VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

# DIPLOMOVÁ PRÁCE

Brno, 2016 Bc. Ján Gardian


# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

## FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

## ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS


# SERVEROVÁ APLIKACE PRO ZPRACOVÁNÍ DAT Z DATABÁZE MYSQL A JEJICH INTERPRETACI
SERVER APPLICATION TO PROCESS DATA FROM A MYSQL DATABASE AND THEIR INTERPRETATION

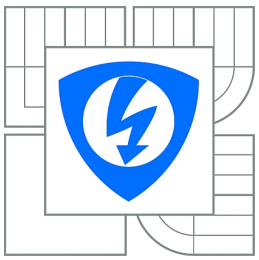
**DIPLOMOVÁ PRÁCE**
MASTER'S THESIS


**AUTOR PRÁCE** — **Bc. Ján Gardian**
AUTHOR

**VEDOUCÍ PRÁCE** — **Ing. Rastislav Červenák**
SUPERVISOR



**BRNO 2016**



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**Fakulta elektrotechniky a komunikačních technologií**

**Ústav telekomunikací**

# Diplomová práce

magisterský navazující studijní obor
**Telekomunikační a informační technika**

***Student:*** Bc. Ján Gardian ***ID:*** 165839
***Ročník:*** 2 ***Akademický rok:*** 2015/2016

**NÁZEV TÉMATU:**

## Serverová aplikace pro zpracování dat z databáze MySQL a jejich interpretaci

**POKYNY PRO VYPRACOVÁNÍ:**

Diplomová práce se zabýva vytvořením serverové aplikace pro zpracování dat z databáze MySQL, jejich interpretaci pomocí interaktivní webové aplikace a back-end podporou pro mobilní aplikace. V teoretické časti bude rozebrána problematika zpracování dat z databáze pro velké množství požadavku v reálnem čase, tak aby byla dostupná data pro uživatele služby co nejvíc aktuální a to v krátkem čase. V praktické časti bude vytvořené rozhraní služby pro zpracovaní dat z datábaze MySQL. Toto rozhraní bude slouži pro aplikaci, která bude vytvořená pro internetový prohlížeč a bude umožnovat zobrazit data o operátorech a parametrech síti v přehledných interaktivních grafech. Uživatel si bude moct vybírat a kombinovat vstupní data jako lokalitu, operátora a datovou technologii, na základě kterých se grafy vykreslí. Rozhraní služby bude sloužit taky jako služba pro mobilní aplikace. Aplikace bude navrhnutá tak, aby data z databáze zpracovávala průběžne s ohledem na výkon serveru. Celková funkčnost celého systému bude ověřena na reálné databázi.

**DOPORUČENÁ LITERATURA:**

[1] FREEMAN, E.: Head First HTML5 Programming: Building Web Apps with JavaScript. Sebastopol: O'Reilly Media, 2011, ISBN: 978-1449390549.

***Termín zadání:*** 1.2.2016 ***Termín odevzdání:*** 25.5.2016

***Vedoucí práce:*** Ing. Rastislav Červenák
***Konzultanti diplomové práce:***

**doc. Ing. Jiří Mišurec, CSc.**
*Předseda oborové rady*

## ABSTRAKT

Diplomová práca je o vytvorení serverovej aplikácie, ktorá spracuje a interpretuje dáta z databázy. Cieľom takejto aplikácie je dokázať spracovať veľký počet požiadavkou v reálnom čase. Poskytnutá databáza, s ktorou aplikácia pracuje obsahuje merania rýchlosti a kvality mobilného spojenia pomocou rádiových technológií ponúkaných jednotlivými mobilnými operátormi. Merania pochádzajú od užívateľov z celého sveta a množstvo meraní neustále rastie. Výsledkom práce je preto naprogramovaná serverová aplikácia, ktorá sa prispôsobuje narastajúcemu množstvu dát pomocou agregácie. Metóda agregácie a použitie indexov v tabuľkách databázy sú bližšie rozobrané v teoretickej časti diplomovej práce. Hlavne definícia indexov v databáze priniesla značné zrýchlenie spracovania databázových požiadaviek. Finálnym produktom diplomovej práce je aplikácia, ktorá sa skladá z troch komponentov: serverová aplikácia vykonávajúca agregáciu, webová stránka interpretujúca namerané dáta a back-end rozhranie taktiež poskytujúce namerané dáta. Webová stránka reprezentuje dáta vo forme grafov pre jednotlivé krajiny a rádiové technológie. Webová adresa prezentačnej stránky a návod na použitie sú poskytnuté v štvrtej kapitole diplomovej práce. Záverom sú uskutočnené testy rýchlosti odozvy vytvorenej aplikácie, ktoré potvrdzujú efektivitu vybraných a opísaných metód na zrýchlenie práce s databázou.



## KĽÚČOVÉ SLOVÁ

Databáza, MySQL, aggregácia, index, transakcia, rozhranie, aplikácia, webová stránka, API rozhranie, graf, databázová požiadavka, rýchlosť odozvy



## ABSTRACT

Diploma thesis is about creating server application that process and interprets data from the database. Main aim of such application is able to process a large number of database requests in real-time environment. Provided database contains records of measuring download speed and quality of mobile connection via different radio technology from various providers. Those measured data are sent from users all around the world and amount of data collected is still growing. Therefore created server application can adapt to increasing size of database thanks to aggregation. This method of aggregation and use of index in database tables are further discussed in the theoretical part. Mainly putting indexes in tables produce significant acceleration of processing database requests. Final product of this thesis is an application that consist of three components: a server application running aggregation, website that interprets measured data and back-end interface providing measured data as well. Data at the website are presented in form of graphs for different countries and used radio technologies. Web address and user manual for finished applications are provided in the fourth chapter of diploma thesis. In the last part of thesis are performed various speed tests of programmed application that confirm the effectiveness of selected and described methods to accelerate work with the database.



## KEYWORDS

Database, MySQL, aggregation, index, transaction, interface, application, website, API, graph, database request, response time

GARDIAN, Ján : diplomová práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2016. 61 s. Vedúci práce bol Ing. Rastislav Červenák

# PREHLÁSENIE

Prehlasujem, že som svoju diplomovoú prácu na tému „“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/nebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., o právu autorském, o právoch súvisejúcich s právom autorským a o zmeně niektorých zákonov (autorský zákon), vo znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhé, hlavy VI. diel 4 Trestného zákoníka č. 40/2009 Sb.

Brno ..... .....

(podpis autora)

## POĎAKOVANIE

Rád by som poďakoval vedúcemu diplomovej práce za účinnú metodickú, pedagogickú a odbornú pomoc a ďalšie cenné rady pri spracovaní mojej diplomovej práce.


Brno ..... .....

(podpis autora)


Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

## POĎAKOVANIE

Výzkum popsaný v tejto diplomovej práci bol realizovaný v laboratóriách podporených projektom SIX; registračné číslo CZ.1.05/2.1.00/03.0072, operačný program Výzkum a vývoj pro inovace.

Brno ..... .....
(podpis autora)



# OBSAH

# ZOZNAM OBRÁZKOV

# ZOZNAM TABULIEK

# ÚVOD

Hlavnou myšlienkou diplomovej práce je reprezentácia dát v databáze bežnému užívateľovi. Dáta sú tvorené záznamami o kvalite internetového pripojenia mobilného zariadenia v rôznych krajinách a pomocou rozdielnych metód mobilného spojenia. Spomenutá prezentácia je vo forme webovej aplikácie, ktorá spracuje a poskytne dáta v reálnom čase.

V prvej kapitole sa približuje pojem databázy. Opisujú sa jej vlastnosti a systém správy. Viac sa táto kapitola venuje vlastnostiam databázy MySQL, ktorá je nasadená na ukladanie meraných dát. Uvádza hlavné dôvody, prečo je MySQL tak rozšírená a stručne rozoberá jej typy tabuliek a pomocou čoho sa dá pracovať s touto databázou.

Druhá kapitola nadväzuje na databázy a spracovanie dát v reálnom čase a uvádza všeobecnú definíciu databáz, ktoré dokážu odpovedať na požiadavky v reálnom čase. Bližšie spracuje fungovanie a vlastnosti samotnej transakcie, ktorá tvorí kľúčovú súčasť spracovania dát v obmedzenom čase. Následne rozoberá metódy indexov a agregácie v databáze za účelom docieliť čo najrýchlejšiu odozvu databázy pre požiadavky v reálnom čase. Obidve metódy spracované v tejto kapitole sú použité pri návrhu a realizácii aplikácii v praktickej časti diplomovej práce.

Nasledujúca kapitola sa venuje všeobecnému opisu aplikácie, jej požiadaviek zo strany užívateľa a foriem prezentácie. Celá aplikácia je rozdelená na serverovú a webovú časť. Medzi hlavnú formu prezentácie webovej aplikácie patrí vykreslenie nameraných údajov do čitateľných grafov podľa stanovených kritérií. Okrem webovej aplikácie sa v návrhu rieši aj poskytovanie dát z databázy vo forme back-end podpory pre mobilné aplikácie.

V štvrtej kapitole sa spracuje konkrétny návrh aplikácie, jej realizácia, výber programovacieho jazyka a dostupných systémov, ktoré dovoľujú rýchlu komunikáciu s databázou. Výsledný návrh je rozdelený podľa zmienenej funkčnosti na agregačnú serverovú časť, back-end podporu a prezentáciu formou webovej stránky. Opis jednotlivých častí je vo forme vývojového diagramu. V tejto kapitol sa taktiež uvádzajú potrebné rozhrania a programy nutné pre beh aplikácie na servery a manuál na jej prípadnú konfiguráciu a použitie.

Posledná kapitola obsahuje konkrétne testovanie a porovnanie výsledkov. Merania sú predovšetkým na rýchlosti odozvy a celkového fungovania pre každú časť aplikácie zmienenú v štvrtej kapitole.

Záver práce sumarizuje dosiahnuté výsledky a štatistiky rýchlosti odozvy aplikácie a dosiahnutého vyťaženia, ktoré môže spôsobovať. Následne sú navrhnuté nápady na rozšírenie práce o ďalšie moduly prezentácie a doladenie podpory na všetkých platformách.

# 1 DATABÁZA JEJ VLASTNOSTI A TYPY

Databáza predstavuje úložisko dát, vďaka ktorému môžu aplikácie a programy pracovať s veľkým množstvom dát efektívnejšie a rýchlejšie. Túto vlastnosť zaručuje organizovaná štruktúra ukladania, kde sú jednotlivé dáta rozdelené do tabuliek podľa využitia. Každá tabuľka má definované typy dát, ktoré sú v nej uložené, a spoločné transakcie, ktoré tieto dáta spájajú. Pri vzniku databázy je dôležité správne vybrať typy dát, ich vlastnosti a identifikátory a vytvoriť model, do ktorého sa môžu dáta najefektívnejšie ukladať a zároveň aj čerpať. Podľa modelu ukladania dát poznáme tieto typy databáz[1]:

- sieťová databáza,
- hierarchická databáza,
- relačná databáza,
- objektová databáza,
- objektová relačná databáza,
- korelačná databáza.

## 1.1 Databázový spravovací systém

Celkovú správu nad dátami v databáze zaručuje databázový spravovací systém (*DMBS - Database Management System*). Databázový spravovací systém je aplikácia spájajúca užívateľské požiadavky, ostatné aplikácie, prísun a analýzy dát so samotnou databázou. Pomocou DMBS definujeme štruktúru databázy pri vzniku, spravujeme ju a na základe požiadaviek z nej vyberáme hľadané dáta alebo pridávame nové. Tento systém sa postupne vyvíjal a zdokonaľoval až sa dostal k súčasným najznámejším spravovacím systémom ako sú: MySQL, PostgreSQL, Oracle, Sybase, Microsoft SQL Server a pod. Najviac používaným modelom v týchto systémoch sa stal relačný model, kde sú dáta identifikované na základe vybraných kľúčov a pomocou nich vytvárajú vzťahy[1]. Príkladom kľúča v relačnom modelu databáze, ktorá obsahuje študentov na škole, môže byť ich študentské identifikačné číslo. Pre tento relačný model sa časom utvoril a zosúladil spoločný manipulačný jazyk SQL[1]. SQL jazyk pozostáva z množiny príkazov, ktorými definujeme, upravujeme a hľadáme dáta v databáze. V nasledujúcej sekcii sa viac priblíži konkrétny databázový spravovací systém MySQL.

[1]SQL(Structured Query Language) - jazyk na ovládanie a programovanie databázy. Pozostáva z príkazov vo forme programovacieho jazyka, ktoré dovoľujú výpis, vkladanie, upravovanie a hľadanie informácii v databáze.

## 1.2 MySQL databáza a jej vlastnosti

V diplomovej sa práci bude pracovať s už existujúcou databázou MySQL. MySQL patrí medzi najviac používané databázové systémy. Patrí, ako som už spomenul, medzi databázové systémy relačného modelu. Popularitu získal vďaka voľne dostupnému zdrojovému kódu, ktorý spadá pod otvorenú licenciu GNU GPL[2] a jeho rôzne dostupné variácie pre rôzne druhy využitia. Vývoj tohto systému bol sponzorovaný spoločnosťou MySQL AB a momentálne jej vlastníkom a hlavným sponzorom spoločnosťou Oracle[2]. MySQL sa časom stal vyhľadávaným databázovým systémov v rozrastajúcom sa odvetví webových aplikácii. Príkladom je veľmi známa webová aplikácia LAMP[3]. LAMP je čím ďalej tým viac používaný webovými vývojármi pre svoju dobrú štruktúru a otvorenému zdrojovému kódu, ktorý je neustále aktualizovaný a prispôsobuje sa novým trendom Internetu[3].

### 1.2.1 Vlastnosti MySQL

Medzi základné vlastnosti MySQL patria[3]:

1. **Škálovateľnosť a flexibilita** - Databáza je dostupná na väčšine počítačových platformách ako napríklad Windows, Linux, Solaris. Vďaka otvorenému kódu obsahuje veľký rozsah úprav podľa požiadavku užívateľov a vývojárov.
2. **Veľká výkonnosť** - Možnosť konfigurácie na rôzne typy požiadaviek pre rozdielne aplikácie. Databáza je prispôsobená podľa toho či je treba odpovedať na veľa požiadaviek v reálnom čase, spracovať vysokorýchlostné zaťaženie alebo poskytnúť objemné dáta.
3. **Vysoká dostupnosť** - Podobne ako v predošlom bode, MySQL je dostupné vo viacerých prevedeniach ako je napr. Master-Slave pre rýchle spracovanie, Cluster servery poskytujúce stabilitu a rýchlu obnovu a prepnutie.
4. **Zálohovanie** - Alebo pod iným názvom technológia replikácie. MySQL podporuje a doporučuje možnosť viacerých *Slave* staníc databázy pravidelne synchronizovaných s hlavnou databázou. Toto zabezpečuje lepšiu ochranu dát.
5. **Robustnosť** - MySQL ponúka jeden z najsilnejších transakčných databázových systémov, ktoré sú momentálne na trhu. Funkcie zahŕňajú kompletný ACID[4], podporu distribuovaných transakcií a spravovanie prístupu viacerými užívateľmi v jeden okamžik.

[2]GNU General Public License - globálne používaná otvorená softvérová licencia, ktorá zaručuje voľnú úpravu a zdieľanie softvéru

[3]LAMP(Linux, Apache, MySQL, PHP) - sústava vyladená a prispôsobená pre webové aplikácie. Pozostáva z operačného systému, webového servera, databázy a programovacieho jazyka

[4]ACID(Atomic, Consistent, Isolated, Durable) - množina vlastností, ktorá zaručuje, že databázové transakcie sú spoľahlivo spracované

6. **Široká webová podpora** - Táto databáza sa dnes už stala skoro štandardom pre webové aplikácie s vysokým dátovým tokom.
7. **Silná ochrana dát** - Bezpečnosť dát býva na prvom mieste a to isté platí aj pre MySQL. V ponuke je veľa bezpečnostných prvkov ako povinná autentifikácia užívateľov, použitie SSL certifikátov, rozdelenie práv podľa stupňa autorizácie a nakoniec schopnosti zálohovania a obnovy celej databázy.
8. **Neustály vývoj** - Stále nové aktualizácie a rozšírenia sú dostupné pre vývojárov webových a iných aplikácii, ktorí môžu používať rozdielne programovacie jazyky.
9. **Dobrá správa a podpora** - MySQL je ľahko dostupná databáza s jednoduchou inštaláciou a správou. Keďže sa jedná o softvér pod otvorenou licenciou, existuje nespočetne veľká komunita pre podporu a pomoc s touto databázou. Pre väčšie projekty sú dostupné už upravené databázové štruktúry na komoditnom hardvéri za nízke ceny.

### 1.2.2 Správca databázy a typy tabuliek

V srdci MySQL databáze leží takzvaný databázový ukladací správca, ktorý odpovedá za ukladanie a poskytovanie dát. MySQL ponúka viacero typov takýchto správcov podľa toho, čo architekti databázy budú vyžadovať a na čo bude databáza používaná. Medzi najznámejšie typy správcov a typy tabuliek, ktoré sa používajú, patria[4]:

- **MyISAM** - základný ukladací správca v MySQL, ktorý je rýchly, umožňuje kompresiu dát a *FULLTEXT* vyhľadávanie[5]
- **InnoDB** - Robustná databáza prispôsobená neustálym požiadavkám a transakciám. Veľmi často používaná pri aplikáciách, ktoré majú veľké množstvo požiadavkou v reálnom čase.
- **MERGE** - Zlučuje viacero identických MyISAM tabuliek, čo sa používa napríklad u *DSS(Decission Support System)* alebo *OLAP(Online Analytical Processing)*
- **MEMORY** - Tabuľky sú navrhovane na extrémne rýchle spracovanie požiadavkou. Preto je ich využitie vo forme virtuálnej pamäte pomocou SQL rozhrania.
- **ARCHIVE** -Názov sám naznačuje, že ide o tabuľku prispôsobenú primárne na archiváciu dát. Nepoužívajú sa často. Vyznačuje sa, ako všetky archivačné mechanizmy, tým, že úsporne využíva miesto na disku.
- **CSV** - Obľúbený a veľmi primitívny formát tabuliek, kde sú jednotlivé dáta separované jednoduchým znakom ako napríklad čiarka. Vývojári preferujú takto

[5]Možnosť vyhľadávať na základe celého textu uloženého v databáze, prehľadáva sa každé slovo a nie len kľúče a identifikátory.

jednoduchý formát často ako vstupné dáta pre aplikácie.

