

**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

Úvod	1
1 Analýza problému	2
1.1 Relačné databázy	2
1.2 Grafový model	2
1.3 Výkon grafovej databázy	3
2 Špecifikácia	4
3 Návrh	6
3.0.1 Prípady použitia	6
3.0.2 Štruktúra databázy	6
3.1 Výber grafovej databázy	8
3.1.1 NEO4J	8
3.1.2 OrientDB	9
3.1.3 Titan	9
4 Implementácia	10
4.1 Použité technológie	10
4.1.1 Laravel framework	10
4.1.2 NeoEloquent OGM	10
4.1.3 Mapbox.js	10
4.1.4 Handlebars.js	10
4.1.5 Rome2Rio Api	10
4.2 Inštalácia a konfigurácia Laravel Framework-u	11
4.3 Inštalácia a konfigurácia databázy neo4j	11
4.4 OGM	11
4.5 Autentifikácia	13
4.6 API	14
4.7 Rome2rio API	15
4.8 TSP	15
4.9 Mapa	17
4.10 Vykresľovanie tabuliek	18
Záver	20

Zoznam použitej literatúry

21

Prílohy

I

Zoznam obrázkov a tabuliek

Obrázok 1	Prípad použitia - Registrácia, autentifikácia a nastavenia	7
Obrázok 2	Prípad použitia - Registrácia, autentifikácia a nastavenia	7
Obrázok 3	Prípad použitia - dashboard	7
Obrázok 4	Prípad použitia - tsp	8
Obrázok 5	Class diagram	8

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 - Content delivery network UX -

AJAX - RESTful -

Zoznam algoritmov

1	Ukážka algoritmu	12
2	Ukážka triedy neoEloquent	13
3	Ukážka autentifikovateľnej triedy	14
4	Ukážka riešenia TSP pomocou dopytovacieho jazyka Cypher	17
5	Implementácia dynamickej tabuľky	19

Úvod

V dnešnej dobe, keď

1 Analýza problému

1.1 Relačné databázy

Relačná databáza je databáza, v ktorej sú údaje uložené podľa relačného databázového modelu[?] podľa E. F. Codd a 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 nedeterministickými optimalizáciami, no pokiaľ sú naše dáta štruktúrované s veľkým množstvom vzťahov systém sa spravidla 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 vzťahu v relačnej databáze má teoretickú výpočtovú zložitosť $O(\log(n))$. Na druhej strane natívne grafové databázy používajú bez indexovú príľahlosť [2]. 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 priamymi ukazovateľmi na vrcholy, ktorých vzťah reprezentujú. Tak isto vo vrcholoch sú uložené priame ukazovatele na všetky hrany vychádzajúce z a mieriace do dotyčného vrcholu. 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

Aplikácia ktorú sme sa rozhodli implementovať bude slúžiť na plánovanie a optimalizáciu trasy cestovateľa po svete. Dáta našej aplikácie budú teda pozostávať z používateľov, destinácií a prepojeniami medzi nimi. Štruktúra grafovej databázy je obzvlášť vhodná na návrh a implementáciu prepojení medzi jednotlivými mestami, keďže tento model má presne štruktúru klasického grafu.

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 používateľove 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 spať. (TSP)

2. Nefunkcionálne požiadavky

- (a) Systém bude zrealizovaný na webovej platforme.
- (b) Aplikácia bude využívať natívnu grafovú 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 Návrh

V tejto kapitole si priblížime proces návrhu našej aplikácie, popíšeme prípady použitia ktoré pre našu aplikáciu očakávame. Ďalej priblížime návrh štruktúry databázy, ktorú plánujeme použiť.

3.0.1 Prípady použitia

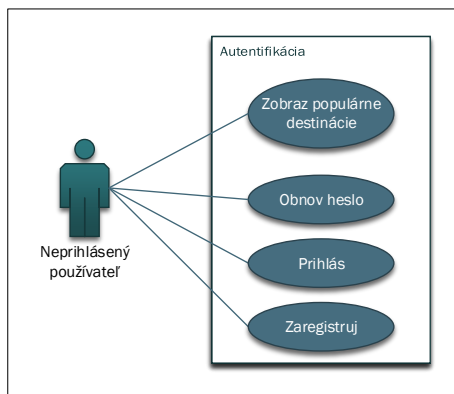
Prípady použitia popisujú interakcie medzi aplikáciou a používateľom, popisujú roly, ktoré rôzni hráči hrajú v týchto interakciách.

