

# **Webová aplikace pro automatizované zpracování velkého množství statistických dat**

Michael Hozza

---

Bakalářská 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 BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Michael Hozza  
Osobní číslo: A19684  
Studijní program: B3902 Inženýrská informatika  
Studijní obor: Softwarové inženýrství  
Forma studia: Prezenční  
Téma práce: Webová aplikace pro automatizované zpracování velkého množství statistických dat  
Téma práce anglicky: Web Application for Automated Processing of Large Amounts of Statistical Data

## Zásady pro vypracování

1. Vypracujte literární rešerši na dané téma.
2. Zvolte vhodné prostředky pro ukládání a optimalizaci databázových dotazů a automatizované zpracování velkého množství dat.
3. Vyberte vhodné technologie pro implementaci webové aplikace.
4. Proveďte návrh všech potřebných součástí webové aplikace.
5. Implementujte vámi navrženou webovou aplikaci.
6. Věnujte pozornost zabezpečení aplikace.
7. Vaše řešení vhodně otestujte a vyhodnoťte.

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

**Seznam doporučené literatury:**

1. WISBORG, Krogh Jesper. MySQL 8 Query Performance Tuning. 1. New York, New York, USA: APress, 2020. ISBN 9781484255834.
2. Martin Grandjean. A social network analysis of Twitter: Mapping the digital humanities community. Cogent Arts & Humanities, Taylor & Francis, 2016, pp.1171458. 10.1080/23311983.2016.1171458. hal-01517493
3. WASSERMAN, Stanley. Social Network Analysis. 1. Cambridge, Velká Británie: Cambridge University Press, 1994. ISBN 0521387078.
4. TILMANN, Rabl. Big Data Benchmarking. 1. Wien, Rakousko: Springer-Verlag, 2015. ISBN 3319202324.
5. BIEHL, Matthias. GraphQL API Design (API-University Series) (Volume 5). 1. Suurstoffi – 6343 Rotkreuz – Switzerland: API-University Press, 2018. ISBN 1979717524.

Vedoucí bakalářské práce: **Ing. Petr Žáček, Ph.D.**  
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **3. prosince 2021**

Termín odevzdání bakalářské práce: **23. května 2022**

**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 bakalářské 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 bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou 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 – bakalářskou 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 bakalářské 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 bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské 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 bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 22. 5. 2022

Michael Hozza v. r.  
podpis studenta

## **ABSTRAKT**

Práce se zabývá rozborem problematiky Big Data. V teoretické části je vysvětlena definice Big Data, možnosti jejich ukládání, zpracovávání a interpretování. Dále jsou zde popsány databáze, které se aktivně využívají pro různé typy aplikací. Teoretická část je zakončena teorií grafů, které se běžně využívají a jak se interpretují. Praktická část se zabývá procesem implementování aplikace, která dokáže zpracovávat Big Data do přijatelného výstupu.

Klíčová slova: Big Data, databáze, grafy, vývoj aplikace, zabezpečení systému

## **ABSTRACT**

The thesis deals with the analysis of Big Data challenges. The theoretical part explains the definition of Big Data, options to store them, process them and interpret them. There are also described database systems actively used in different systems. The theoretical part ends with graph theory, which charts are commonly used and how they are interpreted. The practical part deals with the process of implementing an application for Big Data processing.

Keywords: Big Data, databases, charts, implementation of application, system security

“God will not look you over for medals, degrees or diplomas but for scars.” - Elbert Hubbard

Na tomto místě bych chtěl poděkovat vedoucímu mé bakalářské práce Ing. Petrovi Žáčkovi, Ph.D. za jeho odborné rady, trpělivost a vedení práce.

Dále bych chtěl poděkovat Patrikovi Hajšovi za jeho morální podporu při implementaci praktické části této práce a při procesu výzkumu.

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

# OBSAH

<b>ÚVOD</b> .....	<b>10</b>
<b>I TEORETICKÁ ČÁST</b> .....	<b>11</b>
<b>1 DATABÁZE</b> .....	<b>12</b>
<b>1.1 RELAČNÍ DATABÁZE</b> .....	<b>12</b>
1.1.1 SQL DATABÁZE .....	12
1.1.2 MYSQL .....	13
1.1.3 MARIADB .....	14
<b>1.2 NOSQL DATABÁZE</b> .....	<b>15</b>
<b>1.3 DOKUMENTOVÉ DATABÁZE</b> .....	<b>15</b>
1.3.1 MONGODB.....	16
<b>1.4 GRAFOVÉ DATABÁZE</b> .....	<b>17</b>
1.4.1 NEO4J .....	17
<b>2 VELKÁ DATA (BIG DATA)</b> .....	<b>18</b>
<b>2.1 TYPY VELKÝCH DAT</b> .....	<b>18</b>
2.1.1 STRUKTUROVANÁ DATA.....	18
2.1.2 NESTRUKTUROVANÁ DATA .....	18
2.1.3 SEMI-STRUKTUROVANÁ DATA.....	19
<b>2.2 UKLÁDÁNÍ VELKÝCH DAT</b> .....	<b>19</b>
<b>2.3 PROBLÉMY S VELKÝMI DATY A JEJICH ŘEŠENÍ</b> .....	<b>19</b>
2.3.1 NEDOSTATEČNÝ POČET ODBORNÍKŮ .....	19
2.3.2 ROSTOUCÍ MNOŽSTVÍ DAT .....	20
2.3.3 VÝBĚR VHODNÝCH NÁSTROJŮ PRO VELKÁ DATA.....	20
2.3.4 INTEGRACE ÚDAJŮ Z RŮZNÝCH ZDROJŮ.....	20
2.3.5 BEZPEČNOST DAT .....	20
<b>2.4 VIZUALIZACE VELKÝCH DAT</b> .....	<b>21</b>
<b>3 PRŮZKUM TECHNOLOGIÍ</b> .....	<b>22</b>
<b>3.1 UBUNTU</b> .....	<b>22</b>
<b>3.2 NGINX</b> .....	<b>22</b>
<b>3.3 DOCKER</b> .....	<b>23</b>
<b>3.4 JAVA</b> .....	<b>23</b>
<b>3.5 JAVA SPRING</b> .....	<b>24</b>
<b>3.6 VAADIN FLOW</b> .....	<b>25</b>
<b>3.7 GIT</b> .....	<b>26</b>

3.7.1	GITLAB .....	26
3.7.1.1	GitLab DevOps .....	26
<b>3.8</b>	<b>GRAF.....</b>	<b>27</b>
<b>II</b>	<b>PRAKTICKÁ ČÁST.....</b>	<b>29</b>
<b>4</b>	<b>VÝVOJ APLIKACE.....</b>	<b>30</b>
<b>4.1</b>	<b>STANOVENÍ POŽADAVKŮ .....</b>	<b>30</b>
<b>4.2</b>	<b>VÝBĚR VHODNÝCH TECHNOLOGIÍ.....</b>	<b>31</b>
4.2.1	FYZICKÝ SERVER.....	31
4.2.2	WEBOVÝ SERVER .....	31
4.2.3	PROGRAMOVACÍ PROSTŘEDKY .....	32
4.2.3.1	Obecný programovací jazyk .....	32
4.2.3.2	Webová technologie.....	32
4.2.3.3	Frontend balíček.....	33
4.2.3.4	Správa závislostí.....	33
4.2.3.5	Knihovna pro grafy .....	33
4.2.3.6	Monitoring programu .....	33
4.2.4	DATABÁZOVÉ PROSTŘEDKY .....	34
4.2.4.1	MariaDB.....	34
4.2.4.2	Neo4J.....	34
4.2.5	PODPŮRNÉ TECHNOLOGIE.....	34
4.2.5.1	GitLab .....	34
4.2.5.2	Docker .....	34
<b>4.3</b>	<b>PŘÍPRAVA, INSTALACE A KONFIGURACE JEDNOTLIVÝCH TECHNOLOGIÍ.....</b>	<b>35</b>
4.3.1	INSTALACE WEBOVÉHO SERVERU .....	35
4.3.2	INSTALACE MARIADB .....	36
4.3.3	INSTALACE NEO4J.....	37
4.3.4	INSTALACE GITLAB .....	37
4.3.5	ZABEZPEČENÍ DOMÉN HTTPS PROTOKOLEM .....	38
4.3.6	INSTALACE DOCKERU .....	38
4.3.7	KONFIGURACE GITLAB DEVOPS.....	39
4.3.8	ZABEZPEČENÍ SERVERU POMOCÍ FIREWALLU.....	39
<b>4.4</b>	<b>MODELOVÁNÍ APLIKACE .....</b>	<b>40</b>
4.4.1	DATABÁZOVÝ DIAGRAM .....	40
4.4.2	DRÁTĚNÉ MODELY .....	40
<b>4.5</b>	<b>IMPLEMENTACE APLIKACE .....</b>	<b>43</b>
4.5.1	PŘÍPRAVA.....	43
4.5.2	IMPLEMENTACE DATABÁZÍ.....	45

4.5.3	SBÍRÁNÍ DAT .....	46
4.5.4	DEADLOCK DETEKCE.....	47
4.5.5	OPTIMALIZACE MARIADB DOTAZŮ .....	48
4.5.5.1	Návrhy tabulek .....	48
4.5.5.2	Systémové prostředky .....	48
4.5.5.3	Optimalizace dotazů.....	49
4.5.6	TVORBA POHLEDŮ.....	49
<b>4.6</b>	<b>VÝSTUPY APLIKACE.....</b>	<b>49</b>
4.6.1	VELIKOST DATABÁZÍ.....	49
<b>ZÁVĚR .....</b>	<b>.....</b>	<b>52</b>
<b>SEZNAM POUŽITÉ LITERATURY.....</b>	<b>.....</b>	<b>53</b>
<b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....</b>	<b>.....</b>	<b>56</b>
<b>SEZNAM OBRÁZKŮ .....</b>	<b>.....</b>	<b>57</b>
<b>SEZNAM PŘÍLOH.....</b>	<b>.....</b>	<b>58</b>
<b>PŘÍLOHA P I: ELEKTRONICKÁ VERZE BAKALÁŘSKÉ PRÁCE, ZDROJOVÝ KÓD APLIKACE A MANUÁL .....</b>	<b>.....</b>	<b>59</b>

## ÚVOD

Data jsou základní jednotkou internetu, od samotného vzniku internetu neustále přibývají. Organizace nepřetržitě přijímají, generují a zpracovávají data o klientech, zaměstnancích a všech procesech ve firmách. S přibývajícím množstvím dat na internetu se postupně objevuje problém s jejich ukládáním a následným zpracováním.

V dnešní době je možné najít na internetu téměř cokoli, vyhledávače dokáží indexovat, zpracovávat a zobrazovat data ve zlomku vteřiny. Sociální sítě dokáží zobrazovat relevantní informace na profilech všech uživatelů i přesto že se jedná o miliony aktivních jedinců.

S neustálým nárůstem dat vznikly nové metody a techniky, jak je uchovávat, zpracovávat a zobrazovat. Právě tato práce se snaží vysvětlit, popsat a ukázat jak s daty nakládat, jaké jsou stabilní technologie pro vývoj tohoto druhu aplikací. Přestože velké korporace řešení již našly a implementovaly, velká část malých firem se každodenně potýká s tímto problémem, který zpomaluje jejich systémy a zahlcuje pracovníky.

Teoretická část popisuje problematiku Big Data, jejich uchování, zpracování a interpretování. Dále vysvětluje běžně používané technologie jako jsou databázové systémy a webové systémy. V neposlední řadě řeší teorii, která se skrývá za základními druhy grafů a také za složitějšími druhy jako je síťový graf.

V praktické části je popsán způsob implementace vlastní aplikace pro zpracování Big Data, od návrhu požadavků, přes výzkum technologií, modelování aplikace až po samotnou implementaci aplikace do produkčního prostředí. Dále je zde vysvětleno, jak takový systém zabezpečit a tím zajistit stabilní běh celé aplikace.

## **I. TEORETICKÁ ČÁST**

## 1 DATABÁZE

Databáze je kolekce obsahující informace, ke kterým je možné přistupovat a spravovat je. Udržují záznamy dat a informací, jako například finanční toky nebo údaje zákazníků.