- **FEDERATED** - Definuje prístup k vzdialeným dátam tak, ako keby sa nachádzali na lokálnom počítači.
- **NDB Cluster**[6] - NDB Cluster umožňuje viacerým počítačom udržiavať ich používanú pamäť synchronizovanú, čo vedie k lepšej výkonnosti a prispôsobivosti.

Okrem spomenutých typov tabuliek a správcov pre tieto typy existuje v MySQL ešte aj rozdelenie správy databázy z pohľadu užívateľov. Z histórie MySQL bol príkazový riadok jediným nástrojom na konfiguráciu a správy databáze. Postupom doby sa nároky užívateľ zvyšovali a MySQL ponúka tieto grafické rozhrania[4]:

- **MySQL Administrator** - Umožňuje administrátorom konfiguráciu, monitorovanie a správu jednotlivých MySQL databázových serverov.
- **MySQL Query Browser** - Poskytuje návrhárom databáze a jednotlivým užívateľom grafické rozhranie, ktoré dovoľuje vidieť naraz viaceré výsledky databázových požiadavkou.
- **Configuration Wizard** - Taktiež ako *MySQL Administrator* dovoľuje konfiguráciu a nastavenie databázy a ponúka tiež vzory alebo príklady možných konfigurácii.
- **MySQL System Traz** - Je tvorené grafickým rozhraním na báze okien, ktoré ponúka správu databázových inštancii vrátane funkcii ako je zastavenie a spustenie databázy.

Keďže je MySQL databáza populárna a široko používaná vo svete informačných technológii časom sa vyvinuli na jej správu a ovládanie mnohé aplikácie. Tieto aplikácie sa zväčša rozdeľujú podľa toho, pod ktorým programovacím jazykom vznikli. Nazývajú sa *API*[7] a sprostredkúvajú funkcionalitu medzi užívateľskými aplikáciami a MySQL databázou[4]. Medzi najpoužívanejšie programovacie jazyk s vlastnými rozhraniami na komunikáciu s databázou patria:

- *C*,
- *C++*,
- *.NET*,
- *Perl*,
- *PHP*,
- *Python*,

[6] Cluster je skupina viacerých počítačov alebo úložísk, ktoré navonok pôsobia ako jeden celok.

[7] API - Application Programming Interface čo v preklade znamená aplikačné programovacie rozhranie, ktoré poskytuje zbierku funkcií a procedúr pre programátora na komunikáciu a pracovanie s hlavným programom alebo aplikáciou. Príkladom môže byť Google Mail API, ktoré poskytuje rovnaké funkcie a výsledky ako webová stránka, ale môže byť použité pri programovaní vlastného webového klienta.

- *Ruby.*

# 2 REAL-TIME DATABÁZA

Pojem *real-time* databáza označuje úložisko dát, ktoré je používané v reálnom čase na ukladanie a získavanie dát. Už prvé systémy, ktoré pracovali v reálnom čase, si spravovali dáta vo vlastných štruktúrach. Avšak neustálym rozvojom vznikol tlak na veľkosť a objem dát, ktoré môžu takéto autonómne systémy zvládať a vznikli tak špeciálne prispôsobené real-time databázy. Databázové spravovacie systémy *DMBS* ponúkajú organizovanú správu a zaobchádzanie s dátami. Postupným zlučovaním s aplikáciami a systémami, ktoré pracujú v reálnom čase vznikli takzvané databázové systémy v reálnom čase (*RTDBS - Real-time Database System*)[5].

Podobne ako iné databázové systémy aj RTDBS obsahuje funkcie, ktoré zabezpečujú spoľahlivé ukladanie, manipuláciu a poskytovanie dát. Okrem tohto sa databázový systém pracujúci v reálnom čase vyznačuje dôležitým faktorom, ktorým je časový limit na spracovanie požiadaviek. Príkladom takejto databázy môže byť automatický preklad telefónnych čísel na telefónnej ústredni, kedy rýchlosť prekladu predstavuje kvalitu poskytovanej telefónnej služby. Podobný príklad časovo náročného databázového systému je obchod na burze. Jednotlivé zmeny kurzov bývajú často krátke a obchodníci sa snažia dostať k týmto údajom v čo najkratšom čase[5].

Oproti klasickým databázovým systémom sa real-time databázy odlišujú vo viacerých smeroch. Real-time databázy majú rozdielne ciele pre výkon, schopnosť opravovať chyby a vzťahu k aplikáciám, ktoré tieto databázy používajú. Medzi hlavné faktory, podľa ktorých je RTDBS systém hodnotený, patria[5]:

- ako často transakcia dobehne v stanovenom čase,
- priemerné oneskorenie,
- priemerne spomalenie transakcie,
- cena spôsobená oneskorenými transakciami,
- konzistenciu dát z databázy k užívateľom (dáta musia korešpondovať s reálnymi dátami vo svete),
- konzistenciu dát zo sveta (dáta musia byť aktualizované v rovnaký čas)

## 2.1 Transakcie v reálnom čase

Pri tvorbe systémov, ktoré budú pracovať v reálnom čase, vývojári určujú jednotlivé hodnoty. Jednou z najdôležitejších hodnôt je špecifikovanie maximálneho časového intervalu behu jednotlivej transakcie v systéme alebo maximálneho možného oneskorenia. Transakcie v reálnom čase alebo real-time transakcie sú práve také, ktoré majú takéto oneskorenie definované a dávajú veľkú váhu na spracovanie transakcie v potrebnom čase. Pokiaľ sa neuvažuje ideálny databázový systém, kde sa všetky

transakcie vykonajú do stanoveného oneskorenia, musí sa vo výpočtoch a pri konštrukcii systému rátať s takzvanými oneskorenými transakciami. Pre rôzne spracovanie transakcií sa prideľuje jednotlivým transakciám číselná hodnota vo forme známky, ktorá môže svoju hodnotu v čase meniť[7]:

- Známka má kladnú hodnotu. V tomto prípade sa systém snaží vykonať oneskorenú transakciu, aj napriek tomu, že celkový výsledok nie je závislý na výsledku tejto transakcie. Avšak možným riešením je aj znižovanie priority behu a tým aj hodnoty známky, čím sa uprednostňujú dôležitejšie transakcie. Znižovanie sa môže dostať na nulovú hodnotu.
- Pokiaľ má známka nulovú hodnotu je beh transakcie prerušený a transakcia je zahodená bez úspešného ukončenia. Vďaka tomuto sa uvoľnia systémové zdroje pre ostatné transakcie.
- U zápornej hodnoty sa môže systém rozhodnúť zvýšiť prioritu a snažiť sa ukončiť transakciu ako najrýchlejšie to ide alebo ďalej znižovať prioritu, prípadne ukončiť transakciu na úkor zdrojov pre ostatné transakcie.

Rozhodovanie systému, ktorou voľbou spracovávať oneskorené procesy, závisí od aplikácie, ktorá tento systém používa, a jej význame v reálnom svete. Napríklad databázový systém v nukleárnej elektrárni nedovolí zahadzovať transakcie a v prvom rade sú pre tento systém oneskorené transakcie neprijateľné. Takýto databázový systém v reálnom čase nazývame striktným (*hard real-time database system*). Na druhej strane je voľný databázový systém v reálnom čase(*soft real-time database system*), ktorý oneskorené transakcie toleruje. K momentálnemu stavu výpočtovej techniky je ťažké zaručiť, aby všetky transakcie prebehli do stanoveného času. Preto je väčšina existujúcich databázových systémov v reálnom čase voľného typu[7]. Neschopnosť databázového systému vykonať všetky transakcie v stanovený čas je odvodená od nasledujúcich faktorov[7]:

1. Vykonanie databázovej transakcie je závislé na dátach a systémových zdrojoch. Garancia vykonania všetkých transakcií do stanoveného času potrebuje dostatočné množstvo systémových zdrojov a celkové zaťaženie systému, čo môže zásadne ovplyvňovať beh iných aplikácii.
2. Celkový beh transakcie zahŕňa viaceré databázové protokoly, ktoré sú nepredvídateľné na spotrebu času pre ich vlastný beh. Niektoré takéto protokoly môžu tiež sami blokovať a reštartovať transakcie kvôli sporu o systémové zdroje.
3. Databázy, ktoré sú ukladané na *harddisk* musia neustále vykonávať I/O[1] operácie. Správa *harddisk* má vlastné časové požiadavky, možné zásobníky a riešenia výpadkov. Toto zapríčiňuje, že databáza nedokáže predvídať správanie a

[1] *Input* - vstupné signály alebo dáta prijaté počítačovým systémom / *Output* - výstupne signály alebo dáta odoslané počítačovým systémom

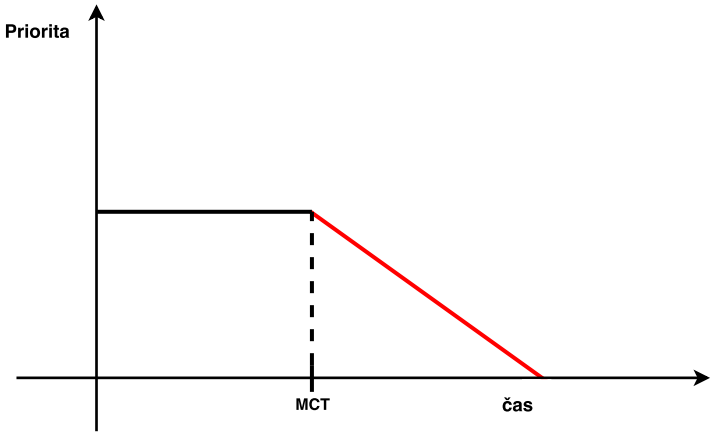
odozvu disku. Ak nie je veľkosť databázy príliš rozsiahla, dá sa celá databáza načítať do pamäte počítača a tým sa vyhnúť správe disku a zlej časovej odozvy.

### 2.1.1 Model a požiadavky na real-time transakcie

Pri požiadavke, aby náš databázový systém efektívne pracoval v reálnom čase, je treba svoju pozornosť venovať návrhu transakcií. Definovať si ich typy, vlastnosti a parametre podľa potreby a ich významu pre aplikáciu. Existujú nasledujúce informácie, ktoré je dobré si definovať, pretože pomôžu vyhýbať sa oneskoreným transakciám a dopomôžu efektívnemu plánovaniu[7]:

- **Časové intervaly** - napríklad už spomenuté maximálné časove oneskorenie.
- **Priorita transakcie** - určuje ako je dôležité, aby transakcia prebehla do stanoveného oneskorenia. Viac dôležité transakcie pre beh a spracovanie dát majú vyššiu prioritu ako ostatné transakcie.
- **Cena** - funkcia, ktorá vypočíta podľa priority transakcie na časovej osy, kedy môže byť transakcia vykonaná. Príklad - viz Obrázok 2.1
- **Systémové zdroje** - množstvo I/O operácii, predpokladané vyťaženie počítačového procesoru alebo pamäte.
- **Počet dátových požiadaviek** - predpokladaný počet transakcií.
- **Opakovanie** - pokiaľ sa vyskytnú rovnaké transakcie viackrát, môže sa definovať ich opakovanie a predvídať ich výskyt.
- **Čas výskytu** - počet transakcií sa môže líšiť v časovej ose.
- **Špeciálne vlastnosti** - transakcia môže mať iba právo na čítanie alebo konflikt s ostatnými transakciami, ktoré siahajú po rovnakých dátach. Riešenie semaforov a zámkou na dátach v prípade zápisu.

Obr. 2.1: Kladná hodnota známky s tendenciou znižovania priority




Bežné databázové systémy dbajú na to, aby dodržovali ACID vlastnosti trans-

akcií. Beh správy, ktorá tieto vlastnosti zaručuje, nie je lacný na systémové zdroje. Okrem tohto existujú špeciálne kontrolné protokoly, väzby medzi transakciami a mechanizmy obnovy databázy a jej zálohovania. Všetky tieto prvky vyžadujú svoj vyhradený čas systémových zdrojov a bránia dokonalému pracovaniu v reálnom čase pomocou blokovania, rušenia alebo zamykania transakcií. Preto sa musia transakcie rozdeliť do tried podľa toho, ako veľmi na akú transakciu vplývajú zmienené problémy. Pre každú takúto triedu je treba definovať minimálnu správu transakcií, aby sa tým ušetrilo na systémových zdrojoch a dostupnom čase pre transakcie[7].

Medzi najdôležitejší atribút transakcie v reálnom čase patrí hraničný alebo konečný čas predpokladaný pre beh tejto transakcie. Tento kúsok informácie sa použije vo viacerých fázach behu RTDBS, pri plánovaní behu jednotlivej transakcie, behu súbežných transakcií, plánovaní viacerých možných prípadov behu transakcií alebo na vypočítanie prípadných kritických miest. Obyčajne je hraničný čas behu transakcie špecifikovaný vývojármi jednotlivých aplikácii. Avšak, ak transakčný model podporuje funkcie vnorených transakcii alebo subtransakcií, naskytuje sa otázka, ako sú hraničné časy priradené takýmto vnoreným transakciám. Jednotlivé konečné termíny pre beh subtransakcií by mali vždy skončiť do hraničného termínu materskej transakcie. Vnorené transakcie môžu mať nezamknutý maximálny termín, pokiaľ sa na časovej osy behu nachádzajú na začiatku celkového času materskej transakcie. Transakcia, ktorá je posledná, musí skončiť do konečného termínu materskej transakcie. Pokiaľ vnorené transakcie zo začiatku neskončia v ich naplánovanom čase, tak môžu posunúť finálny čas. Toto môže spôsobiť vytlačenie poslednej transakcie z časovej osy celkového behu alebo predĺžiť túto časovú os a tým aj konečný termín materskej transakcie[7].

### 2.1.2 Triedy real-time transakcii

V predošlej sekcii je spomenuté, že najdôležitejším atribútom je konečný termín jednotlivých transakcií a často je táto veličina špecifikovaná výrobcom a vývojármi aplikácie, ktorá s transakciami pracuje. Na druhej strane tiež existuje iný model, ktorý upravuje a zadáva konečné časové termíny transakciám, aby sa prispôsobili konzistencií celkového behu aplikácie. Okrem transakcií, ktoré zabezpečujú stabilný beh databázy, sa môžu vyskytovať aj transakcie inicializované mimo databázy z externého zdroja. Príkladom externého zdroja môžu byť senzory monitorujúce prostredie mimo databázy ako napríklad teplotný senzor. Z tohto dôvodu je nutné, aby sa transakcie bežiace v reálnom čase a hlavne ich maximálny beh alebo takzvaný konečný termín prispôsobili externým zdrojom informácii. K tomuto slúžia opravné transakcie, ktoré musia upraviť základne atribúty vrátane času pre jednotlivé transakcie. Opravná transakcia musí správne vyrátať veličiny, aby nenarušila beh aplikácie

cie a zároveň uchovávala dojem real-time spracovania transakcii. Na základe zdroja a výstupu transakcie sa real-time transakcie delia na tieto triedy[7]:

1. **Transakcie z externého zdroja**: Tieto transakcie sa vykonávajú na zápis dát pochádzajúcich z externého zdroja do databázy. Obyčajne sa jedna o jednoduché transakcie, ktoré len zapisujú do databázy a trvajú krátku dobu. V záujme zachovania obrazu konzistentnej databázy pre externého používateľa, sú tieto transakcie vykonané ihneď bez čakania vo fronte alebo blokovania inými transakciami. Môžu avšak spôsobiť narušenie konzistencie. Pre tento prípad je potrebná správne fungujúca opravná interná transakcia, ktorá dokáže takýto stav opraviť.
2. **Interné transakcie**: Tieto transakcie sú veľmi podobné štandardným databázovým transakciám. Ako bolo spomenuté, sú tiež používané ako opravné transakcie. Vykonanie takejto transakcie trvá dlhšiu dobu kvôli jej zložitosti a náročnejším algoritmom, ktoré vykonáva.
3. **Transakcie s externým výstupom**: Tieto transakcie nemusia byť vykonané ihneď ako boli v prípade zdrojových transakcii. Často je doba ich behu krátka podobne ako pri transakciách z externého zdroja.

## 2.2 Indexy v databáze na zvýšenie rýchlosti transakcii

Medzi zásadný prvok, ktorý dokáže databázu priblížiť k Real-time databáze, je správne použitie indexov v tabuľkách databázy, ktorá bude spracovávať najviac požiadaviek vo forme transakcií. Index v databáze je dátová štruktúra, ktorá zvyšuje rýchlosť poskytovania dát z databázy. Index je vlastne kópia vybraných stĺpcov tabuľky v databáze, ktorá je uložená do zoznamu alebo stromovej štruktúry na základe určitého logického usporiadania pre lepšie vyhľadávanie. Každý index, ktorý je vytvorený je uložený na disku a obsahuje ukazovateľ na celý riadok alebo množinu riadkov v tabuľke. Tým, že sa ukladajú na disk, môžu indexy zvyšovať nároky na diskovú kapacitu ale nie v takom meradle ako samotná databáza. Veľkosť fyzický dát v tabuľke nie je ovplyvnená v prípade pridania indexov

Pri databáze bez indexov sú jednotlivé riadky v tabuľke uložené podľa toho, ako boli do tejto tabuľky vložené. V prípade, že sa pošle požiadavka na vyhľadanie konkrétneho riadku v tabuľke podľa hodnoty určitého stĺpca, tak je tabuľka prehľadávaná vždy od začiatku tabuľky riadok po riadku. Pri požiadavke kde má byť výstup viacero riadkov je hľadanie od začiatku tabuľky vykonané toľko krát, aký je počet výsledných riadok. Pri jednoduchých tabuľkách v databáze toto nemusí znamenať veľkú záťaž na zdroje, ale ako náhle obsahuje tabuľka veľký počet riadkov a

dát, je takéto vyhľadávanie náročne na časové a serverové zdroje kde sa databáza nachádza.

