TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Systém pro získávání a analýzu dat o dopravních nehodách

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika

Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Robin Dvořák**

Vedoucí práce: Ing. Bc. Marián Lamr Ph.D.

Liberec 2017

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2016/2017**

ZADÁNÍ DIPLOMOVÉ PRÁCE

Jméno a příjmení:	**Bc. Robin Dvořák**
Název práce:	**Systém pro získávání a analýzu dat o dopravních nehodách**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**
Vedoucí práce:	**Ing. Bc. Marián Lamr**
Rozsah práce:	**40—50 stran**
Konzultant:	**Ing.Přemysl Svoboda**

Zásady pro vypracování:

1. Seznamte se s daty o dopravních nehodách a jejich strukturou.
2. Prostudujte nejběžnější dostupné mapové systémy, zhodnoťte jejich možnosti a vyberte jeden pro účely diplomové práce.
3. Vytvořte webovou aplikaci umožnující přehledně vytvářet analýzy a exporty dat o dopravních nehodách.
4. Webová aplikace bude obsahovat administrační rozhraní pro vkládání a úpravu dat o nehodách.
5. Navrhněte aplikační rozhraní umožnující komunikaci s různými klientskými aplikacemi.
6. Vytvořte klienta pro OS Android umožňující vkládání záznamů o dopravní nehodě do databáze nehod.

Seznam odborné literatury:

[1] PECINOVSKÝ, Rudolf. Návrhové vzory: [33 vzorových postupů pro objektové programování]. Vyd. 1. Brno: Computer Press, 2007. ISBN 978-80-251-1582-4.

[2] GUTMANS, Andi, Stig Sæther BAKKEN a Derick RETHANS. Mistrovství v PHP 5. Vyd. 2. Brno: Computer Press, 2008. ISBN 978-80-251-1519-0.

[3] ZAKAS, Nicholas C. JavaScript pro webové vývojáře. Brno: Computer Press, 2009. Programujeme profesionálně. ISBN 978-80-251-2509-0.

V Liberci dne .....

.....
Ing. Bc. Marián Lamr

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že texty tištěné verze práce a elektronické verze práce vložené do IS STAG se shodují.

28. 9. 2017 Bc. Robin Dvořák

Abstrakt

Tato diplomová práce popisuje proces tvorby webové aplikace pro získávání a analýzu dat o dopravních nehodách. Shrnuje problematiku související s interpretací dat o dopravních nehodách ve vybraném mapovém systému a poskytuje základní možnosti analýzy těchto dat. Dále práce zabývá vytvořením aplikačního rozhraní pro vkládání dat z aplikací třetích stran. Funkčnost tohoto rozhraní je demonstrována vytvořením aplikace pro operační systém Android.
Klíčová slova: analýza, Android, clusterování, dopravní nehoda, filtrování, mapa, PHP, SQL, React, React Native, Redux, Typescript, web, webová aplikace

Abstract

This thesis discuss the process of creation web application for acquisition and analysis of data about car accidents. It summarizes the problematics related to interpretation of car accident data in the selected map system and provides basic possibilities of analysing these data. Thesis also follow up the creation of an application interface for data entry from third party applications. The functionality of this interface is demonstrated by creating an Android application.
Keywords: analysis, Android, car accident, clustering, filtering, map, PHP, SQL, React, React Native, Redux, Typescript, web, web application

Poděkování

Obrovský dík patří panu Ing. Bc. Mariánu Lamrovi Ph.D. nejen za vedení této práce, ale i za jeho podporu v těžkých chvílích a především za jeho nekonečnou trpělivost a pochopení.

Dále bych rád poděkoval svým rodičům, za to, že se mě vždy snažili vést správným směrem a umožnili mi studovat.

