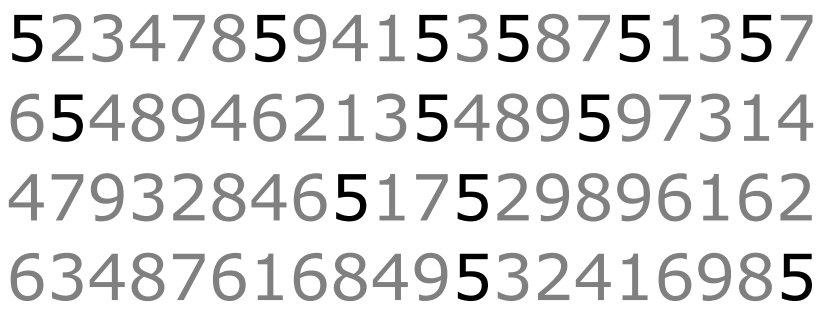
- senzorická paměť – slouží k uchovávání zrovna vnímaných informací, u kterých se rozhoduje, jestli jsou důležité nebo ne,
- krátkodobá paměť – slouží k operacím nad informacemi. Tyto informace mohou být právě přijaté ze senzorické paměti, nebo získané z dlouhodobé paměti,
- dlouhodobá paměť – slouží k dlouhodobému ukládání informací. Pokud je nutné informace znovu zpracovat musí být vyvolány zpět do krátkodobé paměti.

Příklady dopadu obou dvou principů je možné pozorovat na příkladu 4.1. V následujícím obrázku je velmi složité vyhledat všechny číslice 5.



Obrázek 4.1: Příklad efektu intenzity barvy na vnímání, převzato z [16] .

Vyhledávání je velmi náročné kvůli tomu, že číslice od sebe nejsou žádným způsobem odlišené. Z toho důvodu je nutné použít pro vyhledávání číslic pomalejší pozorné vnímaní. V následujícím obrázku 4.2 je vyhledávání daleko jednodušší.



Obrázek 4.2: Příklad efektu intenzity barvy na vnímání, převzato z [16] .

Důvod je jednoduchý, použité barvy jsou kontrastní. Kontrast, na rozdíl od tvaru číslic je jeden z parametrů, který je vnímaný ve fázi předběžného zpracování. Díky tomu je možné analyzovat obsah obrázku daleko rychleji. Tyto koncepty popisuje Colin Ware [43] ve své knize. Dalším principem, který má dopad na lidské vnímání jsou Gestalt principy.

Gestalt principy [8] se zaměřují na to, jak lidský mozek vnímá vzory nebo konfigurace elementů. Původní německé slovo *Gestalt* je do češtiny překládáno jako podoba, tvar nebo struktura, odtud pochází alternativní název tvarová psychologie, který se často používá více v psychologii než v oboru informačních technologií.

Dobrým příkladem, který ukazuje proč jsou tyto principy důležité je takzvaný princip blízkosti, který nám říká, že objekty, které jsou blízko sebe máme tendenci vnímat jako jednu skupinu. Tohoto principu můžeme využít pro vytváření skupin, které budou velmi jednoduché na vnímání i bez použití dalších vizuálních prvků. Tyto principy jsou pro vytváření jakýchkoli vizuálních prvků velmi důležité a proto se často používají jak pro dashboardy tak i jiné vizualizace, jako například animované vizualizace síťových dat [34].

4.3 Typy vizualizací

Pro velké množství typů dat, které je nutné zobrazovat již existují efektivní nástroje, nebo způsoby, které mohou tvorbu dashboardu velmi usnadnit. V typických dashboardech je často nutné zobrazovat kvantitativní data jako například různé teploty, průměrnou cenu zakoupených produktů v dané kategorii a další. Všechny prvky v této kategorii zobrazují 2D data, což reprezentuje problém. Data, která je nutné zobrazit mohou mít více dimenzí. K těmto účelům může být praktické využít speciální vizualizace, jako například ty popsané v [30], nebo za pomocí tabulky. Příkladem těchto dat může být například následující tabulka:

Období	Místo	Kategorie	Počet objednávek
Q1	Praha	elektronika	12
Q1	Brno	elektronika	20
Q2	Praha	nábytek	18
Q2	Brno	nábytek	10

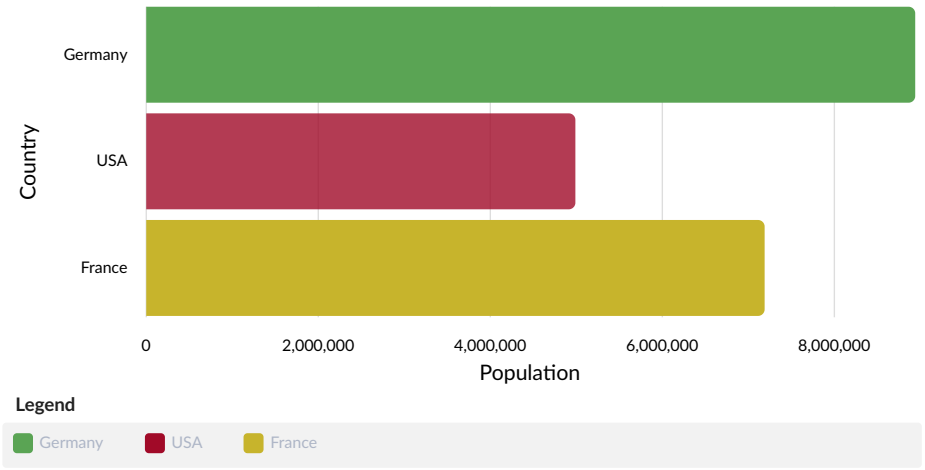
Tabulka 4.1: Příklad multidimenzionálních dat.

K zobrazení těchto dat pomocí tradičních 2D vizualizací je nutné snížit počet dimenzí vstupních dat. Za tímto účelem mohou být provedeny agregace, funkce, které redukují více vstupních hodnot na jednu. Mezi 2D vizualizace patří nejen tradiční grafy sloupcové, spoj-

nicové, odrážkové a takzvané sparklines, ale také další vizualizace, které například využívají mapu.

Sloupcové grafy

Tento typ grafu je vhodný zejména v tom případě, kdy je nutné zobrazit hodnoty měření, které spadají do nějaké kategorie. Hodnoty v tomto grafu jsou reprezentovány sloupci umístěnými v systému souřadnic, kde jedna osa reprezentuje danou kategorii, kde se může jednat například o nominální nebo ordinální hodnotu a druhá vlastní hodnotu této kategorie.



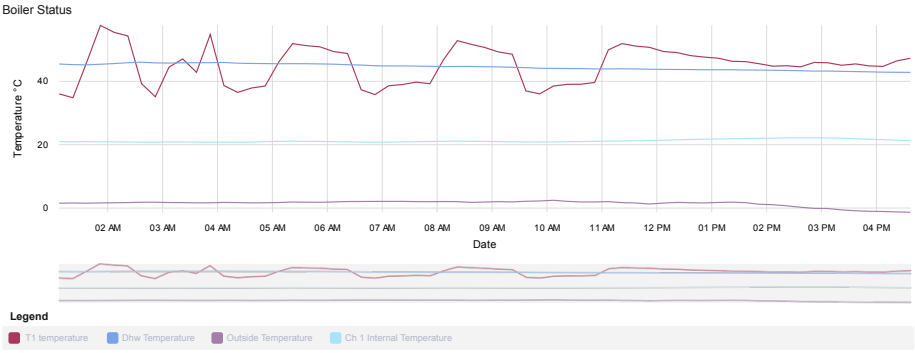
Obrázek 4.3: Příklad sloupcového grafu, převzato z[1]

Podle umístění osy s kategorií se poté jedná o horizontální nebo vertikální sloupcový graf. Díky tomu zdůrazňování jednotlivých hodnot umožňuje tento graf zobrazené hodnoty a je proto častým kandidátem pro zobrazování dat ve strategických dashboardech. Tento graf je praktické použít například při zobrazení počtu obyvatel jako na obrázku 4.3.

Spojnicové grafy

Tento typ grafu je velmi praktický v případech, kdy je nutné podrobně zobrazit veličinu, která se například postupně vyvíjí v čase nebo je takto závislá na jiné veličině. Stejně jako předchozí typ využívají spojnicové grafy souřadnicový systém, na který vynáší spojnice dat.

[1]Ukázky grafů ngx-charts: https://stackblitz.com/edit/swimlane-horizontal-bar-chart

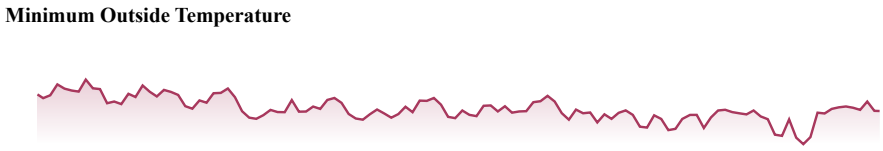


Obrázek 4.4: Příklad spojnicového grafu

Díky tomuto přístupu je možné na spojnicových datech dobře pozorovat trend zobrazované hodnoty. Na první pohled tedy můžeme zjistit, zdali daná hodnota roste, nebo naopak klesá nebo jak stabilní je. Tyto vlastnosti jsou žádoucí v dashboardech provozního typu. Konkrétním případem užití může být zobrazení vývoje teplot, jako například venkovní teploty na obrázku 4.4.

Sparklines

Sparkline je typ grafu, který je velmi podobný spojicovému grafu a je autorem popisován jako „jednoduchá, datově intenzivní grafiku o velikosti slova“ [40]. Na rozdíl od spojnicového grafu sparkline nepoužívá ani jednu osu.



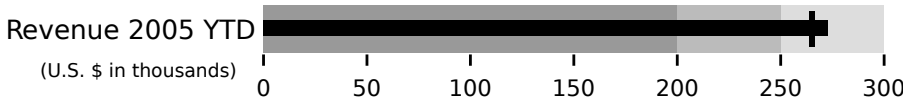
Obrázek 4.5: Příklad sparkline

Tato vlastnost se může na první pohled zdát jako nevýhoda, ale díky tomu, že se grafy typu sparkline používají k zobrazení historických trendů, které jsou jasné z prvního pohledu jsou hlavně díky jejich malé velikosti při tvorbě dashboardů velmi praktické. Jejich využití může být například zobrazení trendu minimální venkovní teploty za poslední měsíc na obrázku 4.5.

Odrážkové grafy

Odrážkový graf, častěji nazývaný původním anglickým názvem *bullet graph*, je speciální typ grafu, který představil Stephen Few [15] za účelem odstranit nevýhody budíkového grafu. Tento a ostatní jemu podobné grafy zobrazují hodnotu pouze jedné veličiny často s porovnáním cílovou nebo předpovídanou hodnotou a nebo kvantitativní škálou. Podle autora budíkové a podobné grafy nevytváří nejjasnější a nejsmysluplnější reprezentaci dat, která

by zabírala co nejmenší množství místa. Jako řešení navrhuje novou vizualizaci, která stejně jako sloupcové grafy využívá sloupců. Jeden z těchto sloupců vždy reprezentuje samotnou hodnotu a zbylé sloupce v pozadí reprezentují kvantitativní rozsahy dané hodnoty.



Obrázek 4.6: Příklad odrážkového grafu, převzato z [15]

Díky tomu může být tato vizualizace vhodná v momentech, kdy je nutné srovnat aktuální hodnotu s odpovídajícím prahem, nebo předchozí hodnotou. Na obrázku 4.6 je vyobrazen případ srovnání příjmů společnosti.

Mapové

Často je v dashboardech nutné zobrazit i data, která vyžadují jinou vizualizaci. Jedním z častých případů je danou vizualizací nějaký typ mapy. Může se jednat například o geografickou mapu zobrazující cestu zvoleného vozidla za zvolené období nebo o abstraktní mapu zařízení připojených k lokální síti. Tyto zobrazení jsou často velmi specifické k jejich použití a není jednoduché navrhnout konkrétní komponenty, které by bylo možné znovu použít jako v předchozích případech.

4.4 Klíčové ukazatele výkonnosti

K vyhodnocení stavu systému nestačí pouze samotná data, ale je potřeba vytvořit speciální odvozené hodnoty. W. Eckerson ve své knize [14] definuje klíčové ukazatele výkonnosti (KPI) jako ukazatele sloužící k vyhodnocení úspěšnosti organizace nebo jednotlivce v provádění operačních, taktických nebo strategických aktivit, které jsou důležité pro nynější i budoucí úspěch organizace. K tomu, aby byly tyto metriky přínosné pro danou společnost musí odpovídat strategii, kterou firma využívá.

Eckerson také dále rozděluje typy KPI na dvě skupiny. První z nich nazývá *leading indicators* a druhý *lagging indicators*. Leading indikátory určují hodnoty aktivit, které mají dopad na budoucí výkon firmy, jako například počet klientů, který by měl zaměstnanec obsloužit každý den. Indikátory spadající do druhé kategorie měří výkon aktivit, které se udály v minulosti. Příkladem těchto indikátorů může být například počet klientů, který by měl zaměstnanec obsloužit každý týden.

V dnešní době není vždy nutné vytvářet nová KPI, ale spíše vybírat z předem připravených kolekcí, jakou je například kniha [31] Bernarda Marra, ve které je prezentováno 75 KPI pro použití manažery. V některých případech ale není výběr tak jednoduchý. Jedním z těchto případů jsou například komunitní KPI pro chytrá města, kdy je nutné vyhovět požadavkům nejen samotné společnosti ale také jednotlivých uživatelů systémů poskytovaných městem s ohledem na jejich soukromí. Touto problematikou se zabývali Drew Hemment a spol., kteří ve své zprávě [23] prezentovali způsob vytvoření komunitních KPI ve městě Manchester v rámci projektu CityVerve. Vytváření KPI je v této zprávě rozděleno do následujících 4 fází:

- Návrh cílů – je zaměřen na objevování a definování cílů a k nim potřebných KPI. Výsledkem jsou vysokoúrovňové KPI ze kterých budou odvozeny KPI pro individuální projekty.
- Iterace procesu – je zaměřena na vylepšení a upřesnění jednotlivých KPI založené na bližším popisu.
- Vytváření aktiv – je zaměřeno na způsoby integrace nových KPI do již existujícího projektu.
- Vyhodnocení – obsahuje hodnocení finálního řešení a pochopení nově získaných zjištění.

Do celého procesu jsou zahrnuti nejen zástupci společnosti, která dodává programové řešení a členové městského zastupitelstva, ale také obyvatelé samotného města.

K zobrazení hodnot KPI se často využívají vizualizace, které byly představeny v předchozí kapitole, ale jen některé z nich jsou efektivní pro dané použití. Efektivitu dané vizualizace je proto nutné vždy zvážit a zhodnotit. Například Yorick Heidema se v jeho diplomové práci [22] zabývá hodnocením vizualizací KPI pro produkční plánování ve výrobě. V některých případech je dokonce nutné poskytovat data jiným způsobem, než za pomocí dashboardů a to například zasláním zprávy, která obsahuje potřebné vizualizace uživateli [23].

4.5 Detekce anomálií

Jak už bylo zmíněno v kapitole 3, data ze senzorů mohou být určitým způsobem zašuměná nebo jiným způsobem nekvalitní. Nekvalitní data mohou obsahovat hodnoty, které jsou mimo měřitelný rozsah zařízení k čemuž často dochází při spuštění zařízení, kdy byla odečtena hodnota senzoru dříve, než došlo k prvnímu měření senzoru.

Data také mohou obsahovat chybějící nebo odlehlé hodnoty. Odlehlé hodnoty mohou být důležitá i pro detekci rychlých a neočekávaných změn v systému, které nemusí naznačovat pouze chybu systému, ale také například jeho netradiční chování.

