VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

NÁVRH BACK-ENDU PRO ANALYTICKÝ DASHBOARD POZIČNÍHO SYSTÉMU
BACK-END DESIGN FOR ANALYTICAL DASHBOARD OF POSITIONING SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MAREK HRIVŇÁK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VÁCLAV ŠIMEK**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2021/2022

Zadání diplomové práce

Student: **Hrivňák Marek, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Vývoj aplikací
Název: **Návrh back-endu pro analytický dashboard pozičního systému**
Back-End Design for Analytical Dashboard of Positioning System
Kategorie: Uživatelská rozhraní
Zadání:

1. Prostudujte lokační systém od společnosti SEWIO, který využívá technologii UWB (širokopásmová komunikace). Pozornost zaměřte především na část dashboard, databázovou infrastrukturu a uživatelské API rozhraní.
2. Prostudujte aktuálně používaný formát pro reprezentaci pozičních dat a způsoby jejich ukládání v rámci lokačního systému. Identifikujte nedostatky stávajícího řešení a pokuste se navrhnout možnosti vylepšení.
3. S ohledem na poznatky zjištěné v bodech 1) a 2) zadání navrhněte rozšíření stávající back-end části lokačního systému tak, aby bylo možno zobrazit poziční data v časovém horizontu alespoň jednoho měsíce.
4. Implementujte navržené úpravy z bodu 3) zadání. Dbejte na jejich náležitou integraci do infrastruktury lokačního systému společnosti SEWIO.
5. Podrobně analyzujte vlastnosti výsledného řešení na množině historických i nově získaných dat z reálného provozu lokačního systému.
6. Zhodnoťte dosažené výsledky a pokuste se navrhnout možnosti dalšího vývoje projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Šimek Václav, Ing.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 18. května 2022
Datum schválení: 29. října 2021

Abstrakt

Táto práca sa zaoberá optimalizáciou pôvodného riešenia a navrhnutím nového riešenia pre ukladanie a spracovanie pozičných dát nástroja Sage Analytics systému RTLS vytvoreného spoločnosťou Sewio Networks s.r.o. Cieľom tejto práce je nájdenie a aplikácia spôsobu akým je možné skrátiť čas vyhotovenia metrík nástroja Sage Analytics. Optimalizáciou pôvodného riešenia v tejto práci sa podarilo jednoduchým spôsobom (bez nutnosti úpravy implementácie) výrazne zrýchliť proces zhotovovania metrík (pre časový interval 24 hodín zobrazovaných dát) v priemere až o 503 %. Navrhnuté riešenie využíva k ukladaniu pozičných dát databázu InfluxDB a upravuje spracovanie dát takým spôsobom, aby bolo kompatibilné s využitím nového spôsobu ukladania a získavania pozičných dát. Súčasťou nového riešenia je aj vytvorenie testov pre kontrolu správnosti navrhnutého riešenia. Aplikovanie tohto riešenia prináša zrýchlenie pri procese zhotovenia metrík (pre časový interval 24 hodín zobrazovaných dát) od 725 % až po 2085 % a v priemere až okolo 1010 %. Súčasťou práce je aj vykonanie niekoľkých experimentov, ktoré majú za cieľ priblížiť dôvody dĺžky trvania metrík v nástroji Sage Analytics.

Abstract

This thesis deals with the optimization of the original solution and the design of a new solution for the storage and processing of positional data for the tool Sage Analytics of RTLS developed by Sewio Networks s.r.o. The objective of this study is to find and implement a solution to reduce the production time of Sage Analytics metrics. The optimized original solution provides in a very simple way (without the need to modify the implementation) significant acceleration in the production of metrics (for a time interval of 24 hours of displayed data) on average by up to 503 %. The proposed solution uses the InfluxDB database to store positional data and modifies the data processing in such a way, that it is compatible with the use of a new method of storing and retrieving positional data. The new solution also includes tests to check the correctness of the proposed solution. The application of this solution brings acceleration in the production of metrics (for the time interval of 24 hours of displayed data) from 725 % up to 2085 % and on average up to about 1010 %. Part of the work is also the performance of several experiments, which aim to reveal the reasons for the duration of metrics in Sage Analytics.

Kľúčové slová

RTLS, Sewio, Sage Analytics, InfluxDB, MySQL, pozičné data, optimalizácia, zrýchlenie

Keywords

RTLS, Sewio, Sage Analytics, InfluxDB, MySQL, position data, optimization, acceleration

Citácia

HRIVŇÁK, Marek. *Návrh back-endu pro analytický dashboard pozičního systému.* Brno, 2022. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Václav Šimek

Návrh back-endu pro analytický dashboard pozičního systému

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Václava Šimka s potrebnými informáciami od Ing. Lubomíra Mráza a Ing. Tomáša Kočana zo spoločnosti Sewio Networks s.r.o. Taktiež prehlasujem, že som uviedol všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Marek Hrivňák
11. mája 2022

Poďakovanie

Rád by som sa poďakoval vedúcemu práce pánovi Ing. Václavovi Šimkovi za odborné vedenie práce. Poďakovať chcem aj Ing. Lubomírovi Mrázovi za možnosť pracovať na diplomovej práci pre spoločnosť Sewio Networks s.r.o. a Ing. Tomášovi Kočanovi za cenné rady, poznatky, pripomienky a pozitívny prístup. Zároveň ďakujem aj celej svojej rodine za podporu a mojej priateľke Laure Hiščárovej za oporu a čas strávený kontrolou práce.

Obsah

Kapitola 1

Úvod

Lokalizačné riešenia sa plynutím doby vo veľkej miere stávajú súčasťou našich životov. Pomáhajú nám, uľahčujú život, robia ho bezpečnejším a poskytujú veľké množstvo dát. Zatiaľ čo v exteriéri sa už dlhé roky využíva systém GPS, v interiéri toto riešenie dostatočné nie je. Preto bolo potrebné nájsť vhodné riešenie na jeho náhradu.

Vhodným riešením pre získavanie pozičných dát v interiéri sa zaoberá spoločnosť Sewio Networks s.r.o. Táto spoločnosť vytvára zariadenie, ktoré s využitím RTLS (Real-time Locating System) technológie získava pozičné dáta v budovách. Takýmto spôsobom je schopná lokalizovať, či už rôzne zariadenia, alebo ľudí pohybujúcich sa v danej budove. Takéto zariadenia sú schopné celý deň monitorovať situáciu v budove, a tým získavať dáta, z ktorých je potom možné vytvárať rôzne štatistiky, analýzy a prehľad o zariadeniach alebo zamestnancoch pohybujúcich sa v budove.

Cieľom tejto diplomovej práce je pozrieť sa na existujúce riešenie back-endu pre analýzu pozičných dát, respektíve na spôsob ukladania a spracovania takýchto dát. Aktuálne riešenie totiž dokáže z databázy získať a spracovať potrebné dáta len za krátky časový úsek. To z dôvodu toho, že pre dlhší časový úsek získavanie a spracovanie dát trvá veľmi dlhú dobu, a to je pre zobrazenie analýzy pre užívateľa veľmi obmedzujúce a pomalé.

Mojim cieľom bolo ako prvé zanalyzovať existujúce databázové riešenie a nájsť jeho nedostatky. Následne bolo nutné navrhnúť možné vylepšenia aktuálnej databázy, respektíve nájsť vhodnú náhradu a vytvoriť návrh takejto databázy. Po preskúmaní aktuálnej databázy a rôznych pokusoch o vylepšenie sa síce žiadosti o požadované dáta výrazne zrýchlili, no aj napriek tomu bolo vhodné pozrieť sa na možné alternatívy. Práca sa preto zameriava aj na iné, vhodné spôsoby, akými by sa dalo takýto typ dát ukladať. A to od relačných, No-SQL až post-relačné (konkrétne priestorové). Dôležitým míľnikom však bolo uvedomiť si, že hlavnou časťou ukladaných dát a súčasťou každej požiadavky na databázu je nielen pozícia zariadení, ale práve čas. Práve podľa časovej značky sa najčastejšie dáta z databázy vyberajú. To viedlo k preskúmaniu databáz časových radov (timeseries database). Po preskúmaní a vykonaní experimentov bolo jasné, že využitie práve tohto typu ukladania dát môže dopomôcť k výraznému urýchleniu získavania potrebných dát z databázy, čím by bolo možné následne vizualizovať získané dáta za rozumný čas.

Pre využitie jednej z databáz časových radov bolo potrebné navrhnúť spôsob ukladania a zároveň vyriešiť, akú časť z aktuálnej databázy bude potrebné previesť na nové riešenie, a ktorú časť je potrebné ponechať ako bola.

Navrhnuté riešenie si vyžadovalo aj implementačné zmeny, pri ktorých jednotlivé typy metrík potrebujú špecifické spracovanie. Súčasťou práce sú aj vytvorené testy, ktoré nové riešenie porovnávajú so starým, čím kontrolujú správnosť nového riešenia.

Podstatou tejto práce tak bolo nájsť vhodný spôsob uloženia a spracovania pozičných dát, tak aby došlo k vyhotoveniu metrík v akceptovateľnom čase. Výsledky navrhnutého a následne implementovaného riešenia poukazujú, či a akým výrazným spôsobom došlo k zrýchleniu vyhotovovania jednotlivých metrík, a či sa čas vyhotovenia približuje k akceptovateľnému času.

Práca je rozdelená do niekoľkých kapitol. Kapitola 2 popisuje vstup do problematiky RTLS, základné pojmy, komponenty a princípy fungovania. 3. kapitola opisuje konkrétne riešenie tohto systému spoločnosťou Sewio. V 4. kapitole je popísané existujúce riešenie a technológie v ňom využité. 5. kapitola sa venuje rôznym spôsobom optimalizácie existujúceho riešenia ako aj alternatívnym spôsobom. Návrh uloženia dát alternatívneho riešenia je popísaný v kapitole 6. Zmeny, ktoré bolo nutné implementovať pre správnu integráciu nového riešenia s RTLS studiom sú zhrnuté v kapitole 7. V poslednej kapitole sú zhrnuté dosiahnuté výsledky, vplyv nového riešenia a vykonané experimenty.

Kapitola 2

Úvod do problematiky

Táto kapitola sa zaoberá predstavením problematiky a technológie, o ktoré sa celá táto diplomová práca opiera. Opisuje základný princíp pozičného systému RTLS (Real-time Locationg System) v sekcii 2.1. Sekcia 2.2 sa zaoberá komponentami, ktoré RTLS tvoria. V poslednej sekcii 2.3 je porovnanie komunikačných technológií, na ktorých je celý systém založený.

2.1 Lokalizačná platforma RTLS

Zatiaľ čo v exteriéri sa už niekoľko rokov na lokalizáciu v reálnom čase s veľkým úspechom využíva GPS (global positioning system), v interiéri GPS zlyháva, a tak bolo potrebné nájsť vhodnú náhradu. Práve takouto náhradou je RTLS. Táto sekcia je prebratá z [14].

RTLS alebo Real-time location system (Lokalizačný systém v reálnom čase) umožňuje nájsť, sledovať, analyzovať a spracovávať informácie o tom, kde sa nachádzajú aktíva alebo ľudia. S týmto pozičným systémom sme schopní pridaním tagu (malého bezdrôtového zariadenia) lokalizovať objekty (body záujmu) v interiéri, ktoré budú takýmito tagmi disponovať. Medzi body záujmu môžeme zaradiť aktíva, zariadenia (vysokozdvižný vozík, robot, automatizovaný stroj, nástroje, počítače a podobne), alebo ho poskytnúť ľuďom (zamestnancom), čím sme schopní získavať informácie o ich pohybe a výskyte v budove.

RTLS poskytuje jeden z najefektívnejších a najdostupnejších prostriedkov na meranie a sledovanie toho, čo sa operačne deje v rámci podniku. Pozičné systémy môžu mať svoje využitie v rôznych odvetviach ako napríklad:

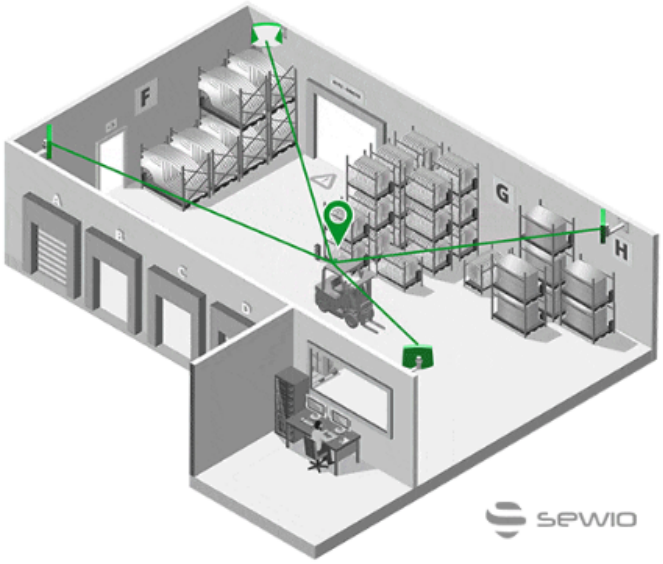
- **Zdravotníctvo**
  - **Lokalizácia zdravotníckeho personálu** – rýchla lokalizácia zdravotníckeho personálu je rozhodujúca, keď pacient alebo zamestnanec privolá pomoc počas núdzovej situácie.
  - **Sledovanie pohybu pacientov** – sledovanie fyzického pohybu pacientov pomáha zaistiť bezpečnosť pacienta. Toto je obzvlášť dôležité pre bezpečnosť a ochranu pacientov s Alzheimerovou chorobou a demenciou.
  - **Lokalizácia nástrojov** (zariadení) – sledovanie drahého alebo zdieľaného vybavenia, ako sú infúzne pumpy, šetrí čas a pomáha personálu ľahšie nájsť vybavenie. Rýchlejšie sledovanie zariadenia tiež zlepšuje kontrolu zásob, čo znižuje náklady na prenájom a nákup.

  - **Zvýšenie bezpečnosti** – personál môže čeliť násiliu zo strany pacientov či návštevníkov, a tak sa RTLS môže stať dôležitou súčasťou riešenia tým, že personálu poskytne nástroj na vyžiadanie núdzovej pomoci.

- **Výroba**
  - Koordinácia zariadení alebo zamestnancov.
  - Zvýšená miera bezpečnosti (predchádzanie kolíziám, úrazom).
  - Získavanie významných dát pre nasledovnú analýzu, ktorej výsledkom môže byť zvýšenie efektivity.
  - Zaznamenávanie pozície rôznych pracovných nástrojov čo zjednoduší ich dostupnosť a nájdenie.
  - Pre lepšiu predstavu využitia vo výrobe slúži obrázok 2.1.

- **Maloobchod**
  - **Pochopenie správania zákazníka** – pripnutím tagov na košíky, môžu predajcovia získať cenné údaje o správaní zákazníkov a použiť tieto informácie na usporiadanie tovaru.
  - **Zlepšenie služby zákazníkom** – zamestnanci predajne nosia tagy, zákazníkovi je pridelený najbližší zamestnanec, keď požiada o pomoc.
  - **Zlepšenie predaja** – priložením štítkov k produktom môžu predajcovia nastaviť automatizované procesy správy regálov, kedy môžu byť predavači v sklade okamžite informovaní, keď počet položky v regáli klesne pod určitú úroveň.

- a veľa ďalších (baníctvo, armáda, výskum a vývoj, verejná bezpečnosť, vzdelanie ...)



Obr. 2.1: Zobrazenie využitia pozičného systému RTLS vo výrobe. Jedná sa o situáciu kedy je poloha vysokozdvižného vozíka zaznamenávaná pomocou RTLS komponentov. Zdroj [24].

Popularitu a využitie tohto systému odzrkadľuje aj veľké množstvo článkov, ktoré sa danou problematikou zaoberajú, či už v zdravotníctve [3], [19], vo výrobe [29], [13], v maloobchode [9], [20] a v iných. Spomedzi všetkých zaujímavých článkov by som poukázal napríklad na:

Ho, Hanley J. a spol. sa vrámci svojho článku [10] venovali použitiu lokalizačného systému v reálnom čase na vystopovanie kontaktov zdravotníckych pracovníkov počas pandémie COVID-19 v centre pre infekčné choroby v Singapure.

Bingbing Liu a spol. sa vo svojom článku [12] venovali lokalizácií mobilných robotov, kde vytvorili prepojenú architektúru, ktorá kombinuje RTLS s dead reckoning (DR)[1].

Z odvetvia zdravotníctva vo svojom článku [4] autori Boulos, M. N. K., Berry, G. poskytujú stručný základ RTLS v zdravotníctve a popisujú mnohé možnosti, technológie a aplikácie RTLS v zdravotníckych zariadeniach, potenciálne výhody vrátane zníženia výdavkov a zlepšenia pracovného nasadenia.

2.2 Časti RTLS

Na fungovanie systému RTLS je potrebných niekoľko častí. Medzi hlavné tri časti patria: tagy, prijímač a software. Každá z týchto troch častí vykonáva špecifickú úlohu a má svoje opodstatnenie. Tieto časti spolu vytvárajú RTLS systém. Jednotlivé časti sa môžu líšiť podľa vybranej technológie (UWB/Bluetooth/Wi-Fi), avšak ich podstata a zmysel použitia ostávajú rovnaké. Veľkou výhodou je, že systémy môžu byť ľubovoľne škálovateľné, pridávaním komponentov a zväčšovaním pozičnej infraštruktúry. Táto sekcia čerpá informácie z [14].

2.2.1 Tag

Tagy sú bezdrôtové zariadenia, pripnuté k ľuďom, objektom alebo akýmkoľvek fyzickým objektom, ktoré sú určené na digitálne sledovanie ich polohy a pohybu v reálnom čase vo veľkých zariadeniach alebo oblastiach. Môžu sa medzi sebou líšiť rôznymi aspektmi ako je veľkosť, vybavenie, špecifikácie. Tie sú ovplyvnené, či už podľa objektu, na ktorom sú pripevnené resp. ktorý majú lokalizovať, alebo technológiou, ktorú využívajú ku komunikácii.

(a) Tag od firmy RTLOC [2]

(b) Tagy od firmy Sewio[3]

Obr. 2.2: Zobrazenie rôznych tagov využívajúcu technológiu UWB.

[1] dead reckoning je špeciálny spôsob vypočítania aktuálnej polohy pohybujúceho sa objektu s využitím predchádzajúcej polohy

[3] RTLS Sewio: https://www.sewio.net/

[3] RTLS RTLOC: https://uwb.woxuwireless.com/views/product/rtls.html

Typickým príkladom ich využitia môžu byť továrne, kde sa tieto zariadenia využívajú na sledovanie zamestnancov, čím sa zvýši ich bezpečnosť. Aby to bolo možné, je potrebné tagmi vybaviť aj ťažké výrobné zariadenia, kedy bude systém schopný lokalizovať všetky objekty v reálnom čase, čo umožní predísť akýmkoľvek kolíziám. To, ako tagy môžu vyzerať je možné vidieť na obrázku 2.2.

2.2.2 Pozičný senzor

Pozičné senzory sú zariadenia so stálym prísunom energie, ktoré majú známu a stabilnú polohu. Pomocou týchto zariadení sme schopní lokalizovať tagy, ktoré sú pripnuté k objektom. Zvyčajne sú umiestnené na vyšších pozíciach aby signál medzi ich komunikáciou s tagmi mal čo najmenší počet rušivých elementov. K dobrému signálu je zároveň potrebné umiestniť hneď niekoľko takých senzorov, ktoré potom vytvárajú akúsi lokalizačnú infraštruktúru, aby bol každý tag vždy na dosah. Veľkosť, vybavenie a špecifikácia pozičných senzorov sa opäť rôzne líšia, a to primárne podľa technológie, ktorú využívajú.

(a) Kotvy (anchors) od firmy Sewio

(b) Kotva (anchor) od firmy Woxu [4]

Obr. 2.3: Zobrazenie rôznych pozičných senzorov (kotiev) využívajúcu technológiu UWB.

2.2.3 Software

Rovnako dôležitou súčasťou systému je aj jeho software. Väčšinou sa jedná o aplikáciu, ktorej účelom je vhodnou formou reprezentovať dáta, ktoré sú zaznamenávané tagmi a pozičným senzorom. Zároveň slúži na správu celého systému, jeho nastavenie a konfiguráciu zariadení. Jednotlivé aplikácie systému sa od seba rôzne líšia. To aké sú komplexné a čo všetko ponúkajú záleží od ich poskytovateľa.

Súčasťou RTLS aplikácie väčšinou sú:

- vizualizácia polohy zariadení v reálnom čase,
- analýza a výpočet pohybu zariadení,
- konfigurácia systému.

Konkrétny RTLS software je opísaný v sekcii 3.2, ktorá sa venuje aplikácií s názvom *RTLS Studio* od spoločnosti Sewio a to z dôvodu, že optimalizácia jednej z časti tejto aplikácie (analytický dashboard) je cieľom tejto diplomovej práce.

[4] RTLS Woxu: https://uwb.woxuwireless.com/views/product/rtls.html

2.3 Komunikačné technológie RTLS

RTLS sa každým rokom zlepšuje a vyvíja. Spolu s ním aj technológie, ktoré tvoria základ tohto systému. Medzi takéto technológie môžeme zaradiť wifi, bluetooth, RFID alebo ultra-wideband (UWB). Jedná sa o technológie určené na bezdrôtovú komunikáciu s krátkym dosahom a nízkou spotrebou energie. Každá z týchto technológií je niečim špecifická a poskytuje určité výhody, ale aj nevýhody. Líšia sa medzi sebou mnohými aspektmi ako sú: spotreba energie, dosah, frekvencia a podobne. Všeobecne porovnanie technológií je v tabuľke 2.1. Táto sekcii vychádza z [11] a [1].

Technológia	Presnosť	Spotreba energie	Cena	Dosah [m]	Zhrnutie
UWB	cm-dm	nízka	vysoká	1-50	Najpresnejšia interiérová technológia a najlepšia pre priemyselné použitie
Wi-Fi	m	vysoká	nízka	1-50	Jednoducho aplikovateľná a vie dobre sledovať všetky zariadenia s podporou wifi
Bluetooth	m	nízka	nízka	1-20	Menej presná, ale flexibilnejšia a jednoduchšia na nasadenie
RFID	dm-m	nízka	nízka	1-50	Najlacnejšia ale zastaralá technológia s nízkou spotrebou

Tabuľka 2.1: Porovnanie jednotlivých RTLS technológií. Ako je možné vidieť, každá ma svoje výhody aj nevýhody.

Kapitola 3

Lokalizačný systém spoločnosti Sewio

Táto kapitola sa zameriava na konkrétne riešenie systému RTLS spoločnosťou Sewio. V prvej časti je popísaný špeciálny prístup získavania pozičných dát s využitím technológie UWB. Následne je v sekcii 3.2 popísaný software spoločnosti Sewio, ktorý získané pozičné dáta prevádza do vizuálnej podoby vhodnej pre lepšie porozumenie veľkého množstva dát.

3.1 Získavanie pozičných dát

Dôležitou súčasťou pozičných systémov je získavanie pozičných dát. Cieľom je, aby bol systém schopný získavať dáta na čo najväčší dosah, s čo najväčšou presnosťou bez akéhokoľvek prerušenia. Takýto ideálny scenár samozrejme nie je možný, ale snahou je sa k nemu aspoň priblížiť. Spoločnosť Sewio sa rozhodla z rôznych komunikačných technológií 2.3 pre získavanie dát využiť technológiu UWB a na nej postavať jej produkt.

3.1.1 Ultra-Wideband

UWB [18], [15], ako už vyplýva z 2.3 je bezdrôtový komunikačný protokol na krátku vzdialenosť, ktorý je s využitím vysokých frekvencií schopný poskytovať smerové a polohové dáta. Ako je možné vidieť z tabuľky **??**, výhoda, ktorú UWB ponúka oproti existujúcim technológiám je, že dokáže poskytnúť oveľa vyššiu presnosť pri väčšom dosahu.

To, že technológia UWB existuje už toľko rokov a používa sa pre širokú škálu aplikácií je silným dôkazom životaschopnosti a flexibility tejto technológie. Jej najväčší potenciál spočíva v úplne novej generácii mikrolokačných systémov. UWB prinesie vlnu aplikácií, ktoré využívajú vysoko presné snímanie polohy a vzdialenosti na poskytovanie nových možností. Práve takú možnosť vyžadujú ľudia, ale hlavne väčšie podniky (továrne, logistické centrá a podobne), ktoré chcú byť schopné nájsť takmer čokoľvek v reálnom čase, bez ohľadu na veľkosť, čo prináša množstvo benefitov.

Kľúčové vlastnosti UWB sú:

- vysoká presnosť a dosah
- má potenciálne nízku zložitosť a nízke náklady
- odolnosť voči šumu
- žiadne rušenie s inými rádiovými komunikačnými systémami

3.1.2 Meranie polohy

Na rozdiel od iných rádiových technológií UWB nevyužíva amplitúdovú alebo frekvenčnú moduláciu na zakódovanie informácií, ktoré prenášajú jeho signály. Namiesto toho využíva krátke sekvencie veľmi úzkych impulzov (iba 2 nanosekundy) pomocou kľúčovania s fázovým posunom (BPSK) a/alebo burst position modulation (BPM) na kódovanie dát. Tieto úzke pulzy majú čisté hrany, umožňujúce presné určenie času príchodu a vzdialenosť v prítomnosti viac-cestných efektov spôsobené odrazmi signálu. Tento odsek vychádza z [15].

