# Datamining pro IoT integrační platformu

Bc. David Bulawa

Diplomová práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **David Bulawa** |
| Osobní číslo: | **A20625** |
| Studijní program: | **N0613A140022 Informační technologie** |
| Specializace: | **Softwarové inženýrství** |
| Forma studia: | **Kombinovaná** |
| Téma práce: | **Datamining pro IoT integrační platformu** |
| Téma práce anglicky: | **Datamining for IoT integration Platform** |

## Zásady pro vypracování

1. Vypracujte literární rešerši na dané téma.
2. Analyzujte aktuální strukturu dat v IoT platformě.
3. Navrhněte systém vhodný pro perzistenci velkých dat z IoT prvků.
4. Prozkoumejte možnosti implementace algoritmu pro předzpracování dat.
5. Implementujte vybraný algoritmus pro detekci vzorů a anomálií v datech. Alternativně prověřte další možné algoritmy, které mohou být vzhledem ke struktuře dat přínosné.
6. Proveďte testování, vhodnou interpretaci výsledků a závěr.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:


1. *Data science & big data analytics: discovering, analyzing, visualizing and presenting data*. Indianapolis: Wiley, [2015], xviii, 410 s. ISBN 9781118876138.
2. GRUS, Joel. *Data science from scratch*. Sebastopol: O'Reilly, 2015, xvi, 311 s. ISBN 9781491901427.
3. OJEDA, Tony, Sean Patrick MURPHY, Benjamin BENGFORT a Abhijit DASGUPTA. *Practical data science cookbook: 89 hands-on recipes to help you complete real-world data science projects in R and Python*. Birmingham: Packt Publishing, 2014, 380 s. ISBN 9781783980246.
4. MILES, Matthew B., A. M. HUBERMAN a Johnny SALDAÑA. *Qualitative data analysis: a methods sourcebook*. Fourth edition. Los Angeles: SAGE, [2020], xxi, 380 s. ISBN 9781544371856.
5. DORSEY, Richard. *Data analytics*. [CreateSpace Independent Publishing Platform], [2017], 67 s. ISBN 9781547089291.
6. ANKAM, Venkat. *Big data analytics: a handy reference guide for data analysts and data scientists to help to obtain value from big data analytics using Spark on Hadoop clusters*. Birmingham: Packt, 2016, xv, 300 s. ISBN 9781785884696.
7. KERAMIDAS, Georgios, Nikolaos S. VOROS a Michael HÜBNER. *Components and services for IoT platforms: paving the way for IoT standards*. Switzerland: Springer, [2016], ©2017., 1 online resource (382 pages). ISBN 9783319423043. Dostupné také z: https://proxy.k.utb.cz/login?url=https://link.springer.com/10.1007/978-3-319-42304-3.
8. KAUR, Gurjit; TOMAR, Pradeep (ed.). *Handbook of Research on Big Data and the IoT*. IGI Global, 2019.


Vedoucí diplomové práce: **doc. Ing. Roman Šenkeřík, Ph.D.**
Ústav informatiky a umělé inteligence

Konzultant diplomové práce: **Ing. Adam Viktorin, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **3. prosince 2021**
Termín odevzdání diplomové práce: **23. května 2022**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

**doc. Mgr. Milan Adámek, Ph.D.** v.r.
děkan

**prof. Mgr. Roman Jašek, Ph.D., DBA** v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 18.5.2022 David Bulawa, v.r.

## ABSTRAKT

Tato práce se věnuje aplikaci metod z oblasti data miningu na data pocházející z integrační platformy pro internet věcí. V teoretické části je nastíněna problematika big dat a získávání znalostí z nich. Cílem praktické části je navrhnout vhodné prostředí pro uchování hodnot ze senzorů a akčních členů tak, aby bylo možné k nim pohodlně a rychle přistupovat. Poté je provedena aplikace data miningu na uchovávaná data s cílem identifikovat anomálie a opakující se vzory.



Klíčová slova:

data mining, internet věcí, IoT, big data, zpracování dat, analýza dat, datová věda



## ABSTRACT

This thesis focuses on the application of data mining methods to data stored in the Internet of Things (IoT) integration platform. The theoretical part outlines the issues of big data and knowledge extraction from it. The practical part aims to design a suitable environment for storing values from sensors and actors so that they can be accessed quickly and conveniently. Data mining is then applied to the stored data to identify anomalies and repeating patterns.



Keywords:

data mining, internet of things, IoT, big data, data processing, data analysis, data science

Chtěl bych poděkovat především vedoucímu této práce, kterým byl pan doc. Ing. Roman Šenkeřík, Ph.D. Děkuji mu za zodpovědné vedení práce a věcné připomínky. Dále bych chtěl poděkovat panu Ing. Adamu Ulrichovi a Ing. Adamu Viktorinovi, Ph.D. za technické konzultace a praktické rady.

V neposlední řadě bych chtěl také poděkovat své rodině a přítelkyni za jejich podporu při psaní této práce.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

# OBSAH


**ÚVOD** ..... **10**

**I TEORETICKÁ ČÁST** ..... **12**

**1 BIG DATA** ..... **13**

1.1 DEFINICE ..... 13

1.2 OBECNÉ PRINCIPY ..... 15

1.3 ŠKÁLOVATELNOST ..... 15

1.3.1 Vertikální škálování ..... 15

1.3.2 Horizontální škálování ..... 16

1.4 KONZISTENCE ..... 16

1.5 DISTRIBUCE ..... 16

1.5.1 Rozdělení dat ..... 17

1.5.2 Replikace ..... 18

**2 UKLÁDÁNÍ BIG DAT** ..... **20**

2.1 DATOVÉ SKLADY ..... 20

2.1.1 Definice ..... 20

2.1.2 Architektura ..... 21

2.1.3 Struktura ..... 22

2.2 ÚVOD DO NOSQL DATABÁZÍ ..... 23

2.2.1 Princip ..... 23

2.2.2 CAP teorém ..... 24

2.3 DATABÁZE TYPU KLÍČ-HODNOTA ..... 25

2.3.1 Princip indexace ..... 25

2.3.2 Rozhraní databáze ..... 27

2.3.3 Jmenné prostory klíčů ..... 27

2.3.4 Distribuce dat ..... 28

2.3.5 Použití ..... 29

2.4 SLOUPCOVÉ DATABÁZE ..... 29

2.4.1 Struktura datového modelu ..... 29

2.4.2 Rozhraní databáze ..... 30

2.4.3 Distribuce dat ..... 31

2.5 DOKUMENTOVÉ DATABÁZE ..... 32

2.5.1 Struktura datového modelu ..... 32

2.5.2 Rozhraní databáze ..... 34

2.5.3 Distribuce dat ..... 35

2.6 GRAFOVÉ DATABÁZE ..... 35

2.6.1 Struktura dat ..... 36

**3 DATA MINING** ..... **37**

3.1 DEFINICE POJMU ..... 37

3.2 DATA MINING JAKO PROCES (CRISP-DM) ..... 38

3.2.1 Fáze pochopení obchodu ..... 39

3.2.2 Fáze porozumění datům ..... 39

3.2.3 Fáze přípravy dat ..... 40

3.2.4 Fáze modelování ..... 40
3.2.5 Fáze vyhodnocení ..... 40
3.2.6 Fáze nasazení ..... 40
3.3 Úlohy data miningu ..... 41
3.3.1 Shluková analýza ..... 42
3.3.2 Sumarizace ..... 43
3.3.3 Asociační pravidla ..... 43
3.3.4 Objevování sekvencí ..... 43
3.3.5 Klasifikace ..... 44
3.3.6 Regrese ..... 44
3.3.7 Predikce ..... 45
3.3.8 Analýza časových řad ..... 45
3.4 Architektura v softwarových systémech ..... 46
3.4.1 Rozložení vrstev data mining nástrojů ..... 46
3.4.2 Umístění jádra data miningu ..... 47
3.4.3 Typy data mining architektury ..... 47
3.5 Data mining a strojové učení ..... 48
3.5.1 Společné rysy ..... 48
3.5.2 Rozdíly ..... 49
**4 PREPROCESSING DAT ..... 50**
4.1 Čištění dat ..... 51
4.1.1 Odstranění šumu ..... 52
4.1.2 Opravení chybějících hodnot ..... 52
4.2 Redukce dat ..... 53
4.2.1 Redukce dimenze ..... 53
4.2.2 Převod na alternativní reprezentaci ..... 54
4.3 Transformace dat ..... 54
4.3.1 Konstrukce atributů ..... 54
4.3.2 Normalizace ..... 55
4.3.3 Diskretizace ..... 56
**II PRAKTICKÁ ČÁST ..... 57**
**5 AKTUÁLNÍ STRUKTURA DAT V IOT PLATFORMĚ ..... 58**
5.1 IoT ..... 58
5.1.1 Definice ..... 58
5.1.2 IoT a big data ..... 59
5.2 Popis platformy MyMight ..... 59
5.2.1 Možnosti platformy ..... 60
5.2.2 Integrované technologie ..... 61
5.2.3 Technická specifikace platformy ..... 61
5.3 Struktura ukládaných hodnot ..... 62
5.3.1 Původ dat ..... 63
5.3.2 Komunikace v IoT technologiích ..... 63
5.3.3 Typy hodnot v IoT platformě ..... 64
5.4 Perzistence dat z IoT prvků ..... 65
5.4.1 Problém odezvy čtení a zápisu ..... 66

5.4.2 Problém nárůstu velikosti dat ..... 67
5.4.3 Problém rozmanitosti datových typů ..... 68
5.4.4 Návrh řešení ..... 68
5.5 TESTOVACÍ DATASET ..... 69
5.5.1 Formát souboru dat ..... 69
5.5.2 Charakteristika dat ..... 69
**6 NÁVRH SYSTÉMU PRO PERZISTENCI HISTORICKÝCH HODNOT ..... 71**
6.1 POŽADAVKY NA SYSTÉM ..... 71
6.1.1 Bezpečnost ..... 71
6.1.2 Integrace ..... 71
6.1.3 Kvalita ..... 71
6.1.4 Jednoduchost ..... 72
6.1.5 Rychlá odezva ..... 72
6.2 NÁVRH SYSTÉMU ..... 72
6.2.1 Systém jako webová služba ..... 73
6.2.2 Výběr programovacího jazyka ..... 73
6.2.3 Spring ..... 74
6.2.4 Spring boot ..... 76
6.2.5 Výběr frameworků ..... 76
6.2.6 Architektura ..... 76
6.2.7 Definice REST rozhraní ..... 79
6.2.8 Zabezpečení systému ..... 81
6.3 UCHOVÁNÍ DAT ..... 82
6.3.1 Možnosti systémů pro uchování dat ..... 82
6.3.2 Požadavky na NoSQL databázi ..... 83
6.3.3 Porovnání možných NoSQL databází ..... 84
6.3.4 Výběr databáze ..... 85
6.3.5 Konfigurace a struktura databáze ..... 86
6.3.6 Nastavení indexů ..... 87
6.3.7 Optimalizace velikosti dokumentu ..... 89
6.4 ZHODNOCENÍ ..... 89
**7 DATA MINING IOT DAT ..... 91**
7.1 MOŽNÉ ÚLOHY DATA MININGU ..... 91
7.1.1 Detekce anomálií ..... 91
7.1.2 Rozpoznání častých vzorů a kombinací ..... 92
7.1.3 Predikce ..... 92
7.1.4 Klasifikace ..... 92
7.2 DBSCAN ..... 93
7.2.1 Algoritmus ..... 93
7.2.2 Využití pro detekci anomálií ..... 94
7.2.3 Preprocessing ..... 94
7.2.4 Nastavení počátečních parametrů ..... 96
7.2.5 Možnosti použití algoritmu ..... 97
7.2.6 Použití pouze pro dimenzi hodnot ..... 97
7.2.7 Použití algoritmu pro 2D data ..... 98
7.3 ARIMA ..... 99
7.3.1 Preprocessing ..... 101

7.3.2 Nastavení počátečních parametrů ..... 101
7.3.3 Možnosti použití modelu ..... 101
7.3.4 Detekce anomálií v reálném čase ..... 102
7.3.5 Predikce budoucích hodnot ..... 103
7.4 APRIORI ..... 104
7.4.1 Algoritmus ..... 104
7.4.2 Způsob rozpoznání ..... 105
7.4.3 Preprocessing ..... 105
7.4.4 Nastavení počátečních parametrů ..... 107
7.4.5 Rozpoznání častých kombinací událostí ..... 107
7.4.6 Možné budoucí vylepšení ..... 107
**8 INTERPRETACE VÝSLEDKŮ ..... 109**
8.1 DETEKCE ANOMÁLIÍ ..... 109
8.1.1 Zpětná detekce algoritmem DBSCAN ..... 109
8.1.2 DBSCAN bez časové dimenze ..... 110
8.1.3 DBSCAN s časovou dimenzí ..... 111
8.1.4 Detekce v reálném čase modelem ARIMA ..... 113
8.2 ROZPOZNÁNÍ ČASTÝCH KOMBINACÍ UDÁLOSTÍ ..... 115
8.2.1 Časté kombinace událostí za duben 2021 ..... 116
8.2.2 Časté kombinace událostí za říjen 2021 ..... 117
8.3 PREDIKCE ..... 118
**ZÁVĚR ..... 121**
**SEZNAM POUŽITÉ LITERATURY ..... 122**
**SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK ..... 127**
**SEZNAM OBRÁZKŮ ..... 129**
**SEZNAM TABULEK ..... 131**
**SEZNAM PŘÍLOH ..... 132**

# ÚVOD

Data a informace jsou všude kolem nás. Jejich množství exponenciálně narůstá a s rozšířením internetu se růst ještě více zrychlil.

Problémem se stává jejich uchovávání. A to především tak, aby k datům byl zajištěn konzistentní a rychlý přístup, a to v nejlepším případě vždy, když je to potřeba. Technologie uchovávání dat proto za poslední dobu prochází velkou inovací. Systémy, které byly ještě před 30 lety naprosto dostačující, dnes v některých případech narážejí na problémy škálovatelnosti. Není možné do nekonečna zvyšovat kapacitu databází a přitom doufat, že rychlost přístupu k datům zůstane stejná. Klasické relační databáze proto v dnešní době velkých dat nemusí vždy dostačovat. Je tedy potřeba se zaměřit na systémy umožňující pracovat s velkými objemy dat s co nejmenší ztrátou rychlosti a konzistence.

Vzhledem k nepřebernému množství dat, které servery v nynější době zpracovávají a uchovávají se objevila myšlenka, proč nevyužít potenciál dat a získat z nich přínosné informace. Otevírají se tedy nové možnosti práce s daty. Objevují se nástroje, které s využitím statistiky pomáhají data zpracovávat strojově a získávat z nich právě ony přínosné informace. Algoritmy z oblasti softcomputingu jsou schopné aproximovat řešení na základě předchozích výsledků, aniž by byly na daný problém konkrétně naprogramovány. I na trhu práce se vytvořil prostor pro oblasti jako je datová analýza nebo datová věda. Data a informace zkrátka ovlivňují náš život ať už chceme nebo ne.

Někdy však není jednoduché informace v datech najít. Data mining je název pro vědeckou disciplínu, která se věnuje vytěžování dat z databází a datových skladů. Využívá znalosti z vědních oborů jako je Matematika, Statistika či Fuzzy logika. Pojem data mining je často v odborných publikacích považován za nepřesný a nahrazuje se pojmem KDD (Knowledge Discovery in Data), který přesněji vystihuje proces získávání informací. Nicméně v této práci je možné se setkat s oba termíny a lze je považovat za synonyma.

První část této práce, označována též jako teoretická, se v počátku věnuje problematice uchovávání velkých dat. Obsahuje definici pojmu Big data a dalších okolností, které s pojmem souvisí. Dojde k porovnání několika typů vhodných datových úložišť. Poté se práce přesune od samotných dat k jejich zpracování. V obecné rovině je zde přiblížena oblast data miningu. Popis data miningu jako procesu (CRISP-DM), ale také pojmenování některých jeho typických úloh. Detaily konkrétních použitých algoritmů se nachází v praktické části. Teoretická

část je zakončena oblastí preprocessingu dat, kde je uveden popis čištění, redukce a transformace dat.

Druhá část reflektuje poznatky z teoretické části, snaží se je převést do praxe a ukázat na konkrétním příkladu. Tímto příkladem bude, jak již samotný název práce napovídá, data mining v integrační platformě pro IoT (Internet of Things – internet věcí). Zpracování dat bude předcházet popis aktuální struktury uchovávaných historických hodnot z IoT zařízení v platformě od společnosti MyMight a návrh nového vhodného systému, který bude umožňovat rychlý přístup k datům kdykoli bude potřeba. Následovat bude aplikace data mining algoritmů na zmíněné historické hodnoty. Pro účely testování poslouží dataset (sada dat) historických hodnot ze vzorového domu, který obsahuje data za několik měsíců reálného provozu několika IoT technologií v různých případech užití. Závěrem této práce je vhodná interpretace výsledků dosažených vybranými algoritmy a zhodnocení využitelnosti získaných informací.

# I. TEORETICKÁ ČÁST

# 1 BIG DATA

Pojem **data** je velmi složité definovat. Obecně o nich je možné hovořit jako o zaznamenaných údajích, které mají popisný charakter a lze je relativně snadno uchovávat. Je možné je získat různými způsoby jako je např. měření, pozorování, odvození a další. Lze je dělit podle oborů, ve kterých byla získána.

Mezi definicemi pojmu data, se často objevuje i mylná definice, která hovoří o datech jako o informacích. To je však chybné tvrzení, jelikož tyto pojmy spolu sice souvisí, ale rozhodně nejsou ekvivalentní.

Mezi daty a informacemi je nepatrný rozdíl. Data jsou fakta nebo podrobnosti, ze kterých se odvozují informace. Jednotlivé kusy dat jsou zřídka užitečné samostatně. Aby se z dat staly informace, je třeba je dát do kontextu. [1]

Tato práce se bude věnovat datům uchovaným v digitální podobě. Momentálně se jedná o bezesporu o nejrychleji rostoucí oblast uchovávání dat. Digitalizace dat je trend, ze kterého se nedá uhnout. Veškeré větší úřady, instituce, firmy a společnosti data v digitální podobě uchovávají nebo k tomu alespoň směřují. S rozvojem technologií se tento způsob uchování stal rychlým, levným i bezpečným. Existují i typy dat, které do digitální podoby převést nelze. Ty tvoří, ale pouze minoritní zastoupení. Většinou je možné vytvořit jejich digitální aproximaci.

V roce 2005 Roger Mougalas z nakladatelství O'Reilly, které je známé díky jejich technickým publikacím v oblasti informatiky, poprvé použil termín **Big data** [2]. Odkazoval na velký počet dat, který je téměř nemožné spravovat a zpracovávat pomocí tradičních business intelligence nástrojů. Od té doby uplynulo již několik let a digitální data se stále těší exponenciálnímu růstu. Položil tak základy pojmu, který by se dnes dal považovat za buzzword[1].

## 1.1 Definice

Je tedy jasné, že pojem Big data označuje zkrátka velké množství dat. To je, ale velmi abstraktní pojem. Množství, jež je označováno za velké se může lišit v závislosti na aktuálním

[1] Buzzword – Populární, někdy až módní, slovo.

kontextu. V dnešní době není nic neobvyklého, že osobní počítač má k dispozici úložiště s kapacitou v jednotkách terabajtů (TB), tj. $10^{12}$ bajtů (B).

V roce 2020 se celkové množství dat vytvořených, zachycených, zkopírovaných a spotřebovaných na světě odhadovalo na 60 zettabajtů (ZB) tj. 33 x $10^{21}$ bajtů. Předpokládá se, že do roku 2025 objem dosáhne ohromujících 180 zettabajtů. Za tímto velkým nárůstem stojí především popularizace sociálních sítí, komunikačních aplikací, video streaming platforem, ale třeba i IoT. WhatsApp, známá aplikace pro rychlé posílání zpráv, zaznamenává 65 miliard odeslaných zpráv každý den. Na světě nejrozšířenější vyhledávač Google provede 40 tisíc hledání za 1 vteřinu. Všichni uživatelé internetu tak celkem v roce 2020 každým dnem vygenerovali 2,5 kvintilionů bajtů, tj. 2,5 x $10^{18}$ bajtů. Konkrétní hodnoty jsou převzaty ze článku publikovaném na serveru Techjury [3], který agreguje statistiky o datech z několika důvěryhodných zdrojů. Nicméně se údaje mohou lehce lišit v závislosti na konkrétním zdroji. Odlišnost je však v řádech jednotek procent. Tyto informace slouží primárně pro představu aktuální situace v oblasti digitálních dat.

V předchozím odstavci jsou zmíněny příklady firem, které jsou známé po celém světě a mají miliardy uživatelů. Leckdo by si tak mohl říct, že se big data týkají pouze velkých korporátních společností a menší firmy tak otázku zpracování big dat nemusí vůbec řešit. To však není pravda. Jak už bylo zmíněno velikost je v definici pojmu relativní a vždy záleží především na kontextu. Nebude mít smysl řešit zpracování big dat např. ve stavební firmě o velikosti 10 zaměstnanců, která má v digitální podobě pouze účetnictví. Ale např. i malá společnost věnující se sběru hodnot ze senzorů v agroprůmyslu už může čelit zpracování datového objemu, který by se dal kvantifikovat jako dostatečně velký pro big data.

Je možné pojem Big data definovat pomocí třech vlastností, které by měla data nabývat, pokud jsou takto označována. Definice těchto vlastností je známá jako **3 V**.

- **Volume** – „Velký“ objem datových sad.
    - Typicky v řádech terabajtů či větších.
- **Velocity –** Tempo, jakým objem dat narůstá.
    - Objem by měl mít exponenciální nebo alespoň lineární nárůst.
- **Variety –** Rozmanitost, která lze v datech najít.
    - Data mohou obsahovat strukturované i nestrukturované části.
    - Rozmanitost spočívá ve schopnosti třídit příchozí data do různých kategorií.

K těmto třem základním vlastnostem se postupem času začali přidávat další. Nicméně definice známá jako 3 V se zachovala a obsahuje pouze zmíněné tři základní. Některé další vlastnosti související s big daty jsou:

- **Veracity –** Nejistá věrohodnost dat.
- **Value –** Data mají vysokou hodnotu pro jejich vlastníka.
- **Validity –** Doba platnosti dat je limitovaná.
- **Volatility –** Doba nutného ukládání dat je přechodná.

## 1.2 Obecné principy

Běžné způsoby ukládání dat do souborů, klasických relačních databází atd. pro big data nemusí dostačovat. V takovém případě je třeba se obrátit na nástroje, které jsou pro práci s velkým objemem dat připravené. Připravenost v tomto podání bude znamenat především podporu distribuovaného ukládání, streamingu dat nebo například replikace. Samozřejmostí je, aby k datům byl zachován rychlý přístup, dodržena struktura jejich uložení a integrita. Těmito nástroji mohou být NoSQL databáze nebo vybrané datové sklady.

## 1.3 Škálovatelnost

Velmi důležitá vlastnost v oblasti big dat. V kontextu zpracování a uchovávání dat se jedná o schopnost flexibilně reagovat na měnící se požadavky [4]. V tomto případě především zvyšující se objem dat, zvyšující se počet operací nad daty a také zvyšující se objem dat zpracovávaný v těchto operacích.

Škálování se dělí rozdělit na vertikální a horizontální.

### 1.3.1 Vertikální škálování

Někdy označované anglickým výrazem **scaling up**. Znamená škálování zvyšováním výkonu konkrétního prvku, v tomto případě většinou serveru. Principiálně se jedná o poměrně jednoduchý proces, kdy se zkrátka dokupuje výkonnější hardware. Z jednoduchého postupu plyne však několik nevýhod.

1. Cena HW v závislosti na výkonu roste exponenciálně.
2. Existuje určitý limit, přes který se nedá dostat ani s neomezeným rozpočtem.

### 1.3.2 Horizontální škálování

Občas označované anglickým výrazem **scaling out**. Jedná se o princip tzv. distribuovaného škálování, které umožňuje distribuovat data přes více prvků (serverů). Ty se pak v kontextu distribuované sítě nazývají uzly. Jsou sdružovány do tzv. clusterů. Díky tomu mohou být operace s daty vykonávány nad clusterem a uživatel by v ideálním případě ani neměl poznal, že pracuje v distribuované síti. Nicméně i tento způsob škálování naráží na problémy.

1. Podpora pouze u některých vybraných systémů.
2. Problémy s transakčním zpracováním.
3. Problémy se zajištěním konzistence dat.

Přes tyto neduhy je však horizontální škálování v oblasti big dat hojně využíváno. To především kvůli výhodnému poměru cena/výkon.

## 1.4 Konzistence

Splnění této vlastnosti znamená, že uložená data jsou korektní a odpovídají realitě [4]. V běžných relačních databázích nám na splnění konzistence dohlíží integritní omezení. Konzistence je také jednou ze čtyř vlastností databázových transakcí v relačních databázích známých pod zkratkou **ACID**.

- **Atomicity** – Nedělitelnost databázové transakce. Ta musí proběhnout buď celá nebo vůbec.
- **Consistency** – Transakce zajišťuje přechod z konzistentního do konzistentního stavu.
- **Isolation** – Více najednou probíhajících transakcí se nesmí vzájemně ovlivňovat.
- **Durability** – Jakmile je transakce potvrzena, její změny mají trvalý charakter.

V NoSQL databázích bývá dodržení těchto vlastností obtížnější. To je podrobně rozebráno v kapitole věnované konkrétním databázím.

## 1.5 Distribuce

V případě, že se náš systém skládá z vícero serverů. Což je při zpracovávání velkých objemů dat běžná praxe, jelikož je často využíváno horizontálního škálování. Je nutné, aby se data mezi jednotlivými servery správně distribuovaly. Rozlišují se 2 **typy distribuce dat** mezi uzly.

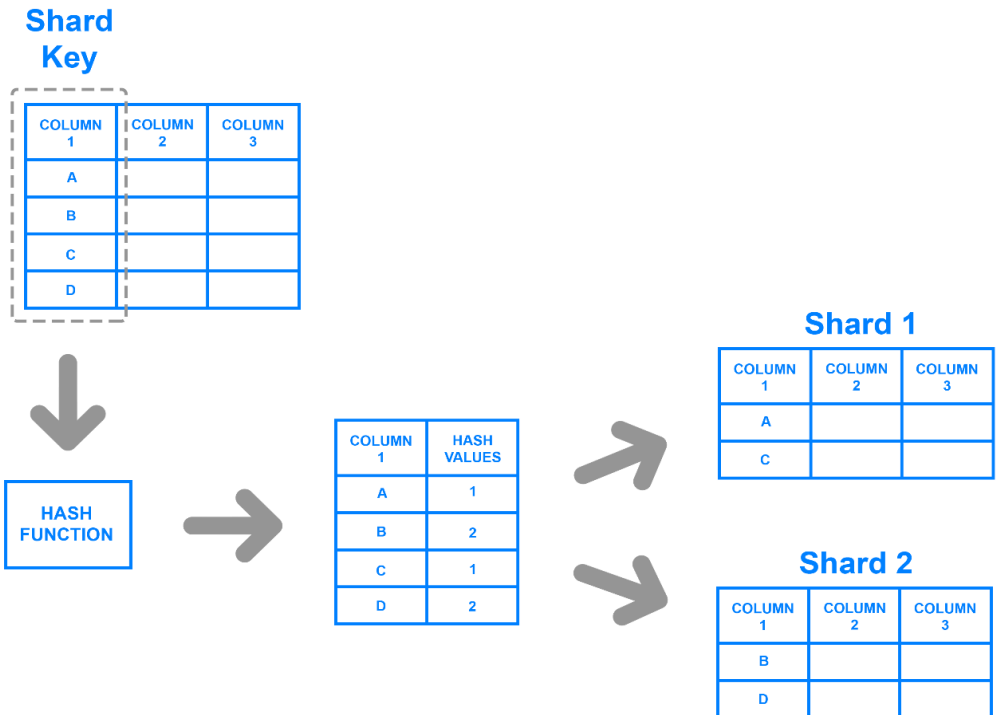
1. **Replikace** – Kopírování dat mezi jednotlivými uzly v clusteru.
    - Cílem je zvýšit odolnost a propustnost clusteru.
2. **Rozdělení** – Rozmístění částí dat mezi jednotlivé uzly v clusteru.
    - Cílem je zvýšit kapacitu a výkon clusteru.

### 1.5.1 Rozdělení dat

V případě použití **horizontálního škálování** je nutné si data vhodně rozmístit. Podmnožiny s daty nemusí být nutně disjunktní. V takovém případě, pak je možné hovořit o kombinaci rozdělení a replikace. Způsob rozmístění je velice důležitý. Při návrhu je dobré si zodpovědět otázku proč vlastně je žádoucí data rozdělit. Odpověď na tuto otázku totiž může zjednodušit návrh výsledné architektury. Důvody rozdělení mohou být následující.

- Rovnoměrné rozmístění dat mezi uzly.
- Optimalizace rychlosti odpovědí na dotazy.
- Optimalizace fyzického uložení dat.

Rozdělení probíhá pomocí hodnot sloupce nazývaného jako „sharding key“. Jednotlivé uzly, na které jsou data rozmísťovány, se pak nazývají „shards“. Hodnoty mohou být transformovány do správného rozsahu pomocí hashovací funkce, jak je vidět na obrázku (Obr. 1).




Obr. 1 Příklad rozdělení dat pomocí hashovací funkce. [37]

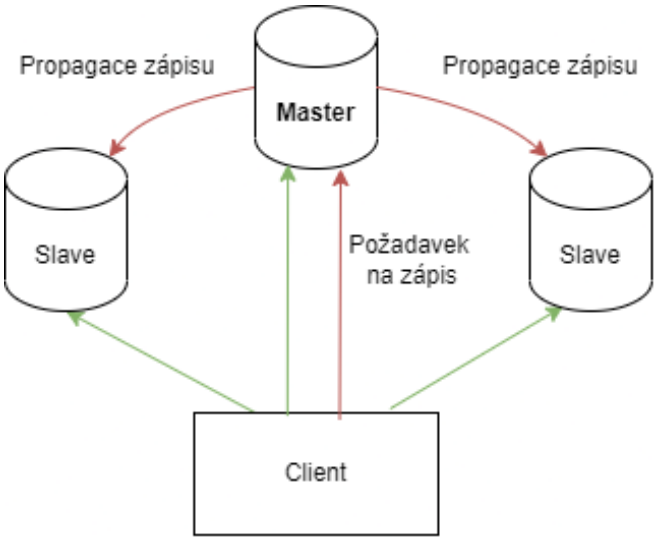
### 1.5.2 Replikace

Pokud je cílem **zvýšit odolnost a robustnost** databáze je vhodné použít replikaci. Zjednodušeně se dá říct, že stejná data jsou uložená na více uzlech v clusteru. Tímto se zajistí, že data budou k dispozici i při výpadku jednoho z uzlů. Další výhodou může být snížení odezvy při dotazu na data, jelikož replikovaný uzel může ležet geograficky blíže klientovi, či může být méně zatížen. Požadovaný počet replik pak určuje tzv. **replikační faktor**. [4]

Podobně jako u rozložení dat i zde se musí vhodně určit architektura replikace. Mezi nejznámější architektury patří.

**Mater-slave**

Jeden z uzlů je určen jako primární (master), ostatní se pak stávají sekundárními (slave). Požadavky na čtení dat může obsloužit libovolný uzel, zatímco požadavky na zápis dat musí vždy zpracovat primární uzel. Ten poté informuje sekundární uzly o změně. Díky tomu je tento způsob replikace vhodnější pro databáze, kde předpokládáme, že bude počet čtení převyšovat počet zápisů. Propagaci požadavků v této architektuře je možné vidět na obrázku (**Chyba! Nenalezen zdroj odkazů.**).




Obr. 2 Ukázka čtení a zápisu v replikaci master-slave.

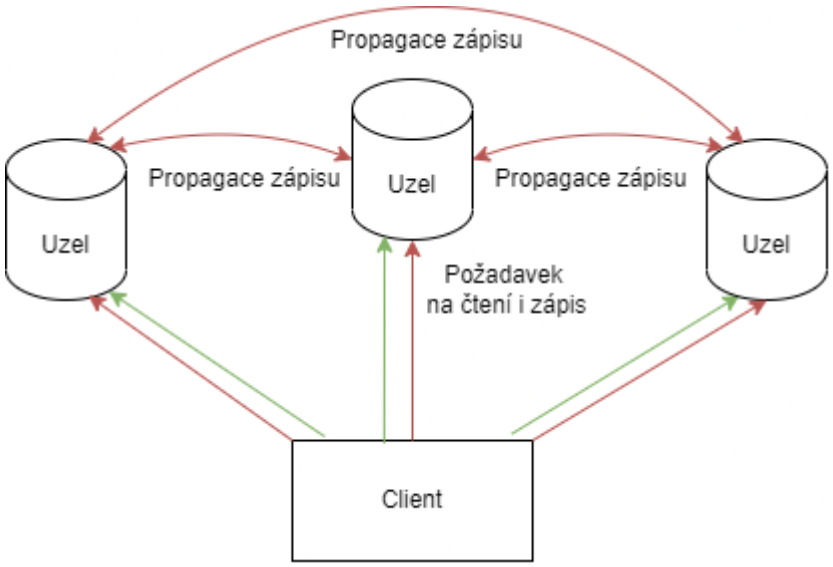
**Peer-to-peer**

Tento způsob replikace řeší primární nevýhodu architektury master-slave, která tkví v přetěžování master uzlu při větším počtu zápisů. V peer-to-peer replikaci mohou všechny uzly obsluhovat požadavky pro čtení i zápis. Díky tomu však mohou vznikat problémy

v dodržování konzistence dat mezi jednotlivými uzly. Tyto problémy se pak dají v zásadě řešit dvěma způsoby.

- Hlasováním – O správnosti dat rozhodne většina.
- Koordinací – Jednotlivé uzly si mezi sebou dávají vědět o změnách.