Kontrolu dat na již zmíněné nedostatky není možné provádět ručně a to z velmi jednoduchého důvodu. Data z IoT zařízení jsou příliš rozsáhlá a přibývají velmi rychle. Proto je nutné se s některými nedostatky automaticky vypořádat pokud možno již při samotném vkládání dat tak, aby nebyla ovlivněna odvozená data. Občasné výpadky je možné doplnit aproximací založenou na předchozí a následující hodnotě a hodnoty mimo měřený rozsah mohou být ignorovány a nebo uloženy do jiné databáze, která je k tomuto účelu vyhrazena.

Mezi anomálie, které se tradičně detekují v časových sériích [29, 35] patří odlehlé hodnoty, takzvané body změny a anomální časové série.

Odlehlé hodnoty

Detekování odlehlých hodnot bývá často považováno jako nejdůležitější typ detekce anomálií. Detekce těchto hodnot se často provádí dvěma způsoby a to buď modelováním tradičního chování této časové série nebo pomocí metod založených na dekompozici.

První způsob k detekci využívá dvou částí a to algoritmu, který slouží k předpovědi a algoritmu, která vypočítá relativní chybu mezi předpovídanou a naměřenou hodnotou. Pokud je poté tato relativní chyba mezi hodnotami větší než uživatelem zadaný práh, je naměřená hodnota označena jako odlehlá.

Tato metoda často produkuje dobré výsledky na sériích, které nevykazují náhlé změny a je možné je namodelovat způsobem, který dobře odráží realitu. Pokud ale časová série obsahuje časté výkyvy bude tato metoda detekovat velké množství falešně pozitivních hodnot.

Dekompoziční metoda poté rozděluje časovou sérii na trend, sezónní data a šum. Právě v šumu jsou poté detekovány odlehlé hodnoty na základě prahu zadaného uživatelem.

Body změny

Body změny odpovídají momentům, kdy se mění chování dané časové série z očekávaného na neočekávané. Na rozdíl od odlehlých hodnot tento bod označuje trvalejší změnu.

Jeden ze způsobů, jak tyto změny detekovat, představují metriky rozdílnosti založené na posuvném okénku, které srovnávají rozdíly získaných dat ve dvou po sobě následujících okénkách. Tyto techniky jsou často nazývány jako takzvané absolutní techniky protože nevyužívají žádných předpokladů k vyhodnocení.

Druhým způsobem jsou metody relativní nebo metody založené na modelech. Tyto metody se snaží předpovědět očekávané chování. Datové vzorky, které neodpovídají předpovídaným hodnotám jsou poté nazývány jako zbytkové hodnoty a to, jestli obsahují body změny je vyhodnoceno pomocí absolutních technik.

Anomální časové série

Poslední anomálii představují anomální časové série. Střední odchylka těchto sérií je často výrazně rozdílná od ostatních podobných časových sérií v systému.

Tuto detekci je možné provést vytvořením shluků z podobných časových sérií, které je možné vytvořit například autokorelací nebo spektrální entropií. V takto vytvořených shlucích je poté možné detekovat anomální časové série pomocí výpočtu střední odchylky mezi těžišti jednotlivých časových sérií.

Kapitola 5

Vizualizace dat

Po uložení dat a vytvoření návrhu dashboardů je nutné vybrat technologie k jejich implementaci. V dnešní době je velmi praktické vytvářet informační systém s dashboardy jako webovou aplikaci, což dovoluje uživatelům zobrazit potřebná data na prakticky jakémkoliv zařízení připojeném k internetu.

Populární je provádět samotné sestavování výsledné stránky na klientovi, což je sice limitující z pohledu použití technologií a nutnosti implementovat kontrolu uživatelských vstupů jak na serveru tak na klientovi. Na druhou stranu tento přístup přináší možnost přesunout podstatnou část výpočtu právě na klienta.

K vytvoření zmíněných vizualizací je možné použít nástroje, které jsou vestavěné do všech moderních webových prohlížečů, které podporují specifikaci HTML5 a elementy *canvas* a *svg* v kombinaci s programovacím jazykem JavaScript, nebo jeho rozšířením TypeScript. Oba dva typy elementů je možné při vytváření vizualizací využít, ale vektorová grafika v podobě formátu SVG nám oproti rastrovému elementu canvas přináší možnost vytvářet grafiku, kterou bude možné lépe přizpůsobovat konkrétním zařízením uživatele. U obou přístupů je ale vytváření tradičních vizualizací jejich přímým použití velmi pracné a to nejen v případě vykreslování, ale hlavně předzpracování dat. Oba tyto problémy pomáhá řešit populární knihovna D3.js.

5.1 D3.js

Knihovna D3.js[1] slouží k zjednodušení operací nad daty a objektovým modelem DOM[2]. Za tímto účelem poskytuje jak funkce pro zjednodušení předzpracování dat, tak k práci nad modelem DOM. K funkcím zjednodušujícím prací nad modelem DOM patří například vylepšené selektory, které umožňují zvolit element na základě CSS selektoru. Tento přístup je velmi praktický kvůli možnosti dynamicky generovat elementy a vnořit je do zvoleného elementu. Kromě funkcí obsažených v základním balíčku obsahuje oficiální rozšíření, která jsou určená pro více specifické operace, jako například *d3-geo-projection*[3], které slouží k zjednodušení provádění geografických projekcí.

Tato knihovna poskytuje nízkou míru abstrakce, ale velmi dobrý výkon. Své reprezentace vytváří pomocí standardních HTML prvků díky čemuž je možné zajistit vysokou kompatibilitu napříč různými webovými prohlížeči. I přes to, že je knihovna D3.js typicky

[1] Domovská stránka projektu https://d3js.org/
[2] Specifikace DOM: https://www.w3.org/TR/1998/WD-DOM-19980416/
[3] Rozšíření d3-geo-projection: https://github.com/d3/d3-geo-projection

používána v programovacím jazyce JavaScript, existují pro ni i takzvané typové definice, které dovolují vytvářet nové vizualizace pomocí programovacího jazyka TypeScript, což omezí chyby vzniklé špatným použitím knihovny. I přesto, že D3.js obsahuje vestavěné způsoby vizualizace, jedná se spíše o matematickou knihovnu, která poskytuje velké množství funkcí pro předzpracování dat.

Knihovna D3.js velmi oblíbená při vytváření specializovaných řešení, pro které ještě nebyly navrženy abstraktnější knihovny, nebo právě při tvorbě nových knihoven. Z vytvořených vizualizací je velmi praktické vytvořit komponenty, které bude možné znovu používat jak v rámci aplikace tak napříč dalšími aplikacemi.

5.2 Knihovny pro tvorbu uživatelských rozhraní

Vytvořené komponenty nemusí obsahovat pouze vizualizace dat, ale také zbylé prvky uživatelského rozhraní. Po tomto rozdělení bude možné výsledné uživatelské prostředí tvořit kompozicí vzniklých komponentů, které bude jednodušší na údržbu díky jejich malé velikosti. Tyto znovupoužitelné komponenty je zajisté možné vytvářet ručně, ale častějším případem je využití některé z dostupných a velmi populárních knihoven, které tento proces velmi zjednodušují.

Mnoho knihoven přináší kromě možnosti vytvoření komponentů i další funkce, které mohou být při vytváření uživatelských rozhraní velmi užitečné. Dobrým příkladem přidané funkce jsou šablonovací jazyky, nebo možnosti jednoduše obsloužit interakce uživatele s jednotlivými komponenty. Mezi často používané knihovny patří React a Angular.

React

React[4] je velmi populární a minimalistický aplikační rámec vyvíjený společností Facebook. Aplikace v něm vytvořené se skládají pouze z komponent, které mají definovaný vzhled ve speciálním jazyce JSX[5]. Díky tomuto speciálnímu jazyku je možné kombinovat strukturální zápis HTML s kódem v jazyce JavaScript, který dovoluje obsluhovat akce uživatele.

Jak již bylo zmíněno tento aplikační rámec je velmi minimalistický, a proto je často potřeba ho rozšířit o další funkčnost. Díky jeho popularitě toto nečiní výrazný problém a existuje velká řada komunitních rozšíření, z nichž jsou některá dokonce i doporučená.

Angular

Angular[6] je komplikovanější aplikační rámec, který vyžaduje přesnou strukturu a využití jazyka TypeScript, ale poskytuje robustní způsob k vytvoření webových aplikací. Stejně jako React i Angular rozděluje části vzhledu na komponenty, ale narozdíl od něj mají samotné komponenty pevně předepsanou strukturu, která od sebe odděluje strukturu, vzhled a samotnou funkčnost.

Kromě samotných komponent poskytuje Angular další nástroje, které zjednodušují aspekty, které je často nutné řešit při vytváření frontendových webových aplikací. Díky tomu je možné zavést vyšší míru abstrakce, kterou je velmi jednoduché použít a udržovat. Příkladem této abstrakce může být zjednodušení komunikace mezi serverem a klientem přes takzvané *Observers*, které dovolují jednoduše získávat data ze serveru a současně na jejich

[4] Domovská stránka projektu https://reactjs.org/
[5] Více o jazyce JSX https://reactjs.org/docs/introducing-jsx.html
[6] Domovská stránka projektu: https://angular.io/

základě jednoduše aktualizovat jednotlivé komponenty. Z těchto jej může být výhodnější použít pro velké projekty místo stahovanějšího[7] rámce React.

5.3 Vizualizace tradičních dat

Přístup vytváření znovupoužitelných komponent, které představují grafy, nebo jiné vizualizace, které by bylo možné použít opakovaně již před námi zvolilo více vývojářů. Díky tomu už existují knihovny, které obsahují časté typy vizualizací. Jednotlivé vizualizace jsou poté reprezentovány jako samostatné komponenty, které jsou přizpůsobeny pro některou z vizualizačních knihoven.

Využitím těchto knihoven můžeme zrychlit vývoj samotné aplikace. Při výběru knihovny ale musíme dbát nejen na to, zdali je kompatibilní s námi vybranou knihovnou pro tvorbu uživatelských rozhraní, ale také na to, jaké množství námi požadovaných vizualizací poskytuje. Při výběru většího počtu menších balíčků mohou rozdílné vzhledy vizualizací působit neprofesionálně. V tomto případě je nutné ručně upravit vzhled jednotlivých vizualizací. Tuto situaci velmi zjednodušují knihovny, které dovolují jednoduše změnit svůj vzhled.

Při výběru knihoven jsem se primárně zaměřil na knihovny, které poskytují velké množství vizualizací, aby se předešlo tomuto problému. Sekundárním výběrovým kritériem bylo i to, zdali je možné tyto knihovny jednoduše integrovat do projektů vytvořených pomocí knihovny Angular. Tato knihovna byla vybrána nejen kvůli její popularitě a nástrojům, které poskytuje, ale také kvůli tomu, že je využívána firmou Logimic.

Ngx-charts

Ngx-charts[8] je deklarativní knihovna, která pro vytváření svých vizualizací využívá knihovny D3.js a Angular. Tuto knihovnu vyvíjí společností Swimlane[9] jako software s otevřeným zdrojovým kódem. Narozdíl od mnohých ostatních knihoven ale tato knihovna nemění pouze vzhled výsledných komponent nebo grafů, ale využívá vlastního jádra, matematických funkcí a generátorů obsažených v D3.js a datových vazeb aplikačního rámce Angular k vytvoření vlastních komponentů a vizualizací. Všechny komponenty jsou také zpřístupněny přímo v modulu, díky čemuž je možné je velmi jednoduše využít při vytváření vlastních grafů a vizualizací.

Carbon Charts

Carbon Charts je aplikační rámec vytvoření firmou IBM jako část jejich návrhového systému Carbon[10]. Tento návrhový systém se snaží poskytnout kolekci předpřipravených, znovu použitelných komponentů a vzorů pro zrychlení vývoje.

Carbon Charts narozdíl od aplikačního rámce Ngx-charts vytváří své vizualizace využitím D3.js a vlastních stylů, ale přidává i další a pokročilejší vizualizace, které Ngx-charts neposkytuje, jako je například stromové vizualizace nebo základní geografické vizualizace. Dalším důležitým rozdílem je i platforma. Jelikož se jedná o součást návrhového systému, existují verze pro mnoho aplikačních rozhraní pro tvorbu uživatelských rozhraní včetně již zmíněného Reactu a Angularu, ale i pro použití bez nich.

[7] Srovnání počtu stáhnutí obou balíčků pomocí nástroje npm: https://www.npmtrends.com/angular-vs-react

[8] Domácí stránka projektu: https://github.com/swimlane/ngx-charts

[9] Oficiální stránky společnosti: https://swimlane.com/

[10] Oficiální stránky návrhového systému https://www.carbondesignsystem.com/

Nevýhodou celého rámce je závislost na aplikačním rámci Bootstrap[11]. Tato nevýhoda se projevuje případech, kdy je nutné vizualizace integrovat do systému, který využívá jinou knihovnu.

5.4 Redakční systémy

Alternativou k vytváření vlastního řešení je použití takzvaných redakčních systémů. Tyto systémy jsou většinou tvořeny základní částí aplikace a pomocí uživatelských rozšíření, které celou aplikaci více adaptují pro řešení konkrétního problému. Zástupcem často používaného redakčního systému je systém WordPress[12]. Tento systém je často používaný pro vytváření blogů nebo internetových obchodů a není příliš určený k tvorbě dashboardů. K tomuto účelu je možné použít aplikaci Grafana.

Grafana je multiplatformní webová aplikace s otevřeným zdrojovým kódem, která poskytuje možnost rychle vytvářet vlastní dashboardy a jiné pohledy z velké množiny databázových systémů nebo jiných zdrojů dat. Narozdíl od předchozích technologií je u této aplikace daleko jednodušší začít rychle provozovat a to díky možnosti spustit aplikaci přímo v cloudu výrobce nebo spuštění již hotové aplikace lokálně. Grafana poté dovoluje vytvářet dashboardy přímo v aplikaci a to jednoduchým výběrem.

Kromě již hotové aplikace je možné využít komponent vytvořených pomocí aplikačního rámce React při vytváření vlastní aplikace. Kromě vestavěných vizualizací existuje i velká řada komunitních rozšíření.

5.5 Vizualizace geografických dat

Oproti vizualizaci tradičních dat se vizualizace geografických dat významně liší. Zatímco v případě tradičních dat je předzpracování relativně jednoduché u geografických dat musíme počítat s kulovitým tvarem země. Z tohoto důvodu se používají projekce. Tyto operace transformují zploštělé koule do dvoudimenzionálního prostoru. O předzpracování se může starat například oficiální rozšíření knihovny D3.js s názvem *d3-geo*.

Stejně jako v předchozím případě i pro geografická data existuje mnoho aplikačních rámců, které poskytují velké množství již předpřipravených řešení, které je možné použít. Stejně jako v předchozím případě existují aplikační rámce s otevřeným zdrojovým kódem, ale velmi populární bývají i proprietární a placená řešení například v podobě Google Maps[13]. Tato sekce je zaměřená pouze na některé aplikační rámce s otevřeným zdrojovým kódem.

Leaflet

Leaflet je knihovna založená na jazyce JavaScript a poskytuje možnost vytvářet jednoduchá a velmi výkonná řešení, která budou efektivní i na zařízeních s dotykovou obrazovkou jako jsou například chytré mobilní telefony. K vytváření vizualizací tato knihovna využívá standardních HTML elementů, díky čemuž je velmi jednoduché změnit vzhled nebo další vlastnosti finálního zobrazení.

I přes značné abstrakci vyžaduje Leaflet dobrou znalost jazyka JavaScript. Základní balíček obsahuje pouze nástroje pro tvorbu a práci se samotnou mapou, pro jakékoli složitější

[11] Knihovna Bootsrap: https://getbootstrap.com/
[12] Redakční systém WordPress: https://wordpress.com/
[13] Google maps: https://developers.google.com/maps/documentation

použití je možné použít velké množství již existujících rozšíření, která jsou vyvíjena velmi aktivní komunitou.

Geovisto

Geovisto je knihovna[14] založená na rámcích Leaflet a D3.js a rozšiřuje možnosti, které Leaflet podporuje. K vytvoření mapy je využit aplikační rámec Leaflet a k vytváření vizualizací nad tímto podkladem slouží D3.js. Jednotlivé vizualizace jsou uspořádány do vrstev, díky čemu je možné zpracovávat a vizualizovat vícevrstvá geografická data.