Obsah

Seznam zkratek

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DOM	Document Object Model
GB	Gigabyte
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JS	JavaScript
JSON	JavaScript Object Notation
MB	Megabyte
MVC	Model View Controller
OSM	Open Street Map
PBF	Protocolbuffer Binary Format
PHP	Hypertext Preprocessor
PNG	Portable Network Graphics
RAM	Random Access Memory
SDK	Software Development Kit
SSD	Solid State Drive
URL	Uniform Resource Locator
VPS	Virtual Private Server
XML	Extensible Markup Language

Úvod

Dopravní nehody se stávají každý den po celém světě. V případě, že je k dopravní nehodě přivolána policie, v rámci vyšetřování a protokolování nehody vzniknou data, která lze analyzovat a hledat mezi nimi souvislosti.

Data, která byla využita při řešení této diplomové práce byla získána na internetových stránkách ministerstva dopravy a jedná se o databázi s celkovým počtem přesahujícím 650 000 dopravních nehod od roku 2007 až do současnosti (2017).

Ke hledání souvislostí v těchto datech je potřeba vytvořit systém, který se postará o správu dat a umožní je jednoduchým způsobem analyzovat. Pro vyhodnocení dat, je vhodné mít možnost informace zobrazit na mapě a umožnit analýzu konkrétní oblasti vybrané uživatelem. Toho lze využít např. při hledání nebezpečných úseků na silničních komunikacích.

Cílem práce je vytvořit webovou aplikaci, která umožní data o dopravních nehodách spravovat, zobrazit informace o nehodách v mapě a umožnit základní analýzu nehodových oblastí. Pro tyto účely je třeba vytvořit pokročilý filtr, který umožní vyfiltrování nehod na základě zadaných parametrů pro uživatelem vybranou oblast. Na základě výběru dopravních nehod pak spočítat základní statistiky a umožnit export dat do aplikací třetích stran, např. do data-minigových nástrojů, pro hledání skrytých asociací v datech.

Webová aplikace by měla obsahovat rozhraní pro komunikaci s jinými aplikacemi. Toto rozhraní pak bude využívat mobilní aplikace pro operační systém Android, umožňující vkládat do databáze nová data.

Práce je kromě úvodu a závěru rozdělena do pěti kapitol, které tvoří 2 logické celky. První kapitola uvede čtenáře do problematiky a seznámí ho s daty o dopravních

nehodách. Následně shrnuje analýzu dostupných mapových systémů.

Druhý celek provází čtenáře návrhem systému, podle požadavků na aplikaci a implementací jednotlivých částí. Tento celek je rozdělen do čtyř kapitol, kde první kapitola se zabývá návrhem systému a další kapitoly provedou čtenáře implementací jednotlivých částí, rozdělených na serverovou, klientskou a administrační část a mobilní aplikaci.

1 Analýza

Kapitola čtenáře uvede do problematiky a seznámí ho s daty o dopravních nehodách. Každá dopravní nehoda se skládá ze 44 atributů, které jsou rozděleny do šesti logických skupin, na které bude odkazováno jako na skupinu atributů.

Druhá část kapitoly porovnává mapové systémy využitelné on-line - od vlastního řešení a po aplikace třetích stran.