Propagaci požadavků v této architektuře je možné vidět na obrázku (Obr. 2).




Obr. 2 Ukázka čtení a zápisu v replikace peer-to-peer.

# 2 UKLÁDÁNÍ BIG DAT

Zatímco v první kapitole se nachází především obecný popis principů z oblasti big dat, tato kapitola se věnuje možnostem jejich uchování. Obvykle se data běžné velikosti uchovávají např. v klasických relačních databázích. S nárůstem počtu dat se však tyto způsoby uchování začínají jevit jako nedostačující. Je to způsobeno především tím, že v době vzniku klasických databázových databází bylo ukládané množství dat řádově nižší.

Existují systémy, které byly vyvinuty přímo pro uchovávání větších objemů dat. Obvykle mají podporu pro práci v clusteru a s tím spojené horizontální škálování. Některé z nich používají obdoby jazyka SQL (Structured Query Language) pro definici i manipulaci s daty. Jiné nástroje mohou používat vlastní dotazovací jazyky.

## 2.1 Datové sklady

Technologie, která má kořeny již v 90. letech minulého století, je stále dnes hojně využívána pro analýzu zvanou **business inteligence**. Důvod jejich vzniku byl takový, že tradiční relační databáze nesplňovaly požadavky, které na ně byly s ohledem na kvalitativní analýzu dat kladeny. Nicméně právě relační databázové databáze tvoří jádro datových skladů. Byť jsou upravené oproti těm klasickým.

### 2.1.1 Definice

Formální definice datového skladu, charakterizuje datový sklad jako soubor integrovaných, stálých, subjektově orientovaných a časově neměnných dat pro podporu manažerských rozhodnutí [5]. Z definice je poměrně zřejmé, že se nejedná pouze o systém k uchování dat. Většina datových skladů má integrované i nástroje pro analýzu, a to především právě pro podporu rozhodování v rámci strategického řízení. Jelikož se často tyto data sbírají z různých systémů, je jich hodně a rychle přibývají lze je považovat za big data.

**Integrovanost**

Data jsou obvykle sbírána z několika interních i externích zdrojů. Mezi těmito zdroji často bývají velké rozdíly, co se týče operačních systémů či formátu uložení dat.

**Stálost**

Data, která jsou v datových skladech již uložena, jsou určena pouze pro čtení. To znamená, že je nelze jakkoli modifikovat a data jsou tak uložena ve stejné podobě po dobu několika let.

### Orientace na subjekt

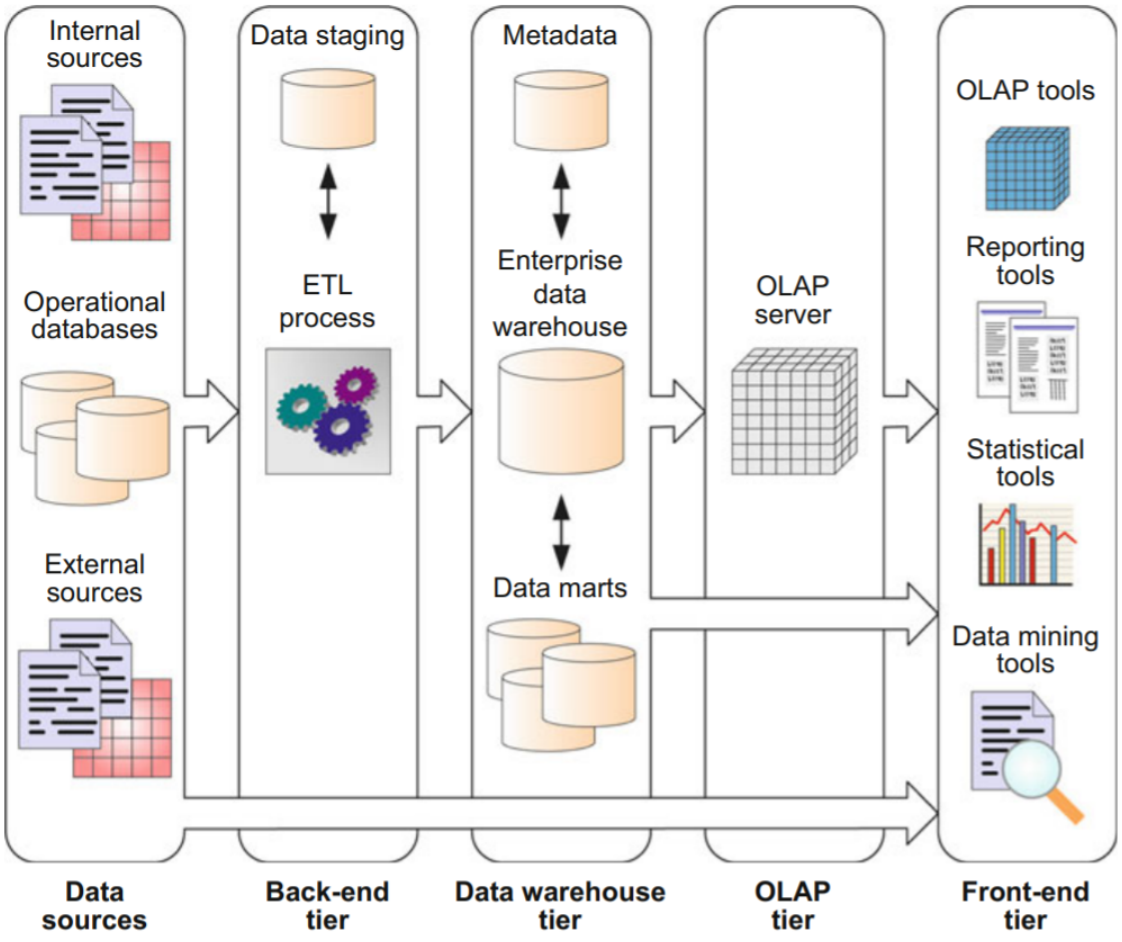
Datové sklady se snaží data uchovávat v podobě, která je vhodná pro následnou analýzu. Výsledné uložení se pak liší v závislosti na potřebách dané organizace. Tím se liší od klasických relačních databází, které se snaží data uchovávat s ohledem na nejmenší redundanci.

### Časová variabilita

Data jsou udržovaná v jejich historických podobách, nikoli jen v aktuální verzi. Jsou tedy zaznamenány tak, aby byl patrný jejich vývoj v čase. Díky tomu je pak možné provádět komplexní časové analýzy za určité historické období.

## 2.1.2 Architektura

Architektura obecných datových skladů se skládá z několika vrstev. Tyto vrstvy se v různých publikacích mohou lehce lišit, princip jejich dělení však zůstává stejný. Jednotlivé vrstvy jsou viditelné na obrázku (Obr. 3).




Obr. 3 Architektura datového skladu. [5]

**Back-end vrstva**

Zajišťuje extrakci, transformaci a načítání dat do datového skladu z několika různých zdrojů. Tento proces bývá označován jako ETL (Extraction, Transformation and Loading). [5]

**Vrstva datového skladu (Data warehouse tier)**

Slouží k ukládání dodávaných dat. Skládá se z podnikového datového skladu, metadat a datových tržišť, což jsou speciální datové sklady určené pro jednotlivá oddělení v rámci společnosti. Uložená data v nich mohou být odvozena z podnikového datového skladu nebo přímo sbírána z datových zdrojů.

**OLAP vrstva**

Obsahuje tzv. OLAP (Online Analytical Processing) server, který umí poskytovat multidimenzionální pohledy na data, a to bez ohledu na způsob jejich uložení. Často je možné se s pojmem OLAP setkat i mimo datové sklady. Technologie je v oblasti business inteligence velice oblíbená, neboť nabízí srozumitelné pohledy na multidimenzionální data, která jsou v jejich syrové podobě pro běžné uživatele neuchopitelná. [5]

**Front-end vrstva**

Umožňuje uživatelům pohodlně pracovat s daty uloženými v datovém skladu. Obvykle obsahuje nástroje pro OLAP analýzu, reportovací nástroje, statistické nástroje nebo také nástroje pro data mining.

### 2.1.3 Struktura

Data jsou v datovém skladu strukturována odlišným způsobem než v klasických relačních databázích. Jsou členěna do **schémat**. Přičemž každé schéma by mělo odpovídat jedné analyzované funkční oblasti.

Schéma se skládá z jedné nebo několika:

- **Faktových tabulek**
  - Jsou v nich jsou uložena samotná data.
  - Zabírají většinu kapacity datového skladu.
  - Granularita určuje úroveň podrobnosti. Čím je nižší, tím detailnější je pohled na data.
  - Jsou spojeny pomocí cizích klíčů s dimenzemi.

- **Dimenzionálních tabulek**
  - Obsahují seznamy hodnot, které jsou určené pro kategorizaci a třídění dat ve faktových tabulkách.

## 2.2 Úvod do NoSQL databází

Přestože se jedná o poměrně nové odvětví, rychle si získalo přízeň řady uživatelů. Dalo by se říct, že jde o trend poslední doby. Je možné, že zájem postupně opadne a firmy se začnou vracet ke klasickým relačním databázím, zatím však trend vypovídá o opaku.

I když alternativy k relačním databázím se na trhu vyskytovaly snad od dob jejich vzniku, termín „NoSQL“ byl poprvé použit až v roce 1998. Použil ho Carlo Strozzi jako název pro svou open source relační databázi, která používala pro komunikaci s klienty jiné rozhraní než standardní SQL. Do širšího povědomí se pak termín dostal, když ho v roce 2009 zmínil Eric Evans ve spojení s open source distribuovanými databázemi. Neodkazoval na konkrétní systém nebo technologii, ale spíše tím myslel posun od klasických relačních databází. [6]

Spojení „NoSQL“ dnes již neznamená označení pro technologie, které se nutně musí distancovat od jazyka SQL. Lépe popisuje dnešní systémy tvrzení „Not only SQL“, které svým významem vystihuje skutečnost, že tyto technologie mohou používat jazyky odvozené z jazyka SQL. Stále však kolem tohoto pojmu panují nejasnosti. Je tedy zřejmé, že jedná o databáze, které pro uchování dat používají jiné principy a postupy než klasické relační databáze.

### 2.2.1 Princip

Jelikož se jedná o poměrně nové odvětví databází a jejich vývoj probíhal v době, kdy množství zpracovávaných dat bylo obecně mnohem větší, než tomu bylo při vývoji klasických relačních databází. Dá se říct, že NoSQL databáze jsou připravenější, co se týče zpracování big dat. To především díky nativní podpoře distribuovaného zpracování v clusteru.

Nicméně v distribuovaném prostředí bývá problémem dodržení plně transakčního zpracování dotazů s nejvyšší úrovní izolace jednotlivých transakcí, které bývá v klasických relačních databázích bez podpory horizontálního škálování „běžnou praxí“. I tento problém se však dá, byť s mírnými ústupky, řešit. Částečně se tomuto problému věnuje tzv. CAP (Consistency, Availability and Partition tolerance) teorém, jehož popis je v samostatné sekci. Jestli je výhodné se vzdát podpory všech vlastností ACID na úkor distribuovaného zpracování v clusteru, pak záleží vždy na konkrétním systému a jeho případech užití.

U klasických relačních databází bývají data atomizována do jednotlivých tabulek. Což umožňuje efektivně realizovat operace zápisu, ale také může způsobovat nárůst režie při čtení z vícero tabulek současně. Většina NoSQL databází poskytuje možnosti, jak upravovat strukturu vzhledem k případu užití konkrétní databáze. To se provádí většinou s ohledem na frekvenci jednotlivých dotazů tak, aby ty nejčastěji prováděné byly co nejrychlejší.

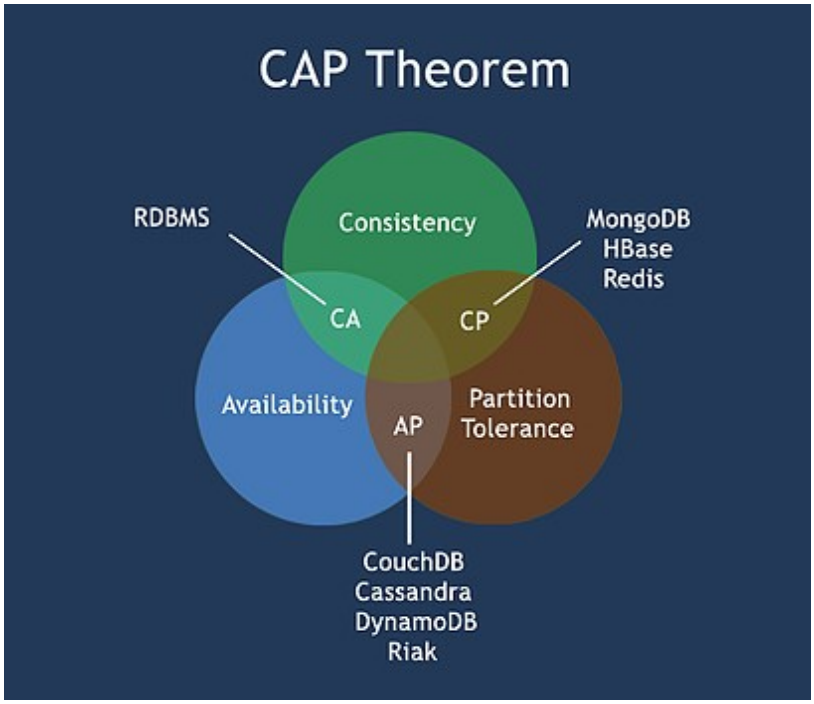
Flexibilita databázové struktury je jednoznačně jeden z největších rozdílů oproti klasickým relačním databázím. U nich se totiž předpokládá, že se struktura dat navrhne při tvorbě databáze a poté se mění jen velmi výjimečně. I proto je často jazyk pro definici dat – DDL (Data Definition Language), pomocí kterého se v relačních databázích mění struktura, dostupný často pouze vybraným uživatelům. I tak může být změna databázové struktury „za chodu“ poměrně složitá, často až nebezpečná. U NoSQL databází je datové schéma více flexibilní a proměnlivé. Úroveň této flexibility se liší v závislosti na konkrétní databázi. Některé typy dokonce vůbec nedefinují strukturu dat. Struktura uložení je tedy plně v režii aplikace nebo uživatele.

### 2.2.2 CAP teorém

Velký problémem distribuovaného prostředí je dodržení všech vlastností ACID. Proto se zavádí pojem CAP, který definuje ideální vlastnosti, které by měl distribuovaný systém splňovat.

- Konzistence (Consitency)
    - Operace čtení vždy musí vrátit aktuální data.
- Dostupnost (Availability)
    - Systém musí obsloužit všechny požadavky na něj kladené.
- Odolnost vůči rozpadu sítě (Partition tolerance)
    - Systém musí fungovat i při rozpadu sítě.

V ideálním případě by systém splňoval všechny tři tyto vlastnosti. Nicméně v distribuovaném prostředí je možné dosáhnout splnění maximálně dvou vlastností. Přičemž odolnost vůči rozpadu sítě je v distribuovaných systémech nutnou podmínkou. Takže zbývá si zvolit mezi dostupností a konzistencí. Většina NoSQL databází se snaží najít mezi těmito vlastnostmi kompromis. Jednotlivé systémy se pak odlišují právě různou prioritizací těchto dvou vlastností. Na obrázku (Obr. 4) jsou tyto vlastnosti zobrazeny pomocí Vennových diagramů.




Obr. 4 CAP vlastnosti zobrazené pomocí Vennových diagramů. [7]

## 2.3 Databáze typu klíč-hodnota

Funguje na obecně známém způsobu ukládání dat, který se používá mimo databáze také např. jako abstraktní datová struktura v některých programovacích jazycích. Data se ukládají ve dvojích klíč-hodnota do tzv. asociativního pole. Klíč pro uložení dvojice musí být v daném kontextu unikátní. Podle tohoto klíče dochází k indexaci konkrétní hodnoty. Hodnota, pak obsahuje ukládaná data.

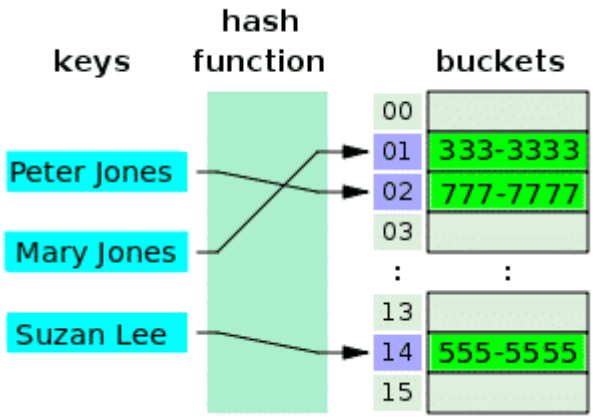
### 2.3.1 Princip indexace

Asociativní pole bývá nejčastěji implementováno pomocí hashovací tabulky. Jedná se o pole s hodnotami, na jehož indexy jsou mapovány klíče. Mapování pak probíhá transformací klíče na index pomocí hashovací funkce. Obecnou rovnici hashovací funkce pro klíč $\varphi$ definovaný k-ticí $(x_1, x_2, \dots, x_k)$ lze vidět ve formuli (1), jež byla převzatá z [8].

$$\varphi\big((x_1, x_2, \dots, x_k)\big) = \sum_{i=1}^{k} x_i p^{i-1} \bmod m \tag{1}$$

kde $p$ je vhodné prvočíslo – dostatečně velké a nesoudělné s $m$. Jestliže je klíč definován jako znak nebo k-tice znaků je nutné pro výpočet za $x_i$ dosadit ordinální hodnotu tohoto znaku.

Obrázek (Obr. 5) ukazuje, jak probíhá hashovaní a následné uložení dat do hashovací tabulky, kde pojem „buckets“ reprezentuje jmenné prostory klíčů (viz. 2.3.3).




Obr. 5 Princip ukládání do hashovací tabulky. [9]

Může se stát, že i přes unikátnost klíčů, transformuje hashovací funkce dva rozdílné klíče na stejný index a dochází ke kolizi. Taková situace se pak může řešit dvěma způsoby.

1. **Samostatné řetězení**
   - V samostatném řetězení, když dojde ke kolizi, je nová hodnota přidána do seznamu za hodnotu, která již na tomto indexu existuje. Tudíž se může stát, že dvě nebo tři hodnoty nakonec mají stejný index. Konečné vyhledání hodnoty na konkrétním indexu pak probíhá pomocí algoritmu pro vyhledávání v seznamu. [9]
2. **Otevřené adresování**
   - Při otevřeném adresování jsou všechny hodnoty uloženy v poli slotů. To znamená, že nedochází k řetězení jako v předchozím případě. Když dojde ke konfliktu, nová hodnota se vloží do jiného, algoritmem určeného, prázdného slotu. Při vyhledávání je pak nutné projít stejnou posloupnost indexů, dokud není prvek nalezen. [9]

Kolize nastává, jelikož je počet možných vstupů větší než počet různých výsledků hashovací funkce. Proto je žádoucí zvolit funkci s co nejmenším počtem kolizí. Tím se sníží režie při vyhledávání v tabulce. Asymptotická časová složitost vyhledávání v tabulce bez kolizí je **O(1)**. Což je velice výhodné, protože konstantní složitost je nejlepší možná složitost, které může algoritmus dosáhnout.

### 2.3.2 Rozhraní databáze

Uložiště typu klíč-hodnota, bývají obecně velice jednoduchá, a to jak v rámci struktury ukládaných dat, tak i v rámci rozhraní, které databáze poskytují pro práci s daty [4]. Jádrem rozhraní jsou operace inspirované protokolem HTTP (Hypertext Transfer Protocol). Některé databáze přímo podporují přístup k datům skrze HTTP REST (Representational State Transfer) rozhraní.

**GET**

Získání hodnoty pro daný klíč.

```
GET /buckets/zbozi/keys/asus-760K HTTP/1.1

{"nazev" : "Notebook Asus 760K", "cena" : "26000"}
```

**PUT**

Vložení hodnoty pro daný klíč.

```
PUT /buckets/zbozi/keys/asus-760K HTTP/1.1
-d {"nazev" : "Notebook Asus 760K", "cena" : "26000"}
```

**DELETE**

Smazání dvojice klíč-hodnota, podle daného klíče.

```
DELETE /buckets/zbozi/keys/asus-760K HTTP/1.1
```

Kromě HTTP rozhraní většina databází nabízí API (Application Programming Interface) pro konkrétní známé programovací jazyky jako je Java, C#, PHP, Python, Javascript a další.

### 2.3.3 Jmenné prostory klíčů

Většina těchto databází umožňuje alespoň částečně ukládaná data strukturovat pomocí rozčlenění podle jejich typu do oddělených úložišť nazývaných buckety [4]. Jedná se pouze o

logické rozdělení, fyzicky jsou data uložena stále v jednom hashovacím prostoru. Na úrovni komunikace s klientem je možné ukládat hodnoty se stejným klíčem v různých bucketech. V rámci připodobnění ke klasickým relačním databázím, by se buckety daly přirovnat k tabulkám (relacím).

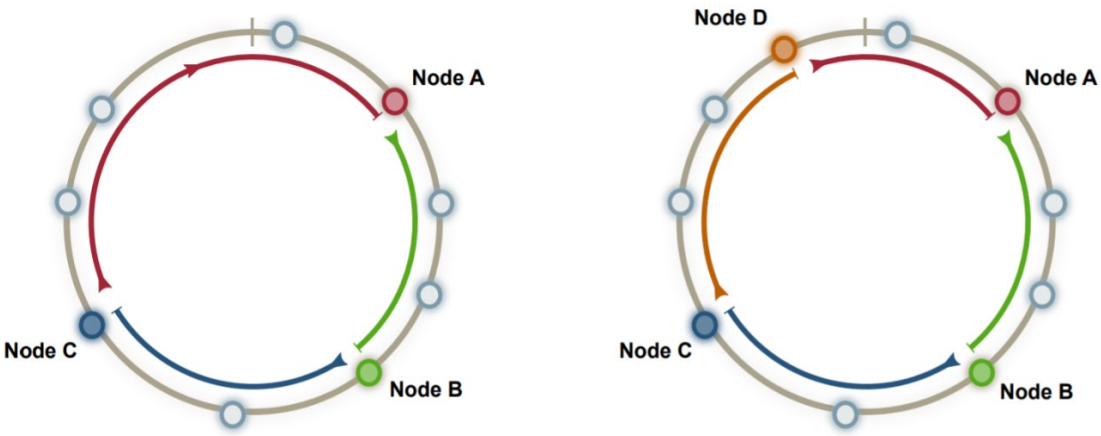
### 2.3.4 Distribuce dat

Jelikož většina NoSQL databází je stavěna pro distribuovaný provoz, je potřeba zajistit co možná nejrovnoměrnější rozložení dat mezi uzly. Nejjednodušším způsobem, jak toho docílit je použít zbytek po dělení hashované hodnoty počtem uzlů $N$, tak jako ve formuli (2). [4]

$$\check{c}\acute{i}slo\ uzlu = hash(kl\acute{i}\check{c})\ \%\ N \tag{2}$$

Zde nastává problém, pokud je nutné vložit nový uzel do již zaběhnutého clusteru. Změní se totiž počet uzlů $N$ a tím i výsledné hodnoty čísel uzlů. Řešením je použít princip tzv. konzistentního hashovaní.

**Konzistentní hashování**




Obr. 6 Rozdělení intervalů při konzistentním hashování. [10]

Uzlům se přidělí klíč ze stejného rozsahu jako má hashovací funkce pro klíče k hodnotám. Každý uzel spravuje všechny klíče v intervalu mezi klíčem předcházejícího uzlu a svým klíčem [4]. Hashovací prostor poté figuruje jako kruh, to znamená, že zmíněný interval mezi dvěma uzly může přecházet přes nulu. Při přidání uzlu, pak systém rozdělí jeden interval přidáním nového uzlu a přesune do něj část dat, které mu podle intervalu nyní náleží. Tento princip je možné vidět na obrázku (Obr. 6).

### 2.3.5 Použití

Mezi hlavní výhody databází typu klíč-hodnota patří především jednoduchá struktura a rychlý přístup k datům. Právě díky těmto přednostem se prosazují v aplikacích, jež provádí velké množství operací nad daty, které nemají složitou strukturu. Stejně jako ostatní typy NoSQL databází se vyznačují větší mírou rozličnosti mezi jednotlivými databázemi, než je tomu u klasických relačních databází. Primárně lze databáze typu klíč-hodnota použít jako:

- vyrovnávací paměť
  - Memcached, Ehcache, Hazelcast
- knihovny pro tvorbu vestavěných diskových úložišť
  - Berkeley DB, LevelDB, RocksDB
- perzistentní distribuované systémy
  - Riak, Redis

## 2.4 Sloupcové databáze

Další typ NoSQL databází se označuje anglickým výrazem column family stores či wide column stores. Díky tomuto anglickému názvu se někdy chybně tyto databáze zaměňují se sloupcově orientovanými relačními databázemi (column-oriented DBMS). To jsou však klasické relační databáze, které se liší především ve struktuře ukládaných dat. Databáze orientovaná na sloupce serializuje všechny hodnoty sloupce dohromady. Zatímco sloupcové NoSQL databáze mají od klasických relačních databází více rozličností.

Sloupcové databáze používají pro ukládání dat struktury známé z relačních databází jako jsou sloupce či řádky. Nicméně na rozdíl od nich se formát a název sloupců může lišit řádek od řádku v jedné tabulce. Každý sloupec je pak z hlediska fyzického uložení dat uložen samostatně. [11]

### 2.4.1 Struktura datového modelu

Základním prvkem v těchto databázích je **řádek** identifikovaný klíčem. Řádek pak může mít několik **sloupců**, které mají v daném řádku svůj název, hodnotu a časové razítko, které říká, kdy byla hodnota uložena. Sloupce jsou pak sdružovány do tzv. **rodin sloupců** (column families).

Rodiny sloupců by se daly přirovnat k tabulkám v klasických relačních databázích, protože sdružují sloupce, které nesou související informace. Nicméně rozdíl je v tom, že při návrhu

databáze se definují pouze rodiny sloupců. Ukládané řádky pak mohou mít v rámci dané rodiny různé sloupce s různými názvy. Také poskytují možnost vertikálního rozdělení jednotlivých sloupců napříč uzly v clusteru. Jak zhruba může taková rodina sloupců vypadat je možné vidět v tabulce (Tabulka 1).

Tabulka 1 Ukázka rodiny sloupců pro zboží.

| **zbozi_id** | **Název sloupce** | **Název sloupce** | **Název sloupce** |
|---|---|---|---|
| | Hodnota | Hodnota | Hodnota |
| 1 | **nazev** | **cena** | **hodnoceni** |
| | Notebook Asus 760K | 26000 | 9.1 |
| 2 | **nazev** | **pocet kusu** | |
| | Raspberry Pi 4 2GB | 19 | |

Kromě jednoduchých sloupců umožňují některé databáze vytvářet také tzv. **supersloupce** [4]. Jedná se o sloupce, jejichž hodnota je složená z podsloupců. Použití je možné najít v situacích, kdy se ukládají neatomické hodnoty, které by se v klasických relačních databázích podle první normální formy (1NF) ukládaly rozděleně. Příkladem může být adresa objektu, která je složena z města, ulice a popisného čísla budovy.

Stejný klíč řádku lze použít v různých rodinách sloupců. Tyto řádky pak dohromady většinou tvoří logický celek.

### 2.4.2 Rozhraní databáze

Způsob komunikace s databází se liší v závislosti na konkrétní databázi. Některé používají jazyky vycházející z SQL, jiné zase nabízí odlišné způsoby dotazování. Pro vytvoření představy o komunikaci se sloupcovou databází bude představen způsob komunikace pomocí dotazovacího jazyka **CQL** (Cassandra Query Language), který používá systém **Apache Cassandra**.

Jazyk CQL vychází ze standardních konstrukcí jazyka SQL. Množina příkazů se však od standardních SQL dotazů lehce liší. Dotaz v jazyce CQL pak může vypadat např.:

```
SELECT nazev, pocet_kusu
FROM zbozi
WHERE zbozi_id = 2
ORDER BY pocet_kusu ASC;
```

Výše uvedený příkaz pro selekci a projekci dat je syntakticky podobný příkazu v jazyce SQL. Nicméně i zde se najdou odlišnosti, které plynou především z distribuovaného prostředí. Databáze Cassandra nevyhodnotí složitější dotazy. Konkrétně složité omezující podmínky v části WHERE. Také umí dotazovat pouze jednu tabulku[2] v rámci jednoho dotazu.

Syntaxe jazyka CQL pro definici dat i manipulaci s daty je téměř totožná se standardní SQL syntaxí. Vložení záznamu do tabulky může vypadat např.:

```
INSERT INTO zbozi (_id, nazev, pocet kusu)
VALUES (3, 'Sencor SES 1701BK', 13);
```

Může to vypadat, jako kdyby systém Apache Cassandra byl pozadu oproti klasickým relačním databázím. Dalo by se říct, že tyto omezení jsou daní za možnost horizontálního škálování. Kvůli tomu také není možné dosáhnout plně transakčního zpracování dotazů, ale používají se tzv. **odlehčené transakce**. Jsou to speciální složené příkazy, které jsou vyhodnocovány atomicky. [4]

### 2.4.3 Distribuce dat

Rozdělení dat mezi uzly probíhá pomocí klíče tak jako u databází typu klíč-hodnota. Přesto se princip lehce liší, jelikož výběr konkrétního uzlu většinou neprobíhá pomocí výsledku hashovací funkce, ale přímo podle hodnoty klíče. Ta je rozdělena do určitých intervalů, z nichž každý náleží jednomu uzlu. Pokud pak hodnota spadá do tohoto intervalu, je záznam umístěn na daný uzel.

Způsob distribuce se liší v závislosti na konkrétní databází. Důležité ale je, že téměř všechny tyto databáze mají nativní podporu distribuce i replikace dat.

[2] Apache Cassandra používá pojem tabulka pro označení rodiny sloupců.

## 2.5 Dokumentové databáze

Jedná se o asi **nejpoužívanější typ** NoSQL databází[3]. Jak již název napovídá, hlavním prvkem těchto databází je dokument. Nejedná se, ale o dokument, který ukládá formátovaný text jako např. Microsoft Word Document. Dokumentem se v kontextu NoSQL databází myslí datová struktura, která má samopopisný charakter. Což znamená, že kromě samotných dat obsahuje také metadata popisující význam jednotlivých částí datové struktury [4]. Další vlastností dokumentů je jejich stromová struktura.

Stejně jako v ostatních typech NoSQL databází, se i zde najdou značné rozdíly i mezi konkrétními dokumentovými databázemi. Proto při popisu datového modelu bude použita struktura dat, kterou používá nejznámější dokumentová databáze **MongoDB**. Ostatní databáze se mohou ve formátu či struktuře ukládaných dat lišit.

### 2.5.1 Struktura datového modelu

Konkrétní struktura se odvíjí od použitého jazyku pro ukládání dat. Mezi nejrozšířenější patří XML (Extensible Markup Language) a v poslední době především JSON (JavaScript Object Notation) nebo jeho binární varianta tzv. BSON (Binary JSON). Formáty XML i JSON jsou velice výhodné při mapování na objekty v některém z objektově orientovaných jazyků. Také se tyto formáty často používají při přenosu informací mezi aplikacemi např. pomocí REST API. Moderní webové aplikace a služby s těmito formáty často již pracují a není tedy problém je využít i pro persistenci. Právě to je bezesporu jeden z důvodů proč se dokumentové databáze těší tak velkému úspěchu.

Podstatnou vlastností dokumentů je flexibilita. Přestože je poměrně dobrým zvykem uspořádávat dokumenty stejných resources[4] spolu do tzv. kolekcí, jejich struktura se může lišit dokument od dokumentu bez explicitní změny schématu databáze. Jedinou povinnou položkou bývá identifikátor dokumentu, který je v rámci kolekce unikátní. Na následujícím příkladu je vidět právě rozličnost 2 dokumentů zapsaných v jazyce JSON, které by byly ukládány do stejné kolekce.

[3] https://db-engines.com/en/ranking
[4] Resource lze chápat jako objekty stejného typu. Např. Zboží, Zákazník...

```
{
    "_id": "1",
    "nazev": "Notebook Asus 760K",
    "cena": "26000"
}
{
    "_id": "2",
    "nazev": "Raspberry Pi 4 2GB",
    "pocet kusu": "19"
}
```

Dokumentové databáze, na rozdíl od klasických relačních, nabízí dva způsoby modelování vztahů mezi entitami.

**Vnořené dokumenty**

Dokument je součástí rodičovského dokumentu jako jeden z jeho atributů. To platí při modelování vztahu 1:1. Pokud bychom modelovali vztah 1:N, tak by v atributu rodičovského dokumentu byl seznam dalších sub-dokumentů. Tento způsob je v jazycích JSON a XML poměrně „běžnou praxí“. Problémem je, že nelze modelovat vztah M:N. Výhodné je použití především v situacích, kdy je dopředu známé, že se bude ve většině případů pracovat s rodičovským dokumentem společně se sub-dokumenty.

```
{
    "_id": 1,
    "nazev": "Notebook Asus 760K",
    "cena": "26000",
    "vyrobce": {
        "nazev": "ASUSTek Computer Inc.",
        "zeme_puvodu": "Čínská republika"
    }
}
```

#### Použití odkazů

Odpovídá použití cizích klíčů v relačních databázích. Vztah je modelován pomocí identifikátorů do jiné kolekce. Výhodou je pak možnost modelování vztahů M:N a také snížení velikosti dokumentů v komplexnějším schématu.

```
{
    "_id": 1,
    "nazev": "Notebook Asus 760K",
    "cena": "26000",
    "výrobce_id": "12863"
}
```

### 2.5.2 Rozhraní databáze

Komunikační rozhraní se opět, stejně jako v jiných typech NoSQL databází, liší. Pro ilustraci bude vycházeno z aktuálního rozhraní systému MongoDB, které nabízí svůj vlastní dotazovací jazyk používající struktury jazyka BSON. Nicméně v ostatních dokumentových databázích je možné setkat se i s komunikací pomocí modifikovaných variant jazyka SQL, tak jako v předešlých typech NoSQL databází.