1. V prvom prípade použitia 1 si bude môcť neprihlásený používateľ prezerat mapu a zoznam najpopulárnejších destinácií v našej aplikácii, ďalej sa môže prihlásiť zaregistrovať a požiadať o obnovu hesla.
2. Ďalší prípad použitia 2 opisuje nastavenia. Používateľ si bude môcť vyhľadať destináciu a zvoliť si ju ako svoje domáce miesto alebo si zmeniť heslo.
3. Tretí prípad použitia 3 opisuje akcie ktoré bude môcť používateľ vykonávať na hlavnej stránke. Bude môcť vyhľadávať destinácie a pridávať si ich do obľúbených. Destinácie ktoré má v obľúbených bude môcť mazať a zobrazíť detaily trasy, ktorá k destinácii vedie. Ďalej si bude môcť prezerat ostatných používateľov ktorý majú rovnaké miesto v obľúbených a prejsť na zoznam obľúbených destinácií jedného s týchto používateľov.
4. Posledný prípad použitia 4 popisuje obrazovku TSP. Na tejto obrazovke bude môcť používateľ prezerat svoju mapu optimálnej cestovateľskej trasy, pridávať a odoberať z trasy destinácie zo zoznamu obľúbených.

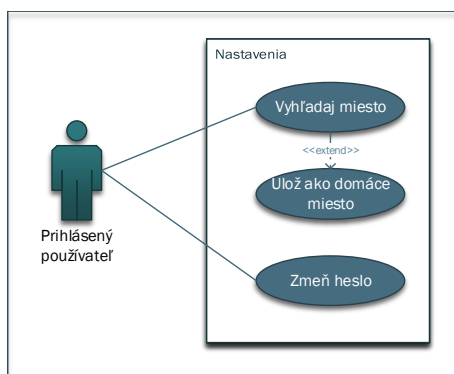
3.0.2 Štruktúra databázy

Na obrázku č. 5 našej databáze sa budú nachádzať dve hlavné entity: entita používateľa *appUser* a entita destinácie *appPlace*. Entita používateľa bude obsahovať iba základné atribúty ako meno a atribúty potrebné pre autentifikáciu používateľa email, a heslo. Entita destinácie bude obsahovať atribúty potrebné na identifikáciu tejto destinácie: unikátny názov a krátky názov vhodný na zobrazovanie a atribúty potrebné na lokalizáciu destinácie: zemepisná šírka a zemepisná dĺžka. Keďže použijeme grafovú databázu nebude potrebné pridávať cudzie kľúče reprezentujúce ich vzťahy.

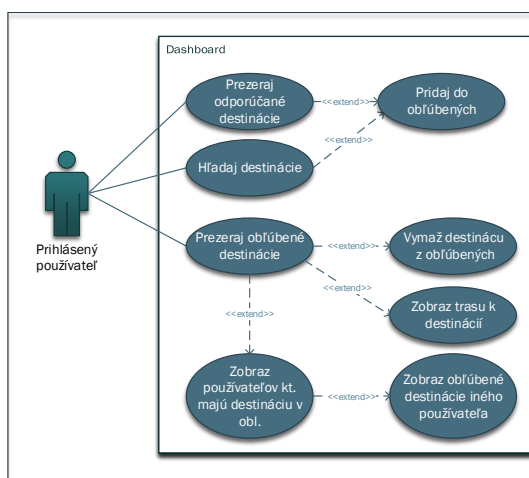
Ďalej bude naša databáza obsahovať hrany: trasa *ROUTE*, obľúbené *FOLLOWS*, a zoznam pre optimalizáciu trasy *TSP*. Hrana trasy bude vždy viesť od jedného miesta k druhému reprezentuje zoznam trás ktoré budú viesť od jednej destinácie k druhej.



