

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE  
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-5382-52598

**VYUŽITIE GRAFOVEJ DATABÁZY V PRAXI  
BAKALÁRSKA PRÁCA**

**2017**

**Juraj Kubričan**

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE**  
**FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-5382-52598

**VYUŽITIE GRAFOVEJ DATABÁZY V PRAXI**  
**BAKALÁRSKA PRÁCA**

Študijný program: Aplikovaná informatika  
Číslo študijného odboru: 2511  
Názov študijného odboru: 9.2.9 Aplikovaná informatika  
Školiace pracovisko: Ústav informatiky a matematiky  
Vedúci záverečnej práce: Ing. Maroš Čavojský

**Bratislava 2017**

**Juraj Kubričan**



## ZADANIE BAKALÁRSKEJ PRÁCE

Študent: **Juraj Kubričan**  
ID študenta: **52598**  
Študijný program: **aplikovaná informatika**  
Študijný odbor: **9.2.9. aplikovaná informatika**  
Vedúci práce: **Ing. Maroš Čavojský**  
Miesto vypracovania: **Ústav informatiky a matematiky**

Názov práce: **Využitie grafovej databázy v praxi**

Jazyk, v ktorom sa práca vypracuje: **slovenský jazyk**

Špecifikácia zadania:

V dnešnej dobe sa okrem tradičných zaužívaných relačných databáz, využívajú aj menej známe grafové databázy, v ktorých sú dáta uložené odlišným spôsobom ako v relačných databázach. Cieľom práce je oboznámiť sa s jednotlivými predstaviteľmi grafových databáz, vybrať jedného a navrhnúť a implementovať využitie vybranej grafovej databázy na reálnom príklade.

Úlohy:

1. Naštudujte si literatúru ohľadom jednotlivých predstaviteľov grafových databáz
2. Vyberte jedného predstaviteľa grafových databáz
3. Navrhnite reálny príklad pre implementáciu grafovej databázy
4. Implementujte reálny príklad pre implementáciu grafovej databázy
5. Zhodnoťte a uveďte výhody použitia grafovej databázy oproti iným typom databáz (relačné, dokumentové,...) v implementovanom reálnom príklade

Zoznam odbornej literatúry:


1. Ian Robinson, Jim Webber, and Emil Eifrem: Graph Databases, O'Reilly Media, Inc. USA 2015, p.224, ISBN: 978-1-491-93200-1


Riešenie zadania práce od: **19. 09. 2016**

Dátum odovzdania práce: **19. 05. 2017**

  
**Juraj Kubričan**  
študent

SLOVENSKÁ TECHNICKÁ UNIVERZITA  
V BRATISLAVE  
Fakulta elektrotechniky a informatiky  
Ústav informatiky a matematiky  
Ilkovičova 3, 812 19 Bratislava

  
**prof. RNDr. Otokar Grošek, PhD.**  
vedúci pracoviska

  
**prof. Dr. Ing. Miloš Oravec**  
garant študijného programu

# SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE  
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Juraj Kubričan
Bakalárska práca:	Využitie grafovej databázy v praxi
Vedúci záverečnej práce:	Ing. Maroš Čavojský
Miesto a rok predloženia práce:	Bratislava 2017

Bakalárska práca sa zaoberá oboznámením sa s rôznymi predstaviteľmi grafových databáz a naimplementovaním aplikácie ktorá bude využívať jedného predstaviteľa grafovej databázy. V prvej časti sa nachádza teoretický úvod do problematiky grafových databáz, ich výhody a nevýhody oproti relačným databázam z hľadiska štruktúry a výkonu. V druhej časti sa nachádza špecifikácia a návrh našej aplikácie s využitím UML diagramov. V časti implementácia sa nachádza prehľad ostatných použitých technológií, a popísaný proces implementácie našej aplikácie

Kľúčové slová: Využitie grafovej databázy v praxi

# ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Juraj Kubričan
Bachelor Thesis:	Graph database use in a real world application
Supervisor:	Ing. Maroš Čavojský
Place and year of submission:	Bratislava 2017

abstractEN

Keywords: Graph database use in a real world application

# Obsah

<b>Úvod</b>	<b>1</b>
<b>1 Analýza problému</b>	<b>2</b>
1.1 Relačné . . . . .	2
1.2 Grafový model . . . . .	2
1.3 Výkon grafovej databázy . . . . .	3
<b>2 Špecifikácia</b>	<b>4</b>
<b>3 Vývoj</b>	<b>6</b>
3.1 Návrh . . . . .	6
3.1.1 Usecase diagramy . . . . .	6
3.1.2 Class diagram . . . . .	6
3.1.3 Role hráčov . . . . .	6
3.2 Výber grafovej databázy . . . . .	6
<b>4 Implementácia</b>	<b>11</b>
4.1 Použité technológie . . . . .	11
4.1.1 Laravel framework . . . . .	11
4.1.2 NeoEloquent OGM . . . . .	11
4.1.3 Mapbox.js . . . . .	11
4.1.4 Handlebars.js . . . . .	11
4.1.5 Rome2Rio Api . . . . .	11
4.2 Inštalácia a konfigurácia Laravel Framework-u . . . . .	12
4.3 Inštalácia a konfigurácia Databázy neo4j . . . . .	12
4.4 OGM . . . . .	12
4.5 Autentifikácia . . . . .	15
4.6 API . . . . .	16
4.7 Rome2rio API . . . . .	16
4.8 GeoJson . . . . .	17
4.9 Mapa . . . . .	17
4.10 Vykresľovanie tabuliek . . . . .	17
<b>Záver</b>	<b>20</b>
<b>Zoznam použitej literatúry</b>	<b>I</b>