Databáze jsou primárně určeny pro ukládání, získávání a spravování různých typů dat. Vzhledem k tomu, že jsou tato data soustředěná na jediném místě, mohou být zkoumána a vyhodnocována, proto je lze považovat za strukturovanou kolekci dat a informací. [1]

Možné využití databází:

- **Zdokonalení obchodních postupů** tím, že společnosti zkoumají a vyhodnocují získaná data různých obchodních údajů, s cílem je zdokonalit a zvýšit tak zisky.
- **Udržování zákaznických dat**, jako jsou informace o jedincích, tedy zákaznicích nebo uživatelích (jména, adresy, věk...). Tyto informace pak slouží ke zlepšení interakce mezi uživatelem a společností.
- **Udržování zdravotních údajů** kvůli bezpečnému uchovávání zdravotních záznamů pacientů a tím zvyšování informovanosti pacientů.
- **Uchovávání osobních údajů**, například fotografií nebo jiných osobních údajů. [1]

### 1.1 Relační databáze

Relační databáze jsou databáze skládající se z dat, která jsou umístěna do tabulek a rozdělena do předem připravených skupin. Všechny tabulky jsou tvořeny sloupci, ve kterých jsou uvedeny skupiny dat, a řádky, které definují, o jaký datový typ ve sloupcích se jedná. Tyto tabulky používají indexaci pro snadné vyhledávání, hlavně pomocí dotazů jazyka SQL. [1]

#### 1.1.1 SQL databáze

SQL je zkratka používaná pro Structured Query Language, v překladu strukturovaný dotazovací jazyk. Komunikuje s databází a považuje se za standard pro systémy, které využívají relačních databází. Příkazy jazyka SQL jsou používány k vykonávání různých operací v databázi, hlavně k načítání nebo aktualizaci dat. Je to nejpoužívanější jazyk v databázových systémech. Nejznámější systémy používající jazyk SQL jsou Oracle, Sysbase nebo Microsoft SQL Server. [2]

### 1.1.2 MySQL

MySQL je open-source systém, který se používá pro správu relačních databází. Byl vyvinutý společností Oracle Corporation. Jedná se o jeden z nejvíce používaných databázových systémů. Je možné jej využívat pro různé aplikace, webové stránky a přístroje. [3]

Příklady použití MySQL jsou:

- **Zpracování dat v reálném čase** – MySQL je vhodné používat v aplikacích, které potřebují jednoduchý SQL systém nebo webové stránky s vysokým provozem.
- **LAMP stack** – MySQL, díky své jednoduché syntaxi, je možno kombinovat s dalšími aplikacemi ze setu LAMP (Linux, Apache, MySQL a PHP/Python/Perl).
- **E-komerce** – MySQL se využívá při práci s velkým počtem požadavků, přičemž je dokáže vyřídit v co nejmenším časovém úseku, například převody peněz nebo zákaznická data.
- **Odhalování podvodů** – MySQL je schopen vyhodnocovat a zabránovat možným neočekávaným chováním a podvodům v reálném čase. [3]

### 1.1.3 MariaDB

MariaDB je open-source systém, který spadá do RDBMS (relational database management system). MariaDB je silně kompatibilní s technologií MySQL, protože MariaDB byla vytvořena jako softwarový fork MySQL. Ve většině případů stačí odinstalovat MySQL, nainstalovat MariaDB a systém je stále funkční. [3]

MariaDB je založena na SQL jazyce a používá ACID styl zpracovávání dat, díky kterému dosahuje čtyř bodů pro bezpečné zachovávání dat:

- **Atomicita** – Někdy také nedělitelnost, zaručuje, že operace se provede najednou a nelze ji přerušit ničím jiným. Pokud dílčí část úkolu selže, selže úkol celý a žádná změna se neprovede.
  - **Konzistence** – Někdy také korektnost, zaručuje, že každá operace musí být v souladu s předem definovanými pravidly. Taková pravidla jsou nejčastěji kaskády, vazby a triggery.
  - **Izolace** – Transakce jsou často spouštěny souběžně – izolace zajišťuje, že výstup každé transakce bude stejný, jako by byly spuštěny postupně. Hlavním cílem izolace je souběžné řízení, kdy je každá transakce izolována od nedokončených transakcí, tudíž jednotlivé kroky transakcí nejsou viditelné ostatním, dokud nejsou potvrzené (committed).
  - **Trvalost** – každá úspěšně zapsaná transakce se zapíše do databáze natrvalo, nesmaže ji ani pád systému, či výpadek proudu. Trvalost se řeší nevolatilním úložištěm, kde se log transakce zapíše a v případě pádu systému se znovu spustí transakce z logu.
- [4]

MariaDB byla vytvořena v roce 2009 týmem, který vedl Michael Widenius, jenž taktéž vedl tým při tvorbě MySQL databáze. [3]

## 1.2 NoSQL databáze

NoSQL je akronymem pro „not only SQL“, což jsou typy databází, které neukládají data do tabulek a jejich vnitřní systém není relační. Byly vytvořeny na základě vzrůstajícího objemu dat. Hlavními typy NoSQL databází jsou dokumentové, klíč-hodnota, sloupcové a grafové. Výhodou NoSQL oproti SQL databázím je vysoká škálovatelnost, taktéž odpadá nutnost znát přesnou strukturu dat. NoSQL se dále zaměřuje na problematiku ACID, kdy je restriktivní pro určité typy aplikací – pořádek v databázi je řešen programátorem na aplikační úrovni. NoSQL databáze by se neměly používat, když jsou nutné transakce a spojování tabulek.

V souvislosti NoSQL databází se používá takzvaný CAP teorém, který se snaží zabezpečit chod databáze. CAP se snaží splnit tři požadavky:

- **Konzistence** – Stejná data jsou k dispozici v celém systému.
- **Dostupnost** – Každý dotaz k databázi dostane odpověď, zdali operace proběhla úspěšně či ne.
- **Tolerance k rozdělení** – Pokud dojde k výpadku části sítě, systém je schopný stále odpovídat na dotazy.

Teorém také vysvětluje, že nelze všechny tři body splnit najednou. [5]

## 1.3 Dokumentové databáze

V případě dokumentových databází se ukládají dokumenty ve formátech JSON nebo BSON. Dokument obsahuje pole klíčů a hodnot. Každý uložený dokument může mít různou strukturu i ve stejné kolekci. Hlavními dokumentovými databázemi jsou MongoDB a Elasticsearch. [6]

### 1.3.1 MongoDB

MongoDB je objektový, dynamický a lehce škálovatelný typ databáze. Používá místo tabulek takzvané kolekce, řádky a sloupce nahrazují dokumenty a pole. Jedná se o bezschémový typ databáze, kdy není nutné předem znát strukturu kolekcí ani dokumentů. Velkou výhodou oproti SQL databázím je, že se objekty do sebe mohou vnořovat v rámci jednoho dokumentu, tudíž není nutné skládat tabulky pomocí funkce JOIN. [6]

Mezi hlavní vlastnosti MongoDB patří:

- **Ad hoc dotazy** – MongoDB umožňuje hledání podle pole, rozsahové dotazy a vyhledávání pomocí regulárních výrazů. Samotné dotazy mohou vracet i specifická pole dokumentů a také mohou zahrnovat uživatelem definované JavaScript funkce. Taktéž lze nakonfigurovat, aby dotaz vrátil náhodný vzorek výstupů o definované velikosti.
- **Indexování** – Pole v MongoDB dokumentech mohou být indexována primárním a sekundárním indexem.
- **Replikace** – MongoDB nabízí vysokou dostupnost s pomocí replikačních sad. Replikační sada obsahuje dvě nebo více kopií dat. Každý prvek replikační sady může fungovat jako primární, či sekundární replika. V rámci primární repliky se provádí veškeré čtecí a zápisové operace. Sekundární repliky uchovávají kopie dat primární repliky. V případě selhání primární repliky replikační sada vybere automaticky náhradu ze sekundárních replik. Sekundární repliky mohou pouze provádět čtecí operace, díky tomu zůstávají data konzistentní.
- **Vyvažování zátěže** – MongoDB využívá horizontálního škálování díky „shardingu“. Uživatel vybere „shard“ klíč, který nese informaci, jak se data v kolekci budou distribuovat. Data se rozdělí na rozsahy pomocí „shard“ klíčů. MongoDB také může běžet napříč několika servery, vyvažovat zátěž, či klonovat data, aby systém běžel i v případě výpadku.
- **Ukládání souborů** – MongoDB lze využít také jako souborový systém, nazývaný se GridFS, který je vestavěn v základních MongoDB ovladačích.
- **Serverové vykonávání JavaScriptu** – JavaScript lze využít v dotazech a agregačních funkcích, které jej vyšlou do databáze pro vykonání.

- **Omezení kolekcí** – MongoDB podporuje kolekce o fixní velikosti.
- **Agregace** – MongoDB poskytuje 3 způsoby, jak vytvořit agregace.
  - Agregáční roura
  - Map-reduce funkce
  - Jednoúčelové agregáční funkce

Agregáční část MongoDB umožňuje získat podobné výsledky, jaké získává SQL JOIN funkce. [6]

## 1.4 Grafové databáze

Grafové databáze ukládají vrcholy a hrany namísto tabulek či dokumentů. Každý vrchol představuje jeden záznam a hrany mezi nimi představují vztahy. Data se ukládají bez předdefinované struktury. V grafových databázích neexistuje JOIN operace, která bývá časově náročná, místo toho se vztahy ukládají přímo s vrcholy.

Každý vrchol může být označen pomocí „labels“, které představují typ role v databázi (např. „label“ Osoba). Vrcholy mohou obsahovat libovolný počet klíč-hodnota párů neboli vlastností, taktěž mohou obsahovat metadata, jako například indexy či vazby.

Každá hrana musí mít směr, typ, začáteční vrchol a konečný vrchol, taktěž mohou obsahovat vlastnosti jako vrcholy. [7]

### 1.4.1 Neo4J

Neo4J je open-source NoSQL typ databáze, který poskytuje podporu ACID. Neo4J je nativní grafová databáze, což znamená, že implementuje grafový model až na úroveň úložiště. Taktěž podporuje clustering.

Neo4J využívá Cypher jazyk pro tvorbu dotazů, který je podobný SQL, nicméně je optimalizovaný pro grafy. Taktěž zaručuje konstantní čas při procházení mezi vrcholy. [7]

## 2 VELKÁ DATA (BIG DATA)

Velká data neboli Big Data, jsou obrovská a komplikovaná data, přičemž jejich zpracování běžnými metodami je téměř nemožné. S jejich definicí přišel analytik Doug Laney, který tento pojem definoval jako pojem tří V:

- **Volume (Objem)** – Společnosti a organizace sbírají různá data z různých zdrojů, jako jsou například transakce, videa, fotografie, sociální sítě a jiné. Tato data by bylo velmi náročné a nákladné ukládat.
- **Velocity (Rychlost)** – Datové toky do organizací, jež rostou s celkovým růstem informací na internetu, musejí být zvládnutelné v přijatelném čase. Proto byly vyvinuty různé senzory nebo RFID štítky, které dokáží zpracovat velké množství dat v co nejlepším čase.
- **Variety (Různorodost)** – Různorodá data přicházejí v odlišných formátech – od organizovaných, přes databázové struktury, emaily, textové dokumenty, finanční toky a dalších. [8]

### 2.1 Typy velkých dat

Velká data mohou být dělena dle strukturálních typů. [20]

#### 2.1.1 Strukturovaná data

Strukturovaná data zahrnují veškerá data, která jsou udržována a zpracovávána v pevně daném formátu. Dá se u nich jednoduše provést analýza a zpracování. Jejich množství s plynutím času neustále narůstá, díky čemuž je nutnost velkého datového úložiště. [20]

#### 2.1.2 Nestrukturovaná data

Nestrukturovaná data slučují veškerá data, která nemají definovanou jasnou formu či strukturu. Na rozdíl od strukturovaných dat je jejich zpracování velmi náročné z důvodu zařazenosti do rozdílných kategorií. U nestrukturovaných dat není sloučení vhodné. Příkladem nestrukturovaných dat jsou kombinace textových souborů, videí, obrázků, zvuků a jiných zdrojů. [20]

### 2.1.3 Semi-strukturovaná data

Semi-strukturovaná data jsou kombinací jak strukturovaných, tak nestrukturovaných dat. Lze je vidět ve formě strukturovaných dat, ačkoliv ve skutečnosti obsahují nestrukturovaná data. Příkladem mohou být data webových aplikací, která obsahují nestrukturovaná data jako jsou například soubory protokolů nebo transakcí. [20]

## 2.2 Ukládání velkých dat

Existuje několik typů ukládacích technologií pro Big Data:

- **Distribuované souborové systémy** jsou systémy schopné ukládat enormní množství neorganizovaných dat, jako například Hadoop File System (HDFS), jenž je součástí Hadoop frameworku. Díky svému rychlému přijímání a zpracovávání velkých objemů dat se stal standardem, přestože existují výkonnostně lepší systémy.
- **NoSQL databáze** jsou jedním z nejdůležitějších technologií pro ukládání velkého množství dat, protože používají modely, u kterých není nutné, aby byly provázány s vlastnostmi modelu ACID.
- **NewSQL databáze** je moderní typ relační databáze, jehož cílem je škálovatelnost, která je ekvivalentní s databázemi typu NoSQL, přičemž zachovává vlastnosti původních databázových systémů.
- **Platformy pro dotazy s velkými daty** poskytují dotazové fasády pro obrovské úložiště dat. Jejich hlavním cílem je nabídka rozhraní na bázi jazyků typu SQL, které dosahují nízké latence dotazů. [9]

## 2.3 Problémy s velkými daty a jejich řešení

Při zpracovávání velkých dat se často objevují problémy, které se spojují s jejich ukládáním, zpracováváním či analyzováním. Tyto problémy představují jisté výzvy, které je potřeba podstoupit s cílem snížit náročnost manipulace s velkými daty. [21]

### 2.3.1 Nedostatečný počet odborníků

Společnosti a organizace se často setkávají s problémy ohledně provozu technologií a nástrojů pro správu dat z důvodu chybějících profesionálně kvalifikovaných pracovníků. Jedním z hlavních důvodů je příliš rychlý vývoj nástrojů pro práci s daty, kdy pracovníci nezvládají s vývojem držet krok. Tím se snižuje efektivita práce s velkými daty.