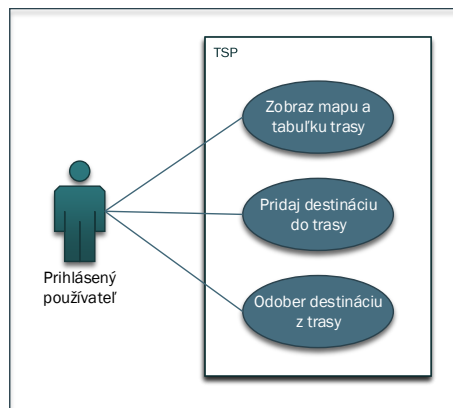
Obrázok 1: Prípád použitia - Registrácia, autentifikácia a nastavenia



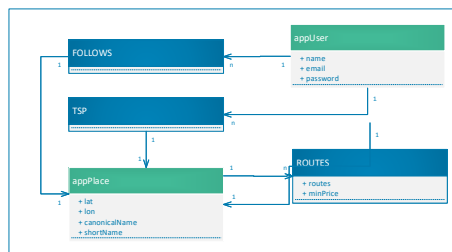
Obrázok 2: Prípád použitia - Registrácia, autentifikácia a nastavenia



Obrázok 3: Prípád použitia - dashboard



Obrázok 4: Prípád použitia - tsp



Obrázok 5: Class diagram

Bude obsahovať atribút *minPrice* hovoriaci o cene najlacnejšej trasy medzi spomínanými destináciami a atribút *routes* obsahujúci serializované detaily všetkých trás. Hrany *FOLLOWS* a *TSP* budú vždy smerovať od používateľa k destinácii a nebudú obsahovať žiadne atribúty.

3.1 Výber grafovej databázy

Vybrali sme si troch najpopulárnejších predstaviteľov grafových databáz podľa rebríčka DB-Engines.com[1]. V nasledujúcej časti v skratke priblížime históriu každej grafovej databázy ich výhody, nevýhody pre naše použitie.

3.1.1 NEO4J

Prvá verzia Neo4J bola vydaná v roku 2007 od vtedy sa stala dlhodobo najpoužívanějšíou grafovou databázou. Je vyvíjaná Neo Technology, Inc. Neo4j je 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 dlhodobá najrozšírenejšia grafová databáza, má dobrú dokumentáciu a natívne podporuje OGM (mapovanie objektov na graf) mapovanie dát na objekty. Práve táto natívna podpora OGM umožnila vznik prehľadného nástroja NeoEloquent, ktorý sa dá využiť na integrácie Neo4J do frameworku Laravel. Ďalšou výhodou je dopytovací jazyk Cypher, ktorý táto databáza môže využívať. Tento jazyk má širokú, prehľadnú dokumentáciu. Nevýhodou je, že podporuje iba grafový model ukladania údajov, nepodporuje ukladanie dokumentov ani key-value úložisko.

3.1.2 OrientDB

OrientDB je vyvíjaný od roku 2010 firmou OrientDB LTD. Databáza OrientDB rýchlo nabralo popularitu 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 dokumentové a key-value úložisko a podľa nezávislých testov [?] je v niektorých testoch až desaťkrát rýchlejšie ako Neo4J. Nevýhodou OrientDB je horšia dokumentácia a nástroj na integráciu to frameworku Laravel *oriquent* [?] nepodporuje zapisovanie a čítanie z atribútov hrán.

3.1.3 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 a Apache Spark podporuje rôzne distribuované úložné priestory ako napr. Apache HBase a Oracle BerkeleyDB Ďalej podporuje rôzne vyhľadávacie enginy ako napr. Elasticsearch a Solr

Toto riešenie je pre naše použitie nevhodné, lebo Titan nepodporuje ovládače pre PHP, celé riešenie je skôr orientované na veľké distribuované enterprise riešenia.