V tabuľke s indexami správa databázy pri požiadavke neprehľadáva všetky riadky od začiatku ale ide do zoznamu indexov, kde tiež nevyhľadáva od začiatku ale podľa podobných a najbližších hodnôt. Podobnosť môže byť napríklad v rovnakej hodnote určitého stĺpca. Akonáhle je hľadaná hodnota v indexe nájdená, ten iba ukáže na konkrétny riadok alebo riadky v tabuľke, ktoré boli hľadané 2.2. Vďaka tomuto princípu správa vykoná ďaleko menej operácii čítania v tabuľke. Z tohto dôvodu takmer každá väčšia databáza, pri ktorej sa čaká veľký počet požiadaviek na dáta má nastavené indexy. Použitie indexov spolu s optimálnym formulovaním požiadaviek patria medzi základne nástroje pre zvýšenie výkonnosti databázy. Správny vývojár, ktorý vytvára databázu, by mal tieto nástroje zahrnúť už pri samotnom návrhu databázovej štruktúry a jednotlivých tabuliek. Existuje taktiež aj riziko, kedy pri nesprávnom nastavení indexov sa výkonnosť nezlepší[8].

Existujú dva spôsoby, ako definovať konkrétny index[8]:

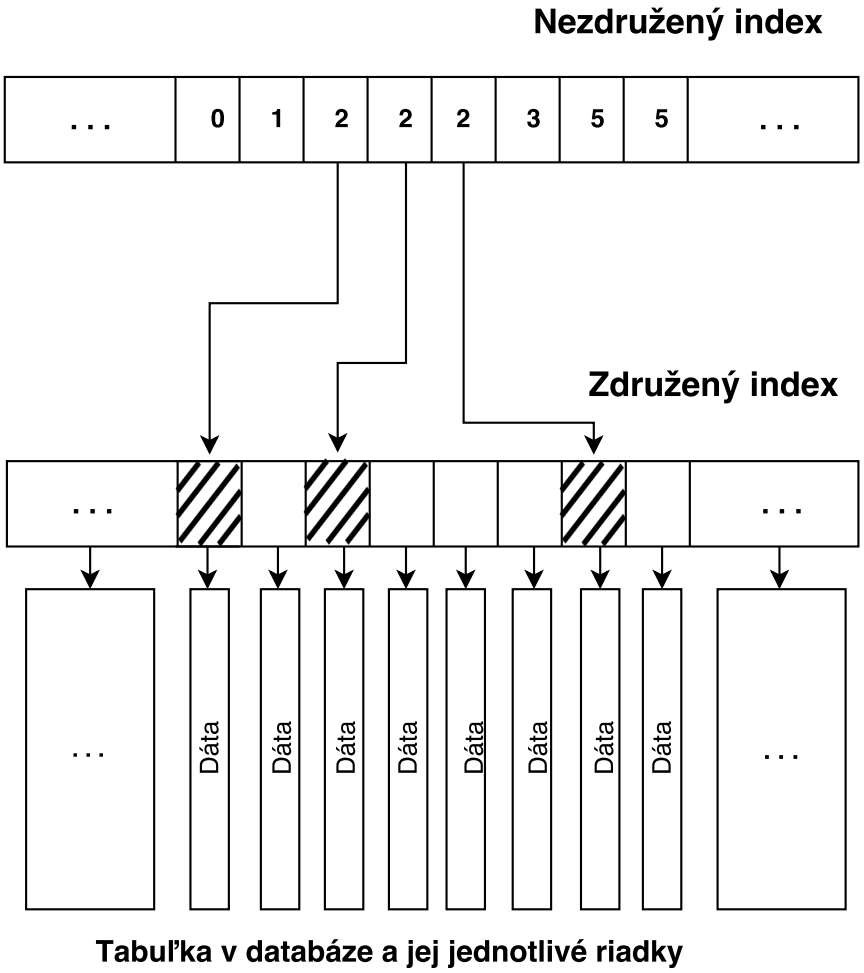
- *Index na princípe slovníka* - Slovník je zoznam slov, ktoré sú usporiadané podľa abecedy. Index definovaný týmto spôsobom je množina dát usporiadaných lexikograficky alebo podľa slovníkového poradia. Vďaka tomu vyhľadávanie neprechádza všetky riadky ale vyhľadá pozíciu dát na základe slovníkového poradia indexu.
- *Knižný index* - Tento spôsob nezmení rozmiestnenie dát v tabuľke podobne ako slovníkový index ale podobne ako zoznam alebo index v knihe upresní pozíciu dát v tabuľke. Index, ktorý je takto definovaný, sa skladá zo stĺpca tabuľky, ktorý ma index a príslušne číslo riadku v tabuľke, pre tento index.

### 2.2.1 Združené a nezdružené indexy

Podľa toho, ako sa index chová voči tabuľke a aký má význam v tabuľke sa delia indexy na združené a združené[8].

**Združený index**[8] je index, ktorý obsahuje tabuľku dát usporiadanú v určitom fyzickom poradí. Vo fáze vytvorenia združeného indexu v tabuľke sú všetky riadky tabuľky preskupené na disku podľa sekvencie indexového kľúča. Z tohto dôvodu je možné vytvoriť len jeden združený index pre jednotlivú tabuľku. Najznámejším príkladom združeného indexu je primárny kľúč, ktorý definujeme pre každý riadok tabuľky. Takýmto indexom môže byť buď numerická hodnota alebo slovná hodnota. V oboch prípadoch sú po vytvorení združeného indexu riadky s dátami numericky alebo abecedne usporiadane podľa hodnoty indexu a taktiež aj fyzicky uložené podľa tohto poradia v databáze. V prípade, ak sa pridáva nový riadok do tabuľky, tak je uložený na pozíciu v tabuľke podľa poradia svojho združeného indexu a nie na koniec

Obr. 2.2: Vzťah medzi združeným a nezdruženým indexom a príklad ukazovateľov




alebo začiatok tabuľky. Pri použití združeného indexu je treba dbať na nasledujúce hľadiská[8]:

- Združený index by mal byť vždy vytvorený skôr ako nezdružený index. Dôvodom je fakt, že nezdružený index by mal ukazovať na príslušný kľúč v združenom indexe. Viz Obrázok 2.2
- Zmena poradia združeného indexu sa vykoná pri každej modifikácii tohto indexu. Pri modifikácii netreba zabúdať na vzťahy a ukazovatele medzi združeným a nezdruženými indexami. Najlepším riešením je zahodenie všetkých indexov a ich znovu vytvorenie, aby sa vzťahy obnovili správne.
- Združený index sa neodporúča používať v tabuľkách, kde sú dáta veľmi často menené kvôli následnej častej prestavbe indexov.
- Združený index je efektívny práve v prípadoch keď vyhľadávame usporiadané

dáta alebo väčší rozsah dát.

**Nezdružený index**[8] je uložený vo forme vyváženej stromovej štruktúry, ktorá začína od koreňa. Hlavným rozdielom medzi združeným a nezdruženým indexom je, že kľúčové hodnoty nezdruženého indexu nie sú usporiadané a samotný nezdružený index neobsahuje dáta, ale iba smeruje na jednotlivé kľúče združeného indexu, za ktorým sú až reálne dáta s riadkami v tabuľke ako môžeme vidieť na obrázku 2.2.

Na základe tohto tento index nespôsobuje žiadne fyzické zmeny v tabuľke a ani nemení usporiadanie riadkov. V prípade zmeny poradia spôsobeného zmenou združeného indexu si nezdružený index uchováva stále ten istý ukazovateľ pozície do združeného indexu. SQL server v prípade takejto zmeny pridáva do konkrétneho listu ďalší ukazovateľ smerujúci na nové dáta. Na rozdiel od združeného indexu môžu v tabuľke existovať viaceré nezdružené indexy. Pri použití nezdruženého indexu je treba zobrať ohľad na tieto faktory[8]:

- nezdružený index je užitočný v prípade vyhľadávania malého množstva dát z veľkej tabuľky, pri použití zložitejších kľúčov a pri častých modifikáciách tabuliek,
- neodporúča sa používať tento index pri načítaní veľkého množstva dát, pretože by sa stal neefektívny kvôli zložitejšej štruktúre ukazovateľov,
- používa sa podobne ako združený index na usporiadaných dátach,
- efektívne zvláda špecifickejšie požiadavky pri vyhľadávaní.

### 2.2.2 Typy indexov a ich použitie v MySQL databáze

Diplomová práca pracuje s už dopredu poskytnutou MySQL databázou. Z tohto dôvodu je dobré si priblížiť, ako sa indexy používajú v tejto databáze a aké rôzne typy tam existujú. Väčšina týchto typov sa dá rozdeliť na združené a nezdružené typy, ktoré sú opísane bližšie v predošlej sekcii. Funkcia indexu v MySQL je rovnaká ako vo všetkých databázach. Index pracuje podobne ako zoznam v knihe, ale namiesto čísla stránky ukazuje na pamäťové miesto s dátami. Správou indexu sa zaoberá databázový server, ktorý pri jeho vytvorení zarezervuje pre požadovaný index určitú časť pamäťového priestoru a uloží do tohto indexu rozmiestnenie hodnôt indexových stĺpcov v tabuľke[9].

**Vytvorenie indexu** môže byť buď vo fáze definovania jednotlivých tabuliek alebo neskôr, keď už tabuľky existujú a objaví sa potreba indexov. Pred použitím indexov by mal návrhár databázy prejsť všetky stĺpce a identifikovať, ako často sa hľadané záznamy z určitej tabuľky triedia podľa ktorého stĺpca. Tie stĺpce, ktoré sú najviac používane na vyhľadávanie, patria medzi kandidátov pre zavedenie indexov. V MySQL sa na vytvorenie indexu používa príkaz *CREATE* a následne definícia typu indexu a stĺpca, na ktorom sa tento index zavedie[10]. Jednoduchý príklad

SQL príkazu je možno vidieť na obrázku 2.3. Tento jednoduchý kód vytvorí tabuľku, kde sa budú ukladať knihy, ktoré sa predali na určitej predajni v určitom čase. V tomto príklade je vidieť vytvorenie vo fáze definovania novej tabuľky a taktiež neskôr pridávanie ďalších indexov. Prvý zadefinovaný index je primárny kľúč ako identifikátor a následne index pre dátum predaja. Neskôr, keď sa tabuľka používa, vývojári zistia, že okrem vyhľadávania podľa dátumu predaja chcú výber pre jednotlivých autorov a predajne. To je zadefinované nasledujúcimi dvoma príkazmi *CREATE* na obrázku 2.3.

Obr. 2.3: Príklad kódu v jazyku SQL, ktorý ukazuje definovanie indexov

```
CREATE TABLE PredaneKnihy (
 id INT(6) NOT NULL AUTO_INCREMENT,
 nazovKnihy VARCHAR(30) NULL,
 autor VARCHAR(30) NULL,
 predajna VARCHAR(30) NULL,
 datum_predaja TIMESTAMP default CURRENT_TIMESTAMP,
 PRIMARY KEY (id),
 INDEX (datum_predaja)
);

CREATE INDEX autor ON PredaneKnihy (autor);
CREATE INDEX predajna ON PredaneKnihy (predajna);
```

V MySQL existujú tieto základne **typy indexov**[10]:

- **Primary** - je tvorený stĺpcom, ktorý obsahuje primárny kľúč. Tento typ indexu sa môže vyskytovať v tabuľke iba raz. Vybraný by mal byť stĺpec, ktorý jednoznačne identifikuje každý záznam v tabuľke. Existujú určité pravidlá, ktoré sa dodržujú pri správnom definovaní tabuľky. Medzi tieto pravidlá patrí pravidlo, že primárnym kľúčom je vždy unikátny identifikátor číselnej hodnoty, ktorý sa môže volať *„id“*, ako je vidieť v príklade kódu na obrázku 2.3. Primárny kľúč je zvláštny prípad typu *Unique*.
- **Unique** - je index vytvorený zo stĺpcov, ktoré obsahujú unikátnu hodnotu pre každý riadok. Tabuľka môže obsahovať viacero unikátnych kľúčov, narozdiel od primárneho kľúča. Jednotlivý index môže byť taktiež zložený z viacerých stĺpcov tabuľky, ako napríklad krstné meno a priezvisko v tabuľke, kde vieme, že žiadna osoba v tabuľke nebude mať rovnaké celé meno. Keďže sa jedná o unikátny kľúč, má MySQL ošetrenú výnimku a nedovolí záznam pridať v prípade, že do tabuľky pridávam nový záznam, ktorý bude obsahovať rovnaký unikátny index.
- **Index** - nazývaný taktiež sekundárnym alebo druhotným kľúčom. Druhotný index je pravý nezdružený typ indexu, ktorý optimalizuje vyhľadávanie podľa

ďalších stĺpcov rozdielnych od primárneho alebo unikátneho indexu. Tento sekundárny index sa ukladá do logickej stromovej štruktúry, kde každý list stromu ukazuje ďalej na primárne alebo unikátne indexy. Vďaka správnemu definovaniu indexov dokážeme markantne zrýchliť vyhľadávanie záznamov v tabuľkách. Miesto zložitého prehľadávania v abecednom alebo numerickom poradí dokáže sekundárny index vyhľadať pozíciu podľa veľkosti a ihneď nasmerovať cez primárny kľúč na záznamy v tabuľke. Sekundárny index môže byť definovaný aj na fiktívnom stĺpci, ktorý je výsledkom nejakej operácie na reálnych stĺpcoch tabuľky.

- **Fulltext** - už samotný názov ukazuje, že tento index je používaný na *FULLTEXT* vyhľadávanie[2] stĺpcov v tabuľke. Ako presne je toto vyhľadávanie uskutočnené, záleží na databázovom servery samotnom. Niektoré môžu využívať slovníkové ukladanie najčastejšie vyhľadávaných slov alebo pomocné *hash* funkcie[3].

## 2.3 Agregácia databázových dát

Aby databáza splňovala podmienky a vlastnosti Real-time databázy, musí mať dobrú správu transakcií, korektne používať indexy, ako je opísané v predchádzajúcej sekcii, a nakoniec pracovať s dátami, ktoré sú rýchlo dostupné a často vyhľadávané. Pre tento posledný bod existuje viacero metód, ako si preferované dáta získať alebo upraviť. Medzi najznámejšie metódy patrí agregácia dát.

Agregácia dát je každý proces kedy, sú informácie zhromažďované a prehľadne prezentované za účelom štatistickej analýzy. Agregácia spracuje dáta a na základe určitých vlastností a hodnôt a tieto spracované dáta ukladá. Uložením takýchto dát sa zaberie miesto na disku, ale o mnoho menej miesta ako dáta, ktoré agregácia prepočítala. Základom agregačného procesu alebo návrhu pre tento proces je stanovenie agregačných veličín, podľa ktorých sa dáta spracujú. Každá takáto veličina sa určuje podľa toho, čo užívateľ hľadá medzi informáciami. Príkladom takejto veličiny môže byť časový interval. Ako príklad môže existovať tabuľka dát obsahujúcich hodinové záznamy nameranej teploty pre určitú polohu. Ak je treba vidieť priemernú teplotu za celý deň, tak zoberú všetky namerané hodnoty toho dňa a vypočíta sa priemerná teplota. Presne takéto príklady výpočtov zahŕňa problematika agregácie[11].

Vďaka agregácii sa dáta v databáze dokážu spracovať, prepočítať a uložiť. Niektoré systémy používajú agregáciu na redukciu množstva uložených informácii. Podľa

[2]Možnosť vyhľadávať na základe celého textu uloženého v databáze, prehľadáva sa každé slovo a nie len kľúče a identifikátory.

[3]Hash funkcie je matematický algoritmus na prevod vstupných dát do malého čísla. Táto funkcia sa používa k rýchlejšiemu prehľadávaniu tabuliek a porovnávanie dát databázy.

typu ukladaných dát si určia hodnoty, na ktorých prebehne agregácia a namiesto ukladania všetkých historických dát ukladajú len prepočítané dáta z agregácie. Takto ušetria na úložnom priestore disku a nestrácajú dáta z minulosti. Dobrým príkladom sú aplikácie, ktoré monitorujú výkonnosť a vyťaženie sieťových zariadení. Pomocou sieťovej komunikácie sa každých päť minút zbierajú informácie o sieťovom prenose, vyťažení procesoru a pamäte. Následne každú hodinu prebehne agregácia, namerané dáta spriemeruje, nájde maximálne a minimálne hodnoty a uloží ich len ako hodinovú vzorku. Takže na miesto toho, aby takáto aplikácia ukladala do databázy všetky päťminútové merania za celý deň, tak jednoducho uloží iba hodinové vzorky. Ak sú dáta príliš staré, môžu prejsť agregáciou na iba denné vzorky a pod[11].

### 2.3.1 Metódy agregácie a dátové typy

Hlavným použitím agregácie sú výpočty štatistických dát. Preto je logické, že medzi základne metódy agregácie patria matematické operácie používané pri štatistických výpočtoch. Už samotné databázové systémy dovoľujú takéto výpočty a dokážu získať agregované dáta pomocou SQL príkazov. Dostupné agregačné metódy v databázach sú[12]:

1. *súčet* (spočíta hodnoty pre čísla alebo intervaly),
2. *priemer* (môže byť v databáze použitý pre čísla, dátumy, časy a intervaly),
3. *medián* (má rovnaké použitie ako priemer),
4. *minimálna hodnota* (podobne ako priemer sa používa pri viacerých numerických hodnotách, ktoré sa dajú porovnať),
5. *maximálna hodnota*,
6. *rozptyl* (vypočíta strednú hodnotu kvadrátov odchýlok od strednej hodnoty),
7. *smerodajná odchýlka* (je odmocnina z rozptylu a spolu s rozptylom sa v databáze dá použiť pre číselné hodnoty, dátumy, časy a intervaly),
8. *množina* (namiesto matematického výpočtu, vytvára zoznamy všetkých unikátnych hodnôt na akomkoľvek stĺpci tabuľky).