**Zoznam obrázkov a tabuliek**

Obrázok 1	Usecase - Registrácia, autentifikácia a nastavenia . . . . .	7
Obrázok 2	Usecase - dashboard . . . . .	8
Obrázok 3	Usecase - tsp . . . . .	8
Obrázok 4	Class diagram . . . . .	9



## **Zoznam skratiek a značiek**

TSP - The Travelling Salesman Problem

UML - Unified Modeling Language verzie 2.5

GDBMS RDBMS GPLv3 ORM - Object-Relational Mapping OGM - Object-Graph Mapping URL - SSH - Secure Shell WebScket CDN - UX - AJAX - RESTful -

## Zoznam algoritmov

1	Ukážka algoritmu . . . . .	13
2	Ukážka triedy neoEoquent . . . . .	14
3	Ukážka autentifikovateľnej triedy . . . . .	15
4	Zobrazenie odporúčaných miest . . . . .	18
5	Implementácia dynamickej tabuľky . . . . .	19

# Úvod

Pri návrhu aplikácie treba myslieť na štruktúru dát a podľa toho vybrať správnu databázu/DBMS. Správny výber DBMS vie zabezpečiť rádovo nižšie prístupové časy a tým aj väčšiu scalability. Pri aplikáciách kde sú dáta štruktúrované do uzlov a prepojení, je vhodné zvážiť použitie grafovej databázy. My sme navrhli a naimplementovali aplikáciu cestovného plánovača, ktorý potrebuje uchovávať dáta o destináciách a cestách medzi nimi. Preto je pre túto aplikáciu vhodné využiť grafový DBMS

# 1 Analýza problému

## 1.1 Relačné

Relačná databáza je databáza, v ktorej sú údaje uložené podľa relačného databázového modelu podľa E. F. Codd z roku 1970. Podľa Relačného modelu sú dáta uložené v tabuľkách. Jeden riadok tabuľky je jeden záznam. Stĺpec tabuľky reprezentuje jeden atribút objektu.

Väzby (vzťahy) medzi tabuľkami sú riešené pomocou unikátnych identifikátorov tzv. kľúčov. Jedna tabuľka obsahuje kľúč, čo je atribút ktorý unikátne identifikuje každý záznam a druhá tabuľka obsahuje tzv. cudzí kľúč, atribút ktorý odkazuje na záznam v prvej tabuľke.

Nevýhoda tohto prístupu sa však ukazuje v škálovateľnosti. Pri vyhľadávaní každé toto prepojenie pridáva výpočtovú komplexitu, keďže v každej ďalšej prepojenej tabuľke treba vyhľadať záznam s požadovaným kľúčom ( $O(\log(n))$ ). Všetky používané relačné databázové systémy riešia tento problém škálovateľnosti použitím indexov a rôznymi inými optimalizáciami, no pokiaľ sú naše dáta štruktúrované s veľa prepojeniami systém sa spomalí.

## 1.2 Grafový model

Modelovanie grafovej databázy prirodzene zapadá do spôsobu akým bežne abstrahujeme problémy pri vývoji softvéru. Pri návrhu softvéru objekty opisujeme obdĺžnikmi alebo kruhmi a súvislosti medzi nimi šípkami, či čiarami. Moderné grafové databázy sú viac ako akákoľvek iná databázová technológia vhodná na takúto reprezentáciu, lebo to, čo namodelujeme na papier vieme priamo neimplementovať v našej grafovej databáze.

Grafové databázy využívajú model ktorý pozostáva z vrcholov, hrán, atribútov a značiek. Vrcholy obsahujú atribúty, a sú označené jednou alebo viacerými značkami. Tieto značky zoskupujú vrcholy, ktoré zastávajú rovnakú rolu v rámci aplikácie.

Hrany v grafových databázach spájajú vrcholy a budujú štruktúru grafu. Hrana grafu má vždy smer, názov, východzí vrchol, cieľový vrchol a cieľový vrchol. Fakt, že hrany musia mať smer a názov pridáva to sémantickú prehľadnosť do grafu, ak zvolíme správne názov vieme rýchlo identifikovať štruktúru grafu a identifikovať význam vzťahov. Hrany môžu rovnako ako vrcholy obsahovať aj atribúty. Atribúty v hranách môžu byť praktické na pridanie kvalitatívnych dát (napr. váha, vzdialenosť) ku vzťahom, tieto dáta sa potom môžu použiť pri prehľadávaní grafu.

## 1.3 Výkon grafovej databázy

Ako sme už spomínali v časti 1.1 prehľadanie každého ďalšieho vstahu v relačnej databáze má teoretickú výpočtovú zložitosť ( $O(\log(n))$ ). Na druhej strane natívne grafové databázy používajú bezindexovú príslušnosť [?]. Toto v praxi znamená, že pri prehľadávaní vzťahov v klesá výpočtová zložitosť na  $O(1)$ . Táto rýchlosť je dosiahnutá tak, že všetky hrany sú uložené s priamimi ukazovateľmi na vrcholy, ktorých vzťah reprezentujú. Tak isto vo vrcholoch sú uložené priame ukazovatele na všetky hrany vychádzajúce z a mieriace do dotyčného vycholu. Takáto štruktúra poskytuje už spomínanú výpočtovú zložitosť  $O(1)$  v oboch smeroch hrany, takže nielen v smere z východzieho bodu do cieľového ale aj opačným smerom. Pri relačnej databáze by toto muselo byť riešené reverzným vyhľadávaním v cudzích kľúčoch.