Hlavní výhodou je způsob ovládání knihovny. Zatímco Leaflet vyžaduje znalost jazyka JavaScript Geovisto je založené na deklarativním způsobu čímž usnadňuje vytváření vizualizací. Knihovna jako taková je velmi modulární, díky čemuž je možné stahovat používat pouze ty části, které jsou potřeba nebo velmi jednoduše vytvářet další potřebná rozšíření.

[14] Webová stránka produktu Geovisto https://github.com/geovisto

Kapitola 6

Analýza

Před výběrem technologií je nutné analyzovat cílovou skupinu uživatelů, pro které bude vytvořený software určen. Každá skupina uživatel může na systém klást různé požadavky a tím ovlivnit data, která bude nutné zobrazit, a způsoby, kterými budou tato data zobrazena.

Cílovou skupinu může představovat například analytik zodpovědný za správný chod zařízení chytrého města, bude nutné připravit pohledy takovým způsobem, aby odpovídaly požadavkům analytických dashboardů popsaných v 4.1.2. Tyto požadavky ovlivní i výběr technologií. Ve stejném případě využití analytikem bude nutné zvolit databázový systém i vizualizace vhodné k provádění agregací a dalších transformací dat.

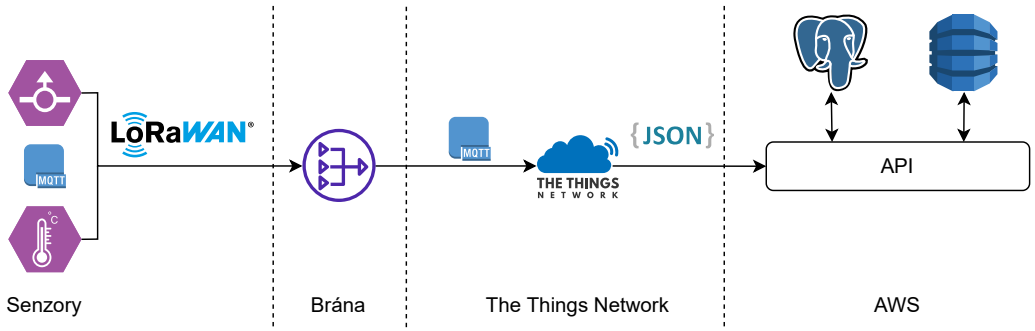
Kvůli rozmanitosti požadavků je vhodné se při řešení zaměřit na jednu konkrétní skupinu uživatelů. Při řešení této diplomové práce jsem se zaměřil na analytické dashboardy, které své využití typicky nachází jak v průmyslu, tak v chytrých městech. Stejnou problematikou se zabývá i firma Logimic.

6.1 Logimic

Logimic[1] je česká firma, která vytváří IoT systémy pro chytrá města, nebo průmyslové účely a to nejčastěji pro sledování a nebo řízení. Firma se hlavně zaměřuje na bezdrátový přenos dat a ovládacích příkazů k zařízením a svým zákazníkům poskytuje nejen návrh systémů, ale i samotné hardwarové i softwarové řešení, cloudovou infrastrukturu a odborné konzultace. K provozu cloudových řešení Logimic využívá již zmíněnou službu AWS od firmy Amazon. Díky tomuto přístupu může firma jednoduše poskytovat služby s vysokou dostupností, z velké části vyřešeným zabezpečením a nízkou odezvou prakticky po celém světě. Zpracování dat je firmou řešeno následujícím způsobem:

Data ze senzorů a jiných koncových zařízení, která jsou připojena do sítě, jsou ve formátu MQTT přenesena nejčastěji pomocí technologie LoRaWan k nejbližší dostupné bráně. Ta data odešle, stále v podobě MQTT zpráv, na cloudové řešení The Things Network, kde dojde k dekódování zprávy a její reprezentaci odpovídající zprávou ve formátu JSON. Takto připravené zprávy jsou z velké části připravené k zpracování a uložení do databázových systémů. Před samotným uložením je ale nutné provést další úpravy, kterými je například převedení do odpovídajícího formátu a výpočet odvozených hodnot, které samotné senzory neposkytují. Takto předzpracovaná data jsou použita k aktualizaci aktuální hodnoty uložené v relačním databázovém systému PostgreSQL a také uložena do NoSQL databázového systému DynamoDB. Schéma architektury celého procesu zpracování dat je na obrázku 6.1.

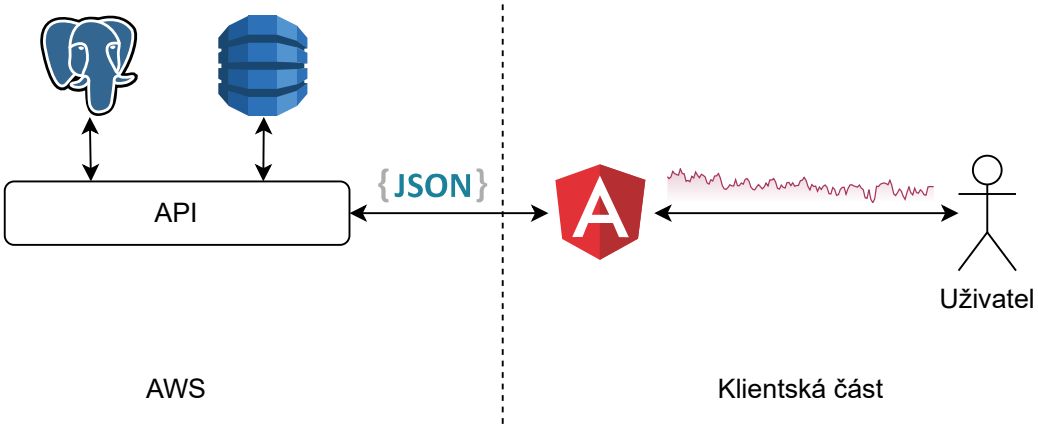
[1] Oficiální stránky společnosti: https://www.logimic.com/



Obrázek 6.1: Postup ukládání dat ze senzorů.

O toto zpracování se stará vrstva, která je tvořená z takzvaných AWS Lambda funkcí, které dovolují vykonávat obslužný kód bez vytváření aplikačního serveru. Jejich další výhodou je to, že tyto funkce jsou po určitou dobu[2] od posledního požadavku připraveny k obsloužení dalších příchozích požadavků bez nutnosti provádět celou inicializaci. V případech, kdy není tato funkce zapotřebí delší dobu, je její běh plně zastaven. Díky tomuto přístupu je možné, oproti tradičním aplikačním serverům, redukovat cenu na úkor nízkého výkonu požadavků, které provádí plnou inicializaci.

Získávání dat z databázových systémů probíhá podobným způsobem jako jejich zápis. Při výběru je tedy kontaktována odpovídající Lambda, která autentifikuje a autorizuje uživatele a pokud je k dané operaci oprávněn, vyhodnotí daný dotaz. Výsledky dotazů jsou frontendové aplikaci zasílány ve formátu JSON. Získaná data jsou uživatelům dodaných systémů zobrazována frontedovou aplikací založenou na aplikačním rámci Angular. Vizualizace, které nevyžadují celé časové série získávají svá data z relační databáze. Schéma architektury pro zobrazení dat na obrázku 6.2. Časové série jsou získávány z databázového systému DynamoDB, což přináší nevýhody popsané v 3.2.1.



Obrázek 6.2: Postup zobrazování dat.

[2]Více o chování AWS Lambda: https://lumigo.io/blog/this-is-all-you-need-to-know-about-lambda-cold-starts/

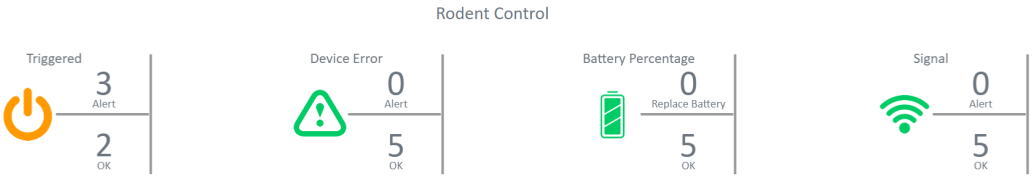
Tyto nevýhody jsou velmi omezující při tvorbě dashboardů, které využívají velkého množství dat, nebo vyžadují výpočet jiných hodnot a to například agregací. Dále také představují značné omezení v případě, že by se firma rozhodla některému ze svých zákazníků zřídit systém v interní infrastruktuře a prostoru zákazníka.

6.2 Současný stav

V současném stavu odpovídá velká část poskytovaných dashboardů analytickému typu. Uživateli při vstupu do aplikace zobrazena vizualizace na obrázku 6.3, která poskytuje agregované pohledy nejdůležitějších KPI daného typu senzoru. Pokud jsou hodnoty všech KPI v této skupině v nastaveném rozsahu, jsou odpovídající části vizualizace podbarveny zelenou barvou. Pokud je ale některá z hodnot mimo daný rozsah, je vygenerováno upozornění a je změněna barva na úvodním dashboardu. Na tomto obrázku můžeme uprostřed vidět stav zařízení pro deratizaci. Všechna zařízení tohoto typu mají dostatečný signál, dostatečnou zbývající kapacitu baterie a žádné z nich nevygenerovalo chybové upozornění. Avšak barva pravé horní části indikátoru je změněná, což značí, se došlo k sepnutí pasti.

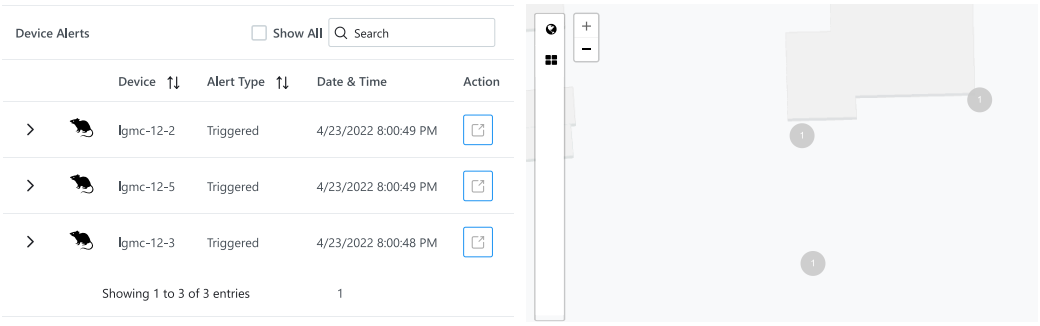
Obrázek 6.3: Zobrazení skupiny KPI

Po kliknutí na vizualizaci je možné přejít k méně agregovanému pohledu, kde vidíme detail každé kategorie KPI. V tomto přehledu jsou zobrazeny všechny skupiny KPI podobných způsobem jako v předchozím pohledu. Na obrázku 6.4 je zachycen stav stejného deratizačního zařízení. Konkrétně zde vidíme, že 3 z 5 zařízení se nachází v sepnutém stavu.



Obrázek 6.4: Zobrazení agregovaných KPI ve skupině

Po kliknutí na číslo 3 přecházíme k pohledu zobrazujícímu všechna zařízení, která vygenerovala upozornění o stavu zařízení. Kromě názvu zařízení a data a času vytvoření upozornění vizualizace obsahuje i mapu zobrazující umístění zařízení. Na obrázku 6.5 jsou zobrazena právě ta 3 zařízení, která vygenerovala upozornění.

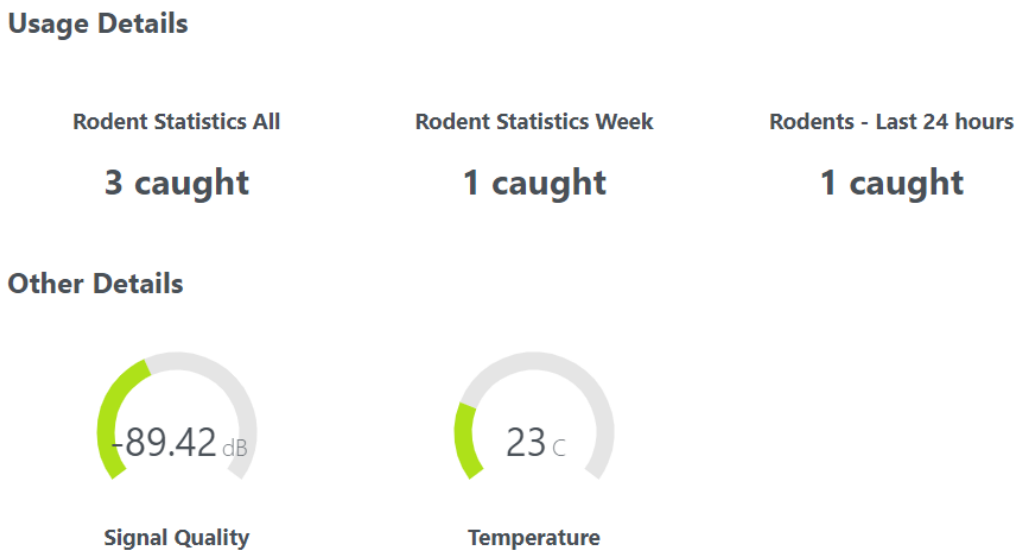


Obrázek 6.5: Zobrazení upozornění konkrétního typu zařízení.

V neposlední řadě je možné se zanořit na nejnižší úroveň a sledovat samotné hodnoty jednotlivých zařízení. V současné době je tato část dashboardu ve vývoji a jedná se pouze o její prvotní návrh. Obrázek 6.6 obsahuje statistiku signálu zařízení.

Obrázek 6.6: Detail konkrétních hodnot zvoleného zařízení

Toto zobrazení dovoluje vybrat jednu z měřených hodnot senzoru a zobrazit její průběh za zvolené časové období. Průběh je zobrazen ve formě interaktivního spojnicového grafu, kdy je po najetí na část grafu získat odpovídající hodnotu v daném čase. Dále také nejnižší úroveň obsahuje textové informace o samotném senzoru a přehled rychlý přehled aktuálních informací a vybraných agregovaných hodnot. Na obrázku 6.7 vidíme příklad tohoto zobrazení.



Obrázek 6.7: Přehled agregovaných hodnot zvoleného zařízení

Z velké části jsou zmíněné vizualizace efektivní pro zobrazení vyžadovaných dat. Výjimkou je zobrazení samotných hodnot zařízení, které může být v některých případech neefektivní. Například při zobrazení dlouhého časového úseku dojde k načtení velmi velkého počtu neagreagovaných vzorků. Počet vzorků způsobí zvýšení potřebného výkonu jak pro zařízení uživatele, tak pro server poskytující data. Ideálním řešením by bylo zvolit úroveň agregace automaticky na základě délky zvoleného časového období, které by mohl uživatel ručně změnit.

Časové období, ve kterém se uživateli zobrazují data je možné zvolit pouze z předdefinovaných hodnot týden, měsíc a rok. Tento fakt přináší další limitaci. V některých případech může být pro uživatele potřebné zobrazit časové období, které neodpovídá žádné z těchto možností.

Poslední pro uživatele zajímavou funkcí by mohla být možnost srovnat data dvou zařízení, nebo data stejného zařízení v odlišném období. Například tedy srovnání průběhu teploty stejného senzoru ve stejném měsíci jiného roku.

V současné době systém neprovádí žádnou ze zmíněných kontrol odlehlých hodnot. Tyto hodnoty, je ale stále možné detekovat, protože bude vyhodnocena v rámci KPI.

Výsledné požadavky se tedy týkají jak backendové tak frontednové části řešení. Na backendové části je zapotřebí buď přejít na efektivnější databázový systém, který by byl schopný obsloužit požadavky rychleji a efektivně vytvářet agregace, nebo rozumnou metodu předzpracování, uložení a výběru dat pro zobrazení. Na frontendové části je nutné se zaměřit primárně na statistické pohledy, které jsou nyní pouze velmi jednoduché.