# 3 SERVEROVÁ A PREZENTAČNÁ APLIKÁCIA

Hlavným cieľom diplomovej práce je vytvorenie aplikácie, ktorá bude poskytovať požadované dáta v požadovanej forme. Pre tento cieľ je potrebné sa sústrediť na miesto, kde sa primárne dáta nachádzajú a požiadavky spracovať už pri tomto zdroji. Keďže databáza sa nachádza na servery, ktorý merané dáta prijíma a ukladá ich, bude logické na tomto servery taktiež vybudovať serverovú aplikáciu, ktorá tieto dáta poskytuje. Výhodou tohoto riešenia je ušetrenie sieťovej komunikácie po internete. Namiesto toho, aby sa požadované dáta celé presúvali k užívateľovi a spracovávali na jeho lokálnom počítačovom zariadení, sú tieto dáta spracované na servery s databázou. Takto sa k užívateľovi pošlú len tie dáta, ktoré potrebuje a ušetrí sa aj výpočtový výkon na užívateľovej strane. Servery sú taktiež lepšie prispôsobené na zložitejšie výpočty než bežné užívateľské zariadenia.

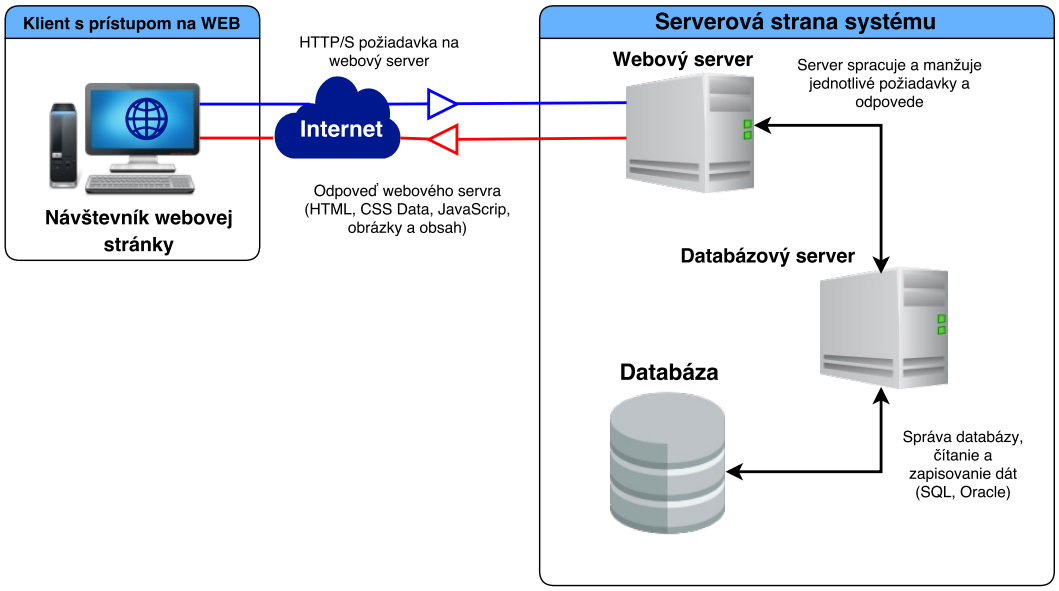
Aby erverová aplikácia spĺňala požiadavky na rýchlu odozvu užívateľa, využíva Real-time databázy spomenuté v predošlej kapitole. Preto je treba pri jej tvorbe prejsť možnosti databázových transakcií, správneho použitia indexov v databáze a technológii, ktoré poskytujú predpočítané dáta, ako je zmienená agregácia. Vďaka použitiu prebraných metód sa docieli rýchlosť a efektivita už na strane servera, kde sa databáza nachádza a dovolí efektívnejší prístup k týmto dátam aj pre užívateľov.

## 3.1 Vzťah medzi klientom a serverom

Jedným z najznámejších modelov, ktoré znázorňujú vzťah medzi užívateľom, serverom a aplikáciami je model Client-Server. Vyjadruje už zmienený vzťah medzi užívateľom alebo klientom a serverom. Klient pošle požiadavku na server. Ten požiadavku príjme a spracuje, vykoná potrebné výpočty pomocou aplikácií a pošle odpoveď klientovi ako vidieť na Obrázku 3.1. Server môže obsluhovať viacero klientov naraz, čo dovoľuje používanie aplikácie viacerým užívateľom bez toho aby, museli čakať na svoje poradie alebo na zdroje. Klient nevidí, čo sa deje na pozadí s jeho požiadavkou a väčšina správy a spracovania dát je na strane servera. Toto odľahčuje klienta od zbytočného používania lokálnych výpočtových zdrojov a nutnosti inštalácie externých programov, ktoré sú potrebné na spracovanie dát. Jedným z najlepších príkladov klient-server modelu je webové rozhranie. Užívateľovi stačí mať webový prehliadač a pomocou tohto rozhrania sa dostane k dátam na rôznych serveroch v Internete a môže jednoducho s týmito dátami operovať a modifikovať bez toho, aby vedel kde a ako sú uložené[13].

Všetko toto je možné vďaka webovým aplikáciám, ktoré pracujú na pozadí nedostupnom normálnemu užívateľovi a poskytujú mu iba tie dáta, ktoré potrebuje. Webové aplikácie začali iba ako forma prezentovania dát pomocou textu a grafickej úpravy na internete. Časom sa stali čoraz viac populárnejšie a to spôsobilo ich rozvoj a rozšírenie aj pre zložitejšie aplikácie ako prezentovanie. Dnes už je možné cez webové aplikácie obsluhovať mail, upravovať dokumenty, spravovať financie alebo dovoliť užívateľom ovládať iné aplikácie. Vďaka tomuto sú webové aplikácie jedny z najviac využívaných služieb dnešnej doby, a preto sa ich vývoju venuje veľa pozornosti. Vzniklo viacero technológii na vývoj takýchto aplikácii. Vytvorila sa množina programovacích alebo značkovacích jazykov, ktoré sú predovšetkým používané na webové aplikácie. Avšak v pozadí každej webovej aplikácie existuje funkcionalita alebo mechanizmus serverovej aplikácie, ktorá využíva špecifické technológie prispôsobené podľa potreby a funkcie[13].

Obr. 3.1: Komunikácia a vzťah medzi klientom a serverom




## 3.2 Technológie webovej aplikácie

Webová aplikácia je každá aplikácia, ktorú užívateľ používa a zobrazuje si ju pomocou webového prehliadača alebo komunikuje pomocou sieťových webových protokolov. Podľa toho ako aplikácia funguje alebo čo používa delíme technológie, ktoré bežia pomocou[13]:

- webového prehliadača,
- klientskej aplikácie,
- mobilnej aplikácie.

Cez **webový prehliadač** sú posielané inštrukcie formou dotazov v znakovom jazyku ako je napríklad HTML[1]. Formát a vzhľad prezentácie je upravovaný podľa špecifikácie jednotlivých stránok. Dnešné stránky využívajú viacerých funkčností na grafickú manipuláciu a interaktívne zobrazovanie, ktoré vyžaduje modernejšie programovacie jazyky ako je samotné HTML. Príkladom takéhoto jazyka je JavaScript alebo PHP. Užívateľ webového prehliadačka zadáva príkazy cez jeho rozhranie. Tieto príkazy sú odoslané na server, kde webová aplikácia beží, následne spracované a dáta odoslané naspäť do webového prehliadača. Dáta pre jednotlivé webové stránky môžu byť uložené buď iba na servery, lokálne u užívateľa alebo na oboch miestach.

Webová aplikácia môže bežať aj bez prítomnosti webového prehliadača. Tento spôsob funguje pomocou **klientskej aplikácie**, ktorá je stiahnutá a nainštalovaná na stanici užívateľa alebo sa stiahne pri každom požiadavku na spojenie. Táto aplikácia komunikuje pomocou webových protokolov. Táto technológia je príbuzná modelu klient-server, pričom server sa nachádza na internete a operuje pomocou webových protokolov. Dáta sú ukladané rovnako ako v prípade webového prehliadačka.

**Mobilné aplikácie** sú v skutočnosti klientskou aplikáciou iba s tým rozdielom, že boli vybudované pre mobilné aplikácie telefónov, smartphone a tabletov. Na mobilný telefón sa nainštalujú z webových stránok a bežia ako samostatné aplikácie bez potreby prehliadača. Jednoduché dáta, ako textové informácie, môžu byť uložené lokálne a zvyšné obrazové a aktualizované informácie sa sťahujú pomocou webových protokolov ako je HTTP[2].

## 3.3 Výber programovacieho jazyka serverovej aplikácie

Predošlá sekcia sa zamerala na webovú aplikáciu. Väčšina webových aplikácií je stavaných do modelu klient-server. Preto je vždy potreba vytvoriť funkčné jadro aplikácie na samotnom servery. Pre tento účel sa používajú nielen špecializované jazyky webových aplikácií ako sú PHP alebo JavaScript ale jadro naprogramované aj inými populárnymi programovacími jazykmi. Programovacie jazyky sa líšia technológiami, ktoré používajú, a funkcionalitou, ktorú podporujú. Niektoré môžu byť

[1]HTML(Hypertext Markup Language) - značkovací jazyk, ktorý popisuje webový dokument. Jeho pomocou sa vytvárajú webové stránky a zobrazuje sa ich obsah.

[2]HTTP(Hypertext Transfer Protocol) - je aplikačný protokol na distribúciu a ovládanie dát vo webovom prehliadači. Obsahuje štruktúru pravidiel na vytváranie spojenia prípadne autentifikácie

lepšie stavané na komunikáciu s webovými aplikáciami, iné preferované zas na zložité matematické výpočty.

Súčasťou správneho návrhu webovej aplikácie je dôležitý výber správneho programovacieho jazyka a porovnanie jednotlivých výhod. Medzi najznámejšie a momentálne najviac používane programovacie jazyky patria *Python*, *Java*, *C++*, *C#*, *JavaScript*, *PHP* a *Ruby*. Každý z týchto jazykov má svoje výhody a nevýhody, ktoré sú bližšie detailnejšie opísane so zameraní na použitie pre webovú aplikáciu.

- **Python** je dnes jeden z najpoužívanejších programovacích jazykov. Python je objektovo-orientovaný, interpretovaný programovací jazyk s dynamickou sémantikou. Vyznačuje sa dobre čitateľným kódom, ktorý má striktné pravidlá, a tým môže vyjadriť rovnaké funkcie ako iné programovacie jazyky na menšom počte riadkov kódu. Taktiež podporuje moduly a balíčky čím zabraňuje zbytočnému opakovaniu kódu a vytvára lepšiu štruktúru[14]. Vývojármi je tento jazyk používaný na rýchly vývoj aplikácií s dobrou čitateľnosťou kódu, ako skriptovací jazyk alebo na spájanie modulov a funkcionality medzi rozdielnymi aplikáciami. Ďalšími výhodami, okrem dobre čitateľného kódu, sú jeho dostupnosť, stabilita vydaných verzií, podpora modulov, integrácia s ostatnými jazykmi ako *C* a *Java*. Nevýhodou je absencia plnej multiprocesorovej podpory, komerčná podpora a výkonnosť programov napísaných v Python oproti iným jazykom[15]. Python sa dá použiť aj na tvorbu webových aplikácií, ale potrebuje webovú nadstavbu alebo takzvaný *Framework*. Populárne webové nadstavby sú *Django*, *TurboGears* alebo *web2py*.
- **Java** je tiež objektovo-orientovaný programovací jazyk, ktorý sa zakladá na štruktúre tried. Má veľmi široké využitie a často sa používa ako vyučujúci jazyk na vysvetlenie objektovo orientovaného jazyka. Kód je rozdelený do hierarchických objektov, ktoré sa navzájom využívajú. Tento jazyk bol navrhnutý, aby bol čo najmenej závislý na balíčkoch a knižniciach. Skompilovaný Java kód je spustiteľný na všetkých výpočtových zariadeniach, ktoré majú *JVM*[3]. Vďaka veľkej popularite sa Java stala používaná hlavne pri vývoji webových aplikácií. Okrem veľkej portability a možnosti behu na takmer všetkých systémov má Java jednu z najlepších dokumentácií pre developerov. Mínusom tohto jazyka je slabšia správa čistenia voľného miesta v pamäti, časté výskyty bezpečnostných dier v JVM a neustála inštalácia aktualizácii. Aj napriek tomu sú aplikácie stavané na Java jazyku veľmi rozšírené aj v komerčnej sfére[16].
- **C++** spadá tiež do skupiny objektovo-orientovaných jazykov. Tento jazyk má syntax z jazyka **C** a oddelil sa od tohto jazyka, pretože zaviedol používanie tried ako objektov. Významnou schopnosťou C++ je, že ponúka vlast-

[3]JVM(Java Virtual Machine) - abstraktný virtuálny stroj. ktorý umožňuje beh Java programov.

nosti vyššieho zložitejšieho jazyka a jednoduchšieho jazyka. Vyšší jazyk je zdedený z jazyka C a umožňuje detailnú prácu so systémovými zdrojmi a procesmi. Jednoduchší jazyk je viditeľný v použití mnohých knižníc, ktoré existujú. Oproti jazyku Java má lepšiu správu pamäte, ktorá môže byť explicitne naprogramovaná. Vďaka týmto vlastnostiam je tento jazyk viac preferovaný na programy pre zložitejšie výpočty a efektívnu prácu so systémovými zdrojmi. C++ je dobre prepojené s jadrom operačných systémov, avšak na rýchle programovanie pre neustály vývoj webových aplikácii je tento jazyk ešte málo prispôsobený[17].

- **C#** je objektovo-orientovaný jazyk, ktorý dbá na silnú typovosť, deklarácie premenných a funkčnosť. Zlieva dokopy vlastnosti programovacích jazykov Java a C++. Vývoj tohto jazyka prevzala spoločnosť *Microsoft*, čo viedlo k vytvoreniu silnej nadstavby a rozhraniu *.NET* s veľkou podporou nových funkcií. Tento jazyk sa stal populárny vďaka jeho zrozumiteľnosti a dobrému rozhraniu s aktualizovanými knižnicami. Nevýhodou programovacieho jazyka C# je slabšia efektivita pri práci s výpočtovými zdrojmi a prenositeľnosť na operačné systémy, ktoré nie sú pod správou spoločnosti *Microsoft*. C# ponúka možnosť tvorby webovej aplikácie a webovej stránky, avšak pre serverovú aplikáciu bežiacu na iných operačných systémoch je potreba emulácie nadstavby a dostupných knižníc, ktoré nemajú takú podporu a aktualizácie[18].
- **JavaScript** je dynamický, netypový a interpretovaný programovací jazyk. Skôr ako iné programovacie jazyky je tento využívaný hlavne ako skriptovací jazyk do webového rozhrania. Spolu s *HTML* jazykom a *CSS*[4] je dôležitou technológiou pre webové aplikácie. JavaScript je multi-paradigmový jazyk, ktorý podporuje objektovo-orientovaný, imperatívny a funkcionálny typ programovania. Hlavnou úlohou a zameraním tohto jazyka sú aplikácie alebo skripty, ktoré sú vložené v HTML kóde a svoj beh vykonajú v klientskom webovom rozhraní a klientskych výpočtových zdrojov. Vďaka tomuto je tento jazyk viac orientovaný na stranu a zdroje klienta ako na stranu servera. Nie je veľmi používaný pre programovanie serverových aplikácii kvôli zameraniu na webové prehliadače. Nevýhodou tohto jazyka je, že je často používaný na vykonanie aktivít na strane klienta, ktoré narúšajú jeho bezpečnosť a zisťujú systémové informácie[19].
- **PHP** je serverovo zameraný voľne dostupný skriptovací jazyk, ktorý je vsadený do *HTML* kódu so syntaxou podobnou programovaciemu jazyku *C*. PHP jazyk je široko používaný v kombinácii s viacerými webovými vývojárskymi rozhraniami a ich funkcionalitou. Tento jazyk je taktiež integrovaný s popu-

[4]CSS(*Cascading Style Sheets*) definuje výzor a štýlovú stránku webových stránok. Používa sa na opis vzhľadu dokumentu napísaného v značkovacom jazyku.

lárnymi databázami, ako je *MySQL*, *Oracle*, *PostgreSQL*, *Sybase*. Programy a aplikácie vytvorené pomocou tohto jazyka sú rýchle pre svoj beh, hlavne ak sú skompilované ako *Apache*[5] modul na Unixovom operačnom systéme. PHP dokáže spracovať a spustiť i zložité SQL príkazy v MySQL databáze a načítať výsledky v rekordnom čase. Taktiež ako jazyk pre kostry a rozhrania webových stránok podporuje veľkú časť sieťových a aplikačných protokolov Internetu. Veľkou výhodou takéhoto skriptovacieho jazyka je jeho jednoduchosť pre začiatočníkov. Zároveň ponúka široké spektrum funkcii i pre skúseného programátora. Okrem všetkých týchto výhod je PHP jazyk často aktualizovaný novými funkciami, čo spôsobuje spätnú nekompatibilitu na serveroch, ktoré nemajú nainštalovanú poslednú verziu. Pre ciele diplomovej práce, kde bude aplikácia komunikovať s MySQL databázou a tiež prezentovať cez webové rozhranie, je PHP jazyk dobrou voľbou[20].

- **Ruby** je dynamický, objektovo-orientovaný programovací jazyk s komplexnou syntaxou, stálym jadrom a dobre podporovaným rozhraním. Ruby čerpá inšpiráciu z programovacích jazykov ako sú *Lisp*, *Smalltalk* a *Perl*, ale zároveň používa syntax z jazykov *C* a *Java*. Podobne ako *Python* je Ruby čisto objektovo-orientovaný jazyk, ktorý podporuje procedurálny a funkcionálny štýl programovania[21]. Toto vytvára z Ruby mutli-paradigmový jazyk, ktorý ma veľmi široké spektrum využitia. Dobrým príkladom môže byť schopnosť spolupráce so značkovacím jazykom *HTML*[21]. Oproti jazyku PHP poskytuje lepšiu bezpečnosť naprogramovaných aplikácií a nie je tak zásadne často a rozdielne aktualizovaný, čo prináša stabilitu kódu z dlhodobejšieho hľadiska. Nevýhodou tohto jazyka môže byť zložitejšia syntax pri komplexnom kóde programu, čo môže spôsobiť dlhšiu dobu vývoja aplikácii a programov v Ruby oproti spomínanému jazyku *PHP*.