Technológia UWB môže byť implementovaná rôznymi spôsobmi a na riešenie širokého spektra rôznych potrieb. Jedným z takýchto spôsobov je metóda Time Difference of Arrival. Metóda Time Difference of Arrival (TDoA) [2] je založená na presnom meraní časového rozdielu medzi signálmi, ktoré prichádzajú do kotiev. Aby bolo možné vzdialenosť vypočítať musia byť všetky kotvy presne synchronizované. Priebeh TDoA je znázornený na obrázku 3.1.

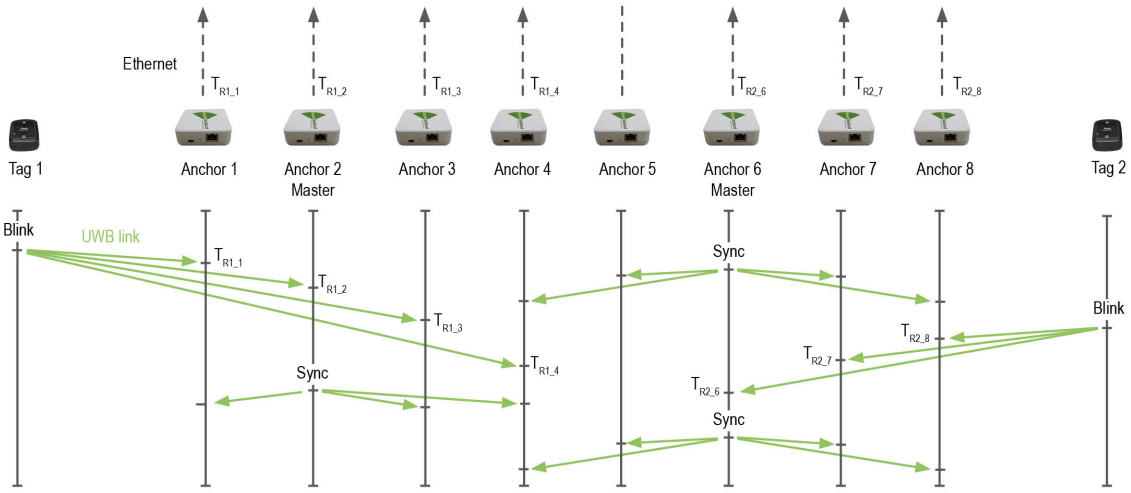
Výpočet vzdialenosti metódou TDoA ilustrujú nižšie uvedené matematické vzťahy:

$$d = c * (T_{arrival} - T_{sent}) \tag{3.1}$$

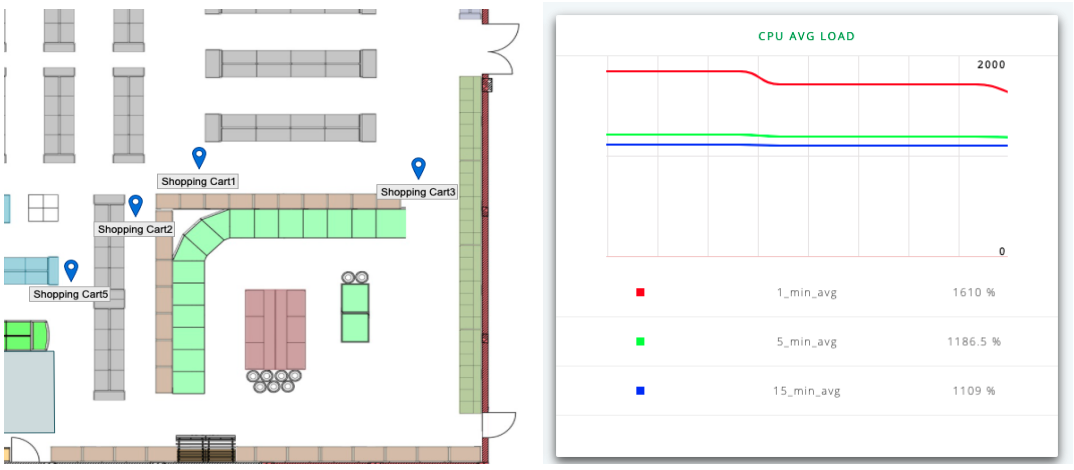
kde c je rýchlosť svetla. Pomocou tejto vzdialenosti je možné určiť množinu umiestnení cieľa. Pre dvojrozmerný priestor je to kruh s rovnicou:

$$d = \sqrt{(x_{ref} - x)^2 + (y_{ref} - y)^2} \tag{3.2}$$

kde body x_{ref} a y_{ref} tvoria známu polohu referenčného bodu. Akonáhle je táto množina vypočítaná pre dostatok referenčných bodov (aspoň tri pre dvojrozmerný priestor), presnú polohu cieľa možno vypočítať nájdením prieniku (zdroj [25]).



Obr. 3.1: Tagy vysielajú v pravidelných intervaloch krátku správu. Takáto správa je spracovaná všetkými kotvami v komunikačnom dosahu. Bez ohľadu na stav ich synchronizácie, kotvy posielajú všetky časové záznamy na server RTLS cez Ethernet. Na výpočet polohy tagu server RTLS berie do úvahy iba časové pečiatky pochádzajúce z najmenej troch kotiev s rovnakou základňou hodín. Synchronizácia je realizovaná bezdrôtovo, automaticky a to tak, že vybrané kotvy s optimálnym pokrytím nazývané Master Anchors v pravidelnom intervale vysielajú synchronizačný signál. Zdroj [2].



(a) Sensmap (b) RTLS Monitor

Obr. 3.2: Zobrazenie nástrojov RTLS studia.

3.2 RTLS studio

RTLS studio je software od spoločnosti Sewio. Tento software tvorí jednu z hlavných súčastí (popísaných v podsekcií 2.2.3) RTLS. Jedná sa o softwarové riešenie typu všetko v jednom s jednoduchou inštaláciou, nasadením, údržbou a ovládaním. Je navrhnutý tak, aby poskytoval nástroje pre presnú konfiguráciu systému, prehľady dát, vizualizáciu v reálnom čase, prehľad výkonu v reálnom čase a analýzu systému. Táto sekcia vychádza z oficiálnej dokumentácie [24]. Demo RTLS studia si je možné vyskúšať na internetovej stránke[1] tejto spoločnosti.

Súčasťou RLST studio sú tieto nástroje:

- RTLS Manager
- Sensmap
- RTLS Monitor
- Open API Documentation
- Sage Analytics

3.2.1 RTLS Manager

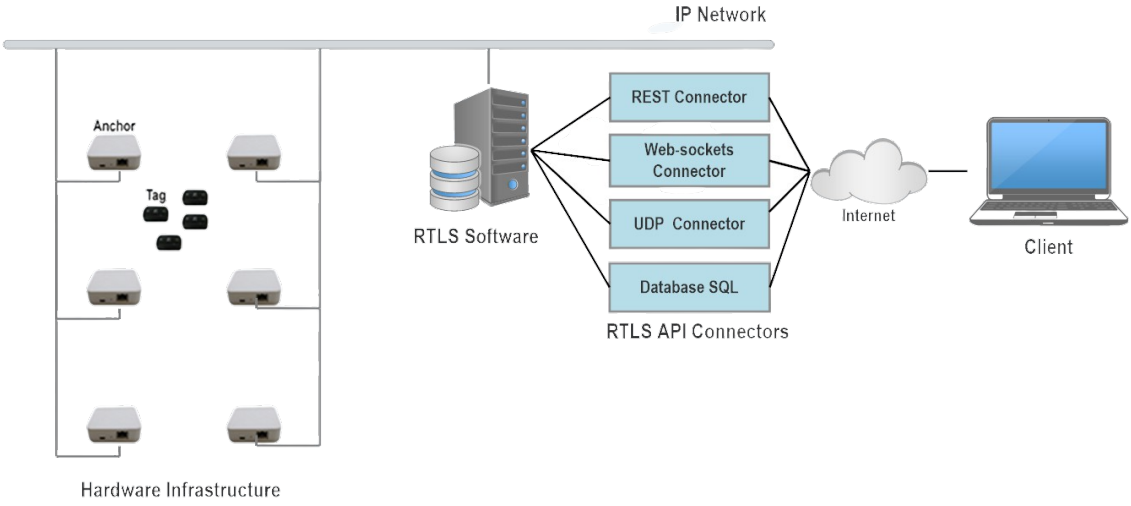
RTLS Manager je jadrom pozičného systému, ktorý umožňuje jeho jednoduché nasadenie a konfiguráciu. Zobrazuje prehľad o zdravotnom stave všetkých kotiev, tagov a serverov. Poskytuje plnú kontrolu nad nastaveniami a systémovými parametrami, ako aj aktualizáciu celého systému alebo rekonfiguráciu zariadení.

3.2.2 Sensmap

Sensmap (obrázok 3.2) je nástroj, ktorý slúži k vizualizácií všetkých procesov v reálnom čase v zariadení, v ktorom je nainštalované RTLS. Sensmap umožňuje pridávať plány budov,

[1] https://demo.sewio.net/.

automaticky rozmiestniť kotvy, pridať rôzne zóny, cesty, steny a východy, pričom už za niekoľko minút, môže užívateľ získavať prvé informácie o svojom zariadení.



Obr. 3.3: Zobrazenie celkovej architektúry RTLS s detailom pre Open API dokumentáciu. Obrázok je prevzatý z [24].

3.2.3 RTLS Monitor

RTLS Monitor (obrázok 3.2) je nástroj, s ktorým má užívateľ k dispozícií úplný prehľad o výkone systému, o aktuálnom stave siete a o užívateľoch a aplikáciách v reálnom čase. Tento prehľad je užívateľovi zobrazovaný s využitím grafov a tabuliek tak, aby šlo jednoducho porozumieť vyťaženiu systému.

3.2.4 Open API Documentation

Nástroj, ktorý popisuje API, kedy užívateľ môže bez problémov využiť integráciu tretích strán s RTLS. S využitím API je možné jednoducho integrovať lokalizáciu v reálnom čase do vlastného riešenia pomocou služieb REST, WebSocket a UDP streamu. Umiestnenie v architektúre RTLS je možné vidieť na obrázku 3.3.

3.2.5 Sage Analytics

Sage Analitics je nástroj, na analýzu, vizualizáciu a celkový prehľad systému RTLS. Pohyb jednotlivých zariadení je zaznamenávaný a následne ukladaný tak, aby ho bolo možné potom vhodným spôsobom reprezentovať užívateľovi v zrozumiteľnej a prehľadnej forme.

Jedná sa o aplikáciu, ktorá je založená na webovom klientovi napísanom v Angular 2 a webovom serveri API vytvorenom pomocou frameworku `Laravel`. Hlavnou myšlienkou návrhu Saga Analytics sú tri jednoduché otázky: „Kto? Kde? Kedy?“. Tieto 3 otázky sú základom každej požiadavky uskutočnenej v Sage Analytics.

- Kto? – označuje účastníkov systému, teda objekty, na ktorých sú upevnené tagy, a ktorých pozícia je nepretržite monitorová a zaznamenávaná.

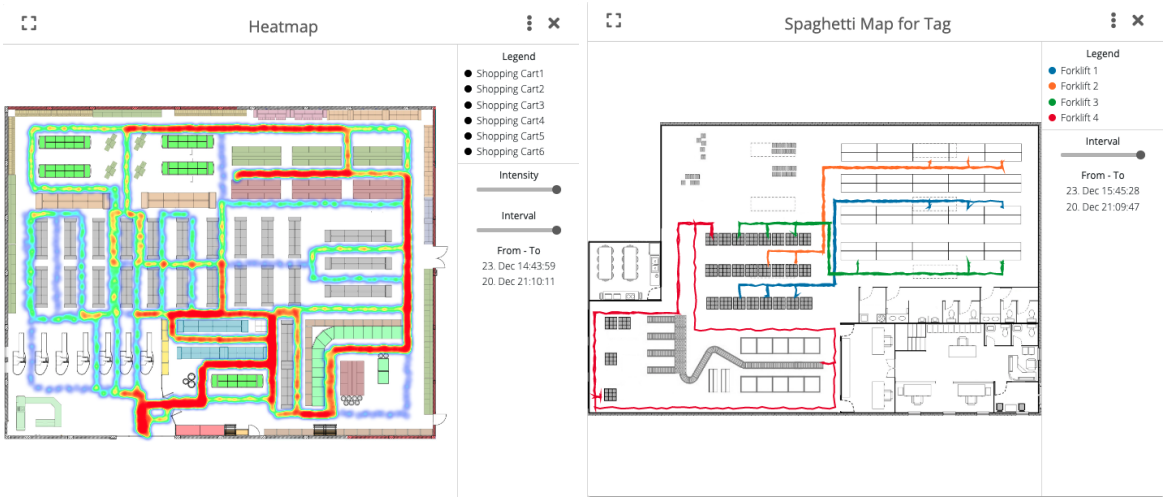
- Kde? – označuje oblasť, miesto a miestnosť, v ktorej sa objekty pohybujú, a z ktorej si užívateľ vyžiada ich zobrazenie.
- Kedy? – označuje časový interval, pre ktorý majú byť pozičné dáta analyzované a následne zobrazené.

Metriky pre zobrazenie dát

Pozičné dáta respektíve ich analýzu je možné zobraziť rôznymi spôsobmi. Každý z týchto spôsobov poskytuje jednoduchší prehľad nad požadovanými dátami.

Medzi tieto metriky patria:

- **Heatmap** – grafické zobrazenie hustoty výskytu respektíve pohybu objektov;
- **First & Last appearance in zone** – tabuľka vstupu alebo výstupu objektu do alebo z vybranej zóny;
- **Distance** – graf zobrazujúci vzdialenosť prejdenú objektom alebo skupinou objektov v určitom časovom intervale;
- **Speed** – graf zobrazujúci rýchlosť objektu alebo skupiny objektov v určitom časovom intervale;
- **Activity** – graf zobrazujúci počet lokalizačných výskytov objektu alebo skupinu objektov v určitom časovom intervale;
- **Spaghetti map** – grafické znázornenie polohy objektu (alebo skupiny objektov) v danom časovom intervale, pričom polohy sú prepojené čiarami;
- **Attendance** – graf zobrazujúci prítomnosť objektu vo vybranej zóne;
- **Zone map** – grafická reprezentácia času, ktorý objekt strávil vo vybraných zónach;
- **Contact tracing** – tabuľka zobrazujúca všetky prípady interakcie medzi objektami.



(a) Heatmap (b) Spaghetti map

Obr. 3.4: Zobrazenie nástrojov sage analytics.

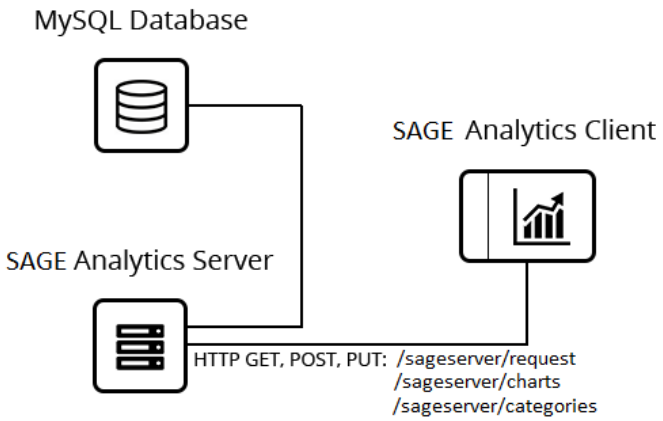
Priebeh získavania pozičných dát

Poloha zariadení, je zaznamenávaná pomocou UWB technológie. Táto poloha sa pravidelne a po častiach ukladá do MySQL databázy.

Následne, ak si užívateľ vyžiada analýzu, respektíve zobrazenie dát v nejakej z možných foriem (metrík), odošle sa požiadavka na server. Server požiadavku príjme a na základe vyžiadaných dát od klienta posiela žiadosť pre získanie potrebných dát z databázy. Po prijatí dát ich vhodným spôsobom spracuje a pripraví do vhodnej formy. Spracované dáta tak posiela ako odpoveď na požiadavku klientovi, kde prehliadač s využitím knižnice vyžiadané dáta zobrazí podľa požiadavky užívateľa. Celý takýto proces získania dát sa dá jednoduchšie predstaviť pri pohľade na obrázok 3.5.

Celý proces získavania a spracovania dát však môže trvať rôzne dlhú dobu v závislosti od toho, akú požiadavku (koľko objektov, za aký časový interval) užívateľ požaduje. V prípade krátkeho časového intervalu sú požiadavky spracované zväčša v priebehu milisekúnd až sekúnd. Problémom však je spracovanie dlhšieho časového intervalu (dni, týždne, mesiac). V tomto prípade tak žiadosti na dáta môžu trvať až príliš dlho.

Práve týmto problémom sa táto diplomová práca zaoberá a pokúša sa vyriešiť problém získavania a spracovania dát pre dlhšie časové intervaly.



Obr. 3.5: Schéma základných blokov architektúry systému pre Sage Analytics. Obrázok je prevzatý z [24].

Kapitola 4

Existujúce riešenie a použité technológie

Nástroj Sage analytics poskytuje užívateľom RTLS sofistikovaný prehľad a analýzu celého systému, a to vhodným zobrazením v grafoch, tabuľkách, či iným spôsobom. Užívatelia si majú možnosť tento prehľad zobraziť pre rôzne objekty, miestnosti, či časové intervaly. Súčasné riešenie je stabilné a dobre spracované, avšak naráža na niekoľko problémov, a to primárne z časového hľadiska. Zatiaľ čo požiadavky pre zobrazenie analýzy formou jednej z metrík (popísaných v sekcii 3.2.5) pre kratší časový interval sú uspokojivé a získané dostatočne rýchlo, situácia sa výrazne mení v prípade, že zvolený časový interval je dlhší (dni, týždne). V takomto prípade sa požiadavky spracovávajú výrazne dlhšiu dobu (nie dostatočne rýchlo).

V tejto kapitole je opísané aktuálne riešenie, spôsoby uloženia dát, ich získavanie pri zadaní žiadosti od užívateľa a následne ich spracovanie. Zároveň je otestovaná a nameraná rýchlosť pri získavaní a spracovaní požiadaviek, ktorá poslúži ako spôsob porovnania aktuálneho riešenia s riešením navrhnutým a implementovaným v tejto diplomovej práci. Popisuje výhody aj nevýhody súčasného riešenia a poukazuje na nedostatky a dôvody dlhšieho získavania a spracovania dát.

4.1 Uloženie dát

Pri pohybe objektov sa v RTLS s využitím technológie UWB a jej komponentov zaznamenáva ich aktuálna pozícia v priestore, respektíve ich pohyb po priestore (konkrétne pohyb tagov). Takto získané dáta môžu byť využité rôznym spôsobom. Napríklad reprezentáciou a analýzou v reálnom čase, alebo zaznamenávaním pozície a jej následným vhodným uložením pre historické záznamy. Takéto historické dáta slúžia priamo pre Sage analytics, a presne tieto dáta sú po zadaní požiadaviek od užívateľa získavané a vhodným spôsobom spracovávané.

Na to aby sa takéto dáta dali s jednoduchosťou ukladať, meniť, ale hlavne získavať, je potrebné ich správnym spôsobom uložiť. V dnešnej dobe sa k akémukoľvek perzistetnému ukladaniu dát, a to či už v prípade webovej alebo akejkoľvek inej aplikácie, zväčša využívajú databázy.

Databázy ako rozšírený spôsob ukladania dát rozlišujeme na viacero typov, ako sú napríklad: relačné, objektovo-orientované, NoSQL a podobne. Medzi najpoužívanejšie sa považujú relačné databázy. Práve v takejto sú uložené pozičné dáta súčasného riešenia RTLS.

4.1.1 Relačná databáza

Relačné databázy [7] uchovávajú údaje v tabuľkách a poskytujú efektívny, intuitívny a flexibilný spôsob ukladania a prístupu k štruktúrovaným informáciám. Tabuľky, známe aj ako vzťahy, pozostávajú zo stĺpcov obsahujúcich jednu alebo viacero kategórií údajov a riadkov, známych aj ako záznamy tabuľky, ktoré obsahujú množinu údajov definovanú kategóriou. Táto schopnosť umožňuje získať úplne novú tabuľku z údajov v jednej alebo vo viacerých tabuľkách pomocou jedinej žiadosti (query). Dopomáha k lepšiemu porozumeniu vzťahov medzi všetkými dostupnými dátami a získaniu nových poznatkov.

Každá tabuľka má jedinečný identifikátor alebo primárny kľúč, ktorý identifikuje informácie v tabuľke, a každý riadok obsahuje jedinečný údaj (v ideálnom prípade) pre kategórie definované stĺpcami.

Vytvárať, aktualizovať a spravovať relačné databázy dopomáha *Systém riadenia bázy dát* (DBMS – Database Management System). Väčšina systémov riadenia bázy dát používa na prístup k databáze jazyk `SQL` a mnohé sa riadia vlastnosťami databázy **ACID**:

- Atomicita – Transakcia môže byť úspešná celá alebo vôbec. Ak zlyhá jedna časť transakcie, zlyhá celá transakcia.
- Konzistencia – Zaručuje, že všetky údaje budú konzistentné respektíve, že transakcia nesmie narušiť databázovú integritu.
- Izolácia – Zaručuje, že všetky transakcie prebehnú izolovane. Žiadna transakcia nebude ovplyvnená žiadnou inou transakciou.
- Trvanlivosť – Trvanlivosť znamená, že akonáhle je transakcia potvrdená, zostane v systéme aj keď bezprostredne po transakcii dôjde k zlyhaniu systému.

Výhodou relačnej databázy je presný a formálny opis tabuľkovej štruktúry, z ktorej je možné veľmi jednoducho ukladať, vyhľadávať, filtrovať, počítať, zoskupovať a kategorizovať dáta bez potreby reorganizácie tabuliek. Medzi ďalšie výhody relačných databáz patria:

- **Škálovateľnosť** – Nové údaje môžu byť pridané rýchlo, jednoducho a nezávisle od existujúcich záznamov.
- **Jednoduchosť** – Jednoduchosť pri vykonávaní zložitých a komplexných žiadostí (queries) s využitím `SQL`.
- **Presnosť dát** – Normalizácia spôsobí elimináciu nezrovnalostí.
- **Integrita dát** – Silné typovanie dát a kontrola platnosti zaisťujú presnosť a konzistentnosť.
- **Bezpečnosť** – Údaje v tabuľkách môžu byť v rámci systému riadenia bázy dát obmedzené pre určitých užívateľov.
- **Spolupráca** – Viacerí používatelia môžu pristupovať k rovnakej databáze súčasne.

Mnoho databázových produktov implementuje model relačnej databázy. Tieto databázy siahajú od malých desktopových systémov až po masívne cloudové systémy. Medzi takéto produkty zaraďujeme napríklad: *MySQL*, *PostgreSQL*, *SQLite*, *Oracle Database* a podobne.

Relačné databázy však nepredstavujú ani zďaleka najlepšie a univerzálne riešenie. Každý problém si vyžaduje špecifické riešenie, a preto je podstatné prvotne zanalyzovať situáciu,

kde sa ma databáza používať a vyhodnotiť aký typ databázy je vhodný pre daný problém. Relačné databázy aj napriek svojej veľkej popularite a histórii majú aj svoje nevýhody a nemusia svojim využitím predstavovať vhodné riešenie.

Medzi nevýhody relačnej databázy patria:

- **Problém s údržbou** – S nárastom dát narastá aj zložitosť údržby relačnej databázy, a tým aj časová náročnosť údržby pre vývojárov.
- **Cena** – Nastavenie a údržba relačnej databázy môžu byť nákladné, a to obzvlášť pre menšie podniky.
- **Fyzické úložisko** – Požiadavky na fyzickú pamäť sa môžu zvyšovať spolu s nárastom údajov.
- **Problém škálovateľnosti** – Ako sa databáza zväčšuje alebo distribuuje na väčší počet serverov, mení sa štruktúra databázy čo má negatívne účinky, ako je latencia a problémy s dostupnosťou, ovplyvňujúce celkový výkon.
- **Zložitosť štruktúry** – Relačný model je postavený na špecifickej tabuľkovej štruktúre, tým zabezpečuje integritu, avšak v prípade flexibility (napríklad v prípade nekonzistentných údajov) by bolo vhodnejšie vybrať iný typ databázového riešenia.

4.1.2 Schéma uloženia pozičných dát

Základ pre nástroj Sage Analytics tvorí práve história pozičných dát. Pozičné dáta sa pravidelne zaznamenávajú a ukladajú v relačnej databáze *MySQL*[1]. Ako je popísané v predchádzajúcej podsekcii, tento typ databázy poskytuje množstvo výhod, ukladanie a načítanie dát prebieha veľmi jednoducho.

Databáza, ktorú využíva RTLS obsahuje niekoľko tabuliek, ktoré slúžia k ukladaniu potrebných údajov. Z hľadiska Sage Analytics sú však podstatné len niektoré z nich.