#### Čtení

Poměrně jednoduchá syntaxe, dělí dotaz pro provedení selekce na 3 hlavní části.

1. Specifikace kolekce (Collection)
2. Dotazovací kritéria (Query criteria)
3. Modifikátory aplikované na odpověď (např. Sort order)

Následující příklad vybere položky z kolekce „zbozi“ s cenou větší než 10000 seřazené podle ceny sestupně.

```
db.zbozi.find({ cena: { $gt: 10000 }}).sort({ cena: 0 })
```

#### Zápis

Pro zápis se používá stejná syntaxe jako pro selekci. Následující příklad vloží nový dokument do kolekce „zbozi“.

```
db.zbozi.insert{"nazev":"Notebook Asus 786K", "cena":"36000"})
```

Identifikátor dokumentu je pro nově uložený dokument vždy automaticky vygenerován. V případě aktualizace již existujícího dokumentu se použije operace **update** a dokument se

opatří identifikátorem. Přičemž v příkazu se definují pouze atributy u BSONu pouze atributy, které se mění.

V následujícím příkladu je možné vidět, jak dokument z kolekce odstraníme.

```
db.zbozi.remove{"_id": 1 })
```

Kromě základního rozhraní, nabízí systém MongoDB další možnosti přístupu k jejich databázi. Disponuje totiž širokou škálou knihoven pro různé nejpoužívanější programovací jazyky. Knihovny pak definují jednoduché API pro snadnou integraci aplikace a databáze.

### 2.5.3 Distribuce dat

Dokumentové databáze používají podobné principy replikace a rozdělení dat jako předchozí typy NoSQL databází.

**Replikace**

Systém MongoDB využívá principů master-slave replikace. Celý proces funguje tak, že klient navazuje komunikaci s některým z uzlů, jehož adresu zná. **Ovladač MongoDB** v knihovně pro daný programovací jazyk může vhodně přesměrovávat požadavky na jednotlivé uzly. Požadavky pro zápis směruje vždy na uzel master, zatímco požadavky na čtení s využitím load balancingu na libovolné uzly podle jejich vytíženosti.

Cluster obsahuje mechaniky pro zajištění co možná nejlepší konzistence dat. Např. při výpadku uzlu master si ostatní uzly v clusteru dokážou automaticky zvolit nový master uzel.

**Rozdělení dat**

Probíhá podobně jako v jiných NoSQL databázích. Využívá se primární identifikátor dokumentu nebo umělý klíč přímo pro rozdělení (sharding key). Systém MongoDB obsahuje směrovač, který pomocí informací od konfiguračního serveru rozhoduje o správném umístění dat na jednotlivé uzly či množiny replik. Na rozdíl od replikace se směrování na konkrétní uzly provádí až na straně serveru.

## 2.6 Grafové databáze

Zatímco předcházející typy NoSQL databází pro ukládání dat používaly model sdružovaných struktur stejného typu, kde jednotlivé záznamy jsou indexované pomocí unikátních klíčů. Data v grafových databázích jsou uložena vzhledem k přechozím třem typům zcela **odlišně**, a to v grafu.

### 2.6.1 Struktura dat

Grafem se v kontextu tohoto typu databází myslí množina uzlů a hran. Díky tomu tato struktura slouží k ukládání objektů a vztahů mezi nimi.

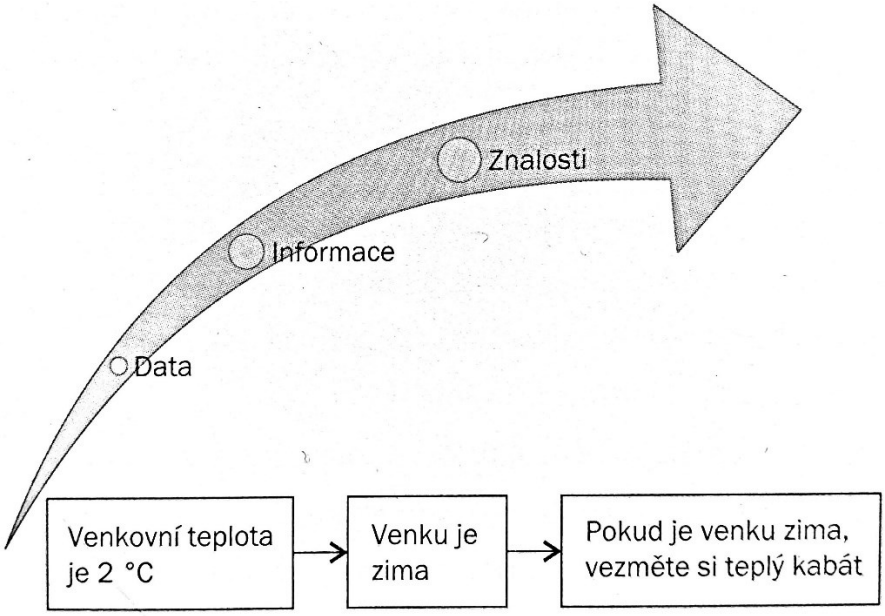
- **Uzel** – odpovídá jednomu objektu
    - Může obsahovat atributy (vlastnosti).
- **Hrana** – reprezentuje vztah mezi objekty
    - Může mít i směrový význam.
    - Taktéž může obsahovat atributy.

Jelikož je tento typ databází zaměřen na poněkud specifičtější problémy, které nejsou předmětem této práce, nebude rozebrán tak podrobně jako ostatní typy NoSQL databází. Také se tento typ databází příliš nehodí pro zpracování big dat, protože většina implementací nemá nativní podporu distribuce dat nebo je velmi omezená.

# 3 DATA MINING

Rychle narůstající množství dat může být pro společnost problémem, ale i přínosem. Data mohou být obtížně interpretována. Zasazením do kontextu se z nich, ale stávají informace a ty už mohou být velice cenné. Zvláště pak, když se rozhodneme z nich získat znalosti a tím zlepšovat svoje služby.

O **znalostech** se může hovořit v okamžiku, kdy se data a informace změní na sadu pravidel, které pomáhají v rozhodování. Znalosti není možné uložit, protože zajišťují teoretické i praktické porozumění dané oblasti. Příklad převodu dat na znalosti je možné vidět na následujícím obrázku (Obr. 7). [12]




Obr. 7 Příklad převodu dat na znalosti. [12]

## 3.1 Definice pojmu

Právě převod dat na informace či znalosti je úlohou procesu, který bývá nepřesně označován jako data mining. Definovat pojem je možné různě a určitě se nedá říct, že by existovala jen jedná správná definice. Jako příklad uveďme tuto definici: *Data mining je proces, který nalézá užitečné vzory a trendy ve velké množině dat* [13].

Fakt, že se jedná o nepřesný termín, je odvozen z významu jednotlivých slov „data“ a „mining“ (v překladu „těžba“). V českém překladu tak celý pojem vyzní jako „těžba dat“. Taková interpretace však špatně popisuje daný proces, jelikož se nejedná o získávání dat, ale

spíše o získávání informací z dat. Přesnější pojem pro označení tohoto procesu by pak mohl znít: „Knowledge mining from data“ nebo „Information mining from data“. Jelikož se ale pojem „data mining“ již zažil, nejspíše se pojmenování již nezmění. Nicméně je dobré o této nepřesnosti v názvu vědět.

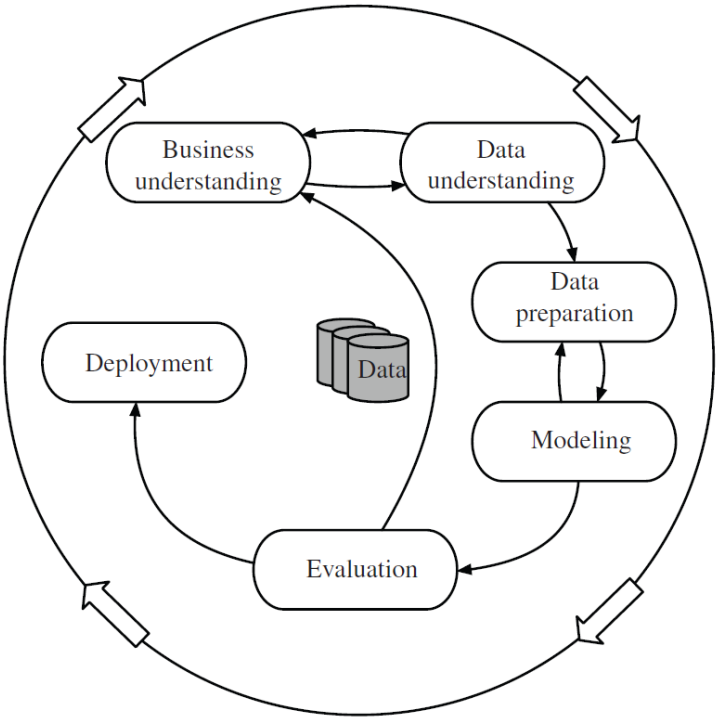
Pojem také bývá často zaměňován s pojmem KDD (Knowledge Discovery in Databases*)*. Data mining je však pouze ústřední fází celého procesu KDD, který se obecně věnuje získávání znalostí z dat. Data mining je tak vlastně pouze aplikace konkrétních algoritmů na základě cíle KDD. Časté zaměňování způsobuje, že i odborné publikace někdy hovoří o data miningu přestože popisují spíše fáze spadající do KDD. I tato práce se bude z části věnovat dílčím cílům, které by se měly zařadit spíše do KDD. [13]

Data mining pomáhá získávat z databází či jiných datových zdrojů přínosné informace a zákonitosti. Ty pak mohou být využity pro zkvalitňování dodávaných služeb, lepší cílení marketingových kampaní či jiné specifičtější vylepšení dodávaných produktů.

## 3.2 Data mining jako proces (CRISP-DM)

Když docházelo k prvním aplikacím data miningu napříč různými odvětvími, společnosti měly tendence přistupovat k data miningu nahodile a tím by mohlo dojít k jevu zvanému "zbytečně znovu-objevovat kolo". Bylo tedy zapotřebí vytvořit mezioborový standard, který by byl nezávislý na konkrétní doméně, na kterou je data mining aplikován. Standard CRISP-DM (Cross-Industry Standard Process for Data Mining) byl vyvinut v roce 1996 analytiky zastupujícími společnosti DaimlerChrysler, SPSS a NCR. CRISP-DM poskytuje nepatentovaný a volně dostupný standardní proces pro začlenění data miningu do obecné strategie řešení problémů podniku nebo výzkumné jednotky. [13]

CRISP-DM definuje projekt pro data mining jako životní cyklus skládající se z šesti po sobě jdoucích fází, které jsou zobrazeny na obrázku (Obr. 8). Jednotlivé fáze na sebe navazují, tudíž vstup následující fáze je velmi silně závislý na výstupu z aktuální fáze. Přičemž šipky mezi jednotlivými fázemi definují návaznosti těchto fází. Některé s fází mohou svým výstupem zpětně ovlivňovat přechozí fáze a tím iterativně vylepšovat mezistav procesu. Fakt, že se jedná o iterativní proces pak symbolizuje vnější šipka.




Obr. 8 Životní cyklus CRISP-DM. [14]

### 3.2.1 Fáze pochopení obchodu

V první fázi, označované též „Business understanding“, probíhá ujasnění cílů a požadavků projektu. Jedná se o standardní proces, který se vyskytuje i v dalších typech projektů, jež nemusí vůbec souviset s data miningem. Přesto je nutné této fázi věnovat dostatek prostoru, jelikož se zde definují věci, které jsou pro další směřování projektu klíčové.

- Definice cílů a požadavků projektu z hlediska podniku jako celku.
- Převod těchto cílů a požadavků do definice problému z hlediska data miningu.
- Příprava předběžné strategie pro dosažení těchto cílů.

### 3.2.2 Fáze porozumění datům

Tato fáze doplňuje předešlou a zaměřuje se na identifikaci, sběr a analýzu datových souborů, které vám pomohou dosáhnout cílů projektu.

- Shromáždění potřebných dat.
- Zjištění zajímavých informací o datech s pomocí průzkumné analýzy.
- Analýza kvality a čistoty dat. Případně dokumentace problémů spojených s kvalitou a čistotou.

- Identifikace podskupin v datech, které by mohly být z hlediska data miningu zajímavé.

### 3.2.3 Fáze přípravy dat

Fáze přípravy dat může být velice náročná, jelikož je jejím výstupem již finální dataset, který bude sloužit pro další fáze.

- Transformace nezpracovaných dat na dataset, který bude vstupovat do algoritmů.
- Výběr vhodných případů a proměnných, které se budou dále analyzovat.
- Provedení transformace hodnot určitých proměnných, pokud je to nutné.
- Čištění nezpracovaných dat tak, aby byla připravena pro modelovací nástroje.

### 3.2.4 Fáze modelování

Tato fáze spočívá ve vytváření a posuzování různých modelů založených na několika modelovacích technikách.

- Výběr a aplikace vhodných modelovacích technik. Výběr algoritmů, které by mohly stát za vyzkoušení. V závislosti na typu modelování může být nutné rozdělit množinu dat na trénovací, testovací a validační.
- Kalibrace nastavení modelů pro optimalizaci výsledků.
- Možné použití více různých modelovacích technik pro získání jiných výsledků.
- V případě nutnosti je možné se vrátit zpět do předchozí fáze a upravit dataset pro specifické potřeby konkrétní vybrané data mining techniky.

### 3.2.5 Fáze vyhodnocení

Fáze vyhodnocuje, který model nejlépe vyhovuje potřebám podniku.

- Vyhodnocení účinnosti modelů použitých v přecházející fázi před jejich nasazením.
- Vyhodnocení dosažení cílů z fáze obchodu a výzkumu.
- Zjištění, zda byly pokryty všechny důležité aspekty obchodního i výzkumného charakteru.
- Určení dalšího směru podle výsledků data miningu.

### 3.2.6 Fáze nasazení

Poslední fáze se věnuje nasazení modelu tak, aby byl co nejlépe využitelný.

- Využití vytvořených modelů.
- Vypracování plánu pro monitorování a údržbu.
- Shrnutí a závěrečná prezentace dosažených výsledků.
- Retrospektivní zhodnocení kvality dosažených výsledků a identifikace chyb pro případné další iterace.

Kromě standardizovaného procesu CRISP-DM lze jednotlivé kroky data miningu na určitý problém popsat i jinak. Je možné data mining definovat jako iterativní proces s cílem získávat zajímavé a využitelné informace z dat, které jsou k dispozici, pomocí následujících sedmi kroků. [15]

1. **Čištění dat** – odstranění šumu a nekonzistentních dat.
2. **Integrace dat** – kombinace více datových zdrojů.
3. **Selekce dat** – výběr dat relevantních pro analýzu.
4. **Transformace dat** – data jsou přetransformována do podoby, která je vhodná pro algoritmus data miningu.
5. **Data mining** – využití inteligentních metod pro nalezení užitečných vzorů a trendů.
6. **Vyhodnocení vzorů a trendů** – identifikace skutečně důležitých vzorů a trendů založená na vyhodnocení míry důležitosti.
7. **Prezentace informací** – prezentace informací získaných z dat.

Proces CRISP-DM spadá spíše do širší oblasti zvané KDD, která byla zmíněna na začátku této kapitoly. Samotný data mining je pak pouze jedním, za to ale nejdůležitějším krokem.

## 3.3 Úlohy data miningu

Data mining se věnuje především získávání informací z dat, která jsou nestrukturovaná a nijak neoznačená. V této sekci jsou nastíněny některé typické úlohy, pro které lze využít data miningu. Obecně lze jednotlivé úlohy rozdělit do dvou kategorií.

1. **Deskriptivní data mining**

   Nabízí možnost získání podrobného popisu dat. Zahrnuje veškeré informace, které umožňují pochopit, co se v datech děje bez předchozí představy. [16]

   Z následujících úloh se pak jedná o clusterovou analýzu, sumarizaci, hledání asociačních pravidel a objevování sekvencí.

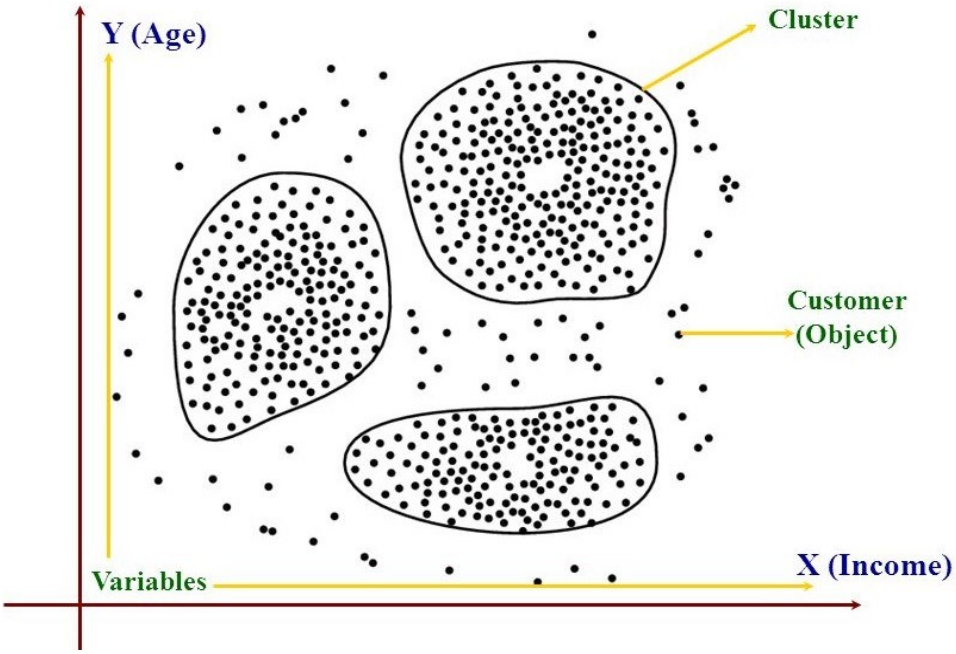
2. **Prediktivní data mining**

   Obecně prediktivní analýza předpovídá nebo odvozuje vlastnosti z dříve dostupných údajů. [16]

   Z následujících úloh se pak jedná o klasifikaci, predikci a analýzu časových řad.

### 3.3.1 Shluková analýza

Někdy též označovaná jako clusterová analýza. Cílem je získat informace o podobnosti jednotlivých vzorků dat. Podobnost se v tomto případě může definovat jako vzdálenost hodnot v rámci několika předem daných dimenzí[5]. Jednotlivé vzorky dat jsou tříděny do tzv. shluků neboli clusterů. A to takovým způsobem, že vzorky ve stejném clusteru by si měly být navzájem nejpodobnější.




Obr. 9 Příklad použití dvoudimenzionální clusterové analýzy na zákazníka. [17]

**Příklad**

E-shop má k dispozici data o produktech, který si daný zákazník zakoupil. Podle těchto dat může díky clusterové analýze nabízet zákazníkovi produkty, které si zakoupili zákazníci ve

[5] Dimenze dat – Počet hodnot (atributů), které obsahuje jeden vzorek řešeného problému.

stejném clusteru. Samotné roztřídění do clusteru pak může algoritmus zajistit např. podle informací, které zákazník vyplnil při registraci. Obdobně jako na obrázku (Obr. 9).

### 3.3.2 Sumarizace

Jeden z klíčových konceptů data miningu. Zahrnuje techniky pro nalezení kompaktních informací o data setu. Cílem je tedy poskytnout shromážděné informace o datech, které mohou být natolik obsáhlé, že se v nich tyto informace nedají běžným způsobem najít. Často se ve shrnutí používají běžné statistické funkce jako je např. průměr, medián, směrodatná odchylka atd.

**Příklad**

Příkladem můžou být výsledky statistických funkcí (medián, průměr či jiné) např. pro počet žáků na jednotlivých středních školách v kraji Vysočina. Díky tomu je možné získat informace o atraktivitě jednotlivých oborů.

### 3.3.3 Asociační pravidla

Pomocí technik data miningu je možné odkrýt asociace a souvislosti mezi jednotlivými vzorky v data setu. Díky tomu se mohou identifikovat vztahy mezi objekty. A tím například lépe interpretovat chování jednotlivce při výběru zboží.

**Příklad**

Mějme e-shop s libovolným zbožím. Jelikož máme data z předchozích objednávek, můžeme určit, které produkty byly často koupeny společně. Díky těmto vztahům budeme zákazníkům nabízet přednostně produkty, které mají souvislosti se zbožím v jejich košíku.

### 3.3.4 Objevování sekvencí

Tato technika je určená k nalezení důležitých, ale mnohdy skrytých vzorů či sekvencí v datech. Pomocí vyhodnocení určitých kritérií je možné v datech najít z části se opakující vzory. Tato úloha je podobná předchozí úloze o hledání asociačních pravidel. Na rozdíl od ní se však více soustředí na opakující se posloupnosti v datasetu.

**Příklad**

Tento typ úlohy je možné uvést podobně jako předchozí. Uvažujme tedy příklad jako v předchozí úloze. Rozdíl bude v tom, že souvislé produkty můžeme analyzovat i z více objednávek, které šly po sobě od stejného uživatele.

### 3.3.5 Klasifikace

Klasifikace hledá hodnotu cílové proměnné podle hodnot ostatních atributů vzorku. Model data miningu zkoumá velkou sadu vzorků, z nichž každý obsahuje informace o cílové proměnné a sadu vstupních nebo prediktivních proměnných [13]. Právě pomocí vstupních a prediktivních proměnných dokáže naučený systém určit hodnotu ještě nespecifikované cílové proměnné. Ta pak může nabývat pouze hodnoty z předem definované třídy hodnot. Tím dochází ke klasifikaci ještě neklasifikovaných vzorků z datasetu.

Pro správné fungování algoritmu je nutné prvně model natrénovat na trénovacím datasetu. Poté se provádí validace oproti validačnímu datasetu. Ta slouží pro úpravu parametrů modelu, aby nedošlo k jeho přeučení. Což by znamenalo, že je model příliš zacílen na trénovací dataset a mohl by poskytovat špatné výsledky v případě jiného než trénovacího datasetu.

**Příklad**

Předpokládejme, že výzkumník by chtěl být schopen klasifikovat příjmové skupiny osob (nízká, střední, vysoká), které nejsou v současné době v databázi, na základě dalších charakteristik spojených s těmito osobami. Jedná se o věk, pohlaví a povolání. Algoritmus nejprve prozkoumá trénovací dataset a tím se „naučí“. Poté se podívá na nové záznamy, pro které nejsou k dispozici žádné informace o příjmové skupině. Na základě klasifikací v trénovací množině dokáže přiřadit klasifikaci novým vzorkům. Ukázku datasetu s právě probíhající klasifikací vzorku č. 43 je možné vidět v tabulce (Tabulka 2). [13]

Tabulka 2 Ukázka probíhající klasifikace datasetu z příkladu.

| Vzorek č. | Věk | Pohlaví | Povolání | Příjmová skupina |
|---|---|---|---|---|
| 1 | 33 | Muž | Softwarový vývojář | Vysoká |
| 2 | 22 | Žena | Kadeřnice | Nízká |
| 3 | 24 | Žena | Marketingový specialista | Střední |
| … | … | … | … | … |
| 43 | 29 | Muž | Marketingový specialista | ? |

### 3.3.6 Regrese

Jedná se úlohu, která je velice podobná klasifikaci. Opět je snahou získat hodnotu cílové proměnné nového vzorku podle existujících vzorků, které již tuto hodnotu mají. Na rozdíl

od klasifikace, hodnoty cílové proměnné nemusí pocházet z předem definované třídy, ale mohou nabývat libovolné číselné hodnoty v daném intervalu hodnot. Existuje mnoho typů regresí. Liší se především ve tvaru křivky hodnot nezávislé cílové proměnné na hodnotách ostatních argumentů všech vzorků.

**Příklad**

Uvažujme systém realitní agentury, který má databázi jednotlivých bytů včetně jejich lokality, velikosti, počtu místností, patra, data rekonstrukce či dokončení a dalších atributů. Díky regresní analýze je možné vytvořit systém, který dokáže odhadnout cenu nově zadávané nemovitosti dle atributů a cen již zadaných bytů.

### 3.3.7 Predikce

Predikce je technika, která se používá pro získání hodnoty cílové proměnné. Na rozdíl od klasifikace a regrese však cílí na předpověď budoucích trendů pomocí aktuálně dostupných dat. Kterákoli z metod a technik používaných pro klasifikaci a regresi lze použít za vhodných okolností také pro predikci [13].

**Příklad**

Sázková kancelář má k dispozici data všech odehraných zápasů fotbalových týmů za několik posledních let. Také sbírá data o měnících se statistikách jednotlivých hráčů, změnách v týmech atd. Díky všem těmto datům, si může strojově pomoci prediktivní analýzou a odhadnout výsledky budoucích zápasů.

### 3.3.8 Analýza časových řad

Časová řada znamená posloupnost datových bodů, které byly zaznamenány v určitých časových úsecích. Následující hodnota časové řady je obvykle určena jednou nebo více předchozími hodnotami. Rozdíl oproti předchozím úlohám je v použití časového otisku v rámci analýzy. Snahou je totiž najít opakující se vzory a trendy v čase. Dalo by se říci, že se jedná o součást prediktivní analýzy. Účelem analýzy je tedy předpovědět budoucí hodnoty časových řad.

**Příklad**

Analýza časových řad se velmi často používá na finančních trzích. Její využití lze uplatnit např. pro predikci hodnot na akciovém trhu nebo i v IoT.

## 3.4 Architektura v softwarových systémech

Kromě základních znalostí v oblasti data miningu je také velice důležité vědět, jaké jsou možnosti integrace funkcionalit z této oblasti do architektur podnikových i jiných softwarů. V případě integrace data miningu do již existujícího softwaru je nutné dobře rozmyslet cíl a přínos pro zákazníky daného softwaru. Podle stupně technické a informační orientace vašich zákazníků se musí určit s jakou abstrakcí a jakým způsobem budou výsledky data miningu využívat. Z druhé strany je složitá i technická integrace na již existující softwarovou architekturu a datové úložiště. V případě, že se jedná o softwarový produkt, který je navrhnut pro možnou integraci data miningu je situace jednodušší.

### 3.4.1 Rozložení vrstev data mining nástrojů

Data mining nástroje obvykle obsahují více možností, jak s nimi komunikovat. Standardně je možné takové nástroje rozdělit na 4 vrstvy. [18]

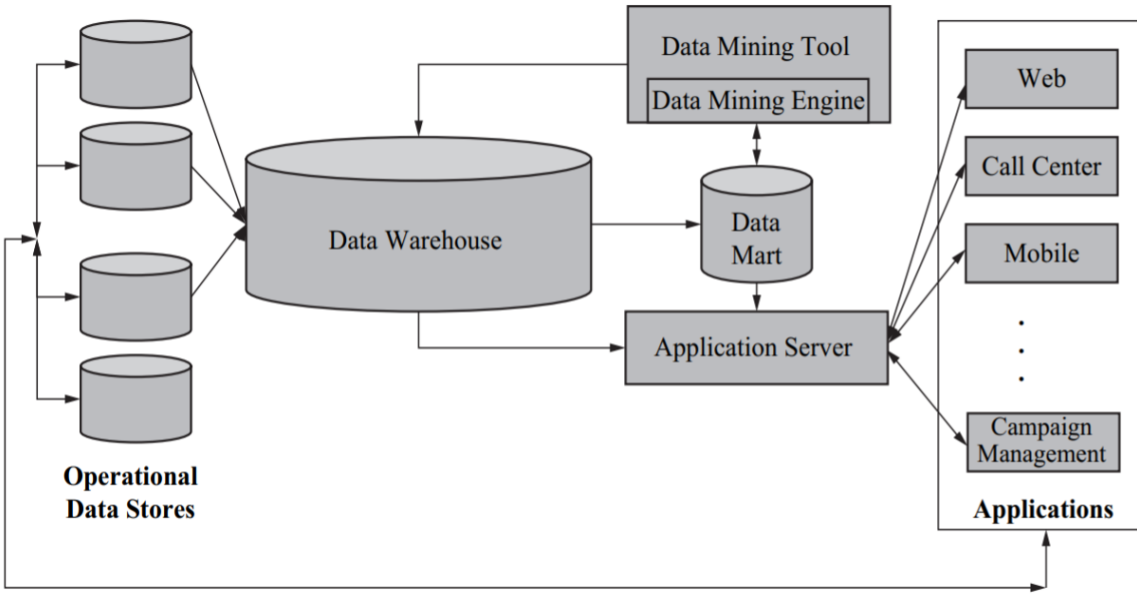
1. Grafické uživatelské rozhraní (GUI)
    - Pro práci s daty, pro interaktivní data mining.
    - Poskytuje přímý přístup uživatele k data mining enginu.
2. Aplikační rozhraní (API)
    - Pro tvorbu aplikací, které integrují data mining techniky.
    - Často ve formě knihovny v programovacích jazycích (Java, C#, Python, …) nebo jako webové (např. REST) API.
3. Nástroje pro dolování dat (DME),
    - Zde probíhá aplikace data mining algoritmů na vstupní data.
4. Úložiště těžebních objektů (MOR)
    - Poskytuje perzistenci pro objekty, které je třeba trvale ukládat pro zajištění správného běh algoritmů.
    - Může jim být databáze, datový sklad i file systém.

### 3.4.2 Umístění jádra data miningu

Jádro data mining systému tvoří engine, který obsahuje algoritmy pro zpracování dat. Zásadní otázkou pak je, kam tento engine umístit z pohledu softwarové architektury. V zásadě se nabízí 2 možnosti.

- Součást nástroje pro data mining
  - Přesouvají se data směrem k algoritmům.
- Součást databáze či datového skladu
  - Přesouvají se algoritmy směrem k datům.

Příklad napojení data miningu do softwarové architektury je možné vidět na (Obr. 10). Jedná se o softwarové řešení používající datový sklad a datové tržiště pro perzistenci objektů souvisejících s činností algoritmů data miningu. Engine zde vystupuje jako součást data mining nástroje. Aplikační server zpracovává informace, které následně předává klientům a ty je poté vhodně interpretují. [18]




Obr. 10 Příklad data mining architektury s odděleným data mining nástrojem. [18]

### 3.4.3 Typy data mining architektury

Architektura data mining systému se standardně dělí do čtyř typů podle stupně využívání systému určeného k perzistenci dat. Tímto systémem se obvykle myslí databáze či datový sklad.

**No Coupling (bez spojení)**

Jedná se o aplikaci data miningu, která nevyužívá výhody ukládání do databáze či datového skladu. V podstatě pouze načítá požadovaná data z různých datových zdrojů a nad nimi poté provádí jednoduché procesy pro získávání informací bez nutnosti cokoli perzistovat.

**Loose Coupling (volné spojení)**

V tomto typu architektury se využívá databáze či datový sklad k vyhledávání dat pro data mining algoritmy. Již zpracované informace bývají ukládány zpět do těchto systémů. Mezivýsledky data mining algoritmů jsou však uloženy v paměti aplikace. Architektura se používá pro systémy, které nevyžadují vysokou škálovatelnost a velký výkon.

**Semi-tight Coupling (polotěsné spojení)**

Snaží se využívat funkce databází či datových skladů k optimalizaci průběhu data miningu. Jedná se o operace jako je např. indexace, agregace, třídění a další. Někdy se používá i ukládání mezivýsledků do databází či datových skladů, pokud je to z hlediska optimalizace výhodné.

**Tight Coupling (těsné spojení)**

Poslední z typů maximálně využívá databázi či datový sklad k optimalizaci procesu. Taktéž se snaží co nejvíce využívat funkce a procedury, které systém pro ukládání dat nabízí. Tato architektura je velice výhodná především svojí škálovatelností a vysokým výkonem.

## 3.5 Data mining a strojové učení

Mezi pojmy „data mining“ a „strojové učení“ často panují nejasnosti v jejich rozdílech i společných vlastnostech. Mnohdy bývají zaměňovány ze strany běžných médií prostřednictvím neodborných článků. Nicméně i v odbornějších publikacích občas k záměně dochází. Hranice mezi oběma pojmy je totiž díky řadě společných vlastností poněkud nepřesná.

### 3.5.1 Společné rysy

Strojové učení stejně jako data mining spadá do oblasti s názvem **data science**, která se stala v poslední době velkým fenoménem a dostává se i do povědomí obyčejných lidí. Oba procesy také spadají do novinářsky oblíbeného termínu „umělá inteligence“. To bude nejspíše i jeden z důvodů, proč tak často dochází k zmíněné záměně pojmů. Důležité je také zmínit, že základem obou těchto procesů není nic jiného než statistika.

Data mining často používá algoritmy z oblasti strojového učení pro účinnější získávání informací a znalostí. Z druhé strany mohou „vydolovaná“ data posloužit jako počáteční zdroj informací pro strojové učení. Oba procesy tedy občasně vzájemně profitují jeden z druhého.

### 3.5.2 Rozdíly

I když mají oba procesy mnoho společného jsou mezi nimi i nezanedbatelné rozdíly. Vzhledem k faktu, že se tyto dva procesy v praxi velice často používají společně a doplňují se v konkrétních případech, je nutné následující rozdíly brát lehce s rezervou. Rozdíly se mohou týkat těchto vlastností:

**Účel**

Nejdůležitějším rozdílem je bezesporu účel obou procesů. Data mining je metoda zkoumání, jejímž cílem je získat informace a znalosti na základě souhrnu shromážděných dat. Zatímco strojové učení trénuje systém k provádění složitých úkolů a využívá nasbíraná data a zkušenosti k tomu, aby se stal chytřejším. [19]

**Použití dat**

Pro data mining jsou už podle názvu nejdůležitější data. Na nich totiž stojí veškeré procesy a algoritmy. A to nejlépe velké množství dat (big data). Strojové učení je naproti tomu postavené především na algoritmech, které sice s daty také pracují, ale nejsou pro ně natolik klíčové.