## 2 Špecifikácia

### 1. Funkcionálne požiadavky

- (a) Aplikácia bude umožňovať registráciu a prihlásenie používateľa
- (b) Pri registrácii sa budú vyžadovať prihlasovacie údaje: e-mail, heslo. Okrem toho sa bude vyžadovať zdieľanie mena a domáceho miesta. Toto domáce miesto bude možné vyhľadať v online databáze.
- (c) Po prihlásení používateľa sa mu zobrazí obrazovka s mapou, zoznamom obľúbených destinácií, ktoré chce navštíviť a zoznam odporúčaných destinácií
- (d) Na mape bude vyobrazené používateľovo domáce miesto a všetky destinácie, ktoré má v zozname obľúbených destinácií. Po kliknutí na destináciu sa používateľovi otvorí príslušný riadok v zozname obľúbených.
- (e) V zozname obľúbených budú všetky miesta, ktoré si používateľ pridal. Zoznam bude vo forme tabuľky, riadok bude obsahovať meno destinácie a lokalitu, v ktorej sa destinácia nachádza. V riadku tiež bude tlačidlo na vymazanie destinácie z obľúbených a tlačidlo na zobrazenie detailov.
- (f) V detailoch obľúbeného miesta bude zoznam ostatných ľudí, ktorí dané miesto majú v obľúbených a výpis možných trás z domáceho miesta používateľa do destinácie.
- (g) V zozname odporúčaných destinácií budú destinácie, ktoré majú v obľúbených používatelia, ktorí majú v obľúbených rovnaké miesta ako miesta, ktoré má v obľúbených prihlásený používateľ. Vynechané budú miesta, ktoré už prihlásený používateľ má obľúbených. Každá položka z odporúčaných sa bude dať jednoducho pridať do obľúbených prihláseného používateľa.
- (h) V aplikácii bude obrazovka s nastaveniami, na ktorej si bude človek môcť zmeniť heslo, domáce miesto.
- (i) V aplikácii bude obrazovka, kde si bude používateľ vybrať niekoľko zo svojich obľúbených miest a nechať si vyrátať najkratšiu trasu z domáceho miesta cez všetky zvolené miesta a potom späť. (TSP)

### 2. Nefunkcionálne požiadavky

- (a) Systém bude zrealizovaný na webovej platforme.
- (b) Aplikácia bude využívať natívnu grafickú databázu.

- (c) Aplikácia bude byť kompatibilná s webovými prehliadačmi Google Chrome Mozilla Firefox, Microsoft Edge, Microsoft Internet Explorer.
- (d) Užívateľské rozhranie systém musí byť plne použiteľné aj na mobilných telefónoch s OS Android a IOS.
- (e) Aplikácia bude implementovaný s použitím jazyka PHP a PHP frameworku.
- (f) Systém bude nasadený na virtuálnom serveri s operačným systémom Ubuntu 16.04.2 LTS poskytnutom Ústavom informatiky a matematiky FEI STU.

## 3 Vývoj

### 3.1 Návrh

V časti návrhu projektu popíšeme prípady použitia,

#### 3.1.1 Usecase diagramy

#### 3.1.2 Class diagram

#### 3.1.3 Role hráčov

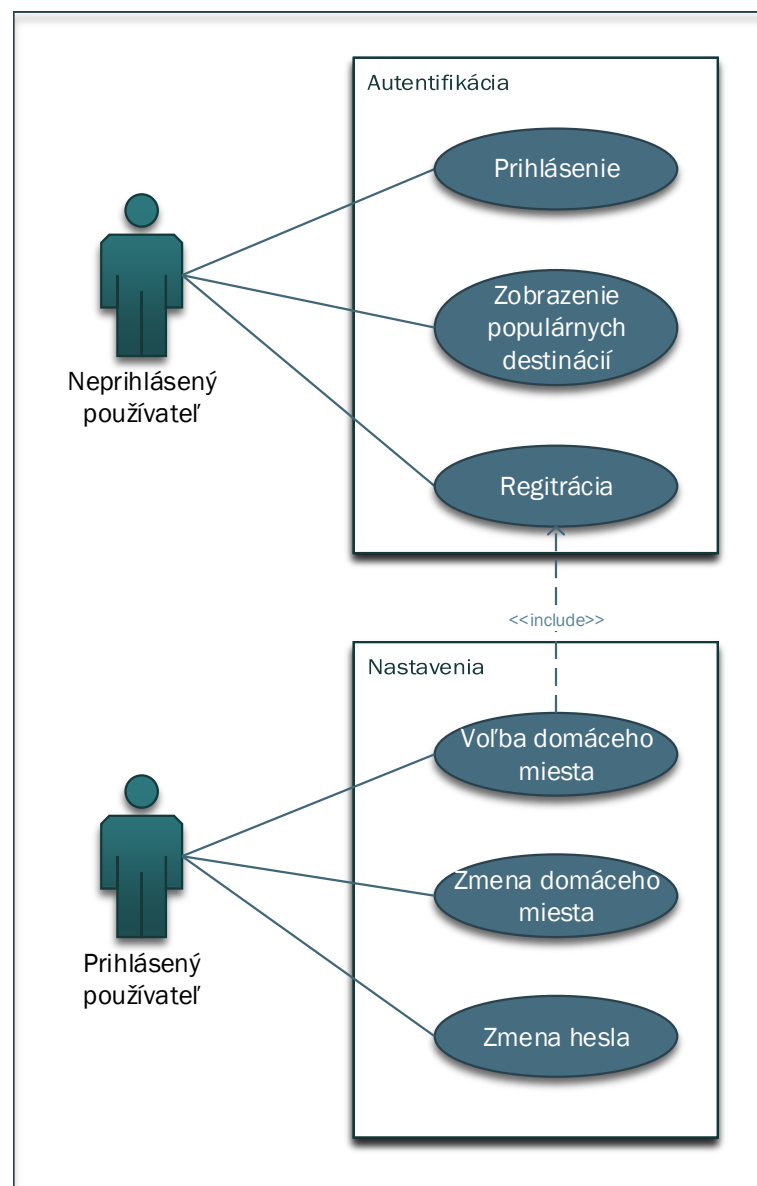
1. Neprihlásený používateľ bude mať prístup na úvodnú stránku. Na úvodnej stránke sa bude môcť buď zaregistrovať, prihlásiť ak už má vytvorený účet, alebo si bude môcť pozrieť mapu a zoznam s najpopulárnejšími destináciami používateľov aplikácie.
2. Prihlásený používateľ má prístup do štyroch subsystémov
  - (a) Dashboard je domáca obrazovka používateľa bude môcť na nej vyhľadávať, pridávať a odstraňovať miesta medzi svoje obľúbené. Ďalej si bude môcť pozrieť detaily destinácie ako rôzne trasy ktoré vedú z jeho domáceho miesta do destinácie a zoznam ostatných používateľov, ktorí majú rovnaké miest v obľúbených. Kliknutím na používateľa bude môcť prejsť na obrazovku používateľa.
  - (b) Na obrazovke nastavení si bude môcť používateľ nastaviť heslo a zmeniť svoje domáce miesto.
  - (c) Na obrazovke TSP si bude používateľ môcť naplánovať trasu medzi niektorými zo svojich obľúbených miest. Bude si môcť pridávať miesta do trasy a odoberať ich. Ďalej si bude môcť prezrieť detaily aktuálnej trasy s cenami a mapu s vyobrazenou trasou.
  - (d) Na obrazovke používateľa si bude prihlásený používateľ prezrieť obľúbené destinácie konkrétneho používateľa na mape a v zozname.