Základnou a najdôležitejšou tabuľkou z hľadiska Sage Analytics je tabuľka s názvom `position_history`. Jedná sa o tabuľku, ktorá v sebe uchováva pozičné dáta (pozícia X, Y a Z) pre určitú časovú známku. Tieto dáta sú priradené k Tagu (*feed_reference*), ktorého pozícia bola snímaná, a tak je jasné, ktorému objektu daný záznam patrí. Akékoľvek nástroje, ktoré sú súčasťou Sage Analytics vychádzajú väčšinou z tejto tabuľky, ktorá predstavuje akúsi históriu pohybu daných objektov. Záznamy sa do tejto tabuľky ukladajú pomerne často, a tak je jasné, že s postupom času sa veľkosť tabuľky enormne zväčšuje. Tabuľka `position_history` obsahuje tieto atribúty:

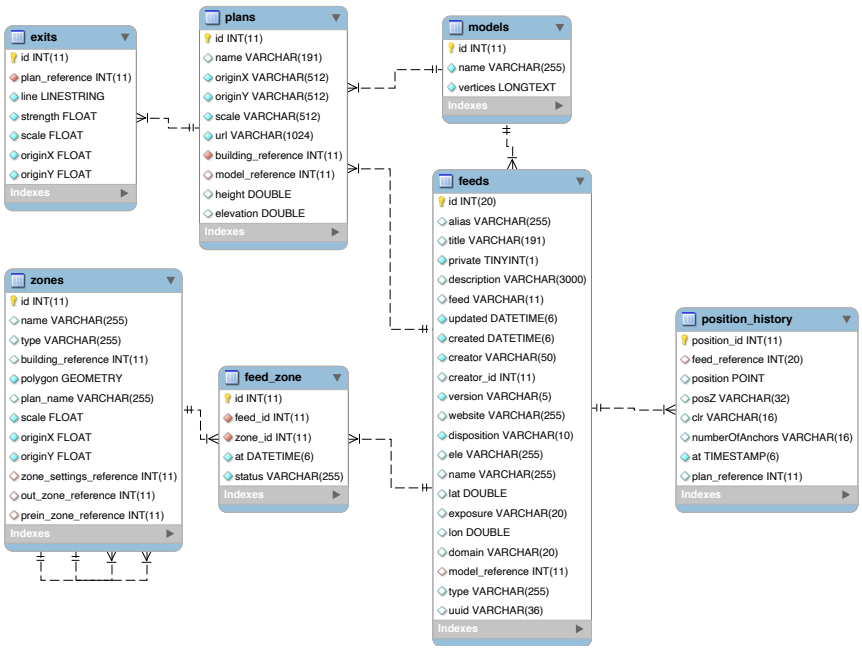
- `position_id` – primárny kľúč, identifikačné číslo záznamu;
- `feed_reference` – cudzí kľúč, predstavuje ID feed (tagu), pre ktorý bol tento záznam vytvorený, a ktorého pozícia bola v tomto čase zaznamenaná;
- `position` – hodnota súradnice X a Y, pre pozíciu objektu (na ktorý odkazuje feed_reference). Súradnice sú uložené v špeciálnom dátovom type `POINT`[2], ktorý poskytuje MySQL pre ukladanie pozície;
- `posZ` – hodnota súradnice Z, pre pozíciu objektu. Táto súradnica sa však zväčša nevyužíva;

[1] https://github.com/mysql/mysql-server
[2] https://dev.mysql.com/doc/refman/8.0/en/gis-class-point.html

- `clr` – CLR (Confidence Level Radius) je hodnota, ktorá udáva kvalitu spočítanej pozície;
- `numberOfAnchors` – počet kotiev, ktorými bol daný objekt snímaný;
- `at` – časová značka, reprezentujúca presný čas vytvorenia záznamu v tabuľke;
- `plan_reference` – predstavuje ID plán (miestnosti/budovy), v ktorej sa objekt pri zaznamenaní jeho pozície práve nachádzal.

Potrebné dáta pre Sage Analytics obsahujú však aj ďalšie tabuľky napríklad: `zones`, `feeds`, `plans` a podobne. Tieto tabuľky slúžia na rôzne účely napríklad: `zones` na vytváranie zón, v ktorých je potom možne kontrolovať pohyb objektov, `feeds` predstavuje jednotlivé objekty (konkrétne tagy), ktoré sa pohybujú v priestore a `plans` sú zase plány jednotlivých miestností/budov, po ktorých sa objekty pohybujú. Každá z týchto tabuliek ma vlastné atribúty adekvátne k danému objektu. Schému týchto tabuliek je možné vidieť na obrázku 4.1.

Postupom času sa pre Sage Analytics môžu pridávať aj ďalšie funkcionality a s nimi tak aj nové tabuľky, každopádne všetky nástroje využívajú základnú tabuľku, a to históriu pohybu teda `position_history`.



Obr. 4.1: Schéma hlavných tabuliek využívaných v Sage Analytics

4.2 Získavanie dát

Hneď po tom, ako si užívateľ vyžiada zobrazenie (analýzu) nejakých dát (pre určité objekty, časový interval a miesto), ako prvá potrebná činnosť, ktorá sa vykoná na strane servera je získanie dát z databázy. Táto činnosť je veľmi dôležitá, a to z toho dôvodu, že dokáže výrazne ovplyvniť čas, za ktorý sa celá požiadavka, ktorú server od klienta dostal spracuje.

Vzhľadom nato, že dáta sú súčasne uložené v relačnej *MySQL* databáze, sme k nim schopní pristupovať s využitím jazyka `SQL`.

4.2.1 SQL

SQL alebo Structured Query Language [8] je počítačový jazyk určený na správu relačných databáz a vykonávanie rôznych operácií s ich dátami.

SQL funguje tak, že programátorom a ostatným používateľom počítačov poskytuje spôsob, ako získať požadované informácie z databázy pomocou niečoho čo pripomína anglický jazyk. Na najjednoduchšej úrovni pozostáva SQL len z niekoľkých príkazov: `SELECT`, ktorý zachytáva údaje, `INSERT`, ktorý pridáva údaje do databázy, `UPDATE`, ktorý mení dáta v databáze a `DELETE`, čím sa vymazávajú informácie z databázy. Existujú aj iné príkazy, a to na vytváranie, úpravu a správu databáz.

SQL je v súčastnosti veľmi populárne vzhľadom na vysokú popularitu relačných databáz, a preto sa s ním môžeme stretnúť takmer všade, od vládnych databáz, až po stránky elektronického obchodu na internete. S rastom popularity sa neustále zlepšoval a optimalizoval spôsob, akým relačné databázy fungujú dnes.

4.2.2 Proces získavania dát

SQL príkazy môžu mať rôznu formu. Záleží to od toho akým nástrojom chce užívateľ dáta prezentovať, ale hlavne od toho pre aký časový interval, aké objekty, a v akej miestnosti si dáta vyžaduje.

Dáta z databázy je možné získať vytvorením rôznych SQL požiadavok. Príkladom môžu byť požiadavky vo výpise 4.1.

```
1. SELECT name, description FROM feeds WHERE id = 1;
    /* Ziskanie mena a popisu z tabulky feeds,
    kde id zaznamu je 1. */
2. SELECT ST_X(position), ST_Y(position) FROM position_history
    WHERE at > '2021-08-15 00:00:00' AND at < '2021-08-15 23:59:59'
    AND feed_reference = 243;
    /* Ziskanie polohy (suradnice x a y) tagu s ID 243 pocas dna
   15.8.2021 */
3. SELECT feed_reference FROM position_history
    WHERE at > '2021-08-15 00:00:00' AND at < '2021-08-15 23:59:59'
    AND plan_reference = 2;
    /* Ziskanie ID tagu, ktory sa pocas dna 15.8.2021 vyskytoval
    v plane s ID 2 */
```

Výpis 4.1: Rôzne príklady SQL požiadaviek

Príkazy pre získavanie dát z databázy môžu vyzerať akokoľvek, ide však hlavne o to, aby boli správne optimalizované a nevyžadovali si zbytočne nepotrebné údaje z databázy, ale len tie, ktoré k analýze a prezentovaniu dát potrebujeme. Zle postavené príkazy môžu viesť k zbytočne zdĺhavému procesu získavania dát.

To za aký čas sú dáta z databázy získavané nezáleží len od toho ako sú príkazy postavené, ale aj od toho, akou formou sú dáta uložené, aký engine[3] databáza využíva a akú výpočetnú silu má k dispozícií.

[3]Databázový engine je základným softwarovým komponentom, ktorou SRBD používa na manipuláciou s dátami.

Existujúce riešenie, poskytuje dáta v čase, ktorý pri získaní malého množstva dát nie je zlý, avšak pri získavaní väčšieho množstvo dát (napr. pre niekoľko dní alebo týždňov) sa čas, za ktorý sme schopní získať dáta z databázy výrazne zväčšuje, až sa stáva neprípustným pre normálny chod webovej aplikácie. Na to, aby bolo možné porovnať za aký čas poskytne dáta pri zadaní príkazu súčasne riešenie a riešenie navrhnuté v tejto diplomovej práci, bolo nutné vykonať niekoľko zaujímavých požiadavok na databázu a získať čas vykonania súčasného riešenia. Databáza, na ktorej testy prebiehajú má v tabuľke `position_history` vyše 284 miliónov záznamov a veľkosť cez 38GB.

	Test [SQL]	Čas [s]
1.	SELECT ST_X(position) FROM position_history WHERE at > '2021-08-15 12:00:00' AND at < '2021-08-15 13:00:00';	248.879
2.	SELECT ST_X(position) FROM position_history WHERE at > '2021-08-15 12:00:00' AND at < '2021-08-15 13:30:00';	247.624
3.	SELECT ST_X(position) FROM position_history WHERE at > '2021-08-15 12:00:00' AND at < '2021-08-15 14:00:00';	256.103
4.	SELECT COUNT(*) FROM position_history;	58.996
5.	SELECT COUNT(*) FROM position_history WHERE at > '2021-08-10';	156.057
6.	SELECT COUNT(*) FROM position_history WHERE at > '2019-01-01';	157.345
7.	SELECT * FROM position_history WHERE at > '2021-08-15 00:00:00' AND at < '2021-08-15 23:59:59' AND feed_reference = 243;	219.211
8.	SELECT ST_X(position), ST_Y(position) FROM position_history WHERE at > '2021-08-15 00:00:00' AND at < '2021-08-15 23:59:59' AND feed_reference = 243;	249.377
9.	SELECT ST_X(position), ST_Y(position) FROM position_history WHERE at > '2021-08-15 00:00:00' AND at < '2021-08-16 23:59:59' AND feed_reference = 243;	205.022
10.	SELECT ST_X(position), ST_Y(position) FROM position_history WHERE at > '2021-08-15 00:00:00' AND at < '2021-08-17 23:59:59' AND feed_reference = 243;	218.113

Tabuľka 4.1: Zobrazenie času pre rôzne požiadavky na získanie dát z databázy.

Kapitola 5

Možné spôsoby optimalizácie a návrh nového riešenia

Existujúce riešenie poskytuje spôsob uloženia pozičných dát, ktorý vzhľadom na časovú náročnosť nevyhovuje nástroju Sage Analytics. Požiadavky na takéto dáta trvajú pre dlhšie časové intervaly až príliš dlho, a preto je potrebné pozrieť sa nato, akým spôsobom by bolo možné súčasné riešenie optimalizovať. V prípade, že ani optimalizácia súčasného riešenia neposkytne vyhovujúce výsledky je nutné pozrieť sa na možné alternatívy, ktoré by boli schopné nahradiť aktuálny spôsob ukladania pozičných dát, a zároveň aby takéto spôsoby uloženia poskytovali značné urýchlenie pri požiadavkách a splnili by tak účel, na ktorý existujúce riešenie nebolo dostatočné.

Táto kapitola v sekcii 5.1 opisuje to, akým spôsobom je možné optimalizovať aktuálne riešenie a aké výsledky to prináša. V ďalšej sekcii 5.2 sa opisujú alternatívne prístupy, ktoré by mohli byť vhodné na uloženie pozičných dát a poskytujú modernejšie riešenie ukladania dát v porovnaní s relačnými databázami. Sú v nej popísané ich výhody, nevýhody a zmeranie času, za aký zvládajú požiadavky na získanie dát v porovnaní s existujúcim riešením.

5.1 Optimalizácia existujúceho riešenia

Relačné databázové riešenie so sebou prináša množstvo výhod čomu nasvedčuje aj jeho popularita. Je dobrým spôsobom uloženia dát, kedy tabuľky obsahujú veľké množstva záznamov (riadkov). Na to, aby databáza poskytovala dostatočné rýchle získavanie dát je potrebné ju určitým spôsobom optimalizovať a prispôsobiť pre dáta, ktoré bude v sebe uchovávať. Zároveň takáto optimalizáciu musí byť vykonaná tak, aby nepriniesla so sebou aj množstvo nechcených zmien ako napríklad neadekvátne zväčšenie databázy alebo značné predlženie doby pre ukladanie dát.

Spôsoby ako optimalizovať relačnú databázu je hneď niekoľko, avšak v tomto prípade bolo potrebné sa zamerať hlavne na to, ako urýchliť získavanie dát čo najviac. Vzhľadom na to, že najdôležitejšie a najčastejšie požadované dáta sú uložené v jednej tabuľke, a to v `position_history` ako je popísané v 4.1.2. To dosť obmedzuje spôsob, akým by sa dala databáza optimalizovať a súčasne priniesla dostačujúce zlepšenie.

Niekoľko zaručených spôsobov, ktoré dopomáhajú k urýchleniu získavania dát sú napríklad: indexovanie, optimalizácia požiadavok, ale aj praktiky ako sharding alebo zvýšenie výkonu. Táto sekcia vychádza z [30], [16].

5.1.1 Zvýšenie výkonu a sharding

Okrem optimalizácie štruktúry databázy a požiadavkov je možné optimalizáciu vykonať aj zvýšením výkonu respektíve poskytnutím väčšieho množstva výpočetných prostriedkov k získavaniu dát. Zvýšenie výkonu je možné vykonať dvoma spôsobmi:

- Zväčšením pamäte – efektivita databázy môže výrazne utrpieť, ak systém nedisponuje dostatkom pamäte na to, aby databáza fungovala správne. Zvýšenie alokácie pamäte pomôže zvýšiť efektivitu a celkový výkon.
- Posilnením CPU – Zlepšenie výpočetného výkonu, a to či už pridaním ďalšieho procesora, alebo nasadením výkonnejšieho procesora sa premieta priamo do efektívnejšej databázy. Platí teda, že čím výkonnejší je procesor, tým menšie zaťaženie bude mať pri riešení viacerých požiadaviek a aplikácií.

Iný spôsob, ako zvýšiť výkon a časovo urýchliť požiadavky na databázu je tzv. **sharding**. Jedná sa o metódu, kedy sa záznamy databázy prerozdelia na viacero strojov, a tak následne pri získavaní potrebných dát sa záťaž prerozdelí na viacero strojov, kedy tieto stroje súčasne prechádzajú databázu a získavajú potrebné dáta.

Takéto spôsoby zaručene urýchlia získavanie dát, ale majú však jednu veľkú nevýhodu, a to výrazne navýšenie nákladov. Práve zvýšenie nákladov je dôvod, pre ktorý som sa v rámci svojich možností nemohol tomuto spôsobu optimalizácie venovať. Avšak vzhľadom na jej účinnosť ju v tejto sekcii zmieňujem.

5.1.2 Indexovanie

Index je databázová štruktúra, ktorú je možné použiť pri získavaní dát z databázy. Umožňuje databáze rýchlejšie nájsť relevantné riadky udržiavaním efektívnej štruktúry údajov vyhľadávania (napr. B-strom). Pridanie indexu môže byť výpočtovo nákladné, a musí sa vykonávať v produkčnom systéme, takže sa zvyčajne vykonáva zriedkavo. Ako indexy sa ukladajú napríklad kľúče (primárny, cudzí), čo urýchľuje vyhľadávanie podľa kľúča. Niekedy musíme byť schopní rýchlo vyhľadať údaje, ktoré nie sú uložené ako kľúč. Môžeme napríklad rýchlo potrebovať vyhľadať užívateľov podľa mena. Nebolo by vhodné použiť jedinečné obmedzenie, pretože môžeme mať viacero užívateľov s rovnakým menom, v týchto prípadoch môžeme vytvárať vlastné indexy. Z pomedzi všetkých techník pre optimalizáciu databázy pre získavanie dát práve indexovanie prináša najlepšie výsledky. Indexovanie so sebou prináša niekoľko výhod, ale aj nevýhod (popísaných v tabuľke 5.1), a preto by mali byť využívané len v prípade, keď sú naozaj potrebné.

Výhody	**Nevýhody**
Urýchľuje požiadavky typu SELECT	Zaberá značné miesto na disku
Jedinečný riadok, bez duplikátov	Spomaľujú operácie INSERT, UPDATE a DELETE
Môžu byť použité na triedenie.	

Tabuľka 5.1: Porovnanie výhod a nevýhod indexovania

Aby indexovanie malo zmysel a bolo efektívne, je potrebné umiestniť index na vhodný stĺpec (alebo viac stĺpcov). Väčšinou sa jedná o stĺpec, cez ktorý získavame údaje z tabuľky po klauzule `WHERE` a stĺpec, ktorý v požiadavkách využívame najčastejšie. V prípade tabuľky `position_history` som vyskúšal viacero možností, kde index umiestniť. Vzhľadom na fakt, že požiadavky sú vždy vytvárané pre časový interval, vytvorenie indexu práve pre stĺpec s časovými záznamami pre každú pozíciu, čiže pre stĺpec „`at`“ prinieslo významné urýchlenie.

5.1.3 Optimalizácia požiadavok

Optimalizácia požiadavok sa dá meniť až priamo pri zadávaní požiadavok databáze. Aj napriek tomu, že sa nejedná o žiaden zložitý úkon, je potrebné aby príkazy, pomocou ktorých sme schopní vyžadovať dáta boli písané tak, aby pracovali čo najefektívnejšie. Efektívna požiadavka by mala byť taká, ktorá od databázy požaduje len naozaj potrebné dáta a nevyžaduje aj navyše zbytočné záznamy.

Medzi základné praktiky pre vytvorenie efektívnych požiadaviek patria:

1. využívanie názvov jednotlivých stĺpcov namiesto *,
2. vyhnutie sa vnoreným požiadavkám a pohľadom (views),
3. pri požiadavkách na stĺpec s indexom je vhodnejšie použiť predikát IN namiesto použitia matematických operátorov, ako je „=“, alebo logických operátorov,
4. a niekoľko ďalších[1].

5.1.4 Výsledky optimalizácie

Optimalizácia priniesla pozitívne výsledky, dokázala určité požiadavky značne urýchliť. Najväčší vplyv na urýchlenie malo práve indexovanie. Nato, aby indexovanie malo správny a čo najväčší účinok bolo potrebné preskúmať, zistiť a rozhodnúť, na ktorý stĺpec tabuľky je umiestnenie indexu najvhodnejšie, poprípade, či nemôže využitie tzv. zloženého indexu priniesť ešte lepšie výsledky.

Prvé lepšie výsledky prinieslo umiestnenie indexu na stĺpec s časovým údajom (stĺpec s názvom `at`) Ako môžeme z tabuľky 5.2 vidieť, pri požiadavkách, ktoré sú primárne založené na získaní dát podľa časového údaju dochádza k výraznému zrýchleniu.

Problém však nastáva v situácii, kedy získavanie dát z databázy neprebieha len na základe zvoleného časového intervalu, ale aj výbere konkrétneho tagu (stĺpec s názvom `feed_reference`). To je však problém, pretože väčšina požiadaviek na databázu prebieha práve s využitím aj tohto stĺpca. Z tohto dôvodu bolo nutné pozrieť sa na spôsob, akým by sa táto situácia dala vyriešiť. Veľmi vhodným riešením bolo využitie práve zloženého indexu, ktorý sa skladal zo stĺpcov s tagom a časovým údajom (`feed_reference` a `at`). Výsledky využitia takéhoto indexu môžeme opäť pozorovať v tabuľke 5.2, kde je jasne vidieť, že takéto využitie indexu prináša značne lepšie výsledky, a to práve v prípade, kedy sa požadujú dáta na základe časového intervalu a tagu zároveň. Práve táto situácia je súčasťou väčšiny metrík v nástroji Sage Analytics, a preto je zrýchlenie požiadaviek tohto typu kľúčové.

Aj napriek tomu, že optimalizáciu existujúceho riešenia priniesla veľmi dobré výsledky, tieto výsledky sú ale akousi kombináciou využitia indexov, čo v praxi nie je ideálne, pretože zaberá veľké množstvo priestoru na disku.

[1] https://ubiq.co/database-blog/how-to-speed-up-sql-queries/

V dnešnej dobe klasické relačné riešenie uloženia dát nie je jediná možnosť ako vhodne uložiť dáta, a preto je potrebné sa pozrieť aj po iných, novších a sofistikovanejších možnostiach, ako pozičné dáta uložiť.

Číslo testu z 4.1	Čas[s] pred optimalizáciou	Čas[s] jednoduchý index	Čas[s] zložený index	Najväčšie zrýchlenie
1	248	10	161	2380 %
3	256	28	171	814 %
5	156	99	76	105 %
7	219	209	145	51 %
8	249	199	14	1678 %
10	218	92	30	626 %

Tabuľka 5.2: Zobrazenie času pre rôzne požiadavky na získanie dát z databázy.

5.2 Alternatívne prístupy

Relačná databáza (*MySQL*) síce poskytuje vhodný spôsob pre štrukturovaný typ dát, ktoré potrebujeme ukladať, avšak neposkytuje dostatočnú rýchlosť nato, aby sme dáta získavali v požadovanom čase. Po optimalizovaní tejto databázy síce došlo k výraznejšiemu časovému zlepšeniu, avšak aj napriek tomu bolo vhodné pozrieť sa na iné, alternatívne spôsoby, ktoré by mohli aktuálne databázové riešenie nahradiť a splnili by požadovaný cieľ.

V dnešnej dobe už dávno neplatí, že jediným a spoľahlivým databázovým riešením je relačná databáza. Postupom času sa stále viac do popredia dostávajú alternatívne prístupy, ktoré so sebou nesú množstvo výhod (ale aj nevýhod) a sú lepšie prispôsobené na iný typ dát. Medzi takéto alternatívne typy patria napríklad *NoSQL*, *post-relačné*, *objektovo-orientované*, *grafové* a podobne. Pri práci na tejto diplomovej práci som sa pozrel na niekoľko z nich, ktoré by mohli byť vhodnou alternatívou a porovnal som výsledky zo spracovania požiadavok s existujúcim riešením.

5.2.1 NoSQL

NoSQL [23],[17] (alebo „not only SQL“) sú netabuľkové databázy, ktoré ukladajú dáta iným spôsobom ako relačné tabuľky. Dovoľujú ukladať veľké množstvo neštrukturovaných dát, čím sú maximálne flexibilné. Dodržiavanie ACID nevhodne obmedzuje prácu s databázou, práve z tohto dôvodu NoSQL databázy zanedbávajú tieto vlastnosti pre získanie vyššej rýchlosti a dostupnosti dát.

NoSQL databázy rozdeľujeme na základe typu dátového modelu na (vychádza z [21]):

1. **kľúč-hodnota** – Záznam je tvorený jedným kľúčom a hodnotou, kde sa k hodnote pristupuje pomocou kľúča cez hash tabuľky (veľmi rychlé) a hodnoty sú uložené ako BLOB. Neefektívna, pri získavaní len časti z hodnoty. Medzi také patria napríklad: *Dynamo DB*, *Redis*, *BerkleyDB*.

2. **dokumentové** – Veľmi podobné ako kľúč-hodnota ale hodnota je štrukturovaná pomocou XML/JSON alebo ako objekt. Dovoľuje aj zložitejšie požiadavky ako cez kľúč. Patria medzi ne napríklad: *MongoDB*, *CouchDB*, *Lotus Notes*.

3. **grafové** – ukladajú dáta pomocou uzlov a hrán. Uzly väčšinou obsahujú informácie o objektoch (ľudia, miesta, veci) a hrany vzťahy medzi nimi. Medzi také patria napríklad: *Neo4j*, *OrientDB*, *Allegrograph*.

4. **stĺpcové** – ukladajú údaje do tabuliek, riadkov a dynamických stĺpcov. Medzi také patria napríklad: *Google's Bigtable*, *Cassandra*, *HBase*.

Podľa [23], kľúčové vlastnosti (nájdenie jednej alebo viacerých) pre výber NoSQL databáze sú: rýchly agilný vývoj, ukladanie štruktúrovaných a polo-štruktúrovaných dát, obrovské objemy dát, požiadavky na škálovateľnú architektúru, moderné aplikačné paradigmy, ako sú mikro-služby a streamovanie v reálnom čase. Vzhľadom nato, že s niektorými z týchto vlastností (ukladanie štruktúrovaných dát a obrovské objemy dát) sa dáta pre Sage Analytics zhodujú, rozhodol som sa tento prístup vyskúšať.