4 Implementácia

V tejto kapitole si priblížime proces

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 knižnica na vytváranie projektov s interaktívnymi mapami. Je založená na voľne šíriteľnej knižnici Leaflet vyvíjanou Vladimírom Agafonkinom, rozširuje túto knižnicu o funkcie ako automatické zoskupovanie bodov do skupín a poskytuje prehľadnú dokumentáciu. My používame verziu zdarma, ktorá je obmedzená na 50000 zobrazení mapy na mesiac.

Túto knižnicu postupne nahrádza Mapbox-gl.js vyvíjaná tou istou firmou, ktorá využíva technológiu WebGL na lepšiu akceleráciu vykresľovania, no v čase začiatku projektu táto knižnica ešte nepodporovala vyššie spomenuté zoskupovanie bodov.

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 nie je 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* [5], 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.

4.3 Inštalácia a konfigurácia databázy neo4j

[4] Aby sme mohli nainštalovať databázu Neo4j musíme si najprv do systému pridať repozitár Neo Technology, potom je nám k dispozícii na inštaláciu balík neo4j. Po inštalácii je nám ihneď dostupné administračné rozhranie databázy na adrese: *localhost:7474*. Pri prvom prihasení 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 administračnému rozhraniu databázy bez otvorenia portu 7474 verejnosti využívame SSH tunel. Administračné rozhranie používa okrem portu 7474 ešte port 7687 lebo na komunikáciu s databázou využíva technológiu Web-Socket.

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

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     ],
```

ich a vyhľadávať v nich. Pri práci s hranami je mierne odlišná. Najprv treba zdefinovať ktorý objekt môže mať aké vzťahy, tieto vzťahy treba unikátne identifikovať ich typom 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 `belongsTo`. 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 reťazec, ktorý sa generuje každému používateľovi pri zobrazení formulára, tento istý reťazec 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, odoberanie obľúbených a zobrazovanie obľúbených destinácií, 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 jenu 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 destináciu 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 nesprávne 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 vystúpila 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 vek cache súboru a referenciu na súbor na disku. Keďže v momentálnej podobe nevyužívame celú odpoveď tohto API mohli by sme zoptimalizovať využitie miesta na disku tým, že by sme najprv údaje spracovali a uložili len tie, ktoré využívame.

4.8 TSP

V aplikácii sa nachádza obrazovka, na ktorej si môže používateľ vybrať destinácie, pre ktoré by chcel vyrátať trasu optimálnu vzhľadom na cenu. Táto trasa bude začínať v domovskej lokalite používateľa, prejde všetkými vybranými destináciami práve raz a vráti sa späť do domácej lokácie. Toto je klasický prípad NP-complete problému pcestného obchodníka.

Pre jednoduchosť a spoľahlivosť sme naimplementovali prístup rekurzívneho prehľadania všetkých možností trás. Naimplementovali sme rekurzívnu funkciu v PHP, ktorá

prejde všetkými možnými trasami medzi našimi destináciami a vyberie prechod grafom, ktorý je optimálny vzhľadom na cenu celkovej trasy. Do argumentu naša funkcia berie východzie miesto, zoznam destinácií ktoré ešte neprehľadala a domáce miesto do ktorého sa má nakoniec vrátiť. V každom vnorení sa zoznam neprehľadaných vrcholov zmenšuje. Vyššie spomenuté argumenty sú perzistentné objekty resp. polia inšancií perzistentnej triedy *Place*.

Keď sa táto funkcia dopytuje na cenu trasy vedúcej z východzieho vrcholu do cieľového zavolá ktorá dá dopyt na našu databázu, ak sa táto trasa nenachádza v databáze vypýta si údaje o konkrétnej trase od API Rome2Rio. Keďže každý dopyt na API trvá cca 200ms prvé volanie pri väčšej trase trvá niekoľko násobne dlhšie ako každé nasledujúce.

Aj po načítaní všetkých potrebných trás však bolo naše riešenie príliš pomalé, pri siedmych miestach už trvalo rátanie TSP až 7 minút. Toto bolo spôsobené tým, že naša funkcia, ktorá počíta s výpočtovou komplexitou $N!$ beží na vrstve PHP a pri každom volaní funkcie invokes knižnicu NeoEloquent a dopytuje sa na hranu medzi dvoma destináciami. Keď funkcia v takomto režime bežala videli sme, že proces neo4j konzistentne bral cca 30% z procesorového času. Z tohto dôvodu sme naimplementovali cache na dopyt k databáze a priemerný čas sa znížil na 80 sekúnd.

Naša implementácia front endu vyžaduje dva dopyty na náš server, jeden na zobrazenie mapy a druhý na zobrazenie tabuľky s výsledkami, z tohto dôvodu sme pridali ešte jeden level cache na celé volanie funkcie pre rátanie TSP. Taktiež sme upravili implementáciu front endu tak, aby sa tieto dva dopyty vykonali vždy synchronne za sebou. Týmto sme znížili nároky na výpočtové kapacity nášho serveru.

Dlhý čas výpočtu pri dopytovaní sa na databázu pripisujeme tomu, že naše dopyty na databázu boli diskkrétne pri každom vnorení funkcie. Aj keď sme používali perzistentné objekty implementácia každého dopytu NeoEloquent pozostáva z jedného dopytu v jazyku Cypher na databázu toto volanie obsahuje vyhľadanie začiatočného a konečného bodu podľa id $O(\log(n))$ a následne vyhľadanie hrany $O(1)$. Tento prístup nevyužíva hlavnú výhodu grafovej databázy, no s použitím NeoEloquent nie je možné postaviť komplexný dopyt, ktorý by vedel vyriešiť TSP za jeden beh.

Takýto dopyt je v Neo4J možné implementovať pomocou dopytovacieho jazyka Cypher príklad takéhoto dopytu môžeme vidieť na ukážke 4. Aj tento dopyt však iba permutuje všetky riešenia a vráti optimálne a preto v je v reálnom prípade vhodnejšie použiť známe algoritmy, v programovej vrstve.

Algoritmus 4 Ukážka riešenia TSP pomocou dopytovacieho jazyka Cypher