### 3.2 Výber grafovej databázy

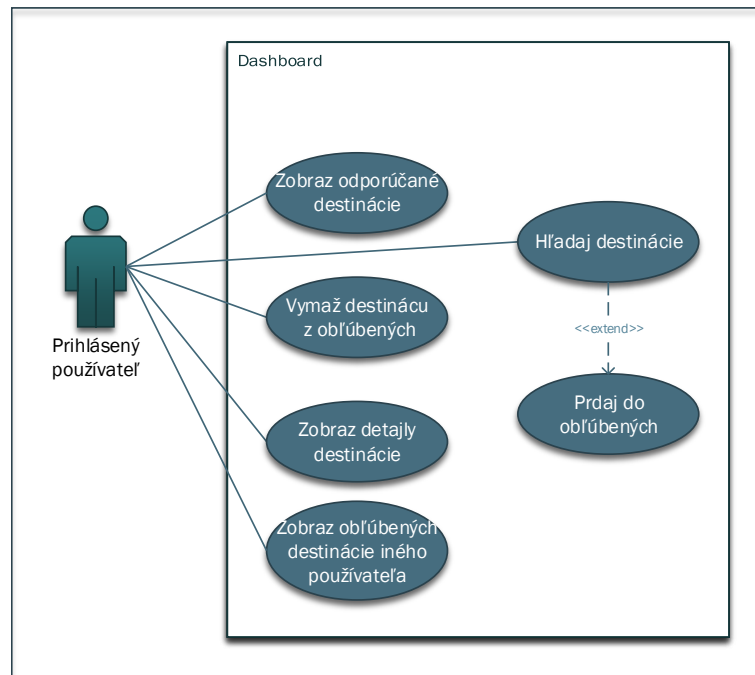
Vybrali sme si troch najpopulárnejších predstaviteľov grafových databáz podľa [?] rebríčka na DB-Engines.com. V nasledujúcej v skratke časti priblížime históriu každého GDBMS ich výhody, nevýhody pre naše použitie a proces výberu použitej databázy.

1. NEO4J Prvá verzia Neo4J bola vydaná v roku 2007 od vtedy sa stala dlhodobo najpoužívanjšou grafovou databázou. Je vyvíjaná Neo Technology, Inc. Neo4j je