Číslo testu z 4.1	MySQL Čas[s]	MySQL opt. Čas[s]	MongoDB Čas[s]
3	256	28	450
5	156	99	452
7	219	145	301
8	249	14	325
12	218	30	322

Tabuľka 5.3: Zobrazenie času niektorých testov pre získanie dát z databázy. Porovnávané sú: existujúce riešenie MySQL, optimalizované riešenie MySQL a MongoDB.

MongoDB

Zo všetkých možných ponúkaných variant, som si vybral dokumentovú databázu `MongoDB`[2]. Táto databáza je veľmi populárna pri NoSQL riešeniach, využívaná v rôznych odvetviach ako sú: finančný sektor, telekomunikácia, zdravotníctvo, maloobchod a podobne. Čo je dôležité, že splňovala vlastnosti, ktoré boli vhodné pre prácu s lokalizačnými dátami, a zároveň poskytuje nový, moderný prístup k ukladaniu dát.

Pre prevedie záznamov z relačnej MySQL databázy do `MongoDB` som napísal skript v jazyku `Python`, ktorý využíval knižnicu `mysql-connector`[3], pre prípojenie na MySQL databázu a získanie dát a knižnicu `PyMongo`[4] pre vytvorenie spojenia s `MongoDB`, vytvorenie databázy a uloženie získaných dát. V niektorých prípadoch, ako napríklad záznamu `position`, bolo potrebné dáta pretransformovať do vhodnej formy.

Po prevedení dát do `MongoDB` som vykonal niekoľko testov, kde som sa snažil zistiť, ako táto databáza pracuje, a či môže predstavovať alternatívne riešenie pre existujúce.

[2] https://www.mongodb.com/
[3] https://pypi.org/project/mysql-connector-python/
[4] https://docs.mongodb.com/drivers/pymongo/

Žiaľ ako je možné vidieť z tabuľky 5.3, databáza `MongoDB` neposkytla požadované zrýchlenie, a tak by jej využitie bolo neúčinné. Z tohto dôvodu sa bolo nutné pozrieť na iné alternatívne riešenia, ktoré by mohli prácu s pozičnými dátami výrazne zrýchliť.

5.2.2 Priestorová databáza

Ďalším z možných riešení, ktoré by mohli vyriešiť správne ukladanie pozičných dát a následne rýchle spracovanie požiadaviek som sa pozrel na priestorovú (spatial) databázu. Táto podsekcia vychádza z [5] a [28].

Priestorová databáza, je post-relačné databázové riešenie, ktoré je schopné spravovať a ukladať priestorové dáta alebo dáta, ktoré sú definované geometrickým priestorom, kde priestor predstavuje 2 a viac dimenzií. Takéto dáta predstavujú často geografickú polohu alebo objekty umiestnené v prostredí. Údaje, ktoré definujú priestorové dáta sú často reprezentované ako:

- **bod** – planéty, mestá, atómy (aspoň 1 bod),
- **čiara** – cesty a prepojenia (aspoň 2 body),
- **polygón** – hranice miest, štátov, štvrtí, lesy, polia, moria, lúky (aspoň 3 body).

Niektoré priestorové databázy sú schopné spracovať aj zložitejšie údaje, ako sú trojrozmerné objekty, topologické pokrytie a lineárne siete.

Zatiaľ čo bežné relačné databázové riešenia používajú indexy pre rýchlejšie a efektívnejšie vyhľadávanie a prístup k údajom, priestorové databázy používajú niečo ako jedinečný index nazývaný **priestorový index** na zrýchlenie výkonu databázy. Priestorové indexovanie je veľmi výhodné a potrebné, pretože pozícia objektu sa získava z veľkej skupiny objektov, bez nutnosti prehľadávania celej skupiny objektov.

Po preštudovaní priestorových databáz a pochopení ich fungovania a účelu, som zhodnotil, že sa ich využitie nie úplne zhoduje s využitím, ktoré sa očakáva pri ukladaní a získavaní dát pre nástroj `Sage Analytics`. Priestorové databázy sa sústreďujú či už na geografické dáta a ich vhodné spracovanie, alebo priamo na spracovanie objektov v priestore. Pozičné dáta, ktoré sú predmetom tejto diplomovej práce, však predstavujú jednoduchú 2D pozíciu (3D v prípade 3 priestorovej súradnice), ktorú ale nie je potrebné získavať priamo z priestoru, respektíve počítať zložité operácie nad priestorom. Zároveň sa požadujú dáta ohraničené časovým intervalom, nie podľa polohy v priestore. Z tohto dôvodu som sa rozhodol, že využitie priestorových databáz nie je dobrý smer a nebude vhodným alternatívnym riešením pre aktuálne riešenie.

5.2.3 TimeSeries databáza

Práve uvedomenie si, že v nástroji `Sage Analytics` sa síce vyžadujú pozičné dáta, tieto dáta sú však stále vyžadované podľa určitého časového intervalu, a tak hlavným údajom, podľa ktorého sa v databáze vyhľadáva je práve časový údaj. Toto tvrdenie zároveň potvrdzuje aj fakt, že pri optimalizácií súčasného riešenia v 5.1, malo najvýraznejší vplyv a najviac urýchlilo získavanie dát z databázy umiestnenie indexu na stĺpec s časovými údajmi. Preto som sa rozhodol, že sa pozriem na databázy, ktoré sa sústreďujú primárne na ukladanie a prácu s časovými údajmi.

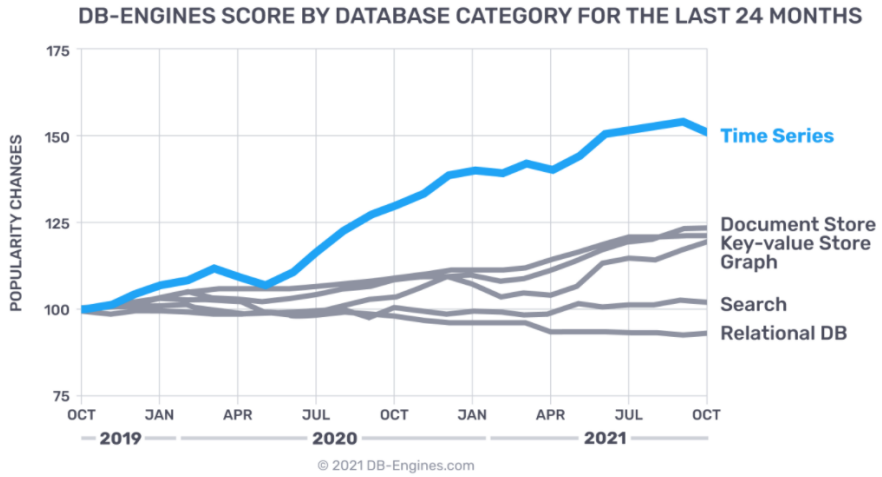
TimeSeries [6] databáza alebo databáza časových radov je databáza optimalizovaná pre údaje s časovou značkou alebo časovými radmi. Jedná sa o jednoduché meranie, respektíve zaznamenávanie udalostí, ktoré sú monitorované a agregované v priebehu času. Medzi

takéto merania môžeme zaradiť serverové metriky, monitorovanie výkonu aplikácií, sieťové údaje, údaje zo senzorov, udalosti, kliknutia, obchody na trhu a mnoho ďalších typov analytických údajov. Táto databáza je vytvorená a optimalizovaná špeciálne pre spracovanie metrík a udalostí, alebo meraní, ktoré sú označené časovou značkou. Tento typ databázy je vhodný pre horizontálne škálovateľné dáta a je predurčený na požadovanie údajov za dlhé časové obdobie, čo presne predstavuje to, čo nástroj `Sage analytics` vyžaduje pre správne fungovanie v rozumnom čase.

Medzi výhody timeseries databázy by som zaradil:

- veľmi rýchle spracovanie dát za dlhé časové obdobie,
- presnejšie a zmysluplnejšie meranie časových radov,
- efektívne ukladanie dát.

Koncept timeseries databázy nepatrí medzi najnovšie, avšak v poslednej dobe je možné pozorovať veľký nárast ich použitia a významnejší posun v popularite v rámci databázových riešení (rast popularity je možné vidieť na obrázku 5.1). Medzi známe timeseries databázy patria napríklad: *InfluxDB*, *Graphite*, *Prometheus*, *TimescaleDB* a rôzne ďalšie[5].



Obr. 5.1: Porovnanie popularity databázových riešení za posledných 24 mesiacov. Merania sú vykonávane portálom https://db-engines.com/. Obrázok je prevzatý z [27].

Vzhľadom na predpoklad, že tento typ databázy v sebe uchováva naozaj veľké množstvo dát, ktoré pravidelne a veľmi často pribúdajú, má za následok hromadenie dát. V takomto prípade by bolo vhodné staré dáta, ktoré už nie sú užitočné zahodiť, čím sa predíde zahlteniu databázy. Preto bolo nutné prísť s riešením, ako si s týmto problémom poradiť. To spočíva v definovaní tzv. `Retention policy` [22], alebo v preklade akési zásady uchovávania dát, ktoré poskytujú jednoduchý a účinný spôsob, ako takúto situáciu vyriešiť. Ide v podstate o dátum vypršania platnosti údajov. Keď údajom vyprší platnosť, budú automaticky vyradené z databázy, čo je akcia bežne označovaná ako presadzovanie zásad uchovávania. Keď príde čas na zmazanie týchto údajov, nezmaže sa len jeden údaj, ale naraz sa zmaže celá skupina údajov alebo tzv. shard.

[5] https://geekflare.com/time-series-database/

Shards sú akési kontajnery, ktoré v sebe uchovávajú rôzne veľké množstvo údajov zväčša zvolené podľa nejakého časového intervalu. Tento časový interval sa nastavuje práve podľa toho, ako je nastavené `Retention policy`, alebo podľa vlastného nastavenia. Pre príklad taká `InfluxDB` sa implicitne riadi pravidlami popísanými v tabuľke 5.4. Využitie Shardov však neslúži len k mazaniu dát, ale hlavne k urýchlenému vyhľadávaniu. Vzhľadom na to, že sú dáta podelené podľa časových úsekov je výrazne jednoduchšie takéto dáta získať.

Obdobie uchovania dát	Obdobie 1 shardu
menej ako 2 dni	1 hodina
medzi 2 dňami až 6 mesiacmi	1 deň
viac ako 6 mesiacov	7 dní

Tabuľka 5.4: Zobrazenie spôsobu nastavenia `Retention policy` a `shardov`. Konkrétne sa jedná o predvolené nastavenia databázy `InfluxDB`, ktoré je možné meniť. Získané z [26].

InfluxDB

Spomedzi veľkého množstva timeseries databázy som si vybral InfluxDB[6].

`InfluxDB` je nielen timeseries databáza, ale aj základná súprava nástrojov, dashboardy(prístrojové dosky), úlohy a agenti na jednom mieste. Jedná sa o open-source databázu vyvinutú spoločnosťou `InfluxData`. Je napísaná v programovacom jazyku `Go` na veľmi efektívne ukladanie a získavanie údajov časových radov v oblastiach, ako je monitorovanie rôznych operácií, údaje zo senzorov internetu vecí (IoT), analýzy v reálnom čase a iné. Je navrhnutá od základov tak, aby poskytovala vysoko škálovateľný nástroj na prijímanie a ukladanie dát. Poskytuje pre-vzorkovania a zásady uchovávania údajov, kde údaje s vysokou hodnotou, vysokou presnosťou uchováva v pamäti, a údaje s nižšou hodnotou na disku. Na interakciu s údajmi používa jazyk `InfluxQL`, ktorý je veľmi podobný jazyku `SQL`.

Funkcie a vlastnosti, ktoré `InfluxDB` ponúka sú:

- vysoká výkonnosť pre časové dáta;
- jazyk `InfluxQL`, podobný jazyku `SQL`
- balík komponentov TICK (*Telegraf*, *InfluxDB*, *Chronograf*, a *Kapacitor*);
- schopnosť spracovať milióny dátových záznamov za sekundu;
- možnosť automatického odstránenia zastaraných údajov.

Podľa popisu a vlastností sa databáza javila ako veľmi vhodný kandidát, a preto bolo potrebné vyskúšať previesť dáta z databázy `MySQL` do novovytvorenej `InfluxDB`. K prevodu dát, som využil, podobne ako pri databáze `MongoDB`, mnou napísaný skript v jazyku `Python`, ktorý využíval knižnicu `mysql-connector`[7] pre pripojenie na `MySQL` databázu a získanie dát a knižnicu `InfluxDB`[8] pre vytvorenie spojenia s `InfluxDB`, vytvorenie databázy a uloženie získaných dát.

[6] https://www.influxdata.com/
[7] https://pypi.org/project/mysql-connector-python/
[8] https://www.influxdata.com/blog/getting-started-python-influxdb/

Na to aby, bolo možné zistiť, či využitie takéhoto typu databázy bude mať zmysel pre naše pozičné dáta, bolo opäť nutné vykonať merania a pozrieť sa, v akých prípadoch by `InfluxDB` priniesla zlepšenie, a či vôbec. Pre overenie rýchlosti vykonávania požiadavkov som vykonal niekoľko testov:

Číslo testu	MySQL Čas[s]	MySQL opt. Čas[s]	InfluxDB Čas[s]	Zrýchlenie MySQL/MySQL opt.
1	248	10	5	4860 %/100 %
2	256	28	10	2460 %/180 %
3	156	76	41	280 %/85 %
4	219	145	7	3028 %/1971 %
5	249	14	5	4880 %/180 %
6	218	40	15	1353 %/167 %

Tabuľka 5.5: Zobrazenie času pre rôzne požiadavky na získanie dát z MySQL databázy, z optimalizovanej MySQL databázy, z InfluxDB a dosiahnuté zrýchlenie použitím InfluxDB.

Z tabuľky 5.5 je veľmi dobre vidieť, aký výrazný vplyv má využitie databázy `InfluxDB` pre získavanie pozičných dát. Pre všetky uskutočnené testy mala databáza `InfluxDB` výrazný náskok oproti existujúcemu riešeniu. Dokonca výrazným spôsobom porazila aj optimalizované riešenie, ktoré už samo dosahovalo dobré výsledky. Z toho vyplýva, že táto databáza je vhodnou a veľmi efektívnou alternatívou pre existujúce riešenie.

Kapitola 6

Návrh uloženia a spôsob prenosu dát

V predchádzajúcej kapitole som dospel k záveru, že optimalizácia existujúceho riešenia prináša výrazne zlepšenie, a teda má zmysel takýto postup brať do úvahy. Zároveň však existuje aj alternatívne riešenie, ktoré svojím účelom presne sedí na pozičné dáta, čo navyše dokazujú aj testy výkonnosti.

Táto kapitola sa zaoberá spôsobom návrhu uloženia pozičných dát v databáze InfluxDB v sekcii 6.1 a migráciou dát z existujúceho riešenia do novo-navrhnutého v sekcii 6.2, čo je nevyhnutné pre otestovanie výkonnosti implementovaného riešenia.

6.1 Návrh uloženia pozičných dát v InfluxDB

Po pokusoch o optimalizáciu existujúceho riešenia a následnom hľadaní alternatívneho riešenia sa ako správna voľba javila databáza InfluxDB. Táto timeseries databáza disponuje vlastnosťami, ktoré vyhovujú uloženiu pozičných dát, a zároveň sa využitie tejto databázy zhoduje s využitím, ktoré je potrebné pre správne fungovanie nástroja `Sage Analytics`. Po vykonaní testov a porovnaní získavania dát sa toto tvrdenie potvrdilo a databázu InfluxDB tak možno považovať za vhodnú alternatívu.

Aby bolo možné správne uložiť pozičné dáta a previesť tak celú tabuľku s pozičnými dátami `position_history`, ktoré sú súčasťou nástroja pre analýzu, do novej databázy InfluxDB, je potrebné pochopiť dátový model tejto databázy, jej fungovanie a syntax jazyka InfluxQL. Prevod iných tabuliek, ktoré sú súčasťou existujúceho riešenia `position_history` do databázy InfluxDB by žiadne zlepšenie neprinieslo, a to halvne preto, že sa zväčša jedná o malé množstvo záznamov. Zároveň by pokus prevádzať aj zvyšok existujúceho riešenia všetko len viac skomplikoval, keďže ostatné tabuľky (okrem `position_history`) nedisponujú časovým záznamom, čo je v InfluxDB požadované. Práve preto bude ideálnym riešením uloženie pozičných dát (tabuľka `position_history`) v databáze InfluxDB a zvyšné dáta v aktuálnej MySQL.

Dátový model InfluxDB

Dátový model InfluxDB je úplne odlišný od iných riešení timeseries databáz. InfluxDB má linkový protokol na odosielanie údajov časových radov, ktorý má nasledujúcu formu (prevzaté z [6]):

$$< measurement >, < tagset >< fieldset >< timestamp >$$

Jednotlivé časti tohto modelu popisuje nasledujúca tabuľka 6.1.

Element	Povinné/Voliteľné	Popis	Typ
measurement	povinné	názov merania	string
tagset	voliteľné	kolekcia párov kľúč-hodnota	string
fieldset	povinné	kolekcia párov kľúč-hodnota	hodnoty môžu byť int64, float64, bool alebo string.
timestamp	povinné (môže sa vytvoriť pri vložení do DB)	časová značka pre údaj	unixový čas

Tabuľka 6.1: Popis elementov dátového modelu InfluxDB.

Pri návrhu som sa zameral na tabuľku `position_history`, ktorá je hlavným úložiskom pozičných dát. Nato, aby bolo možné s dátami správne pracovať, bolo nutné jednotlivé stĺpce z relačnej MySQL databázy previesť na dátový model InfluxDB. Podľa zamerania elementov v dátovom modeli som určil, ktoré časti z existujúceho riešenia budú reprezentované súvisiacim elementom v InfluxDB.

Zvolenie elementu „`measurement`“ bolo jasné, keďže sa jedná v preklade o meranie, čo v kontexte znamená názov tabuľky. Rovnako jasné bolo zvolenie elementu „`timestamp`“, jedná sa o časový záznam, ktorý v tomto prípade predstavuje stĺpec v databáze MySQL pod názvom „`at`“. Vzhľadom na to, že v tejto situácii sa zaoberám už existujúcimi dátami, tak dátový záznam predstavuje časovú známku pozičného záznamu. Ak sa však InfluxDB bude používať ako súčasť reálneho produktu, časová známka bude automaticky vkladaná pri zápise údajov do databázy. Na pozíciu elementu „`fieldset`“ bolo vhodné zvoliť dáta, pomocou ktorých sa nevykonávajú požiadavky na databázu, pretože nebudú indexované. Výhodou však je, že môžu byť uložené v rôznej forme (`string`, `int`, `float` a podobne). Ako typ `field` som teda uložil pozičné dáta (`posX, posY, posZ`), `clr`, `numberOfAnchors` a `plan_reference`. Ako element „`tagset`“ bolo vhodné uložiť záznam o tagu (`feed_reference`), a to z dôvodu, že element typu `tag` je v InfluxDB indexovaný, a teda vyhľadávanie s jeho využitím bude prebiehať rýchlejšie ako s typom `field`.

Navrhnutý model tak vyzeral následovne:

- measurement – position_history
- timestamp – at
- fieldset – posx, posy, posZ, clr, numberOfAnchors, plan_reference
- tagset – feed_reference,

6.2 Migrácia dát do InfluxDB

Súčasné riešenie ukladá pozičné dáta v databáze `MySQL`. Aby bolo možné implementovať zmeny a následne tieto zmeny otestovať, je nutné dáta z aktuálneho databázového riešenia preniesť do nového, navrhnutého riešenia, konkrétne do databázy `InfluxDB`.

Presun pozičných dát nie je úplne jednoduchý, a to z dôvodu, že nové navrhnuté riešenie ukladá dáta úplne odlišným spôsobom. Jednoduchá migrácia dát, ktorú by bolo možné vykonať pri migrovaní dát medzi dvoma relačnými databázami v tomto prípade nebude možná.

6.2.1 Skript na prekopírovanie dát

Vzhľadom na to, že presun dát medzi starým a novým riešením nie je možné vykonať jednoduchou migráciou, bolo potrebné vytvoriť skript, ktorý dáta postupne z existujúcej MySQL databázy vyberá, a vhodným spôsobom zase vkladá do nového spôsobu uloženia – databázy InfluxDB.

Skript je napísaný v skriptovacom jazyku `Python`. Aby bolo možné pristúpiť k databáze MySQL, skript využíva knižnicu `MySQL Connector/Python`[1] vytvorenú priamo spoločnosťou *MySQL*. Práve k jednoduchému prístupu a integrácie programu v jazyku `Python`. Aby sme boli schopní rovnakú integráciu vykonať aj s databázou InfluxDB, a tým pádom s jednoduchosťou zapisovať dáta, skript využíva knižnicu `influxdb-python`[2], vyvinutú rovnako priamo spoločnosťou *InfluxData* pre jednoduchý prístup programom v jazyku `Python` k databáze.

Po presune všetkých dát sme tak dosiahli, že databáza InfluxDB obsahovala potrebné pozičné dáta, na ktorých sme schopní testovať implementované zmeny. Dáta v tabuľke aktuálne vyzerajú takto:

time (timestamp)	clr (field)	feed_ reference (tag)	numberOf- Anchors (field)	plan_ reference (field)	posZ (field)	posx (field)	posy (field)
1629533294628	0.08	160	5	2	2.00	−10.65	32.06
1629533294615	0.82	198	6	2	2.00	−12.69	2.7
1629533294604	1.25	190	6	2	2.00	−13.46	14.86
1629533294593	0.39	175	5	2	2.00	−13.39	14.89
1629533294587	0.12	171	5	2	2.00	−12.83	2.8

Tabuľka 6.2: Zobrazenie reálnych dát uložených v databáze InfluxDB

[1] https://github.com/mysql/mysql-connector-python
[2] https://github.com/influxdata/influxdb-python

Kapitola 7

Implementácia a testovanie

Navrhnutie architektúry pre uloženie pozičných dát a ich následná migrácia do nového databázového riešenia je základná vec preto, aby bolo možné dáta novým spôsobom získavať. Avšak, aby bolo možné dáta skutočne získať, je potrebné vyriešiť implementáciu, ktorou úlohou bude vhodným spôsobom dáta v prvom rade získať, a následne spracovať takým spôsobom, aby boli pripravené na vizualizáciu vo webovej aplikácii. Jednoduché nahradenie získavania dát nebude dostatočné, a to z dôvodu, že získané dáta prichádzajú v trochu inom formáte, ale hlavne preto, že databáza MySQL disponuje rôznymi funkciami, ktoré bude potrebné určitým spôsobom nahradiť.

7.0.1 Existujúca implementácia

Súčasné riešenie pre získavanie a spracovanie pozičných dát pre nástroj využíva triedu `SageChartService.php`. Jedná sa o triedu napísanú v skriptovacom programovacom jazyku `PHP` a je jednou zo súčastí webovej aplikácie s frameworkom `Laravel`[1].

Účel tejto triedy by sa dal rozdeliť ako:

1. Výber metriky
2. Inicializácia a vytvorenie základných štruktúr
3. Získanie dát
4. Spracovanie dát
5. Uloženie a odoslanie dát

Ako už bolo spomenuté, pre využitie nového databázového riešenia je pri implementácií hlavné zamerať sa na získavanie a spracovanie dát. Na ďalšie časti triedy využitie nového riešenia nemá vplyv, a preto ich nie je potrebné riešiť.

Získanie dát

Vzhľadom na to, že aplikácia je napísaná vo frameworku `Laravel`, získavanie dát z databázy `MySQL` prebieha s využitím metód ponúkanými týmto frameworkom. To výrazne celé napojenie na databázu a získanie dát zjednodušuje.

[1] https://laravel.com/

```
1     $sql = "SELECT st_x(position) AS posX, st_y(position) AS posY, at
2     FROM position_history
3     WHERE at >= :from AND at <= :to AND feed_reference = :lid AND clr < 9998
4     ORDER BY at ASC;";
5     $positions = DB::connection("sensmapserver")->select($sql, ["lid" => $feeds[$k
          ], "from" => $data["from"], "to" => $data["to"]]);
```

Výpis 7.1: Jednoduchý výpis získania pozičných dát.

Získavanie dát, ktoré je možné vidieť v 7.1 je jedným z príkladov triedy `SageChartService`. Jedná sa o relatívne jednoduché získanie pozičných dát na základe časového intervalu a tagu. To akým spôsobom, a aké požiadavky pre získanie dát sú posielané, zaleží od metriky zobrazenia dát (jednotlivé metriky sú popísané v 3.2.5). Niektoré požiadavky sú ale výrazne zložitejšie, získanie záznamov vyžaduje spojenia viacerých tabuliek alebo využitia rôznych `MySQL` funkcií, ktoré majú za cieľ zjednodušiť následovné spracovanie získaných záznamov. Medzi takúto funkciu patrí napríklad `ST_CONTAINS`[2], ktorá sa využíva pri zónových metrikách (First/Last appearance, Zone map).

Spracovanie dát

Získané pozičné dáta je potom potrebné spracovať takým spôsobom, aby ich bolo možné následne po odoslaní klientovi vo webovom prehliadači vhodným spôsobom zobraziť. Zložitosť spracovania získaných záznamov opäť závisí od toho, o akú metriku sa jedná. Zatiaľčo v niektorých prípadoch sa jedná len o vloženie pozičných záznamov do vhodnej štruktúry, v iných sa vyžaduje zložitejšie spracovanie, ako napríklad výpočet času v jednotlivých zónach pre metriku `Zone map` (zónová mapa).