Kapitola 7

Návrh

Po provedení analýzy aktuálního stavu bylo nutné nahvrhnout možná efektivní řešení. Tato řešení se týkají jak problému zobrazení dat ze senzorů, tak jejich efektivního ukládání a zpracování. Po zvolení řešení obou problémů je nutné i zvolit způsob, kterým budou tyto aplikace propojeny. Operace by bylo možné provádět přímo ze zobrazení, ale tento přístup není praktický pro reálné použití.

Tato kapitola je rozdělena na tři části, kde každá představuje řešení jednoho z problémů. I přes to, že spolu tyto problémy souvisí, existují různé způsoby řešení, které na sobě nejsou závislé.

7.1 Ukládání a zpracování dat

Při zpracování dat je nutné vzít v úvahu i jiná data než časové série senzorů. Tyto data mohou představovat například neměnné metadata o samotném senzoru, jako například název, identifikátor senzoru nebo jednotky, ve kterých daný senzor měří svá data, ale i ostatní informace, které jsou potřebné pro běh samotného informačního systému jako například informace o uživatelích.

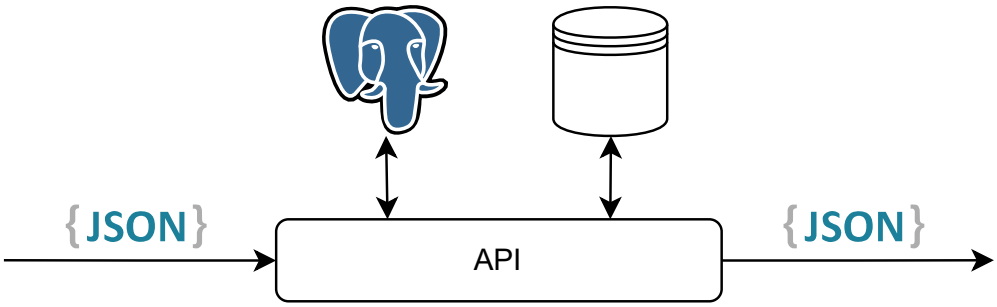
Jedním ze způsobů, jak se postavit k uložení obou typů dat, by bylo vybrat jedinou databázi, která by sloužila k uložení obou typů dat. V případě použití relační databáze se musí počítat s problémy popsanými v kapitole 3. I na straně NoSQL databází není jednoduché vyřešit tento požadavek a to zejména kvůli malému výkonu relací. Ty je možné vyřešit denormalizací, to ale přináší své vlastní problémy, kterými je kromě velikosti i nutnost aktualizovat záznamy, které mohou být několikrát redundantně obsaženy i v několika tabulkách.

Jako lepší možnost připadá v úvahu použití dvou databázových systémů, kde jeden slouží pouze k uložení časových sérií a druhý, často relační databázový systém, slouží k uložení ostatních dat. Přístup k časovým sériím se může lišit. V některých systémech může být výhodné použít dokumentovou NoSQL databázi, ve které budou části sérií reprezentovány dokumentem v formátu, který bude přesně odpovídat konkrétnímu využití.

Tento přístup poskytuje velmi vysoký výkon za cenu prostoru a omezení ve volnosti výběrů. Druhým způsobem je využít takového NoSQL systému, který dokáže efektivně pracovat s časovými sériemi. Tento přístup je pro firmu Logimic nejpřístupnější. Jak již bylo, zmíněno firma využívá relační databázi pro ukládání dat. I přes to toto řešení poskytuje možnost efektivně provádět agregace nad daty může být praktické ukládat hodnoty složitější operací.

Tyto hodnoty mohou být uloženy jak do zvoleného NoSQL systému, tak do relační databáze. Pokud budou tato data ukládána dlouhodobě, je efektivnější data ukládat zpět do NoSQL systému, kde mohou být v případě nutnosti znovu využita, nebo minimálně efektivně uložena. Naopak data, která jsou určena pouze k jednomu zobrazení, například aktuální hodnota senzoru, která bude pokaždé zobrazena ve vizualizaci, je efektivnější uložit do relační databáze.

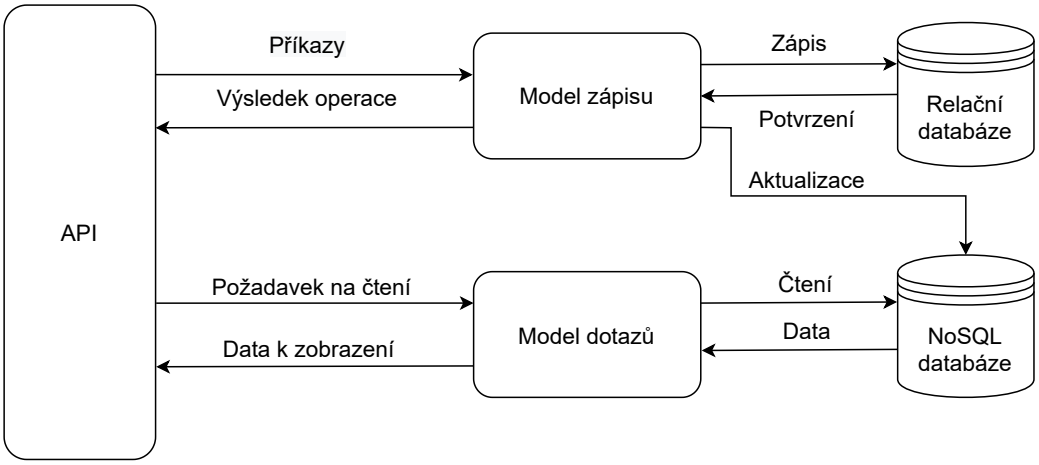
Navrhovaná architektura, zachycená na obrázku 7.1, tedy velmi připomíná architekturu, která už je ve firmě Logimic používaná. Hlavním rozdílem je volba jiného databázového systému, který nahradí používaný databázový systém DynamoDB systémem jiným, a efektivnějším pro ukládání a provádění operací nad časovými sériemi.



Obrázek 7.1: Navrhovaná databázová architektura.

V případě velkého zvýšení počtu požadavků by bylo možné relační databázi škálovat pomocí principu CQRS[1]. Za tímto účelem by bylo nutné přidat do systému další NoSQL databázi, nebo jiné úložiště určené k rychlému čtení a zápisu. Vkládání, úpravu nebo mazání dat, tzv. příkazy, obslouží model zápisu, který zajistí propagaci změn do obou databázových systémů. Aby nedošlo k inkonzistenci dat mezi databázovými systémy je nutné vyčkat na výsledek operace. Pokud je operace úspěšná data mohou být propagována do NoSQL databáze. Obsloužení dotazů je poté provedeno pouze pomocí NoSQL databáze. Díky tomu jsou sníženy nároky na výkon relační databáze. Postup vykonání příkazů a dotazů v obrázku 7.2.

[1]Více o návrhovém vzoru CQRS: https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs



Obrázek 7.2: Architektura při použití principu CQRS.

Databázový systém určený k ukládání časových sérií je možné škálovat přidáním odpovídajících uzlů s replikací, nebo exkluzivním rozdělení dat mezi více instancí zvolené databáze pro uložení časových sérii. V případech extrémně častého čtení by bylo možné zapsat často dotazované série do NoSQL databáze určené pro rychlé čtení.

Se stále zvyšujícím se počtem vzorků v databázi bude narůstat její velikost. Tento problém má jednoduché řešení. Často nás zajímají přesné záznamy pouze u aktuálních dat. Čím jsou data poté starší, tím se můžeme spolehnout na více nepřesné vzorky a staré vzorky můžeme odmazat z databáze úplně. Tyto operace jsou často podporovány v NoSQL databázových systémech a u senzorů, kde odmazání dat nepřináší žádné nevýhody. Oba dva tyto přístupy bývají doporučovány i výrobci databázových systémů[2]

7.2 Návrh zobrazení

Aktuální zobrazení skupin KPI a stupňů zanoření je efektivní pro zobrazení vysokoúrovňových dat v aktuálním stavu. Horší situace je v případě zobrazení statistik konkrétního zařízení. Tyto statistiky jsou velmi jednoduché a dovolují zobrazit pouze jednu hodnotu zařízení. Toto je velmi limitující například v případě, kdy uživatele zajímá vztah více hodnot a to například v závislosti jedné hodnoty na druhé, jako tomu může být v případě vnitřní a venkovní teploty.

Žádným způsobem také není možné srovnat hodnoty dvou zařízení stejného typu. Toto zobrazení je méně důležité než jmenované předchozí a hodnoty zařízení ve stejné skupině je do jisté míry možné pomocí nastavení stejných limitních hodnot KPI. Toto srovnání by bylo více praktické vyřešit například zobrazením hodnot dvou senzorů stejného typu v jednom grafu. U kvantitativních nominálních hodnot je implementace tohoto srovnání jednoduchá a je možné ji provést zobrazením dvou nebo více spojnic spojnicového grafu. Větším problémem je způsob srovnání hodnot v ostatních případech. Například u senzoru stavu otevření dveří může být zajímavé srovnat využití dveří. Toto srovnání by bylo možné provést na základě srovnání počtu změn stavu senzoru.

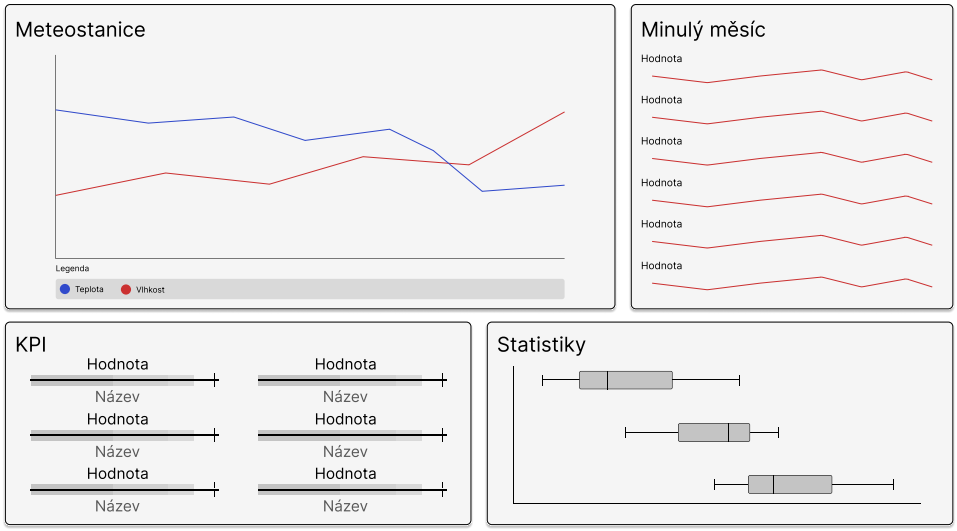
[2]Doporučení společnosti influxdata k používání databázového systému InfluxDB https://www.influxdata.com/the-best-way-to-store-collect-analyze-time-series-data/

Samotná frekvence změny stavu ale není dostatečným indikátorem využití. u předchozího příkladu mohou dveře zůstat například otevřené při velkém počtu lidí, kteří procházejí. Situaci neřeší dokonce ani kombinace obou těchto faktorů. Podobná situace je u více složitých požadavků. Vytvořené řešení by vždy bylo velmi závislé na problémů, který má řešit, takže by nebylo možné rychle přidávat a zprovozňovat a udržovat nová zařízení.

Kvůli těmto limitacím je jediným řešením se snažit co nejvíce použít generické prvky zobrazení, kde by si uživatel mohl zvolit operaci, kterou chce nad zvolenou datovou sadou provést. Konkrétním příkladem může být spojnicový graf, který umožní uživateli zvolit konkrétní pole senzoru, časově období a agregační operaci, která má být použita.

Tato vizualizace může tento problém vyřešit z numerických dat. U jiných typů dat může být užitečnější omezit uživateli výběr typu agregace a zobrazit v grafu pouze momenty, kdy dojde ke změně stavu. Tyto změny mohou být zobrazeny použitím spojnicového grafu, nebo jiným speciálním typem, který by se podobal sloupcovému grafu s jedním sloupcem. Tento sloupec by byl rozdělen na segmenty podle časů jednotlivých změn.

V případě některých zařízení by také mohlo být prospěšné zobrazit jeho celkovou dostupnost. Výpočet dostupnosti může být založen na počtu přijatých zpráv od tohoto zařízení. Problém s tímto řešením je v případech, kdy zařízení zasílá pouze změny svého stavu, jak tomu je například u senzorů stavu dveří.



Obrázek 7.3: Navrhované zobrazení.

Zobrazení limitních hodnot KPI má také určité nedostatky. K tomuto účelu jsou využity budíkové grafy, které mění svoji barvu, pokud dojde k překročení některého prahu. K zobrazení těchto hodnot připadá v úvahu využít prvku, který by poskytoval možnost vizuálně zobrazit i hodnoty prahů a to způsobem umožnujícím i vizuální srovnání. U některých numerických hodnot je také výhodné zobrazit jejich vývoj pomocí malého počtu agregovaných hodnot. K vizualizaci těchto hodnot je nejlepší využít graf typu sparkline.

Z těchto popsaných bloků se může skládat jeden z vytvořených pohledů. Jeho hlavní informativní částí bude interaktivní spojnicový graf, podpořený přehledem agregací vybraných hodnot pomocí grafů typu sparkline doplněný zobrazením aktuálního stavu jak

v podobě jednotlivých KPI tak jiných hodnot jako například datum a čas posledního připojení senzoru k síti nebo verze software zařízení. Grafický návrh tohoto zobrazení 7.3.

Toto navrhované zobrazení nabízí způsob, jak uživateli zobrazit aktuální stav zařízení s dostatečným porovnáním s historií tak, aby bylo na jeho základě možné vyhodnotit aktuální stav. Podobným způsobem s ohledem na problémy, které přináší zobrazení více zařízení v jednom pohledu, by bylo možné vyřešit i srovnání dvou senzorů.

7.3 Zpřístupnění dat

Ke komunikaci mezi databázovým systémem a aplikací zobrazující data je zapotřebí vytvořit dedikovanou vrstvu. Na cloudovém řešení připadá v úvahu využít takzvaných „serverless“ funkcí. Příkladem těchto funkcí jsou AWS Lambda funkce. Druhou možností je implementace aplikačního serveru, který bude tato data zpřístupňovat.

V tomto konkrétním případě přichází v úvahu implementovat vlastní aplikační server, který bude v případě nutnosti možné jednoduše spustit jak na lokálním stroji tak, na cloudovém řešení. Tato aplikace musí zpřístupňovat jak možnosti výběrů dat z databázového systému, tak ukládat data ze samotných zařízení. Kvůli různorodosti vstupních dat je nutné, aby aplikace kladla velmi nízký nárok na data. Ve většině případů, u časových sérií, stačí k jednoznačnému určení daného vzorku pouze unikátní identifikátor zařízení a časová značka s časem, ve kterém byl daný vzorek pořízen. Z tohoto důvodu musí vstupní data vždy obsahovat unikátní identifikátor zařízení. Pokud chybí časová značka může být nahrazena vhodným odhadem, jako například časovou značkou pořízenou při vkládání záznamu. Ostatní položky by měly být co nejvíce volitelné a to jak svým datovým typem tak názvem.

I přes to, že data jsou přenášena pomocí MQTT, bude aplikace zpracovávat pouze předpřipravená data v již dekódovaném formátu. Dekódování mohou provádět brány zmíněné v kapitole 2, cloudová řešení jako The Things Stack[3], nebo jiné mikroslužby. Díky tomu bude možné použít aplikaci ve více případech, než kdyby přijímala data ve formátu MQTT. Zároveň se sníží komplexita celé aplikace. Stejný přístup bude zachován i v případě autentifikace a autorizace.