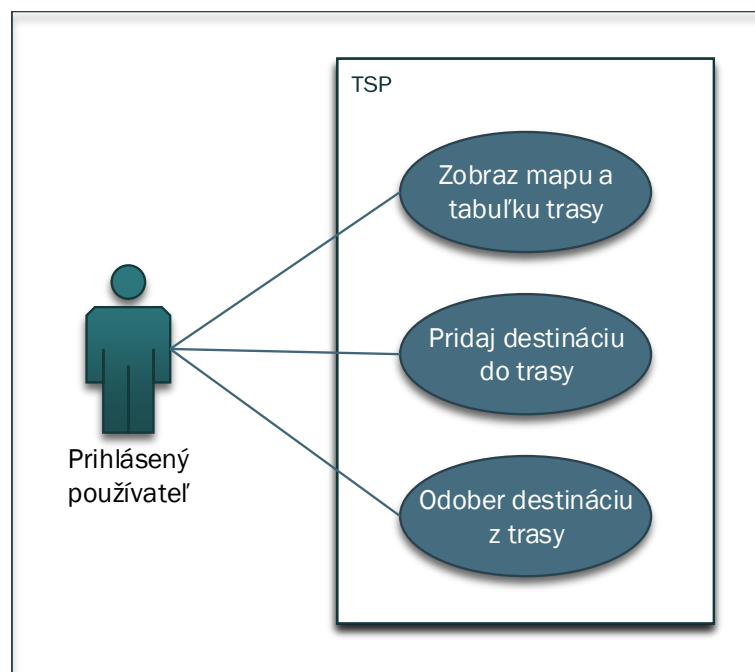




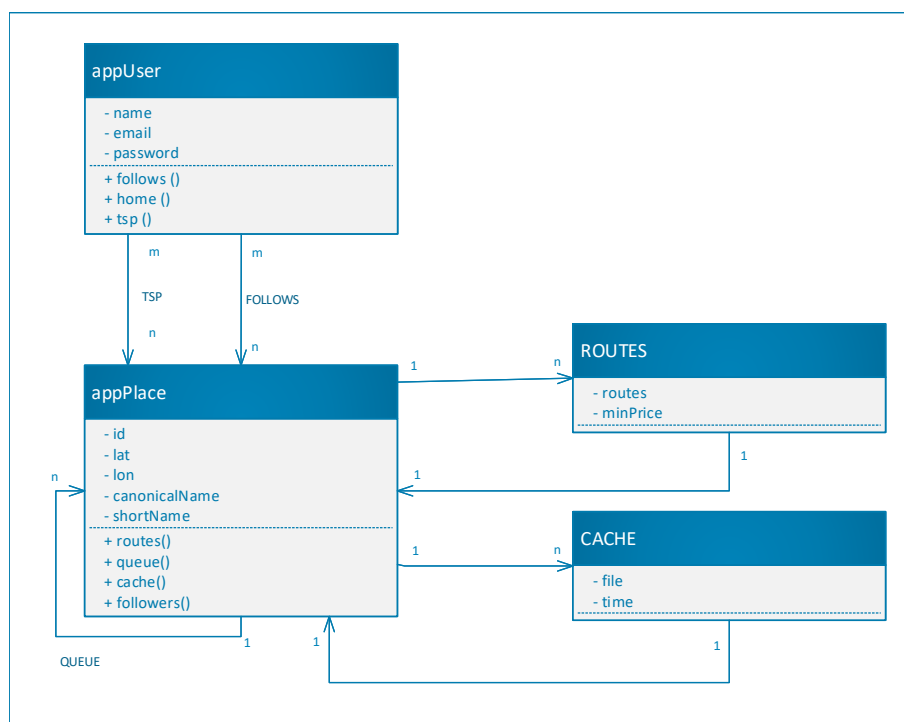
Obrázok 1: Usecase - Registrácia, autentifikácia a nastavenia



Obrázok 2: Usecase - dashboard



Obrázok 3: Usecase - tsp



Obrázok 4: Class diagram

ponúkaná v dvoch variantoch: Neo4j Community je open source (GPLv3) grafová databáza obsahujúca všetky základné funkcie (Ďalej budeme spomínať len túto verziu). A Neo4j Enterprise edícia, ktorá má rozšírené funkcie ako shardovanie cache pamäte, rozšírené monitorovanie a zálohovanie za behu.

Medzi hlavné výhody neo4j patrí to, že je to dlhodobo najrozšírenejšia grafová databáza, má dobrú dokumentáciu, podporuje mapovanie na objekty. Nevýhodou je, že podporuje iba grafový model ukladania údajov.

2. OrientDB OrientDB je vyvíjané od roku 2010 firmou OrientDB LTD. Databáza OrientDB rýchlo nabrala na popularite a v roku 2015 sa dostala na druhé miesto v rebríčku DB-engines. OrientDB sa rovnako ako Neo4j distribuuje v dvoch edíciách: Community - open source (Apache Licence 2.0) so základnými funkciami a Enterprise edíciou s podporou migrácie a synchronizácie na Neo4J a pridanými analytickými nástrojmi.

Výhodou OrientDB je podpora okrem grafového modelu ukladania dát aj dokumentový a key/value model ukladania údajov. Nevýhodou je horšia podpora pre nami

vybraný framework Laravel

3. Titan Titan bol od 2012 vyvíjaný skupinou ThinkAurelius, no v roku 2017 bol odkúpený firmou Datstax a projekt Titan bol zastavený. Projekt je ďalej udržiavaný ako open source verzia pod menom JanusGraph. Titan je projekt určený na veľké distribuované enterprise riešenia, je nasadzovaný na cloudové platformy ako napr. Apache Hadoop, Apache Spark, ... podporuje rôzne distribuované úložné priestory ako napr. Apache HBase, Oracle BerkeleyDB, ... Ďalej podporuje rôzne vyhľadávacie enginy ako napr. Elasticsearch, Solr, ...

Toto riešenie je pre naše použitie nevhodné, lebo na správne fungovanie vyžaduje distribuovaný systém.

## 4 Implementácia

### 4.1 Použité technológie

#### 4.1.1 Laravel framework

Laravel Framework je komplexný voľne šíriteľný framework. Tento framework je od roku 2015 najpopulárnejší PHP framework. Medzi jeho hlavné výhody patrí použitie relatívne novej verzie PHP 5.4, ktorá obsahuje technológie, ktoré v minulosti v PHP chýbali ako menové priestory a anonymné funkcie. Ďalej obsahuje veľmi silný nástroj Eloquent ORM pre objektovo relačné mapovanie databázy, Blade šablónovací nástroj na rýchlu tvorbu dynamického obsahu.

Komplexita Frameworku Laravel je však aj jeho hlavnou nevýhodou, nie je vhodný na menšie projekty. V rýchlosti patrí medzi priemer PHP frameworkov.

#### 4.1.2 NeoEloquent OGM

NeoEloquent OGM je voľne šíriteľná knižnica, ktorá umožňuje využívať grafovú databázu neo4j spolu s existujúcim dátovým modelom vo frameworku Laravel. Štruktúra NeoEloquent je modelovaná podľa natívneho Eloquent Modelu Laravel. Knižnica je vydávaná od roku 2014 pod licenciou MIT spoločnosťou Vinelab

#### 4.1.3 Mapbox.js

Mapbox.js je komerčná knižnica na vytváranie projektov s interaktívnymi mapami. Je založená na voľne šíriteľnej knižnici Leaflet, rozširuje túto knižnicu o funkcie ako automatické zoskupovanie bodov do skupín a poskytuje bohatú a prehľadnú dokumentáciu. My používame verziu zdarma, ktorá je obmedzená na 50000 zobrazení maky na mesiac. Mapbox.js podporuje štandardný formát dát GeoJSON, tento formát umožňuje ukladať dáta o polohe, type bodu, rôznych atribútov upresňujúcich vizuál zobrazovaného bodu. Tento formát ďalej umožňuje ukladať geometrické útvary a krivky.