Většina společností a organizací reagují na tento problém investováním více peněz do vzdělávání a školení pracovníků. Jedním z důležitých kroků je také zajistit řešení dat pomocí strojů, které fungují na bázi umělé inteligence nebo strojového učení. Tyto stroje pak mohou být obsluhovány pracovníky, kteří nemusejí být odborníky na danou problematiku, ale stačí pouhá základní znalost. Toto pomáhá šetřit náklady spojené s náborem kvalifikovaných pracovníků i se vzděláváním pracovníků. [21]

### **2.3.2 Rostoucí množství dat**

Správné ukládání obrovských datových souborů je jednou z největších výzev. Množství ukládaných dat v datových centrech roste exponenciálně s časem, proto je velmi těžké se o ně postarat. Jedná se o data, která nemají strukturu a mohou pocházet odkudkoliv – z dokumentů, videí či jiných zdrojů.

Pro vyřešení problému se zpracováváním obrovského množství dat se používají techniky jako komprese, vrstvení nebo deduplikace. Tyto techniky umožňují snížení velikosti a množství dat a také zajišťují šetrnější ukládání dat do úložiště, které mohou být typu Hadoop systému nebo NoSQL databází. [21]

### **2.3.3 Výběr vhodných nástrojů pro velká data**

Při výběru nástrojů pro správu velkých dat často nastávají situace, kdy není zřejmé, který z nástrojů by byl nejvhodnější pro použití. To má za následek zbytečné plýtvání času a úsilí.

Řešením pro tuto záležitost jsou odborné rady od zkušených expertů a konzultantů, kteří nástroje znají a mohou tak doporučit, které z nich jsou neoptimálnější a nejvhodnější pro danou problematiku. [21]

### **2.3.4 Integrace údajů z různých zdrojů**

Velká data jsou smrští údajů, které pocházejí z různých zdrojů, jako jsou sociální stránky, finanční transakce nebo emaily. Jejich kombinování do celků je proto velmi náročný proces na který se často zapomíná. Integrace těchto dat je důležitá pro analýzu a vyhodnocování.

Problémy spojené s daty se řeší pořízením vhodných strojů pro jejich integraci. [21]

### **2.3.5 Bezpečnost dat**

Zabezpečit ochranu velkého množství dat je jednou z nejtěžších úloh. Často se k tomuto kroku organizace uchylují až po analýze dat, což se nejeví jako nejrozumnější možnost

vzhledem k tomu, že se mohou útočníci dostat k ještě nezabezpečeným datům a připravit tak společnost o zisk a pověst.

O zabezpečení dat se starají odborníci na kybernetickou bezpečnost, kteří se starají o šifrování dat, segregaci a autorizaci přístupu k nim. [21]

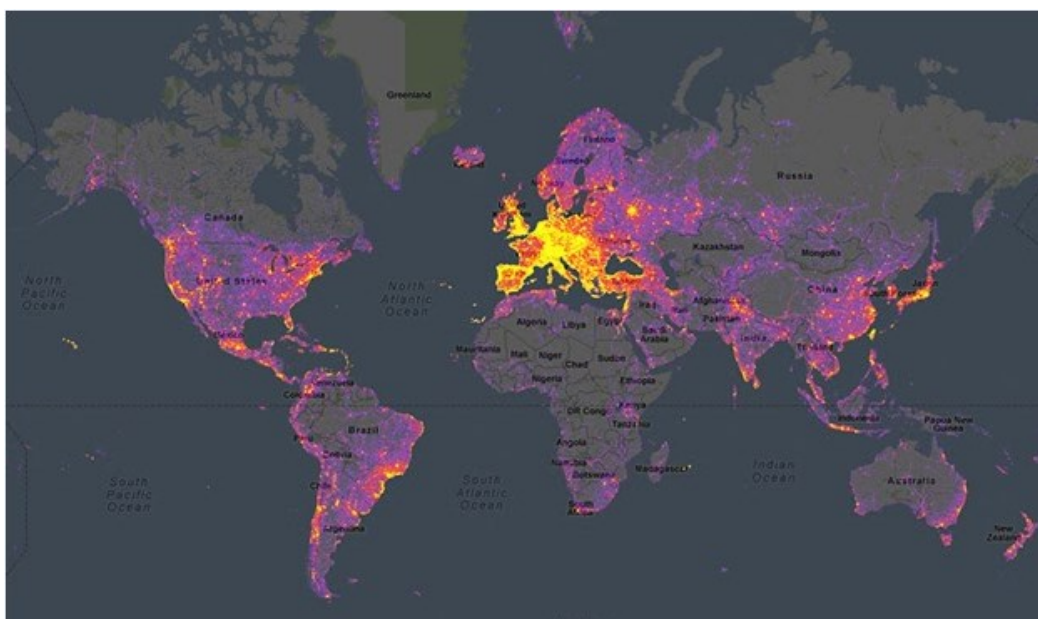
## 2.4 Vizualizace velkých dat

Vizualizace velkých množství dat se opírají o výkonné nástroje, které dokážou interpretovat zpracovaná data za účelem zobrazení jejich vizuálních charakteristik. Tato zobrazení lidem umožňují zjednodušeně porozumět velkému množství dat.

Vizualizace dokáže graficky reprezentovat libovolný druh dat – čísla, funkce či statistické údaje, které jsou vizuálně interpretovány pomocí různých technik. Nejběžněji pomocí analytických sestav nebo grafických interakcí. Tyto techniky následně usnadňují pochopení a interpretaci velkého množství dat. Konkrétní případy vizualizace jsou popsány v sekci 3.8. [22]

Techniky zobrazování jsou důležité z následujících důvodů:

- Umožňují lidem rychlé pochopení, co data reprezentují.
- Zachycují trendy, které usnadňují rozpoznání dat.
- Identifikují vzorce a souvislosti, na které je bez vizualizace nemožné odpovědět.
- Poskytují účinný způsob, jak ostatním předávat informace a poznatky. [22]



Obrázek 1 – mapa světa jako heatmap [22]

### 3 PRŮZKUM TECHNOLOGIÍ

Následující kapitola se zabývá problematikou spojenou s výběrem technologií pro implementaci aplikace.

#### 3.1 Ubuntu

Ubuntu je jedna z distribucí operačního systému Linux, která je díky své jednoduchosti jednou z nejoblíbenějších a nejvíce používaných. [10]

Jejím vynálezcem je Mark Shuttleworth, který založil nadaci Ubuntu v roce 2004. Jedním z hlavních důvodů vzniku byla snaha vyvinout uživatelsky příjemnější systém – než do té doby nejpoužívanější Debian. To se do značné míry podařilo, i když byl zezáátku příliš náročný na instalaci. Uživatelé si volí Ubuntu na základě několika pozitivních faktorů, kterými jsou:

- **Uživatelská přívětivost (User-Friendly)** – Interaktivní a vizuální jednoduchost i pro začátečníky.
- **Bezpečnost a ochrana** – Linuxové distribuce jsou považovány za více bezpečné než jiné operační systémy.
- **Bezplatné používání** – Ubuntu je zcela bezplatný operační systém. [10]

#### 3.2 NGINX

NGINX je open-source webový server, který se používá jako reverzní proxy, na ukládání do vyrovnávací paměti, streamování videa a jiné aplikace. Původně byl vytvořen jako server charakteristický svým výkonem a stabilitou. Může fungovat také jako emailový proxy server nebo vyrovnávač zatížení pro HTTP či TCP a UDP. [11]

Jeho původním cílem bylo vytvořit jeden z nejrychlejších serverů vůbec. Tento cíl se mu daří plnit, protože dokáže výkonnostně porážet další známý a používaný server Apache. [11]

NGINX a jeho škálovatelnost je ideální pro plnění webových úloh, například pro spravování webového provozu a jeho distribuování na méně výkonné servery. [11]

### 3.3 Docker

Docker je open-source platforma pro vývoj aplikací, která používá takzvané kontejnery. Tyto kontejnery jsou definovány jako malé a jednoduché jednotky, které běží nezávisle na sobě, přičemž sdílejí jádro operačního systému. Jsou schopné vytvářet programy, které se vyznačují jednoduchou sestavou, správou a přenášením. Výhody Dockeru z pohledu vývojářů jsou následující:

- **Kontejnery jsou minimalistické a lehce přenosné** – Docker tyto kontejnery izoluje a tím jsou jednoduché na ovládání a přenos.
- **Kontejnery jsou modulární** – Zjednodušují sestavování stavebních bloků do snadno vyměnitelného celku, čímž se zrychluje vývoj a opravování chyb.
- **Kontejnery usnadňují škálovatelnost** – Svou jednoduchostí a minimalismem umožňují vývojářům spustit naráz velké množství kontejnerů, což umožňuje lepší škálovatelnost programu. [12]

### 3.4 Java

Java je populární programovací jazyk, který se používá nejen pro vývoj webových aplikací. Jedná se o objektově orientovaný jazyk, jehož syntax vychází z jazyka C++. Java používá třídy, které specifikují objekty a metody. Jedná se o poměrně striktní jazyk, protože vyžaduje přesné definice proměnných a funkcí. [13]

Programy, které jsou vytvářeny pomocí Javy, jsou interpretovány pomocí Java Virtual Machine. JVM dokáže operovat na několika platformách, jako jsou Macintosh, Windows nebo Unix.[13]

### 3.5 Java Spring

Spring je multiplatformní framework, který představuje komplexní programovací model určený pro enterprise aplikace v Javě. Hlavní výhodou frameworku je podpora aplikační infrastruktury. Má několik užitečných funkcí:

- **Využívá řadu technologií** – dependency injection, události, validace a datové vazby.
- **Testovací technologie** – Spring MVC Test, TestContext framework.
- **Webové frameworky** – Spring MVC, Spring WebFlux.
- **Integrace s technologiemi** – vzdálená komunikace, email, caching a plánování.
- **Kromě Javy je možno používat i další jazyky** – Kotlin, Scala i Groovy. [14]

### 3.6 Vaadin flow

Vaadin flow je framework pro modelování webových aplikací. Je postaven na jazyce Java, přičemž dovoluje sestavovat uživatelské rozhraní buď za pomoci HTML šablon, nebo v samotné Javě a následně je propojit s backend technikami. Vaadin flow poskytuje následující možnosti:

- Architekturu zaměřenou na uživatelské rozhraní, bez nutnosti řešení komunikace mezi klientem a serverem.
- Kolekci komponentů, vytvořených pro uživatelské rozhraní, založenou na zkušenostech koncových zákazníků i vývojářů.
- Možnosti vytváření vlastních jednotek pro uživatelské rozhraní za pomoci HTML nebo samotné Javy.
- Propojující API pro připojování jednotlivých částí uživatelského rozhraní k backendu.
- API pro vytváření struktur a navigací pro uživatele. [15]

```
MainView.java
// Create an HTML element
Div layout = new Div();

// Use TextField for standard text input
TextField textField = new TextField("Your name");

// Button click listeners can be defined as lambda expressions
Button button = new Button("Say hello",
    e -> Notification.show("Hello!"));

// Add the web components to the HTML element
layout.add(textField, button);
```

Obrázek 2 – Vaadin Flow [15]

## 3.7 Git

Git je verzovací systém určený pro správu souborů a sledování změn, které v nich nastanou v průběhu času. Asistuje členům vývojářského týmu při sledování dosažených pokroků a při koordinaci zadávaných úkolů. Také sleduje situaci projektů, tím pomáhá nejen zkušeným programátorům, ale také méně zkušeným a netechnickým typům uživatelů. [16]

### 3.7.1 GitLab

GitLab je open-source webový Git repositář, který nabízí soukromé a otevřené úložiště zcela zdarma. Jedná se o plnohodnotnou platformu DevOps, která umožňuje vývojářům sledovat a spravovat důležité aspekty projektů – od plánování, údržby kódu, až po bezpečnost. [16]

Největší výhodou je možnost interakce a aktivní spolupráce celého týmu ve všech fázích projektu. Zajišťuje sledování pokroku od samotného začátku, až po dokončení projektu, tím pomáhá vývojářům automatizovat životní cyklus s cílem dosáhnout lepších výsledků. [16]

#### 3.7.1.1 GitLab DevOps

V porovnání s tradičními metodami přináší DevOps spojení vývojového procesu s operacemi. Cílem je zlepšit efektivitu, rychlost, bezpečnost vývoje a samotné dodání výsledného software. [17]

Díky automatizaci, týmové spolupráci, zpětným vazbám a vylepšováním postupy DevOps umožňují vývojářským (Devs) a operačním (Ops) skupinám zrychlit samotný vývoj a poskytování aplikací. [17]