I z důvodu jednoduché integrace byla při návrhu zvolena architektura REST. Kromě koncových bodů pro uložení dat bude aplikace poskytovat koncové body reprezentující výběry. Tyto koncové body musí umožnit výběr dat nejen na základě zadaného časového období, ale také zvoleného typu agregace a doby, po kterou se na tato agregace provést. Tyto body nebudou dostačující pro všechny operace, které mohou vyžadovat vizualizace. Příkladem této operace může být výběr časových momentů, ve kterých se změní hodnota parametru.

Praktickým řešením je vytvořit koncový bod pro každou další operaci, kterou bude nutné provádět. Tyto operace mohou odpovídat efektivním dotazům. O použití koncových bodů je praktické dělat záznam, díky kterému je možné vyhodnotit jeho využití, nebo útoky na něj. Tyto záznamy je možné ukládat jak do relačního, tak do NoSQL databázového systému. I samotné záznamy tvoří svým způsobem časové série, protože přibývají a mění svoji důležitost s ohledem na čas.

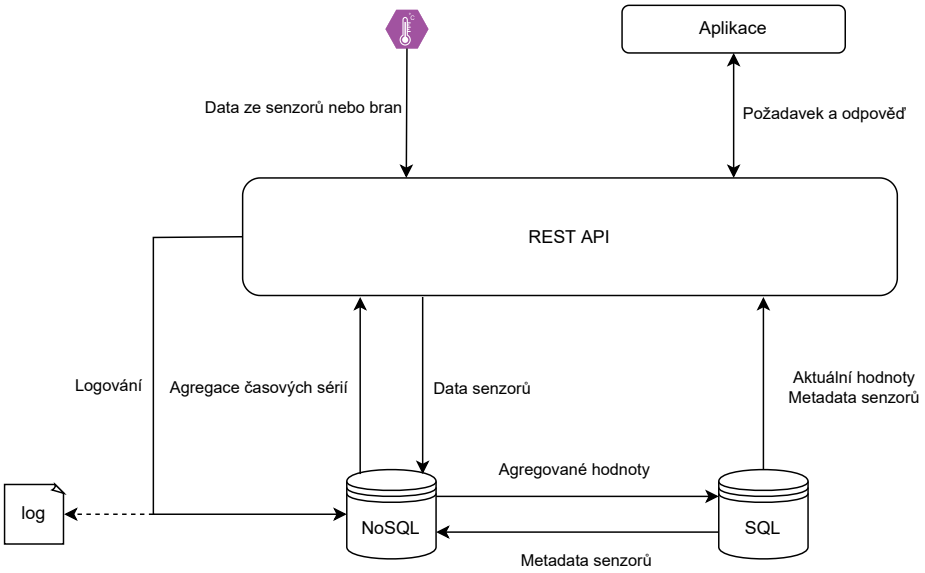
Všechny operace by mělo být možné provádět jak pro jednu hodnotu jednoho senzoru, tak pro více hodnot i různých senzorů v jednom požadavku. Díky tomuto přístupu by bylo možné získat většinu dat pro celý dashboard, nebo všechna data pro složitější vizualizaci

[3]The Things Stack: https://www.thethingsindustries.com/docs/

pomocí jednoho požadavku. Získání informací větším počtem požadavků by mohlo být jednodušší na implementaci, ale na druhou stranu by mohlo vyprodukovat větší množství požadavků, které bude muset obsloužit jak vrstva API tak samotné databázové systémy. Tento způsob tak vytváří opakovanou režii v obsluze každého požadavku, jako například připojení k databázovému systému, pokud nezůstalo otevřené, nebo spuštění odpovídající obslužné rutiny.

Toto řešení má negativní dopad na požadavky, které nabývají na složitosti. Z tohoto důvodu přichází v úvahu neserializovat data potřebná k jejich vykonání do URL, ale zaslat je v těle požadavku. Z tohoto důvodu je z pohledu sémantiky na místě použít HTTP metodu POST. Bylo by možné použít i metodu GET, ale ta nemá definovanou sémantiku těla požadavku[4] a tento přístup není doporučován. Samotné API je také vhodné dělit na základě zodpovědností. V tomto případě přichází v úvahu oddělit část poskytující obecné informace o zařízení, které využívají data z relační databáze od statistik.

Obrázek 7.4 obsahuje celou navrhovanou strukturu. Dekódovaná data ze zařízení a senzorů jsou odesílána na odpovídající koncový bod vytvořeného aplikačního serveru. Při přijetí dat jsou data vždy uložena do NoSQL databáze pro časové série a pokud už existuje záznam v relační databázi je i tento záznam aktualizován. Díky tomu je možné připojit senzory před inicializací záznamu o senzoru v relační databázi.



Obrázek 7.4: Navrhovaná architektura řešení.

Vkládání aktuálních záznamů časových sérií do relační databáze může být obslouženo třemi způsoby. První z nich spoléhá na nástroje poskytované databázovým systémem NoSQL a je zobrazen na obrázku 7.4. Druhý způsob je aktualizace hodnoty při každém vložení do databáze. Toto řešení může být praktické kvůli jeho jednoduchosti na implementaci a také díky tomu, že nespoléhá na na nástroje poskytované zvoleným typem NoSQL databáze. V případě, kdy bude v systému velký počet zařízení, tento přístup způsobí zvýšení nároků na relační databázový systém. Posledním způsobem je aktualizace plánovaným spuštěním speciální rutiny, například použitím utility *cron*.

[4]RFC7231: https://www.rfc-editor.org/rfc/rfc7231#page-25

Kapitola 8

Implementace

Po provedení návrhu je nutné zvolit řešení, jeho konkrétní implementační detaily a v neposlední řadě provést jeho realizaci. Tato sekce popisuje všechny části implementace v pořadí, ve kterém byly při řešení prováděny.

Všechny části řešení s výjimkou databázových testů jsou implementovány v jazyce TypeScript. Tento jazyk byl zvolen díky transpilaci do JavaScriptu, která poskytuje možnost použít stejný jazyk jak pro frontend, tak backend a částečně také kvůli stejnému použití firmou Logimic. Databázové testy jsou implementovány přímo v jazyce JavaScript.

8.1 Výběr databázového systému

Výběr databázového systému byl primárně zaměřen na systém, který bude umožňovat ukládání a zpracování dat ze senzorů. Zbylá data zůstávají uloženy v relačním databázovém systému PostgreSQL. Při výběru byly testovány systémy zmíněné v kapitole 3 na základě následujících metrik:

- prostředí – určuje, zdali je databázový systém možné jednoduše využít jak v cloudu, tak na lokálních instancích bez výrazných omezení,
- využití – k čemu je možné využít daný databázový systém, například zdali je určený pouze k ukládání časových sérii, nebo je možné ukládat i jiná data,
- poskytované operace – určuje, zda databázový systém poskytuje operace, nebo dotazovací jazyk, který umožňuje efektivní práci s časovými sériemi,
- výkon operací – rychlost provádění operací,
- efektivita uložení – velikost uložených dat a režie.

Po stanovení těchto požadavků se projevil dosavadní databázový systém DynamoDB jako nepraktický. I přes to, že poskytuje možnost velmi rychlého zápisu a čtení konkrétní položky libovolné struktury, nepodporuje žádné speciální operace. Tento fakt se projevil v již existujících zobrazeních, kde způsobily nepříjemné čekání.

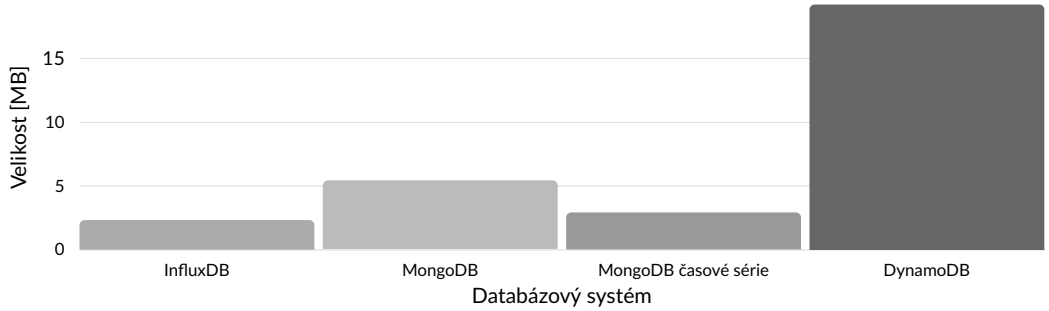
Složitější bylo srovnání u databázových systémů InfluxDB a MongoDB. Na první pohled poskytují oba systémy velmi podobné výhody. Největším rozdílem je to, že InfluxDB je více specializované na práci s časovými sériemi než MongoDB.

Aby bylo možné rozhodnout, který databázový systém je vhodnější pro využití v této konkrétní situaci, bylo nutné využít posledních dvou metrik. Za tímto účelem byl vytvořen jednoduchý program, který testuje rychlosti vložení a čtení dat a provádění agregací.

K testování byla vybrána data tradičně ukládaná firmou Logimic. Použití syntetických dat by mohlo negativním způsobem ovlivnit výsledky testování a vést k výběru méně vhodného řešení. V průběhu testování byl měřen čas jak jednotlivých operací tak celkového trvání celé sady. Díky tomu bylo možné srovnat jak celkový výkon tak trvání operací, které se budou často opakovat.

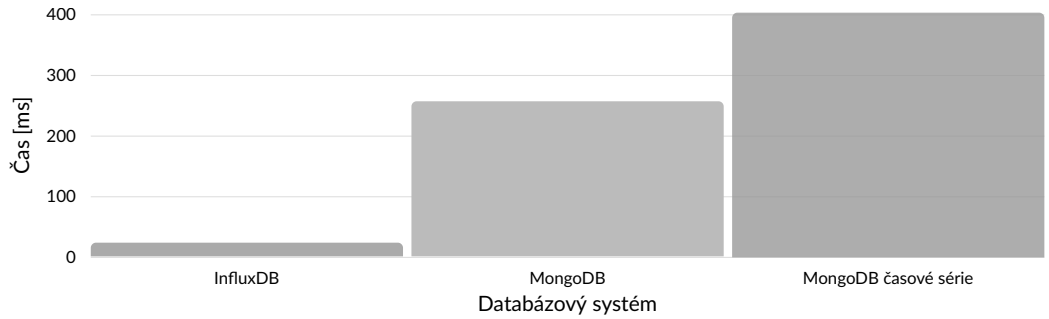
Srovnání velikosti bylo provedené velmi jednoduchým způsobem. Data byla vložena do obou systémů. Následně byla srovnána celková velikost na disku spotřebována k uložení dat. Pro referenci, zdali se přechod doopravdy vyplatí, byly okrajově testovány i už používané systémy DynamoDB a PostgreSQL.

K srovnání velikosti byla použita datová sada 43 tisíc vzorků jednoho zařízení o celkové velikosti 29,6 MB. V tomto srovnání byl nejlepší databázový systém InfluxDB, který k uložení vyžaduje pouhých 2,3 MB. K tomuto výsledku se velmi blíží i MongoDB v režimu pro časové série, které vyžadovalo 2,94 MB. Ostatní databázové systémy vyžadovaly 5,4 MB pro MongoDB, 19 MB DynamoDB a 92 MB pro PostgreSQL. Čtyři nejlepší výsledky v 8.1.



Obrázek 8.1: Výsledky testování velikosti.

Na stejné testovací sadě bylo provedeno i testování operací. Tradiční operace výběru a vložení byly ještě doplněny o agregaci průměrem. V průběhu vývoje testovacích funkcí se hned projevila jedna z výhod systému InfluxDB. Zatím co u ostatních bylo nutné implementovat funkci, ať už v jazyce JavaScript, nebo nativním jazyce daného systému, InfluxDB poskytoval tuto a mnohé další operace ve své standardní knihovně. Srovnání třech nejrychlejších systémů v obrázku 8.2.



Obrázek 8.2: Výsledky testování pro agregační operaci průměr.

Stejně jako v předchozím případě vychází nejlépe systém InfluxDB, který tuto část testu dokončil za 24 ms. Databázový systém MongoDB v obou režimech zaostával s časy 257 ms a

403 ms. Systémy PostgreSQL a dosáhly času 7782 ms a DynamoDB 15589 ms. Tento trend se opakoval i pro ostatní operace. Jedinou výjimkou bylo vložení jednoho záznamu, kde byl nejrychlejší databázový systém DynamoDB s časem 1,27 ms. Systém InfluxDB v tomto případě zaostával za ostatními s časem 39 ms. Tento rozdíl mohl být způsobem nutnou transformací vstupních dat, kterou ostatní systémy neprovádí.

Nakonec byl zvolen databázový systém InfluxDB. Není sice plnou náhradou DynamoDB, ale z vybraných systémů se jeví jako nejpraktičtější a nejrychlejší pro jeho zamýšlené využití v této konkrétní situaci. V případě nedostatku výkonu v budoucnosti by bylo možné využít databázový systém MongoDB jako úložiště optimalizované pro rychlé čtení. Systém DynamoDB je v tomto ohledu sice rychlejší, ale problémem zůstává jeho závislost na cloudovém řešení AWS.

8.2 Aplikační rozhraní

Jak již bylo zmíněno v návrhu 7.3, je praktické vytvořit vstvu, která bude oddělovat aplikaci zobrazující data. Pro implementaci byl zvolen webový aplikační rámec Express.js[1]. Tento rámec prezentuje možnost vytvořit rychlou a robustní aplikaci velmi jednoduchým a intuitivním způsobem. Díky jeho popularitě také existuje řada rozšíření, které je možné při implementaci použít.

Při prvotním testování tohoto řešení byl detekován nárůst zdrojů potřebných k obsloužení odpovídajících lambda funkcí na straně AWS. Ke stejným závěrům došli i další uživatelé, kteří s touto kombinací experimentovali[2]. Kvůli tomuto faktu se vývoj této aplikace nakonec rozdělil na následující dva proudy: lokální API je reprezentováno aplikačním serverem založeným na rámci Express.js a ekvivalentním API založeném na AWS Lambda funkcích.

Tento fakt byl jeden z důvodů, který vedl na implementaci nezávislé vrstvy, která bude zjednodušovat přístup k datům v databázi a bude možné ji využít v případě obou řešení. Díky tomu se omezí kód, který je nutné duplikovat mezi aplikacemi. Tato vrstva je založená na referenčním klientu pro InfluxDB[3].

Na strukturu vstupních dat jsou kladeny velmi malé nároky. Vstupní struktura musí obsahovat název měření, ke kterému patří, nejčastěji reprezentováno pomocí identifikátor senzoru, a následně vlastní data. Při ukládání dat mohou být určité položky označeny za metadata, což umožní rychlejší filtrování na jejich základě.

8.2.1 Lokální API

Aplikační server Express.js byl doplněn dalšími nástroji, které zjednudušují vývoj a zajišťují škálovatelnost řešení. Mezi nejdůležitější z použitých nástrojů patří balíček pro vkládání závislostí typedi[4] a nástroj zjednodušující vytváření REST aplikací tsoa[5].

Díky nástroji tsoa je možné rychle a přehledně vytvářet řadiče, anglicky *controller*. Při použití dostupných dekorátorů v každém řadiči je možné vygenerovat jak odpovídající část směrovače, anglicky *router* tak specifikaci samotného API v podobě OpenAPI[6] specifikace.