[5]Apache je HTTP server, na ktorom bežia webové aplikácie a webové stránky. Je to jeden z najviac používaných webových softvérov na svete. Najčastejšie používaný na Unixových platformách.

# 4 REALIZÁCIA APLIKÁCIE

V tejto kapitole je detailne opísane spracovanie praktickej časti diplomovej práce, ktorej výsledkom sú tri časti. Agregačný skript, ktorý funguje na servery s databázou a pripravuje dáta, back-end podpora vo forme *API* a webová stránka, ktorá prezentuje dáta z databázy. Každá táto funkcia by sa dala naprogramovať vo viacerých programovacích jazykoch, ktoré sú spomenuté na konci predchádzajúcej kapitoly. Na serverovú aplikáciu by najviac vyhovovali jazyky *C*, *Java* alebo *Python*. Avšak pre webovú stránku aplikácie a API je lepšie použiť jazyky s dobrým vzťahom a podporou značkovacích jazykov, ako je už spomínaný jazyk *PHP*. Aby jednotlivé časti neboli naprogramované každá v inom programovacom jazyku zvolil som si ako hlavný jazyk pre serverovú aplikáciu, API a webovú stránku jazyk PHP. Voľbu tohto jazyka ovplyvnili nasledujúce výhody a faktory:

1. **rýchla práca s MySQL databázou** zaručí zlepšenie celkovej odozvy a rýchlosti aplikácie pri práci s databázou či už nameraných alebo agregovaných dát,
2. **ľahké nasadenie aplikácie na web** vďaka vbudovanej kompatibilite s *HTML* jazykom,
3. **existujú viaceré rozhrania**, ktoré už ponúkajú stabilnú základňu pre programovanie aplikácie a funkcie s pravidelnými aktualizáciami,
4. **možnosť serverového skriptu**, ktorý by spĺňal úlohu agregácie,
5. **veľké množstvo zdrojov** a rád vďaka širokej komunite ľudí, ktorí PHP používajú,
6. **predošlé skúsenosti** s programovaním v tomto jazyku,
7. **ľahká inštalácia a prenositeľnosť** na iné servery.

## 4.1 Testovací server a použité vývojové rozhrania

Ako podklad pre vytvorenie aplikácie bola poskytnutá vzorka databázy, ktorá obsahovala tabuľku s nameranými dátami. Databáza, s ktorej pochádzala vzorka a s ktorou bude naprogramovaná aplikácia pracovať, sa nachádza na univerzitnom servery s operačným systémom *Debian Jessie(stable)*. Z dôvodu lepšej kompatibility naprogramovanej aplikácie a jej nasadeniu na univerzitný server sa použil testovací server s Linuxovým operačným systémom a podobným zložením komponentov webového servera a databázy. Testovací server fungoval ako virtuálny server spustený cez virtualizačný program *VMware Workstation 12 Player*, ktorý je voľne dostupný pre nekomerčné účely na stránkach `https://www.vmware.com/products/player`. Následne je použitá sada voľného dostupného softvéru používaného ako platformu

pre vývoj dynamických webových stránok **LAMP**[1]. Aplikácia je naprogramovaná pomocou nasledujúcej sústavy LAMP použitej na testovacom servery:

- **Centos 6.7 i386** - Linuxový operačný systém nainštalovaný s virtuálnymi parametrami 1 procesoru s dvoma jadrami, 1 GB pamäťou a 20 GB diskového priestoru. Inštalačný súbor je použitý zo zdroja `http://isoredirect.centos.org/centos/6.7/isos/i386/`.
- **Apache 2.2.15** - Konkrétna verzia webového servera je zo základných inštalačných balíčkov operačného systému.
- **MySQL 5.1.73** – Vybraná verzia je z inštalačných balíčkov OS.
- **PHP 5.3.3** – Podobne ako u MySQL a Apache verzia pochádza z inštalačných balíčkov.

Okrem tejto základnej sady LAMP, sa používa vývojové rozhranie pre webové stránky **Nette**. Prečo práve toto rozhranie? Po preštudovaní internetových fór zaoberajúcich sa vývojom nových projektov v programovacom jazyku PHP, bolo často toto rozhranie odporúčané. Vo väčšine prípadov sa používa na rýchly vývoj aplikácie nejaké vývojárske a programové rozhranie. Nette patrí medzi špičku najlepších rozhraní pre PHP, pochádza od českých vývojárov a je voľne dostupné. Podľa prieskumu popularity viacerých najznámejších rozhraní zo zdroja `https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/` skončilo Nette na úctyhodnom treťom mieste. Samotný tvorcovia uvádzajú nasledujúce výhody Nette rozhrania[22]:

1. excelentný šablónovací systém,
2. perfektné ladiace nástroje,
3. neobyčajne efektívna databázová vrstva,
4. dômyselné zabezpečenie pred zraniteľnosťami,
5. moderné rozhranie s podporou *HTML5*, *AJAX* alebo *SEO*,
6. kvalitná dokumentácia s aktívnou českou komunitou,
7. čistý objektový návrh,
8. zdarma dostupný.

Z hľadiska praktickej časti diplomovej práce Nette pevne ošetruje komunikáciu s databázou pomocou databázovej nadstavby nad PDO[2]. Používa špeciálne triedy *Nette\Database\Connection*, *Nette\Database\Context* a *Nette\Database\Table*, ktoré uľahčujú prístup k dátam v databáze a ošetrujú prípadné chybové stavy, ktoré môžu nastať na strane databázy[23]. V celom kóde diplomovej práce využívam funkcionalitu všetkých týchto tried na čítanie a zapisovanie v databáze. Hlavne trieda

[1]LAMP(Linux, Apache, MySQL, PHP) - Je sústava vyladená a prispôsobená pre webové aplikácie. Pozostáva z operačného systému, webového servera, databázy a programovacieho jazyka

[2]PDO(PHP Data Objects) – rozšírenie, ktoré definuje rozhranie na pristupovanie do databázy v PHP.

*Nette\Database\Connection* overuje naviazanie spojenia s databázou a prihlasovacie údaje nezverejňuje v samotnom kóde programu, ale čerpá z konfiguračného súboru neprístupného vonkajšiemu svetu[23]. Nette rozhranie sa ukázala ako veľmi dobrá voľba nielen kvôli komunikácii s databázou, ale hlavne na vývoj prezentačnej časti diplomovej práce vo forme webovej stránky. Vďaka viacerým zabudovaným funkcionalitám dovoľuje dobrý import webových štýlov, ako je napríklad *Bootstrap*[3], použitie javascript a kompatibilitu s mobilnými zariadeniami.

Aby sa mohlo začať pracovať s Nette, potrebuje sa najskôr stiahnuť nástroj **Composer**, ktorý dovoľuje deklarovať závislosti jednotlivých knižníc pre rozhranie a následne ich nainštaluje do adresára s projektom. Návod ako nainštalovať Nette pomocou Composer je dostupný na webovej stránke `https://doc.nette.org/cs/2.4/quickstart/getting-started`. Pri inštalácii Nette sa vytvorí takzvaný *Sandbox* alebo kostra webového projektu, ktorá vytvára objektovú štruktúru celého projektu. Príklad takejto kostry môžeme vidieť na obrázku 4.1, ktorý ukazuje výslednú štruktúru naprogramovanej aplikácie v Nette rozhraní. Nette patrí medzi klasické MVP[4] rozhrania[24]. Podľa MVP aj Nette rozdeľuje svoju funkcionalitu do jednotlivých adresárov, ktoré možno vidieť na obrázku 4.1. Medzi tri základné patria[24]:

- riadenie, ktoré sa nachádza v adresári **presenters**,
- logika vo forme tried v adresári **model**,
- výstup vo forme šablón v adresári **templates** a jeho podadresároch.

Pri tvorbe webového projektu v Nette môže finálny web používať viacero stránok alebo odkazov medzi rozdielnymi časťami celého projektu. V prípade diplomovej práce sa používajú len tieto dva odkazy. Jeden na hlavnú prezentačnú stránku a druhý na API rozhranie. Celkové smerovanie a tvorbu správnej adresárovej štruktúry pri zložitejších webových stránkach riadi **Router** alebo smerovač. Úlohou tejto komponenty je rozpoznať podľa *URL*[5] adresy, čo užívateľ chce a podľa toho zavolať príslušné riadenie, ktoré požiadavku užívateľa obslúži. MVP model a smerovač tvoria jadro rozhrania Nette[24]. Z obrázku 4.1 je vidieť, že celá štruktúra má ešte viacero komponentov, ktoré sú príslušne opísané priamo na obrázku. Za spomenutie stojí koreňový adresár, ktorý je jediný prístupný z internetu, adresár pre logy a súbor **bootstrap.php**, ktorý slúži na načítanie celého rozhrania a nastavenie aplikácie. Aktivuje sa tu *autoloading*[6], nastaví sa ladenie a smerovanie[24].

[3]Boostrap je HTML, css, javascript podporujúce rozhranie s výberom viacerých štýlov a foriem prezentácie, ktoré majú dobru kompatibilitu na viacerých platformách vrátane mobilných telefónov

[4]MVP(Model-view-presenter) - je model riadenia, logiky a výstupu, ktorý sa najviac využíva pri programovaní užívateľských rozhraní vrátane webových stránok.

[5]URL(Uniform Resource Locator) je všeobecný názov pre všetky typy mien a adries, ktoré odkazujú na objekty WWW(World Wide Web). Výraz webová adresa je synonymom pre URL adresu, ktorá používa protokol HTTP alebo HTTPS.

[6]autoloading v Nette načíta jedným príkazom všetky potrebné triedy pre celú aplikáciu spolu

Obr. 4.1: Adresárová kostra a štruktúra aplikácie v Nette Sandbox

```
www/                              ← koreňový adresár webu
└── InternetQualityStatistics/
    ├── app/                      ← adresár webové aplikácie
    │   ├── config/               ← konfiguračné súbory
    │   ├── model/                ← triedy modelovej vrstvy (Stats.php)
    │   ├── console/              ← triedy skriptov (AggregationsCommand.php)
    │   ├── presenters/           ← triedy presenterov (HomepagePresenter.php,
    │   │   │                                           ApiPresenter.php)
    │   │   └── templates/        ← šablóny (@layout.latte)
    │   │       └── Homepage/     ← (default.latte)
    │   ├── router/               ← konfigurácia URL adries
    │   └── bootstrap.php         ← spúšťací súbor
    │
    ├── bin/                      ← adresár s utilitami
    ├── log/                      ← chybové logy
    ├── temp/                     ← miesto pro dočasné súbory (cache, atď.)
    ├── test/                     ← adresár pre unit testy
    │
    ├── vendor/                   ← knižnice pre aplikáciu
    │   ├── nete/                 ← nete rozhranie
    │   ├── composer/             ← rozhranie na inštaláciu knižníc
    │   ├── kdyby/                ← rozhranie pre spúšťanie lokálnych skriptov
    │   ├── symfony/              ← ďalšie webové rozhranie a jeho funkcie
    │   └── tracy/                ← ladiace rozhranie
    │
    └── www/                      ← miestny koreňový adresár webu - iba tento
                                    adresár je prístupný z internetu
```

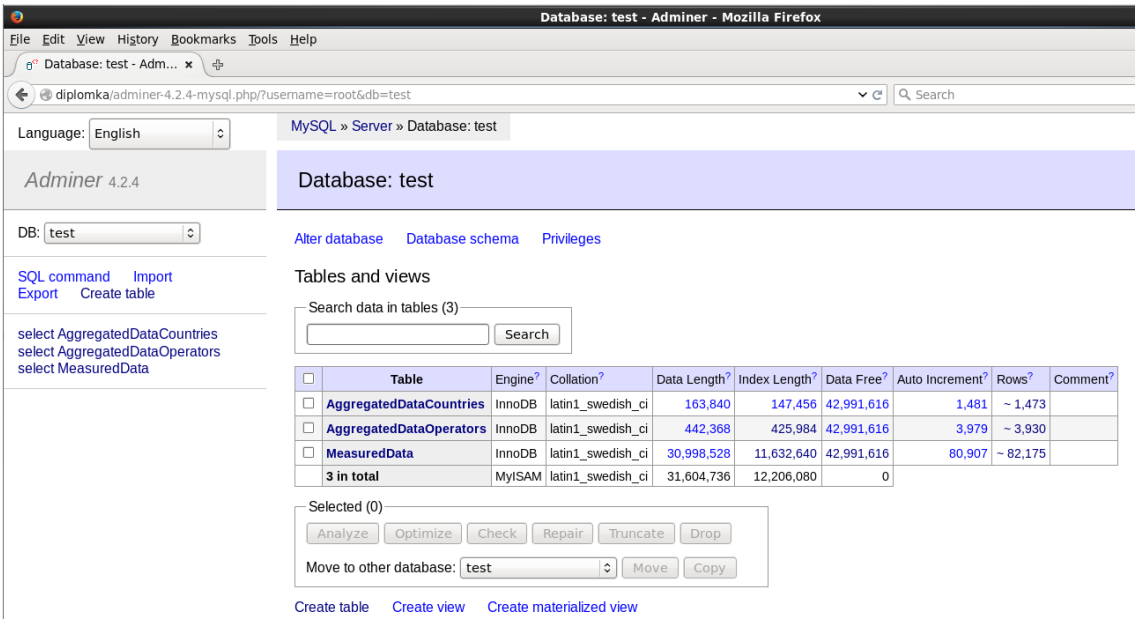
Okrem hlavného rozhrania Nette sa pri programovaní použilo rozhranie *kdyby* a *tracy*. **Kdyby** rozhranie je použité na vytvorenie spustiteľných skriptov napísaných v PHP programovacom jazyku. Pomocou tohto sa vytvoril agregačný skript spustiteľný z operačného systému, ktorý nepotrebuje webové rozhranie, ale stále využíva vlastnosti Nette na komunikáciu s databázou, aby agregoval príslušné dáta. Veľmi užitočným rozhraním pri programovaní je **tracy**, ktoré poskytuje ladiace nástroje. *Tracy* pri každom webovom požiadavku a spracovaní aplikáciou zobrazilo v grafickej forme užitočné informácie ako sú: čas, za ktorý sa celá požiadavka vykonala, využitie procesoru a pamäte servera, počet použitých súborov, verzia PHP a *Apache*, smerovanie v Nette, presný počet a zoznam databázových príkazov, čas, za ktorý sa tieto databázové príkazy vykonali. Pomocou týchto informácii pri počiatočnom testovaní sa zistila potreba indexov v tabuľkách. Informácie z tohto rozhrania sú použité hlavne v piatej kapitole, kde sa porovnávajú doby odozvy webovej stránky a práce s databázou.

Na poslednom mieste medzi rozhraniami použitými pri programovaní treba spomenúť **Adminer**. *Adminer* je plnohodnotný nástroj na správu databázy napísaný v programovacom jazyku PHP. Je ľahko spustiteľný cez webový prehliadač a už

aj s knižnicami tretích strán.

samotné rozhranie Nette má tento nástroj v základných balíčkoch. Podporuje databázové systémy ako sú napríklad: MySQL, PostgreSQL, SQLite, Oracle, SimpleDB a MongoDB. Dovoľuje vytváranie tabuliek, definíciu stĺpcov a funkcií[25]. Viz obrázok 4.2.

Obr. 4.2: Adminer webové rozhranie na správu databázy




## 4.2 Rozdelenie podľa funkcie a opis pomocou vývojových diagramov

Ako je spomenuté v predošlých kapitolách, celkový projekt vytvorený v praktickej časti diplomovej práce bude zastávať viacero funkcií. Tieto funkcie sú rozdelené podľa toho či komunikujú s užívateľom aplikácie a v akej forme. Medzi funkčnú oblasť aplikácie, ktorá nebude komunikovať žiadnym spôsob s návštevníkmi webovej aplikácie patrí **Agregačný skript**. Tento môže operovať čisto len na servery, kde sa nachádzajú zdrojové kódy celej aplikácie a databáza. Spustiteľný je iba v rámci lokálnych účtov na konkrétnom servery. Ďalšou funkčnou oblasťou je výstup vo forme **webovej stránky** a **API rozhrania**. Oba tieto výstupy sú prístupné akémukoľvek užívateľovi, ktorý navštívi špecifikované webové adresy. Vyžadujú interaktívny prístup zo strany užívateľov pre zadanie selekcie dát, ktoré chcú vidieť.

### 4.2.1 Agregačný skript

V teoretickej časti diplomovej práce v kapitole Real-time databáz sa rozobralo akými spôsobmi docieliť čo najrýchlejšiu odozvu databázy na užívateľské požiadavky. Z vlastných pracovných skúsenosti sa denne stretávam s objemnými databázami, ktoré obsahujú veľké množstvo nameraných dát. Väčšina týchto databáz používa agregáciu ako formu zálohovania starších dát a na rýchlejšie vyhľadávanie v tabuľkách. Preto som sa rozhodol, že dobrým spôsobom rýchlej práce s databázou bude najskôr si dáta pripraviť do formy v akej budú prezentované. Po dohode s vedúcim diplomovej práce sa rozhodlo agregovať dáta na určitom počte posledných nameraných dát. Najlepším spôsobom agregácie je výpočet priemernej a medián hodnoty pre tento počet dát a uloženie do novej tabuľky mimo nameraných dát. Tabuľky s prepočítanými dátami obsahujú niekoľkonásobne menej riadkov oproti tabuľke s nameranými dátami. Vďaka menšej tabuľke sú aj požiadavky v takejto tabuľke vykonané ďaleko rýchlejšie. Takže hlavným princípom zrýchlenia odozvy je prebehnutie agregácie, ktorá možno zaberie nejaký čas na vykonanie ale pripraví tabuľku, ktorá je rýchla a dostupná prezentačnej vrstve.