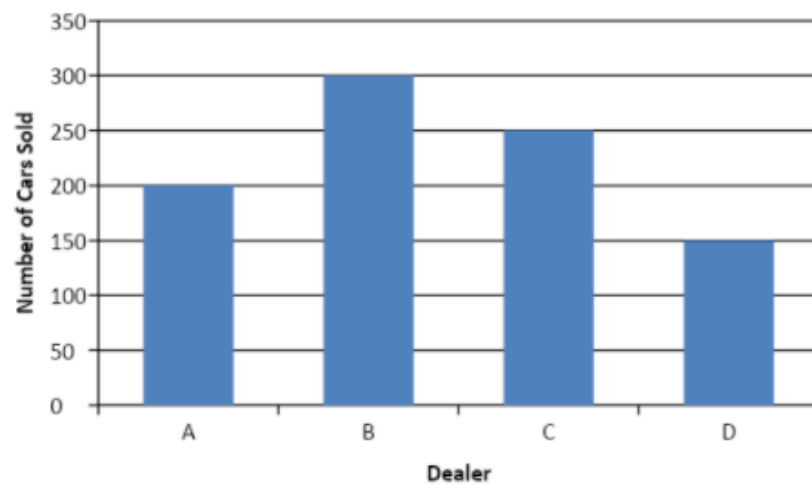
Principy DevOps jsou shrnuty ve čtyřech klíčových zásadách:

- Automatizace procesu vývoje software.
- Týmová spolupráce a komunikace.
- Postupné zlepšování a eliminování ztrát.
- Soustředění na uživatelské potřeby a následné zpětné vazby. [17]

### 3.8 Graf

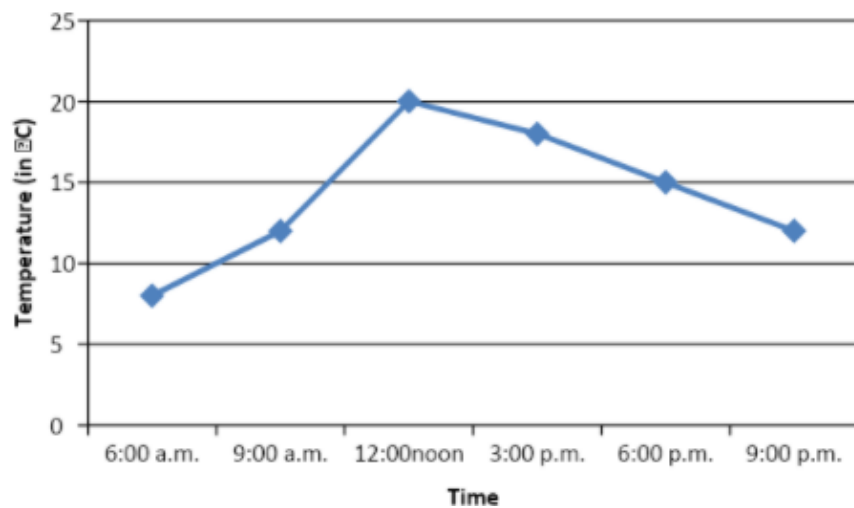
Graf se nejen v matematice používá pro obrazovou reprezentaci dat nebo hodnot. Tyto hodnoty jsou v grafu prezentovány organizovaně a vytvářejí vztahy mezi dvěma nebo více entitami. Rozlišujeme několik druhů grafů:

- **Barový graf** nebo také sloupcový graf, reprezentuje číselná data, která zobrazuje pomocí obdélníků (sloupců). Obdélníky mají různé výšky a stejné šířky, ty mohou být ve vertikální nebo horizontální pozici, tím pádem výška nebo délka sloupce reprezentuje jeho hodnotu. [18]



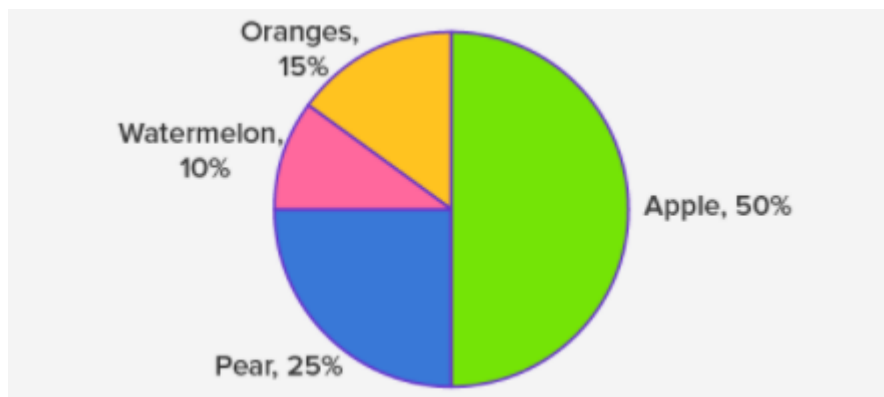
Obrázek 3 – sloupcový graf [18]

- **Liniový graf** je typ grafu, který používá body propojené křivkami. Výstup reprezentuje změny v chování za určité časové období. [18]



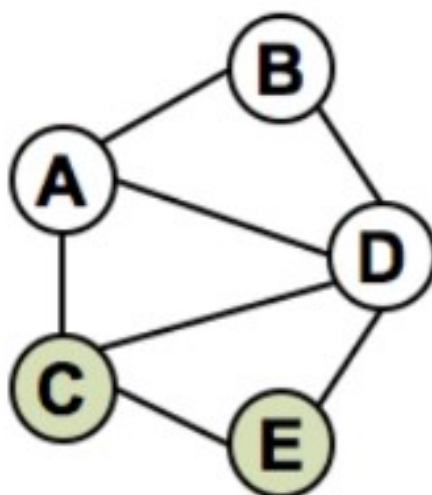
Obrázek 4 – liniový graf [18]

- **Výsečový graf**, také kruhový nebo koláčový graf, představuje celek rozdělený na části, kde každá část reprezentuje velikost nějaké skupiny dat v poměru k celému dostupnému množství dat. Většinou se poměry převádí na procenta, aby bylo zřejmé, jakou část každá skupina dat zabírá. [18]



Obrázek 5 – výsečový graf [18]

- **Sít'ový graf** se skládá ze dvou částí – uzlů a hran. Uzly v grafu reprezentují různé entity v síti, které mohou mít různé vlastnosti – výšku, váhu a jiné. Hrany pak odrážejí relace mezi těmito uzly a také mohou mít vlastnosti, například směr nebo čas. Tyto dva parametry pak popisují různé přírodní, fyzikální nebo sociální vztahy. [19]



Obrázek 6 – sít'ový graf [19]

## **II. PRAKTICKÁ ČÁST**

## 4 VÝVOJ APLIKACE

Každá aplikace se vyvíjí v určitých krocích. V případě této práce byly kroky následující:

1. Stanovení požadavků.
2. Výběr vhodných technologií dle požadavků.
3. Příprava, instalace a konfigurace jednotlivých technologií.
4. Definování struktury aplikace ve formě modelů.
5. Implementace aplikace.

### 4.1 Stanovení požadavků

Aplikace zpracovává volně dostupná data sociální sítě Twitch.tv do databáze, ze kterých se následně sestavují grafy, které popisují získaná data.

Aplikace je strukturovaná do balíčku dle MVC architektury, kdy se každá část aplikace stará jen o jeden úkol (modely získávají data z databáze a taktéž data do ní vkládají, pohledy se starají jen o zobrazování prvků, nikoliv o business logiku, kontrolery data zpracovávají a předávají je do pohledů).

Veškeré využití knihovny a součásti systému jsou volné k použití, taktéž výsledná aplikace.

Aplikace ošetřuje veškeré výjimky, které mohou nastat.

Aplikace neotevřít mimo vnitřní síť jiné porty než 80 a 443 z důvodu bezpečnosti.

Aplikace vrací platné a správné status kódy.

## 4.2 VÝBĚR VHODNÝCH TECHNOLOGIÍ

### 4.2.1 Fyzický server

Základem celého projektu je virtuální privátní server umístěný v Polsku od OVH Cloud hostingu. Tvoří jej osmijádrový procesor o taktovací frekvenci 2.4 GHz, 32 GB operační paměti a 1 TB SSD disk pro databáze.

Tento server byl zvolen z důvodu poměru cena/výkon a také pro jeho vysokou podporu ze strany hostingu. Klíčové pro zabezpečení a stabilitu je, že se hosting stará o anti DDoS opatření a také pravidelné zálohování celého serveru ve formě snapshotů.

Na stroji je nainstalován linuxový operační systém Ubuntu ve verzi 20.04.4 v cyklu LTS. Nejenže ubuntu má rozsáhlou podporu balíčků, které jsou klíčové pro chod a zabezpečení aplikace, ale také rozsáhlou komunitu, která pravidelně opravuje veškeré problémy se zabezpečením.

### 4.2.2 Webový server

Mezi dvě nejpoužívanější technologie webových serverů patří Apache a Nginx. Pro účely tohoto projektu byl zvolen Nginx. Pro výběr bylo stanoveno několik kritérií:

1. Odbavení počtu požadavků za sekundu.
2. Konfigurovatelnost.
3. Bezpečnost.
4. Reverzní proxy.

V rámci rychlosti byl Nginx téměř dvojnásobně rychlejší než Apache. Apache dokázal u statického obsahu odbavit pouze ~10 000 požadavků za sekundu, kdežto Nginx dokázal dvojnásobek ~20 000 požadavků. U dynamického obsahu byl výsledek obou technologií srovnatelný, přestože se Nginx spoléhá na externí komponenty.

Co se týká konfigurace, Apache je jednodušší pro začátečníky v oblasti webových technologií, kdežto Nginx má sofistikovanější architekturu, což občas může být u jednoduchých webů na obtíž. Další výhodou Apache v oblasti konfigurace je možnost .htaccess souborů, které slouží jako konfigurace jednotlivých složek webu, kde si programátor může stanovit, v jakých případech se do složek může přistupovat a jaká rozšíření se volají.

V ohledu bezpečnosti mají oba servery aktivní podporu, která se stará o veškeré bezpečnostní problémy.

Závěrem Nginx umožňuje nativní podporu reverzních proxy narozdíl od Apache, kde to řeší modul s velmi specifickou a striktní konfigurací. Zde má Nginx jednoznačně navrch. Pokud projekt vyžaduje velkou konfigurovatelnost a nezáleží tolik na rychlosti, či není nutná reverzní proxy, Apache je jasnou volbou. Pokud je prioritou odbavování statického obsahu a možnost reverzní proxy, je vhodnější využít Nginx.

### **4.2.3 Programovací prostředky**

#### **4.2.3.1 Obecný programovací jazyk**

Hlavním programovacím jazykem pro tento projekt byla zvolena Java. Java je dlouhá léta zásadní technologií pro programování enterprise úrovně programů. Java pracuje v rámci Java Virtual Machine, což umožňuje výborný výkon aplikací, protože se program překládá do Java bytecode. Java bytecode je podobný assembly kódu, díky čemuž také umožňuje multiplatformní programování, které Java má ve svém mottu „Write once, run anywhere“ – „Napiš jednou, spustíš kdekoliv“. Taktéž je Java výborný jazyk pro problematiku Big Data, jelikož existují knihovny (Apache Hadoop, Apache Spark atd.), které se tímto problémem zabývají.

#### **4.2.3.2 Webová technologie**

Java pro webové aplikace umožňuje výběr z několika frameworků. Mezi nejpopulárnější způsoby vývoje Java webových aplikací patří Java Enterprise Edition umožňující vývoj spolehlivých, bezpečných a škálovatelných aplikací na úkor rychlosti vývoje.

Dalším způsobem, který byl zvolen pro tuto práci, je framework Spring Boot. Spring Boot má integrovanou podporu pro „Repositories“ sloužící pro modelování databází. Dále zahrnuje bezpečnostní část s názvem Spring Security podporující autorizaci a autentizaci v rámci webové aplikace. V neposlední řadě podporuje automatizované úlohy ve formě cronu.

Velkou výhodou frameworku Spring Boot je jeho podpora na portálech StackOverflow a rozsah dokumentace.

#### 4.2.3.3 *Frontend balíček*

Pro jednoduchost kódu byl zvolen balíček Vaadin Flow, který umožňuje skládat webovou aplikaci Java kódem bez nutnosti kombinování Javy s HTML kódem, či nutnosti využívat template systém Thymeleaf nebo Mustache.

Vaadin Flow také obsahuje možnost asynchronních úloh ve formě „Pusheru“, což zajišťuje rychlé načtení stránky, přestože data nemusí být včas k dispozici. Ostatní zmíněné systémy využívají javascriptových XHR požadavků, což se pojí s nutností kombinovat Javu s Javascriptem.

Další výhodou Vaadin Flow je integrovaný routing systém, kdy stačí pouze do anotace definovat URL cestu k danému pohledu.