Využitie `MySQL`, jej funkcií a možnosti, ktoré ponúka pre získanie dát spracovanie značne zjednodušuje, a to z toho dôvodu, že to v akej forme dáta získame, vieme viac ovplyvniť. Takúto situáciu môžeme pozorovať vo výpise 7.2, kde pri získavaní dát dôjde k využitiu `ST_CONTAINS`, čo potom umožňuje výrazne zjednodušiť spracovanie, a to takým spôsobom, že získané dáta sú už len jednoduchou podmienkou kontrolované, či sa v zóne nachádzajú alebo nie. Samotný výpočet, či sa bod v zóne (polygóne nachádza) potrebný nie je.

```
1     $sql = "SELECT st_contains(zones.polygon, position_history.position) AS inside
          , position_history.at FROM position_history, zones
2     WHERE position_history.feed_reference IN ($qMarks) AND position_history.at >=
          ? AND position_history.at <= ? AND clr < 9998 AND zones.id= ?
3     ORDER BY position_history.at ASC;";
4
5     $positions = DB::connection("sensmapserver")->select($sql, $bindings);
6     $isInside = false;
7     foreach ($positions as $position) {
8         if ($position->inside) {
9             $isInside = true;
10            break;
11        }
12    }
```

Výpis 7.2: Úsek kódu pre získanie a následne spracovanie pozičných dát pre metriku first_appearance.

[2]Úlohou funckie ST_CONTAINS je zistiť, či sa bod z pozičného záznamu nachádza v zóne alebo inak polygóne definovaného v zónovej tabuľke.

Práve to, že náhradou súčasného databázového riešenia za nové – alternatívne prichádzame o možnosť využívať spôsoby pre získavanie dát, ktoré sú `MySQL` ponúkané môže značne skomplikovať celé spracovanie dát. To potom znamená, že v určitých situáciach by bolo nutné implementovať aj také časti, ktoré v súčasnom riešení nutné nebolo.

7.1 Implementácia navrhnutých zmien

To, aby bolo možné využiť navrhnuté zmeny z 6.1 a použiť nový spôsob ukladania pozičných dát je potrebné upraviť ich proces získavania a spracovania. Ako je spomenuté už v 7.0.1 existujúce riešenia uloženia pozičných dát poskytuje viacero možností, ako jednoducho tieto dáta z databázy získať. Problémom však je, že rýchlosť získavania týchto dát, je výrazne pomalšia, a tak je potrebné využiť alternatívne riešenie. To však so sebou prináša menšie komplikácie, kvôli ktorým je potrebné upraviť súčasnú implementáciu.

To akým spôsobom, a aké veľké zmeny je potrebné vykonať závisí od jednotlivých metrík. Podľa toho, ako bolo nutné implementáciu pozmeniť som vytvoril 4 skupiny metrík, kde sa každá líši náročnosťou zmien ako aj spôsobom.

- Zmeny v SpaghettiMap a HeatMap
- Zmeny v Speed a Distance
- Zmeny v zónových metrikách
- Zmeny pre Activity a Attendance

Samotné spojenie s databázou `InfluxDB` priebieha pomocu klienta z knižnice `influxdb-python`[3], ktorú som využíval už pri vytvorení skriptu na migráciu dát z 6.2.1. Ten zabezpečí rýchle a jednoduché prepojenie medzi databázou triedou `SageChartService` a triedou `SageChartDataService`, kde `SageChartDataService` je mnou vytvorená trieda, ktorá slúži ako akási medzivrstva pre získavanie pozičných dát.

Veľkou výhodou je existencia už spomínaného `InfluxQL` alebo `Influx Query Language`[4], ktorý vzhľadom na syntaktickú podobnosť s jazykom `SQL` (v existujúcom riešení sa využíva pre vytváranie požiadaviek na získanie pozičných dát) predstavuje výbornú náhradu. Zostavovanie požiadaviek pre InfluxDB tak, nepredstavuje žiadny problém. Stačí zobrať požiadavku z existujúceho riešenia vytvorenú pre zisk dát z `MySQL` a jemne ju pretransformovať tak, aby zodpovedala syntaxi jazyka `InfluxQL`. Nevýhodou ale je už spomínaný fakt, že pri využití `InfluxDB` nie sme schopní vytvárať tak komplexné a špecifické požiadavky s využitím rôznych funkcií. Získané dáta preto budú v niektorých prípadoch viac všeobecné a budú si vyžadovať väčšie spracovanie.

Pre jednoduché pripojenie k databáze sa využíva objekt `Client`. Ten sa po poskytnutí správnych parametrov pripojí na InfluxDB a po výbere databázy s pozičnými záznamami vytvorí priame napojenie na databázu. Aby však nebolo nutné toto spojenie vykonávať manuálne pri získaní dát v každej metrike, v triede `SageChartDataService` som vytvoril metódu `getInfluxData($iql)`, ktorá ako parameter prijíma reťazec vo forme jazyka InfluxQL. Tento reťazec sa následne predá klientovi s napojenou databázou, ktorý s využitím metódy `query($iql)` získava požadované záznamy definované v požiadavke premennej `$iql`. Tieto záznamy sa vhodným spôsobom uložia do poľa a metóda `getInfluxData($iql)` ich vracia. Takýmto spôsobom sa získavanie dát v rámci kódu o niečo zjednoduší a skráti.

[3] https://github.com/influxdata/influxdb-python
[4] https://docs.influxdata.com/influxdb/v1.8/

7.1.1 Zmeny v HeatMap a SpaghettiMap

Metriky HeatMap a SpaghettiMap boli vhodným kandidátom na to, aby ich implementácia bola ako prvá prispôsobená na využívanie navrhnutého riešenia, a to hlavne z dôvodu, že si zmena implementácie nevyžadovala nejaké náročnejšie zmeny. Dôvod malého množstva zmien vychádza z princípu týchto metrík, ktoré pre svoje vykreslenie požadujú priamo pozičné dáta vo svojej základnej forme, a tak nedochádza k zložitému spracovaniu.

Obe tieto metriky využívajú k získaniu pozičných dát statickú metódu `getPositionHistory`, ktorá je súčasťou triedy `SageChartService`. Jednotlivé pozície sa potom získavajú metódou `getPositionsForPositionHistory` z triedy `SageChartDataService`.

```
1     $in = implode("|", $tagsFromOnePlan);
2     $from = $data["from"];
3     $to = $data["to"];
4     $iql = "SELECT posX, posY, posZ, time, feed_reference
5     FROM position_history
6     WHERE clr != '9998' AND time >= '$from' AND time <= '$to' AND feed_reference
          =~ /$in/ AND plan_reference = ".$planDetails[0]->id. "
7     ORDER BY time ASC";
8     $positionHistory = SageChartDataService::getInfluxData($iql);
9
10    return $positionHistory;
```

Výpis 7.3: Úsek kódu pre získanie pozičných dát v metóde getPositionHistory.

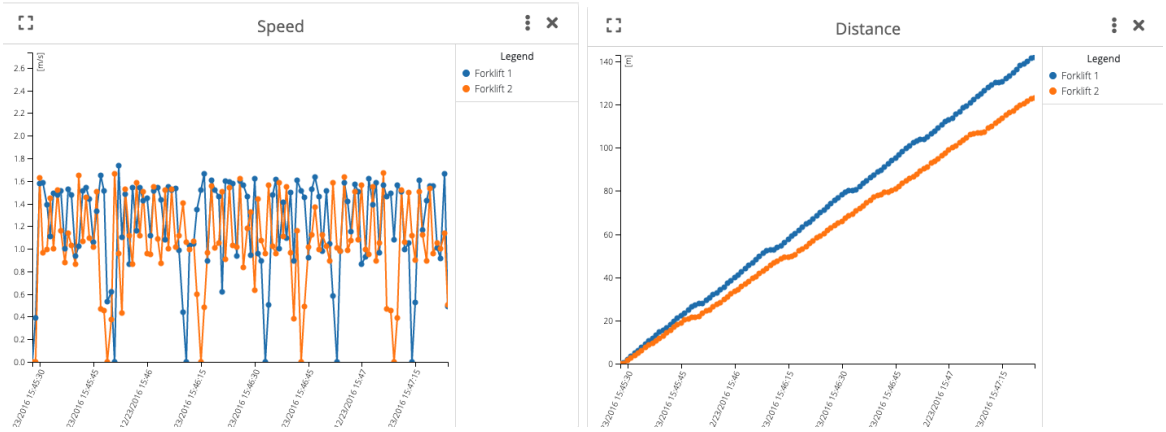
Ako je možné z výpisu 7.3 vidieť, pre zostavenie požiadavky bol využitý spomínaný jazyk InfluxQL. Už na prvý pohľad je vidieť jeho syntaktickú podobnosť, vďaka čomu bolo nutné vykonať len skutočne malé zmeny, aby syntax bola správna, a zároveň aby parametre zodpovedali modelu uloženia dát z InfluxDB.

Získané dáta si následne metriky spracovávajú nezávisle vo svojich metódach. V týchto metódach sa vytvárajú a inicializujú základné štruktúry, do ktorých sa potom získané pozičné dáta ukladajú v správnom formáte, ktorý je potrebný na ich vykreslenie. Na rozdiel od riešenia s využitím MySQL, kde sú získané dáta ukladané v objekte sú teraz dáta uložené v poli, a preto je potrebné prístup k získaným dátam jemne upraviť. To je všetko, čo je potrebné pozmeniť nato, aby metriky HeatMap a Spaghetti map fungovali s novo-navrhnutým riešením správne.

Výsledné dáta pripravené na zobrazenie potom môžu vyzerať následovne:

- **feeds**: ["t22", "t23", "t24"]
- **labels**: {t22: "Forklift 1", t23: "Forklift 2", t24: "Forklift 3"}
- **positions**:
  - t22: [{posX: "37.08", posY: "7.53", posZ: "1", at: "2022-03-23 14:45:28.499600"},...]
  - t23: [{posX: "30.67", posY: "3.50", posZ: "1", at: "2022-03-23 14:45:28.506809"},...]
  - t24: [{posX: "24.13", posY: "8.53", posZ: "1", at: "2022-03-23 14:45:28.513359"},...]

  kde feeds je označenie pre tagy vybraté užívateľom pre zobrazenie v metrike, labels sú patričné pomenovania tagov a positions sú pripravené pozičné dáta určené pre vykreslenie v metrike.



(a) Speed (b) Distance

Obr. 7.1: Zobrazenie metriky speed, ktorá znázorňuje vývoj rýchlosti pre vybrané tagy a metriky distance, ktorá zobrazuje vývoj prejdenej vzdialenosti vybranými tagmi.

7.1.2 Zmeny v Speed a Distance

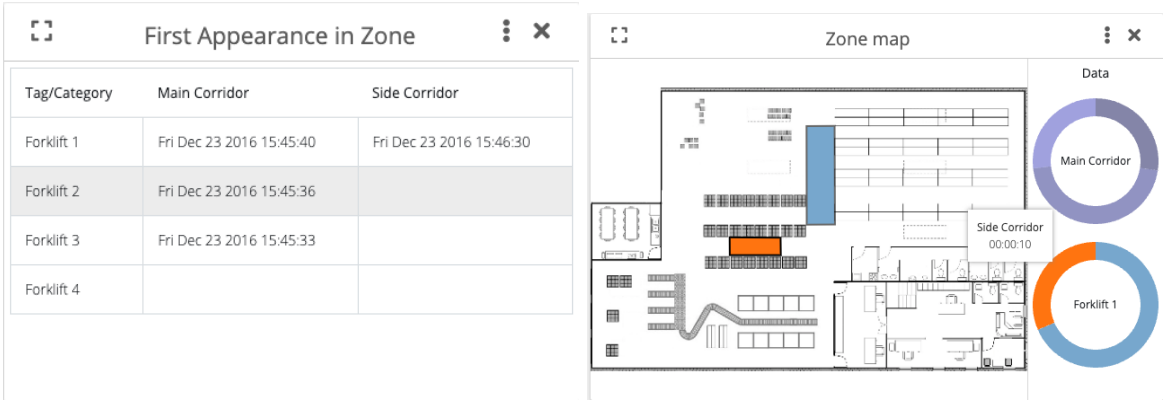
Metriky Speed a Distance sú založené na veľmi podobnom spracovaní. Líšia sa len spôsobom, akým sú dáta predané. Zatiaľ čo cieľom Speed je zistiť rýchlosť, v metrike Distance sa počíta vzdialenosť, ktorú jednotlivé tagy prešli.

Ako prvé sa v metódach určených na získanie a spracovanie pozičných záznamov získavajú hrany intervalov pre žiadané dáta. Tieto hrany sú potom určené na výpočet veľkosti intervalu, pre aký sa budú dáta ohľadom rýchlosti a vzdialenosti zobrazovať. V tomto prípade bolo využitie riešenia s InfluxDB výhodou, pretože poskytuje funkcie ako `FIRST()` a `LAST()`[5], ktoré sú určené na získanie prvého a posledného záznamu z požadovaných dát. Takýmto spôsobom nebolo nutné získané záznamy zoraďovať a požiadavka sa zostavila jednoduchšie a vykonala rýchlejšie.

Samotné získanie potrebných dát potom prebieha veľmi podobne ako v 7.1.1. Pretvorenie existujúceho riešenia na nové je pomerne jednoduché. Zostaví sa požiadavka pre získanie pozičných dát podľa parametrov zvolených užívateľom a získané dáta sa uložia do poľa. Opäť za využitia jazyka `InfluxQL`. Čo ale bolo potrebné vyriešiť je formát získaného času z pozičného záznamu. Časové známky z `InfluxDB` prichádzajú v **ISO 8601**[6] formáte. Na to aby nevznikala nekonzistencia medzi dátami z `MySQL`, a zároveň pre správne fungovanie PHP funkcie `strtotime` som vytvoril jednoduchú funkciu, ktorá časový záznam previedla do vhodného formátu pre spracovanie **YYYY-MM-DD hh:mm:ss**. S takto pozmeneným časovým údajom sa už potom dalo jednoducho postupovať, ako pri existujúcom riešení. Časový záznam je potom prevedený do UNIX formátu, s využitím ktorého sa vyfiltrujú príliš rýchle skoky, ktoré nastali bezprostredne za sebou v krátkom okamihu, čím sa predchádza možným nepresnostiam.

Ako posledné bolo nutné už len vhodným spôsobom pristúpiť k získaným dátam. Dáta sa potom vhodným a jednoduchým formátom (čas, hodnota) reprezentujú vo webovej aplikácii (viď. 7.1). Kde záznamy v tomto formáte môžu vyzerať napríklad takto: {time: "2022-02-23 14:45:30.0", value: 1.96355049434052}

[5] https://docs.influxdata.com/influxdb/cloud/query-data/flux/first-last/
[6] https://www.rfc-editor.org/rfc/rfc3339



(a) First appearance

(b) Zone Map

Obr. 7.2: Zobrazenie metríky First appearance a Zone mapetríky First appearance a Zone map.

7.1.3 Zmeny v zónových metrikách

Jedny z náročnejších zmien, ktoré bolo potrebné vykonať boli implementačné zmeny pre zónové metriky. Medzi tieto metriky patria **first_appearance**, **last_appeatance** a **zone map** (obrázok 7.2). Dôvodom je práve absencia už spomínanej funkcie `ST_CONTAINS`, ktorá je súčasťou MySQL. To, čo táto funkcia dokáže (zistiť, či sa pozičný bod nachádza v zóne), tak bolo nutné nahradiť priamo pri spracovaní. Keďže pri InfluxDB nie sme schopní zistiť priamo pomocou databázy, či sa daný bod v zóne nachádza, je potrebné celý tento proces rozdeliť na 3 časti:

1. Získanie zóny (polygón so súradnicami)
2. Získanie pozičných dat
3. Zistenie, či sa bod nachádza v zóne (polygóne)

Získanie zóny

Získanie zóny prebieha využitím jednoduchej požiadavky na databázu podľa zóny, ktorú si užívateľ vybral. Vzhľadom na fakt, že zónové údaje niesú rozsiahle nebolo nutné ich migrovať do InfluxDB, a tak sa tieto údaje získavajú z MySQL. Na rozdiel od existujúceho riešenia bolo nutné samotné získanie zóny oddeliť od získavania pozičných dát. Samotná požiadavka s využitím funkcie `ST_AsText` zaistí, že zóna sa získa priamo v textovej forme POLYGON(x_1 y_1, x_2 y_2, ...)[7], kde x_n a y_n sú súradnice polygónu.

Získanie pozičných dat

Pozičné dáta sa získavajú rovnako ako aj pri ostatných metrikách, zostaví sa InfluxQL požiadavka, a potom sa s využitím metódy `getInfluxData($iql)` získajú pozície jednotlivých bodov.

[7]Príklad polygónu: POLYGON((54.533 24.234,54.513 32.096,57.593 32.096,57.552 24.214,54.533 24.234))

Zistenie, či sa bod nachádza v zóne

Aby sme však boli schopní určiť relevantné body pre danú zónu, je potrebné zistiť, či sa získané pozičné dáta v zóne nachádzajú. To sme veľmi jednoducho schopní zistiť s využitím knižnice `phpgeo`[8]. Jedná sa o knižnicu, ktorá poskytuje možnosti pre výrazné zjednodušenie práce s pozičnými bodmi, súradnicami a geografickými výpočtami. Pomocou nej sme schopní zo získanej zóny vytvoriť abstrakciu polygónu vytvorením objektu `Polygon()` a využitím `Coordinates()` pozičné body zóny uložíme ako súradnice. Tieto súradnice potom priradíme k objektu `Polygon()`, čím definujeme jeho rozmery. Celý tento proces zahŕňa vytvorená metóda `createPolygon`. Potom už jednoduchou iteráciou cez získané pozičné záznamy zistíme, či sa bod z pozičného záznamu nachádza v zóne.

```
1     foreach ($positions as $position){
2         $geo_point = new Coordinate($position["posX"], $position["posY"]);
3         if ($polygon->contains($geo_point)){
4             $isInside = true;
5             break;
6        }
7     }
```

Výpis 7.4: Úsek kódu z metódy pre spracovanie údajov pre metriku first_appearance. Iterácia cez pozičné body a zistenie, či sa bod nachádza v zóne (polygóne).

7.1.4 Zmeny pre Activity a Attendance

Podobne ako pri zónových metrikách aj pre Activity (obrázok 7.3) a Attendance bolo nutné vykonať výraznejšie zmeny. Tieto metriky nielen, že zisťujú pozície pre užívateľom vybrané zóny (ako pri zónových metrikách), ale zároveň ich zoskupujú po skupinách, ktorých veľkosť sa volí podľa rozsahu časového intervalu pre pozičné dáta zvolené užívateľom.

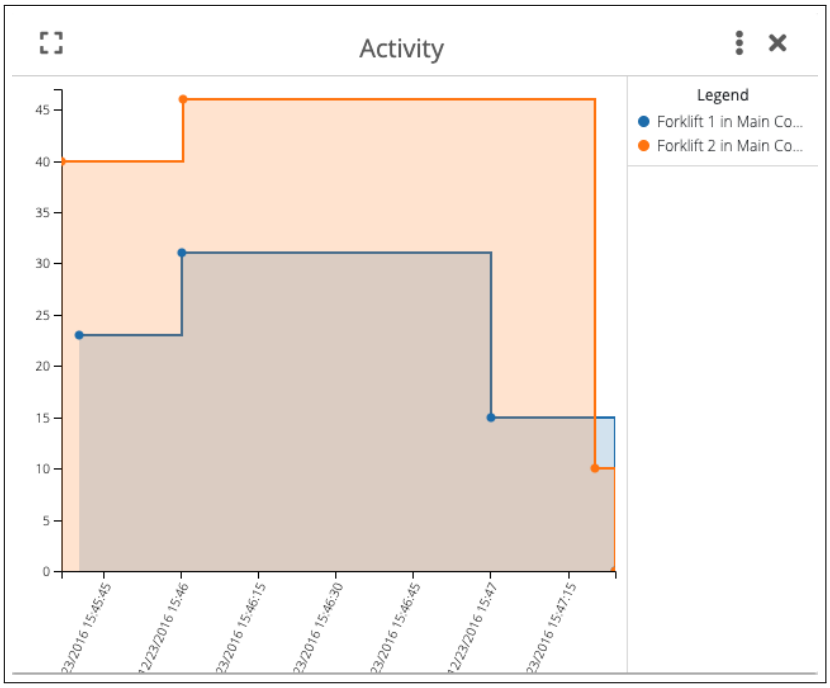
Celý tento priebeh v existujúcom riešení prebieha za využitia jedného zložitého príkazu, kedy sa celá práca vykoná prostredníctvom MySQL.

Vzhľadom už na spomínaný fakt, že pri InfluxDB nie sme schopní použiť niektoré funkcie alebo spôsoby zoskupenia, je nutné toto spracovanie rozdeliť a získané dáta priamo spracovať.

1. Získanie zóny a vytvorenie polygónu
2. Získanie pozičných záznamov
3. Zoskupenie záznamov pre body ležiace v zóne

Získanie zóny a vytvorenie polygónu prebieha veľmi podobne ako pri zónových mapách v 7.1.3. Využítím knižnice `phpgeo`, vytvoríme objekty pre polygón a jemu priradené súradnice. Rovnako jednoducho ako pri všetkých metrikách využitím príkazu zostaveného v `InfluxQL` a metódy `getInfluxData($iql)` získame pozičné záznamy. Výraznejšie zložité zmeny a špeciálny prístup si vyžadovalo zoskupenie záznamov pre body ležiace v zóne.

[8] https://github.com/mjaschen/phpgeo



Obr. 7.3: Zobrazenie metriky Activity, kde môžeme vidieť hodnotu znázoňujúcu počet výskytov v danej zóne pre časový interval.

Zoskupenie záznamov pre body ležiace v zóne

V existujúcom riešení sa záznamy zoskupujú pomocou príkazu `GROUP BY` a `div` pre zvolenie veľkosti intervalu jednej skupiny. Podobným spôsobom by to bolo možné aj s využitím `InfluxDB`, v tej, ale toto zoskupenie funguje inak, a preto bolo nutné získané dáta zoskupiť priamo pri spracovaní.

Pred samotným vytvorením skupín je ešte nutné zistiť, či sa bod nachádza v zvolenej zóne, a to rovnakým spôsobom ako je ukázané vo výpise 7.4.

Následne v cyklus prechádza postupne cez všetky pozičné záznamy, ak sa bod v zóne nachádza spracujeme ho. Body zoskupujeme podľa veľkosti intervalu vypočítanej podľa rozsahu času pre aký si užívateľ metriku vyžiadal. To znamená, že z každého takého intervalu si uložíme prvý a posledný bod, pričom práve tie daný interval označujú. Spolu s týmto intervalom si uložíme aj počet výskytov vybraného tagu (v prípade metriky Attendance si ukladáme len informáciu o tom, či sa tag v zóne počas intervalu nachádzal) v zadanej zóne počas jedného intervalu. Takto získané intervaly s ich hodnotami sa potom už len vhodným spôsobom uložia do požadovanej štruktúry.

Výsledné dáta tak môžu vyzerať následovne:

- time: "2022-02-23 14:45:40"
- timeto: "2022-02-23 14:45:59"
- value: "21"
- time: "2022-02-23 14:46:00"
- timeto: "2022-02-23 14:46:45"
- value: "31"

7.2 Testovanie

Implementácia navrhnutého riešenia prinesie so sebou výsledky výrazne rýchlejším spôsobom, ako poskytuje existujúce riešenie. Potrebné však je, aby pri využívaní nového riešenia nedochádzalo k zmene samotných dát alebo jej formy. Dáta, ktoré po vykonaní implementačných zmien pri určitých parametroch získame sa musia zhodovať s dátami už existujúceho riešenia pri rovnakých parametroch. Práve o to sa stará vytvorenie jednoduchých testov, a ich následne využitie pre kontrolu výstupu.

Samotné testovanie je ideálne vytvoriť už pred implementáciou, a to z toho dôvodu, aby sme s jednoduchosťou dokázali určiť, či vykonané zmeny neovplyvňujú samotné dáta.

7.2.1 Priebeh testov

Samotné testy boli vytvorené v triede `TestSageChartService`. Táto trieda dostane na vstupe typ metriky, pre ktorú sa majú testy previesť. Všetky testy majú predom definované vstupné dáta, ktoré sa postupom rozširujú už iba o počet tagov. Testov sa v rámci testovania jednej metriky vykoná niekoľko, a to zmenou časového intervalu alebo počtom tagov. Tým sa zaručí väčšia robustnosť vykonaných zmien.