[1] Aplikační server Express.js: https://github.com/expressjs/express

[2] Nevýhody Express.js na AWS: https://medium.com/dailyjs/six-reasons-why-you-shouldnt-run-express-js-inside-aws-lambda-102e3a50f355

[3] InfluxDB klient pro JavaScript: https://github.com/influxdata/influxdb-client-js

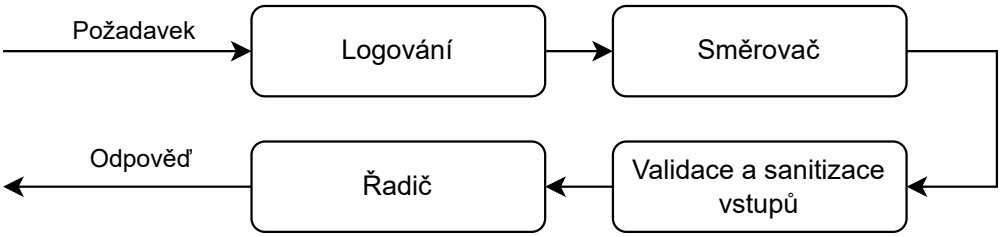
[4] Nástroj typedi: https://github.com/typestack/typedi

[5] Nástroj tsoa: https://github.com/lukeautry/tsoa

[6] Specifikace OpenAPI https://swagger.io/specification/

Další výhodou této kombinace je možnost použití takzvaných *middlewares*, které umožňují vykonávat logiku mimo kód obsluhující konkrétní koncový bod. Z praktického pohledu je možné middlewares srovnat například s aspekty v jiných programovacích jazycích. Při implementaci se nabízí jich využít například pro kontrolu autentizačního tokenu. Tímto způsobem je možné vyřešit i ukládání historie použití.

Vstupní požadavek je obsloužen nejdříve pomocí middleware, provede záznam vstupního požadavku, poté následuje je z dostupných koncových bodů na základě požadavku zvolena odpovídající obslužná funkce pomocí směrovače. Řízení programu je nejdříve předáno middleware pro kontrolu a sanitizaci vstupních požadavků. Pokud všechny operace uspějí vykoná se obslužná funkce. Tento postup je zobrazen na obrázku 8.3.



Obrázek 8.3: Postup vyhodnocení požadavku.

I přes to, že data historie nepřipomínají data ze senzorů, tvoří svým způsobem časové série, které je praktické je uložit do systému InfluxDB. Toto použití není pro tento databázový systém netradiční. K podobnému účelu jej využívají i jeho tvůrci[7].

Kvůli limitacím ukládání je praktické pozměnit formát ukládaných dat. V některých případech může být ukládaná struktura vysoce zanořená, tato situace nastává například v momentu, kdy selže obshluha koncového bodu. Z pohledu následující analýzy chyby může být praktické uložit celý vstupní požadavek a další informace.

Pro následné vyhledávání jsou zanořené struktury problematické. Databázový systém jako takový nepodporuje jejich ukládání a je nutné je buď převést na strukturu, která není zanořená, nebo uložit data určená k vyhledávání duplicitně. Vytvořená vrstva poskytuje obě dvě možnosti, ale při implementaci byla využita možnost druhá. Tato možnost sice duplikuje data, ale je jednoduší ji integrovat do již existujícího systému.

Pro každou logickou část je v aplikaci vytvořen řadič, který tuto část zpravuje. Díky nástroji tsoa je částečně vyřešen i problém validace vstupních dat, jelikož jeho správné využití zajistí alespoň částečnou validaci a sanitizaci těchto dat na základě použitých typových definic jazyka TypeScript. Složitější validaci je nutné vykonat vlastní logikou například v již zmíněných middlewares.

Jedním z požadavků při vytváření aplikace byla dokumentace poskytovaného API. Jako řešení byla zvolena již zmíněná specifikace OpenAPI, kterou je možné generovat z kombinace dekorátorů a komentářů u použitých datových typů a vytvořených koncových bodů.

Implementace tohoto požadavku se nakonec projevila jako velice výhodná. Ze vzniklé OpenAPI specifikace je možné generovat artefakty pro použití ve frontendové aplikaci. Pro toto generování byl zvolen nástroj *ng-openapi-gen*[8], který generuje jak ekvivalenty

[7]Využití InfluxDB pro monitorování https://www.influxdata.com/blog/how-we-use-influxdb-for-security-monitoring/

[8]Nástroj ng-openapi-gen https://github.com/cyclosproject/ng-openapi-gen

použitých datových typů tak Angular služby. Díky tomu je možné odstínit frontend od implementačních detailů samotného API bez nutnosti ručně udržovat tuto vrstvu.

8.2.2 Cloud AWS

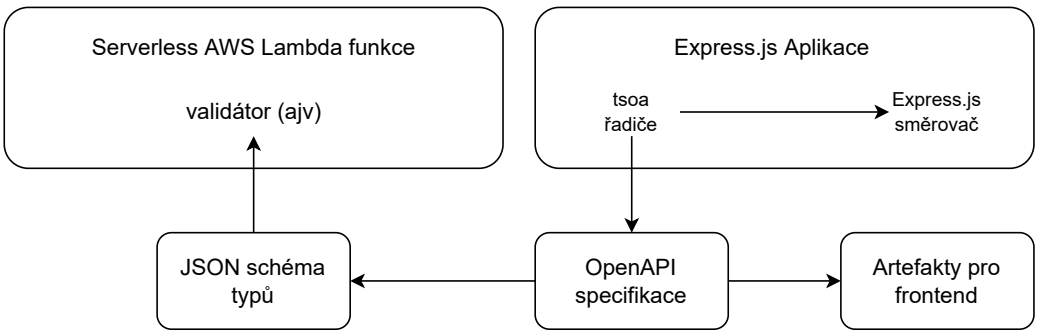
Na straně cloudu je situace podobná. Bohužel zde není možné použít nástroj tsoa a musí se proto zvolit vhodné alternativní řešení. I přes to, že je možné vytvářet a nasazovat jednotlivé Lambda funkce přímo, často bývá lepší využít nástrojů, které proces zjednodušují.

Ve firmě Logimic se používá nástroj *serverless*[9]. Tento nástroj se postará jak o nasazení, tak o zabalení více zdrojových souborů nástrojem *webpack*[10] a ostatní konfiguraci, kterou jsou například oprávnění výsledné Lambda funkce.

Při implementaci byla zvolena strategie stejného členění lambda funkcí a řadičů, kdy jednomu řadiči odpovídá jedna Labmda funkce stejnojmenných metod a funkcí. Díky tomu se velmi usnadňuje celkový vývoj, protože je možné se rychle zorientovat v obou projektech.

Aby bylo možné využívat vygenerované API artefakty i pro cloudové řešení, je nutné synchronizovat cesty koncových bodů a jejich chování. Synchronizaci chování je z velké části možné provést vytvářením sdílených a nezávislých služeb. V případě koncových bodů je nutné ručně nastavit stejné cesty a metody.

Lepší situace je v případě validace, kde je možné velkou část vyřešit pomocí nástrojů pro validaci. Jedním z těchto nástrojů je i *ajv*[11], který poskytuje možnost provádět validaci i na základě JSON schémat[12]. I v tomto případě se ukázala již existující specifikace OpenAPI ve verzi 3 jako výhodná. A to hlavně z toho důvodu, že popsaný postup zjednodušují balíčky, které poskytují možnost provádět validaci na základě popisu OpenAPI. Celý postup generování v obrázku 8.4. Při implementaci byl nejdříve zvolen přístup ruční implementace validace pomocí balíčku ajv, který byl nahrazen balíčkem *openapi-request-validator*[13], který tento proces zjednodušuje a umožňuje provádět validaci jen na základě specifikace, bez nutnosti generovat JSON schémata. Validaci není nutné provádět tímto způsobem, mohla by být například ve sdílených službách. Tento přístup ale přináší další kód, která je nutné udržovat.



Obrázek 8.4: Plný postup generování v případě použití ajv.

[9] Domovská stránka nástroje serverless: https://www.serverless.com/
[10] Domovská stránka nástroje webpack: https://webpack.js.org/
[11] Domovská stránka nástroje ajv https://ajv.js.org/api.html
[12] Zvolená verze JSON schémat https://json-schema.org/draft-07/json-schema-release-notes.html
[13] Nástroj openapi-request-validator: https://github.com/kogosoftwarellc/open-api/tree/master/packages/openapi-request-validator

Stejně jako u generování artefaktů pro frontend je i v tomto řešení stejná nevýhoda. Pro plnou funkčnost je nutné udržovat jak lokální API, které je zdrojem generovaných JSON schémat. V případech, kdy je cílem udržovat obdobné aplikace, je nevýhoda tohoto přístupu, doplněná o přesunutí co největší části logiky do sdíleného kódu, zanedbatelná. Pro budoucí využití celého řešení je slibný vývoj balíčku tsoa, ve kterém probíhají první pokusy o implementaci rozšíření, které by dovolovalo vygenerovat zdrojové kódy pro AWS lambdy z existujících řadičů[14].

8.3 Vrstva zobrazení

Implementované pohledy a komponenty byly vytvořeny za pomocí rámce Angular. Pro další zjednodušení vývoje byly použity i nástroje *PrimeNG*[15], který obsahuje sadu základních komponentů a *PrimeFlex*[16] pro zjednodušení pozicování elementů na stránce.

Tento výběr omezil možnost využití balíčku Carbon Charts, který vyžaduje použití balíčku Bootstrap. Místo něj byla využita kombinace balíčků ngx-charts, vlastních vizualizací a rozšíření na něm založených a *ApexCharts*[17] pro vizualizace, které nejsou ngx-charts dostupné.

Implementovaná knihovna obsahuje jak již hotové dashboardy, které stačí v aplikaci integrovat, tak komponenty, ze kterých jsou vytvořeny. Díky tomu je možné využít tyto části v již existujících systémech, nebo naopak rozšířit existující pohledy o další funkcionalitu pouhým použitím poskytovaných komponent.

Při vývoji bylo myšleno na chování a ovládání vytvořených vizualizací jak na tradičním počítači, tak na mobilních zařízeních. Vytvořené pohledy se vždy snaží vypořádat s rozlišením, nebo jeho změnou. Díky tomu je možné implementované komponenty použít i v mobilních aplikacích. I z tohoto důvodu využívají všechny komponenty buď nativních elementů HTML nebo, hlavně v případě grafů, elementu SVG.

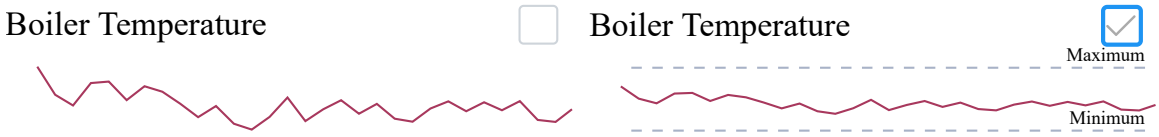
8.3.1 Implementace pohledů

Za pomocí těchto nástrojů a vygenerovaných artefaktů popsaných v předchozí kapitole byl vytvořen pohled navrhovaný v části 7.2. Již při jeho implementaci bylo jasné, že bude nutné jej rozšířit o určité funkce. Příkladem těchto rozšíření je zobrazení informací, které se často nemění, ale je nutné je nějakým způsobem při zobrazování zohlednit. Příkladem těchto dat může být aktuální verze software na zařízení, nebo datum posledního měření.

Další podstatnou částí, která v návrhu chyběla, bylo ovládání zobrazení. Nakonec bylo zvoleno jednoduché ovládání, které uživateli umožňuje určit období a položky zvoleného zařízení, které mají být zobrazeny v hlavní části dashboardu. Po zobrazení dat je také možné využít sparkline umístěné pod grafem pro podrobnější zobrazení menšího časového období.

I sparkline zobrazující trend byly rozšířeny. Nyní dovolují zobrazovat a skrývat uživatelem nastavené limity. Při zobrazení limitů může dojít k překreslení celého grafu v případech, kdy je alespoň jeden z limitů mimo právě zobrazený rozsah. Tento rozdíl je vyobrazen na obrázku 8.5.

[14] Podpora tsoa pro AWS Lambda https://github.com/lukeautry/tsoa/issues/1139
[15] Domovská stránka projektu PrimeNG: https://www.primefaces.org/primeng/
[16] Domovská stránka projektu PrimeFlex: https://www.primefaces.org/primeflex/
[17] ApexCharts https://apexcharts.com/



Obrázek 8.5: Rozdíl při zobrazení nebo skrytí limitů.

Tato vlastnost sice poněkud komplikuje původní záměr tohoto typu vizualizace, ale na druhou stranu zvyšuje srovnávací schopnosti uživatele. Tato možnost je volitelná a je dostupná pouze v případech, kdy má daný typ hodnoty nastavené limity.

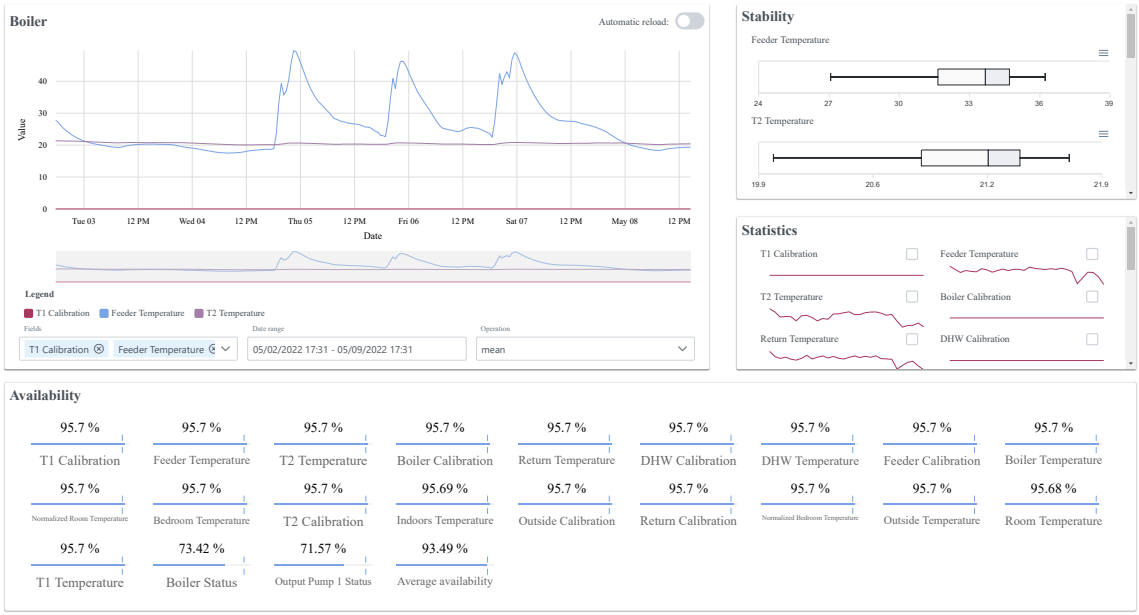
Obrázek 8.6: Statistický pohled.

Stejné zobrazení poskytuje i možnost zobrazit jiná zařízení stejného typu. Na obrázku 8.6 k tomuto výběru slouží výběrové menu s názvem „Devices“. V případě výběru druhého zařízení se pro obě zařízení zobrazí stejné metriky obou zařízení v úvodním grafu. I vzhled sparkline se změní. Při výběru dalšího zařízení se přidá druhá linie, které reprezentuje jeho hodnotu. Poslední zobrazení, které reaguje na výběr zařízení k srovnání jsou odrážkové grafy, kde je hodnota druhého zvoleného zařízení zobrazena pomocí ukazatele předchozí hodnoty. Krabicové grafy nereagují na volbu dalšího zařízení žádným způsobem.

Zařízení, která nejsou stejného typu by neměla být systémem srovnávána. Tato zařízení mohou mít například výstupy, které neměří stejné veličiny, nebo takové veličiny, které by byly závislé na veličinách právě zobrazeného zařízení. Problematické je i určit, zdali je možné srovnat dvě zařízení, které měří shodné veličiny. Například teplotu mohou měřit 2 různá zařízení s velmi rozdílnou přesností.

Druhý vytvořený pohled na obrázku 8.7 je v mnoha pohledech velmi podobný tomu prvnímu. Jeho hlavním prvkem je také spojnicový graf. Oproti předchozímu pohledu, ale nepřiřazuje velkou hodnotu statistickým zobrazením, žádným způsobem nezobrazuje statické, nebo ne často měnící se data a poskytuje omezenější ovládání. Na druhou stranu dává větší prioritu zobrazení dostupnosti zařízení.