#### 4.2.3.4 *Správa závislostí*

Java umožňuje několik technologií pro správu balíčků jako jsou například Maven, Groovy či Gradle. Pro tento projekt byl vybrán Maven z důvodu jednoduchosti konfigurace a plnou kompatibilitou se Spring Boot frameworkem.

Veškeré závislosti v systému se řeší v pom souboru, který je ve formátu XML.

#### 4.2.3.5 *Knihovna pro grafy*

Pro vykreslování grafů byla zvolena knihovna SO Charts, což je obalená Javascript knihovna Apache Echarts pro práci v Javě bez nutnosti psát javascriptové kódy. SO Charts umožňuje vykreslování všech základních grafů (liniové, výsečové...), ale také složitějších grafů jako je například sociální síť. Tato knihovna je plně kompatibilní s frontend balíčkem Vaadin Flow, což umožňuje skládat grafické pohledy a poté je předat do hlavního pohledu.

#### 4.2.3.6 *Monitoring programu*

Pro základní monitoring programu byla zvolena technologie Actuator a Prometheus, které jsou vestavěny již v používaném frameworku Spring Boot. Tyto technologie umožňují monitoring aplikace (náročnost algoritmu na operační paměť, časová náročnost algoritmů) přímo z prohlížeče na definované URL adrese.

#### 4.2.4 Databázové prostředky

Pro vývoj byly použity dva typy databází. SQL databáze MariaDB a grafová databáze Neo4J.

##### 4.2.4.1 *MariaDB*

Jedná se o jednu z nejpoužívanějších databázových technologií. V základu několikanásobně přesahuje svého předchůdce MySQL v oblasti rychlosti. Použita byla z důvodu nutných relací mezi záznamy.

##### 4.2.4.2 *Neo4J*

Neo4J je nejpopulárnější grafová databáze na trhu, která se hodí nejen pro zpracovávání Big Data, ale také pro běžné systémy, jako jsou například e-shopy a propagační webové stránky.

#### 4.2.5 Podpůrné technologie

##### 4.2.5.1 *GitLab*

Při vývoji aplikací jakékoliv velikostí je vždy best-practice používat verzovací systém. Pro účely tohoto projektu byl na webový server nainstalován a nakonfigurován systém GitLab. GitLab umožňuje nejen verzování a správu kódu, ale také DevOps technologii, která slouží k automatickému sestavení aplikace, jejího otestování a samotnému vydání aplikace do produkčního prostředí.

##### 4.2.5.2 *Docker*

Docker je jedna z moderních technologií, která se stává standardem pro vývoj jakékoliv aplikace. Docker zajišťuje, aby aplikace měla vždy správné prostředí, ve kterém běží. Umožňuje to na základě tvorby kontejneru, který izoluje aplikaci od operačního systému a nainstaluje jí potřebné verze závislostí, jako je například verze programovacího jazyka.

Veškerá konfigurace se provádí v Dockerfile, ve kterém se vytvoří uživatel spravující danou aplikaci, přiřadí se porty potřebné pro komunikaci aplikace s operačním systémem a následně se aplikace spouští.

Docker je nástroj, díky kterému může aplikace běžet v separátním prostředí, což zvyšuje samotnou bezpečnost systému.

## 4.3 Příprava, instalace a konfigurace jednotlivých technologií

### 4.3.1 Instalace webového serveru

Pro aplikaci byla zvolena technologie Nginx, která se instaluje v prostředí ubuntu příkazem `apt-get install nginx`, při instalaci není potřeba nic konfigurovat. Jakmile se nginx nainstaluje, musí se vytvořit pravidla pro domény a subdomény. Hlavní konfigurační složka se nachází v `/etc/nginx/sites-enabled`, ve které se nachází soubor `default`, příkazem `cp default domena.conf` se vytvoří soubor pro pravidla domény, pro účely aplikace je nutné nastavit reverzní proxy na port 9696, který Spring Boot v základu používá.

```
1 server
2 {
3     listen 80;
4
5     server_name twitchstats.eu;
6
7     location / {
8         proxy_set_header X-Forwarded-For $remote_addr;
9         proxy_set_header Host $http_host;
10        proxy_pass http://127.0.0.1:9696;
11    }
12 }
```

Obrázek 7 – port forwarding nginx konfigurace

Po vytvoření je nutné zavolat příkaz `service nginx restart`, kterým se webový server restartuje a veškeré požadavky na tuto doménu nyní budou interně přesměrovány na port 9696, kde je služba běžící na tomto portu zpracuje.

### 4.3.2 Instalace MariaDB

Pro instalaci MariaDB je nutno použít příkaz `apt-get install mariadb-server mariadb-client`, při instalaci není vyžadován žádný uživatelský vstup. Po instalaci je nutné zavolat příkaz `mysql_secure_installation`, který zabezpečí instalaci databáze. Poté lze otestovat příkazové rozhraní MariaDB pomocí příkazu `mysql --user=root`. Dalším důležitým prvkem pro prohlížení MariaDB databáze je rozhraní phpmyadmin. Pro jeho funkčnost se nejprve musí nainstalovat php příkazem `apt-get install php php-json php-mbstring php-zip php-gd php-xml php-curl php-mysql`. Pro instalaci phpmyadminu se stáhne balíček z oficiálních stránek dodavatele příkazem `wget`. Po stažení se balíček rozbálí pomocí `tar` příkazu a přesune do `/usr/share/phpmyadmin` složky. Pro přístup webového serveru je nutno nastavit složce oprávnění příkazem `sudo chown -R www-data:www-data /usr/share/phpmyadmin`. Na závěr se vytvoří konfigurace pro nginx.

```
1 server
2 {
3     listen 80;
4
5     server_name maria.twitchstats.eu;
6
7     root /usr/share/phpmyadmin;
8     index index.html index.htm index.php;
9
10    location / {
11        try_files $uri $uri/ =404;
12    }
13
14    error_page 404 /404.html;
15    error_page 500 502 503 504 /50x.html;
16    location = /50x.html {
17        root /usr/share/phpmyadmin;
18    }
19
20    location ~ /\.php$ {
21        fastcgi_split_path_info ^(.+\.php)(/.+)$;
22        fastcgi_pass unix:/run/php/php7.4-fpm.sock;
23        fastcgi_index index.php;
24        include fastcgi_params;
25        fastcgi_param PHP_VALUE "upload_max_filesize = 100M \n post_max_size=100M";
26        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
27        fastcgi_param HTTP_PROXY "";
28        fastcgi_intercept_errors off;
29        fastcgi_buffer_size 16k;
30        fastcgi_buffers 4 16k;
31        fastcgi_connect_timeout 300;
32        fastcgi_send_timeout 300;
33        fastcgi_read_timeout 300;
34    }
35 }
```

Obrázek 8 – phpmyadmin nginx konfigurace

Po restartu nginxu je nyní phpmyadmin dostupný na zadané doméně.

### 4.3.3 Instalace Neo4J

Instalace Neo4J probíhá příkazem `apt-get install neo4j`, po instalaci se příkazem `cypher-shell` a zadáním údajů `username: neo4j; password: neo4j` změní heslo dle uživatelského vstupu. Poté se nakonfiguruje nginx reverse proxy. Zde je nutné nastavit doménu na port 7473, který je výchozí.

```
1  server
2  {
3      listen 80;
4      server_name neo4j.twitchstats.eu;
5
6      location / {
7          proxy_connect_timeout 300;
8          proxy_redirect off;
9          proxy_http_version 1.1;
10         proxy_set_header Host $http_host;
11         proxy_set_header X-Real-IP $remote_addr;
12         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
13         proxy_set_header X-Forwarded-Proto $scheme;
14         proxy_pass https://127.0.0.1:7473;
15         proxy_redirect https://127.0.0.1:7473 https://neo4j.twitchstats.eu;
16     }
17
18 }
```

Obrázek 9 – neo4j nginx konfigurace

Po konfiguraci nginxu v souboru `/etc/neo4j/neo4j.conf` je nutno nastavit hodnotu `dbms.default_advertised_address` na zadanou doménu. Na závěr stačí restartovat nginx a neo4j příkazem `neo4j restart`. Poté se na příslušné doméně zobrazí neo4j rozhraní.

### 4.3.4 Instalace GitLab

Instalace GitLabu probíhá spuštěním dvou příkazů: `curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.deb.sh | sudo bash` a `apt-get install gitlab-ee`, po instalaci konfigurace probíhá v souboru `/etc/gitlab/gitlab.rb`, kde se změní hodnota `external_url` na připravenou doménu. Po instalaci se heslo pro root účet nachází v souboru `/etc/gitlab/initial_root_password`. Toto heslo poté slouží pro přihlášení do webového rozhraní. Dále je nutné nakonfigurovat nginx pro gitlab.

```
1 server {
2     listen 80;
3     server_name gitlab.michaelhozza.eu;
4     server_tokens off;
5     root /opt/gitlab/embedded/service/gitlab-rails/public;
6
7     access_log /var/log/nginx/gitlab_access.log;
8     error_log /var/log/nginx/gitlab_error.log;
9
10    location / {
11        client_max_body_size 1000m;
12        gzip off;
13        proxy_read_timeout 300;
14        proxy_connect_timeout 300;
15        proxy_redirect off;
16        proxy_http_version 1.1;
17        proxy_set_header Host $http_host;
18        proxy_set_header X-Real-IP $remote_addr;
19        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
20        proxy_set_header X-Forwarded-Proto $scheme;
21        proxy_pass http://gitlab-workhorse;
22    }
23 }
```

Obrázek 10 – gitlab nginx konfigurace

Na závěr stačí restartovat nginx a poté i gitlab službu příkazem *gitlab-ctl reconfigure*. GitLab je k dispozici na doméně.

#### 4.3.5 Zabezpečení domén HTTPS protokolem

Pro zabezpečení všech domén slouží nástroj certbot, který se instaluje příkazy *sudo snap install core; snap refresh core* a poté *snap install --classic certbot*, posledním krokem před vytvořením certifikátů je vytvoření symbolického odkazu pomocí *ln -s /snap/bin/certbot /usr/bin/certbot*. Na závěr se zavolá příkaz *certbot certonly --nginx* a čísla se zvolí všechny domény. Po ukončení příkazu se změnila všechny konfigurace nginxu na listen 443 a přesměrování portu 80 na 443. Poté stačí restartovat nginx a všechny domény běží pod https protokolem.

#### 4.3.6 Instalace Dockeru

Instalace probíhá spuštěním příkazu *apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin*. Tím je nástroj úspěšně nainstalován.

### 4.3.7 Konfigurace GitLab DevOps

Pro fungování DevOps je nutno mít nainstalovaného běžce (runner), který zpracovává každou pipeline. Pipeline je úkol, který běžec plní na základě konfigurace. Instalace běžce probíhá příkazem `apt-get install gitlab-runner`, poté stačí běžce zaregistrovat příkazem `gitlab-runner register`, kdy je potřeba zadat token, který je vygenerovaný v administrátorském rozhraní gitlabu. Poté je běžec již aktivní a připravený na přiřazení v každém projektu v sekci CI/CD.

### 4.3.8 Zabezpečení serveru pomocí firewallu

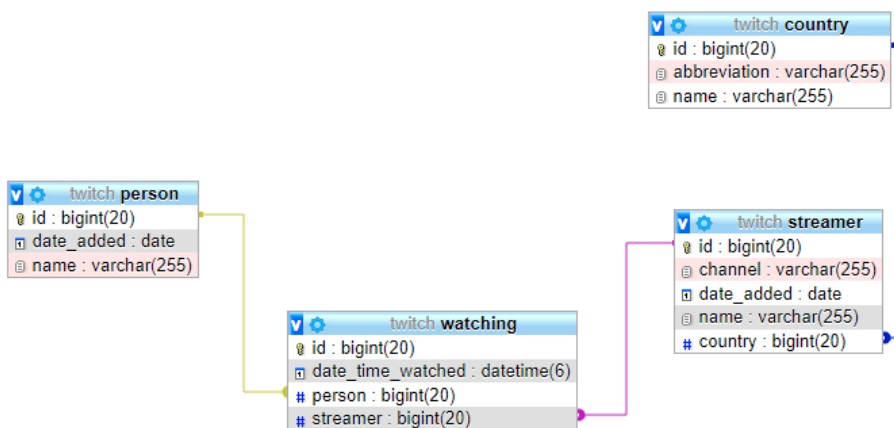
Na závěr konfigurací stačí server zabezpečit firewallem, kdy se mimo interní síť vystaví porty 80 a 443. V ubuntu se to řeší firewallem s názvem ufw. Prvním krokem je odmítnutí všech požadavků na server `ufw default deny incoming` a poté zavoláním příkazů `ufw allow 80` a `ufw allow 443` se povolí protokoly http a https. Tímto posledním procesem je zajištěno, že se nelze dotazovat přímo na databázové a ostatní služby portem z vnější sítě, pouze z vnitřní. Pro komunikaci s vnější sítí probíhá vše pomocí https protokolu za pomoci certbota a nginx konfigurací.

## 4.4 Modelování aplikace

Modely jsou důležitou součástí každého aplikačního vývoje. Pomáhají vývojovému týmu držet jasná pravidla struktur aplikace, designu a funkčnosti.