**Zásah člověka**

Algoritmy strojového učení se snaží získané informace autonomně přetavit na znalosti a pomocí nich vylepšovat algoritmy. Zásah člověka je nutný především při počátečním nastavení parametrů algoritmů a při jejich trénování. Zatímco data mining algoritmy se snaží získat informace a předat je lidem. Ty pak mohou na základě znalostí z těchto informací činit informovaná rozhodnutí.

**Adaptivita**

Data mining se řídí předem nastavenými pravidly a je defacto statický, zatímco strojové učení upravuje algoritmy podle toho, jak se projeví vhodné okolnosti.

# 4 PREPROCESSING DAT

Uložená data, ze kterých je v plánu získávat informace často disponují problémy, které by mohly z hlediska získávání informací způsobit nepříjemnosti. Data mohou být neúplná, nekonzistentní, nepřesná nebo třeba zašuměná. Úkolem preprocessingu (předzpracování) dat je odstranit tyto problémy, aby dataset byl konzistentní, čistý a korektní vzhledem k algoritmům.

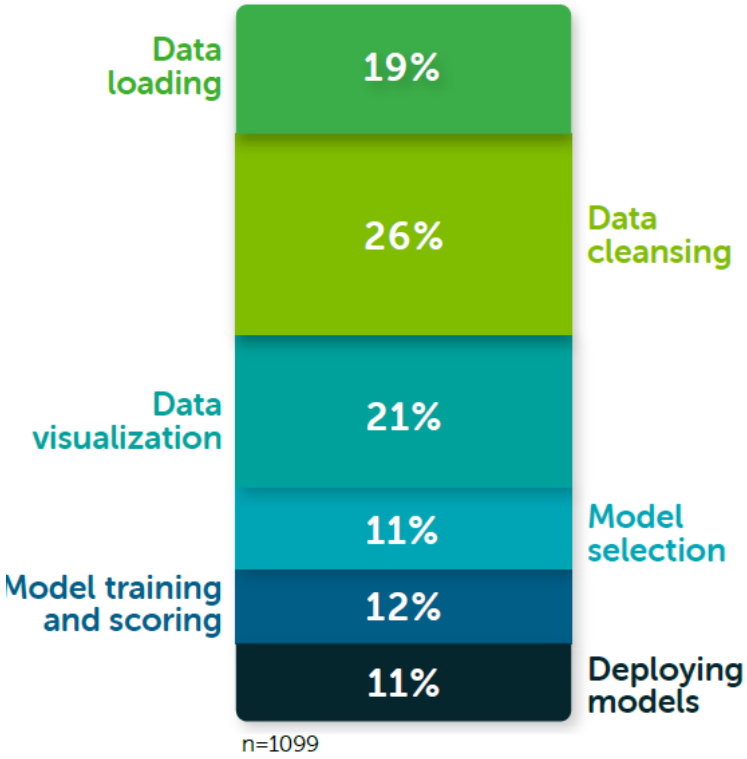
Jedním z důležitých úkolů **fáze porozumění datům**, kterou definuje CRISP-DM, je analýza kvality a čistoty dat. Právě kvalita a čistota jsou vlastnosti, které je žádoucí u dat vstupujících do algoritmů maximalizovat. Ovlivňuje totiž výsledné získané informace.

Ve **fázi přípravy dat** pak probíhá samotný preprocessing. Tento krok je velice důležitý pro další počínání. Správnou přípravou dat je totiž možné nejen předejít chybným informacím, které se z dat mohou získat, ale také mnohdy výrazným způsobem optimalizovat běh algoritmů data miningu. Je žádoucí, aby předzpracovaná data byla [20]:

- Přesná – data neobsahují nepřesnosti
- Úplná – data obsahují všechny klíčové atributy, které by obsahovat měla
- Konzistentní – data jsou ve správném formátu
- Aktuální – data nejsou zastaralá v případě, že je to nežádoucí
- Interpretovatelná – data jsou srozumitelná vzhledem ke strojovému zpracování
- Uvěřitelná – datům je možné věřit

Preprocessing je tedy klíčovou operací v procesu data miningu. Je dokonce natolik klíčový, že může být časově náročnější úlohou než samotná fáze provedení data miningu. Podle průzkumu mezi datovými vědci, který provedla společnost Anaconda[6], stráví datoví vědci přibližně 45 % svého času přípravou dat, včetně jejich načítání a čištění [21]. Srovnání časové náročnosti úloh, se kterými se musí datový vědec vypořádat je vizualizováno na obrázku (Obr. 11).

[6] https://www.anaconda.com/state-of-data-science-2020?utm_medium=press&utm_source=anaconda&utm_campaign=sods-2020&utm_content=report




Obr. 11 Srovnání vynaloženého času jednotlivých úkonů datových vědců. [21]

Postup preprocessingu se dá rozdělit na několik po sobě jdoucích úloh. V různých literaturách se mohou jednotlivé úlohy lišit. Nicméně klíčové operace by měly být vždy podobné. Je třeba brát v uvážení, že data, která se budou jakýmkoli způsobem zpracovávat či předzpracovávat se mohou lišit dataset od datasetu. Základní rozdělení procesu počítá s těmito třemi kroky:

1. Čištění dat
2. Redukce dat
3. Transformace dat

## 4.1 Čištění dat

Data, která neprojdou čištěním mohou způsobit nepřesnosti v získaných informacích. Z tohoto důvodu je úloha čištění dat v rámci přípravy dat důležitá. Je třeba odstranit či opravit chybějící záznamy, chybné záznamy či zašuměné hodnoty.

### 4.1.1 Odstranění šumu

Šumem je možné nazývat náhodnou chybu nebo rozptyl v měřené veličině [22]. Rozhodně, však není žádoucí odstranit extrémy v případě, že právě tyto vzorky budou klíčové z pohledu další analýzy např. detekce anomálií. K tomu abychom bylo možné šum odstranit je možné použít některé z následujících technik.

**Binning**

Hodnoty určité dimenze se seřadí posloupně za sebe. Následně se vytvoří skupiny o velikosti $k$. V rámci těchto skupin se provede tzv. **vyhlazení**. To znamená, že všechny hodnoty v dané skupině se nahradí výsledkem statistické funkce provedené nad těmito hodnotami. Běžně se k tomu používá průměr nebo medián, ale využívají se i minima a maxima.

**Regrese**

Vyhlazení dat lze provést také pomocí regrese, což je technika, která přizpůsobuje hodnoty dat funkci [22]. Regresi se věnuje sekce 3.3.6.

**Analýza krajních hodnot**

Detekci okrajových hodnot je možné provést pomocí aplikace clusterové analýzy [22]. Ta je specifikována v sekci 3.3.1.

### 4.1.2 Opravení chybějících hodnot

Analyzovaná data mohou obsahovat vzorky, které neobsahují hodnoty, jež jsou z pohledu dané analýzy nezbytné. Existuje více různých principů, které definují, jak se k takovému vzorku postavit a co s ním je možné dělat.

**Přeskočení vzorku**

Neúplný vzorek je možné ignorovat a zcela ho tak vynechat. Nicméně dojde tím ke ztrátě dat, která by mohla v určitých případech změnit výsledek. Zvláště pak pokud neúplné vzorky tvoří většinu celého datasetu.

**Doplnění chybějících hodnot**

Druhým často používaným principem je doplnění chybějících hodnot. V tomto případě se nabízí spousta možností, jak je možné chybějící hodnoty doplnit. Používá se doplnění průměrem, mediánem, předem danou konstantou. Existují i složitější a sofistikovanější postupy pro odvození a doplnění chybějící hodnoty jako je např. nejpravděpodobnější hodnotou,

která je odvozená pomocí algoritmu rozhodovacích stromů [22]. V případě, že dataset není příliš obsáhlý, je možné doplnit chybějící hodnoty manuálně.

## 4.2 Redukce dat

Pro velmi obsáhlé datasety by mohla být jejich analýza příliš pomalá. V takovém případě je třeba se zabývat otázkou, jak dataset zmenšit, ale zároveň nepřijít o potenciální znalosti, které je možné z něj získat. Je nutné použít operace, které dokáží zmenšit objem, přesto zachovávají integritu originálního datasetu. Redukce jako taková se může rozdělit na redukci dimenze a transformaci na alternativní dataset.

### 4.2.1 Redukce dimenze

Proces, jehož snahou je zmenšit objem dat pomocí snížení dimenze. Složitost algoritmů totiž může narůstat v některých případech až kvadraticky. Díky snížené dimenzionalitě je také mnohem jednodušší data vizualizovat. [22]

**Vlnková transformace**

Jedna z metod používaných ke snížení dimenze. Princip spočívá v transformaci vektoru $X$ na jiný vektor $X'$, který je složen z vlnkových koeficientů, ale je stejně velký. Tento vektor může v pojetí data miningu reprezentovat jeden vzorek z datasetu. Přestože je výsledný vektor stejně velký ukládá se pouze jeho zkrácená varianta. Pokud se totiž uloží jen část nejsilnějších vlnkových koeficientů, lze z nich poté získat aproximaci původních dat. Tím se dosáhne tíženého snížení dimenze.

**Karhunen-Loeve analýza**

Hledá $k$ $n$-rozměrných ortogonálních vektorů, které lze nejlépe použít k reprezentaci dat, kde $k < n$. Původní data jsou promítnuta do menšího prostoru, což má za důsledek snížení dimenze. Často odhalí vztahy, o kterých se dříve netušilo, a tím i umožňuje interpretaci, která by za normálních okolností nebyla možná. [22]

**Výběr podmnožiny atributů**

Pokud dataset obsahuje atributy, které jsou z pohledu data miningu irelevantní je možné je vynechat. V případě, že disponujeme znalostmi v konkrétní doméně, je možné tuto selekci provést ručně. Selekci je však možné provést také automaticky pomocí optimalizačních algoritmů.

### 4.2.2 Převod na alternativní reprezentaci

Jedná se formu redukce dat, která využívá převodu z původní reprezentace na úspornější alternativní reprezentaci.

**Parametrická redukce dat**

K aproximaci dat lze použít regresní a log-lineární modely. V lineární regresi se data modelují tak, aby odpovídala přímce. Mějme příklad, kde proměnná $y$ je modelována jako lineární funkce jiné náhodné proměnné $x$ (proměnná predikce). Vztah mezi těmito proměnnými je popsán rovnicí (3). [22]

$$y = wx + b \tag{3}$$

Proměnné $w$ a $b$ jsou koeficienty regrese. Vícenásobná lineární regrese umožňuje modelovat atribut $y$ jako lineární funkcí více prediktivních proměnných. Tím se docílí snížení objemu dat. [22]

**Histogram**

Využitím histogramů je možné také značně snížit objem dat. Histogramy reprezentují četnost jednotlivých hodnot nebo předem daných rozsahů. Důležité je jedním histogramem vždy reprezentovat zastoupení hodnot pouze v rámci jednoho atributu.

Jelikož jsou histogramy efektivní, jak pro řídká, tak pro hustá data, jsou velice oblíbenou technikou pro redukci dat.

## 4.3 Transformace dat

Aby algoritmy pro data mining mohly být generické a přesto efektivní, definují formát a rozsah vstupních dat. Úkolem této části preprocesingu je transformovat data vstupující do algoritmu tak, aby splňovala algoritmem daná kritéria pro vstupní data. Transformace může být relativně jednoduchá ale i obtížná. To záleží na podobě syrových dat vzhledem k požadavkům na formát datasetu, který vstupuje do algoritmu.

### 4.3.1 Konstrukce atributů

Transformace je založena na myšlence, že existuje možnost, jak odvodit atributy pro které neexistují žádné hodnoty z již jiných atributů pro které hodnoty existují. Nově vytvořené atributy by měly mít prediktivní schopnosti pro data mining, v opačném případě ztrácí svůj význam. Může se jednat o poměrně jednoduché odvození (např. přepočítání dne v měsíci na

den v týdnu) i mnohem složitější operace. S problémem, jak zkonstruovat nejvhodnější atributy mohou pomoci některé algoritmy.

**Konstrukce atributů pomocí algoritmu greedy search v uzlech rozhodovacího stromu**

Greedy search je jeden z algoritmů, který lze využít pro konstrukci atributů. Při indukci rozhodovacího stromu algoritmus generuje v každém uzlu jeden nový atribut na základě původních i již vygenerovaných atributů a zvolí ten nejlépe ohodnocený. Proces generování nového atributu zahrnuje použití algoritmu greedy search pro vyhledání v prostoru, který je definován množinou použitelných konstrukčních operátorů (např. dělení, konkatenace). Musí být vhodně nastavena vyhodnocovací funkce. Tímto způsobem je možné získat atribut, který by měl být nejlépe využitelný při data minigu. [23]

**Konstrukce atributů pomocí genetických algoritmů**

Genetické algoritmy využívají operace mutace a křížení pro nalezení nejoptimálnějšího řešení. Jednotlivci jsou hodnoceny tzv. fitness funkcí, která udává kvalitu řešení dané tímto jedincem. Jedinec v tomto případě reprezentuje seznam originálních a nově vykonstruovaných atributů. Fitness funkce udává jejich využitelnost. Genetický algoritmus dokáže najít nejoptimálnější kombinaci těchto atributů. Kvalita výsledného řešení závisí na správném nastavení fitness funkce. [23]

### 4.3.2 Normalizace

Úkolem normalizace je zajistit jednotlivým atributům stejnou váhu a stejné měřítko [22]. Mějme příklad, internetového obchodu, který eviduje u položek zboží jejich cenu. Pokud by se jednalo o dražší zboží (např. v řádu stotisíců), tak by váha hodnot atributu "cena" převážila hodnoty jiného atributu v rozsahu jednotek (např. počet kusů skladem). Je tedy nutné oba tyto atributy převést na zhruba stejné měřítko.

Existuje mnoho různých metod, jak provést normalizaci. Mezi nejpoužívanější se řadí min-max normalizace a z-score normalizace.

**Min-max normalizace**

Pro ukázku principu normalizace bude použita metoda min-max. Jedná se totiž asi o nejjednodušší, přesto velice účinnou a používanou metodu.

Min-max normalizace zachovává lineární kombinaci a je definována vztahem popsaným v rovnici č. (4). [22]

$$v'_i = \frac{v_i - \min(A)}{\max(A) - \min(A)} (new\_max_A - new\_min_A) + new\_min_A \qquad (4)$$

Kde $v_i$, je aktuální transformovaná hodnota, $\min(A)$ a $\max(A)$ jsou minimální a maximální hodnoty atributu $A$, $new\_min_A$ a $new\_max_A$ jsou minima a maxima v novém rozsahu.

### 4.3.3 Diskretizace

Úloha má za cíl převést numerické hodnoty na intervaly. Těmto intervalům je poté možné udělit „nálepky“. Příkladem může být, že namísto konkrétní výšky člověka se definují intervaly podle jeho výšky.

- malý – méně než 160 cm
- nižší střední – 160 až 175 cm
- vyšší střední – 175 až 185 cm
- vysoký – více než 185 cm

Samozřejmě je možné tak učinit pouze v případě, že není třeba v rámci data miningu pracovat s konkrétními numerickými hodnotami. Diskretizace také může snižovat objem dat. Může být provedena pomocí různých technik z nichž některé jsou popsány níže.

**Diskretizace pomocí binningu**

Techniku tzv. binningu, který je již popsán v sekci 4.1.1, je možné použít i pro diskretizaci. Princip spočívá v nahrazení původní hodnoty intervalem, jehož rozsah je možné definovat ručně či tak, aby intervaly odpovídaly rovnoměrnému rozložení šířky intervalu nebo rovnoměrnému rozložení četnosti intervalu.

**Diskretizace pomocí analýzy histogramů**

Princip histogramů byl již opět popsán v sekci 4.2.2. Pro diskretizaci je postup velice podobný jako v případě binningu, jelikož i histogramy pracují s objekty, které se někdy nazývají jako tzv. bins. Princip histogramů tak vychází z technik binningu.

**Diskretizace pomocí clusterové analýzy**

Jedná se o opět velice známou techniku v oblasti zpracování dat, která je taktéž již popsána v sekci 3.3.1. Pomocí rozdělení hodnot atributu do shluků je možné získat velice kvalitní výsledky diskretizace, jelikož clusterová analýza bere na vědomí, jak rozdělení hodnot atributu, tak i vzdálenost jednotlivých datových bodů. [22]

# II. PRAKTICKÁ ČÁST

# 5 AKTUÁLNÍ STRUKTURA DAT V IOT PLATFORMĚ

Tato kapitola se věnuje charakteristice a popisu dat získaných z IoT zařízení. Data jsou uložena a zpracovávána integrační platformou pro IoT od společnosti MyMight a později budou sloužit pro testování algoritmů z oblasti data miningu. Nejprve je, ale nutné charakterizovat jejich aktuální strukturu.

Společnost MyMight s.r.o. souhlasila se zveřejněním aktuální struktury ukládaných hodnot z IoT zařízení. Konkrétní data poskytnutá k ladění a testování pochází ze vzorového rodinného domu.

## 5.1 IoT

Tento pojem se poslední dobou objevuje i v běžných mediích a začíná tak prosakovat do retailové společnosti. Tím se samozřejmě i zvýšil zájem o různé druhy zařízení, které spadají do této kategorie.

### 5.1.1 Definice

Organizace IEEE definuje pojem „IoT“ následujících způsobem: Obecně řečeno, **internet věcí – Internet of Things** je systém sestávající ze sítí senzorů, akčních členů a inteligentních objektů, jehož účelem je propojit "všechny" tyto věci, včetně každodenních a průmyslových předmětů, takovým způsobem, aby byly inteligentní, programovatelné a schopné lépe komunikovat s lidmi i mezi sebou navzájem. [24]

Pod pojmem IoT zařízení se může skrývat defacto jakékoli zařízení připojitelné do sítě dalších těchto zařízení. Teoreticky do této definice spadá i mobilní telefon s připojením k internetu nebo hodinky, které umožňují komunikaci pomocí Bluetooth protokolu s mobilním telefonem.

Svoje místo si IoT zařízení našla v automatizaci rezidenčních, komerčních i průmyslových budov. Také jsou součástí konceptů tzv. smart cities. Což jsou města, která se pomocí moderních technologií snaží vytvořit lepší podmínky pro život svým obyvatelům. Snaha o implementaci IoT se v poslední době rozšířila do všech možných odvětví jako např. zemědělství, doprava, průmysl, logistika atd.

### 5.1.2 IoT a big data

Každé IoT zařízení může generovat nebo přijímat jednotky až tisíce zpráv každý den. Je jasné, že v takové IoT síti dochází k velkému přenosu dat mezi jednotlivými zařízeními. Lze tedy prohlásit, že data, které se pohybují ve větších IoT sítích, dosahují dostatečného objemu (volume), narůstají dostatečně rychle (velocity) a jsou dostatečně různorodá (variety) na to, aby se mohly považovat za **big data** podle definice vlastností 3 V, které jsou popsány v sekci 1.1.

Různorodost těchto dat je dána především rozmanitostí používaných datových typů v IoT zařízeních. Komunikace mezi zařízeními probíhá digitálně. Přenášená data jsou ve formě binárních vektorů. Ty však mohou reprezentovat v podstatě jakýkoli základní datový typ používaný v moderních programovacích jazycích. Jsou to binární hodnoty, celá čísla, desetinná čísla či text.

Zaznamenaná historická komunikace senzorů a akčních členů může být přínosná, pokud je třeba zpětně zjistit stav konkrétního zařízení v konkrétním čase. Což je pro uživatele bezesporu přidaná hodnota. Softwarová nadstavba může, ale tyto data více vytěžit a získat z nich cenné informace a znalosti, které poté mohou být uživateli vhodně interpretovány.

## 5.2 Popis platformy MyMight

Platforma od společnosti MyMight aktuálně nabízí možnost integrace několika různých IoT technologií a zařízení od více výrobců do jednoho softwarového řešení. Toto řešení poskytuje svým zákazníkům jednouché a ucelené uživatelské rozhraní pro ovládání konkrétních zařízení. Vzdálené ovládání těchto technologií je možné díky mobilní aplikaci Myjordomus[7], která poskytuje přehlednou strukturu ovládacích prvků s možností customizace. Díky této aplikaci může zákazník přistupovat k zařízením odkudkoli, kde má k dispozici připojení k internetu.

Platforma je vhodná nejen pro automatizaci rezidenčních domů či bytů, ale její uplatnění je možné nalézt i v průmyslových či komerčních prostorách.

[7] Popis aplikace včetně ukázek je dostupný na adrese: https://www.mymight.com/myjordomus/.

### 5.2.1 Možnosti platformy

V kontextu platformy se velice často využívá pojem **prvek** (element). Takovým prvkem může být senzor nebo akční člen (aktor) libovolného podporovaného IoT zařízení. Platforma nad konkrétními zařízeními vytváří abstrakci právě pomocí těchto prvků. Přičemž dané zařízení může v platformě vystupovat jako jeden nebo více těchto prvků. U každého prvku se definuje datový typ a možnost čtení či zápisu.

Kromě vzdáleného ovládání pomocí mobilní aplikace platforma nabízí další užitečné možnosti využití IoT technologií. Jedná se např. o:

**Tvorbu automatických scénářů**

Uživatel může díky scénářům ovládat více prvků najednou, a to samozřejmě i napříč různými technologiemi, které platforma umí integrovat.

**Definice automatických akcí**

Konfigurovatelné prostředí pro tvorbu automatických akcí. Uživatel si může definovat autonomní chování. Např.

- Odložené vykonání akce (např. nastavení hodnoty na určitý aktor) v požadovaný čas.
- Reakce na změnu stavu některého prvku (např. odesláním notifikace do mobilní aplikace).

**Vizualizace do grafů**

Nasbíraná data ze senzorů umí platforma interpretovat svým zákazníkům pomocí webové aplikace, která umožňuje tvorbu nástěnek pro vizualizaci. Uživatel této aplikace si tak může vytvořit vlastní nástěnku a plně si customizovat vizualizaci historických i aktuálních hodnot do přehledných grafů či jiných widgetů. Prozatím se jedná pouze o prezentační nástěnky bez možnosti ovládání. Nástěnky mohou být využity např. pro monitorování spotřeby energií.

**Softwarové zázemí pro implementátory IoT technologií**

Platforma poskytuje software nejen pro koncové uživatele, ale také pro proškolené implementátory IoT technologií. Pomocí tohoto softwaru implementátoři mohou jednoduše importovat IoT zařízení do platformy a provést konfiguraci projektu zákazníkovi.

### 5.2.2 Integrované technologie

Nejdůležitější předností této platformy je bezesporu nezávislost na jedné konkrétní technologii. Samozřejmě není možné, aby platforma uměla integrovat všechny IoT technologie, co existují. Proto se snaží integrovat co nejlépe konkrétní technologie, které se vyznačují profesionálnějším přístupem. Portfolio technologií není konečné a v případě požadavků se jistě bude rozrůstat.

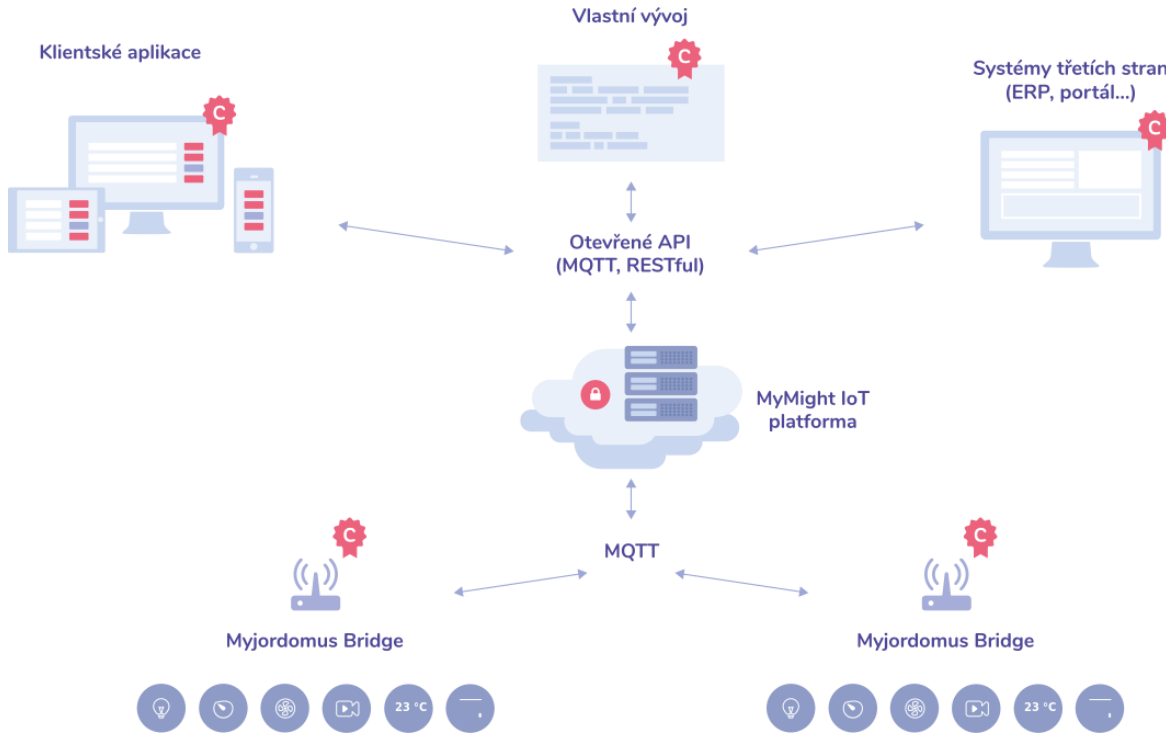
- **KNX**
  - Mezinárodní otevřený standard pro automatizaci komerčních i rezidenčních budov. Jednotlivá zařízení jsou speciálními kabely zapojeny do sběrnicové topologie.
  - Portfolio zařízení kompatibilních s KNX sběrnicí čítá několik výrobců. Zákazník si tak může vybrat z poměrně velkého množství produktů.
- **HDL Buspro**
  - Opět se jedná o sběrnicový systém. V porovnáním s KNX mají některá zařízení na této sběrnici omezenější funkcionalitu, ale jsou cenově dostupnější.
- **Modbus TCP**
  - Otevřený protokol založený na principu master-slave. Definuje strukturu zpráv na úrovni protokolu. Je nezávislý na komunikačním médiu. Používá se především v průmyslu, ale začíná se používat i v oblasti domácí automatizace. Především pro připojení teplených čerpadel a rekuperačních jednotek. [25]
- **Nuki**
  - Společnost Nuki vyvíjí systémy určené převážně pro inteligentní zamykání a odemykání dveří. Jejich zámky najdou využití v domácnostech, hotelech a dalších komerčních objektech. [25]

### 5.2.3 Technická specifikace platformy

Kvůli podpoře vzdáleného přístupu backend platformy běží v cloudovém prostředí. Komunikace s ním probíhá pomocí REST API nebo prostřednictvím MQTT (Message Queuing Telemetry Transport) brokeru. Část REST API je veřejná a umožňuje integraci aplikací třetích stran. Protože platforma klade velký důraz na bezpečnost je veškerá komunikace šifrovaná a zabezpečená pomocí klientských certifikátů nebo bezpečnostního protokolu OAuth2. Obecnou architekturu platformy přehledně vizualizuje obrázek (Obr. 12).

Backend platformy je tvořen několika mikroslužbami[8], které pro komunikaci s klienty a mezi sebou využívají právě zmíněné REST API nebo protokol MQTT. Každá mikroslužba, jak již plyne z její definice, má svoji vlastní databázi, kterou využívá pro perzistenci dat.

Nedílnou součástí platformy je i vlastní hardwarové zařízení nazývané **Bridge**. Integruje komunikaci z různých technologií přímo na místě a v obecném formátu ji přeposílá do cloudu. Připojením přímo na bránu konkrétní technologie může rychle obsluhovat a generovat požadavky.




Obr. 12 Abstraktní pohled na architekturu platformy MyMight. [18]

## 5.3 Struktura ukládaných hodnot

Obsahem této práce je především práce s daty. Předmětem zájmu jsou historické hodnoty ze senzorů a akčních členů jednotlivých IoT prvků. Jak již bylo popsáno v sekci 5.1.2, v IoT sítích se generuje mnohdy obrovské množství dat. Tyto data mohou nést cenné informace, tudíž může být výhodné je ukládat. Z pohledu IoT platformy, si mohou, díky perzistenci

[8] Mikroslužba – Webová služba věnující se konkrétní doméně v systému. S okolím komunikuje pomocí technologicky nezávislých protokolů např. HTTP.

historických hodnot jednotlivých prvků, zákazníci kdykoli zobrazit kompletní historii sítě a zpětně tak zjistit změny hodnot prvků v určitém čase.

### 5.3.1 Původ dat

Množství různých IoT zařízení a způsobů jejich využití je nepřeberně mnoho. Jen a pouze v domácnosti se nejčastěji IoT využívá pro:

- **Zabezpečení** – zámky a zabezpečovací systémy
- **Vytápění, větrání a klimatizaci (HVAC) –** řízení kotle, termostatických hlavic, rekuperace, teplených čerpadel a klimatizací
- **Stínění** – žaluzie, rolety, závěsy a markýzy
- **Osvětlení** – vypnutí/zapnutí, stmívání, nastavení barvy a chromatičnosti
- **Zavlažování** – automatické zapnutí/vypnutí v určitý čas
- **Audio a video systémy**
- **Meteostanice** – spouštění akcí v závislosti na změně počasí
- A další případy, kde je možné ovládat zařízení zapsáním určité hodnoty.

### 5.3.2 Komunikace v IoT technologiích

Komunikace mezi jednotlivými zařízeními v rámci IoT sítě probíhá většinou binárně. Znalost datového typu je však nutná informace, která udává formát binárního vektoru. Tento formát definuje komunikační rozhraní konkrétního zařízení, a to jak pro příjem, tak i odesílání zpráv.

V případě sběrnicových systémů jako je HDL nebo KNX je standardem, že jednotlivá zařízení komunikují spolu po datové sběrnici, ale pokud chceme přistoupit ke sběrnici z venku je nutné provoz směrovat přes tzv. bránu (gateway), která zpracovává požadavky na sběrnici a naopak. V případě bezdrátových technologií je opět nutná brána k převodu zpráv z kabelového řešení do bezdrátové podoby. Lze tedy prohlásit, že brána definuje protokol dané IoT technologie.

V platformě MyMight, aby bylo možné data zpracovávat a směrovat do cloudové platformy je na bránu připojen HW prvek označovaný jako Bridge. Komunikační protokol mezi zařízením Bridge a bránou konkrétní technologie určuje rozhraní brány. Obvykle se veškeré hodnoty přenáší binárně. Nicméně Bridge tyto binární vektory převádí do běžných datových typů. V rámci platformy tak vystupují hodnoty IoT prvků již přetypované do standardních i

konkrétnějších datových typů. Konverze binárních vektorů na konkrétní datové typy a opačně je vždy specifická pro danou technologii a vychází z její dokumentace.

### 5.3.3 Typy hodnot v IoT platformě

Je důležité si uvědomit, že vzhledem k rozmanitosti zařízení platforma pracuje s širokou škálou různých typů hodnot. Některé jsou obecnější jiné jsou specifické přímo pro konkrétní případy užití. Typ je pak v platformě vždy svázán s konkrétním prvkem a definuje tím povolené hodnoty, kterých může daný prvek nabývat. Také definuje defaultní vizualizaci pro mobilní aplikace. Platforma určuje u každého typu tyto vlastnosti:

- Základní datový typ daný programovacím jazykem Java
    - Omezení plynoucí z použití daného datového typu.
    - Např. Boolean, Integer, BigDecimal, Long, String, Calendar
- Minimální povolená hodnota
- Maximální povolená hodnota
- Hodnota kroku
    - Některé zařízení umí zpracovávat pouze hodnoty posloupnosti, která je definovaná minimem, maximem a krokem.
    - Např. Stmívatelné osvětlení je možné nastavit na hodnoty 0–100 % s krokem po 10 %.
- Povolené prefixy jednotek
    - Určují přípustné řády dané jednotkou tohoto typu.
    - Např. kilo, mega, giga
- Sufix hodnoty
    - Zkratka fyzikální jednotky.

V určitých případech se používají obecnější typy hodnot, které nemusí mít nutně určené všechny tyto vlastnosti. Z druhé strany jsou i případy, kdy je lepší pracovat s konkrétnějšími typy, které určují i prefixy jednotek či sufix hodnoty. Platforma je připravena na oba tyto scénáře a disponuje tak škálou různých typů hodnot od obecnějších ke konkrétnějším. Struktura typů je realizována hierarchicky. Prvotní výběr konkrétního typu pro daný prvek určuje technologie a její zařízení. Po načtení prvků z technologie je možné tento typ změnit.

Platforma disponuje více než 60 různými typy. Přepis definice každého z nich do této práce je však mimo rozsah a zadání této práce. Je, ale nutné nahlédnout na základní typy, které

určují důležitá omezení i z pohledu pozdějšího zpracování dat. Všechny základní typy včetně vybraných konkrétních typů jsou přehledně zobrazeny v tabulce (Tabulka 3).

Tabulka 3 Přehled základních typů hodnot v IoT Platformě.

| Název | Základní typ | Min | Max | Krok | Prefix | Sufix |
|---|---|---|---|---|---|---|
| Big decimal | - | $-2^{63}$ | $2^{63}$ | - | - | - |
| Boolean | - | false | true | - | - | - |
| Integer | - | $-2^{31}$ | $2^{31} - 1$ | 1 | - | - |
| Long | - | $-2^{63}$ | $2^{63} - 1$ | 1 | - | - |
| String | - | - | - | - | - | - |
| Calendar | - | 21.09.1677 | 11.04.2262 | - | - | - |
| Percentage | Integer | 0 | 100 | 1 | - | % |
| Light dim | Integer | 0 | 100 | 10 | - | % |
| Electric voltage | Big decimal | 0 | 1 000 000 | - | m, k | V |
| Light cct | Integer | 1 700 | 27 000 | 1 | - | K |

## 5.4 Perzistence dat z IoT prvků

Všechny hodnoty, které IoT platforma zpracovává jsou ukládány do databáze pro jejich pozdější použití. Jelikož je počet zpráv, které denně platformou proteče vysoký je nutné, aby struktura jejich uchovávání byla co nejoptimalizovanější. Momentální situace je taková, že se historické hodnoty ukládají do klasické relační databáze **MySQL**, která je součástí mikroslužby, jež se stará o přeposílání hodnot a další s tím spojené věci. Historické hodnoty jsou uchovány v samostatné tabulce, kam se zapisují po potvrzení jejich nastavení nebo změření. Samotné hodnoty jsou uchovávány jako datový typ **varchar** o délce 255 znaků.