Tento pohled byl navržen jako jednoduší verze zobrazení, kterou bude schopen využít i nezkušený uživatel, pro kterého by mohly být některé vizualizace velmi složité, nebo netradiční. Při tvorbě aplikace může být využito obou pohledů například pomocí uživatelského nastavení, kde by uživatel zvolil, který pohled chce zobrazit.



Obrázek 8.7: Zjednodušená verze statistického pohledu.

Oba dva tyto pohledy jsou více zaměřené na numerické než kategorické hodnoty. V praxi jsou ale časté případy, kdy je nutné zobrazit i kategorická data. Při spolupráci s firmou Logimic nastala přesně tato situace a zákazník vyžadoval pohled, respektive jeho část, která bude více přizpůsobená tomuto použití. V tomto případě se jednalo o zobrazení součtu počtů daných stavů v aktuálním dni, v každé hodině vybraného dne a každého dne vybraného týdne. Po konzultaci se zákazníkem byl za tímto účelem vybrán sloupcový graf, který dokáže tato data zobrazit zákazníkovi přívětivým způsobem.

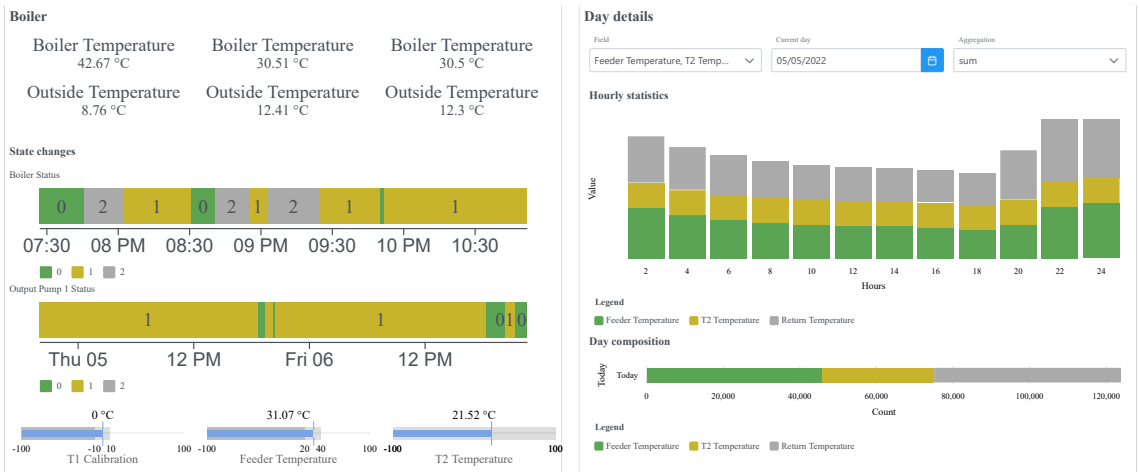
V případě součtu počtu stavů v aktuálním dni bylo hlavním účelem vizuálně zobrazit rozložení hodnot jednotlivých tříd. V původním návrhu měl k tomuto účelu sloužit kruhový diagram. V průběhu vývoje byl rychle vyměněn za horizontální sloupcový graf o jednom sloupci, který zobrazuje jak hodnoty samotných tříd, pomocí „naskládání“ dílčích sloupečků na sebe, tak celkový součet hodnot všech tříd pomocí výšky celého sloupce.

Tento přístup se postupně rozšířil na ostatní vizualizace, a to hlavně kvůli tomu, že představuje jednoduchý, efektivní a pro zákazníka přívětivý způsob zobrazení. Zbylé dvě vizualizace využívají vertikálních sloupcových grafů. V obou vizualizacích jsou zobrazeny i dny, nebo hodiny, které neobsahují žádné hodnoty. Toto rozhodnutí zlepšuje estetickou stránku vizualizace a může umožnit uživateli rychleji vyhodnotit zobrazená data.

Původní komponent pro sloupcový graf neposkytoval možnost dočasně zrušit zobrazení jednotlivých tříd. Tato funkcionalita byla přidána jednoduchým způsobem. Po kliknutím na legendu odpovídající třídy se text legendy přeškrtne a hodnota se skryje.

I přes to, že byl tento pohled původně implementován na základě požadavku cílového zákazníka bylo přidáno ovládání a možnost nastavit počáteční hodnoty vizualizací. Díky

tomu je možné vizualizaci použít i pro jiné zákazníky, kteří budou mít podobný požadavek a to bez nutnosti editovat již vytvořené komponenty. Přidané ovládání poskytuje možnost nejen nastavit období, ve kterém se mají zobrazit jednotlivé vzorky tak agregaci, která má být vykonána.



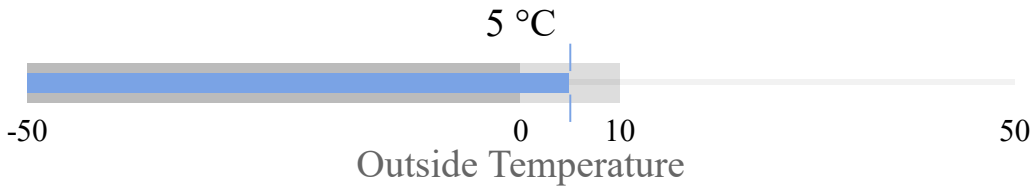
Obrázek 8.8: Část kategorického pohledu.

Kromě těchto požadavků měl výsledný pohled zobrazovat i součet počtů všech tříd ve třech časových úsecích: dnes, posledních 7 dní a posledních 30 dní. K zobrazení těchto dat nebyla využita žádná složitá vizualizace ale pouze jednoduchý text s názvem časového období a samotnou hodnotou.

8.3.2 Vytvořené komponenty

Kromě samotných pohledů byly vytvořeny i komponenty, které v nich byly využity. Některé z těchto komponentů byly založeny na již existujících komponentech a jiné na rámci D3.js.

Prvním vytvořeným komponentem byl graf typu bullet chart, který není implementován v rámci ngx-charts. Pro jeho implementaci jsem zvolil strategii rozšíření již existujícího grafu, která zaručí vysokou vizuální podobnost k vizualizacím, které rámec již poskytuje. Tento přístup je také doporučují vývojáři balíčku[18].



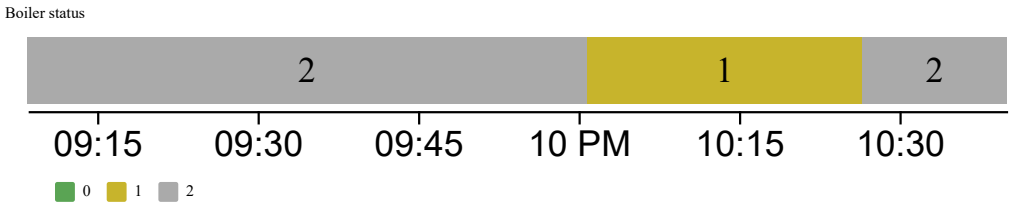
Obrázek 8.9: Komponent bullet chart.

[18]Doporučení vývojářů balíčků ngx-charts k vytváření nových vizualizací: https://github.com/swimlane/ngx-charts/blob/master/docs/custom-charts.md#pointers-when-creating-or-adapting-custom-charts

Implementace byla založena na vizualizaci, kterou vývojáři rámce nazvali lineární budík, původním názvem *linear gauge*. Tento graf je dobrým základem pro graf typu bullet, protože s ním sdílí velké množství rysů. Obě dvě vizualice jsou horizontální, zobrazují aktuální hodnotu pomocí sloupečku a druhou pomocí přerušené vertikální čáry.

V implementaci je tedy nutné přidat popisky a sloupce, které budou sloužit k zobrazení jednotlivých prahů. Původní vizualizace má také jednu velmi nepraktickou vlastnost a to tu, že se při svém zobrazení snaží vyplnit dostupný prostor automatickým zvětšením velikosti textu. Takto zvětšený text nepůsobí dobře s ohledem k nadpisům, jelikož může velikostí. Výsledný komponent poskytuje možnost zapnout původní chování, nebo nastavit očekávanou velikost textu.

Druhá vytvořená komponenta je speciální vizualizace určená k zobrazení změny stavu primárně kategorického atributu. Každému období se stejnou hodnotou atributu ve vizualizaci odpovídá obdélník, jehož šířka odpovídá procentuálnímu podílu tohoto období na celkovém zvoleném období. V případě, že je obdélník dostatečně veliký je v něm zobrazena aktuální hodnota. Při jeho zmenšování dochází ke snížení počtu zobrazených písmen. Každé zobrazené hodnotě je také přiřazena barva.



Obrázek 8.10: Příklad vizualizace.

Obrázek 8.10 obsahuje příklad vytvořené vizualizace. V tomto případě se jedná o hodnotu senzoru, která nabývá diskrétních numerických hodnot 1 – 3. Ze zobrazení je zřejmé, že se hodnota senzoru změnila v zobrazeném období dvakrát. Zjistit přesný čas změny je možné pomocí popisku, který se objeví po najetí kurzorem na odpovídající část.

Tato kombinace dovoluje uživateli rychle vyhodnotit stav zařízení. Toto zobrazení ale není ideální pro použití v případech, kdy se hodnoty rychle mění na relativně dlouhém časovém období. Krátké změny mohou být zobrazeny příliš úzkými obdélníky, které uživatel neuvidí.

Tento komponent byl plně implementován pomocí D3.js. Díky tomu je její velikost minimální a je možné ji použít v jakémkoli projektu prakticky bez přidání dalších závislostí. Při implementaci byl kladen důraz i na responzivitu samotné komponenty. Vizualizace se proto snaží využít co nejvíce prostoru, ve kterém se nachází.

Zbylé vytvořené komponenty představují spíše rozšíření komponent z již existujících balíčků takovým způsobem, aby byla zjednodušena jejich integrace do nových systémů. Některé z nich poskytují zvýšení abstrakce, zatímco jiné přidávají funkcionalitu, která v originálních komponentech chyběla.

Kapitola 9

Testování

Implementované části je vždy nutné nějakým způsobem otestovat. V případě vrstvy, která pracuje s databází a API, je testování relativně jednoduché. V těchto případech je možné využít automatizovaného testování v podobě nástrojů *jest*, *mocha*, *karma* a dalších. Jistým ukazatelem může být i srovnání nákladů na provoz ať už celého systému, nebo zvoleného databázového systému. Situace je výrazně náročnější u testování jednotlivých pohledů a vizualizací. U grafických prvků je nutné zajistit nejen jejich správnou funkčnost, ale také to, zdali odpovídají požadavkům uživatele a jestli je jejich použití dostatečně příjemné pro uživatele.

Na základě tohoto je rozdělena i tato kapitola, která nejdříve prezentuje výsledky testování backendových částí a částí frontendu, které nejsou přímo závislé na pocitech uživatele. Druhá část obsahuje provedené testování s uživateli a ostatní neautomatizované testování.

Díky přístupům, které byly použity, je výsledné řešení i dobře dokumentované, a to ať už zmiňovaným OpenAPI popisem, tak dokumentačními komentáři Javadoc. Díky tomu je usnadněno využívání a udržování vytvořených výstupů. Implementované řešení obsahuje i soubory *docker-compose*[1], což zjednodušuje případné nasazení vytvořeného řešení.

9.1 Automatické testování

Pro testování backendových částí byl použit balíček *jest*[2] v kombinaci s nástrojem *supertest*[3], který zjednodušuje testování HTTP. Jelikož se jedná o oddělené projekty každý z nich obsahuje vlastní testovou sadu. K testování frontendového kódu byl využit balíček *karma*[4], který je integrovaný do zvoleného rámce Angular.

Vrstva zjednodušující přístup k databázovému systému InfluxDB obsahuje jednotkové testy. Kvůli tomu, že je na této vrstvě založen prakticky celý systém, bylo nutné pro testování této části vynaložit nejvyšší úsilí. Testy z velké části využívají možnosti automatického vytvoření testovacích alternativ k použitým závislostem. Díky tomu je možné provádět jednotkové testy třídy bez nutnosti udržovat vlastní adaptér, který by toto testování ulehčil.

V případě API bylo primárně testováno korektní zavolání metod služby, která obaluje funkce poskytované předchozí vrstvou. Pokud bude toto zavolání správné, nemělo by dojít ani k chybám v případě integrace. Validace a sanitizace vstupů byla testována pouze

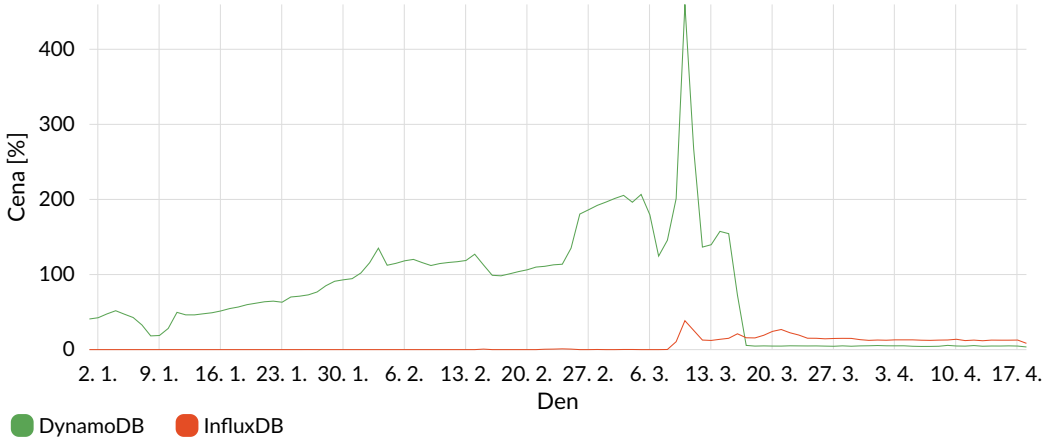
[1] Nástroj docker: https://www.docker.com/
[2] Domovská stránka balíčku jest https://jestjs.io/
[3] Domovská stránka balíčku supertest https://github.com/visionmedia/supertest
[4] Domovská stránka balíčku karma: https://karma-runner.github.io/latest/index.html

v případech, kdy bylo nutné využít vlastních funkcí. Testování zbylých případů není nutné, jelikož je možné využít existujících testů rámce tsoa.

Testy je možné spouštět příkazem *jest*, který spustí všechny testy konkrétní části projektu. Pro testování celého projektu je možné využít npm cílů definovaných v kořenovém adresáři celého monorepozitáře. Mimo jiné bylo prováděno i testování výkonu vybraných databázových systémů, které bylo provedeno v úvodu implementace samotného řešení a je popsáno v sekci 3.

9.2 Chování implementovaného řešení v praxi

Důležitou částí, která přímo nezahrnuje uživatele, bylo testování chování celého systému v reálných situacích. K tomuto testování mohlo dojít díky spolupráci s firmou Logimic, která poskytla informace o ceně provozu části systému s ohledem na využité databázové systémy. Díky těmto datům je možné srovnat cenu původně využívaného databázového systému s novým a vyhodnotit, zdali byl tento přechod zajímavý i z finančního hlediska.



Obrázek 9.1: Srovnání ceny systémů InfluxDB a DynamoDB.

Na obrázku 9.1 jde vidět cena používaných databázových systémů jak před, tak po přechodu z DynamoDB na InfluxDB. Vidíme, že celková cena za databázový systém DynamoDB postupně rostla, a to takovým způsobem, že cena na začátku března je přibližně 3x vyšší než cena na začátku ledna. Počet připojených zařízení v této době drasticky nerostl, takže příčinou zvýšení ceny je zvyšující se objem uložených dat, které je nutné projít pro nalezení potřebných záznamů. Přechod z databázového systému DynamoDB na InfluxDB začal 8. března, kdy u obou systémů je možné pozorovat navýšení ceny způsobené kopírováním části uložených dat. Nejvíce dat bylo kopírováno 10. března, kdy cena na provoz systému DynamoDB přesáhla čtyřnásobek průměrné ceny z období před přechodem. Po přechodu se cena provozu systému InfluxDB pohybuje okolo 15% průměrné ceny za provoz systému DynamoDB.