### 4.4.1 Databázový diagram

Z Twitch.tv se dají získat data o člověku, který vysílá, pouze přezdívka a státní příslušnost. O sledujících daného člověka je zaznamenáváno pouze jméno. Proto jsou definované 4 entity: country (země), streamer (vysílající), person (sledující) a watching (statistika kdo koho kdy sledoval).



Obrázek 11 – ER diagram

Z databázového diagramu se přímo odvíjejí entitní třídy v rámci aplikace, které jsou kopií jedna ku jedné.

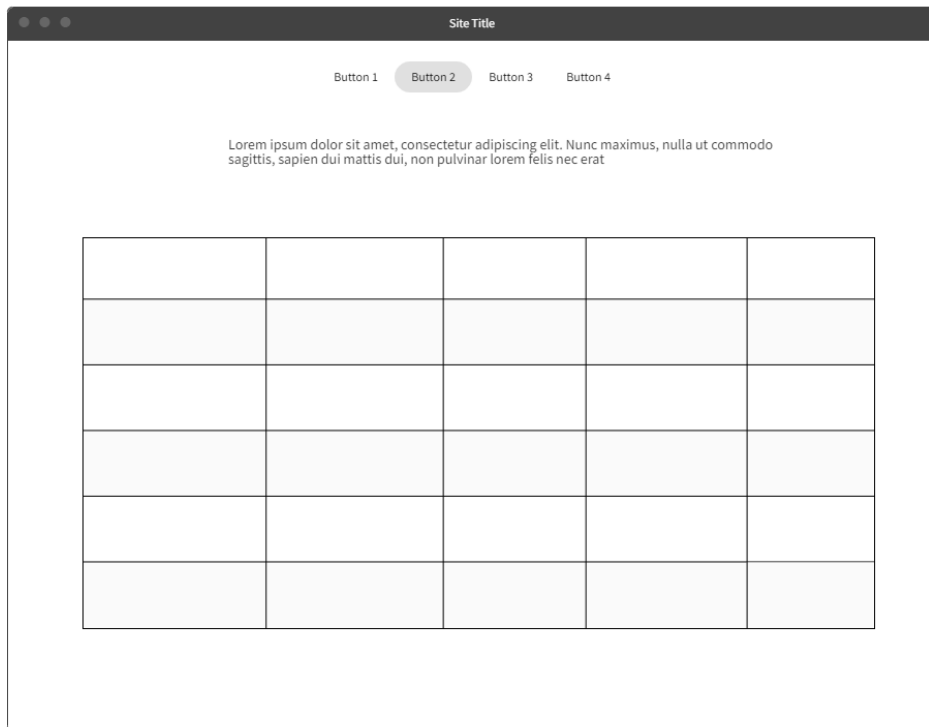
### 4.4.2 Drátěné modely

Pro návrh UI/UX se používají drátěné modely (wireframes), které slouží jako základ pro grafický návrh.

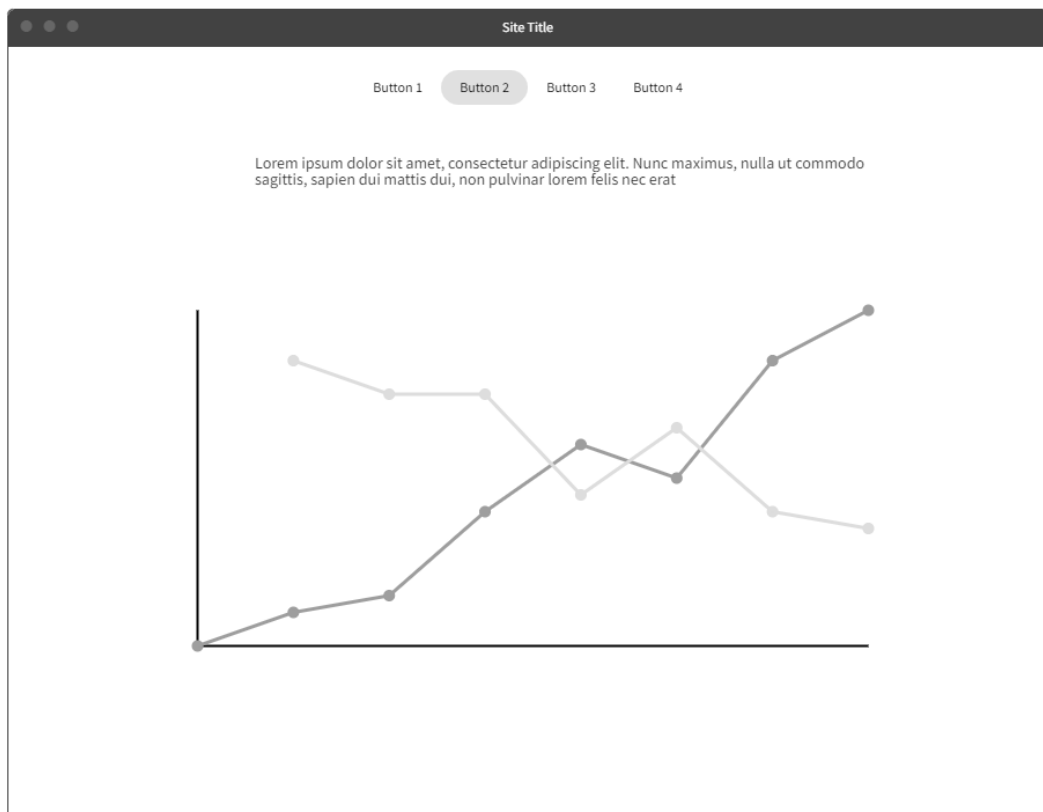
Aplikace má na úvodní stránce tabulku, která zobrazuje informace o velikosti databází používaných v systému.

Na první podstránce se nachází graf aktivity českých a slovenských sledujících na základě dne v týdnu a hodiny.

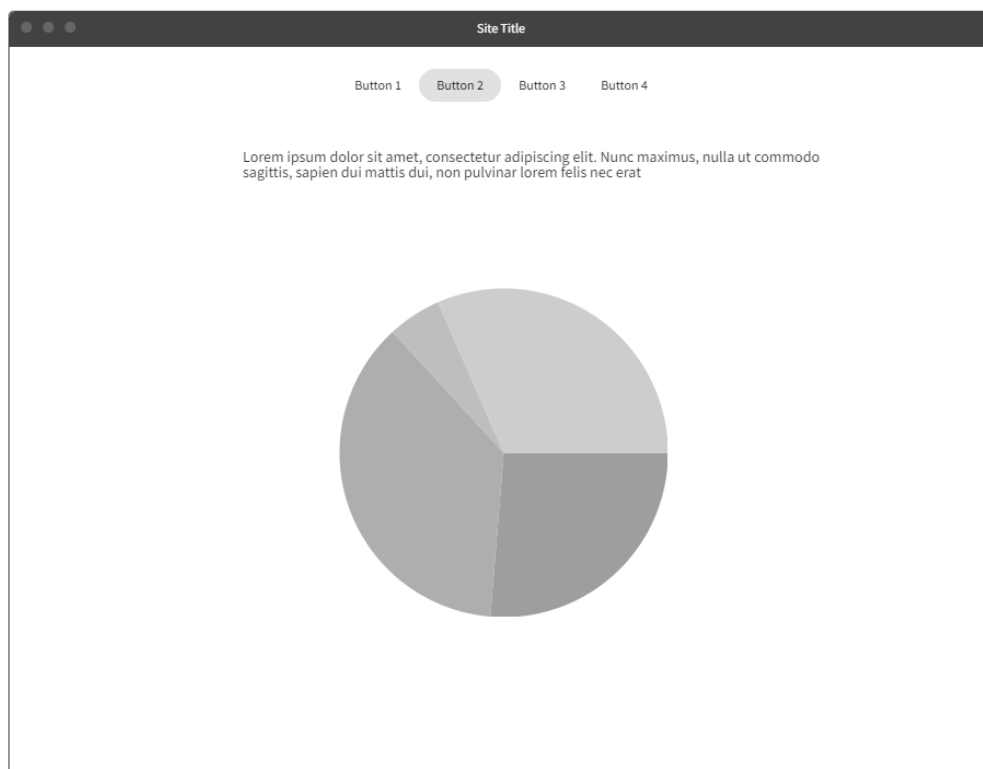
Na poslední podstránce se nachází graf pro národnosti sledujících ve formátu výšečového grafu.



Obrázek 12 – návrh hlavní stránky



Obrázek 13 – návrh 1. podstránky

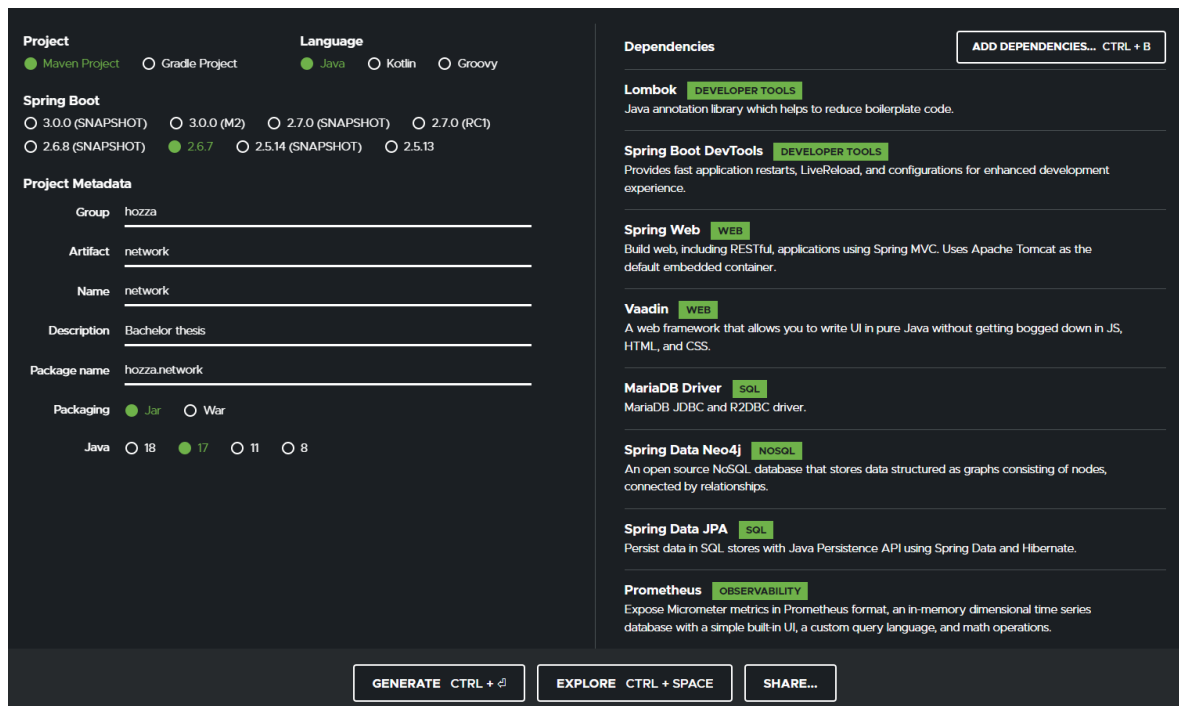


Obrázek 14 – návrh 2. podstránky

## 4.5 Implementace aplikace

### 4.5.1 Příprava

Projekt je psán ve frameworku Spring Boot, který má online nástroj pro generování struktury projektu na url <https://start.spring.io>. Zde se nakonfigurují veškeré základní závislosti a verze.



Obrázek 15 – generátor Spring Boot projektu

Po nakonfigurování a vygenerování struktury byl projekt propojen s GitLabem. Pro automatizování sestavení a vydávání aplikace byl vytvořen soubor `.gitlab-ci.yml`, ve kterém se definují jednotlivé úkoly pro běžce. A také `Dockerfile`, který má za úkol spustit aplikaci v kontejneru.

```
FROM openjdk:11-jre-slim-buster
ADD target/network-0.0.1.jar app.jar
RUN useradd -m myuser
USER myuser
EXPOSE 9696
CMD java -jar -Dspring.profiles.active=prod app.jar
```

Obrázek 16 – příklad Dockerfile

```
stages:
  - build
  - deploy

build:
  stage: build
  script:
    - ./mvnw install -Pproduction
    - docker build -t hozza/network .

deploy:
  stage: deploy
  script:
    - docker stop network_app 2> /dev/null || true && docker rm network_app 2> /dev/null || true && docker run -e TZ=Europe/Prague --net=host --restart=always -d --name network_app hozza/network
```

Obrázek 17 – příklad gitlab-ci.yml

Po vytvoření těchto souborů a první push operaci aplikace běží na předem nastavené doméně.