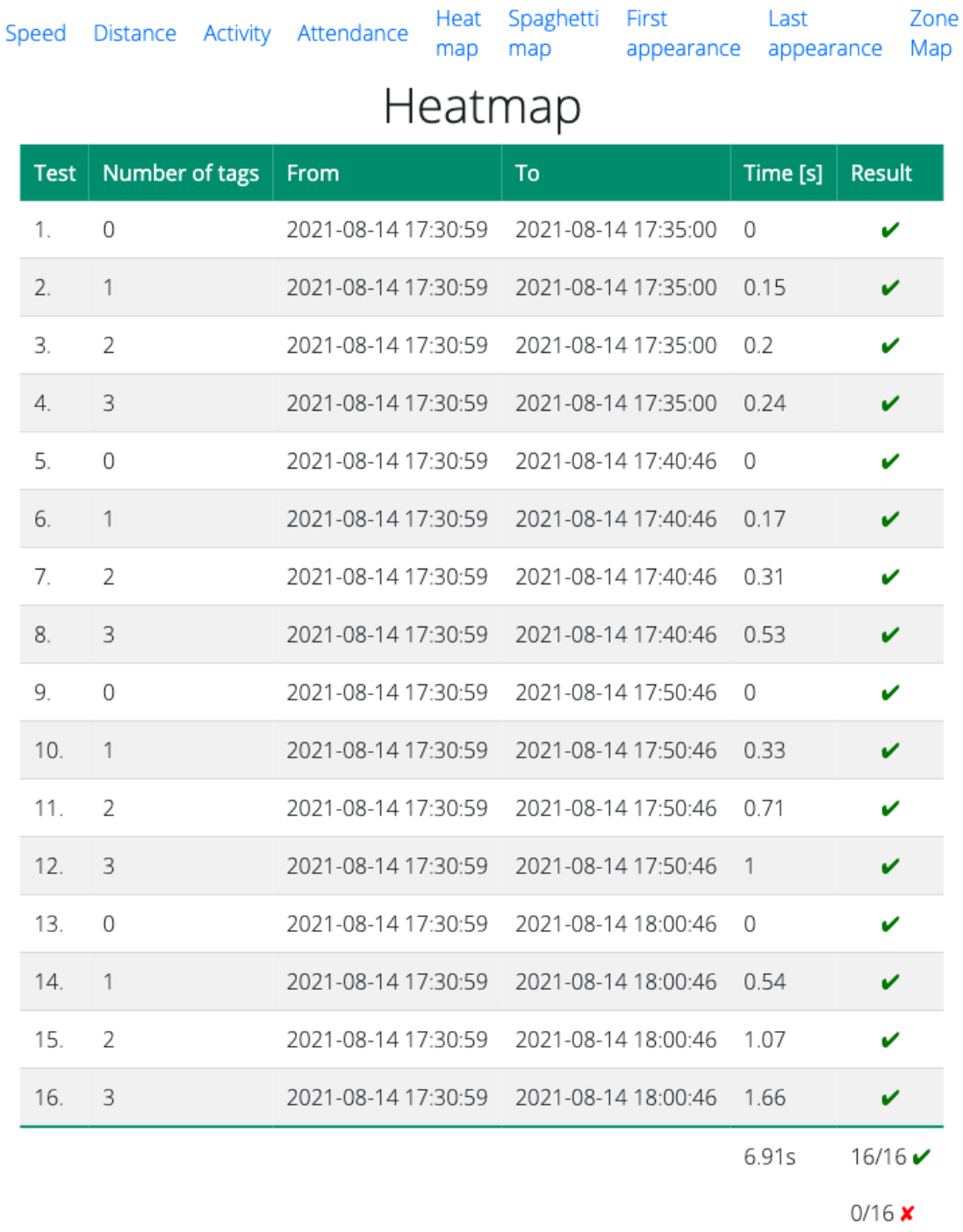
Každý test zo skupiny testov pre metriku vyzerá následovne:

1. Získanie objektu s požadovanými dátami pre InfluxDB
2. Získanie objektu s požadovanými dátami pre MySQL
3. Porovnanie získaných objektov

Samotný priebeh testu je veľmi jednoduchý. Pre vstupné dáta (vybranú metriku, tagy, zóny a časový interval) test zavolá metódu `createChart` triedy `SageChartService`. Tá pre vstup vráti adekvátne dáta na to, aby bolo možné vykresliť graf na strane klienta. Pred samotným zavolaním tejto metódy sa nastaví statická premenná `$USE_INFLUX` typu `boolean` triedy `SageChartDataService`, ktorou definujem, či sa pri získaní a spracovaní dát má využiť existujúce riešenie s využitím MySQL (pre hodnotu `false`), alebo sa má využiť novo-navrhnuté riešenie a pozičné dáta z InfluxDB (pre hodnotu `true`). Potom čo takýmto spôsobom získame objekty obsahujúce požadované dáta, celé tieto objekty porovnáme. V prípade, že sú objekty obsahovo rovnaké vieme, že využitie ktorejkoľvek z metód vracia správny výsledok. Ak sa však objekty nezhodujú znamená to, že využitím nového riešenia dochádza k poskytnutiu nesprávnych dát, a je potrebné takúto zmenu skontrolovať.

7.2.2 Zobrazenie testov

Pre zobrazenie testu som pridal, k webovej aplikácií endpoint "/test", kde po presmerovaní si užívateľ vyberie jednu z ponúkaných metrík. Následne sa mu vo formáte tabuľky zobrazia vykonané testy pre vybranú metriku. Každý riadok tabuľky patrí jednému testu, kde pre každý test je špecifikované, pre koľko tagov a aký časový interval prebehol a koľko času mu celý test zabral. V poslednom stĺpci je označenie, či test prebehol správne alebo chybne. Na konci tabuľky je zosumarizovaný čas a vyhodnotenie všetkých testov. Príklad vyhodnoteného testu je možné vidieť na obrázku 7.4.



Obr. 7.4: Snímok vytvoreného užívateľského rozhrania pre vyhodnotený test.

Kapitola 8

Vyhodnotenie a experimenty

Táto kapitola sa zoberá vyhodnotením vykonaných zmien v tejto diplomovej práci na nástroji `Sage Analytics` systému RTLS spoločnosti Sewio. Existujúce riešenie neposkytovalo dosť uspokojivé spôsoby vizualizácie dát pomocou rôznych metrík, a to v zmysle dlhého načítania. Z tohto dôvodu som v kapitole 5 prešiel niekoľko spôsobov, akými by bolo možné celý proces zobrazenia pozičných dát formou metrík zrýchliť. Výraznejšie zrýchlenie poskytla optimalizácia súčasného riešenia MySQL databázy. Ešte o čosi lepšie výsledky prinieslo využitie databázy časových rád **InfluxDB**. To si však vyžadovalo úpravy pri získavaní a spracovaní pozičných údajov v backendovej časti webovej aplikácie.

Aby bolo možné konečnú snahu a prácu na tejto diplomovej práci vyhodnotiť, táto kapitola sa zaoberá porovnaním a vyhodnotením výsledkov zmien nadobudnutých v tejto práci. Poukazuje na to, ako veľmi, a v akom prípade došlo k zlepšeniu v rámci rýchlosti načítania jednotlivých metrík.

Všetky merania obsahujú 3 základné parametre, medzi ktorými meranie prebiehalo.

1. **Pôvodné riešenie** – existujúce riešenie v súčasnosti využívané v RTLS.
2. **Optimalizované riešenie** – riešenie využívajúce MySQL, v tejto práci optimalizované.
3. **Nové riešenie** – riešenie navrhnuté v tejto práci, využívajúce InfluxDB.

Vyhodnotenia pre všetky metriky a riešenia sú vykonávané na reálnych dátach zozbieraných RTLS spoločnosi Sewio. Dáta obsahujú viac ako týždeň zozbieraných pozičných dát, čo predstavuje viac ako 137 miliónov pozičných záznamov.

Merania sú vykonávané na mojom osobnom počítači[1], a preto sa môžu v závislosti od výkonnosti stroja značne líšiť (predpokladá sa, že rýchlosť vyhotovenia metrík bude na produkčnom serveri ešte väčšia). Z tohto dôvodu nie je až tak dôležitá priama rýchlosť spracovania, ako percentuálne zrýchlenie medzi riešeniami.

V 8.1 je popísané celkové vyhodnotenie vykonaných zmien. Jedná sa o porovnanie vyobrazenia metrík pre pôvodné, optimalizované a nové riešenia navrhnuté v tejto práci. Sekcia 8.2 sa zaoberá vyhodnotením každej metriky zvlášť, a to pre rôzne časové intervaly. Zisťuje, aký vplyv má veľkosť intervalu na metriku pri pôvodnom, optimalizovanom a novonavrhnutom riešení. Posledná časť kapitoly 8.3 skúma, rôzne vplyvy na rýchlosť metrík a poukazuje, čo má na vyhotovenie metrík najväčší vplyv.

[1]CPU: 2,8 GHz 4-jadrový Intel Core i7, RAM: 16 GB 2133 MHz LPDDR3

8.1 Celkové vyhodnotenie

Už pri meraniach v kapitole 5 bolo jasne vidieť, aký výrazný vplyv má získavanie dát pri využití, či už optimalizovaného pôvodného riešenia, alebo novo-navrhnutého riešenia. Otázne však ostávalo, aký veľký, a či vôbec bude mať zmena uloženia dát vplyv na rýchlosť vzhľadom na to, že pre využitie nového riešenia bolo nutné vykonať implementačné zmeny, ktoré samotný čas môžu zase ovplyvniť. Táto sekcia preto poukazuje akým spôsobom a ako výrazne došlo k zrýchleniu celého procesu získania spracovaných dát pre metriky, posielaných zo strany servera na klienta. Jedná sa o akýsi súhrn, ktorý poskytuje základný prehľad výsledkov dosiahnutých po vykonaní zmien v tejto práci.

metrika	**pôvodné [s]**	**optimalizované [s]**	**nové [s]**	**zrýchlenie [%]**
Speed	144,23	11,77	8,54	1125/**1588**
Spaghetti map	55,69	16,86	6,33	230/**780**
First Appearance	53,73	9,92	6,51	442/**725**
Last Appearance	59,15	10,58	6,54	459/**804**
Zone map	60,00	12,81	7,19	368/**734**
Heatmap	57,59	20,33	6,58	183/**775**
Distance	144,21	11,00	6,60	1211/**2085**
Activity	66,12	18,99	7,75	248/**753**
Attendance	72,10	19,91	7,54	262/**856**

Tabuľka 8.1: Porovnanie spracovania jednotlivých metrík pre 3 riešenia: pôvodné, optimalizované a nové. Spracovanie metriky prebiehalo pre časový interval 24 hodín a pre 1 zariadenie (tag), prípadne 1 zónu. Posledný stĺpec poukazuje na zrýchlenie dosiahnuté využitím optimalizovaného/nového riešenia oproti pôvodnému.

Z tabuľky 8.1 je možné pozorovať zmeny dosiahnuté využitím optimalizovaného a nového riešenia v porovnaní s pôvodným. Výrazné zmeny sú pri každej metrike, a to pri oboch modifikáciách pôvodného riešenia.

Najväčšie zrýchlenie je možné vidieť pri využití novo-navrhnutého riešenia s využitím `InfluxDB`. Toto riešenie signifikantným spôsobom prekonalo pôvodné, a to pri každej z pozorovaných metrík. Najnižšie, no stále významné zrýchlenie je vidieť pri metrike `Zone Map` a to o **734 %**. Najmarkantnejšie zrýchlenie je pri metrike `Distance`, kedy sa čas oproti pôvodným 144 sekundám zmenšil na 6,60 sekúnd, čo predstavuje zrýchlenie až o **2085 %**.

Veľmi dobré výsledky prinieslo aj optimalizované riešenie. To však za novým zaostáva. V niektorých prípadoch je vidieť, že nové riešenie dosahuje dokonca až 3 krát väčšie zlepšenie ako optimalizované. Aj napriek tomu stojí určite za zmienku, a to vzhľadom na fakt, že oproti novému riešeniu si toto nevyžadovalo žiadne implementačné zmeny. Najmenšie zrýchlenie je v tomto prípade možné pozorovať pri metrike `Heatmap`, kedy dochádza k 183 % zrýchleniu. Najväčšie je zase pri metrike `Distance`, kedy dochádza k zlepšeniu až o 1211 %.

Už z tohto celkového vyhodnotenia je možné vidieť, že navrhnuté riešenia priniesli naozaj značné zmeny a sú skvelou alternatívou pre pôvodné. Tieto merania boli však vykonané pre

časový interval 24 hodín. Aby bolo možné objektívne zhodnotiť, že sú výsledky také dobré bez ohľadu na časový interval, v sekcii 8.2 je prehľad metrík a rýchlosť jej vyhotovenia pre všetky 3 riešenia, a to pre niekoľko časových intervalov.

8.2 Vyhodnotenie metrík

Výsledky z celkového vyhodnotenia vyzerajú skutočne zaujímavo, otázkou ostáva ako sa bude správať vyhotovenie metrík pre rôzne časové intervaly pre každú z metrík, a či nenastane situácia, kedy by bolo pôvodné riešenie lepšie ako navrhnuté. Týmito výsledkami zároveň zistíme, pre ktorú metriku je možné zobraziť dáta aj v rámci niekoľkých dní, a či čakacia doba na vyhotovenie metrík nebude pre užívateľa neadekvátnou.

Ako už bolo spomenuté v kapitole 7, niektoré metriky majú získavanie a spracovanie dát veľmi podobné, takmer až identické. Z tohto dôvodu pri vyhodnocovaní výsledkov som takéto metriky spojil do jednej skupiny, respektíve jedného vyhodnotenia. Metriky v takejto skupine dosahujú vyhotovenia vo veľmi podobnom čase, a tak nie je dôvod sa na nich pozerať špeciálne zvlášť.

Skupiny som rozdelil podobne ako pri implementácii teda:

- Heatmap a Spaghetti map
- First appearance a Last appearance
- Zone map
- Activity a Attendance
- Distance a Speed

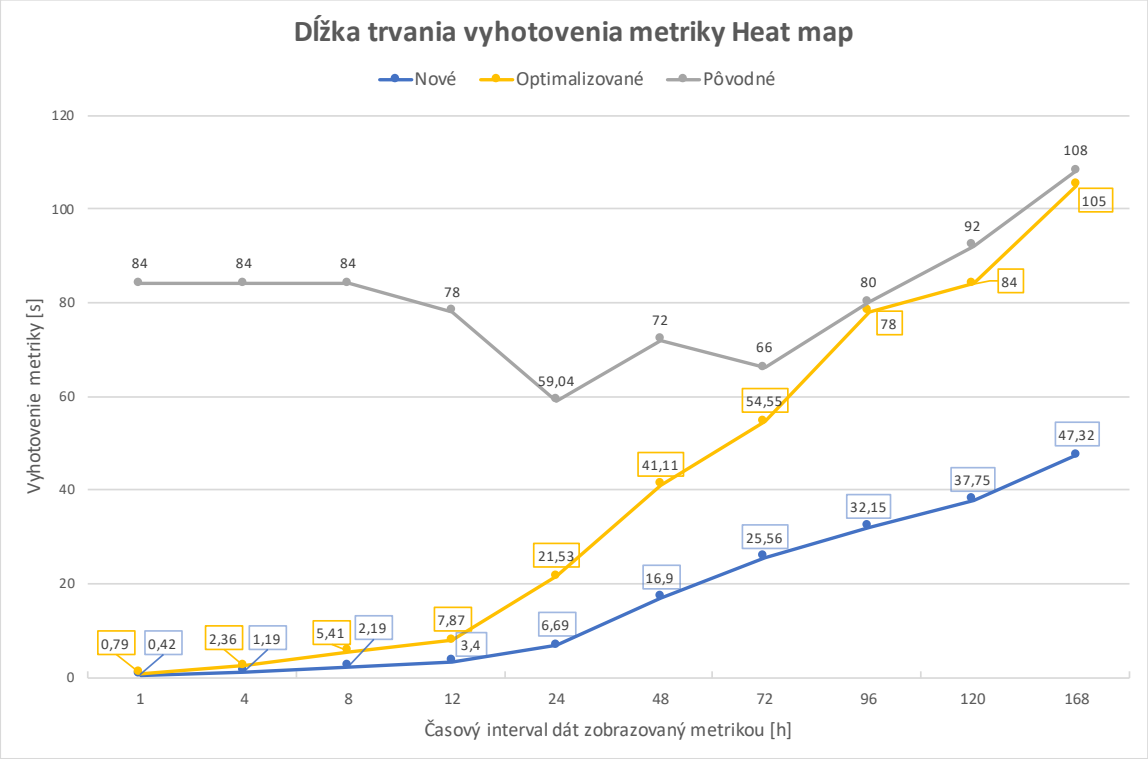
Merania v tejto sekcii sú vykonané podľa časového intervalu. Týmto intervalom je ukázané, pre akú dobu sme dáta pomocou metrík schopní zobraziť a v akom čase.

V každom vyhodnotení boli merania vykonávané na 10 časových intervaloch pre 1 konkrétny objekt (tag) a to:

1. **1 hodina** – 29 076 záznamov
2. **4 hodiny** – 117 001 záznamov
3. **8 hodín** – 234 181 záznamov
4. **12 hodín** – 350 778 záznamov
5. **1 deň** – 702 457 záznamov
6. **2 dni** – 1 404 455 záznamov
7. **3 dni** – 2 106 298 záznamov
8. **4 dni** – 2 804 728 záznamov
9. **5 dni** – 3 441 035 záznamov
10. **1 týždeň** – 4 243 814 záznamov

Tieto časové intervaly boli vybraté tak, aby poskytli čo najväčší prehľad pri vyhodnocovaní metrík v závislosti na čase.

8.2.1 Heat & Spaghetti map



Obr. 8.1: Graf znázorňujúci dĺžku trvania vyhotovenia metriky Heatmap (Spaghetti map), pre rôzne časové intervaly dát, zobrazovaných metrikou od 1 hodiny až po 1 týždeň. Porovnané sú 3 spôsoby riešenia.

V grafe 8.5 môžeme pozorovať vývoj rastu dĺžky vyhotovenia metriky s rastom časového intervalu zobrazovaného metriky tromi spôsobmi riešenia.

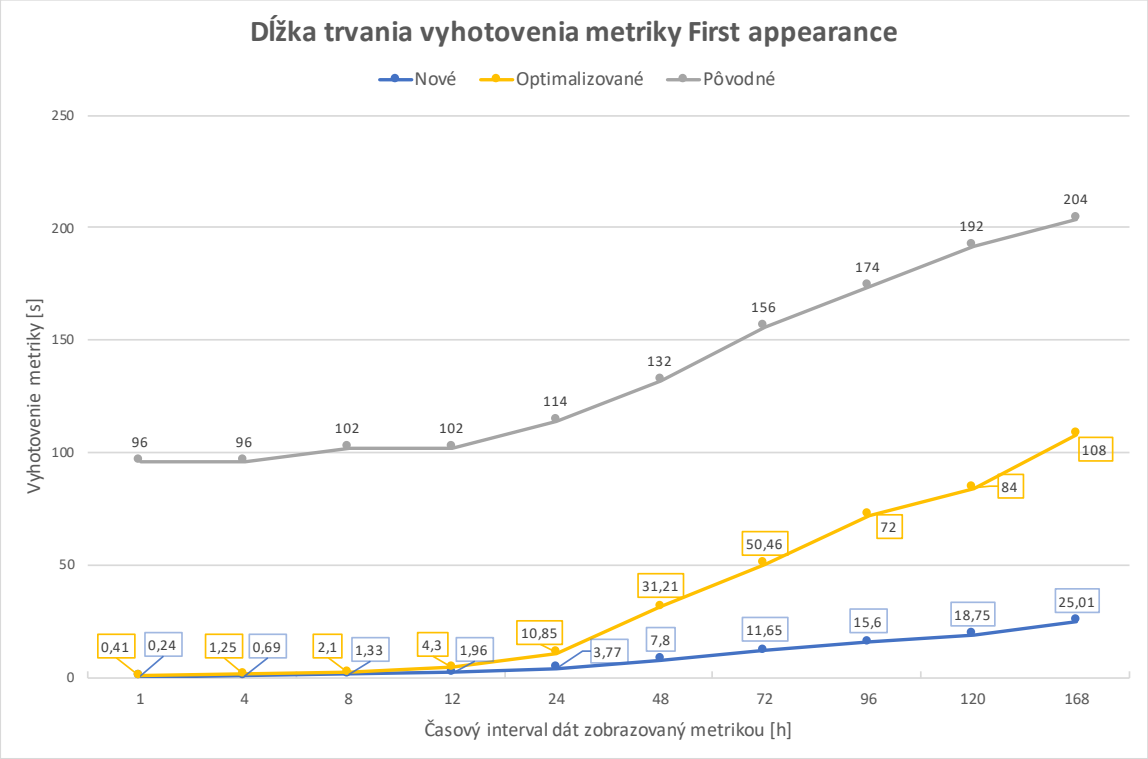
Pri pôvodnom riešeni vidíme, že absolútne nezohľadňuje dĺžku časového intervalu, a tak je jasné, že toto riešenie nie je vôbec pre pozičné dáta s časovými údajmi optimalizované. Postupný nárast je možný pozorovať až asi od polovice grafu.

Optimalizované riešenie ukazuje, ako a kde optimalizácia pomohla. Výrazným spôsobom sa zrýchlilo vyhodnotenie pri malých časových intervaloch. S postupným zväčšovaním intervalu však výhoda optimalizovaného riešenia klesá, až napokon okolo veľkosti intervalu 3 - 4 dni sa toto riešenie vyrovnáva pôvodnému.

Markantný rozdiel prináša nové riešenie, kedy môžeme pozorovať ako vhodné je toto riešenie optimalizované pre pozičné dáta. Pri nízkych časových intervaloch je len o čosi lepšie, ako optimalizované riešenie, no s postupným nárastom časového intervalu vidíme, ako sa rozdiel týchto dvoch riešení značným spôsobom zväčšuje, a keď pri intervale 12 hodín je rozdiel vyhotovenia 4,5 sekundy, pri intervale 1 týždňa (168 hodín) je to už takmer minúta.

Metriky Heatmap a Spaghetti map sa tak stávajú použiteľnými aj v prípade zobrazenia pre 1 celý týždeň. Ich vyhotovenie pri tomto časovom rozpätí trvá síce viac ako 47 sekúnd čo je doba, ktorá môže byť pre užívateľa stále trochu viac ako by si predstavoval, no dá sa povedať, že je adekvátnou vzhľadom na fakt, že ide o zobrazenie viac ako 4 miliónov pozičných dát.

8.2.2 First & Last appearance



Obr. 8.2: Graf znázorňujúci dĺžku trvania vyhotovenia metriky first a last appearance, pre rôzne časové intervaly dát, zobrazovaných metrikou od 1 hodiny až po 1 týždeň. Porovnané sú 3 spôsoby riešenia.

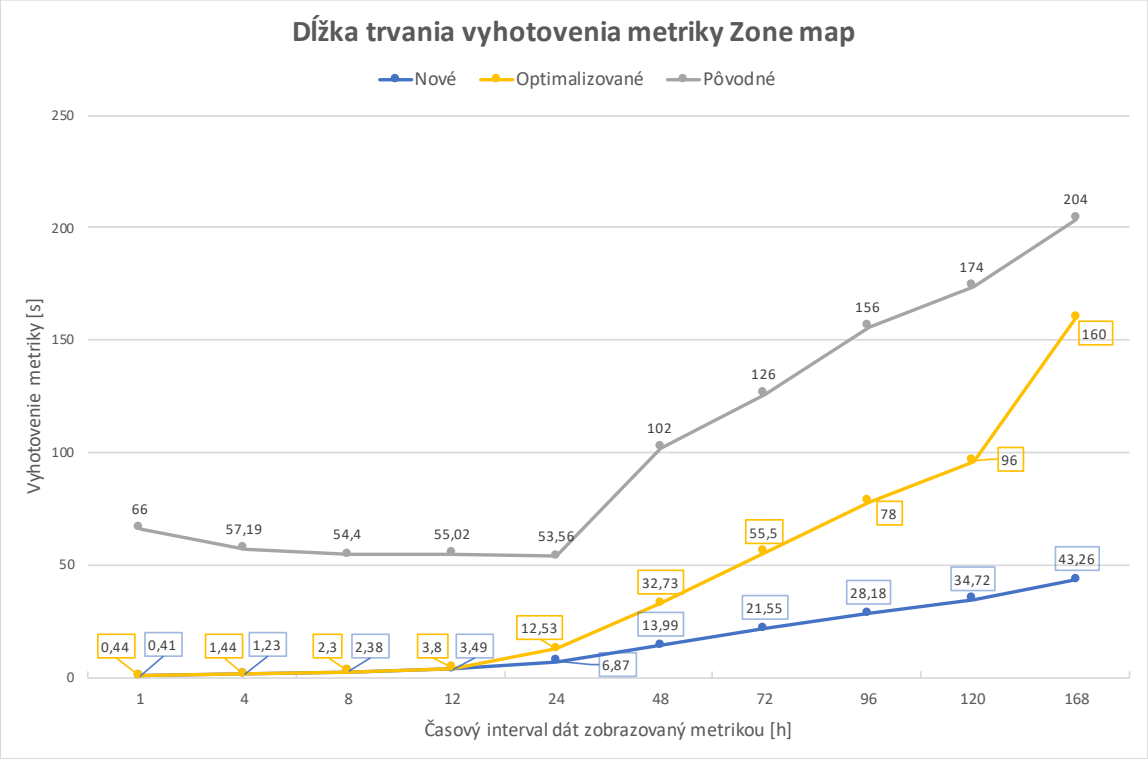
Využitie či už optimalizovaného, alebo nového riešenia prináša značne lepšie výsledky aj pri metrikách `First appearance` a `Last appearance`. V tomto prípade však ide ešte o čosi markantnejšie rozdiely.