Tato data nasvědčují velmi výraznému zlevnění použitého databázového systému. Tato změna měla dopad i na ostatní backendové rutiny, které jsou díky kratšímu trvání jednotlivých operací rychlejší. Podobný dopad má změna i na frontendovou část aplikace, kde je nyní možné rychleji zobrazit větší množství dat než před přechodem.

Výsledky reflektují jen poměrně krátké časové období. V případě, delšího časového období mohou růst náklady díky zpoplatnění množství uložených dat. Tento fakt je možné částečně vyřešit využitím InfluxDB tasks pro snížení počtu vzorků v daném období. Starší vzorky mohou být například agregovány do menšího počtu, tím se sice sníží přesnost celkového měření, ale také počet vzorků, které je nutné uschovávat. Toto řešení ale není možné u všech zařízení, někdy může i zákazník požadovat uschování původních dat.

9.3 Testování s uživateli

Další důležitou částí je již zmíněné testování s uživateli. Firma Logimic již ve svém systému nějakým způsobem zobrazovala statistiky. Pro otestování nově vytvořených komponent bylo nutné je integrovat do existujícího systému a vyzkoušet s koncovými uživateli.

Pokud se náhrada, nebo návrh nové vizualizace, povedla a uživatel byl spokojený, byla vizualizace použita. Tímto procesem prošla většina vizualizací a také statistický dashboard a část kategorického dashboardu. Statistický dashboard na obrázku 8.6 je dokonce novinkou v nové aplikaci, na které firma pracuje.

Příkladem zpracované zpětné vazby může být rozšíření sloupcového grafu. V jedné části dashboardu byly využity dvě vizualizace vedle sebe. Knihovna ngx-charts se snaží a automatickou rotaci popisků v případech, kdy jsou popisky delší než sloupce, kterým odpovídají. V situacích s menším rozlišením docházelo k tomu, že sloupce jedné vizualizace byly kvůli této vlastnosti výrazně menší než druhé.

Další změnou, která měla pozitivní dopad na uživatele, byla změna databázového systému. Se změnou databázového systému se snížila i doba načítání jednotlivých dashboardů. Odpovídající data bohužel nebyla při přechodu zaznamenána.

Kapitola 10

Závěr

V dnešní době se stále zvyšuje počet chytrých zařízení a tím i množství dat, které je nutné získat, předzpracovat, uložit vyhodnotit a zobrazit uživateli. Tato diplomová práce se zaměřila hlavně na způsoby efektivního uložení a zobrazení získaných dat z tradičních časových sérií. Cílem bylo vytvořit knihovnu komponent schopných přívětivě graficky zobrazit data z chytrých zařízení spolu s ukázkovou aplikací a aplikaci schopnou data ze senzorů efektivně ukládat.

V průběhu řešení této práce bylo otestováno jedno z možných řešení tohoto problému složené z frontendové aplikace založené na aplikačním rámci Angular, backendové aplikaci reprezentující API pro práci s daty založené na rámci Express.js a AWS Lambda funkcí odpovídající části API pracující s databází InfluxDB. Díky úvodnímu testování a datům poskytnutým na konci práce je možné vyhodnotit pozitivní dopady nejen na koncového uživatele, který bude aplikaci využívat, ale také na cenu celkového provozu tohoto a podobných systémů. Tento fakt lze nejlépe pozorovat na ceně použitého databázového systému, která přibližně klesla na 15 % ceny původního databázového systému.

Většina výstupů této práce je nyní postupně adaptována do existujícího systému, ve kterém řeší problémy, které jsou pro jeho celkové fungování kritické. Použité způsoby nejen vyhovují aktuálním potřebám systému, ale jsou připraveny a podporují jeho další růst.

I v tomto systému jsou ale části, které by mohly být rozšířeny další prací. Většina senzorů, které jsou nyní v systému firmy, reprezentují měřenou hodnotu numericky. Ve světě IoT zařízení je ale mnoho případů, kdy toto nestačí.

Jedním z těchto případů jsou senzory, které produkují řetězcové hodnoty, které se vytvořené komponenty snaží zobrazit, ale oproti komponentům použitým v případě numerických komponent stále vykazují značné nedostatky.

Dalším směrem, kterým by bylo možné práci rozšířit jsou senzory, které nestačí vzorkovat jednou za určité období, ale je nutné vyhodnocovat teoreticky potencionálně nekonečné proudy dat. U zařízení, která své hodnoty reprezentují numericky, by bylo možné využít navržený systém s určitými úpravami. Horší situace nastává v případě, kdy jsou vstupní data složitá, jak tomu je například u webových kamer.

Poslední možným rozšířením vytvořených vizualizací by mohlo být odstranění závislostí tohoto řešení na rámcích PrimeNG a Angular. V případě PrimeNG by bylo toto rozdělení poměrně jednoduché. Použité komponenty je možné vytvořit ručně, nebo dodat pouze šablony, do kterých by musel koncový uživatel předat své ekvivalenty. V případě rámce Angular by byla tato změna razantnější a to hlavně kvůli použití balíčku ngx-charts, který nemá svoji alternativu pro použití bez rámce Angular.

Na straně backendu připadá v úvahu rozšířit navrhovaný systém tak, aby byly omezeny časté aktualizace dat v relačním databázovém systému. Při velkém počtu dat ze senzorů může okamžitá aktualizace vytvořit z relačního databázového systému úzké hrdlo, které omezí celý systém. Za tímto účelem by bylo možné vyměnit i na tomto místě relační databázi za vhodný NoSQL typ, nebo provádět aktualizaci záznamů jen jednou za určité období. Oba tyto přístupy přináší své výhody i nevýhody, které by bylo nutné pečlivě zvážit při dalším vývoji aplikace.

Literatura

[1] *State of IOT 2021: Number of connected IOT devices growing 9% to 12.3 billion globally, cellular IOT now surpassing 2 Billion*. Sep 2021. Dostupné z: https://iot-analytics.com/number-connected-iot-devices/.

[2] Al Sarawi, S., Anbar, M., Alieyan, K. a Alzubaidi, M. Internet of Things (IoT) communication protocols: Review. In: *2017 8th International Conference on Information Technology (ICIT)*. 2017, s. 685–690. DOI: 10.1109/ICITECH.2017.8079928.

[3] Baddeley, A. D. Working Memory, Thought, and Action. In:. 2007.

[4] Bala, R., Gill, B., Smith, D., Ji, K. a Wright, D. *Magic Quadrant for Cloud Infrastructure and Platform Services*. Červen 2021. Dostupné z: https://www.gartner.com/doc/reprints?id=1-271OE4VR&ct=210802&st=sb.

[5] Baluch, P. a Gonzales, A. *How do we see?* Jul 2015. Dostupné z: https://askabiologist.asu.edu/explore/how-do-we-see.

[6] Bedi, G., Venayagamoorthy, G. K. a Singh, R. Internet of Things (IoT) sensors for smart home electric energy usage management. In: *2016 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)*. 2016, s. 1–6. DOI: 10.1109/ICIAFS.2016.7946568.

[7] Birman, K., Freedman, D., Huang, Q. a Dowell, P. Overcoming CAP with Consistent Soft-State Replication. *IEEE Computer*. Únor 2012, sv. 45, s. 50–58. DOI: 10.1109/MC.2011.387.

[8] Black, A., Luna, P., Lund, O. a Walker, S. *Information Design: Research and Practice*. Taylor & Francis, 2017. ISBN 9781317125297.

[9] Boualouache, A. E., Nouali, O., Moussaoui, S. a Derder, A. A BLE-based data collection system for IoT. In: *2015 First International Conference on New Technologies of Information and Communication (NTIC)*. 2015, s. 1–5. DOI: 10.1109/NTIC.2015.7368748.

[10] Codd, E. F. A Relational Model of Data for Large Shared Data Banks. In: Broy, M. a Denert, E., ed. *Software Pioneers: Contributions to Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, s. 263–294. DOI: 10.1007/978-3-642-59412-0_16. ISBN 978-3-642-59412-0. Dostupné z: https://doi.org/10.1007/978-3-642-59412-0_16.

[11] Coskun, V., Ok, K. a Ozdenizci, B. *Near Field Communication (NFC): From Theory to Practice*. Wiley, 2011. ISBN 9781119966906.

[12] Dean, J. a Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*. New York, NY, USA: Association for Computing Machinery. jan 2008, sv. 51, č. 1, s. 107–113. DOI: 10.1145/1327452.1327492. ISSN 0001-0782. Dostupné z: https://doi.org/10.1145/1327452.1327492.

[13] Dinculeană, D. a Cheng, X. Vulnerabilities and limitations of MQTT protocol used between IoT devices. *Applied Sciences*. Multidisciplinary Digital Publishing Institute. 2019, sv. 9, č. 5, s. 848.

[14] Eckerson, W. *Performance Dashboards: Measuring, Monitoring, and Managing Your Business*. Wiley, 2010. ISBN 9780470589830.

[15] Few, S. *Dashboard Confusion. Retrieved June 7, 2018*. 2004.

[16] Few, S. *Information Dashboard Design: Displaying Data for At-a-glance Monitoring*. Analytics Press, 2013. ISBN 9781938377006.

[17] Ghosh, M., Wang, W., Holla, G. a Gupta, I. Morphus: Supporting Online Reconfigurations in Sharded NoSQL Systems. *IEEE Transactions on Emerging Topics in Computing*. 2017, sv. 5, č. 4, s. 466–479. DOI: 10.1109/TETC.2015.2498102.

[18] Gilbert, S. a Lynch, N. Perspectives on the CAP Theorem. *Computer*. 2012, sv. 45, č. 2, s. 30–36. DOI: 10.1109/MC.2011.389.

[19] Greengard, S. *The Internet of Things*. MIT Press, 2015. The MIT Press Essential Knowledge series. ISBN 9780262527736.

[20] Haerder, T. a Reuter, A. Principles of Transaction-Oriented Database Recovery. *ACM Comput. Surv.* New York, NY, USA: Association for Computing Machinery. dec 1983, sv. 15, č. 4, s. 287–317. DOI: 10.1145/289.291. ISSN 0360-0300. Dostupné z: https://doi.org/10.1145/289.291.

[21] Haxhibeqiri, J., De Poorter, E., Moerman, I. a Hoebeke, J. A Survey of LoRaWAN for IoT: From Technology to Application. *Sensors*. 2018, sv. 18, č. 11. DOI: 10.3390/s18113995. ISSN 1424-8220. Dostupné z: https://www.mdpi.com/1424-8220/18/11/3995.

[22] Heidema, Y., Vanderfeesten, I., Erasmus, J., Keulen, R. a Dizy, K. *Visualizing performance indicators for production planning in BPMN 2.0 in the manufacturing domain*. 2018. Disertační práce. Master's thesis, Eindhoven University of Technology.

[23] Hemment, D., Woods, M., Appadoo, V. a Bui, L. Community Key Performance Indicators (Community KPIs) for the IoT and Smart Cities. *Future Everything: Manchester, UK*. 2016.

[24] Holst, A. *IOT connected devices worldwide 2019-2030*. Oct 2021. Dostupné z: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[25] Jaluta, I., Sippu, S. a Soisalon Soininen, E. Recoverable B+-trees in centralized database management systems. *Internat. J. of Applied Science & Computations*. 2003, sv. 10, č. 3, s. 160–181.

[26] Kaur, K. a Sachdeva, M. Performance evaluation of NewSQL databases. In: *2017 International Conference on Inventive Systems and Control (ICISC)*. 2017, s. 1–5. DOI: 10.1109/ICISC.2017.8068585.

[27] Khasawneh, T. N., AL Sahlee, M. H. a Safia, A. A. SQL, NewSQL, and NOSQL Databases: A Comparative Survey. In: *2020 11th International Conference on Information and Communication Systems (ICICS)*. 2020, s. 013–021. DOI: 10.1109/ICICS49469.2020.239513.

[28] Köhler, H. a Link, S. SQL schema design: foundations, normal forms, and normalization. *Information Systems*. 2018, sv. 76, s. 88–113. DOI: https://doi.org/10.1016/j.is.2018.04.001. ISSN 0306-4379. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0306437917305069.

[29] Laptev, N., Amizadeh, S. a Flint, I. Generic and scalable framework for automated time-series anomaly detection. In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, s. 1939–1947.

[30] Marghescu, D. Multidimensional data visualization techniques for financial performance data: A review. *TUCS Technical Report*. 2007, sv. 810.

[31] Marr, B. *Key Performance Indicators (KPI): The 75 measures every manager needs to know*. Pearson Education Limited, 2012. Financial Times Series. ISBN 9780273750383.

[32] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S. et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*. JMLR. org. 2016, sv. 17, č. 1, s. 1235–1241.

[33] Montenegro, G., Schumacher, C. a Kushalnagar, N. *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals* [RFC 4919]. RFC Editor, srpen 2007. DOI: 10.17487/RFC4919. Dostupné z: https://rfc-editor.org/rfc/rfc4919.txt.

[34] Nesbitt, K. a Friedrich, C. Applying Gestalt principles to animated visualizations of network data. In: *Proceedings Sixth International Conference on Information Visualisation*. 2002, s. 737–743. DOI: 10.1109/IV.2002.1028859.

[35] Ren, H., Xu, B., Wang, Y., Yi, C., Huang, C. et al. Time-series anomaly detection service at microsoft. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, s. 3009–3017.

[36] Rys, M. Scalable SQL: How Do Large-Scale Sites and Applications Remain SQL-Based? *Queue*. New York, NY, USA: Association for Computing Machinery. apr 2011, sv. 9, č. 4, s. 30–37. DOI: 10.1145/1966989.1971597. ISSN 1542-7730. Dostupné z: https://doi.org/10.1145/1966989.1971597.

[37] Sehrawat, D. a Gill, N. S. Smart Sensors: Analysis of Different Types of IoT Sensors. In: *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. 2019, s. 523–528. DOI: 10.1109/ICOEI.2019.8862778.

[38] Soni, D. a Makwana, A. A survey on mqtt: a protocol of internet of things (iot). In: *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)*. 2017, sv. 20.

[39] Stopford, B. *Log structured merge trees*. Feb 2015. Dostupné z: http://www.benstopford.com/2015/02/14/log-structured-merge-trees/.

[40] Tufte, E. R. *Beautiful evidence*. Graphics Press, 2006.

[41] Uddin, M. S., Das, A. K. a Taleb, M. A. Real-time area based traffic density estimation by image processing for traffic signal control system: Bangladesh perspective. In: *2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)*. 2015, s. 1–5. DOI: 10.1109/ICEEICT.2015.7307377.

[42] Wang, F., Liu, S., Liu, P. a Bai, Y. Bridging Physical and Virtual Worlds: Complex Event Processing for RFID Data Streams. In: Ioannidis, Y., Scholl, M. H., Schmidt, J. W., Matthes, F., Hatzopoulos, M. et al., ed. *Advances in Database Technology - EDBT 2006*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, s. 588–607. ISBN 978-3-540-32961-9.

[43] Ware, C. *Information Visualization: Perception for Design*. Elsevier Science, 2012. Interactive Technologies. ISBN 9780123814654.

[44] Yokotani, T. a Sasaki, Y. Comparison with HTTP and MQTT on required network resources for IoT. In: *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*. 2016, s. 1–6. DOI: 10.1109/ICCEREC.2016.7814989.

[45] Zhu, Q., Wang, R., Chen, Q., Liu, Y. a Qin, W. Iot gateway: Bridgingwireless sensor networks into internet of things. In: Ieee. *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. 2010, s. 347–352.

Příloha A

Obsah přiloženého paměťového média

- Tento dokument ve formátu pdf.
- source/README.md – bližší popis repozitáře.
- source/backend – backendová část práce.
  - db-benchmark – testování databází.
  - InfluxDataBase – vrstva zpřístupňující databázi InfluxDB.
  - express – lokální API.
  - influx-lambda – AWS API.
- source/frontend – Angular aplikace.
  - src/app/library – vytvořená knihovna pohledů a vizualizací.