Dalším krokem je konfigurace samotné aplikace a jejích přístupů. V nejnovější verzi Spring Bootu probíhá veškerá konfigurace v souboru `/src/main/resources/application.properties`, zde bylo nakonfigurováno `spring.neo4j.uri`, `spring.neo4j.authentication.username`, `spring.neo4j.authentication.password`, `spring.neo4j.pool.max-connection-pool-size` a také přístupy k MariaDB `spring.datasource` konfiguraci.

```
# suppress inspection "SpringBootApplicationProperties" for whole file
server.port=${PORT:9696}
logging.level.org.atmosphere = warn
vaadin.pnpm-enabled=false
vaadin.compatibilityMode = false
vaadin.productionMode=false
management.metrics.export.prometheus.enabled=true
management.endpoints.web.exposure.include=prometheus
management.metrics.distribution.percentiles-histogram.http.server.requests=true
management.metrics.enable.jvm=true
spring.neo4j.uri=bolt://localhost:7687
spring.neo4j.authentication.username=neo4j
spring.neo4j.authentication.password=heslo tu
spring.neo4j.pool.max-connection-pool-size=600
spring.main.allow-bean-definition-overriding=true
spring.jpa.open-in-view=false
spring.mustache.check-template-location=false
spring.task.scheduling.pool.size=10
```

Obrázek 18 – příklad konfigurace Spring Boot aplikace

Poslední konfigurace byla nastavena v souboru `NetworkApplication.java`, kde je nutné povolit `@EnableNeo4jRepositories` a `@EnableAsync` pro fungování Neo4J modelů a asynchronního získávání dat z pohledů.

#### 4.5.2 Implementace databází

Ve Spring Bootu se databázové entity vytvářejí do separátních tříd s využitím anotací, těm se poté v repositářových rozhraních přiřazují databázové dotazy. Každý repositář komunikující s Neo4J databází musí rozšiřovat rozhraní `Neo4jRepository<T, ID>`, s databází se vše propojí automaticky. V případě MariaDB veškeré repositáře musí rozšiřovat rozhraní `JpaRepository<T, ID>` a taktéž se vše propojí automaticky. Po spuštění programu lze v databázi vidět vytvořené tabulky i s omezeními, které se entitám nastavují.

```
package hozza.network.models.neo4j.entities;

import ...

@Node("Person")
public class NeoPerson
{
    @Id
    @GeneratedValue
    protected Long id;

    protected String name;

    protected LocalDate dateAdded;

    protected NeoPerson()
    {
    }

    public NeoPerson(String name, LocalDate date)
    {
        this.name = name;
        this.dateAdded = date;
    }
}
```

Obrázek 19 – příklad entity

### 4.5.3 Sběrání dat

V cron balíčku byla vytvořena třída TwitchService, která má anotaci `@Component` a poté pomocí Dependency Injection dostane potřebné repositáře. V ní byla založena metoda `statCron()` označená anotací `@Scheduled`, díky které se spouští v definovaném čase bez nutnosti využívat linuxový crontab.

Dopředu byl nachystán seznam všech relevantních československých streamerů a každou hodinu probíhá sběr dat z twitch API – kdo koho v jaký čas sledoval. Toto se přes multivláknový algoritmus ukládá do databáze. Pro Neo4J tento algoritmus trvá 3 minuty, pro MariaDB 5 minut. Tento čas je vysoce ovlivněn objektově relačním mapováním, kdy se v paměti neukládají pouze textové řetězce a čas, ale celá objektová entita představující řádek v databázi.

```
@Scheduled(cron = "0 0 * * *")
public void statCron()
{
    this.possibleEntries.set(0);
    this.savedEntries.set(0);
    final ZonedDateTime starting = ZonedDateTime.now(ZoneId.of("CET"));
    System.out.println("[NE043] CRON STARTED "+starting);

    List<String> viewers = Collections.synchronizedList(personRepository.findAllNames());
    List<NeoStreamer> streamers = Collections.synchronizedList(streamerRepository.findAll());
    List<NeoPerson> sharedPersonCollection = Collections.synchronizedList(new ArrayList<>());

    Neo4JThreadManager manager = new Neo4JThreadManager();

    List<Callable<NeoWatchingDate>> taskList = new ArrayList<>();

    System.out.println("[NE043] PREPARATIONS DONE "+ZonedDateTime.now(ZoneId.of("CET")));

    for(NeoStreamer streamer : streamers)
    {
        NeoTwitchServiceCallable r = new NeoTwitchServiceCallable(streamer, viewers, starting, personRepository, watchingDateRepository, sharedPersonCollection, possibleEntries, savedEntries);
        taskList.add(r);
    }

    var futures :List<Future<NeoWatchingDate>> = manager.invokeCallableList(taskList);
    if(futures == null)
    {
        System.out.println("[NE043] Došlo k erroru při threadingu!");
        return;
    }
    futures.forEach(future -> {
        try {
            watchingDateRepository.save(future.get());
        } catch (InterruptedException | ExecutionException e) {
            System.out.println("[NE043] Došlo k erroru při vkládání do databáze!");
        }
    });
    System.out.println("[NE043] STREAMERS SAVED | POSSIBLE ENTRIES: "+this.possibleEntries.get()*" | GOT: "+this.savedEntries.get()*" | COMPLETED: "+ZonedDateTime.now(ZoneId.of("CET")));
}
```

Obrázek 20 – cron sběrnice dat

Každý sběr dat je zalogován s informací, kolik dat bylo uloženo a kolik bylo očekáváno k uložení na základě API. To vše je nutné v případě, kdy by došlo k chybě se synchronizací mezi vlákny.

#### 4.5.4 Deadlock detekce

Tím, že se pro sběr a vykreslování dat využívá asynchronních operací, je nutné implementovat Deadlock detektor. Ten v případě zaznamenání uváznutí situaci vyřeší a ohlásí. Detektor se spouští každých pět vteřin a ověří, zdali nejsou vlákna uváznutá.

```
public class DeadlockMonitorTask implements Runnable
{
    @Override
    public void run() {
        ThreadMXBean bean = ManagementFactory.getThreadMXBean();
        long[] ids = bean.findMonitorDeadlockedThreads();

        if(ids != null)
        {
            ThreadInfo[] threadInfo = bean.getThreadInfo(ids);

            for (ThreadInfo threadInfo1 : threadInfo)
            {
                System.out.println(threadInfo1.getThreadId());

                System.out.println(threadInfo1.getThreadName());

                System.out.println(threadInfo1.getLockName());

                System.out.println(threadInfo1.getLockOwnerId());

                System.out.println(threadInfo1.getLockOwnerName());
            }
        }
    }
}
```

Obrázek 21 – funkčnost deadlock monitoru

## 4.5.5 Optimalizace MariaDB dotazů

### 4.5.5.1 Návrhy tabulek

Pro vyšší rychlost zpracovávání dotazů je nutné databázi samotnou i příslušné dotazy optimalizovat. Optimalizace databáze probíhá pomocí indexů, které nám umožňují rychlé vyhledávání v tabulkách. Indexy by se měly používat na sloupce, podle kterých se nejčastěji vyhledává nebo se spojují tabulky.

V případě této aplikace se v tabulce person často vyhledává podle jména, převážně u ukládání záznamů do tabulek při sběru dat. Dalším důležitým indexem je v tabulce watching sloupec date\_time\_watched, podle kterého se vyhledávají data při vykreslování. Dále byl vytvořen dvojitý index na tabulce watching pro sloupce person a streamer, protože se jedná o sloupce, na kterých probíhají velké joiny.

Rychlostní vliv má i nedodržení normálních forem, kdy ukládáme data do jednoho sloupce, přičemž by správně mělo být sloupců více. Typickým příkladem je adresa, správně by měla být rozdělena na ulici, město, PSČ a číslo popisné. Protože v případě vyhledávání všech obyvatel města Zlína by dotaz byl neefektivní.

### 4.5.5.2 Systémové prostředky

Dalším důležitým poznatkem v rámci optimalizace databází je dostatečné přidělení systémových prostředků databázi. V případě, že by databáze překročila přiřazené prostředky, může systém databázi shodit.

#### 4.5.5.3 Optimalizace dotazů

Nejčastějším problémem při pomalých databázích jsou subdotazy. Pokud se lze v dotazu vyhnout vnořeným dotazům, je lepší se jim vždy vyhnout. U malých tabulek vliv není znatelný, nicméně již u několika tisíců záznamů je vliv na rychlost vysoce znatelný.

Dalším problémem je nspecifikování řazení u agregací. Pokud není nutné řešit pořadí výsledků vytažených z databáze, měl by dotaz vždy zahrnovat *ORDER BY NULL*. Tím se sníží časová zátěž. Toto lze vyzorovat prostým zavoláním *EXPLAIN* na dotaz, kdy se zjistí, zdali databáze používá „filesort“.

Velký rychlostní rozdíl lze také vyzorovat u převádění sloupce typu *datetime*. Pokud jsou použité interní funkce na převod, je dotaz pomalejší. Například je nutné vytáhnout z databáze dnešní výsledky, interními funkcemi by se jednalo o *WHERE CAST(datetime AS DATE) = CURDATE()*, což je mnohem pomalejší, než použít *WHERE datetime BETWEEN NOW() - INTERVAL (HOUR (NOW()) \* 60 \* 60 + MINUTE (NOW()) \* 60 + SECOND (NOW())) SECOND AND NOW()*.

#### 4.5.6 Tvorba pohledů

Za pomoci Vaadin Flow lze vytvořit pohled jako třídu a pomocí anotací přidělit, na jaké URL se bude zobrazovat a definovat, které role v systému do ní mají přístup.

Pomocí anotace *@Autowire* lze také konstruktorem předat službu, která získává data.

V případě asynchronního získávání dat musí pohled prepisovat metody *onDetach* a *onAttach*, aby se vlákna mohla inicializovat a v případě vypnutí pohledu přerušit. Poté stačí jen pomocí *pusher* metody předat objekt UI, který řeší změny v rozhraní.

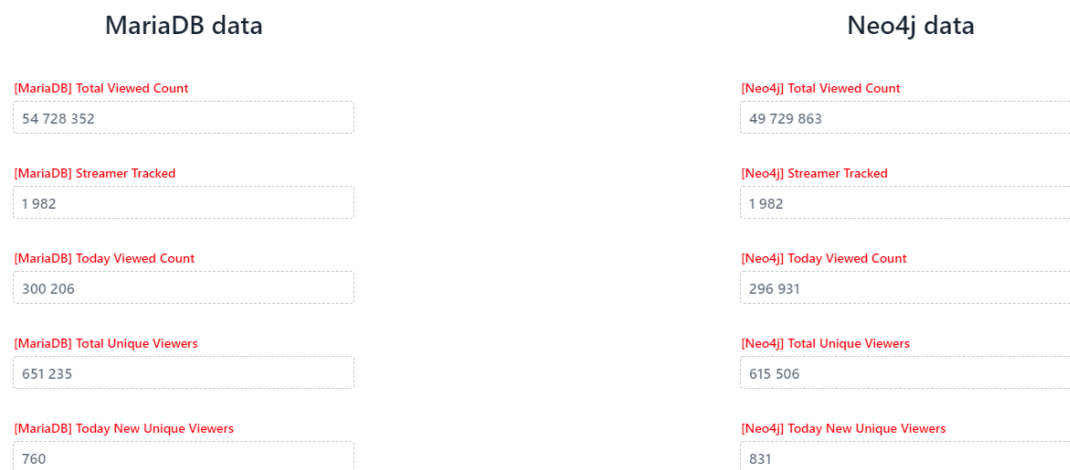
### 4.6 Výstupy aplikace

Aplikace zobrazuje data o velikosti databází a zpracované reaktivní grafy s realtime kalkulací.

#### 4.6.1 Velikost databází

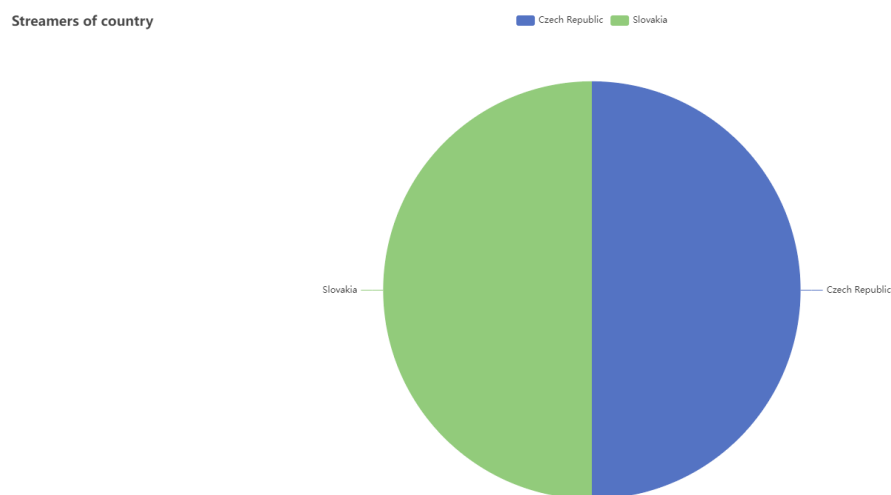
Během automatických úloh probíhajících několik měsíců bylo nasbíráno pro 1982 streamerů dohromady 100 milionů záznamů o sledování s průměrným denním přírůstkem 400 tisíc záznamů denně. Tento údaj je dostupný na hlavní stránce programu.