#### 4.1.4 Handlebars.js

Handlebars.js je jednoduchá silná open source knižnica na prácu z šablónami, vyvíjaná je Yehudom Katzom a komunitou na GitHubu od roku 2010 a vydávaná pod licenciou MIT.

#### 4.1.5 Rome2Rio Api

Rome2Rio je portál ktorý zbiera údaje o cenách dopravy po celom svete a umožňuje vyhľadať cenu cesty medzi dvomi ľubovoľnými destináciami. Rome2Rio taktiež poskytuje niekoľko otvorených a platených API. My využívame dve z nich: Search API a Autocomplete API. Autocomplete API slúži na vyhľadávanie destinácií z databázy rome2rio a nieje

obmedzené na počet volaní. Search API slúži na vyhľadávanie trás medzi jednotlivými destináciami, je potrebné sa identifikovať API kľúčom a je obmedzené na 100 000 volaní za mesiac.

## 4.2 Inštalácia a konfigurácia Laravel Framework-u

Na inštaláciu frameworku Laravel sme použili nástroj pre správu PHP balíkov *Composer*. Pomocou tohto nástroja sme nainštalovali balík *laravel/installer* [?], následne sme použitím príkazu *laravel new projekt* vytvorili priečinok so základnou inštaláciou frameworku.

Jediné nastavenie ktoré bolo po tom potrebné bolo už len nastavenie názvu aplikácie, a databázy, o tom ďalej v sekcii ??.

## 4.3 Inštalácia a konfigurácia Databázy neo4j

[?] Aby sme mohli nainštalovať databázu Neo4j musíme si najprv do systému pridať repozitár Neotechnology, potom je nám k dispozícii na inštaláciu balík neo4j. Po inštalácii je nám ihneď dostupné administratívne rozhranie databázy na adrese: *localhost:7474*. Pri prvom prihlasení sme vyzvaní na zmenu hesla.

Keďže základná inštalácia frameworku Laravel neobsahuje ovládač pre databázu neo4j museli sme použiť ovládač integrovaný v balíku NeoEloquent, to sa registráciou poskytovateľa služby. Po zaregistrovaní služby NeoEloquentServiceProvider sa automaticky zaregistruje ovládač pre databázu a pridajú sa nové možnosti pre konfiguráciu databázy. Následne stačí vykonať štandardnú konfiguráciu mena hostiteľa, port a prístupových údajov.

Na získanie vzdialeného prístupu k administratívnejmu rozhraniu databázy bez otvorenia portu 7474 verejnosti využívame SSH tunel. Administratívne rozhranie používa okrem portu 7474 ešte port 7687 lebo na komunikáciu s databázou využíva technológiu WebSocket.

## 4.4 OGM

Keďže framework Laravel natívne obsahuje len ovládače pre relačné databázové ovládače a nástroj na objektovo relačné mapovanie Eloquent.

Použili sme open source balík NeoEloquent, ktorý obsahuje ovládač pre databázu Neo4J a zároveň rozširuje dátový model o prvky grafovej databázy. NeoEloquent umožňuje manipuláciu s vrcholmi aj hranami v Neo4J. Manipulácia s vrcholmi je rovnaká ako s entitami v relačnej databáze, NeoEloquent umožňuje vytvárať perzistentné objekty, upravovať ich a vyhľadávať v nich. Pri práci s hranami je mierne odlišná. Najprv treba zadať ktorý objekt môže mať aké vzťahy, tieto vzťahy treba unikátne identifikovať ich typom

---

**Algoritmus 1** Ukážka algoritmu

---

```
1 $app->register('Vinelab\NeoEloquent\NeoEloquentServiceProvider');
2 ...
3 'default' => env('DB_CONNECTION', 'neo4j'),
4
5 'connections' => [
6 'neo4j' => [
7 'driver' => 'neo4j',
8 'host' => env('DB_HOST', 'neo4j'),
9 'port' => env('DB_PORT', 'neo4j'),
10 'username' => env('DB_USERNAME', 'neo4j'),
11 'password' => env('DB_PASSWORD', 'neo4j'),
12 ],
13 ],
```

---

a kardinalitou. Tento vzťah vraciame ako návratovú hodnotu funkcie daného objektu. Vrátený objekt sa správa podobne ako štandardná entitná trieda Eloquent.

V nasledujúcom príklade 3 vidíme implementáciu dvoch rôznych tried. Trieda používateľa dedí od triedy NeoEloquent a teda sa stáva naviazanou na vrchol v našej databáze. Názov tejto entity v databáze je spojením menového priestoru v ktorom bol vytvorený a názvu triedy, takže v našom prípade AppUser. Trieda obsahuje verejné funkcie, ktoré vracajú objekty hrán. Objekty hrán dostávame volaním zdedených funkcií hasMany, hasOne a belongToMany. Ako prvý argument funkcie berú názov triedy ktorou vzťah chceme vrátiť, ako druhý argument berie typ hrany, pomocou tohto typu je identifikovaný typ hrany v databáze.

Trieda NeoEloquent funguje vo väčšine prípadov presne ako Eloquent no v jednom prípade sme mali problém z CSRF tokenmi. CSRF token je bezpečnostný prvok, ktorý ochraňuje webovú stránku pred útokom falšovania požiadaviek z inej adresy. Laravel má tento bezpečnostný prvok vstavaný v sebe, je to 40 znakový retazec, ktorý sa generuje každému používateľovi pri zobrazení formulára, tento istý retazec sa zároveň uloží do databázy a keď Laravel prijme formulár overí či sa jeho token nachádza v databáze. Táto funkcionálnosť však po prejení na grafovú databázu nefungovala. Pri každom odoslaní formulára vyhlasovalo nezhodu CSRF tokenu a v databáze sa neobjavila entita ktorá by tieto tokeny mohla obsahovať. Tento problém sme zatiaľ obišli deaktiváciou tohto bezpečnostného prvku.