Tabulka obsahuje tyto sloupce:

- **id** – unikátní identifikátor řádku (char).
- **value** – hodnota ve formátu řetězce (varchar).
- **timestamp** – datum a čas, kdy byla hodnota změřena či nastavena (datetime).
- **status** – stav hodnoty (varchar).
    - Určuje, zda byla hodnota nastavena v platformě (FINISHED) nebo získána z technologie (MEASURED).
    - Tabulka může obsahovat i neplatné hodnoty, ty jsou označeny statusem CANCELED.
- **element_id** – cizí klíč do tabulky IoT prvků (varchar).
    - Určuje IoT prvek, kterého se hodnota týká.
- **author_id** – cizí klíč do tabulky uživatelů (varchar).
    - Určuje uživatele, který v platformě vykonal akci vedoucí ke změně hodnoty.
    - V případě, že hodnotu nastavila technologie (MEASURED) a není tak možné zjistit uživatele, pole obsahuje hodnotu null.

Ukázku této databázové relace je možné vidět v tabulce (Tabulka 4).

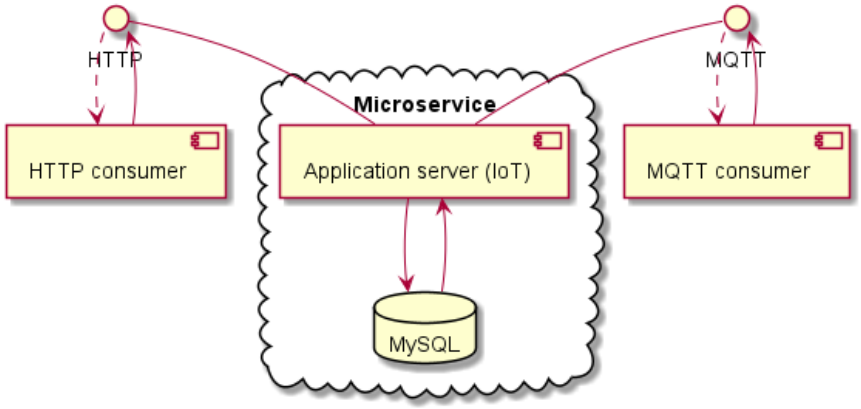
Tabulka 4 Ukázka databázové relace ukládající historické hodnoty IoT prvků.

| id | value | timestamp | status | element_id | author_Id |
|---|---|---|---|---|---|
| 1a3 | 100 | 2021-09-02 12:37:49 | MEASURED | c21 | null |
| 1b2 | true | 2021-09-02 12:39:25 | FINISHED | bb2 | 152 |
| 1ac | false | 2021-09-02 12:39:28 | FINISHED | 21b | 132 |
| 2ca | 24.12 | 2021-09-02 12:41:18 | MEASURED | 138 | null |
| 3aa | 23 | 2021-09-02 12:43:18 | CANCELED | 23d | 3ac |

### 5.4.1 Problém odezvy čtení a zápisu

Tento způsob uchování hodnot je poměrně efektivní v případě, že jedná maximálně o vyšší tisíce zpráv denně. Problém nastává v případě, že se tento průtok zpráv bude zvyšovat. Jelikož relační databáze, která navíc kromě těchto dat pracuje s kompletním databázovým modelem mikroslužby, má určité limity. Potřebný výkon při vyhodnocování dotazů nad

tabulkou historických hodnot může do jisté míry ovlivňovat dostupnost ostatních dat této mikroslužby. Model komponent mikroslužby je na následujícím obrázku (Obr. 13).




Obr. 13 Aktuální model komponent mikroslužby uchovávající historické hodnoty.

Možnosti škálování jsou v klasických relačních systémech poměrně omezené. Pro zlepšení výkonu relačních databází je nutné zvyšovat výkon stroje, na kterém databáze běží. Aktuální výpočetní parametry by se jistě daly zlepšit, ale každý stroj má své fyzické limity a není možné vertikálně škálovat do nekonečna.

Aktuální struktura relační tabulky pro uchování historických hodnot má také svá úskalí a její optimalizace by jistě posunula zmíněné limity dále. Nicméně v případě expanze platformy se jedná pouze o oddalování problémů.

I když proces vyhledání v MySQL databázi s použitím engine InnoDB je poměrně rychlý, je takřka neškálovatelný. Jakmile databáze začne narážet na svoje limity jedinou cestou je již zmíněné zvýšení výkonu stroje.

### 5.4.2 Problém nárůstu velikosti dat

Dalším problémem je narůstající velikost dat. V testovacích datech ze vzorového domu je **5 234 632 záznamů** historických změn hodnot jednotlivých IoT prvků. Tyto data zabírají **1 281 MB**. Což vychází cca 224 bytů na jeden řádek. Je třeba snížit kapacitu jednoho záznamu co nejvíce je možné s použitím vhodné struktury, uchování jen nutných dat či jinou optimalizací.

### 5.4.3 Problém rozmanitosti datových typů

Přestože jsou získaná data ze senzorů, co se týče datových typů různorodá, v aktuální struktuře jsou všechny historické hodnoty uloženy jako SQL typ **varchar (255)**. Nehledě na skutečný datový typ, ve kterém byly ukládány. Takové řešení má za důsledek zbytečné plýtvání kapacitou, jelikož např. číselná reprezentace může být uchována mnohem efektivněji. Velikost hodnoty uložené v datovém typu **varchar je vždy L + 1**, kde L je počet znaků v řetězci. Porovnání způsobu uchování podle datových typů je v následující tabulce (Tabulka 5).

Tabulka 5 Porovnání velikostí pro číselné hodnoty podle různých datových typů.

| Hodnota | VARCHAR | MEDIUMINT | SMALLINT |
|---|---|---|---|
| 753200 | 7 bajtů | 3 bajty | Mimo rozsah |
| 6280 | 5 bajtů | 3 bajty | 2 bajty |

Kvůli omezením plynoucím z použití relačních databází by se jako nejvýhodnější jevilo ukládat hodnoty ve formě binárních vektorů a přetypování na konkrétní datové typy řešit aplikačně. Což by obnášelo určitou režii vzniklou přetypováváním na úrovni aplikace. V jiných databázích, něž klasických relačních, je možné ukládat hodnoty v různých datových typech bez nutnosti cokoli přetypovávat. Většina NoSQL databází díky svojí flexibilitě umožňuje ukládat do stejné struktury hodnoty různých datových typů.

### 5.4.4 Návrh řešení

Platforma by měla být připravena na vysoký nápor zpráv, které musí ukládat. Je tedy nutné tyto data oddělit od stávajícího databázového modelu mikroslužby a přemístit je do samostatné databáze, která bude součástí nové mikroslužby. Tímto krokem se oddělí režie potřebná k rychlým operacím nad aktuálními daty a režie potřebná k práci s historickými hodnotami.

Oddělením do samostatné mikroslužby také vzniká určitá svoboda při výběru konkrétní databázové technologie. Od datového modelu stávající mikroslužby se totiž oddělí část zpracovávající historické hodnoty. Nově oddělená část lze relativně snadno přestrukturovat, jelikož se jedná v původním databázovém modelu o jednu tabulku.

Nově vybraná databázová technologie by měla umožňovat pokročilé možnosti škálování a být optimalizovaná pro práci s velkým objemem dat. Výběrem konkrétní databázové technologie se bude zabývat následující kapitola.

## 5.5 Testovací dataset

Pro potřeby ladění, testování a validaci algoritmů pro data mining je k dispozici dataset obsahující historii komunikace IoT prvků ze vzorového domu. Struktura této části datového modelu byla popsána v předešlé sekci.

### 5.5.1 Formát souboru dat

Data jsou uchována jako export SQL insertů vygenerovaných pomocí nástroje DBeaver z produkční databáze platformy MyMight. Nástroj umí vygenerovat SQL skript, který obsahuje seznam příkazů v jazyce SQL. Skript je takto přímo spustitelný a provede nahrání dat, které sám drží v rámci příkazů, do předem definované databázové tabulky. Ukázku operace INSERT pro jeden řádek lze vidět v následujícím textovém poli.

```
INSERT INTO sense_element_log
(id,element_id,project_id,`timestamp`,value,status,status_change,
updatedby_id)
VALUES
('01c6da71-b971-4b11-bf14-194395d481e9','7079e33d-8381-43ea-a769-
2720a99c03cf','e4f022c7-9113-4361-9571-64df30a7dcf6','2021-11-15
02:08:51.602000000','false','MEASURED','2021-11-15
02:08:51.602000000',NULL);
```

### 5.5.2 Charakteristika dat

Jedná se o soubor historických hodnot zaznamenaných z IoT zařízení, které jsou součástí vzorového domu. Jde o reálnou implementaci několika technologií v rodinném domě. IoT v tomto domě pokrývá většinu základních případů užití jako je osvětlení, stínění, zabezpečení, topení a klimatizaci i zásuvky a různé spínání. Pro představu množství přenesených dat nad takovým běžným projektem je zde uvedena tabulka (

| Popis | Hodnota |
|---|---|
| Počet záznamů | 5 234 632 |
| Datum a čas prvního záznamu | 27.01.2021 13:53:22 |
| Datum a čas posledního záznamu | 15.11.2021 09:50:23 |
| Počet IoT prvků | 112 |
| Počet zaznamenávaných dní | 293 |

| | |
|---|---|
| Průměrný počet záznamů za 1 den | 17 865 |
| Průměrný počet záznamů z 1 prvku za 1 den | 159 |

Tabulka 6) se základními statistickými údaji o tomto datasetu.

Běžné agregační funkce, které by dávaly smysl jako např. průměrná historická hodnota v datasetu a další podobné, není možné vyjádřit nad celým tímto projektem, jelikož jsou historické hodnoty příliš rozmanité, co se týče datových typů.

Tabulka 6 Statistika datasetu.

| **Popis** | **Hodnota** |
|---|---|
| Počet záznamů | 5 234 632 |
| Datum a čas prvního záznamu | 27.01.2021 13:53:22 |
| Datum a čas posledního záznamu | 15.11.2021 09:50:23 |
| Počet IoT prvků | 112 |
| Počet zaznamenávaných dní | 293 |
| Průměrný počet záznamů za 1 den | 17 865 |
| Průměrný počet záznamů z 1 prvku za 1 den | 159 |

# 6 NÁVRH SYSTÉMU PRO PERZISTENCI HISTORICKÝCH HODNOT

Vzhledem k problémům, které jsou zmíněny v předchozí kapitole, se stávající systém uchovávání historických hodnot jeví jako trvale neudržitelný a neefektivní. Jedním z cílů této práce je navrhnout systém, který by byl optimalizovaný pro uchování historických hodnot získaných z IoT zařízení a také připravený pro jejich případné zpracovávání s pomocí algoritmů z oblasti data miningu.

## 6.1 Požadavky na systém

Možností vývoje takového systému je mnoho. Jestli však musí být něco dáno, ještě předtím, než se samotný vývoj zahájí, jsou to požadavky, které musí daný systém splňovat. Specifikace požadavků je jedním z prvních kroků, téměř každé metodiky vývoje softwaru. Proto ani v tomto případě nebudou zanedbány. I vzhledem k návrhu je nutné dopředu znát tyto požadavky.

### 6.1.1 Bezpečnost

Data, které systém bude sbírat pochází z automatizace rezidenčních, komerčních i průmyslových budov. Svým způsobem tak zpětně popisují chování přístrojů v tomto objektu. Je tedy zcela nezbytné, aby data byla dostatečně zabezpečena proti neoprávněnému přístupu nebo úniku.

### 6.1.2 Integrace

Systém by měl být lehce integrovatelný do stávající infrastruktury IoT platformy MyMight. Zároveň by však měl být dostatečně obecný na to, aby mohl být integrován i do jiného systému. Integrace je jedním ze základních pilířů v oblasti IoT. Co možná nejlepší integrace by měl systém docílit vystavením dostatečně obecného a lehce použitelného API, na které se může jiný systém napojit.

### 6.1.3 Kvalita

Požadavkem na softwarový produkt je také co možná nejlepší kvalita kódu, které by měl systém dosahovat, aby byl zdrojový kód systému přehledný, snadno testovatelný a lehce rozšiřitelný. Zdrojový kód by měl reflektovat aktuální trendy a návrhové vzory tak, aby byl pro jiné vývojáře pochopitelný, a to i v případě složitého systému.

### 6.1.4 Jednoduchost

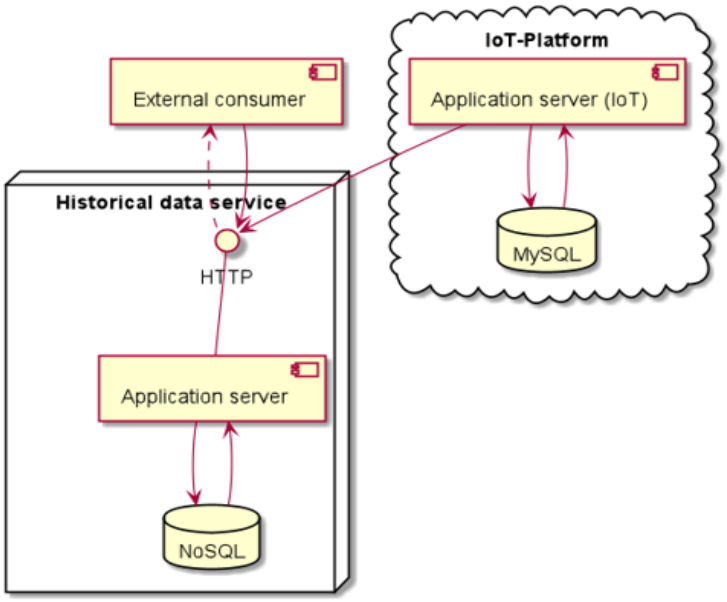
Inženýrský přístup k vývoji softwaru je snahou najít co nejjednodušší a nejefektivnější řešení daného problému. Systém by měl sestávat z menších jednoduchých částí tak, aby mohl být dále rozšiřován s co nejmenším nárůstem složitosti. Také by měl obsahovat pouze části, které jsou po něm žádány a měl by znát své hranice. V tomto případě by měl navrhovaný systém pracovat pouze s historickými hodnotami a neměl by např. zprostředkovávat přenos hodnot pro ovládání v reálném čase.

### 6.1.5 Rychlá odezva

Systém by měl být konstruován tak, aby při zpracování velkých objemů dat dokázal rychle a pokud možno konzistentně odpovídat na příchozí požadavky. Zároveň by měl být také dostupný po celou dobu jeho běhu. Nicméně v distribuovaném prostředí není možné splnit všechny tyto požadavky (viz. 2.2.2 CAP teorém), ale je žádoucí najít vhodný kompromis.

## 6.2 Návrh systému

V této sekci bude proveden kompletní návrh systému pro perzistenci big dat z IoT prvků. Výsledný návrh by měl reflektovat všechny požadavky zmíněné v předchozí sekci. Samozřejmostí je i příprava integrace na stávající infrastrukturu integrační IoT platformy MyMight. Návrh této integrace je načrtnut pomocí diagramu komponent v obrázku (Obr. 14).




Obr. 14. Ukázka možné integrace systému na IoT platformu MyMight.

Není požadováno, aby systém obsahoval grafické uživatelské rozhraní. Komunikace se systémem by měla probíhat přes univerzální API. Univerzálním se zde myslí takové aplikační rozhraní, které není závislé na konkrétních technologiích a programovacích jazycích použitých v systému.

### 6.2.1 Systém jako webová služba

Aby došlo ke splnění požadavků na dostupnost je nutné nasadit systém jako stále běžící proces, který bude dostupný pro komunikaci směrem ke klientům přes ucelené rozhraní po celou dobu svého běhu. Právě tyto možnosti nabízí webové služby.

Aktuálními trendem v oblasti webových služeb je REST API, které je univerzální k technologiím a umožňuje definovat ucelené rozhraní podle určitých pravidel. Další výhodou při použití tohoto rozhraní je snadná integrace na platformu MyMight, která využívá REST API pro komunikaci mezi klienty a mikroslužbami.

Architektura mikroslužeb umožnuje nezávisle vytvořit samostatně fungující softwarový produkt, který disponuje univerzálním rozhraním pro možnosti budoucí integrace.

### 6.2.2 Výběr programovacího jazyka

Díky aktuálnímu trendu architektur orientovaných na služby většina nejpoužívanějších programovacích jazyků umožňuje ať už nativně nebo s použitím knihoven a frameworků tvorbu webových služeb. V tomto případě je tedy výběr programovacího jazyka poměrně otevřený.

Při výběru by se určitě mělo přihlížet také k účelu webové služby a vlastnostem daného jazyka. Jelikož služba bude pracovat s velkými objemy dat vybraný programovací jazyk by neměl v tomto ohledu zaostávat. Mezi požadované vlastnosti patří rychlost, objektová orientace, bezpečnost, podpora více vláknových aplikací a dostupnost rozhraní pro práci s datovými sklady či NoSQL databázemi. Jistou výhodou pak může být i dostupnost knihoven pro zpracování a preprocessing dat.

Nejznámější jazyky aktuálně používané pro tvorbu webových služeb, které v určité míře splňují výše uvedené vlastnosti jsou:

- Python
- C#
- Java
- Go

- Rust

Pro tvorbu systému byl zvolen programovací jazyk **Java**. Především z důvodů rychlosti, snadné možnosti paralelizace a dostatečnou zásobou frameworků, knihoven a rozhraní. Taktéž z důvodu potřeby budoucí integrace na platformu MyMight, která je tvořena mikroslužbami postavenými na frameworku Spring Boot v programovacím jazyce Java.

Java disponuje několika knihovnami, které ulehčují tvorbu webových služeb. Některé jsou nativní součástí Java-EE (Enterprise Edition) jiné jsou externí, ale volně ke stažení, většinou pod licencí open-source.

Nativní knihovny pro tvorbu webových služeb jsou dvě.

- JAX-WS – Pro tvorbu rozhraní typu SOAP.
- JAX-RS – Pro tvorbu rozhraní typu REST.

Kromě nativních knihoven se objevují i řešení ze strany externích společností. Často bývají propracovanější s možností jednoduché integrace na další frameworky a knihovny této společnosti. Jsou také častěji udržované a disponují bezesporu kvalitnější dokumentací. Nevýhodou může být nutnost externě dodat knihovnu do projektu. Dnes již však existují nástroje, které toto velice snadno řeší pomocí definice závislostí (Maven, Gradle). Jedním z externích frameworků, který je určen k tvorbě webových služeb je také Spring.

### 6.2.3 Spring

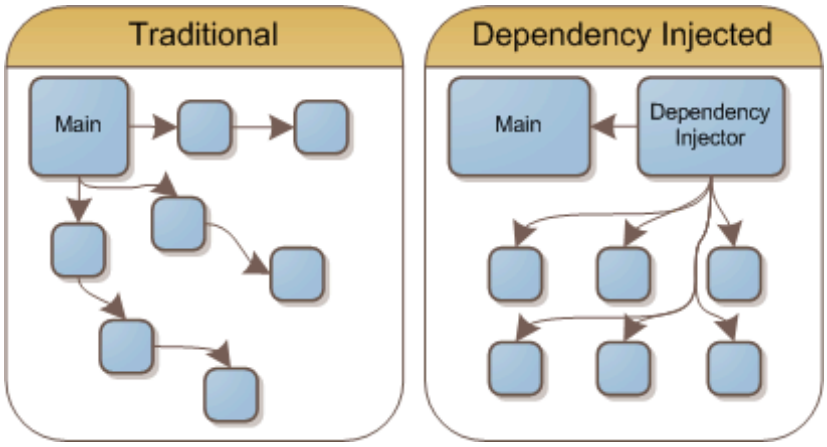
Spring Framework[9] je Java framework, který poskytuje komplexní infrastrukturní podporu pro vývoj aplikací v jazyce Java. Spring se stará o infrastrukturu tak, aby se vývojáři aplikací mohli plně soustředit na aplikační kód. Je rozdělen do modulů. Aplikace si mohou vybrat, které moduly potřebují. Jádrem jsou moduly „core“, včetně konfiguračního modelu a mechanismu dependency injection. [26]

#### Dependency injection

Dependency injection je aktuálním trendem v objektově orientovaném programování. Umožňuje vytvářet mezi komponentami vazby bez referencí definovaných v době překladu [27]. Pokud objekt potřebuje jiné objekty (je na nich závislý), dependency injection

[9] https://spring.io/

kontejner je schopný mu při spuštění aplikace tyto objekty dodat, aniž by si je musel vytvářet sám. Tato technika pomáhá udržovat aplikační kód vzájemně nezávislý a lehce testovatelný. Je základním prvkem frameworku Spring. Obrázek (Obr. 15) vizualizuje rozdíl v dodávce závislostí při použití depedency injection a bez něj.




Obr. 15 Vizualizace závislostí mezi objekty při použití dependency injection. [27]

Momentálně Spring poskytuje sadu modulů a knihoven jejich cílem je zajistit pohodlnější a rychlejší vývoj a správu Java aplikací. V rámci tohoto systému by byly využity následující moduly a knihovny:

- **Spring core container**
  - Poskytuje základní části frameworku Spring pro podporu technik jako je depencency injection nebo IoC (Inversion of Control).
- **Spring web MVC**
  - Podpora pro tvorbu webových služeb podle návrhového vzoru Model-view-controller.
  - Podpora pro tvorbu REST API.
- **Spring data**
  - Objektově relační mapování s možností integrace na několik různých druhů databází.
  - Disponuje knihovnami i pro integraci s různými NoSQL databázemi (Redis, MongoDB, Cassandra, Couchbase, Elasticsearch).
- **Spring security**
  - Obsahuje podporu pro různé druhy autentizace a autorizace.

### 6.2.4 Spring boot

Jedná se o rozšíření frameworku Spring, které umožňuje tvorbu samostatně spustitelných mikroslužeb. Vychází z paradigma **convetion-over-configuration**, což je přístup, který se snaží snížit počet různých nutných nastavení na minimum za cenu, že vše bude fungovat konvenčním způsobem. Cílem je co nejvíce zjednodušit tvorbu, nasazení i následnou správu webových aplikací. Spring boot vytváří kontejner, který obsahuje webovou aplikaci spolu s embedovaným webovým serverem. Díky tomu je možné, jak i sám Spring inzeruje, kontejner „**pouze spustit**“. V případě, že není třeba přetížit žádnou konfiguraci mikroslužby, lze Spring Boot kontejner jednoduše spustit jako JAR:

```
java -jar hist-ws.jar
```

### 6.2.5 Výběr frameworků

Pro tvorbu webové služby jsem se rozhodl použít framework **Spring web MVC** a **Spring Boot**. Hlavním důvodem je jednoduchost vývoje a nasazení webových služeb postavených na tomto frameworku. Spring je oproti nativním řešením JAX-RS propracovanější, rozšiřitelnější, udržovanější, také umožňuje jednoduchou integraci na další moduly frameworku, které jsou v tomto systému též využitelné – **Spring security** a **Spring data**.

Nespornou výhodou rozšíření Spring Boot je zjednodušení nasazení a konfigurace webové služby jako celku. Díky multiplatformnímu kontejneru je tak možné výslednou komprimovanou webovou službu spustit na jakémkoli hardwaru, kde se nachází JVM (Java Virtual Machine).

### 6.2.6 Architektura

Jelikož je Spring opravdu plnohodnotný framework, tak svým způsobem z části definuje architekturu systému. Architekturu aplikační části si však definuje sám vývojář, potažmo architekt.

Pro webové služby je mnoho doporučovaných architektur, jejichž výhody pak závisí na konkrétním systému a problémové doméně. Některé doporučené architektury jsou vhodnější pro služby se složitější business logikou, jiné se více hodí pro informační systémy, jejichž primární účel je transformace vstupních dat a jejich uložení do databáze.

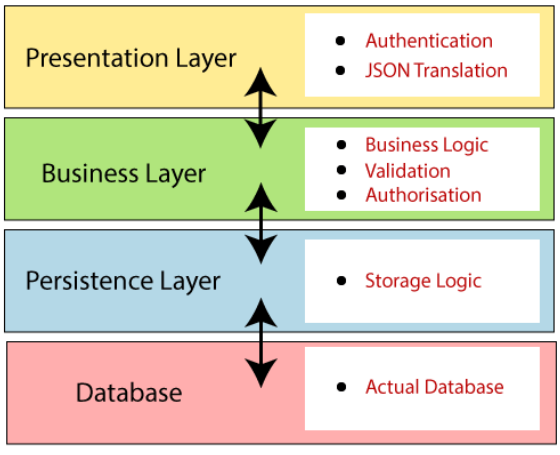
### Vrstvená architektura

Velice oblíbená architektura používaná nejen pro webové služby. Často se v případě použití frameworku Spring kombinuje s návrhovým vzorem Model-view-controller.

Hlavní myšlenou architektury je rozdělit systém na několik vrstev, z nichž každá komunikuje pouze s vrstvou architektonicky umístěnou nad nebo pod ní. Obvykle se používá dělení na tři nebo čtyři vrstvy. [28]

- Prezentační vrstva
  - Zpracovává HTTP požadavky, převádí vstupní JSON na objekt, ověřuje autentizaci a přenáší vstupní objekty do aplikační vrstvy. [28]
- Aplikační vrstva
  - Obsahuje aplikační kód (tzv. business logiku). Provádí autorizaci a validaci vstupních objektů.
  - Někdy se také označuje jako servisní vrstva.
- Perzistentní vrstva
  - Definuje logiku potřebnou k perzistenci objektů. Obsahuje rozhraní, které využívá aplikační vrstva.
  - Pro relační databáze definuje objektově relační mapování.
- Databázová vrstva
  - Konkrétní použitá databáze. Zodpovídá za provádění CRUD operací (Create, Read, Update, Delete). [28]

Jak jsou jednotlivé vrstvy propojeny a co obsahují lze vidět na obrázku (Obr. 16).



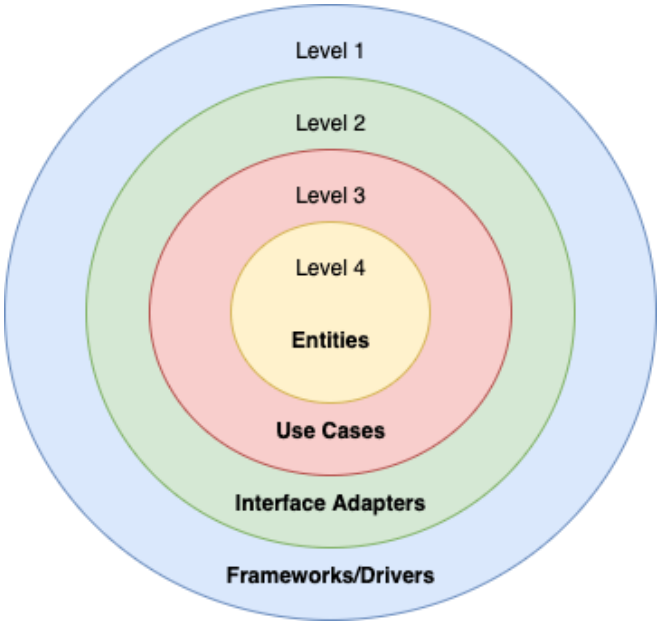

Obr. 16 Vrstvená architektura používaná pro Spring Boot aplikace. [28]

Tato architektura je vhodná pro jednoduché webové služby či aplikace, které neobsahují složitou business logiku. V případě tohoto systému se dá předpokládat, že služba bude obsahovat složitější logiku zahrnující zpracování dat. Nicméně z pohledu doménového modelu se jedná o velice jednoduchou službu. Primárně totiž pracuje pouze s historickými hodnotami z IoT prvků. Z tohoto důvodu se jeví vrstvená architektura jako **vhodná** pro tento systém.

**Alternativní architektury**

Při vývoji mikroslužeb postavených na frameworku Spring Boot není vrstvená architektura jedinou používanou. Oblíbené jsou i další architektury jako např.

- **Čistá architektura**
  - Základní myšlenkou je rozdělení systému do úrovní na základě business hodnoty. Nejvyšší úroveň obsahuje business pravidla, přičemž každá nižší se více přibližuje vstupně-výstupním operacím. [29]
  - Kód ve vnitřních vrstvách nemá žádnou znalost kódu ve vnějších vrstvách.
  - Rozdělení do vrstev je patrné na obrázku (Obr. 17) Obr. 17 Používané úrovně v Čisté architektuře.




Obr. 17 Používané úrovně v Čisté architektuře. [29]

- **Hexagonální architektura**
    - Pracuje s podobnou myšlenkou jako čistá architektura. Snahou je vytvářet softwarové aplikace kolem problémové domény a business logiky s cílem izolovat ji od vnějších faktorů. [30]
    - Na rozdíl od čisté architektury netvoří vrstvy, ale snaží se pomocí adaptérů kompletně oddělit business logiku od ostatních částí aplikace.

Obě výše zmíněné architektury jsou bezesporu zajímavé a jistě si ve světě vývoje softwaru najdou své místo. Zároveň také existuje mnoho dalších jiných architektur, které jsou opět v něčem výhodnější v něčem jiném zase méně výhodné. Pro potřeby tohoto systému, jak již bylo zmíněno, bude využita **vrstvená architektura**.

### 6.2.7 Definice REST rozhraní

Primárním rozhraním pro komunikaci s mikroslužbou bude REST. Mezi hlavní důvody výběru tohoto rozhraní se řadí:

- Technologická nezávislost
- Všeobecná rozšiřitelnost
- Podpora frameworku Spring
- Jednoduchost a dobrá čitelnost zasílaných zpráv

Rozhraní jako takové je možné rozdělit na 2 části:

1. Rozhraní pro uchování dat
2. Rozhraní pro práci s daty

**Rozhraní pro uchování dat**

Účelem tohoto rozhraní je přijímaní požadavků na zápis hodnot. Přijímané hodnoty prochází jistou formou preprocessingu a následnou perzistencí pro jejich případné další použití.

Základ tohoto rozhraní by měla tvořit jedna webová metoda, která zpracovává vstupní JSON. Ten obsahuje položky pro všechny potřebná data, která by měla být uložena v rámci jednoho vzorku hodnoty IoT prvku. Vstupní JSON by měl obsahovat všechny položky definované tabulkou (Tabulka 7).

Tabulka 7 Přehled všech položek vstupního JSONu pro uložení hodnoty.

| Název | Popis | Datový typ |
|---|---|---|
| value | Samotná ukládaná hodnota. | Libovolný |
| timestamp | Časové razítko určující, kdy byla daná hodnota získána. | Datetime |
| type | Určuje datový typ hodnoty. | String (Enum) |
| status | Určuje, zda byla hodnota získána z technologie (MEASURED) nebo přišla jako odpověď na požadavek z platformy (FINISHED). | String (Enum) |
| elementId | Identifikátor IoT prvku | String |
| createdBy | Identifikátor uživatele zodpovědného za změnu hodnoty. V případě uložení hodnoty iniciované z technologie není možné uživatele identifikovat (null). | String |

Není nutné jednotlivé položky hierarchicky zanořit. Webová metoda by měla být typu HTTP **POST**. Tento typ je podle definice rozhraní REST určený k zápisu dat. Výsledný vstupní JSON by pak mohl vypadat jako:

```
{
    "value": 1,
    "timestamp": "2015-11-18T18:05:38.000+0200",
    "type": "INTEGER",
    "status": "MEASURED",
    "elementId": "c12e9e85-df89-4000-ab13-273e95fb7865",
    "createdBy": "dc3cf9bf-9753-4005-b86c-cc1d737ae932"
}
```

**Rozhraní pro získávání dat a informací**

Toto rozhraní je složitější, protože by mělo obsahovat několik různých pohledů na historické hodnoty včetně možnosti jejich filtrace. Všechny tyto dotazy by měly být definovány jako webové metody HTTP typu **GET.** Tento typ je určený k získávání dat. Nese s sebou výhody v podobě podpory cachování a možností tvorby záložek.

Velice jednoduchým pohledem je získání všech historických hodnot za jeden prvek v určitém časovém intervalu. Požadavek takové webové metody by měl vypadat jako:

```
GET /hist-ws/log/?elementId=c12e9e85-df89-4000-ab13-273e95fb7865
&from=2021-11-12T21%3A28%3A52.186708600%2B01%3A00
&to=2021-11-15T21%3A28%3A52.186708600%2B01%3A00 HTTP/1.1
```

Výstupem této metody by měl být JSON, který obsahuje jednotlivé vzorky historických hodnot splňující filtrovací kritéria (elementId, from, to).

```
{
    "responseCode": "OK",
    "elementLogDTO": [
        {
            "value": "4",
            "timestamp": "2022-03-28T10:05:38Z",
            "type": "INTEGER",
            "status": "MEASURED",
            "createdBy": "dc3cf9bf-9753-4005-b86c-cc1d737ae932"
        },
        {
            "value": "2",
            "timestamp": "2022-03-28T11:05:38Z",
            "type": "INTEGER",
            "status": "MEASURED",
            "createdBy": "dc3cf9bf-9753-4005-b86c-cc1d737ae932"
        }
    ]
}
```

Podobných metod by mělo rozhraní obsahovat několik, z nichž každá by měla nabízet jiný pohled na data. Některé mohou vracet např. agregované informace o hodnotách. Jelikož se tato práce později bude věnovat především oblasti **data miningu**, měl by systém disponovat metodami, které poskytují vydolované informace z dat. A to tak, aby klienti mohli tyto informace konzumovat a činit podle nich určitá rozhodnutí.