Ako je možné vidieť v grafe 8.2 optimalizované riešenie pri tomto type metriky v porovnaní s pôvodným prináša rýchlejšie vyhodnotenie pre všetky časové intervaly. Zatiaľ čo pri malých hodnotách intervalu dosahuje rovnako dobre výsledky ako v 8.2.1, tentoraz je toto riešenie výhodné aj pre niekoľko dňové intervaly. Dá sa povedať, že výkonnosť tohto riešenia v porovnaní s pôvodným ostáva rovnaká, keďže pri intervale 1 hodiny je rozdiel okolo 96 sekúnd, pri intervale 1 týždňa sa tento rozdiel výrazne nezmenil a pohybuje sa rovnako.

Pre nové riešenie je jasné vidieť, že opäť dosahuje najlepšie výsledky v porovnaní s ostatnými, v tomto prípade ešte o čosi výraznejšie. Zatiaľ čo pri nízkych časových intervaloch je doba vyhotovenia metriky v porovnaní s pôvodným riešením približne 96 sekúnd, s postupným nárastom intervalu sa tento rozdiel ešte viditeľnejšie zväčšuje, čo zase predstavuje jasný dôkaz toho, ako dobre je databáza `InfluxDB` na tento typ dát optimalizovaná, a ako značné je zlepšenie s jej použitím, a to aj napriek tomu, že nedisponuje funkciami, ktoré pri pôvodnom a optimalizovanom riešení zónových metrík využívané boli a pri novom ich bolo potrebné implementovať.

V porovnaní s optimalizovaným riešením, nové dosahuje opäť lepšie výsledky, ktoré sú s nárastom časového intervalu ešte výraznejšie.

8.2.3 Zone map



Obr. 8.3: Graf znázorňujúci dĺžku trvania vyhotovenia metriky Zone map, pre rôzne časové intervaly dát, zobrazovaných metrikou od 1 hodiny až po 1 týždeň. Porovnané sú 3 spôsoby riešenia.

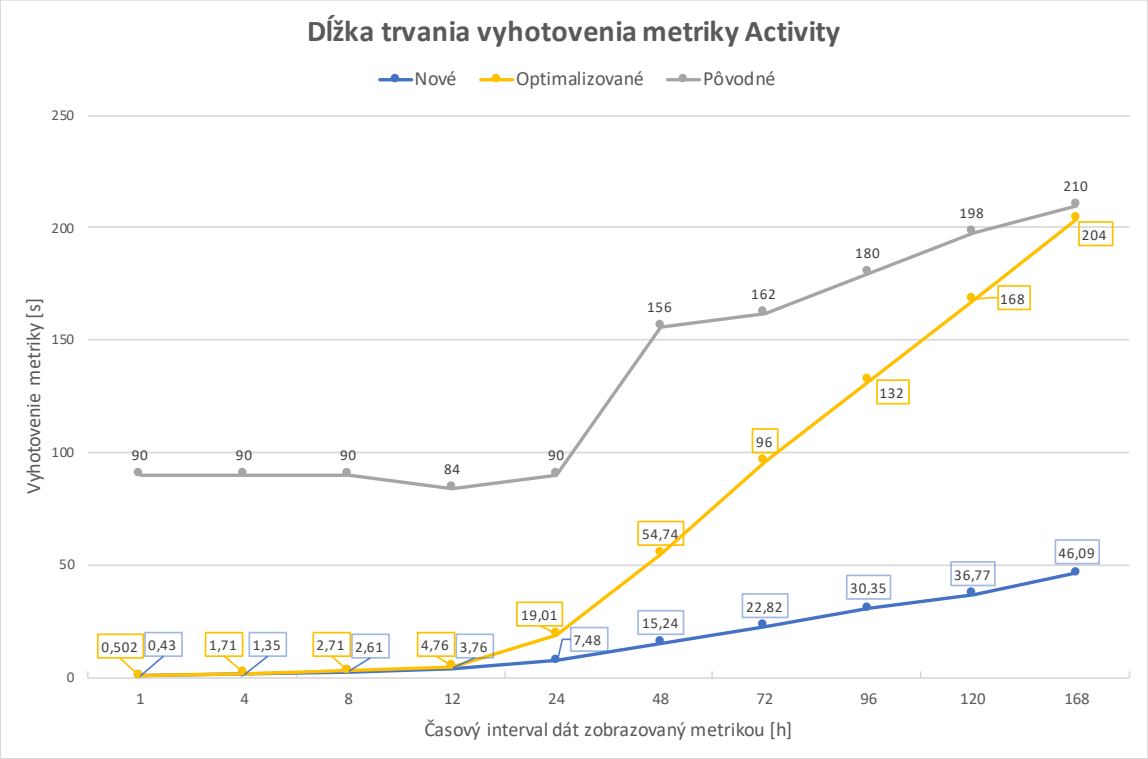
Zónová metrika podobne ako first a last appearance metriky, filtruje záznamy, ktoré ležia v zónach no ich spracovanie prebieha inak, čo bolo jemne viditeľné aj na rozdielnych výsledkov.

Optimalizované riešenie prinieslo aj pri tejto metrike zlepšenie primárne pri nižných časových intervaloch, kedy je čas vyhotovenia veľmi krátky – len niekoľko sekúnd, no pri pôvodnom riešení takmer minúta. S narastajúcim intervalom sa táto výhoda zmenšuje, avšak aj pri 1 týždni je stále výhodnejšie optimalizované riešenie, ktoré je o 40 sekúnd rýchlejšie.

Jasným víťazom je aj pri tejto metrike nové riešenie, ktorému sa opäť to optimalizované približuje pri intervale do 1 dňa, avšak potom sú už výsledky značne v prospech nového riešenia, čo vidíme ak sa pozrieme na graf 8.3, že pri intervale 24 hodín je rozdiel len 6 sekúnd, pri intervale 1 týždňa sa toto rozpätie ešte zväčší a rozdiel predstavuje takmer 120 sekúnd, čiže 2 minúty.

Porovnanie nového riešenia s pôvodným je rovnako ako aj pri ostatných metrikách výrazné, no narozdiel od metriky Heatmap, je možné pozorovať, že nové riešenie má ešte miernejšiu krivku stúpania, čo naznačuje, že so zväčšujúcim sa intervalom bude dochádzať ešte k väčším rozdielom, a to či už s pôvodným riešením, alebo optimalizovaným (rovnakú situáciu je možné pozorovať aj pri metrikách first a last appearance).

8.2.4 Activity & Attendance



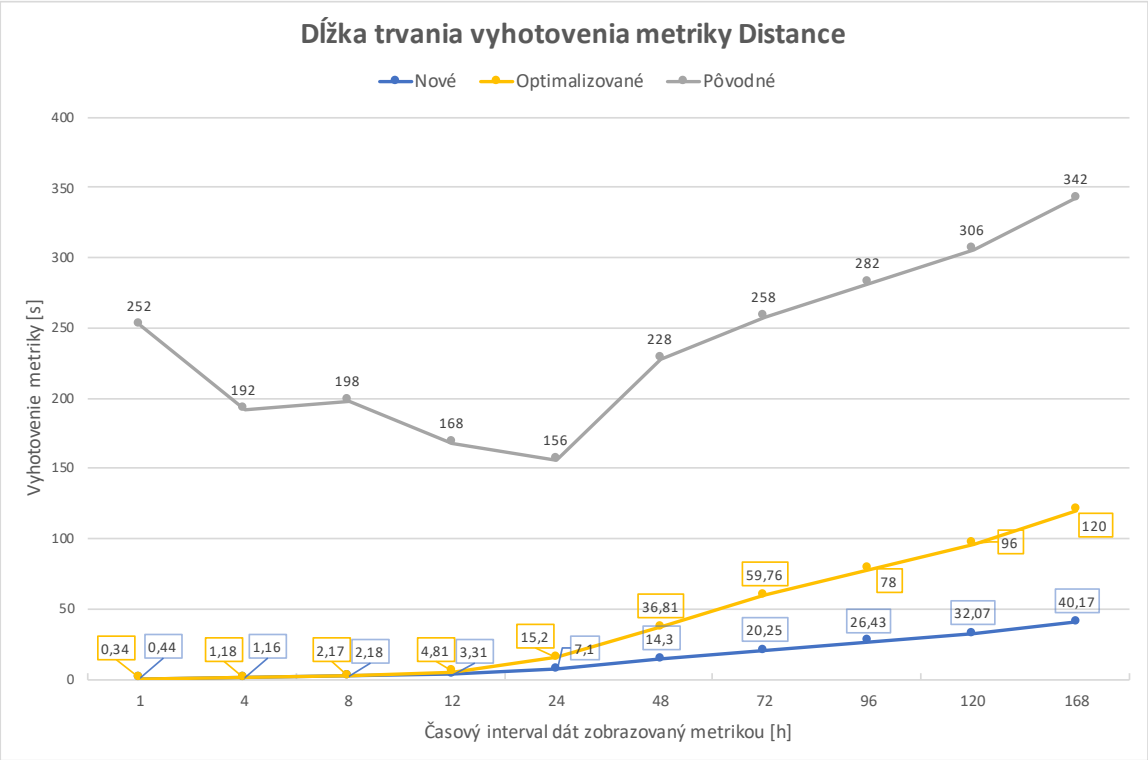
Obr. 8.4: Graf znázorňujúci dĺžku trvania vyhotovenia metriky Activity a Attendance, pre rôzne časové intervaly dát, zobrazovaných metrikou od 1 hodiny až po 1 týždeň. Porovnané sú 3 spôsoby riešenia.

Metriky Activity a Attendance si pri využití `InfluxDB` vyžadovali značnú zmenu implementácie, keďže nebolo možné využiť funkcie poskytované priamo `MySQL`. Ostala preto otázka, či takáto zmena implementácie neovplyvní v zlom smere vyhotovenie dát týchto metrík.

Z grafu 8.4 môžeme jasne pozorovať, že ani pri tomto type nedochádza k výnimke a výsledky sú opäť v prospech využitia nového riešenia.

Nové riešenie poskytuje rovnako ako pri predchádzajúcich metrikách výrazne rýchlejšie časové vyhotovenie v porovnaní s pôvodným. Rast krivky vyhotovenia je dokonca znova miernejší pri novom riešení, ako pri pôvodnom a keď pri intervale 1 hodiny je rozdiel vyhotovenia asi 1,5 minúty, pri intervale 1 týždňa je tento rozdiel už viac ako 2,7 minúty. Z čoho vyplýva, že s postupným zväčšením časového intervalu sa tieto rozdiely budú ešte zväčšovať. Vyžadovaná zmena implementácie nepriniesla horšie výsledky, možno práve naopak sú rozdiely ešte o čosi výraznejšie.

Pri pozorovaní optimalizovaného riešenia si môžeme všimnúť situáciu, ktorá nastala aj pri metrikách Heat a Spaghetti map v 8.5. Vyhodnotenie metriky síce je s malými časovými intervalmi výrazne rýchlejšie, no problém nastáva s rastúcim intervalom, kedy je vidieť, že pri intervale 1 týždňa už optimalizácia riešenia nehrá žiadnu rolu a rýchlosť vyhotovenia metriky je rovnaká. To by samozrejme platilo aj pre všetky väčšie intervaly, čo je viditeľné na grafe. Aj pri tejto metrike dosahuje nové riešenie najlepšie výsledky a udáva jasný dôvod pre náhradu pôvodného.



Obr. 8.5: Graf znázorňujúci dĺžku trvania vyhotovenia metriky Distance a Speed, pre rôzne časové intervaly dát, zobrazovaných metrikou od 1 hodiny až po 1 týždeň. Porovnané sú 3 spôsoby riešenia.

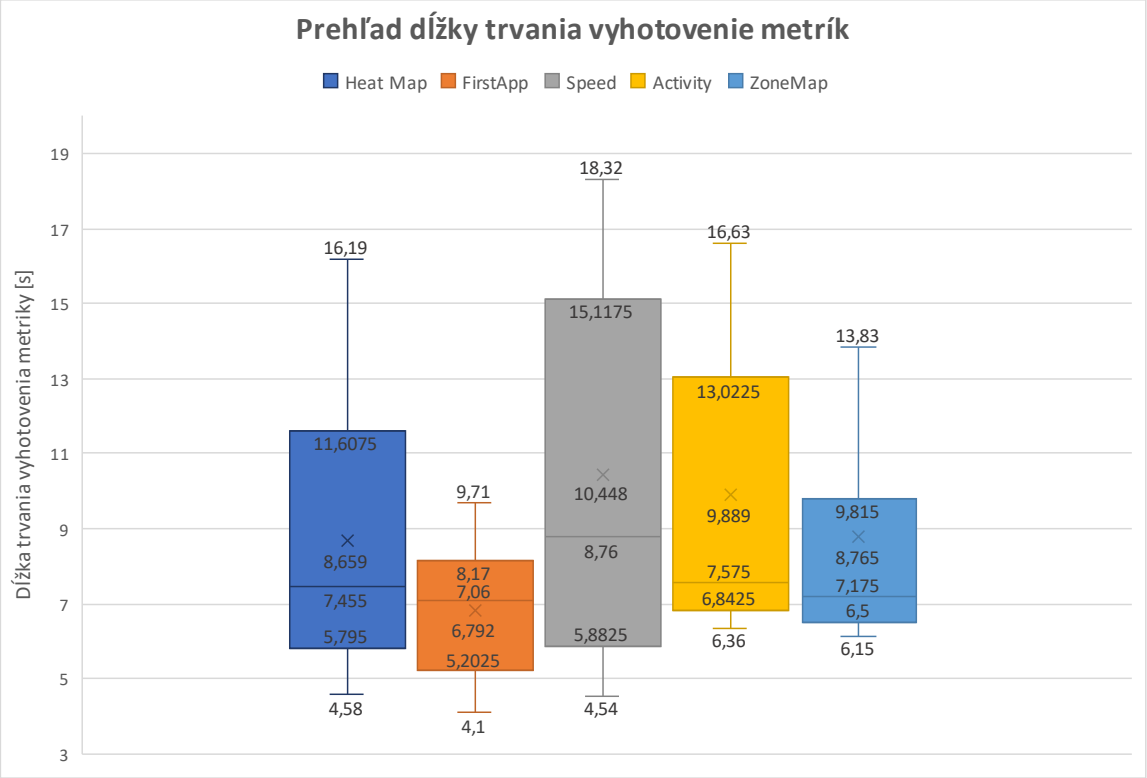
Vyhotovenie metrík Speed a Distance v prípade pôvodného riešenia trvá skutočne veľmi dlho, to aj pri malých časových intervaloch. Navrhnuté riešenia však aj v tomto prípade výrazne pomohli a priniesli žiadúce výsledky.

Optimalizované riešenie až prekvapivo markantne zvyšuje rýchlosť vyhodnotenia. Keď pri intervale 1 hodiny bol čas vyhotovenia okolo 4 minút, pri pôvodnom riešení optimalizované už klasicky prináša zrýchlenie do 1 sekundy. Pri tejto metrike je však možné pozorovať, že rovnako veľký rozdiel ostáva aj pri intervale 1 týždňa, kde rozdiel dosahuje takmer 4 minúty. Pri tejto skupine metrík by optimalizované riešenie predstavovalo naozaj vhodnú alternatívu.

Faktom však ostáva, že ani tu optimalizované riešenie neprekoná nové riešenie. Rovnako ako aj pri zvyšných metrikách je rozdiel zanedbateľný do 24 hodín časového intervalu. Postupným zväčšovaním sa však tento rozdiel prehlbuje a pri intervale 1 týždňa je nové riešenie o viac ako 80 sekúnd rýchlejšie.

Krivka vyhodnotenia rastie v tomto prípade pri pôvodnom riešení výrazne rýchlejšie ako pri už optimalizovanom alebo novom riešení. Dá sa skonštatovať, že rozdiely dĺžky vyhodnotenia by sa zväčšovaním intervalu len prehlbovali.

Aj pri tejto skupine metrík je tak jasným víťazom nové riešenie, ktoré poskytuje bezkonkurenčne rýchlejšie vyhodnocovanie.



Obr. 8.6: Krabicový graf zobrazujúci prehľad dĺžky trvania jednotlivých metrík. V každom grafe sú hodnoty (z hora) maximum, horný kvartil, priemer, medián, dolný kvartil, minimum. Pre každú z metrík som vykonal 10 meraní. Tieto merania boli vykonávané pre časový interval 24 hodín.

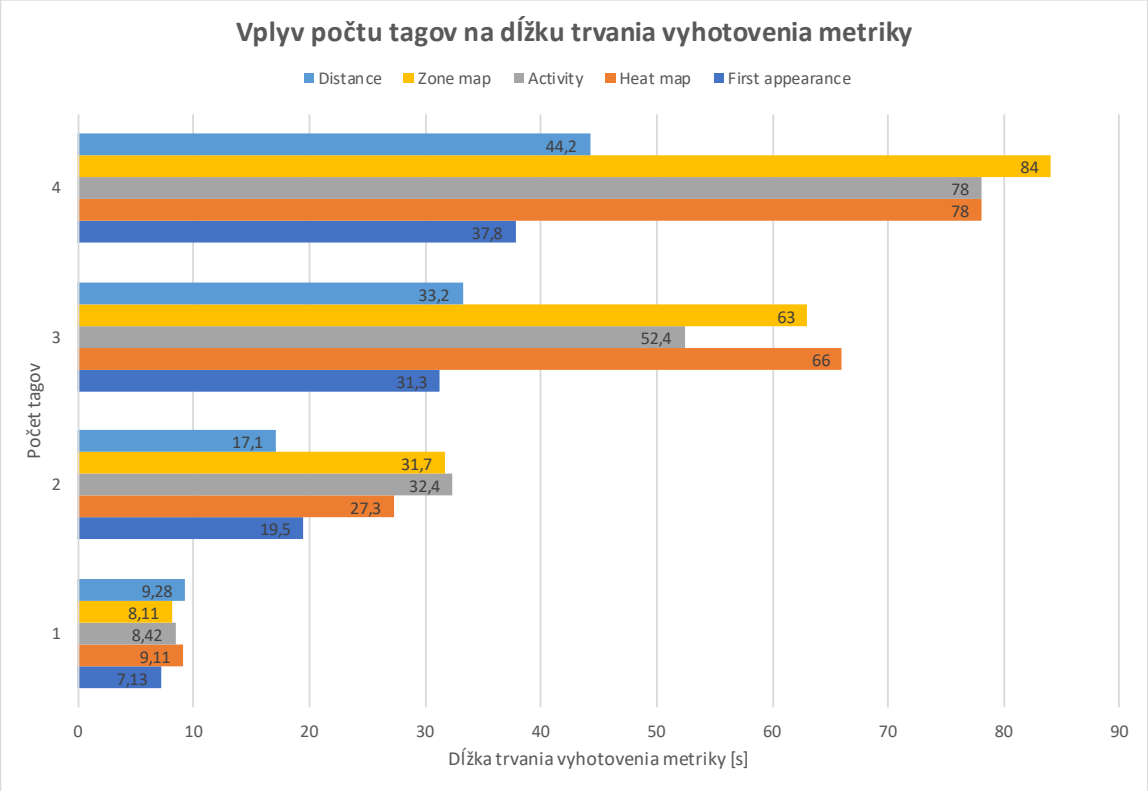
8.3 Experimenty

Táto sekcia sa zameriava na experimenty vykonávané na novom riešení navrhnutom v tejto diplomovej práci. Poukazuje na rôzne činitele, ktoré môžu ovplyvniť čas vyhotovenia metriky a v grafoch zobrazuje ako značne dokážu zvýšiť dĺžku trvania pri vyhotovení metriky. Zároveň tým poukazuje aj nato, kde a aký čas je pri vyhotovení potrebný pre každú z metrík. Všetky experimenty boli vykonávané pre časový interval 24 hodín. Hodnoty v grafoch sú výsledkom 3 meraní ak nie je uvedené inak.

8.3.1 Prehľad výsledkov pri viacerých meraniach

Krabicový graf z 8.6 poukazuje na reálny čas vyhotovenia metrík. Ako je možné vidieť, časové rozpätie sa od jednotlivých metrík líši. Najväčšie rozpätie dosahuje metrika speed (skupina Speed, Distance) a to aj viac ako 10 sekúnd. Najmenšie rozpatie je možné pozorovať pri metrike First appearance (Last appearance), kedy je rozpatie len niečo cez 5 sekúnd, avšak horný a dolný kvartil sa líšia len o niečo menej ako 3 sekundy. Dôležitým ukazovateľom je ale aj priemer. Ten nám dokáže dať skutočnú predstavu o tom, ako dlho by vyhotovenie mohlo trvať.[2]

[2]Ako je však už spomínané na začiatku tejto kapitoly, dĺžka trvania značne závisí aj od toho na akom stroji je vyhotovenie vykonávané, alebo ako je daný stroj vyťažený.



Obr. 8.7: Stĺpcový graf znázorňujúci vplyv počtu tagov na jednotlivé metriky.

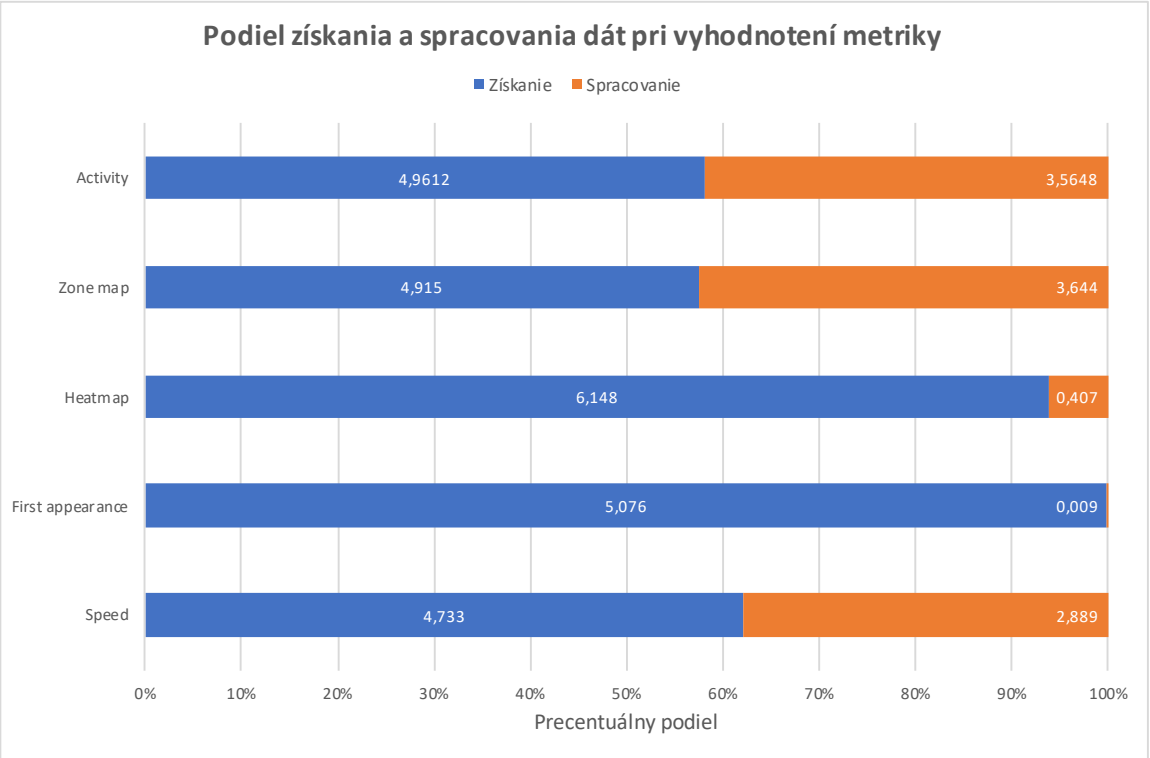
8.3.2 Vplyv počtu tagov na metriku

Stĺpcový graf na obrázku 8.7 ukazuje ako závisí čas potrebný na vyhotovenie metriky od toho, koľko tagov pre vizualizáciu si užívateľ vyberie. V prípade jedného tagu sa časy v tomto meraní pohybovali v priemere do 10 sekúnd. Pre všetky metriky bol čas veľmi podobný.

Pri vyhodnotení 2 tagov sú už medzi metrikami pozorovateľne značné rozdiely. Zatiaľ čo pri metrikách Distance (Speed) a First appearance (Last appearance) došlo k pochopiteľnému dvojnásobnému nárastu času, pri ostatných metrikách je vidieť miestami aj trojnásobný nárast. Takýto nárast je spôsobený rozdielnym získaním a spracovaním dát ako pri ostatných metrikách.

Rovnaký trend je potom viditeľný aj s postupným pridávaním tagov, teda konkrétne pre počet tagov 3 a 4 z obrázku 8.7. Opäť aj v týchto prípadoch sa metrikám Distance a First appearance zvýšil čas v porovnaní s ostatnými metrikami miernejšie. Ak sa pozrieme, tak pre 4 tagy to v priemere vychádza asi 10 sekúnd na tag, zatiaľ čo pri metrikách Zone map, Activity a Heatmap je to približne dvakrát toľko.