---

**Algoritmus 2** Ukážka triedy neoEloquent

---

```
1 namespace App;
2
3 class User extends \NeoEloquent implements Authenticatable {
4
5     // Jeden používateľ môže mať v obľúbených viac miest
6     public function follows() {
7         return $this->hasMany('App\Place', 'FOLLOWS');
8     }
9
10    // Jeden používateľ má jedno miesto ako domáce
11    public function home() {
12        return $this->hasOne('App\Place', 'HOME');
13    }
14    ...
15 }
16 ...
17 class Place extends \NeoEloquent {
18     // Inverzný vzťah – jedno miesto má v obľúbených viac používateľov
19     public function followers(){
20         return $this->belongsToMany('App\User', 'FOLLOWS');
21     }
22     ...
23 }
```

---



## 4.5 Autentifikácia

Jednou zo silných stránok Frameworku Laravel je práve autentifikácia. Pre vytvorenie základnej funkcionality registrácie, prihlasovania a obnovenia zabudnutého hesla stačí použiť Artisan - konzolu frameworku príkaz (*php artisan make:auth*) , ktorá vytvorí URL cesty, obrazovky, triedu používateľa a triedy obsluhujúce túto funkcionality. My sme potrebovali použiť vlastnú triedu používateľa, ktorá dedí od nášho balíka NeoEloquent, na implementáciu autentifikácie stačilo neimplementovať rozhranie Authenticatable, pridať do triedy používateľa pole skrytých a verejných atribútov, a použiť charakteristiku 'AuthenticableTrait' a autentifikácia fungovala rovnako ako s relačnou databázou vďaka tomu, že NeoEloquent pokrýva všetky funkcie natívneho Eloquent ORM.

---

### Algoritmus 3 Ukážka autentifikovateľnej triedy

---

```
1
2 namespace App;
3
4 use Illuminate\Contracts\Auth\Authenticatable;
5 use Illuminate\Auth\Authenticatable as AuthenticableTrait;
6
7 class User extends \NeoEloquent implements Authenticatable {
8
9     use AuthenticableTrait;
10
11     // pole verejných atribútov
12     protected $fillable = [
13         'name', 'email', 'password', 'tspCache'
14     ];
15
16     // pole skrytých atribútov
17     protected $hidden = [
18         'password', 'remember_token',
19     ];
20     ...
21 }
```

---

## 4.6 API

Keďže sme sa rozhodli využiť javascriptové zobrazovanie dynamického obsahu museli sme vytvoriť interné API, aby sme mohli javascriptu poskytnúť údaje. Na tento účel sme vytvorili niekoľko ciest pomocou routovacieho nástroja frameworku. Router Laravel-u je relatívne jednoduchý, ale silný nástroj na vytváranie RESTful API. Na naše potreby sme potrebovali vytvoriť tri REST cesty: */place*, */placeapi* a */tsp*. Prvá je na pridávanie destinácií, druhá na pridávanie do, odoberanie z obľúbených a zobrazovanie obľúbených na mape, tretia na pridávanie odoberanie a zobrazovanie destinácií na zozname TSP.

## 4.7 Rome2rio API

Všetky údaje o destináciách a trasách berieme z API Rome2Rio. Pomocou Autocomplete API umožňujeme používateľovi pridávať destinácie. Používateľ zadáva písmená do autocomplete textového poľa na stránke toto pole posiela dotazy na Autocomplete api a ono vracia pole objektov s miestami. Tieto objekty obsahujú informácie o type destinácie (obec, mesto, región, štát, letisko, ), geografickú polohu, názov v dlhom a krátkom tvare a kanonický názov. Používateľ si potom vyberie jednu z destinácií a príslušný objekt sa zašle na náš server. Na unikátnu identifikáciu objektu používame kanonický názov, ktoré je podľa dokumentácie unikátnym identifikátorom miesta.

Ak miesto miesto ešte nemáme v databáze, pridáme tento objekt do databázy. Toto riešenie nie je úplne ideálne z bezpečnostného hľadiska, lebo umožňuje zaslanie falošného miesta do našej databázy. Ak by útočník vyrobil objekt s reálnym kanonickým názvom no falošnými údajmi napr o zemepisnej šírke, dĺžke toto miesto by sa potom nespávne zobrazovalo všetkým používateľom. Na vyriešenie tohto problému by stačilo urobiť ešte jeden dotaz z nášho servera na autocomplete api ktorým by sme si len vypýtali údaje k miestu za pomoci kanonického názvu.

Ďalšie údaje berieme z Rome2Rio Search API. Sú to údaje o možných trasách a ich cenách. Toto API je obmedzené na počet volaní preto sme na volanie toho API implementovali pamäť cache. Vždy keď voláme search api voláme ho na dve miesta, ktoré už máme uložené v našej databáze ako vrcholy. Tento fakt sme využili a vytvorili sme ďalší typ hrany - CACHE. Keďže pri mestách ktoré sú dopravné uzly vystúpala veľkosť odpovede API až na rádovo 500kb a tento typ dopytu sa nerobí veľmi často rozhodli sme sa odpoveď servera neukladať priamo do databázy ale v nezmenenej podobe na disk a do databázy uložiť len veľkosť cache súboru a referenciu na súbor na disku. Keďže v momentálnej podobe nevyužívame celú odpoveď tohto api mohli by sme optimalizovať využitie miesta na disku tým, že by sme najprv údaje spracovali a uložili len tie, ktoré využívame.

## 4.8 GeoJson

Ako zdrojový formát dát pre všetky mapy v našej aplikácii používame štandardný formát GeoJson [?].