Pred samotnou agregáciou sa vytvoria cez príkazový riadok databázy dve tabuľky na ukladanie agregovaných dát. Tabuľku **AggregatedDataCountries**, kde sa ukladajú vypočítane priemerné a median hodnoty rýchlosti sťahovania, latencie a *qoe*[7] pre každú krajinu a mobilnú technológiu z tabuľky nameraných dát. Druhá tabuľka **AggregatedDataOperators** ukladá rovnaké typy hodnôt ako predošlá tabuľka ale výpočty sú pre každú krajinu, mobilnú technológiu a všetkých operátorov jednotlivých krajín. V časti nasledujúceho SQL kódu, ktorý sa použil je vidieť definíciu týchto tabuliek.

```
CREATE TABLE AggregatedDataCountries
(id INT(6) NOT NULL AUTO_INCREMENT PRIMARY KEY,
isoCountryCode VARCHAR(30) NULL, radioTechnology VARCHAR(30) NULL,
avgDownloadSpeed DOUBLE, avgLatency DOUBLE, avgQoe DOUBLE,
medDownloadSpeed DOUBLE, medLatency DOUBLE, medQoe DOUBLE,
calculated_date TIMESTAMP default CURRENT_TIMESTAMP,
INDEX isoCountryCode (isoCountryCode),
INDEX radioTechnology (radioTechnology),
INDEX calculated_date (calculated_date))
ENGINE=InnoDB;

CREATE TABLE AggregatedDataOperators
```

[7] QOE(Quality of Experience) meria ako je užívateľ spokojný s určitou službou.

```
(id INT(6) NOT NULL AUTO_INCREMENT PRIMARY KEY,
isoCountryCode VARCHAR(30) NULL, radioTechnology VARCHAR(30) NULL,
operator VARCHAR(30) NULL,
avgDownloadSpeed DOUBLE, avgLatency DOUBLE, avgQoe DOUBLE,
medDownloadSpeed DOUBLE, medLatency DOUBLE, medQoe DOUBLE,
calculated_date TIMESTAMP default CURRENT_TIMESTAMP,
INDEX isoCountryCode (isoCountryCode),
INDEX radioTechnology (radioTechnology),
INDEX operator (operator),
INDEX calculated_date (calculated_date))
ENGINE=InnoDB;

CREATE INDEX isoCountryCode ON MeasuredData (isoCountryCode);
CREATE INDEX radioTechnology ON MeasuredData (radioTechnology);
CREATE INDEX saved_date ON MeasuredData (saved_date);
CREATE INDEX operator ON MeasuredData (operator);
CREATE INDEX reachableVia ON MeasuredData (reachableVia);
CREATE INDEX Filtered ON MeasuredData (Filtered);
```

Z kódu vidieť, že už na začiatku definície tabuliek sa vytvoril primárny kľuč a indexy. Následne na konci sa definujú indexy v tabuľke s nameranými dátami. Výber konkrétnych indexov je ovplyvnený testovaním a tým, podľa akých stĺpcov sa vyhľadáva v každej použitej tabuľke. Napríklad v tabuľke **MeasuredData** s nameranými dátami sa vyhľadáva pre agregáciu vždy podľa krajiny, radio technológie, operátora a času merania. Preto sú na týchto stĺpcoch definované indexy. Zvyšné definície indexov sú pri použití zložitejších selekcií a na odfiltrovanie falošných dát nameraných cez bezdrôtové pripojenie *Wi-Fi*.

Agregačný skript je naprogramovaný pomocou Nette rozhrania a na jeho spustiteľnosť je použité v Nette ďalšie rozhranie *Kdyby*. Nachádza sa v adresári console pod názvom **AggregationsCommand.php**, čo je možno vidieť na obrázku 4.1. Štruktúra tohto skriptu sa delí na tri časti:

1. pomocné funkcie,
2. inicializácia premenných a načítanie databázy,
3. cyklus kde sa načítané dáta prepočítavajú a ukladajú do agregačných tabuliek.

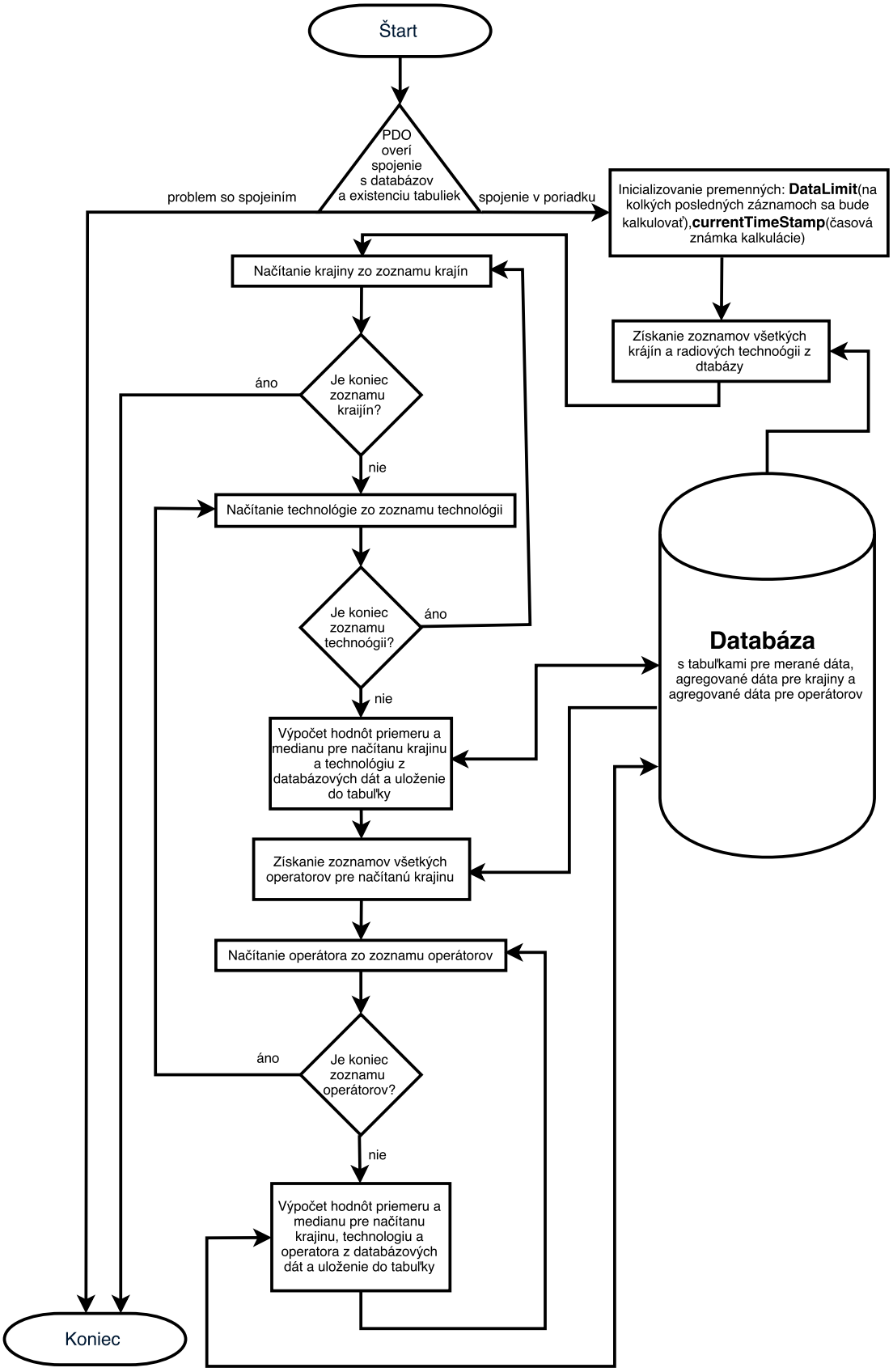
V časti inicializácie je zadefinovaný počet posledných nameraných záznamov, pre ktorý prebehne agregácia a výpočty. Všetko načítavanie a selekcia dát z databázy nie len v agregačnom skripte je napísaná pomocou syntaxi rozhrania Nette ako v príklade na obrázku 4.3. Celková funkcionalita výsledného agregačného skriptu je dobre opísaná pomocou vývojového diagramu 4.4. Na začiatku sa vždy overí

Obr. 4.3: Príklad použitia Nette syntaxi na selekciu z databázy

```
// Getting list of all existing radio technology types from MeasuredData table
$mData = $this->database->table('MeasuredData');
$radioTechList = $mData->select('DISTINCT radioTechnology')
                ->order('radioTechnology ASC')
                ->fetchPairs('radioTechnology', 'radioTechnology');
```

komunikácia s databázou a prebehne inicializácia premenných. Následne sa zistia zoznamy všetkých krajín 4.3 a rádiových technológií a začne sa iterovať na týchto zoznamoch. Aby sa vypočítali agregované hodnoty aj pre všetkých operátorov, načíta sa príslušnou selekciou v nameranej tabuľke zoznam všetkých operátov v iterácii pre jednotlivú krajinu. V každej iterácii pre jednu krajinu a technológiu sa selekciou získajú pre túto krajinu a technológiu dáta z tabuľky s nameranými hodnotami, vypočítajú sa priemerné a medián hodnoty pre merané veličiny a zapíšu do tabuľky **AggregatedDataCountries**. Pred zapísaním do tabuľky sa overí či už existujú dáta pre túto krajinu a technológiu. V prípade, že áno dáta sa len aktualizujú o nové vypočítané hodnoty a časovú známku merania. V opačnom prípade sa vytvorí celý nový riadok do tabuľky. Rovnaký princíp je použitý aj na vypočítanie hodnôt operátorov, iba tieto výpočty sa spracujú v iteráciách pre jednotlivých operátorov a uložia do tabuľky **AggregatedDataOperators**. Po ukončení behu agregačného skriptu sú všetky časové známky a vypočítané hodnoty v agregačných tabuľkách aktuálne. Aby bol beh skriptu úspešný je nutné overiť existenciu tabuliek, ktoré sú opísané na začiatku tejto sekcie.

Obr. 4.4: Vývojový diagram agregačného skriptu



### 4.2.2 Back-end podpora

Prezentácia dát formovou webovej stránky je dobrý spôsob z užívateľského hľadiska a pre ľudí, ktorý hľadajú grafický výstup. Avšak väčšina aplikácii na webe ponúka tiež takzvanú *Back-end* podporu. Ta je v mnoho prípadoch vo forme *API*. API je aplikačné programové rozhranie. Znamená to, že výstup aplikácie nemusí byť v grafickej forme ale vo forme dát, ktoré sa dajú strojovo spracovať inou aplikáciou. Hlavným dôvodom vytvorenia back-end podpory aplikácie v praktickej časti diplomovej práce je tvorba rozhrania, ktoré môže byť ďalej použité v iných projektoch a čerpať agregované dáta na ďalšie výpočty a porovnania. Ďalším dobrým dôvodom je špecifická aplikácia, ktorá by vykresľovala grafy na iných platformách a jednoducho by čerpala dáta z agregačných tabuliek pomocou tohto API rozhrania.

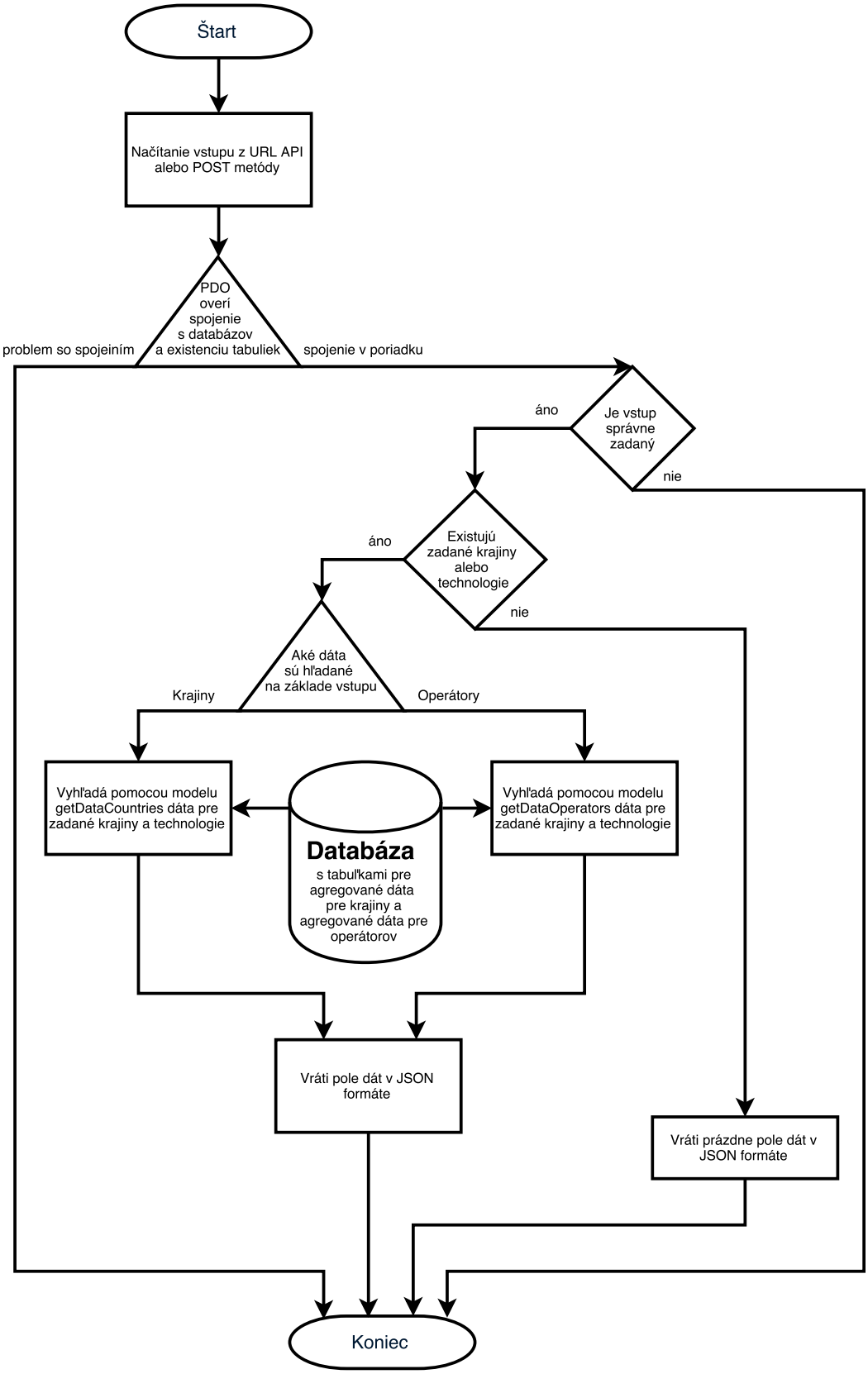
Funkciou API sú ľahko nastaviteľné vstupné parametre a výstup v univerzálnej strojovo čitateľnej forme. Zdrojový kód tohto rozhrania sa nachádza v adresári presenters pod názvom **ApiPresenter.php** 4.1. Štruktúra API naprogramovaného kódu je jednoduchá vďaka vbudovaným PHP a Nette funkciám, ktoré kontrolujú a poskytujú vstupné dáta. Podľa vstupného parametra, ktoré vychádza z toho ako sa rozhranie API zavolá, sú v kóde dve základné funkcie:

- získavanie dát z agregačnej tabuľky pre jednotlivé krajiny,
- získavanie dát z agregačnej tabuľky pre jednotlivých operátorov.

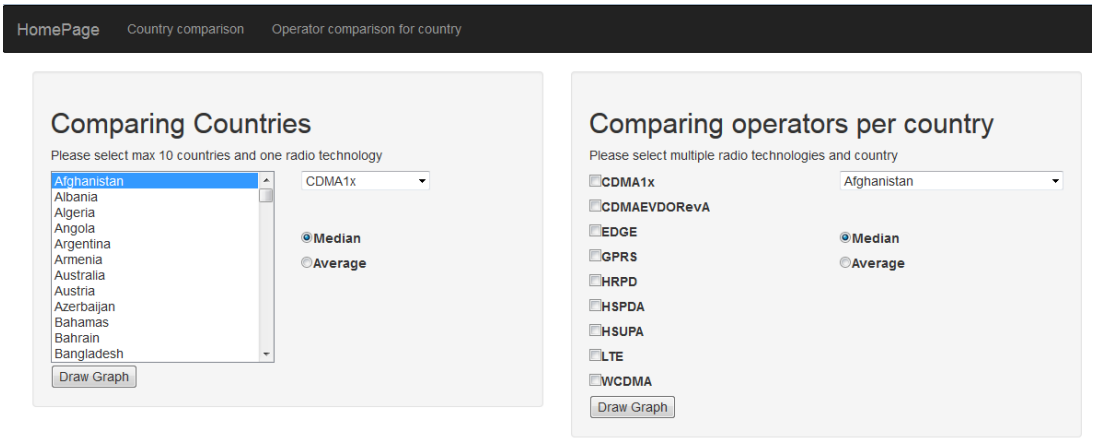
Podobne ako v prípade agregačného skriptu je výsledná funkcionalita a správanie back-end rozhrania opísane vo forme vývojového diagramu 4.5. Na základe vstupu je rozpoznaná, ktorá funkcia bude použitá na získavanie dát z databázy. V oboch funkciách je na začiatku overenie vstupu. Kontrolujú sa zadané krajiny a rádiové technológie. Viac o použití tohto rozhrania a formách vstupu je v sekcii(link na návod na použitie). Po overení vstupu sú podľa vstupu použité funkcie na komunikáciu s databázou a preformátovanie výstupu do JSON[8]. Z dôvodu čistého a jednoduchého kódu je vytvorený v adresári modul trieda **Stats.php** 4.1, v ktorej sú funkcie **getDataCountries** a **getDataOperators**. Získavanie dát z agregačných tabuliek je použité aj v kóde webovej stránky. Vďaka triede Stats sa odstránil zbytočný duplicitný kód a poskytol priestor pre lepšiu údržbu kódu. Výstupom spomenutých funkcii sú špecifické dáta pre hľadané krajiny, technológie a operátorov. V poslednom kroku API rozhrania sú dáta poskytnuté užívateľovi v čistom formáte JSON.