## Database statistics



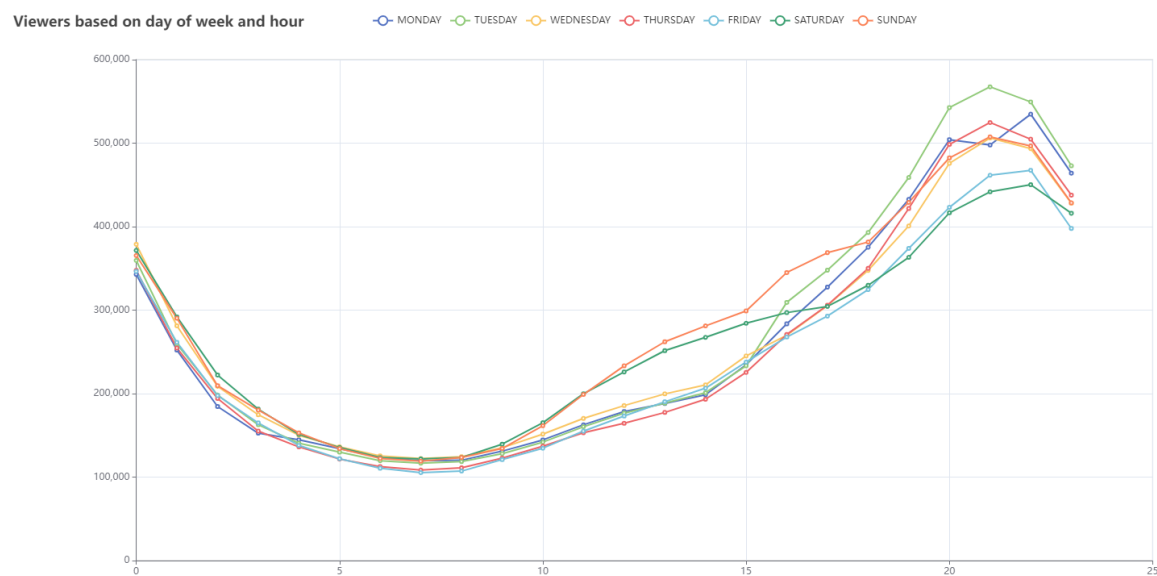
Obrázek 22 – stav databází k datu 19. 5. 2022

V další sekci aplikace se nachází národnostní rozdělení streamerů ve formě koláčového grafu.



Obrázek 23 – národnostní rozdělení streamerů

V poslední sekci webových stránek se nachází liniový graf, který zobrazuje aktivitu sledujících dle dnů v týdnu a hodin.



Obrázek 24 – liniový graf aktivity

## ZÁVĚR

Tato práce vysvětlila pojem Big Data, jeho problémy a překážky v moderním světě týkajících se jejich hromadného zpracovávání a interpretování. Pro zpracovávání dat bylo nejprve nutné nastudovat problematiku spojenou s databázovými systémy, jejich charakteristikami a na základě informací identifikovat jejich přínos. V rámci interpretace dat bylo vysvětleno fungování nejpoužívanějších typů grafů i případy použití. Práce zdůrazňuje vliv zpracování Big Data ve firemní oblasti.

Praktická část se věnuje zejména implementaci samotné aplikace. Celý vývoj proběhl od stanovení požadavků, modelování jednotlivých součástí systémů přes implementaci systému a jeho konfiguraci až po samotné zabezpečení všech komponent a následné nasazení výsledného produktu.

V rámci implementace aplikace zvládá

- shromažďovat data z aplikace Twitch.tv
- ukládat data do vícero databází
- interpretovat data pomocí reaktivních grafů
- hlídat asynchronní operace za pomoci Deadlock Monitoru
- asynchronní předávání dat do pohledů

V rámci budoucího vylepšení by bylo možné implementovat následující vylepšení:

- rozšíření množství typů grafů
- využití komplexnějšího zpracovávání nezávislého na relačních databázích
- použití systému pro optimalizaci datových struktur (Apache Spark, Apache Hadoop)
- vylepšení grafického prostředí, zejména zaměření na UX/UI

**SEZNAM POUŽITÉ LITERATURY**

- [1] LUTKEVICH, Ben. Database (DB). *TechTarget* [online]. c2005-2022 [cit. 2022-05-19]. Dostupné z: <https://www.techtarget.com/searchdatamanagement/definition/database>
- [2] What is SQL ?. *SQL course* [online]. [cit. 2022-05-19]. Dostupné z: <https://www.sqlcourse.com/beginner-course/what-is-sql/>
- [3] JANKOV, Tonino. MariaDB vs MySQL: A Database Technologies Rundown. *KINSTA BLOG* [online]. 2022 [cit. 2022-05-22]. Dostupné z: <https://kinsta.com/blog/mariadb-vs-mysql/>
- [4] GRAY, Jim. *The Transaction Concept: Virtues and Limitations* [online]. 19333 Vallco Parkway, Cupertino CA 95014, 1981 [cit. 2022-05-22]. Dostupné z: <http://jimgray.azurewebsites.net/papers/thetransactionconcept.pdf>. Tandem Computers Incorporated.
- [5] HANZLÍK, Roman. *Big Data Ecosystem* [online]. Univerzita Tomáše Bati ve Zlíně, 2019 [cit. 2022-05-22]. Dostupné z: [https://digilib.k.utb.cz/bitstream/handle/10563/44471/hanzlík\\_2019\\_dp.pdf](https://digilib.k.utb.cz/bitstream/handle/10563/44471/hanzlík_2019_dp.pdf). Bakalářská práce. Univerzita Tomáše Bati ve Zlíně. Vedoucí práce Doc. Ing. Roman Šenkeřík, Ph.D.
- [6] KOBIELUS, James. MongoDB Drives NoSQL More Deeply into Enterprise Opportunities. *Wikibon* [online]. 2018 [cit. 2022-05-22]. Dostupné z: <https://wikibon.com/mongodb-drives-nosql-deeply-enterprise-opportunities/>
- [7] What is a Graph Database?. *Neo4j* [online]. [cit. 2022-05-22]. Dostupné z: <https://neo4j.com/developer/graph-database/>

- [8] Big Data: What is it and why it matters. SAS [online]. c2022 [cit. 2022-05-19]. Dostupné z: [https://www.sas.com/en\\_us/insights/big-data/what-is-big-data.html](https://www.sas.com/en_us/insights/big-data/what-is-big-data.html)
- [9] CAVANILLAS, Jose Maria, Edward CURRY a Wolfgang WAHLSTER. *New Horizons for a Data-Driven Economy: A Roadmap for Usage and Exploitation of Big Data in Europe* [online]. Springer, 2016, s. 123-124 [cit. 2022-05-19]. ISBN 978-3-319-21569-3. Dostupné z: [https://www.researchgate.net/publication/280625241\\_New\\_Horizons\\_for\\_a\\_Data-Driven\\_Economy\\_A\\_Roadmap\\_for\\_Usage\\_and\\_Exploitation\\_of\\_Big\\_Data\\_in\\_Europe](https://www.researchgate.net/publication/280625241_New_Horizons_for_a_Data-Driven_Economy_A_Roadmap_for_Usage_and_Exploitation_of_Big_Data_in_Europe)
- [10] ABUBAKAR, Mohammed. What is Ubuntu ?. *How-To Geek* [online]. 2021 [cit. 2022-05-19]. Dostupné z: <https://www.howtogeek.com/763775/what-is-ubuntu/>
- [11] What is NGINX ?. *NGINX* [online]. [cit. 2022-05-19]. Dostupné z: <https://www.nginx.com/resources/glossary/nginx/>
- [12] CAREY, Scott. What is Docker? The spark for the container revolution. *InfoWorld* [online]. 2021 [cit. 2022-05-19]. Dostupné z: <https://www.infoworld.com/article/3204171/what-is-docker-the-spark-for-the-container-revolution.html>
- [13] Java. *TechTerms* [online]. 2012 [cit. 2022-05-19]. Dostupné z: <https://techterms.com/definition/java>
- [14] Spring Framework. *Spring* [online]. c2022 [cit. 2022-05-19]. Dostupné z: <https://spring.io/projects/spring-framework>
- [15] Flow. *Vaadin* [online]. 2021 [cit. 2022-05-19]. Dostupné z: <https://vaadin.com/docs/latest/flow/overview>

- [16] KELLEY, Karin. What is GitLab and How to Use It?. *Simplilearn* [online]. 2022 [cit. 2022-05-19]. Dostupné z: <https://www.simplilearn.com/tutorials/git-tutorial/what-is-gitlab>
- [17] What is DevOps?. *GitLab* [online]. c2022 [cit. 2022-05-19]. Dostupné z: <https://about.gitlab.com/topics/devops/>
- [18] Graph – Definition with Examples: What is a Graph?. *SplashLearn* [online]. c2022 [cit. 2022-05-19]. Dostupné z: <https://www.splashlearn.com/math-vocabulary/geometry/graph>
- [19] GOLDENBERG, Dima. Social Network Analysis: From Graph Theory to Applications with Python. *Towards Data Science* [online]. 2021 [cit. 2022-05-19]. Dostupné z: <https://towardsdatascience.com/social-network-analysis-from-theory-to-applications-with-python-d12e9a34c2c7>
- [20] BEATRICE, Adilin. ALL ABOUT THE BASICS OF BIG DATA: HISTORY, TYPES AND APPLICATIONS. *Analytics Insight* [online]. 2021 [cit. 2022-05-22]. Dostupné z: <https://www.analyticsinsight.net/all-about-the-basics-of-big-data-history-types-and-applications/>
- [21] GAUR, Chandan. Top 6 Big Data Challenges and Solutions to Overcome. *Xenonstack* [online]. 2020 [cit. 2022-05-22]. Dostupné z: <https://www.xenonstack.com/insights/big-data-challenges>
- [22] Big data visualization: what it is, techniques and best tools. *Rockcontent* [online]. 2020 [cit. 2022-05-22]. Dostupné z: <https://rockcontent.com/blog/big-data-visualization/>

**SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK**

SQL	Structured Query Language
LAMP	Linux, Apache, MariaDB, PHP
ACID	Atomicity, Consistency, Isolation, Durability
NoSQL	Not only SQL
CAP	Consistency, Availability, Partitioning
JSON	JavaScript Object Notation
BSON	Binary JSON
RFID	Radio Frequency Identification
HDFS	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
JVM	Java Virtual Machine
MVC	Model, View, Controller
HTML	Hypertext Markup Language
API	Application Programming Interface
DevOps	Development and Operations
DDoS	Distributed Denial of Service
LTS	Long Term Support
XHR	XML Http Request
URL	Uniform Resource Locator
XML	Extensible Markup Language
CI/CD	Continuous Integration / Continuous Delivery
UI/UX	User Interface / User Experience

**SEZNAM OBRÁZKŮ**

Obrázek 1 – mapa světa jako heatmap [22] .....	21
Obrázek 2 – Vaadin Flow [15] .....	25
Obrázek 3 – sloupcový graf [18] .....	27
Obrázek 4 – liniový graf [18] .....	27
Obrázek 5 – výsečový graf [18].....	28
Obrázek 6 – síťový graf [19] .....	28
Obrázek 7 – port forwarding nginx konfigurace .....	35
Obrázek 8 – phpmyadmin nginx konfigurace.....	36
Obrázek 9 – neo4j nginx konfigurace .....	37
Obrázek 10 – gitlab nginx konfigurace.....	38
Obrázek 11 – ER diagram.....	40
Obrázek 12 – návrh hlavní stránky .....	41
Obrázek 13 – návrh 1. podstránky .....	41
Obrázek 14 – návrh 2. podstránky .....	42
Obrázek 15 – generátor Spring Boot projektu .....	43
Obrázek 16 – příklad Dockerfile .....	43
Obrázek 17 – příklad gitlab-ci.yml .....	44
Obrázek 18 – příklad konfigurace Spring Boot aplikace.....	44
Obrázek 19 – příklad entity .....	45
Obrázek 20 – cron sběrnice dat.....	46
Obrázek 21 – funkčnost deadlock monitoru .....	47
Obrázek 22 – stav databázi k datu 19. 5. 2022 .....	50
Obrázek 23 – národnostní rozdělení streamerů .....	50
Obrázek 24 – liniový graf aktivity .....	51

## SEZNAM PŘÍLOH

Příloha P I: Elektronická verze bakalářské práce, zdrojový kód aplikace a manuál

# **PŘÍLOHA P I: ELEKTRONICKÁ VERZE BAKALÁŘSKÉ PRÁCE, ZDROJOVÝ KÓD APLIKACE A MANUÁL**

Složková struktura přílohy:

```
/
  fulltext.pdf
  /aplikace
    MANUAL.md – příručka k aplikaci
  /network
    /src – zdrojové kódy
    /.mvn – package manager
    .gitignore
    Dockerfile
    HELP.md
    mvnw
    mvnw.cmd
    pom.xml
```