## 4.9 Mapa

Na vizualizáciu destinácií a trás sme na rôznych miestach a aplikácií použili JavaScriptovú knižnicu Mapbox.js. Na inicializáciu mapy sme naimplementovali funkciu 4.

Keďže Mapbox.js rozširuje knižnicu Leaflet, všetky funkcie knižnice sa volajú z globálneho objektu *L*. Táto funkcia sa zavolá po načítaní stránky. Keďže sa mapy Mapbox.js sa sťahujú z CDN Mapbox musíme najprv aplikáciu identifikovať API kľúčom, ktorý sme si vygenerovali po registrácii. Následne inicializuje samotná mapa volaním funkcie *L.mapbox.map*. Táto funkcia berie ako prvý parameter id HTML elementu, do ktorého sa má mapa zobrazíť, ako druhý parameter berie textový identifikátor typu mapy, ktorý chceme zobrazíť. Po inicializácii sa vytvorí vrstva pre mapu ktorá bude obsahovať markery destinácií. Dáta ktoré obsahujú geografickú polohu destinácií sa vyžadujú vo formáte GeoJson z API našej aplikácie. Keďže pri malom priblížení mapy by sa nedali zreteľne rozlíšiť destinácie ktoré sú blízko pri sebe, po načítaní dát vytvoríme vrstvu clusterov pomocou funkcie *L.MarkerClusterGroup*. Táto vrstva spojí blízke body do clusterov, následne skryje značky týchto bodov a nahradí ich značkou clusteru. Na pridanie bodov do clusterov sa iteruje cez všetky body, ktoré sú po načítaní v mape a každý sa pridá do vrstvy clusterov. Následne sa vrstva clusterov pridá do objektu mapy

## 4.10 Vykresľovanie tabuliek

Aby sme zlepšili UX stránky rozhodli sme sa na vykresľovanie dynamického obsahu využiť namiesto HTML obsahu renderovaného na serveri javascriptovú knižnicu na prácu so šablónami a, dynamický obsah načítavame z API našej aplikácie vo formáte JSON. Na vykresľovanie dynamického obsahu sme zvolili knižnicu Handlebars.js. Na príklade 5 môžeme ukázať príklad našej implementácie dynamickej tabuľky pre hlavnú tabuľku zobrazujúcu destinácie a ich detaily.

Najprv je zadeklarovaná premenná v *template* ktorá bude neskôr obsahovať funkciu renderujúcu šablónu. Potom je zadeklarovaná funkcia *refreshPage*, ktorá bude volaná vždy, keď nejaká funkcia spustí event s menom *appRefresh*. Táto funkcia využije AJAX API knižnice jQuery a stiahne potrebné dáta, potom využije funkciu *template*, ktorá do argumentu berie dáta vo forme poľa objektov a vracia vygenerované HTML ktoré sa následne vkladá na stránku.

---

#### Algoritmus 4 Zobrazenie odporúčaných miest

---

```
1  var map;  
2  function initMap() {  
3    // API kľúč  
4    L.mapbox.accessToken = 'key';  
5  
6    // Inicializácia mapy s voľbou typu mapy  
7    map = L.mapbox.map('main_map', 'mapbox.k8xv42t9');  
8  
9    // Pridávanie vrstiev na mapu  
10   L.mapbox.featureLayer()  
11   .loadURL("/placeapi?type=geojson&filter=suggested")  
12   .on('ready', function(e) {  
13     //Vytváranie vrstvy clusterov  
14     var clusterGroup = new L.MarkerClusterGroup();  
15     e.target.eachLayer(function(layer) {  
16       clusterGroup.addLayer(layer);  
17       //Pridanie funkcionality kliknutia na mesto v zozname  
18       $(document).on('click', '.zoom-map[data-id="'+ layer.feature.properties.ti  
19       map.setView(layer.getLatLng(),8);  
20     })  
21   });  
22   // Úvodné pridanie vrstvy a centrovanie  
23   map.addLayer(clusterGroup);  
24   map.fitBounds(clusterGroup.getBounds());  
25   });  
26 }  
27 }
```

---

V druhej časti kódu sa najprv zavolá funkcia *Handlebars.compile* do argumentu zoberie šablónu ktorá je uložená v elemente s id *places-template*. Ako návratovú hodnotu vráti funkciu, ktorú uložíme pod menom *template*. Následne sa na event *appRefresh* naviaže volanie funkcie *refreshPage* a prvý krát sa spustí tento event.

Naša šablóna *places-template* obsahuje aj aktívne linky a v nasledujúcom bloku je príklad implementácie vymazania miesta z obľúbených. Na generálny event *click* je naviazaný filtrovaný event, ktorý spustí AJAX dotaz na vymazanie miesta z obľúbených ak element ktorý event spustil obsahuje triedu *delete* a atribút *data-id*.

---

**Algoritmus 5** Implementácia dynamickej tabuľky

---

```
1  var template;
2  function refreshPage () {
3    $.get('/placeapi?type=template', function (data) {
4      data = JSON.parse(data);
5      home = data.places[0];
6      $('#places_body').html(template(data));
7    });
8  }
9
10 $(document).ready(function() {
11
12   template = Handlebars.compile($("#places-template").html());
13   }).on('appRefresh', refreshPage).trigger('appRefresh');
14
15   $(document).on('click', '.delete[data-id]', function (e) {
16     e.preventDefault();
17     $.ajax({ url: '/placeapi/' + $(this).data('id'), type: 'POST',
18       success: function () {
19       $(document).trigger('appRefresh');
20     }
21   });
22 });
```

---

# Záver

Cieľom práce bolo oboznámiť sa s rôznymi predstaviteľmi gravoých databáz, vybrať jedného, navrhnúť a naimplementovať využitie tejto databázy na reálnom príklade.

# Prílohy