### 6.2.8 Zabezpečení systému

Jelikož systém bude pracovat s citlivými údaji je nutné, aby byl příslušným způsobem zabezpečen. Prvním krokem zabezpečení je použití šifrovaného protokolu pro komunikaci. REST API, využívá protokol HTTP, který má i zabezpečenou variantu – **HTTPS** (Hypertext Transfer Protocol Secure). Ta funguje na principu kombinace protokolu HTTP pro přenos zpráv a protokolu SSL (Secure Sockets Layer) nebo TLS (Transport Layer Security) pro zajištění zabezpečené komunikace. Použití protokolu HTTPS se stalo standardem v oblasti webových služeb.

#### OAuth 2.0

Dalším bezpečnostním prvkem je implementace standardu OAuth 2.0 pro autentizaci a autorizaci požadavků na webové API. Jedná se o otevřený protokol použitelný pro klienty webových, mobilních i sever-side aplikací.

Protokol využívá přístupové tokeny s omezenou platností, které vydává tzv. autorizační server. Ten může token vydat pouze uživateli či klientovi, který se prokáže platnými přihlašovacími údaji. Token je poté jakousi vstupenkou pro získání chráněných dat. Server, jež chráněná data drží, kontroluje platnost tokenu proti autorizačnímu serveru, který jej vydal. Tím získá nejen informaci o platnosti, ale také role a oprávnění spojené s tímto tokenem, které může využít pro autorizaci.

Systém pro uchovávání historických hodnot by v tomto vystupoval v roli serveru, který má k dispozici chráněná data. Není tedy třeba řešit problém získávání přístupových tokenů. Je však nutné implementovat mechanismus kontroly platnosti při přístupu k webovému API. Framework Spring Security obsahuje sadu knihoven, které mají pro OAuth 2.0 podporu a tím velice ulehčí implementaci tohoto protokolu do systému.

OAuth 2.0 byl vybrán z důvodu jeho dostatečné bezpečnosti a jednoduché implementaci díky podpoře frameworku Spring Security. Dalším důvodem je možnost využití autorizačního serveru integrační platformy MyMight, která jej provozuje a aktivně používá pro autentizaci i autorizaci několika svých aplikací a webových služeb.

## 6.3 Uchování dat

V předchozích sekcích se nacházely požadavky na systém jako celek včetně jeho návrhu. Velice důležitou součástí infrastruktury tohoto systému je i úložiště historických hodnot. Stejně tak důležitý je i jeho správný výběr, nastavení a optimalizace. Jedná se totiž o kritickou část, která musí být dostupná po celou dobu běhu systému.

### 6.3.1 Možnosti systémů pro uchování dat

Data, která se by toto uložiště mělo uchovávat, by se dala klasifikovat jako IoT big data. Jsou poměrně specifická v tom, že obsahují vždy časové razítko, které poté v uložené struktuře funguje jako vyhledávací index.

Obecně se pro uchování big dat používají datové sklady nebo NoSQL databáze. Obě tyto varianty byly rozebrány v kapitole 2.

**Datové sklady**

Řešení v podobě datových skladů je vhodné především pro podnikové systémy, které potřebují uchovávat data primárně pro účely analýzy v rámci Business Inteligence. Jsou optimalizované pro ukládání big dat, nicméně se jedná o data s vlastnostmi, které jsou typické pro oblast manažerského rozhodování. Proto se použití datového skladu v tomto systému jeví jako **nevhodné**.

**NoSQL databáze**

Flexibilnější možností uchování big dat obecně nabízí NoSQL databáze. Je to dáno i faktem, že existuje nespočet různých konkrétních NoSQL databázových systémů, z nichž každý má jiné vlastnosti a je vhodný pro jiný typ dat. Některé databázové systémy obsahují optimalizaci pro časové řady. Právě tato optimalizace umožňuje zvýšit efektivitu a snížit kapacitu nutnou pro uchování těchto specifických dat s časovým razítkem.

### 6.3.2 Požadavky na NoSQL databázi

Jak již bylo zmíněno existuje mnoho různých NoSQL databází. Výběr té nejlepší pro daný systém tedy může být poněkud komplikovaný úkol. Kromě všech vlastností, které by konkrétní databáze měla splňovat, je také důležité, aby byla udržovaná, dobře zdokumentovaná a nejlépe s aktivní komunitou. Přehledný seznam databází seřazených podle počtu uživatelů je dostupný na stránce DB-Engines[10].

[10] https://db-engines.com/en/ranking

Požadavky na databázi jsou:

1. Horizontální škálování
    - Množství ukládaných dat může až exponenciálně narůstat.
2. Flexibilní schéma
    - Historické hodnoty mohou být uloženy ve formě různých datových typů.
3. Indexace
    - V databázi je třeba rychle vyhledávat podle určitých atributů (např. ID IoT prvku).
4. Výhodou je nativní podpora časových řad
    - Tato podpora představuje úsporu kapacity a zefektivnění práce s tímto typem dat.
5. Co nejmenší výsledná velikost dat
    - Na systém mohou být kladeny požadavky na uložení až několika stovek hodnot za jeden IoT prvek denně. Cílem je co nejefektivněji využít zdroje a prostředky.
6. Zachování rychlé odezvy i při větším množství dat
    - I přes narůstající velké množství dat je nutné, aby databáze reagovala rychle, jelikož se předpokládají časté dotazy na data v ní uložená.

### 6.3.3 Porovnání možných NoSQL databází

V této sekci budou nastíněny vybrané databázové systémy, které nejlépe odpovídají požadavkům stanoveným v předchozí sekci.

**InfluxDB**

Hlavní předností této databáze je nativní podpora časových řad. Díky tomu je databáze pro časové řady optimalizovaná a splňuje tedy i kritéria pro dosažení co nejmenší kapacity a dostatečně rychlé odezvy. IoT data ze senzorů jsou typem dat, který je přímo typický pro tuto databázi. Může se tedy zdát, že je tato databáze pro systém sloužící k ukládání historických hodnot z IoT prvků ideální.

Problémem je, že kromě perzistence IoT historických hodnot by databáze měla umožňovat ukládat i jiná data, které nemusí nutně disponovat charakteristikou časové řady. Vzhledem k požadavku na data mining a celkově zpracování dat bude nutné ukládat doprovodná data různých algoritmů. Pro tyto účely databáze **není dostatečně flexibilní**.

**Apache Cassandra**

Jedná se o nejznámější sloupcovou databázi. Podporuje horizontální škálování a je vhodná pro aplikace náročné na zápis. Díky ukládání dat do sloupců dosahují uložená data menší velikosti, než tomu je např. u klasických relačních systémů.

Neobsahuje však nativní podporu časových řad, což znamená, že nedisponuje některými vlastnostmi nebo operacemi, které jsou typické pro databáze s nativní podporou tohoto typu dat (time series databáze). Mezi tyto funkcionality patří např. různé agregační funkce nebo interpolace. Podobné vlastnosti mají i ostatní sloupcové NoSQL databáze. Dále **nepodporuje tvorbu sekundárních indexů**, což může být nevýhodné, jelikož právě pomocí těchto indexů systém bude nejčastěji historické hodnoty získávat.

**MongoDB**

Nejpoužívanější a nejznámější NoSQL databáze obecně. Pro ukládání dat používá dokumenty, které jsou uloženy ve formátu BSON (binární JSON). Podobně jako Apache Cassandra má podporu horizontálního škálování a preferuje dostupnost nad konzistencí. I když se jedná primárně o dokumentovou databázi, od verze 5.0 disponuje nativní podporou časových řad.

Databáze umožňuje vytvořit speciální kolekci dokumentů označenou jako **time series**. Od klasické kolekce dokumentů se liší především optimalizací velikosti ukládaných dokumentů a podporou některých agregačních funkcí. Je možné kombinovat standardní dokumentové kolekce s time series kolekcemi v jedné databázi. Což je velice výhodné v případě, že systém pracuje kromě s časovými řadami, také s jinými daty. Což je případ právě navrhovaného systému, který kromě historických dat ze senzorů bude poskytovat možnost uložení různých mezivýsledků vzniklých při běhu data mining algoritmů.

Mezi nevýhody se řadí špatná podpora transakčního zpracování, což bývá typické pro většinu NoSQL databází. Problémem je také nedostatečná podpora relací mezi jednotlivými dokumenty. Uložené časové řady mohou v porovnání s časovými řadami v jiných, pro tento účel optimalizovaných, databázích zabírat více kapacity na disku. To je dáno především faktem, že databáze MongoDB nebyla od začátku vyvíjena jako time series.

### 6.3.4 Výběr databáze

V předchozí sekci byly vybrány a popsány nejvhodnější databáze pro navrhovaný systém. Všechny tyto databáze byly podrobněji prozkoumány a analyzovány spolu s požadavky na

systém a databázi. Vzhledem k požadavkům, byla vybrána databáze **MongoDB**. Především z důvodu podpory nativního optimalizovaného ukládání časových řad spolu s jinými typy dat v jedné databázi.

Databázové schéma podle MongoDB je dostatečně flexibilní pro potenciální ukládání jiných, aktuálně neznámých dat, spojených s historickými hodnotami. Výběru této databáze přispíval i fakt, že je nerozšířenější, často aktualizovaná a disponuje kvalitní dokumentací s podporou komunity.

Systém je architektonicky navržen tak, aby v případě výměny databázového systému stačilo upravit pouze část komunikující přímo s databází a provést migraci dat.

### 6.3.5 Konfigurace a struktura databáze

Datový model tohoto systému je velice jednoduchý, jelikož v základním modelu figuruje pouze jedna kolekce dokumentů. Není tedy třeba podrobnějšího návrhu datového modelu např. v podobě diagramu tříd. Tím pádem v aktuálním datovém modelu ani neexistují žádné vazby (1:1, 1:N, M:N) mezi dokumenty.

Kolekce dokumentů s názvem **historic_values** je tzv. time series kolekcí neboli kolekcí s nativní podporou časových řad. U tohoto typu kolekce je nutné při jejím založení definovat následující parametry.

- **timeField –** Jméno pole, ve kterém je uložen časový údaj. Podle něj se pak vytváří časová řada. Hodnota tohoto pole musí být platné datum dle formátu BSON date. Pro tento systém to bude pole s názvem "timestamp".
- **metaField –** Jedná se o pole, které obsahuje metadata daného vzorku časové řady. Může se jednat o samostatné pole nebo i vnořený dokument. Při ukládání dokumentů databáze pomocí tohoto pole rozděluje časové řady tak, aby byly jednotlivé dokumenty se stejnými metadaty seskupeny a tím tak eliminuje duplicitní ukládání metadat. V tomto systému bude jako metaField sloužit vnořený dokument obsahující pole s názvem "elementId" a "createdBy".
- **granularity –** Definuje časové rozpětí po sobě jdoucích dokumentů, které mají stejná metadata. Přesné nastavení parametru zlepšuje výkon tím, že optimalizuje způsob interního ukládání dat v kolekci časových řad. Pro tento systém bude použita výchozí granularita („seconds"), jelikož předem není možné odhadnout granularitu uživatelských akcí.

- **expireAfterSeconds –** Časový údaj, který určuje, po jaké době se dokumenty automaticky odstraní. Při návrhu této kolekce se parametr nepoužije, avšak není vyloučeno jeho budoucí použití.

Definice kolekce dokumentů historic_values v jazyce MQL (MongoDB Query Language) podle návrhu bude vypadat takto:

```
db.createCollection("historic_values",
  {
    timeseries: {
      timeField: "timestamp",
      metaField: "metadata",
      granularity: "seconds"
    }
  }
);
```

Dokument, který se bude do kolekce ukládat by měl obsahovat atributy, které jsou specifikovány v tabulce (Tabulka 8).

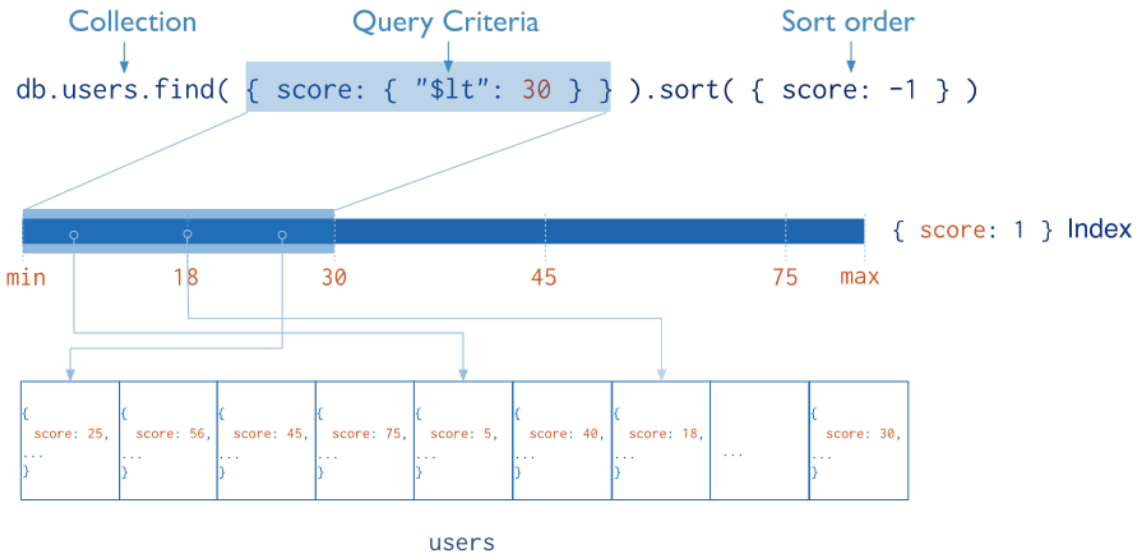
Tabulka 8 Atributy dokumentů ukládaných do kolekce historic_values.

| Název | Datový typ MongoDB | Popis |
|---|---|---|
| _id | ObjectId | Generovaný identifikátor dokumentu |
| timestamp | Date | Časové razítko získané hodnoty |
| value | Mixed (Libovolný typ) | Hodnota |
| type | Int32 | Typ hodnoty |
| status | Int32 | Status (MEASURED, FINISHED) |
| elementId | Binary (UUID) | ID IoT prvku |
| createdBy | Binary (UUID) | ID uživatele spojeného s nastavením hodnoty |

### 6.3.6 Nastavení indexů

Databáze MongoDB implicitně nastavuje index na primární klíč, což je vždy pole s názvem **_id**. V případě, že se předpokládá časté vyhledávání dokumentů v dané kolekci podle jiného pole, než je primární klíč, je vhodné vytvořit na těchto polích indexy. Vyhledávání konkrétních dokumentů bude pak mnohem rychlejší.

Bez použití indexů totiž databázový engine musí projít každý dokument v kolekci a porovnat, jestli hodnoty polí splňují kritéria vyhledávacího dotazu. Indexy jsou speciální datové struktury, které ukládají malou část datové sady kolekce ve snadno průchodné podobě. Index uchovává seřazené hodnoty určitého pole nebo množiny polí. Uspořádání položek indexu je velice efektivní při vyhledávání konkrétních hodnot indexu. Při použití indexu je třeba počítat s nárůstem kapacity databáze, jelikož se indexy ukládají zvlášť. Příklad použití indexu při vyhledání (viz. 2.5.2) podle kritéria „menši než“ ($lt) je možno vidět v obrázku (Obr. 18). [31]




Obr. 18 Ukázka procesu vyhledání podle indexu v databázi MongoDB. [31]

V případě kolekce dokumentů **historic_values** je nutné vyspecifikovat pole, které budou figurovat v kritériích vyhledávání. Znalost problémové domény a návrh systému pomůžou analyzovat, která pole budou použita pro vyhledávání. Podle doporučení MongoDB je pro time series kolekci nejvýhodnější vytvořit složené indexy (compound index), které se skládají z **timeField** a různých kombinací atributů v **metaField**. V případě systému pro ukládání historických hodnot by jednalo o vytvoření 2 složených indexů – kombinace (timestamp, elementId) a kombinace (timestamp, createdBy). Příkaz pro vytvoření těchto indexů, by pak mohl vypadat jako:

```
db.historic_values.createIndex(
      {"timestamp": 1, "metadata.elementId": 1}
)
db.historic_values.createIndex(
      {"timestamp": 1, "metadata.createdBy": 1}
)
```

Indexy je možné upravovat u již existujících a naplněných kolekcí, takže v případě potřeby je možné index přidat, odstranit nebo modifikovat.

### 6.3.7 Optimalizace velikosti dokumentu

Za optimalizaci velikosti dokumentu je možné považovat ukládání hodnoty (pole "value") v původním získaném datovém typu. V případě použití klasických relačních systému by tento způsob uložení nebyl možný, jelikož se u každé tabulky musí pevně definovat datový typ pro každý sloupec. Musely by se hodnoty ukládat např. jako binární vektory a jejich převod do konkrétních datových typů by se pak řešil aplikačně při každém získání hodnoty, což by způsobovalo režii navíc.

Další optimalizací výsledné velikosti dokumentu může být uložení polí výčtového typu jejich ordinální hodnotou namísto uložení celé textové reprezentace. Tímto způsobem je možné především u delších textových reprezentací zajistit, že namísto $L + 1$ bytů, kde $L$ je délka textu, bude pole v dokumentu mít pevnou velikost 4 bytů pro datový typ Int32, případně 1 byte pro datový typ Byte. Takový způsob má ještě výhodu a to, že při případném přejmenování textové reprezentace výčtového typu se nerozbije konzistence s daty v databázi. Nevýhodou je, že při změně pořadí jednotlivých hodnot dojde k právě zmíněnému rozbití konzistence.

Díky těmto optimalizacím zabírá jeden dokument v této kolekci velikost **cca 220 bytů**. Tato velikost se může napříč různými dokumenty v kolekci kvůli rozdílnosti dat lišit. Také je nutné specifikovat, že se jedná se o velikost před kompresí, která může u databáze MongoDB snížit velikost dokumentu i na méně než 50 % z původní velikosti.

## 6.4 Zhodnocení

Díky datasetu ze vzorového domu je možné porovnat uložení dat v původní databázi MySQL, kde byla data uložena ve struktuře popsané v kapitole 5 a databázi MongoDB se strukturou popsanou v této kapitole. Dataset, který poslouží pro porovnání, byl podrobněji rozebrán v sekci 5.5.

### Porovnání velikosti dat

Jeden z faktorů, který je možné porovnat je velikost databáze. Porovnávat se musí databáze se stejnými daty. V tomto případě budou obě databáze naplněny datasetem ze vzorového domu, který obsahuje 5 234 632 záznamů historických hodnot.

- MySQL – tabulka historických hodnot **sense_element_log**
  - 1 281 MB
- MongoDB – kolekce dokumentů **historic_values**
  - 637 MB

Jak je vidět data ukládaná databází MongoDB zabírají poloviční kapacitu disku. Oba údaje uvádí výslednou konečnou komprimovanou velikost dat bez indexů.

### Porovnání rychlosti

Při porovnání rychlosti se bude porovnávat dotaz (selekce) na historické hodnoty, které mají ID IoT prvku "elementId" rovno hodnotě '7705f900-91f2-4f35-a917-1526b0e42958' a "timestamp" menší než '2021-04-09 14:30:52'.

- MySQL - 128 ms
- MongoDB – 16 ms

Doba odezvy pro databázi MySQL byla získána z databázového klienta DBeaver a pro databázi MongoDB pomocí funkce „executionStats“. Tento rozdíl v měření mohlo způsobit rozdílné nastavení indexů v obou systémech i rozdílný nástroj pro měření. Získané výsledky jsou tak spíše orientační.

### Výhody samostatné mikroslužby

Oddělením aplikační logiky spojené s historickými hodnotami do samostatné mikroslužby se docílí zmírnění zátěže na IoT platformu MyMight, především na mikroslužbu, která se stará o čtení a zápis aktuálních hodnot a nyní historické hodnoty zpracovává a perzistuje.

# 7 DATA MINING IOT DAT

Jeden z hlavních cílů této práce je na data, která jsou popsána v předchozích kapitolách, aplikovat algoritmy z oblasti data miningu za účelem získání informací a znalostí. V případě, že se podaří získat zajímavé informace, vybrané algoritmy by mohly být v budoucnu integrovány do systému pro uchování historických hodnot (viz. kapitola 6).

Systém umožňuje integraci na IoT platformu MyMight, která by tyto algoritmy mohla využít a postavit na informacích získaných z dat zajímavé funkce pro svoje zákazníky. Tento krok by platformě mohl pomoci se odlišit od konkurence.

V sekci 7.1 jsou uvedeny úlohy data miningu, které mohou být vzhledem ke struktuře dat přínosné. Následující sekce obsahují popisy konkrétních implementovaných algoritmů včetně způsobu jejich implementace.

## 7.1 Možné úlohy data miningu

Prvním krokem podle procesu CRISP-DM je definice způsobu aplikace data miningu a jeho cílů. Účelem této práce není aplikovat data mining podle procesu CRISP-DM, ale je možné se inspirovat některými jeho kroky a postupy. Cílem této práce je především prověřit možnosti algoritmů pro získání informací z IoT dat.

Způsoby aplikace data mining algoritmů a modelů se liší v závislosti na různých požadavcích, které od data miningu očekáváme. Následuje tedy definice několika úloh, které je možné vzhledem k charakteristice dat provést a jsou z pohledu IoT zajímavé. Rozpoznání vzorů a detekce anomálií jsou úlohy, které jsou dány zadáním této práce a je tedy nutné je zahrnout.

### 7.1.1 Detekce anomálií

Jedná se o poměrně častou úlohu, co se týče IoT dat. Cílem je získat informace o neobvyklých hodnotách, které se v datech mohou vyskytovat. Tyto hodnoty totiž mohou indikovat, že je něco jinak než obvykle a je třeba tomu věnovat pozornost. Konkrétní případy z praxe, které využívají detekci anomálií jsou např.

- Rozpoznání chybového stavu senzoru či akčního členu.
- Rozpoznání nechybové, ale neobvyklé hodnoty (např. teplota, vlhkost vzduchu, hodnota $CO_2$ a podobně).
    - Nebezpečné hodnoty.

- Špičky ve spotřebě.
- Identifikace nebezpečného trendu.

### 7.1.2 Rozpoznání častých vzorů a kombinací

V IoT datech se mohou skrývat opakující se vzory, které nemusí být bez aplikace data miningových metod a algoritmů viditelné. To je dáno množstvím dat. Čím je množství vyšší, tím obtížněji se v datech hledají opakující se vzory.

Opakující se vzory či kombinace nesou informace, které mohou být přínosné např. při **automatizaci rutinních činností**. V kontextu IoT platformy MyMight je možné uživatelům, kteří vykonávají několik posloupných akcí téměř vždy společně nabídnout tyto kroky zautomatizovat pomocí automatických scénářů nebo akcí (viz. 5.2.1). Takové rozhodnutí poté může pomoci uživateli poskytnout komfort.

Další informací, která může z této detekce vyvstat je rozpoznání opakujících se nežádoucích chování. Může se jednat o chyby nebo jiné neočekávané chování. Případná identifikace může pomoci odhalit chybu a najít její řešení.

### 7.1.3 Predikce

Obecně u časových řad je predikce nejčastější úlohou data miningu. Jedná se o využití informací z historických dat posloupných v čase pro predikování budoucího vývoje. Popularita predikce se těší především ve finanční a investiční oblasti, kdy je využívána pro předpověď vývoje akciových či jiných trhů. V IoT je možné predikci využít např. pro:

- Predikci vývoje spotřeby energií.
- Předpověď teploty, vlhkosti či jiných meteorologických veličin.
- Kombinovanou predikci na základě více časových řad.
    - Např. Předpověď vývoje dodávky elektrické energie z fotovoltaických panelů na základě historického průběhu teploty, slunečního svitu případně dalších veličin.

### 7.1.4 Klasifikace

Obecně klasifikace nachází uplatnění v různých oblastech, které generují dostatečné množství dat. Výjimkou není ani IoT. Klasifikace je jako úloha velice abstraktní a je ji možno využít opravdu v mnoha situacích. Řadí se do skupiny úloh, které vyžadují interakci lidí pro

správné natrénování modelu (tzv. učení s učitelem). Konkrétní případy pro využití klasifikace v IoT jsou např:

- Rozpoznání IoT zařízení v síti.
- Generování kombinovaných ovládacích prvků.
- Klasifikace bezpečnostních hrozeb v IoT sítích.

## 7.2 DBSCAN

**Density-based spatial clustering of applications with noise (DBSCAN)** je algoritmus z oblasti clusterové analýzy. Princip této oblasti byl vysvětlen v sekci 3.3.1 teoretické části práce. Algoritmus přiřazuje clustery na základě identifikace hustoty vzorků dat, což umožňuje detekovat clustery libovolného tvaru [32]. Algoritmus rozděluje vzorky dat neboli body v prostoru na jádrové, hraniční a šumové [32].

### 7.2.1 Algoritmus

Algoritmus umožňuje nastavení dvou parametrů.

- $\varepsilon$ – Poloměr sousedství
- μ – Minimální počet bodů v clusteru

Pro pochopení algoritmu je nutné znát následující pojmy.

**$\varepsilon$-sousedství** bodu $p$ označované jako $N_\varepsilon(p)$ je definováno vztahem (5). [32]

$$N_\varepsilon(p) = \{q \in D \mid \varphi(p,q) \leq \varepsilon\} \tag{5}$$

Kde $D$ označuje dataset a $\varphi(p,q)$ je vzdálenost mezi body $p$ a $q$.

**Přímá dosažitelnost –** bod $p$ je přímo dosažitelný z bodu $q$, pokud platí vztah (6) a podmínka jádrového bodu (7). [32]

$$p \in N_\varepsilon(q) \tag{6}$$

$$|N_\varepsilon(q)| \geq \mu \tag{7}$$

**Dosažitelnost** – bod $p$ je dosažitelný z bodu $q$, pokud existuje řetězec bodů $p_1, \ldots p_n$, $p_1 = q$, $p_n = p$ takových, že $p_{i+1}$ je přímo dosažitelný z bodu $p_i$. [32]

**Propojení –** bod $p$ je propojený s bodem $q$, pokud existuje bod o, ze kterého jsou oba body $p$ i $q$ dosažitelné. [32]

Následující pseudokód algoritmu byl převzat z [15].

```
1. Všechny body jsou na začátku algoritmu označeny jako
   NEKLASIFIKOVANÉ
2. Pro všechny body z datasetu
      a. Pokud je bod NEKLASIFIKOVÁN
           i. Rozšíření clusteru
                 1. Pokud bod splňuje jádrovou podmínku, všechny
                    dosažitelné body jsou označeny stejným
                    cluster identifikátorem
                       a. Pro každý dosažitelný bod test podmínky
                          jádra (7) a všechny dosažitelné body
                          jsou klasifikovány stejným cluster
                          identifikátorem
                       b. Opakování předchozího bodu, dokud už
                          nejsou žádné dosažitelné NEKLASIFIKOVANÉ
                          body
                 2. Pokud nesplňuje je klasifikován jako ŠUM
3. Konec
```

### 7.2.2 Využití pro detekci anomálií

Cílem detekce anomálií je objevit neobvyklé hodnoty. Ty mohou znamenat nežádoucí šum nebo právě hledané anomálie. Vždy záleží na konkrétních datech a konkrétní úloze, která se s nimi provádí či bude provádět.

Algoritmus DBSCAN **identifikuje šumové body** neboli body nepatřící do žádného clusteru, čímž je pro detekci anomálií maximálně vhodný.

Oproti nejznámějšímu algoritmu pro clusterovou analýzu K-Means je vhodnější pro detekci anomálií především z následujících důvodů:

- Nesnaží se každý bod přiřadit do clusteru.
- Předem nepožaduje určení počtu clusterů, které mají vzniknout.
    - V případě, že je cílem identifikovat šumové hodnoty jsou informace o výsledném umístění do clusteru redundantní. Zároveň by bylo také velice obtížné počet clusterů předem definovat.

### 7.2.3 Preprocessing

Algoritmus DBSCAN umí pracovat s vícedimenzionálními daty, přičemž hodnoty jedné dimenze mohou nabývat libovolné reálné číslo. IoT data, které by algoritmus měl zpracovávat se dají charakterizovat jako časové řady. Úkolem preprocessingu je v tomto případě převést časové řady do hodnot z oboru reálných čísel.

**Převedení času**

V případě, že je žádoucí, aby algoritmus pracoval i s časovou dimenzí je nutné převést časové hodnoty z formátu datum a čas do číselné oblasti. Nejjednodušším způsobem je převod na UNIX časové razítko.

**Normalizace**

Dalším nevyhnutelným krokem je normalizace dat. Po převodu data a času do časového razítka, se totiž téměř ve všech případech liší hodnoty a časová razítka o několik řádů. Při clusterové analýze není přípustné, aby se hodnoty jednotlivých dimenzí takto výrazně lišili.

Např. V případě hodnoty 27 °C získané dne 23.02.2022 v 22:34:02 je rozdíl o 8 řádů, jelikož časové razítko je 1645652042.

Pro normalizaci časových razítek byl zvolen způsob definovaný vztahem (8).

$$t_f = \frac{(t - t_0)}{C_t} \tag{8}$$

Kde $t$ je transformované časové razítko, $t_0$ je počáteční (nejmenší) časové razítko ze vstupního datasetu a $C_t$ je parametr určující výsledné rozložení. Byla použita hodnota $C_t = 86400$, která reprezentuje počet sekund v jednom dni (24 hodin). Časová razítka za jeden den se pak transformují do intervalu $< 0{,}1 >$. Předpokladem tohoto nastavení $C_t$ je, že jsou vzorky dat získávány několikrát za hodinu. Pokud by interval získávání vzorků byl jiný, je nutno upravit nastavení parametru $C_t$, případně je možné jej určovat vhodným způsobem dynamicky.

Pro normalizaci dimenze hodnot jsem použil min-max normalizaci do intervalu $< 0{,}1 >$ danou vztahem (9).

$$x' = \frac{(x - x_{min})}{(x_{max} - x_{min})} \tag{9}$$

Kde $x$ je transformovaná hodnota, $x_{min}$ je minimální hodnota dimenze v datasetu a $x_{max}$ je maximální hodnota dimenze v datasetu.

Pokud by časová řada obsahovala hodnoty z jiné, než číselné oblasti je nutné vhodným způsobem upravit postup preprocessingu tak, aby došlo k adekvátnímu převodu do oboru reálných čísel. V rámci této práce se bude detekce anomálií provádět pouze nad časovými řadami, které obsahují hodnoty z číselných oborů a tento převod tedy není nutný.

### 7.2.4 Nastavení počátečních parametrů

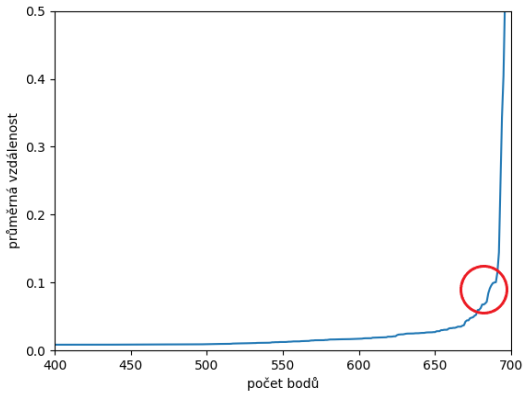
Vstupní parametry algoritmů je nutné odhadnout, případě pomocí experimentů adaptovat. Aby bylo možné tyto experimenty vyhodnotit, je zapotřebí vhodná metrika. Pro clusterovou analýzu se nejčastěji používá např. silluete score[11]. Zde si své místo nachází i optimalizace a evoluční algoritmy, což už však přesahuje hranice této práce. V tomto případě je nutné se zabývat spíše kvalitním počátečním odhadem.

**Nastavení parametru μ**

Obecně se využívá nastavení tohoto parametru podle doporučení, kterým je ho nastavit na dvojnásobek dimenze. Při implementaci bylo přistoupeno na toto doporučení.

**Nastavení parametru *ε***

Pro nastavení tohoto parametru se používá postup, kdy se vypočítá průměrná vzdálenost každého bodu a jeho μ nejbližších sousedů. Vypočítané hodnoty se seřadí vzestupně a vynesou do tzv. „elbow“ grafu. Optimální hodnota parametru *ε* se nachází na ose y v oblasti největšího zakřivení, kde se prudce zvedá hodnota průměrné vzdálenosti. Oblast zakřivení je znázorněná na obrázku (Obr. 19). Pokud nejsou data příliš zašuměná, měla by se tato oblast nacházet v nejvyšších hodnotách osy x a být snadno identifikovatelná.




Obr. 19 Elbow graf nesoucí informaci o optimálních hodnotách parametru *ε*.

[11] Siluete score je metrika používaná k výpočtu kvality shlukování. Výsledné hodnoty se pohybují na škále od -1 do 1.

V případě, že je cílem identifikovat anomálie s velkou odlišností od ostatních hodnot, může být výhodné nastavit parametr $\varepsilon$ na vyšší hodnotu, než se nachází oblast největšího zakřivení. Pro předzpracovaná IoT data, která byla normalizována dle předchozího popisu normalizace, se vyplatilo volit parametr $\varepsilon$ v rozmezí intervalu $< 0.15, 0.3 >$.

### 7.2.5 Možnosti použití algoritmu

Při integraci algoritmu DBSCAN bylo využito implementace z knihovny **Apache Commons Math** pro programovací jazyk Java. Tato knihovna obsahuje implementaci několika základních algoritmů z oblasti data miningu i strojového učení.

Výzkumem jsem dospěl k závěru, že je možné použít algoritmus DBSCAN pro detekci anomálií dvěma různými způsoby.

1. Zanedbání časové dimenze. Proces clusterizace bude probíhat pouze na dimenzi hodnot.
2. Využití i časové dimenze. Ta musí být vhodně předzpracována.