[8]JSON(JavaScript Object Notation) – je odľahčený formát na výmenu dát. Je dobre čítateľný pre človeka a zároveň dobre spracovateľný pre programy a aplikácie[26]

Obr. 4.5: Vývojový diagram API



Obr. 4.6: Domovská stránka a jej ponuka




### 4.2.3 Webová stránka

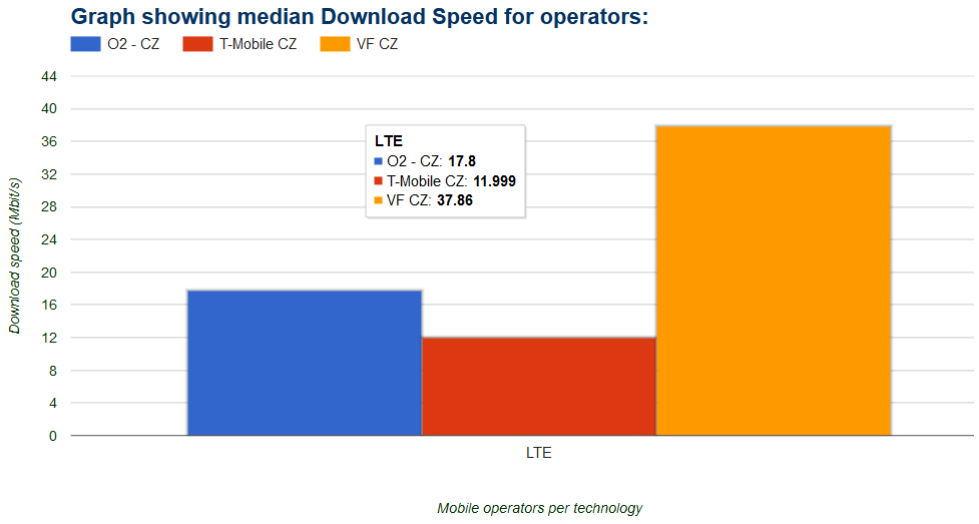
Webová stránka patrí medzi najzložitejšiu partiu praktickej časti diplomovej práce. Okrem toho, že naprogramovaný kód tejto časti zaberá približne 540 riadkov, zároveň je potrebné správne nastavenie šablón a použitie JavaScript. Rovnako ako API rozhranie sa funkčná časť webovej stránky nachádza v adresári presenters pod názvom **HomepagePresenter.php** 4.1. Cieľom webovej stránky je v grafickej podobe zobraziť dáta z databázy s čo najrýchlejšou odozvou. Preto webová stránka čerpá dáta z agregačných tabuliek a nie z tabuľky s nameranými dátami. Takto ušetrí veľké množstvo času vďaka jednoduchým a rýchlim selekciám do pripravených tabuliek. Opis životného cyklu webovej stránky môžeme sledovať na vývojovom diagrame 4.8.

Na základe tohto vývojového diagramu 4.8 sa delí funkčnosť webovej stránky na nasledovné služby:

1. **Inicializácia a načítanie ponuky** webovej stránky. V momente kedy užívateľ otvorí príslušnú webovú adresu sa na pozadí vykoná inicializácia dát a poskytne v grafickej podobe ponuku 4.6. Väčšina behu pri načítaní základnej stránky je uskutočnená na webovom servery Apache, ktorý je súčasťou servera kde sa nachádza databáza a zdrojové kódy celého projektu. Ponuka sa skladá z dvoch komponentov. V jednom je možnosť **porovnania krajín**. V základnom nastavení si užívateľ smie vybrať maximálne desať krajín, rádiovú technológiu, ktorú chce porovnať a spôsob agregácie. Druhá komponenta poskytuje **porovnanie operátorov pre jednu krajinu**. Na výber je ponuka viacerých rádiových technológii, ktoré sa vyskytujú v nameraných dátach, krajina a spôsob agregácie 4.6. Jednotlivé zoznamy krajín a technológii sú načítané z agregačných tabuliek. To zabezpečuje, že v prípade nových krajín alebo technológii sa ponuka aktualizuje sama.

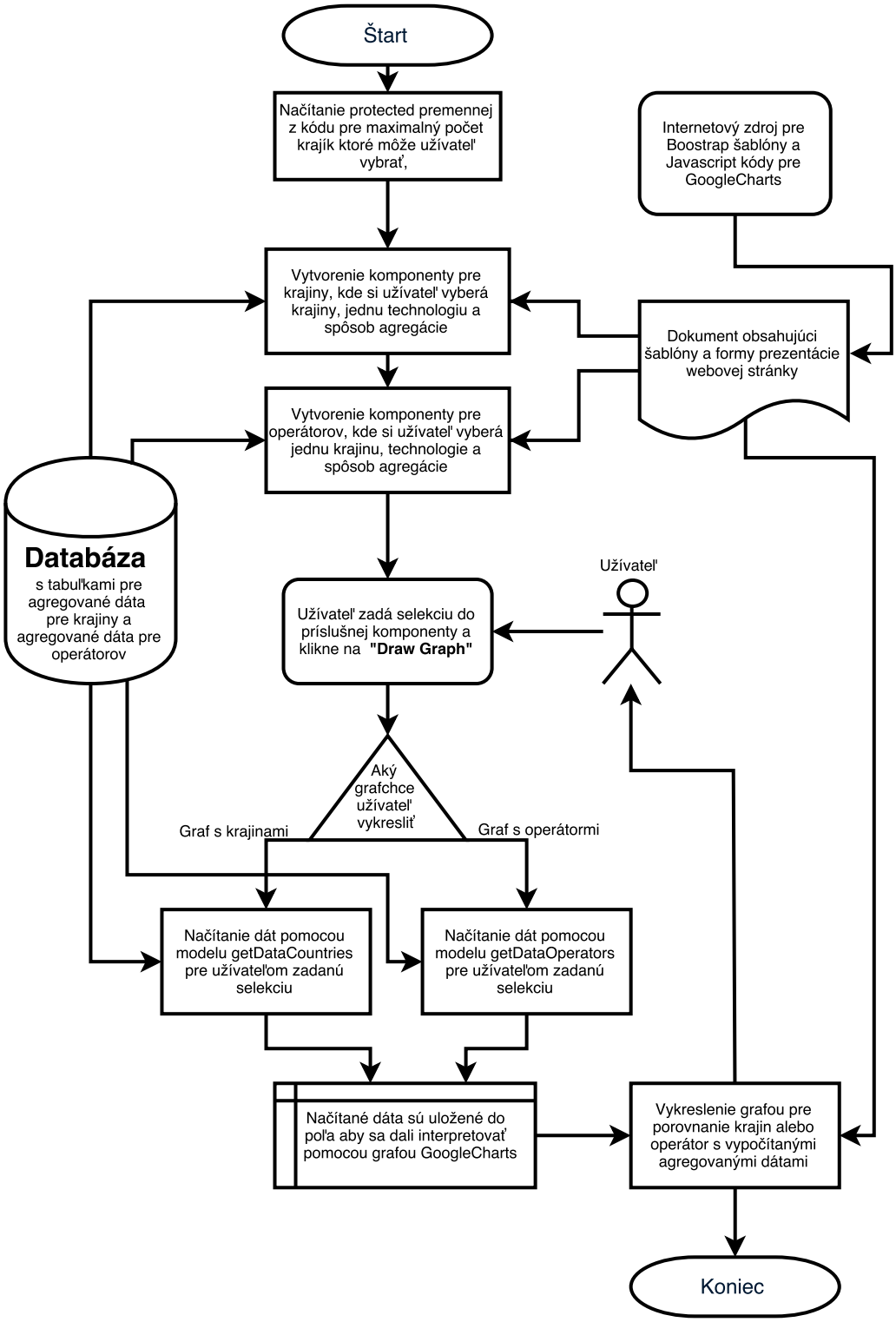
2. **Zaslanie užívateľovej požiadavky a vyhľadanie dát** z databázy. Užívateľ potvrdí svoj výber pomocou tlačidla **Draw Graph**. Vykreslenie grafov môže byť buď len pre krajiny alebo pre operátorov. Po zaslaní požiadavky aplikácia podľa toho či ide o porovnanie krajín alebo operátorov použije jednu z funkcií triedy Stats, ktorá sa používa rovnako ako v prípade API rozhrania spomenutej v sekcii 4.2.2. Získané dáta z agregačných tabuliek sa uložia do správneho dátového formátu čitateľného pre vsádzanie dát do grafu.
3. **Vykreslenie grafov** prebehne potom ako sú databázové dáta poslané v správnej forme do šablóny webovej stránky. Na tomto mieste, konkrétne v súbore **presenters/Homepage/default.latte** je použitý JavaScript kód na vykreslenie grafov od spoločnosti Google. Na webovej stránke používam voľne dostupný JavaScript kód *Column Chart* definovaný na vývojárskych stránkach https://developers.google.com. Pomocou šablóny si špecifikujem výzor a opisy grafu pre každý graf. Správne sa vždy vykreslia tri grafy pre každú meranú veličinu, ktorá agregujem. Vo výsledku sú to grafy, ktoré porovnávajú rýchlosť sťahovania, latenciu a *MOS*[9]. Grafy vykreslené pomocou JavaScript nezaťažujú webový server kde sa webová aplikácia nachádza ale využívajú zdroje stanice, z ktorej užívateľ pristupuje na web. Vďaka tomu sa urýchľuje vykreslenie grafov a taktiež sa šetria zdroje servera. Výhoda tohto riešenia sa ocení hlavne v prípade veľkého množstva požiadavkou na vykreslenie grafov. Webový server bude len posielať hľadané dáta a užívateľská stanica ich vo svojom webovom prehliadači vykreslí. Príklad vykresleného grafu vidieť na obrázku 4.7.

Obr. 4.7: Príklad grafu na webovej stránke




[9]MOS(Mean opinion score) je test, ktorý hodnotí na základe užívateľského zážitku kvalitu mobilného spojenia. Môže naberať hodnoty od 1-5, päť znamená najlepšie a jednotka najhoršie

Obr. 4.8: Vývojový diagram webovej prezentačnej stránky



## 4.3 Nasadenie aplikácie na server

Aplikácia so všetkými jej komponentami a rozhraniami je nainštalovaná na školskom servery **wislabres.utko.feec.vutbr.cz** s parametrami:

- **Debian Jessie(Stable)**,
- **Apache 2.4.10**,
- **MySQL 5.5.49**,
- **PHP 5.6.20**.

Verzie všetkých komponentov spĺňajú nároky rozhrania Nette udané na webovej stránke `https://doc.nette.org/cs/2.4/requirements`.

Pred inštaláciou bol celý projekt webovej aplikácie so všetkými jej funkčnými časťami a agregačným skriptom uložený pomocou zálohovacej aplikácie *GIT* do schránky na webovej adrese `https://github.com/JanGardian/diplomka`. Na tomto mieste sa tiež nachádza stručný návod `https://github.com/JanGardian/diplomka/blob/master/README.md` ako celý projekt uložiť a spojazdniť na cieľovom servery. Vďaka rozhraniu *Composer* dokázal git nainštalovať na školský server všetky potrebné a použité knižnice ku všetkým rozhraniam. Hlavný adresár projektu kde sa nachádza Nette kostra je pod adresárovou štruktúrou **/home/wislab/www/InternetQualityStatistics/** 4.1. Následne sa do konfiguračného súboru **app/config/config.local.neon** zapísali poskytnuté prihlasovacie údaje do databázy. Cez príkazový riadok MySQL databázy sa vytvoria potrebné agregačné tabuľky a indexy pomocou SQL príkazov spomenutých v sekcii 4.2.1. Nette vyžaduje mať vlastnú doménu na to aby fungovali všetky jej komponenty. Vďaka úsiliu vedúceho diplomovej práce bola univerzitou pridelená nová doména, ktorá zabezpečila, aby sa nenarušil žiadny web, ktorý už existoval na poskytnutom školskom servery. Nová doména je **http://wislabstats.utko.feec.vutbr.cz/**. Po otestovaní pridelenej domény sa manuálne spustil agregačný skript podľa návodu spomenutého v nasledujúcej sekcii. Keď tento skript prebehol, overilo sa databáza a existujúce dáta v nových agregačných tabuľkách. V aplikácii *cron* na servery sa naplánovalo, aby agregačný skript prebehol každý deň o dvanástej hodine na obed stredoeurópskeho času.

Finálnym produktom praktickej časti diplomovej práce je webová stránka dostupná z adresy **`http://wislabstats.utko.feec.vutbr.cz/`** a back-end podpora dostupná z adries **`http://wislabstats.utko.feec.vutbr.cz/api/countries`** a **`http://wislabstats.utko.feec.vutbr.cz/api/operators`**. Viac o spôsobe použitia back-end alebo API rozhrania je v nasledujúcej sekcií.

## 4.4 Návod na použitie a prípadnú konfiguráciu

V tejto sekcií je stručne opísaná dostupná konfiguráciu a spôsob použitia všetkých komponentov finálnej aplikácie.

### 4.4.1 Použitie agregačného skriptu

Agregačný skript sa spustí z CLI[10] rozhrania na školskom servery kde je aplikácia nainštalovaná. Užívateľ musí byť prihlásený ako super užívateľ **root**. Príkaz na spustenie agregácie:

- **php /home/wislab/www/InternetQualityStatistics/www/index.php app:aggregations**

Pravidelný beh agregácie je nastavený v programe *cron*, ktorý môžeme editovať podľa príručky na webovej stránke `https://linuxconfig.org/linux-cron-guide`. Aktuálne nastavenie pre agregačný skript v cron aplikácii vyzerá nasledovne:

- **0 12 * * * /usr/bin/php home/wislab/www/InternetQualityStatistics/www/index.php app:aggregations >>/home/wislab/www/InternetQualityStatistics/log/aggregation.log 2>&1**

Znamená to, že skript je spúšťaný každý deň o dvanástej hodine stredoeurópskeho času. Časová zóna je určená podľa nastavenia samotného servera.

Na zmenu počtu posledných záznamov, na ktorých prebieha jednotlivá agregácia je treba zmeniť nastavenie premennej **$dataLimit = 1000;** v súbore **InternetQualityStatistics/www/app/console/AggregationsCommand.php**.

### 4.4.2 Použitie Back-end podpory

Na použitie back-end podpory je dopredu vedieť akým spôsobom sa budú predávať rozhraniu vstupné informácie obsahujúce zoznamy krajín a technológií. V API rozhraním podporujem dva spôsoby predávania vstupných dát. Pomocou metódy **getParameter**, ktorá načíta dáta priamo zo zadanej webovej adresy *URL*. Druhá metóda je pomocou metódy **getPost**, ktorá načíta vstupné dáta vo forme pridaného poľa dát ku zadávanej webovej adrese API. Spôsob akým bude API rozhranie prijímať a spracovávať vstupné dáta nakonfigurujeme tak, že v súbore **InternetQualityStatistics/www/app/presenters/ApiPresenter.php** od komentujem použitie vybranej metódy a druhú za komentujem. Aktuálne je nastavená metóda *getParameter*:

$radTechs = (array)$this->request->getParameter('tech');
$countries = (array)$this->request->getParameter('country');

[10]CLI(Command-line interface) je užívateľské rozhranie s príkazovým riadkom. Ovládanie a zadávanie príkazov alebo spúšťaní programov je možné len z príkazového riadku.

//$radTechs = $this->request->getPost('tech');

//$countries = $this->request->getPost('country');

Príklad použitia API pomocou metódy *getParameter* a definovania si URL s výstupnými dátami:

- z tabuľky agregovaných dát pre krajiny `http://wislabstats.utko.feec.vutbr.cz/api/countries?country[0]=cz&tech[0]=LTE&tech[1]=gprs`,
- z tabuľky agregovaných dát pre operátorov `http://wislabstats.utko.feec.vutbr.cz/api/operators?country[0]=cz&country[1]=es&tech[0]=LTE`.

### 4.4.3 Použitie domovskej webovej stránky

Hlavná webová stránka sa nachádza na webovej adrese `http://wislabstats.utko.feec.vutbr.cz/`. Už samotná ponuka na tejto stránke viz obrázok 4.6 zreteľne vysvetľuje stručnú ponuku selekcie v oboch častiach.

Ak chce užívateľ porovnávať krajiny môže si vybrať maximálne 10 krajín, jednu technológiu a spôsob agregácie. Po kliknutí na tlačidlo **Draw Graph** sú vykreslené grafy porovnávajúce vybrané krajiny. V prípade, ak vyberá užívateľ viaceré krajiny pomocou stlačenej klávesy *CTRL* jedenástu krajinu nedovolí zakliknúť. V prípade výberu pomocou stlačenej klávesy *SHIFT* môže užívateľ vybrať viac ako desať krajín, avšak do grafov sa vykreslia len dáta prvých desiatich zo selekcie.

Pri porovnávaní operátorov môže užívateľ vybrať akýkoľvek počet dostupných technológii a jednu krajinu. V grafoch sa vykreslia dáta pre všetkých operátorov vybranej krajiny a pre jednotlivé technológie. Dobrým príkladom vykreslenia je výber technológií pre krajinu *United States*, ktorá má z nameraných dát približne 20 rôznych operátorov. Vždy sa vykreslia tri grafy. Prvý graf porovnáva krajiny alebo operátorov a rýchlosť sťahovania pomocou vybraných rádiových technológií. Druhý graf zobrazuje rovnaké porovnanie ale pre latenciu alebo oneskorenie počas sťahovania. Na poslednom grafe je pre zobrazené porovnanie veličiny MOS, ktorá je výsledkom užívateľského hodnotenia kvality rádiového spojenia. MOS môže naberať hodnoty od 1-5, pričom jednotka je najhoršia kvalita a päťka najlepšia.

# 5 TESTOVANIE APLIKÁCIE A POROVNÁVANIE VÝSLEDKOV

Záverečnej kapitole sa venuje testovaniu všetkých naprogramovaných súčasti praktickej častí diplomovej práce. Na prvom mieste sa testuje načítanie a rôzne situácie finálnej webovej stránky a API rozhrania. Následne sú vykonané merania na testovacom servery pre dva prípady. V prvom prípade nie sú použité prídavne indexy v databáze. Uchováva sa len primárny kľuč vo všetkých tabuľkách. Meranými hodnotami sú rýchlosť vykonania celkovej požiadavky a počet databázových transakcií. Podobne spracované je aj druhé meranie, ale v tomto prípade sú požité všetky indexy ako sú definované vo finálnej forme tabuliek použitých na školskom servery.

## 5.1 Odozva a zobrazenie webovej stránky a API rozhrania

Na webovej stránke `http://wislabstats.utko.feec.vutbr.cz/api/countries` sa testovali nasledujúce funkcie:

- načítanie a obnovenie domovskej stránky vo webovom prehliadači,
- rôzny výber ponuky a vykreslenie grafov porovnávajúcich krajiny,
- rôzny výber ponuky a vykreslenie grafov porovnávajúcich operátorov.