```
1      MATCH (from:Node {name: "Source node" })
2      MATCH path = (from) -[:CONNECTED_TO*6] ->()
3      WHERE ALL(n in nodes(path) WHERE 1 = length(filter(m in nodes(path)
4      AND length(nodes(path)) = 7
5      RETURN path ,
6      reduce(distance = 0, edge in relationships(path) | distance + edge
7      AS totalDistance
8      ORDER BY totalDistance ASC
9      LIMIT 1
```

4.9 Mapa

Na vizualizáciu destinácií a trás sme na rôznych miestach a aplikácií použili javascriptovú knižnicu Mapbox.js. Na inicializáciu mapu sme naimplementovali funkciu ??.

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.

Ako zdrojový formát dát pre všetky mapy v našej aplikácii používame štandardný formát GeoJson. GeoJson je dátová štruktúra zakódovaná do formátu JSON obsahujúca informácie potrebné na vykresľovanie kartografických veličín. V našej implementácii používame nasledovnú štruktúru GeoJson objektu. Hlavný objekt GeoJson obsahuje typ, u nás *FeatureCollection*, a pole objektov typu *Feature*. Každý z týchto objektov obsahuje pole koordinátov, a objekt dodatočných atribútov slúžiacich na upresnenie vizuálu

vykresleného objektu. Spomínané pole koordinátov nám spôsobovalo mierne nedorozumenia, lebo v číslovanom poli je ako prvý prvok je uložená zemepisná dĺžka (longitude) a ako druhý zemepisná šírka (latitude) [3], čo je presne opačne ako všetky ostatné API ktoré sme používali.

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.

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.

Zoznam použitej literatúry

- [1] AT, S. I. Db-engines ranking - popularity ranking of graph dbms, 2017.
- [2] Ian Robinson, Jim Webber, Emil Eifrem. *Graph Databases*. 2. O'Reilly Media, Inc. USA, 2015. s. 224. ISBN: 9781491932001.
- [3] (IETF), I. E. T. F. The gejson format, 2016.
- [4] NEOTECHNOLOGY. Neo4j debian packages, 2017.
- [5] OTWELL, T. Laravel - the php framework for web artisans, 2017.

Prílohy