### 7.2.6 Použití pouze pro dimenzi hodnot

Jednodušším způsobem použití algoritmu DBSCAN pro detekci anomálií je zanedbat časovou dimenzi. V tomto případě samozřejmě není nutné časové hodnoty jakkoli předzpracovávat. Provede se pouze normalizace hodnot a následně se pomocí vhodně nastavených parametrů $\varepsilon$ a $\mu$ zavolá metoda z knihovny, která provede rozdělení hodnot do clusterů. Nakonec je nutné získat hodnoty, které algoritmus nezařadil do žádného clusteru. Postup implementace volání algoritmu DBSCAN integrovaný do systému pro zpracování historických hodnot je na následující ukázce.

```
public List<ElementLog<Double>> findAnomalies1D(List<ElementLog<Double>>
elementLogs) {
    if(elementLogs == null) throw new IllegalArgumentException("Element logs
cannot be null");
    // Value dimension
    var values =
elementLogs.stream().map(ElementLog::getValue).collect(Collectors.toList());
    // Preprocessing
    var preprocessedData = minMaxNormalization(values).stream().
            map(it -> new DoublePoint(new double[]{it}))
            .collect(Collectors.toList());
    // Clustering
    DBSCANClusterer<DoublePoint> alg = new DBSCANClusterer<>(eps, minPoints);
    List<Cluster<DoublePoint>> clusters = alg.cluster(preprocessedData);
    // Find non-clustered points
    var clusteredPoints = clusters.stream()
            .flatMap(it -> it.getPoints().stream())
            .collect(Collectors.toList());
    return preprocessedData.stream()
            .filter(it -> !clusteredPoints.contains(it))
            .map(it -> elementLogs.get(preprocessedData.indexOf(it)))
            .collect(Collectors.toList());
}
```

Zhodnocení dosažených výsledků se nachází v poslední kapitole.

### 7.2.7 Použití algoritmu pro 2D data

V druhém způsobu použití algoritmu DBSCAN je nutný preprocessing dimenze hodnot i dimenze času. Pro dimenzi hodnot se opět provede min-max normalizace. V časové dimenzi proběhne preprocessing, který zahrnuje transformaci objektů nesoucí datum a čas do oboru reálných čísel. Tato transformace je popsána v sekci 7.2.3.

Po preprocessingu dojde opět k zavolání metody pro clusterizaci a následnému vyhledání anomálních hodnot. Postup implementace volání algoritmu DBSCAN způsobem zahrnujícím časovou dimenzi včetně preprocessingu je v následující ukázce.

```
public List<ElementLog<Double>> findAnomalies2D(List<ElementLog<Double>>
elementLogs) {
    List<DoublePoint> preprocessedData = preprocessing(elementLogs);
    DBSCANClusterer<DoublePoint> alg = new DBSCANClusterer<>(eps, minPoints);
    List<Cluster<DoublePoint>> clusters = alg.cluster(preprocessedData);

    // Find non-clustered points
    var clusteredPoints = clusters.stream().flatMap(it ->
it.getPoints().stream()).collect(Collectors.toList());
    return preprocessedData.stream()
            .filter(it -> !clusteredPoints.contains(it))
            .map(it -> elementLogs.get(preprocessedData.indexOf(it)))
            .collect(Collectors.toList());
}

private List<DoublePoint> preprocessing(List<ElementLog<Double>> elementLogs)
{
    if(elementLogs == null) throw new IllegalArgumentException("Element logs
cannot be null");
    // Value dimension
    var values =
elementLogs.stream().map(ElementLog::getValue).collect(Collectors.toList());
    var normalizedValues = minMaxNormalization(values);
    // Time dimension
    long startSeconds = elementLogs.get(0).getTimestamp().toEpochSecond();
    var normalizedTime = elementLogs.stream()
            .map(it -> timeNormalize(it.getTimestamp(), startSeconds, Ct))
            .collect(Collectors.toList());

    return IntStream.range(0, Math.min(normalizedValues.size(),
normalizedTime.size()))
            .mapToObj(i -> new DoublePoint(new double[]{normalizedTime.get(i),
normalizedValues.get(i)}))
            .collect(Collectors.toList());
}
```

Zhodnocení výsledků proběhne opět v poslední kapitole.

## 7.3 ARIMA

Autoregressive Integrated Moving Average (ARIMA) není algoritmem, ale statistickým modelem. Používá se obecně pro analýzu časových řad a jejich predikci. Jedná se o rozšíření modelu ARMA o diferenci časových řad, aby bylo dosaženo jejich stacionarity. [33]

Stacionární časová řada se vyznačuje tak, že se v čase nemění její statistické vlastnosti. V případě IoT hodnot je již většina časových řad stacionárních a není tedy třeba provádět diferenci. Najdou se však i některé časové řady u kterých bude nutné diferenci provést, a tak je lepší, aby na to byl model připraven.

Model se skládá z několika dílčích částí, které jsou jinak samostatnými modely.

### Autoregresní (AR)

Autoregresní model je model, který obsahuje prediktory založené na předchozích hodnotách. Je matematicky popsán vztahem (10). [34]

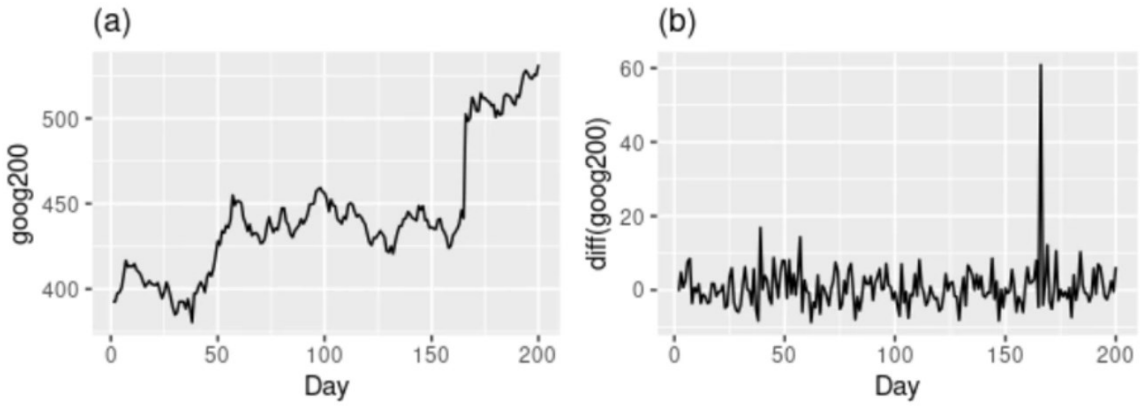
$$y_t = \alpha + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \cdots + \beta_p y_{t-p} + \varepsilon \tag{10}$$

Kde $y_{t-1}$ je první předchozí hodnota řady, $\beta_1$ je koeficient první předchozí hodnoty určený modelem, $\alpha$ je konstanta, $\varepsilon$ je hodnota tzv. bílého šumu. Řád modelu se označuje $p$ a označuje počet předchozích hodnot zpracovávaných modelem.

### Integrační (I)

Smyslem této části je z nestacionární časové řady udělat stacionární pomocí diference hodnot z předchozími hodnotami. Řád této části se označuje $d$ a určuje počet prováděných diferencí. Diference prvního řádu ($d = 1$) je dána vztahem (11). Výsledek diference časové řady je možné vidět na obrázku (Obr. 20).

$$y_t{}' = y_t - y_{t-1} \tag{11}$$




Obr. 20 Porovnání časové řady před a po diferenci. [33]

### Klouzavý průměr (MA)

Model klouzavého průměru používá regresi na chyby předchozích predikcí. Je popsán vztahem (12). [34]

$$y_t = \alpha + e_t + \phi_1 e_{t-1} + \phi_2 e_{t-2} + \cdots + \phi_q e_{t-q} \tag{12}$$

Kde $\alpha$ je konstanta, $e_t$ je tzv. chyba bílého šumu a $\phi_1$ je koeficient první předchozí hodnoty určený modelem. Řád tohoto modelu se označuje $q$ a podobně jako u autoregresního modelu určuje počet předchozích hodnot, které model zpracovává.

### 7.3.1 Preprocessing

Model ARIMA zpracovává posloupnost hodnot v oboru reálných čísel. Dimenze času tedy v tomto případě není využita. Nicméně se předpokládá a pro správné fungování modelu je žádoucí, aby časový interval mezi jednotlivými vzorky byl co nejvíce konstantní. Pokud nejsou intervaly dostatečně konstantní je nutné preprocessingem této konstantnosti docílit. Data, která se budou v této práci modelem ARIMA zpracovávat jsou dostatečně konstantní a preprocessing tedy není nutný.

Normalizace dimenze hodnot také není v tomto případě nutná, jelikož vstupní parametry nejsou závislé na rozsahu zpracovávaných hodnot.

### 7.3.2 Nastavení počátečních parametrů

Základní varianta modelu ARIMA předpokládá nastavení parametrů $p$, $d$ a $q$.

Počáteční odhad parametrů $p$ a $q$ lze provést pomocí funkcí ACF (Autocorrelation Function) a PACF  Partial Autocorrelation Function) [35]. Další možností je průzkumnou analýzou vyzkoušet různé kombinace vstupních parametrů a zhodnotit kvalitu dodaného řešení.

Při implementaci modelu ARIMA v rámci této práce bylo využito obou zmíněných postupů.

### 7.3.3 Možnosti použití modelu

Primární úlohou modelu ARIMA je predikce budoucích hodnot časové řady. Z informací získaných pomocí predikce je možné identifikovat anomální hodnoty. Model umožňuje pokrytí obou úloh.

**Detekce anomálií**

Hlavním důvodem použití modelu ARIMA pro detekci anomálií je možnost detekce anomálních hodnot u příchozích vzorků **v reálném čase**. Nejedná se o jediný způsob použití a napříč různými literaturami lze najít mnoho jiných způsobů. Nicméně pro detekci anomálií nad kompletním datasetem byl již integrován algoritmus DBSCAN, který dosahuje pro většinu případů dostačující výsledky.

**Predikce**

Implementace modelu ARIMA disponují rozhraním, které umožňuje získat predikaci několika budoucích hodnot časové řady na základě již existujících hodnot. Logika potřebná k využití modelu pro samotnou predikci je tedy minimální.

### 7.3.4 Detekce anomálií v reálném čase

Zvolil jsem způsob predikovat pomocí modelu ARIMA $x$ hodnot časové řady. Predikované hodnoty se uloží do databáze včetně jejich dolních a horních mezí. Tyto meze jsou stanoveny tzv. confidence intervalem.

Confidence interval pro predikci je interval v oboru reálných čísel, který je určen parametrem, jež definuje kolika procentní je šance, že další budoucí reálné hodnoty budou patřit do tohoto intervalu. Např. 95 % interval znamená, že je 5 % šance na to, aby budoucí reálné hodnoty byly mimo tento interval. Těchto 5 % je možno prohlásit za hodnoty, jež mohou být považovány za anomální.

Při implementaci bylo využito **95 % confidence intervalu**. Jedná se o defaultní hodnotu, kterou standardně používá většina implementací modelu. Tento parametr je možno postupem času adaptovat podle průběžných výsledků.

Proces je nastaven na **predikci 5 budoucích hodnot** časové řady ($x = 5$). S pomocí experimentů bylo zjištěno, že menší počet predikcí zbytečně zvyšuje režii detekce, jelikož je nutné častěji provádět vyhodnocení modelu. S každou další predikcí se však snižuje kvalita dané predikce. Hodnota $x = 5$ je tedy vhodným kompromisem.

Zvolený způsob jsem implementoval do systému pro uchování historických hodnot. Proces „trénování“ modelu a uložení predikovaných hodnot je možné vidět na následující ukázce. Byla využita implementace modelu ARIMA z knihovny Timeseries-Forecast[12].

```
public void train(UUID elementId, List<ElementLog<Double>> elementLogs) {
    double[] values = elementLogs.stream().map(ElementLog::getValue)
            .mapToDouble(x -> x).toArray();
    ArimaParams arimaParams = new ArimaParams(p, d, q, 0, 0, 0, 0);
    ForecastResult forecastResult = Arima.forecast_arima(
            values, 5, arimaParams
    );
    anomalyConfidenceRepository.removeByElementId(elementId);
    for (int i = 0; i < forecastResult.getForecast().length; i++) {
        double prediction = forecastResult.getForecast()[i];
        double lower = forecastResult.getForecastLowerConf()[i];
        double upper = forecastResult.getForecastUpperConf()[i];
        anomalyConfidenceRepository.save(new AnomalyConfidence(
                null,
                ZonedDateTime.now(),
                prediction,
                upper,
                lower,
                elementId)
        );
    }
}
```

Pokud je nová hodnota, která se ukládá do časové řady, mimo rozsah daný confidence intervalem je prohlášena za anomální. Predikovaná hodnota se odstraní a dochází k uložení nové hodnoty.

Systém využívající tento způsob detekce musí obsahovat logiku, kterou zajistí existenci predikcí, aby bylo možno kontrolovat rozsah confidence intervalu.

### 7.3.5 Predikce budoucích hodnot

Predikce, jako operace, již probíhá při detekci anomálií. Při implementaci se tak jednalo o oddělení části obsluhující predikci do samostatné metody.

Musí se stanovit parametr *windowSize*. Tento parametr určuje maximální počet hodnot, který model bude zpracovávat. Jedná se o prevenci proti příliš pomalému vyhodnocení modelu.

Následující ukázka kódu obsahuje metodu pro predikci budoucích hodnot.

[12] https://github.com/Workday/timeseries-forecast

```
public ForecastResult forecast(UUID elementId, int forecastSize, int
windowSize) {
    var elementLogs = elementLogRepository.findAllByMetadataElementIdAndType(
            Pageable.ofSize(windowSize),
            elementId,
            ElementType.BIG_DECIMAL
    );
    if (elementLogs.size() < 20) {
        throw new RuntimeException(String.format("Not enough BIG_DECIMAL data
points [size: %s]", elementLogs.size()));
    }
    var dataset = elementLogs.stream()
            .map(it -> (ElementLog<Double>) it)
            .collect(Collectors.toList());
    double[] values = dataset.stream()
            .map(ElementLog::getValue)
            .mapToDouble(x -> x).toArray();
    ArimaParams arimaParams = new ArimaParams(p, d, q, 0, 0, 0, 0);
    return Arima.forecast_arima(
            values, forecastSize, arimaParams
    );
}
```

## 7.4 Apriori

V této sekci bude popsán postup implementace rozpoznání častých kombinací událostí pomocí algoritmu Apriori. Událost je v tomto případě změna hodnoty IoT zařízení. Cílem je získat informace o častých kombinacích, které mohou být uživateli IoT platformy přínosně. Např. mohou pomoci identifkovat kombinace IoT prvků pro tvorbu automatických scénářů.

### 7.4.1 Algoritmus

Algoritmus Apriori se používá pro hledání častých množin položek a navrhování asociačních pravidel z databáze transakcí. Je založen na konceptu, že podmnožina časté množiny položek musí být také častou množinou položek. Algoritmus pracuje v iteracích. S každou další iterací se navyšuje délka množiny. [36]

Základní varianta umožňuje nastavit 2 parametry:

- **Minimal Support** – „Support“ měří, kolikrát se určitá se množina položek vyskytuje v datové sadě [36]. „Minimal Support“ pak určuje prahovou hodnotu a udává se v procentech. Pokud je počet výskytů množiny položek menší než „Minimal Support“ algoritmus jej nezahrne do další iterace.

- **Minimal Confidence** – „Confidence“ měří, jaká je pravděpodobnost výskytu určité množiny položek [36]. „Minimal Confidence“ značí opět prahovou hodnotu a udává se v procentech. Pokud je pravděpodobnost výskytu množiny položek menší než práh, algoritmus jej nezahrne do další iterace.

### 7.4.2 Způsob rozpoznání

Aby byly časté kombinace událostí pro uživatele snadno spojitelné s reálnými situacemi, bude probíhat v jednu chvíli rozpoznávání událostí, které nastaly v rámci jedné hodiny. Díky tomu také uživatel získá informaci o přibližné době, kdy byly nalezené kombinace událostí vykonány.

Důležitým parametrem postupu je celkový **časový interval**, ve kterém bude probíhat hledání. Měl by být dostatečně velký na to, aby vynikly časté kombinace událostí. Příliš velký interval by však mohl značně zpomalit běh algoritmu. Obvykle bylo využito intervalu s délkou **jeden měsíc**.

### 7.4.3 Preprocessing

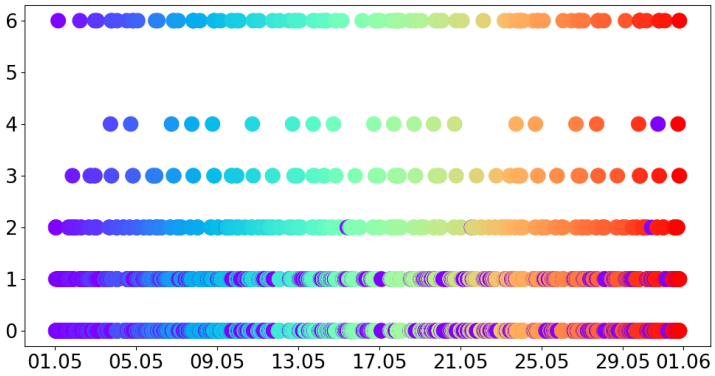
Bude nutné transformovat **události** do transakcí, které vstupují do algoritmu Apriori. Událostí je v tomto případě záznam o hodnotě IoT prvku v určitý čas. Většina IoT prvků s možností zápisu odesílá informace o hodnotě pouze ve chvíli jejich změny. Pokud by některý prvek posílal informaci o své hodnotě i v případě, že se nezměnila, bylo by nutné odfiltrovat záznamy s nezměněnou hodnotou.

#### Identifikace oblastí

Ve zvoleném intervalu se identifikují oblasti, ve kterých nastávají kombinace událostí. Tento krok značně ulehčí pozdější transformaci na transakce, které vstupují do algoritmu Apriori. Pro identifikaci těchto „hustých“ oblastí na časové ose byl použit algoritmus DBSCAN (viz. 7.2), jehož výsledek je viditelný na obrázku (Obr. 21). Důvodem použití algoritmu DBSCAN je možnost nastavení parametrů $\varepsilon$ a $\mu$. Dále také klasifikace šumových bodů. Události, které byly algoritmem klasifikovány jako šumové jsou vyloučeny. Tímto krokem se výrazně sníží režie při dalších výpočtech.

Parametry algoritmu byly určeny experimentováním. Jsou závislé na počtu IoT prvků v projektu a frekvenci změn jejich hodnot.

DBSCAN provádí clusterizaci pouze v časové dimenzi. Výstupem by měly být časové clustery, ve kterých se nachází „hustý“ sled událostí.




Obr. 21 Indentifikované oblasti algoritmem DBSCAN.

Z obrázku je patrné, že bylo identifikováno několik clusterů v rámci jednoho dne. Je třeba tyto clustery dále roztřídit.

**Roztřídění dle hodiny**

U každého získaného clusteru se vypočítá průměr data a času události. Následně dojde k roztřídění clusterů do skupin podle hodiny průměrného data a času v clusteru. Výsledkem by pak mělo být maximálně 24 skupin clusterů.

```
group = avg_date(cluster_datetimes).hour
```

V tomto kroku je rovněž žádoucí odstranit clustery, které mají příliš velký počet událostí. Takové clustery jsou výsledkem např. restartu systému pro automatizaci a mohly by výrazně zpomalit běh algoritmu Apriori. Nejjednodušším způsobem je si zvolit **práh**, který je však opět závislý na počtu IoT prvků v projektu a frekvenci změn jejich hodnot. Pokud dojde k překročení tohoto prahu cluster se zahodí.

**Transformace do transakcí**

Posledním krokem je transformace událostí do transakcí, které zpracovává algoritmus Apriori. Cluster událostí, kde událost je trojice (hodnota, datum a čas, ID prvku), se transformuje na cluster obsahující pouze ID prvků. Jak probíhá tato transformace je na následující ukázce.

```
clusters[(i, avg)] = [
    input_dataset[index]['elementIds'] for index in cluster
]
```

### 7.4.4 Nastavení počátečních parametrů

Určení počátečních parametrů Minimal Support a Minimal Confidence probíhalo pomocí experimentů. Parametry omezují kombinace událostí, které algoritmus zahrnuje do dalších iterací. Je třeba parametry vhodně nastavit takovým způsobem, aby výsledky byly dostatečně podrobné, ale zároveň nedocházelo k přílišnému nárůstu doby běhu.

### 7.4.5 Rozpoznání častých kombinací událostí

Připravený seznam transakcí roztříděný do skupin dle průměrné hodiny je vstupem do procesu rozpoznávání. Proces sestává z iterací, kde každá iterace je jedno volání algoritmu Apriori nad jednou skupinou, která sdružuje transakce událostí přibližně v dané hodině. Některé události mohou v důsledku zařazení podle průměrné hodiny v clusteru spadnout do jiné hodiny, než je skutečná hodina jejich vykonání. Je tedy třeba počítat s mírnou nepřesností.

Proces volání algoritmu Apriori spolu výpisem nejčastějších kombinací událostí je implementovaný v následující ukázce v programovacím jazyce Python.

```
min_support = 0.15
min_confidence = 0.1
for hour, transactions in sorted(group_clusters.items()):
    print("\n --- Hour: " + str(hour) + ":00 ---\n")
    itemsets, rules = apriori(
        transactions,
        min_support=min_support,
        min_confidence=min_confidence
    )

    for itemset_length, itemset_group in itemsets.items():
        print("--" + str(itemset_length) + "--")
        itemset_g_s = sorted(itemset_group.items(), key=lambda it: it[1])
        for itemset, count in itemset_g_s:
            print(str(itemset) + " - " + str(count))
```

### 7.4.6 Možné budoucí vylepšení

V případě, že celkový prohledávaný časový interval, zahrnuje několik měsíců. Mohlo by být přínosné vhodně interpretovat **sezónní charakter** některých častých kombinací. Např. automatizace událostí spojených s vytápěním je doména především zimních měsíců, zatímco akce s událostmi spojenými se stíněním terasy zase v letních měsících.

Asymptotická časová a prostorová složitost algoritmu Apriori je $O(2^D)$, kde $D$ je počet unikátních položek ve všech transakcích. Algoritmus je celkově **pomalý**, avšak existuje několik vylepšení, které jeho běh zrychlují. Díky zrychlení algoritmu by bylo možné zvětšovat

prohledávaný interval při zachování stejné doby běhu, což může mít pozitivní výsledek na rozpoznání některých frekventovaných položek.

# 8 INTERPRETACE VÝSLEDKŮ

Poslední kapitola obsahuje interpretaci výsledků, které byly dosaženy aplikací algoritmů a postupů z oblasti data miningu popsaných v předchozí kapitole. Dále kapitola popisuje průběh testování implementovaných postupů.

## 8.1 Detekce anomálií

Pro úlohu detekce anomálií byly zvoleny dva různé postupy implementace. Každý z postupů využívá pro detekci jiný algoritmus. Postupy se odlišují ve způsobu detekce a možnosti jejich integrace do systému pro zpracovávání historických hodnot z IoT.

### 8.1.1 Zpětná detekce algoritmem DBSCAN

Algoritmus DBSCAN byl implementován do systému, aby detekoval anomálie nad časovými řadami za určité období. Není vhodný pro detekci anomálií u nově zapisovaných hodnot, jelikož by bylo nutné při každé nové hodnotě provést clusterovou analýzu s velikostí časového okna minimálně jeden den, aby byly detekované anomálie relevantní.

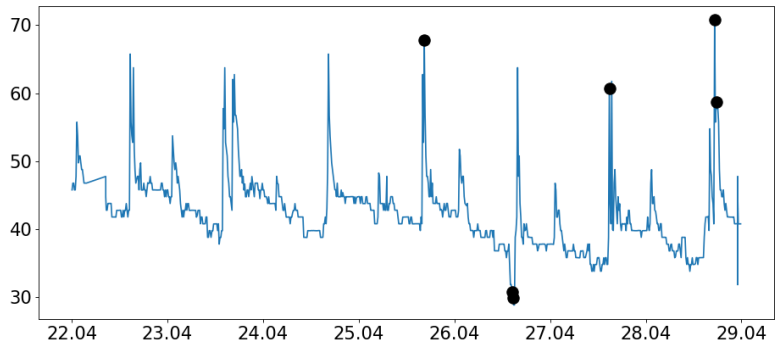
Testovací dataset obsahuje data z několika IoT prvků, pro které může dávat smysl získat informace o anomáliích v běžném rodinném domě. Pro interpretaci dosažených výsledků byly vybrány 3 IoT prvky:

- Senzor vlhkosti vzduchu v koupelně
- Senzor aktuálního příkonu rekuperační jednotky
- Senzor aktuálního výkonu předehřevu

Detekce anomálií bude probíhat na hodnotách těchto časových řad omezených intervalem od 22.04.2021 00:00 do 29.04.2021 00:00. Detekované anomálie jsou znázorněny černým bodem v grafu časové řady. Každému grafu předchází informace, o který IoT prvek se jedná a zvolené parametry algoritmu.

### 8.1.2 DBSCAN bez časové dimenze

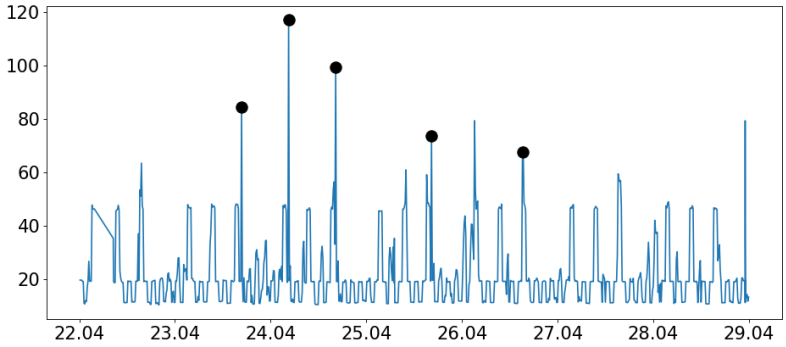
**Senzor vlhkosti vzduchu** ($\varepsilon = 0.02, \mu = 4$) na obrázku (Obr. 22).




Obr. 22 Anomálie pro senzor vlhkosti vzduchu v koupelně. DBSCAN bez časové dimenze.

Z obrázku (Obr. 22) je patrné, že nebyly detekovány všechny hodnoty, které lze považovat za anomální. Problém v tomto případě spočívá ve velikosti časového okna. Hodnoty vlhkosti v denních špičkách jsou velice podobné a algoritmus tyto hodnoty spojuje do clusteru.

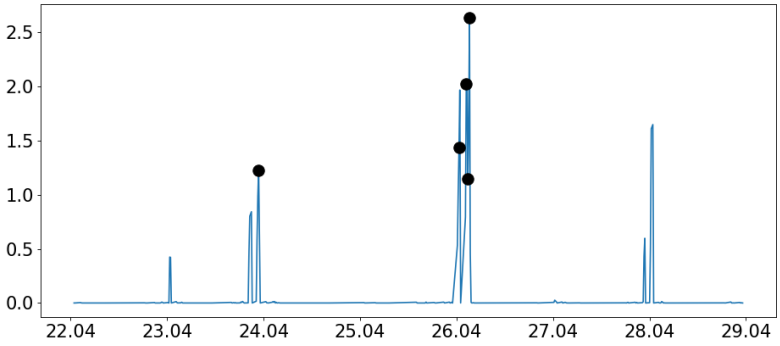
**Senzor aktuálního příkonu rekuperační jednotky** ($\varepsilon = 0.02, \mu = 4$) na obrázku (Obr. 23).




Obr. 23 Anomálie pro senzor aktuálního příkonu rekuperační jednotky. DBSCAN bez časové dimenze.

V tomto případě proběhla detekce nejodlišnějších anomálií korektně. To proto, že se detekované anomálie liší ve svých hodnotách a nebyly tak algoritmem zahrnuty do clusteru.

**Senzor aktuálního výkonu předehřevu** ($\varepsilon = 0.02, \mu = 4$) na obrázku (Obr. 24).

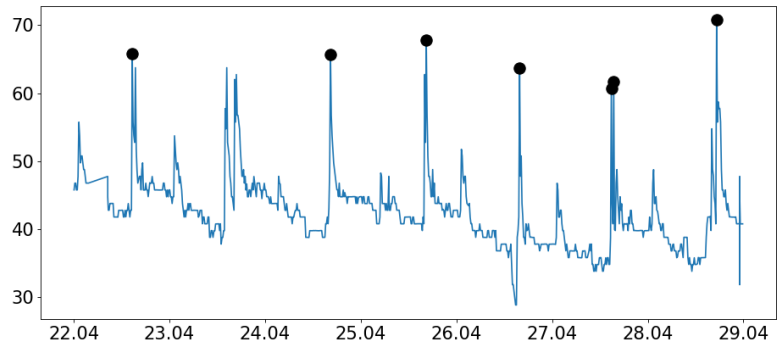



Obr. 24 Anomálie pro senzor aktuálního výkonu předehřevu vody. DBSCAN bez časové dimenze.

Opět se v tomto případě vyskytl problém s detekcí některých anomálií. Příčinou jsou, stejně jako při detekci u senzoru vlhkosti vzduchu, podobné hodnoty anomálií.

### 8.1.3 DBSCAN s časovou dimenzí

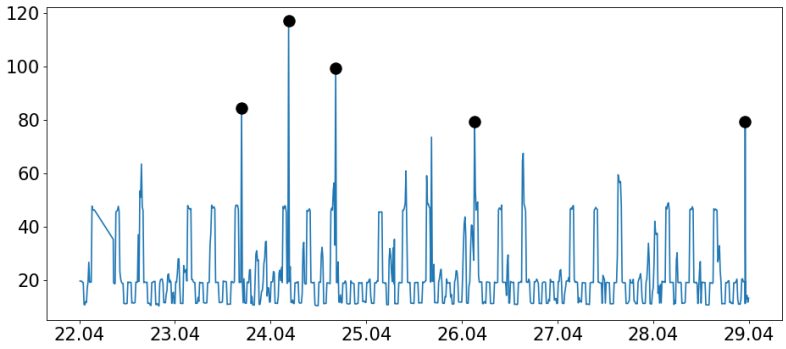
**Senzor vlhkosti vzduchu** ($\varepsilon = 0.2, \mu = 4$) na obrázku (Obr. 25).




Obr. 25 Anomálie pro senzor vlhkosti vzduchu v koupelně. DBSCAN se zahrnutím časové dimenze.

V tomto případě detekce s časovou dimenzí přesněji odhalila anomální hodnoty, než tomu bylo při detekci bez časové dimenze. Důvodem je, že detekce s časovou dimenzí není závislá na velikosti zpracovávaného okna.

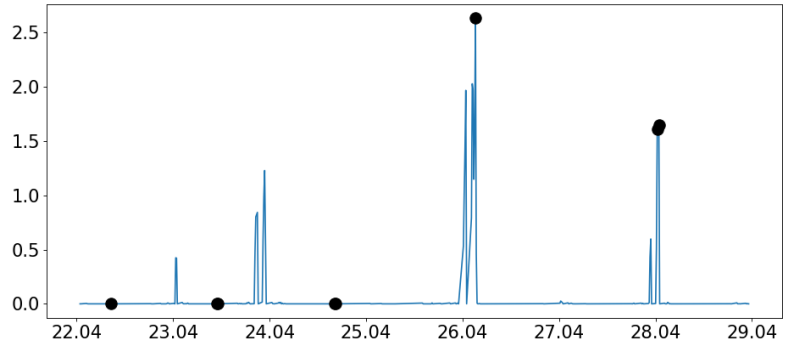
**Senzor aktuálního příkonu rekuperační jednotky** ($\varepsilon = 0.2, \mu = 4$) na obrázku (Obr. 26).




Obr. 26 Anomálie pro senzor aktuálního příkonu rekuperační jednotky. DBSCAN se zahrnutím časové dimenze.

Pro vybranou časovou řadu hodnot aktuálního příkonu rekuperační jednotky byly detekovány korektně podobné anomálie u obou způsobů detekce.

**Senzor aktuálního výkonu předehřevu** ($\varepsilon = 0.2, \mu = 4$) na obrázku (Obr. 27).




Obr. 27 Anomálie pro senzor aktuálního výkonu předehřevu vody. DBSCAN se zahrnutím časové dimenze.

Z obrázku (Obr. 27) jsou patrné chybně detekované anomálie přímo v oblasti nejběžnějších hodnot. Problém je v tomto případě v nekonstantním časovém intervalu mezi jednotlivými vzorky. Detekce anomálií algoritmem DBSCAN se zahrnutím časové dimenze je citlivá na tuto nekonstantnost. Řešením by v tomto případě bylo vhodně doplnit chybějící hodnoty při preprocessingu.

### 8.1.4 Detekce v reálném čase modelem ARIMA

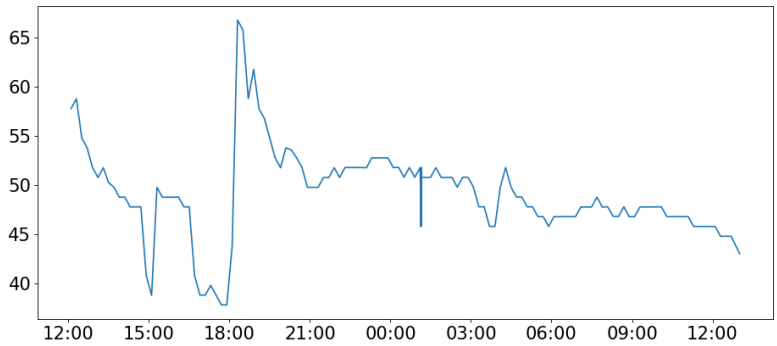
Statistický model ARIMA byl implementován do systému pro uchování historických hodnot způsobem umožňujícím detekci anomálií zapisovaných hodnot v reálném čase. Systém disponuje webovou metodou pro uložení hodnoty (viz. sekce **Chyba! Nenalezen zdroj odkazů.Chyba! Nenalezen zdroj odkazů.**). Při volání této metody se spustí proces, který klasifikuje hodnotu jako anomální na základě predikovaných hodnot.