Všetky testované funkcie prebehli bez problémov zobrazenia vo webových prehliadačoch: *Firefox*, *Opera*, *Chrome* a *Internet Explorer*.

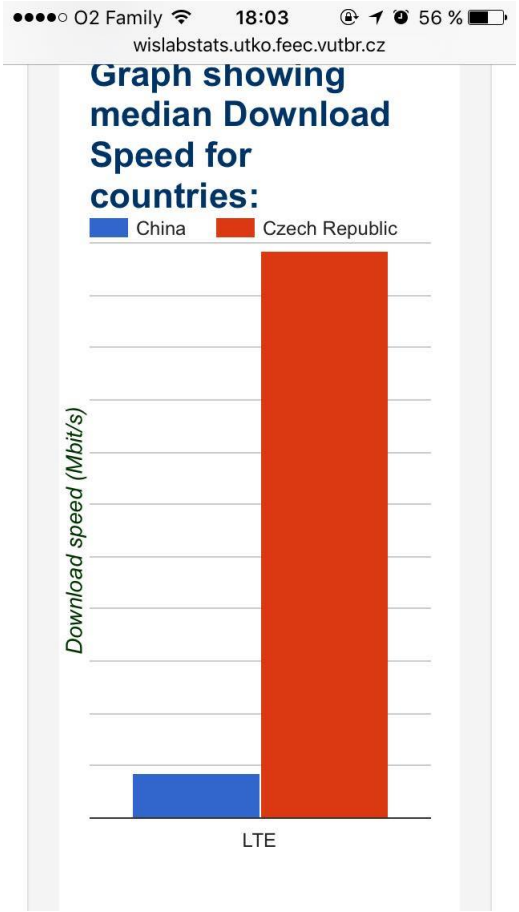
Pre API rozhranie sa testoval rôzny vstup krajín a operátorov v oboch prípadoch API. V Prípade, že nie sú zadané krajiny alebo technológie vráti API prázdne pole dát v JSON formáte. Do vstupu sa krajiny zapisujú vo formáte **ISO country code** a spolu s technológiami nezáleží na použití veľkých alebo malých písmen. Testy vstupu pomocou URL a výstupu vo forme JSON prebehli úspešne na všetkých zmienených testovaných webových prehliadačoch. Príklad testovaných URL adries:

- `http://wislabstats.utko.feec.vutbr.cz/api/countries?country[]=cz&country[]=es&country[]=Us&tech[]=LTE&tech[]=EDGE&tech[]=gprs`
- `http://wislabstats.utko.feec.vutbr.cz/api/operators?country[0]=cz&country[1]=es&country[2]=Us&tech[0]=LTE&tech[1]=EDGE&tech[2]=gprs`

Počas testovania webovej stránky sa zistilo, že webová stránka je plne funkčná a podporovaná aj v mobilných zariadeniach používajúcich operačný systém **Android** a **iOS** viz obrázok 5.1. Podpora týchto zariadení a operačných systémov je funkčná vďaka použitým rozhraniam Nette, *Bootsrap* a JavaScript. Zavedené rozhrania sú

vďaka dobrej podpore a neustálemu vývoju prispôsobené behu a zobrazovaniu v zmienených mobilných platformách.

Obr. 5.1: Test webovej stránky na iPhone zariadení

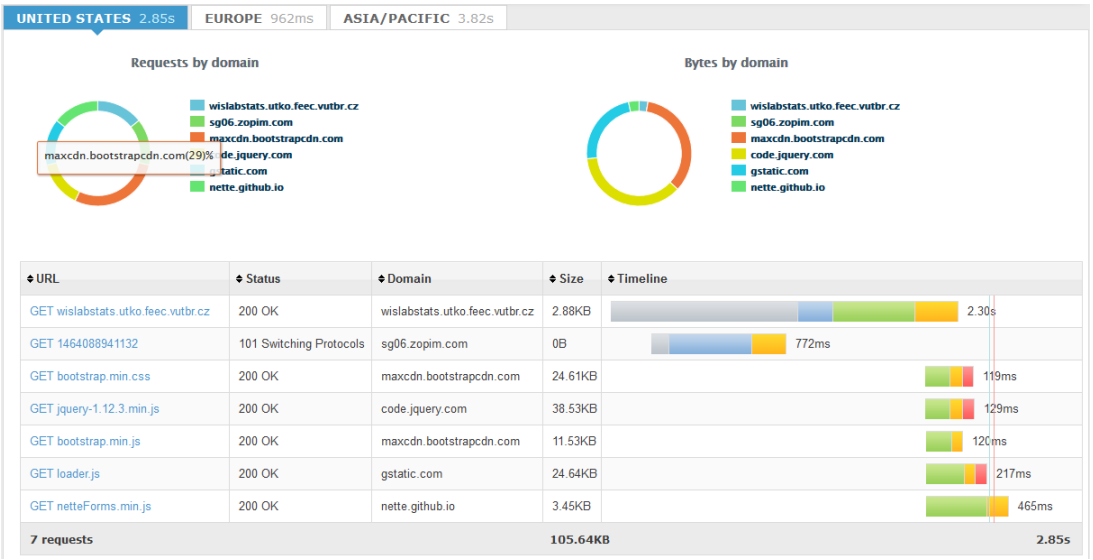



## 5.2 Meranie rýchlosti odozvy webovej stránky

Rýchlosť načítania finálnej webovej stránky nezávisí len od funkcionality stránky ale hlavne od dostupnosti a komunikácii užívateľov s webovým serverom odkiaľ web funguje. Na test hlavnej stránky sa použila voľne dostupná online aplikácia **Full page load tester**, ktorá sa nachádza na webovej stránke `http://www.monitis.com/pageload/`. Výsledky testu pre rôzne kontinenty možno vidieť na obrázku 5.2. Najrýchlejšie sa stránka načítala v Európe s časom 962 ms kde sa server fyzicky nachádza. Obrázok 5.2 ukazuje detailnejšiu správu pre Severnú Ameriku s časom načítania 2,85 s. V Ázii sa na webovú stránku dostanú za 3,82 s.

Druhý test rýchlosti webovej stránky prebehol na testovacom servery kde bola aplikácia vyvíjaná. V tomto prípade boli použité štatistiky poskytnuté ladiacim rozhraním **Tracy** používaného v Nette. Ako dôkaz užitočnosti indexov v databáze

Obr. 5.2: Test rýchlosti načítania webovej stránky




sa merala rýchlosť a načítanie hlavnej stránky a spracovanie požiadavky vykreslenia v dvoch stavoch. V prvom stave sa nepoužili indexy v databáze a v druhom sa následne zaviedli. Testovala som požiadavka s najväčším počtom dát vo výstupe, pri ktorej si vybrali všetky rádiové technológie a krajinu *United States* a následne nechali vykresliť grafy porovnávajúce operátorov. Výsledky tohto merania vidieť v tabuľke 5.1.

Tab. 5.1: Tabuľka s meraniami rýchlosti odozvy a databázových operácii

| | Databáza bez indexov | | Databáza s indexami | |
|---|---|---|---|---|
| | celkový čas | čas a počet selekcií DB | celkový čas | čas a počet selekcií DB |
| Načítanie stránky | 198,1 ms | 16,1 ms, 4 selekcie | 113,0 ms | 2,5 ms, 4 selekcie |
| Vykreslenie grafov | 945,5 ms | 616,9 ms, 195 selekcií | 468,0 ms | 90,6 ms, 195 selekcií |

## 5.3 Agregačný skript a indexy

Keďže agregačný skript najviac pracuje s databázou a prechádza všetky dáta v tabuľke nameraných hodnôt je dobrým kandidátom na testovanie zaťaženia databázy a rýchlosti samotnej agregácie. I napriek tomu, že tento skript v produkčnom prostredí prebehne len raz, môže svojim behom drasticky vyčerpať systémové zdroje servera, na ktorom je použitý. Z tohto dôvodu sa testuje jeho beh s databázou bez

indexov a s indexami. Podobne ako v predošlej sekcii sa test odohral na testovacom servery, aby neafektoval finálnu aplikáciu a tabuľku kde sa denne ukladajú nové namerané hodnoty. Výsledky testu možno vidieť v tabuľke 5.2.

Tab. 5.2: Tabuľka s meraním doby behu agregačného skriptu

| | Databáza bez indexov | Databáza s indexami |
|---|---|---|
| reálny čas behu agregačného skriptu | 1 h 30 m 14,488 s | 4 m 50,322 s |

Z časových hodnôt v tabuľke 5.2 je zreteľne vidieť, že použitie Indexov pomohlo značne urýchliť beh agregačného skriptu. Počas behu tohto skriptu využíva **mysql** proces priemerne viac ako 90% procesoru servera. V prípade ak by agregácia bežala viac ako 10 minút mohlo by to spomaliť funkcie ostatných aplikácií vrátane samotnej webovej stránky. Na porovnanie medzi testovacím serverom a univerzitným serverom, kde je aplikácia nainštalovaná, prebehne agregácia na testovacom servery za 4 minúty 50 sekúnd a na univerzitnom servery za 2 minúty 34 sekúnd.

# 6 ZÁVER

Vo výsledku diplomovej práce bola vytvorená serverová aplikácia na spracovanie a interpretáciu dát z databázy. Za účelom naprogramovať aplikáciu, ktorá by spĺňala nároky na rýchlu odozvu a spracovanie veľkého množstva dát bola spracovaná teória databáz pracujúcich v reálnom čase. V tejto teoretickej časti boli opísané vlastnosti indexov v databáze a možnosti upraviť často používané dáta pomocou agregácie. Poznatky z tejto teoretickej časti boli využité v nasledujúcej kapitole o realizácii aplikácie. Touto kapitolou bol načrtnutý samotný návrh, konštrukcia aplikácie, odôvodnené finálne výbery dostupných programovacích a aplikačných rozhraní a opísane jednotlivé funkčné oblasti finálnej aplikácie. Výsledkom návrhu bolo rozdelenie serverovej aplikácie na tri komponenty: agregačný skript, back-end podpora a webová stránka. Tieto komponenty boli naprogramované pomocou programovacieho jazyka *PHP* a webového rozhrania *Nette*. Konečná aplikácia spĺňa požiadavky zo zadania diplomovej práce, je plne funkčná a bola nasadená na školský server. Hlavný komponent na interpretáciu dát je vo forme webovej stránky dostupnej z adresy `http://wislabstats.utko.feec.vutbr.cz/`. Ako formát interpretácie boli použité grafy porovnávajúce jednotlivé dáta z databázy. Grafy čerpajú dáta z tabuliek, na ktorých prebehol proces agregácie. Počas štádia testovania sa dokázal význam použitia indexov v tabuľkách databázy, kedy serverová aplikácia dokázala rýchlo spracovať databázové požiadavky. Rozdiel doby behu agregačnéhe skriptu nad databázou, ktorá používa indexy a ktorá nepoužíva indexy činil až 1 hodinu a 25 minút. Výsledný čas agregácie na univerzitnom servery, kde je aplikácia nainštalovaná bol počas testovania menší ako 3 minúty.

Vďaka výberu webového rozhrania a použitých webových komponentov počas programovania aplikácie je finálna stránka spustiteľná vo viacerých platformách a webových prehliadačoch vrátane mobilných zariadení s operačným systémom typu *Android* a *iOS*.

Vytvorená aplikácia použila agregáciu vo forme výpočtov priemeru a mediánu. Do budúcnosti by mohlo byť vytvorené rozšírenie, ktoré by agregegovalo aj po časových intervaloch alebo rozšírenie poskytnutých informácií vo forme *histogramu*.

# LITERATÚRA


[1] *DATABASE MANAGEMENT* [online]. University of Massachusetts Boston, 2011 [cit. 2016-05-20]. Dostupné z: http://www.cs.umb.edu/cs630/hd1.pdf. University of Massachusetts Boston.

[2] DYER, Russell J. T. *MySQL in a nutshell*. 2nd ed. Sebastopol, CA: O'Reilly, c2008. ISBN 978-059-6514-334.

[3] Top Reasons to Use MySQL. In: *MySQL.com* [online]. USA: Oracle, 2016 [cit. 2016-05-20]. Dostupné z: https://www.mysql.com/why-mysql/topreasons.html

[4] SCHNEIDER, Robert D. *MySQL database design and tuning*. Indianapolis, Ind.: MySQL Press, c2005. ISBN 06-723-2765-1.

[5] Real-time transaction scheduling in database systems. *Database and expert systems applications: 7th International Conference, DEXA '96, Zurich, Switzerland, September 9-13, 1996 : proceedings*. 1. New York: Springer, c1996, s. 633-643. ISBN 354061656X.

[6] EDITED BY AZER BESTAVROS, Victor Fay-Wolfe. *Real-Time Database and Information Systems: Research Advances* [online]. 1. Boston, MA: Springer US, 1997 [cit. 2016-05-20]. ISBN 978-146-1560-692. Dostupné z: http://www.springer.com/us/book/9780792380115

[7] An Overview of Real-Time Database Systems. *Real Time Computing* [online]. 1. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, s. 261-282 [cit. 2016-05-20]. ISBN 9783642880490. Dostupné z: http://ilpubs.stanford.edu:8090/39/1/1993-6.pdf

[8] CIOLOCA, Cecilia a Mihai GEORGESCU. Increasing Database Performance using Indexes. *Database Systems Journal* [online]. 2011, **2011**(2), 1-10 [cit. 2016-05-20]. ISSN 2069 - 3230. Dostupné z: http://www.dbjournal.ro/archive/4/2_Ciologa_Georgescu.pdf

[9] SCHWARTZ, Baron. a Jeremy D. ZAWODNY. *High performance MySQL*. 2nd ed. Sebastopol: O'Reilly, c2008. ISBN 978-059-6101-718.

[10] Optimization and Indexes. In: *MySQL Documentation* [online]. USA: Oracle, 2016 [cit. 2016-05-21]. Dostupné z: http://dev.mysql.com/doc/refman/5.7/en/optimization-indexes.html

[11] *Aggregation is bigger always better?*. London, 2009 [cit. 2016-05-20]. Dostupné také z: http://www.cosmo-one.gr/publications/Aggregation.pdf

[12] Aggregation methods and the data types that can use them. In: *Oracle Documentation* [online]. USA: Oracle, 2016 [cit. 2016-05-21]. Dostupné z: https://docs.oracle.com/cd/E40518_01/studio.310/studio_users/src/rsu_views_aggregation_methods.html

[13] BLOKDIJK, Gerard. *Client Server - Simple Steps to Win, Insights and Opportunities for Maxing Out Success*. 1. USA: Complete Publishing, 2015. ISBN 9781488895517.

[14] The Python Wiki. *The Python Wiki* [online]. USA: Python, 2016 [cit. 2016-05-21]. Dostupné z: https://wiki.python.org/moin/FrontPage

[15] SHAFER, Dan. Python in the enterprise: Pros and cons. In: *TechRepublic* [online]. USA: TechRepublic, 2002 [cit. 2016-05-21]. Dostupné z: http://www.techrepublic.com/article/python-in-the-enterprise-pros-and-cons/

[16] Learn About Java Technology. *Java* [online]. USA: Oracle, 2016 [cit. 2016-05-21]. Dostupné z: https://www.java.com/en/about/

[17] STROUSTRUP, Bjarne. *The C programming language*. Special ed. Reading, Mass.: Addison-Wesley, c2000. ISBN 02-017-0073-5.

[18] Introduction to the C# Language and the .NET Framework. *Microsoft Developer Network* [online]. USA: Microsoft, 2015 [cit. 2016-05-21]. Dostupné z: https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx

[19] About JavaScript. *Mozilla Developer Network* [online]. USA: MDN, 2015 [cit. 2016-05-21]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

[20] LERDORF, Rasmus. a Kevin. TATROE. *Programming PHP*. Sebastopol, CA: O'Reilly, c2002. ISBN 15-659-2610-2.

[21] About Ruby. *Ruby* [online]. Ruby community, 2003 [cit. 2016-05-21]. Dostupné z: https://www.ruby-lang.org/en/about/

[22] Seznámení s Nette Frameworkem. *Nette Dokumentace* [online]. Česká Republika: Nette comnunity, 2016 [cit. 2016-05-22]. Dostupné z: https://doc.nette.org/cs/2.4/getting-started#toc-proc-prave-nette-framework

[23] Databáze. *Nette Dokumentace* [online]. Česká Republika: Nette community, 2016 [cit. 2016-05-23]. Dostupné z: https://doc.nette.org/cs/2.4/database

[24] ČÁPKA, David. Úvod do Nette frameworku pro PHP. In: *ITnetwork.cz* [online]. Česká Republika, 2015 [cit. 2016-05-23]. Dostupné z: http://www.itnetwork.cz/php/nette/zaklady/uvod-do-php-frameworku-nette/

[25] *Adminer* [online]. USA, 2016 [cit. 2016-05-23]. Dostupné z: https://www.adminer.org/

[26] *The JSON Data Interchange Format* [online]. In: . 1. Ecma International, 2013, s. 1-14 [cit. 2016-05-23]. Dostupné z: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

# ZOZNAM PRÍLOH

# A OBSAH PRILOŽENÉHO CD

Obsahom priloženého CD je zabalený súbor **diplomová_práca_kód_aplikácie.zip**, ktorý obsahuje celú štruktúru projektu uloženej v balíčkovom systéme *github* a diplomová práca v elektronickej podobe formátu **PDF**.

Adresárová štruktúra je opísaná na obrázku 4.1.Hlavné časti naprogramovaného kódu su zobrazené červeným písmom na tomto obrázku. Medzi tieto časti patria:

- **app/model/Stats.php** - trieda s pomocnými funkciami na získavanie dát z tabuliek databázy,
- **app/console/AggregationsCommand.php** - agregačný skript,
- **app/presenters/HomepagePresenter.php** - funkcionalita domovskej webovej stránky,
- **app/presenter/ApiPresenter.php** - API rozhranie,
- **app/presenter/templates/@layout.latte** - základná šablóna, kde sú definované použité externé rozhrania,
- **app/presenter/templates/Homepage/default.latte** - šablóna pre domovskú webovú stránku.