Tento experiment poukazuje nato, ako pridávanie grafov pre zhotovenie metriky rozdielne ovplyvňuje dané metriky. Zatiaľ čo pri metrikách Distance a First appearnce dĺžka vyhotovenia metriky rastie lineárne s počtom tagov, pri ostatných metrikách je tento rast o čosi výraznejší, a tak je jasné, že pri zobrazení väčšieho množstva tagov je nutné pri týchto metrikách počítať s dlhším čakaním na výsledok.



Obr. 8.8: Graf znázorňujúci pomer získania a spracovania dát pri zhotovovaní metrík.

8.3.3 Podiel spracovania a získania dát pri vyhotovení metriky

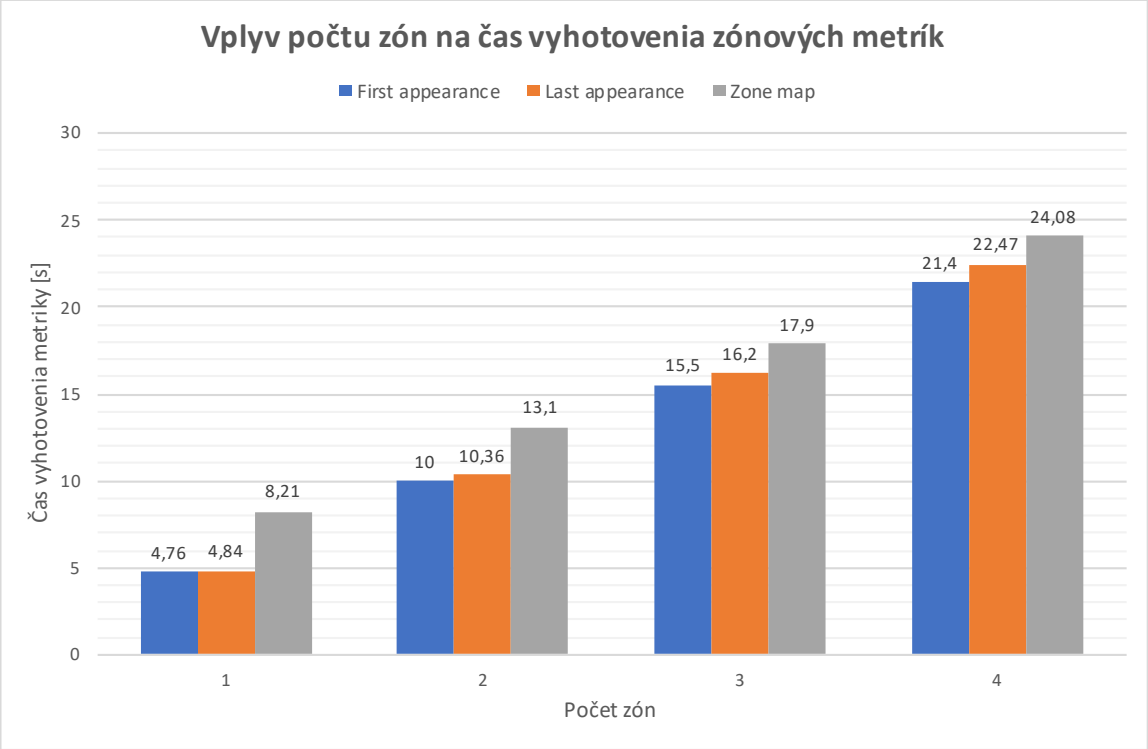
Ako bolo možné vidieť v predchádzajúcich experimentoch, celkový čas potrebný na zhotovenie metriky sa môže rôzne meniť vzhľadom na faktory ako časové rozpätie (pre ktoré sú dáta v metrike zobrazované) alebo množstvo tagov. Všetky tieto časy sú však zložené z 2 hlavných časti, a to z procesu získavania pozičných dát a z procesu ich spracovania.

V grafe na obrázku 8.8 je možné pozorovať podiel získania a spracovania v rámci vyhotovenia jednotlivých metrík. Ako je možné už na prvý pohľad vidieť, nie všetky metriky majú tento podiel rovnaký.

Výraznú rolu pri zhotovovaní hra spracovanie hlavne pri 3 skupinách metrík a to Speed (Distance), Zone map a Activity (Attendance). Práve pri týchto skupinách tvorí spracovanie výrazný podiel na celom vyhotovení metriky. Tento podiel predstavuje od 38 % do 43 %. Taký významný podiel bol samozrejme dosiahnutý po nasadení nového riešenia, a to z dôvodu výrazného zrýchlenia získavania dát. Otázkou tak ostáva, či práve pri týchto skupinách nie je možné aj spracovanie dát optimalizovať tak, aby táto optimalizácia priniesla znateľný posun pri skrátení času zhotovenia.

Výrazne menší podiel spracovania pri vyhotovení je možné vidieť pri 2 skupinách a to Heatmap (Spaghetti map) a First appearance (Last appearance). Z grafu je jasné, že takmer celý čas pri zhotovovaní týchto metrík tvorí práve získavanie pozičných dát.

Graf 8.8 poskytol jednoduchý prehľad o podiele spracovania a získania pozičných dát. Ukazuje možnosti, pri ktorých by malo zmysel sa dôkladnejšie pozrieť na spôsob spracovania dát a pokúsiť sa nájsť spôsob, ktorý by dokázal celý proces získavania pri týchto metrikách zrýchliť.



Obr. 8.9: Stĺpcový graf znázorňujúci vplyv počtu zón na čas vyhotovenia zónových metrík.

8.3.4 Vplyv počtu zón na zónovú metriku

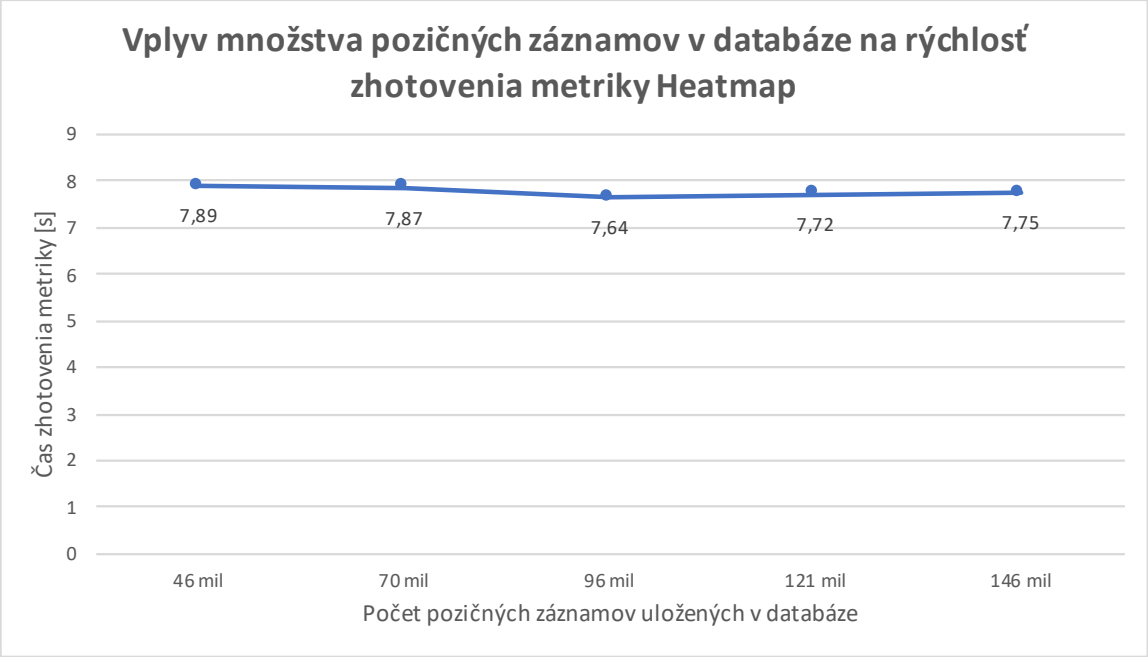
Vplyv počtu tagov v 8.3.2 poukázal aký výrazný vplyv ma pri zhotovení metrík. Pri zónových metrikách si však užívateľ nevyberá len to aké tagy chce pomocou nástroja Sage Analytics vizualizovať, ale aj to v akých zónach. Tento experiment poukazuje práve nato, aký veľký vplyv má počet zón, pre ktorý si užívateľ rozhodne dáta zobraziť.

Z grafu zobrazeného na obrázku 8.9 je možné pozorovať, že pre všetky 3 metriky má počet zvolených zón rovnaký vplyv. Dá sa skonštatovať, že čas, ktorý je potrebný pre vyhotovenie metriky rastie lineárne s počtom zón. Je tak možné očakávať, že pre veľký počet zvolených tagov a zón súčasne môže dochádzať k veľmi zdĺhavému načítaniu metriky, čo je však pochopiteľné.

8.3.5 Vplyv počtu pozičných záznamov v databáze

Pozičné záznamy objektov RTLS sa pravidelne ukladajú do databázy. Problémom však je, že počet týchto záznamov rastie skutočne rýchle (aj 8 záznamov za sekundu pre 1 objekt). S postupom času sa tak veľkosť databázy zväčšuje, a to práve väčším počtom pozičných záznamov. Otázkou tak ostáva, aký vplyv ma zväčšovanie databázy na rýchlosť vyhotovenia.

Z grafu na obrázku 8.10 môžeme vidieť vplyv množstva pozičných záznamov na zhotovenie metriky Heatmap. Už na prvý pohľad je jasne vidieť, že zvyšovanie počtu záznamov na rýchlosť zhotovenia nemá žiaden vplyv. To dokazuje takmer rovnaký čas pri zhotovení metriky pre 2 rôzne situácie, keď sa v databáze nachádza približne 46 miliónov záznamov a keď v nej je viac ako 146 miliónov záznamov.



Obr. 8.10: Čiarový graf znázorňujúci vplyv množstva pozičných záznamov na rýchlosť vyhotovenia metriky Heatmap pre časový interval 24 hodín.

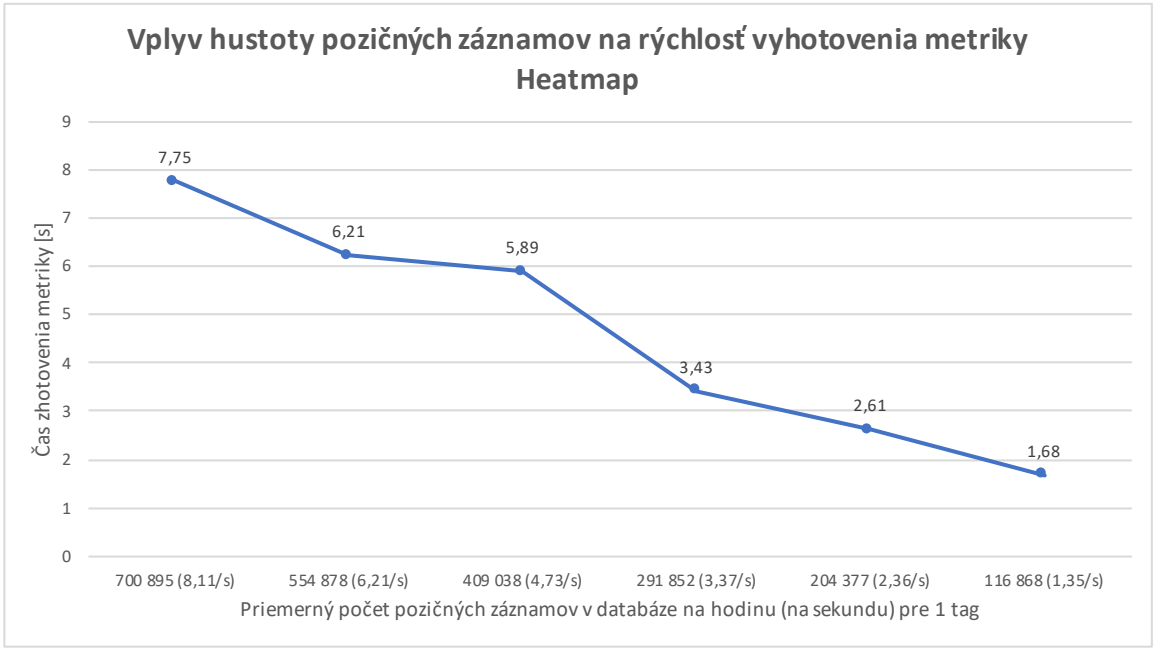
Toto zistenie tak ukazuje, ako dobre je optimalizovaná databáza použitá v novom riešení. Je možné prehlásiť, že s pribúdajúcim počtom pozičných záznamov sa celkový čas zhotovenia nebude zhoršovať.

8.3.6 Vplyv hustoty pozičných záznamov v databáze

Vieme povedať, že pribúdanie nových záznamov v databáze nemá vplyv na čas potrebný pre zhotovenie metriky. Záznamy však pribúdajú vo veľkom množstve, a to v priemere viac ako 8 pozičných záznamov za sekundu pre jeden tag. Otázkou tak je, či by nemalo zmysel ukladať menší počet záznamov za rovnaký čas a aký vplyv by to malo na rýchlosť zhotovenia.

Odpoveď je možné nájsť v čiarovom grafe na obrázku 8.10. Tento graf poukazuje na to, aký vplyv ma hustota pozičných záznamov uložených v databáze na rýchlosť zhotovenia metriky. Je možné ihneď vidieť, že s klesajúcou hustotou pozičných dát klesá aj čas vyhotovenia metriky. Pri počte viac ako 700 tisíc záznamov na deň, čo predstavuje v priemere niečo viac ako 8 záznamov za sekundu, zhotovenie metriky trvá približne 7,75 sekundy, zatiaľčo pri klesajúcom počte záznamov sa čas zmenšuje. Napríklad pri počte niečo viac ako 116 tisíc záznamov, čo predstavuje v priemere oko 1,35 záznamu na sekundu, sa tento čas zníži len na 1,68 sekundy.

Táto skutočnosť dáva najavo to, že zníženie frekvencie ukladania záznamov môže následne priniesť zrýchlenie nástroja Sage Analytics. Je potrebné si však uvedomiť, že v takom to prípade by ale došlo k poklesu presnosti vizualizovaných nástrojov. Z tohto dôvodu je potrebné rozhodnúť aká hustota záznamov je optimálna pre rýchlu ale relatívne presnú vizualizáciu pozičných dát.



Obr. 8.11: Čiarový graf znázorňujúci vplyv hustoty pozičných záznamov na rýchlosť vyhotovenia metriky Heatmap pre časový interval 24 hodín. Hustotou je chápaný počet záznamov na 1 hodinu (sekundu).

8.4 Nasledujúca práca

Využívanie nástroja Sage Analytics s riešením vytvoreným v tejto diplomovej práci dosahuje výrazne lepšie časy pri vyhotovení metrík, čím je tak možné konštatovať, že cieľ práce bol splnený a rýchlosť nástrojov Sage Analytics je akceptovateľná, a to aj pre časový interval niekoľkých dní a dokonca aj viac ako týždňa.

V nadväzujúcej práci by tak bolo možné pozrieť sa na niekoľko aspektov, ktoré by ešte mohli síce menším, ale určite nie zanedbateľným spôsobom zrýchliť celé zhotovovanie metrík. Medzi tieto aspekty by som zaradil: optimalizáciu vizualizácie, spracovania alebo samotný pohľad nato, či je súčasná frekvencia ukladania pozičných dát vhodná. Zmenšením frekvencie ukladania pozičných dát by malo za následok nie len zmenšenie priestoru na disku, ktoré tieto dáta zaberajú, ale ešte výraznejšie zrýchlenie získavania pozičných dat.

Kapitola 9

Záver

Základom tejto práce bolo naštudovanie si fungovania RTLS. V práci popisujem základy tohto systému a následne reálne využitie spoločnosťou Sewio. Bližšie poukazujem na aplikáciu RTLS Studia a hlavne jeho nástroja Sage Analytics, ktorý poskytuje nástroje pre analýzu pohybu objektov.

Cieľom tejto práce bolo pozrieť sa na existujúce riešenie, a vhodným spôsobom ho optimalizovať tak, aby získavanie a spracovanie dát prebiehalo v dostačujúcom čase, a to aj pre väčšie časové intervaly (niekoľko dní).

Ako prvé som sa zameral na existujúce riešenie, popísal jeho výhody, nevýhody a vykonal niekoľko testov pre zmeranie doby získavania dát z relačnej databázy MySQL. Následne som vykonal optimalizáciu existujúceho riešenia, ktorá priniesla veľmi dobre výsledky. Aj napriek veľmi dobrým výsledkom som sa pozrel aj na iné, alternatívne možnosti, ako by bolo možné pozičné dáta uložiť.

Z alternatívnych riešení sa ukázala ako najlepšia databáza InfluxDB. Využitím tejto databázy sa získavanie dát ešte výraznejšie zrýchlilo a zároveň sa jej vlastnosti a účel zhodoval s pozičnými dátami, s ktorými som pracoval. Po vykonaní testov sa ukázalo, že táto databáza dosahuje vynikajúce výsledky, kde sa zrýchlenie pri získavaní dát oproti existujúcemu riešeniu pohybovalo od 281 % až po 5154 %, pričom takmer všetky testy boli nad hranicou 1000 %. Z výsledkov bolo jasné, že práve InfluxDB je dobrým alternatívnym riešením pre existujúcu databázu.

Po nájdení vhodného riešenia na ukladanie dát som navrhol spôsob uloženia dát v tejto databáze a vytvoril skript pre migráciu dát.

Aby bolo možné využívať navrhnuté riešenie, bolo nutné implementovať zmeny, ktoré pristupujú vhodným spôsobom k tomuto riešeniu a upravujú priebeh získavania a spracovania dát tak, aby spolupracovali s navrhnutým riešením.

Navrhnuté a implementované zmeny priniesli očakávané výsledky. Z výsledkov vyplýva, že využitie nového, navrhnutého riešenia výrazne urýchľuje vyhotovenie metrík nástroja Sage Analytics. Merania dosahujú zrýchlenie od 725 % až po 2085 % oproti pôvodnému riešeniu. Všetky merania nového riešenia dosahujú výrazne rýchlejšie vyhotovenie metrík, pričom s rastúcim časovým intervalom, pre ktorý je metrika vyhotovená sa rozdiel v rýchlosti vyhotovenia medzi pôvodným a novým riešením ešte zväčšuje.

Súčasťou práce bolo aj vykonanie rôznych experimentov a poukázanie vplyvu rôznych činiteľov na čas zhotovenia metriky a spôsob, ako by mohla vyzerať nadväzujúca práca.

Literatúra

[1] *RTLS Technology Comparison: Indoor Tracking Technologies Comparison* [https://www.sewio.net/uwb-technology/rtls-technology-comparison]. Accessed: 14-12-2021.

[2] *Time Difference of Arrival* [https://www.sewio.net/uwb-technology/time-difference-of-arrival/]. Accessed: 15-12-2021.

[3] Berg, B., Longley, G. a Dunitz, J. Improving clinic operational efficiency and utilization with RTLS. *Journal of medical systems*. Springer. 2019, zv. 43, č. 3, s. 56.

[4] Boulos, M. N. K. a Berry, G. Real-time locating systems (RTLS) in healthcare: a condensed primer. *International journal of health geographics*. BioMed Central. 2012, zv. 11, č. 1, s. 1–8.

[5] Dempsey, C. *Types of GIS Data Explored: Vector and Raster* [article (english)]. Máj 2021. Dostupné z: https://betterprogramming.pub/8-techniques-to-speed-up-your-database-292754ff7739.

[6] Dix, P. Why Time Series Matters for Metrics, Real-Time Analytics and Sensor Data. *INFLUXDATA TECHNICAL PAPER* [article (english)]. 2021. Dostupné z: https://www.influxdata.com/resources/why-time-series-matters-for-metrics-real-time-and-sensor-data/.

[7] Education, I. C. *Relational Databases* [blogpost (english)]. August 2019. Dostupné z: https://www.ibm.com/cloud/learn/relational-databases.

[8] Encyclopedia Britannica, T. E. of. *Structured query language* [article (english)]. Jún 2021. Dostupné z: https://www.britannica.com/technology/SQL.

[9] Ferracuti, N., Norscini, C., Frontoni, E., Gabellini, P., Paolanti, M. et al. A business application of RTLS technology in Intelligent Retail Environment: Defining the shopper's preferred path and its segmentation. *Journal of Retailing and Consumer Services*. Elsevier. 2019, zv. 47, s. 184–194.

[10] Ho, H. J., Zhang, Z. X., Huang, Z., Aung, A. H., Lim, W.-Y. et al. Use of a real-time locating system for contact tracing of health care workers during the COVID-19 pandemic at an infectious disease center in Singapore: validation study. *Journal of medical Internet research*. JMIR Publications Inc., Toronto, Canada. 2020, zv. 22, č. 5, s. e19437.

[11] Lee, J.-S., Su, Y.-W. a Shen, C.-C. A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi. In: *IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society*. 2007, s. 46–51. DOI: 10.1109/IECON.2007.4460126.

[12] Liu, B., Wu, X., Reddy, V., Kang, D. M. a Liu, W. A RTLS/DR based localization system architecture for indoor mobile robots. In: *2010 IEEE International Symposium on Ä World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. 2010, s. 1–6. DOI: 10.1109/WOWMOM.2010.5534949.

[13] Löcklin, A., Ruppert, T., Jakab, L., Libert, R., Jazdi, N. et al. Trajectory Prediction of Humans in Factories and Warehouses with Real-Time Locating Systems. In: IEEE. *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2020, sv. 1, s. 1317–1320.

[14] Malik, A. *RTLS For Dummies*. For Dummies, 2009. ISBN 047039868X.

[15] Mariani, A. *Getting Back to Basics with Ultra-Wideband (UWB)*. Qorvo, apríl 2021.

[16] MEHEDINTU, A., Pîrvu, C. a Etegan, C. INDEXING STRATEGIES FOR OPTIMIZING QUERIES ON MYSQL. *Annals of the University of Petrosani Economics*. 2010, zv. 10, č. 4.

[17] Moniruzzaman, A. B. M. a Hossain, S. A. NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. *CoRR*. 2013, abs/1307.0191. Dostupné z: http://arxiv.org/abs/1307.0191.

[18] Oppermann, I., Hämäläinen, M. a Iinatti, J. *Introduction*. John Wiley & Sons, Ltd, 2004. 1-8 s. ISBN 9780470869192.

[19] Pancham, J., Millham, R. a Fong, S. J. Evaluation of Real Time Location System technologies in the health care sector. In: IEEE. *2017 17th International Conference on Computational Science and Its Applications (ICCSA)*. 2017, s. 1–7.

[20] Rizzi, A. a Romagnoli, G. Testing and deploying an RFID-based Real-Time Locating System at a fashion retailer: A case study. In: Springer. *Workshop on Business Models and ICT Technologies for the Fashion Supply Chain*. 2016, s. 201–214.

[21] Rychlý, M. *NoSQL databáze* [University Lecture]. Vysoké učení technické v Brně, Fakulta informačních technologií, Október 2020.

[22] Schaedel, M. *Simplifying InfluxDB: Shards and Retention Policies* [article (english)]. Dostupné z: https://www.influxdata.com/blog/influxdb-shards-retention-policies/.

[23] Schaefer, L. *What is NoSQL?* [article (english)]. Dostupné z: https://www.mongodb.com/nosql-explained.

[24] Sewio. *RTLS Studio* [article (english)]. Dostupné z: https://docs.sewio.net/docs/rtls-studio-1017834.html.

[25] Shi, G. a Ming, Y. Survey of indoor positioning systems based on ultra-wideband (UWB) technology. In: *Wireless communications, networking and applications*. Springer, 2016, s. 1269–1278.

[26] team, I. *InfluxDB shards and shard groups* [article (english)]. Dostupné z: https://docs.influxdata.com/influxdb/v2.0/reference/internals/shards/.

[27] team, I. *Time series database (TSDB) explained* [article (english)]. Dostupné z: https://www.influxdata.com/time-series-database/.

[28] Techopedia. *Spatial Database* [article (english)]. Dostupné z: https://www.techopedia.com/definition/17287/spatial-database.

[29] Thiede, S., Sullivan, B., Damgrave, R. a Lutters, E. Real-time locating systems (RTLS) in future factories: technology review, morphology and application potentials. *Procedia CIRP*. Elsevier. 2021, zv. 104, s. 671–676.

[30] Thoma, M. *8 Techniques To Speed up Your Database* [article (english)]. Apríl 2021. Dostupné z: https://betterprogramming.pub/8-techniques-to-speed-up-your-database-292754ff7739.

Príloha A

Obsah priloženého pamäťového média

Priložené pamäťové médium obsahuje:

- performance_scripts/ – priečinok obsahujúci skripty na testovanie výkonnosti DB riešení;
- migration_scripts/ – priečinok obsahujúci skripty pre migráciu dát medzi databázami;
- measurements/ – priečinok s meraniami;
- sage_analytics/ – priečinok so zdrojovými kódmi;
- videa – priečinok s videami demonštrujúcimi riešenie tejto práce;
- README.md – súbor s popisom odovzdaných súborov;
- tex/ – priečinok so zdrojovými súbormi textovej časti práce;
- dp.pdf – textová časť práce;
- hodnotenie.pdf – hodnotenie od spoločnosti Sewio.