Pro interpretaci výsledků bylo opět využito dat ze vzorového domu, a to konkrétně z následujících IoT prvků:

- Senzor vlhkosti vzduchu v koupelně
- Senzor aktuálního příkonu rekuperační jednotky

Pro dosažení všech výsledků zde uvedených byl použit model ARIMA(2,0,2), který predikoval 5 budoucích hodnot na základě 100 předchozích. Hodnota je prohlášena za anomální, pokud se nachází mimo 95 % confidence interval dané predikce.

Následující graf (Obr. 28) zobrazuje hodnoty **senzoru vlhkosti vzduchu** za posledních 24 hodin. Tabulka (Tabulka 9) obsahuje informace, zda byly následně zapsané hodnoty prohlášeny za anomální.



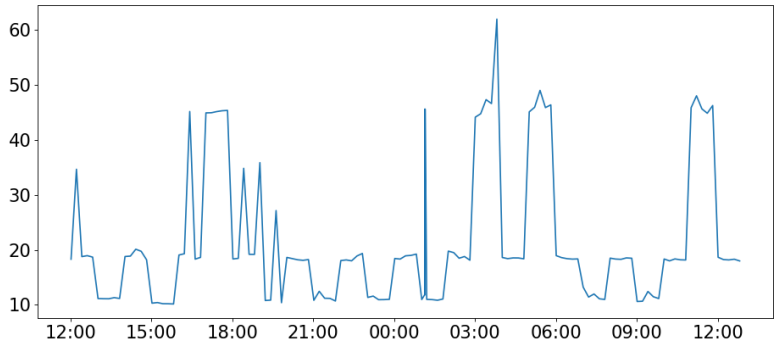

Obr. 28 Hodnoty senzoru vlhkosti vzduchu.

Tabulka 9 Nové hodnoty vlhkosti vzduchu.

| Pořadí | Datum a čas | Hodnota | Anomálie |
|---|---|---|---|
| 1 | 13:00:38 | 43 | ANO |
| 2 | 13:15:38 | 47 | NE |
| 3 | 13:30:38 | 50 | NE |
| 4 | 13:45:38 | 60 | ANO |

Z předchozí tabulky je patrné, že i malé odchýlení může znamenat anomálii. Pokud je citlivost detekce moc vysoká je nutné zvětšit confidence interval (např. z 95 % na 98 %).

Následující graf (Obr. 29) zobrazuje hodnoty **senzoru aktuálního příkonu rekuperační jednotky** za posledních 24 hodin. Tabulka (Tabulka 10) obsahuje informace, zda byly následně zapsané hodnoty prohlášeny za anomální.




Obr. 29 Hodnoty senzoru aktuálního příkonu rekuperační jednotky.

Tabulka 10 Nové hodnoty příkonu rekuperační jednotky.

| Pořadí | Datum a čas | Hodnota | Anomálie |
|---|---|---|---|
| 1 | 13:00:38 | 10 | NE |
| 2 | 13:15:38 | 20 | NE |
| 3 | 13:30:38 | 30 | NE |

| | | | |
|---|---|---|---|
| 4 | 13:45:38 | 42 | ANO |
| 5 | 14:00:38 | 0 | ANO |

V tomto případě je 95 % confidence interval predikcí mnohem širší. Tudíž je detekce méně citlivá než v předchozím případě. To je způsobeno především vyšší kolísavostí hodnot.

## 8.2 Rozpoznání častých kombinací událostí

Cílem úlohy je odhalit časté kombinace událostí a identifikovat, tak kroky pro možnou automatizaci. Postup popsaný v sekci 7.4 byl implementován v programovacím jazyce Python a otestován na testovacím datasetu ze vzorového domu. Programovací jazyk Python byl v tomto případě použit pro snadnější manipulaci a lepší dostupnost knihoven. V případě integrace do systému pro uchování historických hodnot by proběhla reimplementace tohoto postupu do programovacího jazyka Java.

Parametry, které nejsou stanoveny v popisu postupu byly zvoleny následovně.

- Celkový interval prohledávání – 1 měsíc
- Minimal Support – 15 %
- Minimal Confidence – 10 %
- Prahová hodnota pro příliš velký cluster – 100
- $\varepsilon$ – 120 (Datum a čas byl převeden do časového razítka v sekundách)
- $\mu$ – 30

Z testovacího datasetu byly vybrány IoT prvky, které disponují možností zápisu. U senzorů a podobných zařízení, které umožňují pouze číst hodnoty, nemá smysl sledovat události. Pokud by došlo k jejich zahrnutí do postupu mohlo by dojít ke zkreslení výsledků.

Vybrané IoT prvky nemají mechanismus cyklického vyčítání hodnoty a údaj o hodnotě posílají pouze ve chvíli její změny. Není tak třeba provádět preprocessing, který by z časových řad vybral pouze události se změnou hodnoty.

Pro porovnání jsou v této práci jsou interpretovány výsledky rozpoznávání za měsíc duben a říjen. Byly vybrány hodiny, u kterých se nalezly zajímavé kombinace událostí. Pro kombinace hodnot v rámci jedné hodiny je vždy samostatná tabulka obsahující počet detekcí dané kombinace a seznam IoT prvků, které tuto kombinaci tvoří.

### 8.2.1 Časté kombinace událostí za duben 2021

Rozpoznané kombinace událostí cca od **05:00 do 06:00** (Tabulka 11).

Tabulka 11 Časté kombinace událostí za duben 2021od 05:00 do 06:00.

| Počet | Rozpoznané kombinace událostí |
|---|---|
| 14 | Ambientní osvětlení, Ambientní osvětlení – Zádveří, Hlavní osvětlení – Koupelna |
| 12 | Ambientní osvětlení, Kuchyňská linka, Hlavní osvětlení – Koupelna |
| 11 | Ambientní osvětlení, Kuchyňská linka, Šatna, Zabezpečení domu |

V tabulce (Tabulka 11) lze vidět, že uživatelé v tomto časovém horizontu obvykle opakují akce, které jsou spojené s osvětlením šatny, koupelny nebo změnou zabezpečení domu.

Rozpoznané kombinace událostí cca od **20:00 do 21:00** (Tabulka 12).

Tabulka 12 Časté kombinace událostí za duben 2021 od 20:00 do 21:00.

| Počet | Rozpoznané kombinace událostí |
|---|---|
| 29 | Okno v pracovně, Okno v pokoji 2, Okno v jídelně, Portálové okno, Okno v kuchyni |
| 20 | Ambientní osvětlení, Sprcha a záchod, Šatna |
| 10 | Komora, Šatna, Hlavní osvětlení – Koupelna |

Z tabulky (Tabulka 12) je patrný pohyb všech venkovních žaluzií v domě. Žaluzie jsou napojeny na fotorezistor, který je automaticky zatáhne, pokud dojde ke snížení intenzity dopadajícího světla. V tomto případě uživatel neocení návrh na vytvoření automatického scénáře, jelikož scénář již existuje. Platforma by tak měla mít informace o těchto existujících automatizacích a uživateli jej nenabízet. Dále byly v tuto hodinu rozpoznány časté akce spojené s večerní hygienou.

Rozpoznané kombinace událostí cca od **23:00 do 00:00** (Tabulka 13).

Tabulka 13 Časté kombinace událostí za duben 2021 od 23:00 do 0:00.

| Počet | Rozpoznané kombinace událostí |
|---|---|
| 29 | Kuchyňská linka, Zásuvky u čerpadla, Zabezpečení domu |
| 5 | Retenční nádrž, Kuchyňská linka, Zásuvky u čerpadla, Zabezpečení domu, Zásuvky v záhoně |

V tabulce (Tabulka 13) lze vidět, že se v tomto časovém horizontu nejčastěji vyskytují akce spojené se zabezpečením domu či vypnutím venkovních zásuvek.

### 8.2.2 Časté kombinace událostí za říjen 2021

Rozpoznané kombinace událostí cca od **05:00 do 06:00** (Tabulka 14).

Tabulka 14 Časté kombinace událostí za říjen 2021od 05:00 do 06:00.

| Počet | Rozpoznané kombinace událostí |
|---|---|
| 12 | Ambientní osvětlení, Zrcadlo – Koupelna, Hlavní osvětlení – Koupelna |
| 10 | Hlavní osvětlení – Dětský pokoj, Komora, Šatna |
| 8 | Komora, Šatna, Zabezpečení domu |
| 7 | Ambientní osvětlení, Kuchyňská linka, Sprcha a záchod, Zrcadlo – Koupelna, Hlavní osvětlení – Koupelna |

Rozpoznané kombinace událostí mají stejný charakter jako v dubnu ve stejnou hodinu. Nicméně jednotlivé kombinace se lehce odlišují. To může být způsobeno tím, že uživatelé mohli za půl roku pozměnit sekvence jednotlivých akcí.

Rozpoznané kombinace událostí cca od **18:00 do 19:00** (Tabulka 15).

Tabulka 15 Časté kombinace událostí za duben 2021 od 18:00 do 19:00.

| Počet | Rozpoznané kombinace událostí |
|---|---|
| 22 | Okno v pokoji 2, Trvalkové záhony, Okno v jídelně, Portálové okno, Okno v kuchyni, Chodník |
| 13 | Boční podbití, Závětří, Hlavní osvětlení – Technická místnost |

Záměrně tabulka (Tabulka 15) obsahuje výsledky v čase od 18 do 19 hodin, zatímco pro duben byly tabulkou interpretovány podobné výsledky pro čas od 20 do 21. V obou případech totiž byly častokrát rozpoznány události, které se vztahují k době kolem západu slunce.

Rozpoznané kombinace událostí cca od **22:00 do 23:00** (Tabulka 16).

Tabulka 16 Časté kombinace událostí za říjen 2021 od 22:00 do 23:00.

| **Počet** | **Rozpoznané kombinace událostí** |
|---|---|
| 13 | Ambientní osvětlení, Zrcadlo – Koupelna, Šatna |
| 9 | Kuchyňská linka, Trvalkové záhony, Zabezpečení domu, Chodník |
| 8 | Trvalkové záhony, Šatna, Zabezpečení domu, Chodník |

Zatímco v dubnu byly rozpoznány akce spojené se zabezpečením domu a vypnutím venkovních zásuvek mezi 23 hodinou a půlnocí, v říjnu byly některé tyto akce rozpoznány již mezi 22 a 23 hodinou.

Rozpoznané kombinace událostí cca od **23:00 do 00:00** (Tabulka 17).

Tabulka 17 Časté kombinace událostí za říjen 2021 od 23:00 do 00:00.

| **Počet** | **Rozpoznané kombinace událostí** |
|---|---|
| 5 | Kuchyňská linka, Šatna, Zabezpečení domu |
| 4 | Ambientní osvětlení, Zrcadlo – Koupelna, Šatna |

Ne všechny akce spojené se zabezpečením domu byly rozpoznány již mezi 22 a 23 hodinou. Z tabulky (Tabulka 17) je patrné, že některé z těchto akcí byly rozpoznány i mezi 23 hodinou a půlnocí stejně jako v dubnu.

## 8.3 Predikce

Predikce jako operace probíhá již při detekci anomálií pomocí modelu ARIMA. Nebyla však samostatně interpretována jako úloha.

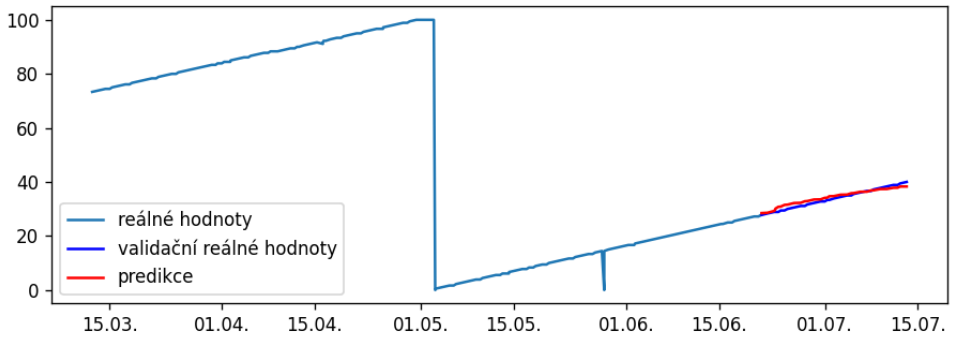
Pro interpretaci výsledků bude opět využito testovacího datasetu ze vzorového domu. Konkrétně zde bude demonstrována predikce pro časové řady následujících IoT prvků:

- Senzor opotřebení filtru rekuperační jednotky
- Senzor aktuálního příkonu rekuperační jednotky

Aby bylo možné porovnat predikci se skutečnými hodnotami je vstupní dataset rozdělen na 2 části.

1. Vstupní část – vstupuje do modelu. Podle ní se vypočítává predikce.
2. Validační část – bude sloužit pro validaci predikovaných hodnot.

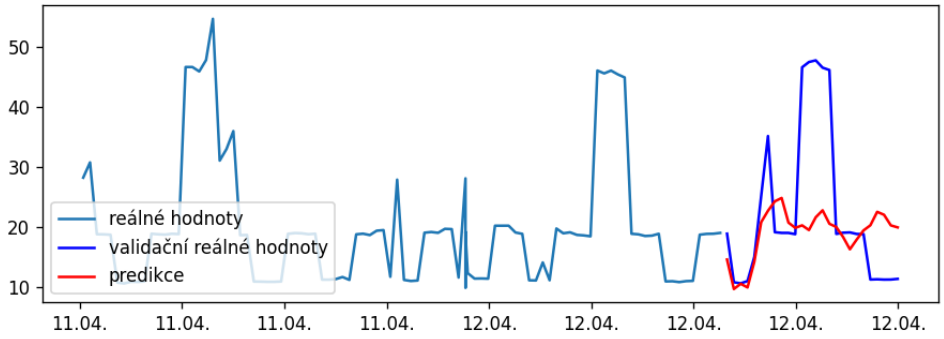
Graf (Obr. 30 Porovnání skutečných a predikovaných hodnot opotřebení filtru.Obr. 30) zobrazuje predikci hodnot **senzoru opotřebení filtru rekuperační jednotky**. Vstupní dataset byl rozdělen v poměru 80 % vstupní část a 20 % validační část. Pro predikci byl použit model ARIMA(2,1,2).




Obr. 30 Porovnání skutečných a predikovaných hodnot opotřebení filtru.

Z grafu je patrné, že hodnota opotřebení filtru lineárně vzrůstá, dokud nedojde k jeho výměně. Díky těmto informacím lze vyvodit, že je predikce v tomto deterministickém prostředí velice přesná.

Graf (Obr. 31) obsahuje predikci hodnot **senzoru aktuálního příkonu rekuperační jednotky**. Vstupní dataset byl opět rozdělen v poměru 80 % vstupní část a 20 % validační část. Jelikož je aktuální příkon nestabilní veličina, bylo nutné navýšit parametry $p$ a $q$. Díky tomu predikce více reflektuje kolísání hodnot. Byl použit model ARIMA(8,0,8).




Obr. 31 Porovnání skutečných a predikovaných hodnot příkonu rekuperační jednotky.

V tomto případě je vidět, že s postupem času se predikované a reálné hodnoty od sebe čím dál tím vice odlišují. V případě, že je požadována predikce pouze několika jednotek až nižších desítek budoucích hodnot, jsou výsledky predikce tímto modelem dostačující. Pro potřeby přesnějších predikcí by bylo nutné implementovat statistické modely, které lépe reflektují opakující se kolísavé hodnoty (např. model SARIMAX).

# ZÁVĚR

V teoretické části práce byly popsány principy a vlastnosti big dat včetně možností jejich ukládání. Podrobněji se práce věnuje jednotlivým typům NoSQL databází a rozdílům mezi nimi. Poté byl obecně představen data mining jakožto proces. Preprocessing, který je obecně nezbytnou součástí práce s daty, byl popsán v poslední kapitole teoretické části.

Praktická část této práce se v první kapitole věnuje popisu aktuální struktury uchovávaných historických hodnot z IoT prvků v platformě MyMight. V následující kapitole byl kompletně navrhnut systém pro uchování historických hodnot tak, aby umožňoval snadnou integraci na platformu MyMight. Systém byl částečně implementován, aby bylo možné ověřit jeho funkčnost. Důvodem změny způsobu uchování historických hodnot v platformě MyMight je zrychlení operací nad historickými daty, aby bylo možné provádět úlohy z oblasti data miningu.

V předposlední kapitole praktické části práce byl popsán způsob návrhu a implementace několika úloh z oblasti data miningu. První touto úlohou je detekce anomálií, která byla implementována způsobem umožňujícím zpětnou detekci již zaznamenaných hodnot s pomocí algoritmu DBSCAN a způsobem umožňujícím detekci u nově příchozích hodnot v reálném čase s pomocí statistického modelu ARIMA. Další implementovanou úlohou je rozpoznání častých kombinací událostí, které uživatelé IoT platformy vykonávají. Pro tuto úlohu bylo využito algoritmu Apriori. Poslední implementovanou úlohou je predikce časových řad. Při implementaci této úlohy se vycházelo z již implementovaného statistického modelu ARIMA, který právě na základě predikovaných hodnot určuje, zdali je příchozí hodnota považována za anomální.

Poslední kapitola interpretuje dosažené výsledky při testování těchto úloh nad reálnými daty ze vzorového domu. V rámci detekce anomálií byly porovnány všechny implementované varianty. Algoritmus Apriori správně odhalil časté kombinace událostí, které uživatel během dne ve svém domě vykonává. Predikce časových řad pomocí statistického modelu ARIMA se ukázala jako přesná pro jednotky až desítky budoucích hodnot. Avšak s rostoucím intervalem predikce její kvalita klesá.

Praktická část záměrně popisuje celý proces tvorby systému s podporou data miningu od návrhu architektury až po implementaci jednotlivých úloh.

# SEZNAM POUŽITÉ LITERATURY

[1] Data vs. Information. In: *Diffen* [online]. Seattle, Washington, United States: Diffen LLC [cit. 2021-12-17]. Dostupné z: https://www.diffen.com/difference/Data_vs_Information

[2] RIJMENAM, Mark. A Short History Of Big Data. In: *Datafloq* [online]. Den Haag: Datafloq, 2013 [cit. 2021-12-17]. Dostupné z: https://datafloq.com/read/big-data-history/239

[3] PETROV, Christo. 25+ Impressive Big Data Statistics for 2021. In: *Techjury* [online]. 2021 [cit. 2021-12-18]. Dostupné z: https://techjury.net/blog/big-data-statistics/

[4] HOLUBOVÁ, Irena, Jiří KOSEK, Karel MINAŘÍK a David NOVÁK. *Big Data a NoSQL databáze*. 1. Praha: Grada, 2015. Profesionál. ISBN 978-80-247-5466-6.

[5] VAISMAN, Alejandro a Esteban ZIMÁNYI. *Data Warehouse Systems: Design and Implementation* [online]. 2. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014 [cit. 2021-12-29]. Data-Centric Systems and Applications. ISBN 978-3-642-54655-6. Dostupné z: https://link-springer-com.ezproxy.lib.vutbr.cz/book/10.1007%2F978-3-642-54655-6

[6] LINTH, Adam a Jakob MATTSSON. *Investigating storage solutions for large data*. Gothenburg, Sweden, 2010. Diplomová práce. Chalmers University of Technology.

[7] THISARA, Kavini. NoSQL — CAP Theorem. *In: Medium [online]. San* Francisco, CA: Medium, 2020 [cit. 2022-02-13]. Dostupné z: https://kavinithisara.medium.com/nosql-cap-theorem-70cc6d0d760a

[8] KYBIC, Jan. Rozptylovací tabulky. In: Fakulta Elektrotechnická - *České* vysoké učení technické [online]. Praha: ČVUT, 2016 [cit. 2022-01-17]. Dostupné z: https://cw.fel.cvut.cz/b191/_media/courses/b3b33alp/prednasky/09_rozptylovaci_tabulky.pdf

[9] FORCHA, Chrysanthus. *Hash Table Data Structure Tutorial.* In: Linuxhint [online]. Morgan Hill, CA, 2021 [cit. 2022-01-17]. Dostupné z: https://linuxhint.com/hash_table_data_structure_tutorial/

[10] ERB, Benjamin. Concurrent programming for scalable web architectures *[*online]. Ulm, Germany, 2012 [cit. 2022-01-23]. Dostupné z: https://www.researchgate.net/publication/236149101_Concurrent_Programming_for_Scalable_Web_Architectures. Diplomová práce. Ulm University, Faculty of Engineering and Computer Science.

[11] WILLIAMS, Alex. NoSQL wide-column stores demystified. In*:* LogRocket [online]. Boston, MA: LogRocket, 2020 [cit. 2022-01-23]. Dostupné z: https://blog.logrocket.com/nosql-wide-column-stores-demystified/

[12] CUESTA, Hector. Analýza dat v praxi. 1. Brno: Computer Press, 2015. ISBN 978-80-251-4361-2.

[13] LAROSE, Daniel T. *Discovering Knowledge in Data: An Introduction to Data Mining*. 1. Hoboken, N.J: Wiley-Interscience, 2005. ISBN 978-0-471-66657-8.

[14] WITTEN, I., Eibe FRANK a Mark HALL. Data Mining: Practical Machine Learning Tools and Techniques. 3. Amsterdam: Morgan Kaufmann, 2011. Morgan Kaufman series in data management systems. ISBN 978-0-12-374856-0.

[15] VIKTORIN, Adam. Moderní techniky pro aplikace dataminingu. Zlín, 2015. Diplomová práce. Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky.

[16] JAIN, Palkesh. Data Mining Tasks – Overview. In: IncludeHelp [online]. New Delhi, Delhi: IncludeHelp, 2021 [cit. 2022-02-18]. Dostupné z: https://www.includehelp.com/basics/data-mining-tasks-overview.aspx

[17] KHAN, Irfan. How many types of Cluster *Analysis* and Techniques using R. In: Analytics Buddhu [online]. Bengaluru, India: Analytics Buddhu, 2016 [cit. 2022-02-19]. Dostupné z: https://analyticsbuddhu.wordpress.com/2016/11/01/types-of-cluster-analysis-and-techniques-using-r/

[18] HORNICK, Mark, Erik MARCADÉ a Sunil VENKAYALA. Java Data Mining: Strategy, Standard, and Practice. 2. Amsterdam: Morgan Kaufmann, 2006. Morgan Kaufman series in data management systems. ISBN 0-12-370452-9.

[19] ARORA, Shivam. Data Mining Vs. Machine Learning*:* The Key Difference. In: Simplilearn [online]. San Francisco, CA: Simplilearn, 2022 [cit. 2022-02-28]. Dostupné z: https://www.simplilearn.com/data-mining-vs-machine-learning-article

[20] ANUNAYA, Sadhvi. Data Preprocessing in Data Mining -*A* Hands On Guide. In: Analytics Vidhya [online]. Gurgaon, Haryana: Analytics Vidhya, 2021 [cit. 2022-03-01]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/08/data-preprocessing-in-data-mining-a-hands-on-guide/

[21] WOODIE, Alex. Data Prep Still Dominates Data *Scientists'* Time, Survey Finds. In: Datanami [online]. San Diego, CA: Datanami, 2020 [cit. 2022-03-01]. Dostupné z: https://www.datanami.com/2020/07/06/data-prep-still-dominates-data-scientists-time-survey-finds/

[22] HAN, Jiawei, Micheline KAMBER a Jian PEI. Data mining: Concepts and techniques. 3. Amsterdam: Elsevier/Morgan Kaufmann, 2012. The Morgan Kaufmann series in data management systems. ISBN 978-0-12-381479-1.

[23] MOTODA, Hiroshi a Huan LIU. Feature Extraction, Construction and Selection. 1. Boston, MA: Springer, Boston, MA, 2002. ISBN 978-1-4615-5725-8.

[24] IEEE STANDARDS ASSOCIATION (IEEE-SA). Internet of Things (IoT) Ecosystem Study. In: IEEE Internet of Things [online]. Piscataway, NJ: IEEE, 2015 [cit. 2022-03-07]. Dostupné z: https://iot.ieee.org/images/files/pdf/iot_ecosystem_exec_summary.pdf

[25] MYMIGHT S.R.O. Myjordomus dává technologiím společnou řeč. In: MyMight [online]. Praha: MyMight, 2021 [cit. 2022-03-11]. Dostupné z: https://www.mymight.com/reseni/chytre-reseni-pro-domacnosti/

[26] SPRINGSOURCE. Spring Framework Overview. In: Spring.io [online]. Palo Alto, California, USA: SpringSource, 2022 [cit. 2022-03-27]. Dostupné z: https://docs.spring.io/spring-framework/docs/current/reference/pdf/overview.pdf

[27] YOUNG, Jason. I finally get the point *of* inversion of control. In: Jason Young [online]. Milwaukee, WI: Jason Young, 2008 [cit. 2022-03-27]. Dostupné z: https://www.ytechie.com/2008/06/i-finally-get-the-point-of-inversion-of-control/

[28] JAVATPOINT. Spring Boot Architecture. In: JavaTpoint [*online*]. Noida, UP, India: JavaTpoint, c2011-2021 [cit. 2022-03-28]. Dostupné z: https://www.javatpoint.com/spring-boot-architecture

[29] ORNELAS, Gilvan. Clean Architecture with Spring Boot. In: Baeldung [online]. Bucharest, Romania: Tarnum Java SRL, 2021 [cit. 2022-03-28]. Dostupné z: https://www.baeldung.com/spring-boot-clean-architecture

[30] Organizing Layers Using Hexagonal Architecture, DDD, and *Spring*. In: Baeldung [online]. Bucharest, Romania: Tarnum Java SRL, 2020 [cit. 2022-03-28]. Dostupné z: https://www.baeldung.com/hexagonal-architecture-ddd-spring

[31] Indexes. In: MongoDB Documentation [online]. New York, New York, USA: MongoDB Documentation, 2021 [cit. 2022-04-02]. Dostupné z: https://www.mongodb.com/docs/manual/indexes/

[32] HENRIKSSON, Amelia Lindroth. Unsupervised Anomaly Detection in *Time* Series Data [online]. Stockholm, 2021 [cit. 2022-04-18]. Dostupné z: https://www.diva-portal.org/smash/get/diva2:1592993/FULLTEXT01.pdf. Studijní projekt. Královský technologický institut.

[33] BORA, Neha. Understanding ARIMA Models for Machine *Learning*. In: Capital One [online]. McLean, Virginia: Capital One, 2021 [cit. 2022-04-28]. Dostupné z: https://www.capitalone.com/tech/machine-learning/understanding-arima-models/

[34] PRABHAKARAN, Selva. ARIMA Model – Complete Guide *to* Time Series Forecasting in Python. In: Machine Learning Plus [online]. Begur, Bengaluru, India: Machine Learning Plus, 2021 [cit. 2022-04-29]. Dostupné z: https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/

[35] ZVORNICANIN, Enes. Choosing the best q and *p* from ACF and PACF plots in ARMA-type modeling. In: BAELDUNG. Baeldung [online]. Bucharest, Romania:

Baeldung, 2021 [cit. 2022-05-01]. Dostupné z: https://www.baeldung.com/cs/acf-pacf-plots-arma-modeling

[36] DEB, Sayantini. Apriori Algorithm — Know How *to* Find Frequent Itemsets. In: Medium [online]. San Francisco, CA: Edureka, 2019 [cit. 2022-05-09]. Dostupné z: https://medium.com/edureka/apriori-algorithm-d7cc648d4f1e

[37] DRAKE, Mark. Understanding Database Sharding. In: *DigitalOcean* [online]. New York: DigitalOcean, 2019 [cit. 2021-12-28]. Dostupné z: https://www.digitalocean.com/community/tutorials/understanding-database-sharding

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|---|---|
| IoT | Internet of Things |
| KDD | Knowledge Discovery in Data |
| ACID | Atomicity, Consistency, Isolation and Durability |
| SQL | Structured Query Language |
| ETL | Extraction, Transformation and Loading |
| OLAP | Online Analytical Processing |
| NoSQL | Not Only SQL |
| CAP | Consistency, Availability and Partition tolerance |
| DDL | Data Definition Language |
| HTTP | Hypertext Transfer Protocol |
| REST | Representational State Transfer |
| API | Application Programming Interface |
| CQL | Cassandra Query Language |
| XML | Extensible Markup Language |
| JSON | JavaScript Object Notation |
| BSON | Binary JSON |
| KDD | Knowledge Discovery in Databases |
| CRISP-DM | Cross-Industry Standard Process for Data Mining |
| MQTT | Message Queuing Telemetry Transport |
| IoC | Inversion of Control |
| JVM | Java Virtual Machine |
| HTTPS | Hypertext Transfer Protocol Secure |
| SSL | Secure Sockets Layer |
| TLS | Transport Layer Security |

| | |
|---|---|
| MQL | MongoDB Query Language |
| DBSCAN | Density-based spatial clustering of applications with noise |
| ARIMA | AutoRegressive Integrated Moving Average |

# SEZNAM OBRÁZKŮ

Obr. 1 Příklad rozdělení dat pomocí hashovací funkce. [37] .....17
Obr. 3 Ukázka čtení a zápisu v replikace peer-to-peer. .....19
Obr. 4 Architektura datového skladu. [5] .....21
Obr. 5 CAP vlastnosti zobrazené pomocí Vennových diagramů. [7].....25
Obr. 6 Princip ukládání do hashovací tabulky. [9] .....26
Obr. 7 Rozdělení intervalů při konzistentním hashování. [10].....28
Obr. 8 Příklad převodu dat na znalosti. [12].....37
Obr. 9 Životní cyklus CRISP-DM. [14] .....39
Obr. 10 Příklad použití dvoudimenzionální clusterové analýzy na zákazníka. [17] ..42
Obr. 11 Příklad data mining architektury s odděleným data mining nástrojem. [18].47
Obr. 12 Srovnání vynaloženého času jednotlivých úkonů datových vědců. [21].....51
Obr. 13 Abstraktní pohled na architekturu platformy MyMight. [18].....62
Obr. 14 Aktuální model komponent mikroslužby uchovávající historické hodnoty. .67
Obr. 15. Ukázka možné integrace systému na IoT platformu MyMight. .....72
Obr. 16 Vizualizace závislostí mezi objekty při použití dependency injection. [27] .75
Obr. 17 Vrstvená architektura používaná pro Spring Boot aplikace. [28] .....77
Obr. 18 Používané úrovně v Čisté architektuře. [29] .....78
Obr. 19 Ukázka procesu vyhledání podle indexu v databázi MongoDB. [31].....88
Obr. 19 Elbow graf nesoucí informaci o optimálních hodnotách parametru $\varepsilon$.....96
Obr. 20 Porovnání časové řady před a po diferenci. [33].....100
Obr. 21 Indentifikované oblasti algoritmem DBSCAN.....106
Obr. 22 Anomálie pro senzor vlhkosti vzduchu v koupelně. DBSCAN bez časové dimenze. .....110
Obr. 23 Anomálie pro senzor aktuálního příkonu rekuperační jednotky. DBSCAN bez časové dimenze.....110
Obr. 24 Anomálie pro senzor aktuálního výkonu předehřevu vody. DBSCAN bez časové dimenze.....111
Obr. 25 Anomálie pro senzor vlhkosti vzduchu v koupelně. DBSCAN se zahrnutím časové dimenze.....111
Obr. 26 Anomálie pro senzor aktuálního příkonu rekuperační jednotky. DBSCAN se zahrnutím časové dimenze. .....112

Obr. 27 Anomálie pro senzor aktuálního výkonu předehřevu vody. DBSCAN se zahrnutím časové dimenze. .....112
Obr. 28 Hodnoty senzoru vlhkosti vzduchu. .....113
Obr. 29 Hodnoty senzoru aktuálního příkonu rekuperační jednotky.....114
Obr. 30 Porovnání skutečných a predikovaných hodnot opotřebení filtru. .....119
Obr. 31 Porovnání skutečných a predikovaných hodnot příkonu rekuperační jednotky. .....120

# SEZNAM TABULEK

Tabulka 1 Ukázka rodiny sloupců pro zboží. .....30
Tabulka 2 Ukázka probíhající klasifikace datasetu z příkladu. .....44
Tabulka 3 Přehled základních typů hodnot v IoT Platformě. .....65
Tabulka 4 Ukázka databázové relace ukládající historické hodnoty IoT prvků. .....66
Tabulka 5 Porovnání velikostí pro číselné hodnoty podle různých datových typů. ...68
Tabulka 6 Statistika datasetu. .....70
Tabulka 7 Přehled všech položek vstupního JSONu pro uložení hodnoty. .....80
Tabulka 8 Atributy dokumentů ukládaných do kolekce historic_values. .....87
Tabulka 9 Nové hodnoty vlhkosti vzduchu. .....114
Tabulka 10 Nové hodnoty příkonu rekuperační jednotky. .....114
Tabulka 11 Časté kombinace událostí za duben 2021od 05:00 do 06:00. .....116
Tabulka 12 Časté kombinace událostí za duben 2021 od 20:00 do 21:00. .....116
Tabulka 13 Časté kombinace událostí za duben 2021 od 23:00 do 0:00. .....117
Tabulka 14 Časté kombinace událostí za říjen 2021od 05:00 do 06:00. .....117
Tabulka 15 Časté kombinace událostí za duben 2021 od 18:00 do 19:00. .....117
Tabulka 16 Časté kombinace událostí za říjen 2021 od 22:00 do 23:00. .....118
Tabulka 17 Časté kombinace událostí za říjen 2021 od 23:00 do 00:00. .....118

# SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH CD/DVD

# PŘÍLOHA P I: OBSAH CD/DVD

| | |
|---|---|
| **dm-scripts** | Zdrojový kód podpůrných skriptů pro data mining v jazyce Python. |
| **hist-ws** | Zdrojový kód mikroslužby pro uchování historických hodnot. |
| **dataset.json** | Testovací dataset exportovaný ve formátu JSON pro MongoDB. |