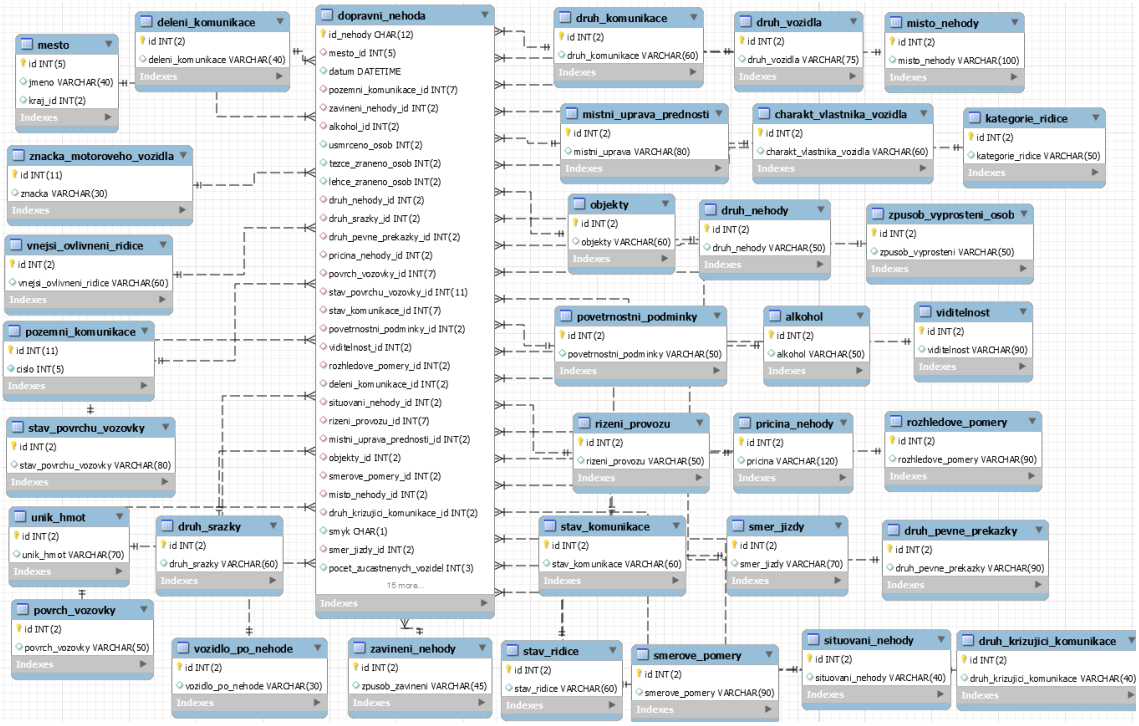
V poslední části kapitoly se autor zabývá výběrem a popisem technologií využitých pro vývoj webového portálu a mobilní aplikace.

1.1 Data o dopravních nehodách

Data byla obdržena v MySQL databázi, která byla vytvořena jiným studentem. Stejný student vytvořil skript, který stahuje data o nehodách z webu ministerstva dopravy a ukládá je do databáze. Zkoumání, či editace existujícího skriptu není součástí této práce.

Databáze obsahuje 36 tabulek, kde nejdůležitější tabulkou je tabulka s názvem „dopravni_nehoda“. Tabulka obsahuje záznamy o jednotlivých nehodách, kterých je více než 600 000 a ostatní tabulky slouží jako číselníky pro jednotlivé atributy.

Atributy dopravní nehody jsou doplňovány Policií České republiky při vyplňování protokolu přímo na místě činu. Číselníky atributů byly staženy zároveň s daty o dopravních nehodách z webových stránek Ministerstva dopravy a autor do nich nebude nijak zasahovat [1]. Původní schéma databáze je zachyceno na obrázku 1.1



Obrázek 1.1: Schéma původní databáze

1.1.1 Atributy dopravní nehody

Z textu výše vyplývá, že atributy mohou být kategoriální - nabývají hodnot z konkrétního číselníku a nominální, které jsou vyjádřeny jednou hodnotou, buď číselným datovým typem nebo datovým typem *boolean*. Do kategoriálních atributů jsou zařazeny informace o tom, zda řidič požil alkohol a obsah naměřeného alkoholu v krvi („ano, obsah alkoholu v krvi do 0,99‰“, „ano, obsah alkoholu v krvi 1‰ a více“, atd.). Číselníky často obsahují intervalové hodnoty, ale i výčty např. ohledně vlastníka vozidla. Tyto výčty nejsou vždy kompletní a obsahují i volbu typu „Žádná z uvedených“, díky čemuž může docházet k odchylce. Do nominálních atributů spadají informace o datu a čase, počtu usmrcených osob, lehce zraněných osob, těžce zraněných osob, ekonomických ztrát a lokaci dopravní nehody.

Místo a datum nehody

První skupina obsahuje základní údaje o dopravní nehodě - GPS souřadnice ve formátu Longitude, Latitude. Dále pak město a datum. Databáze neobsahovala propojení měst s kraji a okresy, což by znemožnilo vytvářet statistiky na základě krajů, respektive okresů. Tato data byla doplněna v rámci optimalizace databáze rozebírané v sekci 2.2.

Silniční komunikace v místě nehody

Druhá skupina upřesňuje informace o silniční komunikaci, kde se dopravní nehoda stala a jsou do ní zařazeny tyto atributy: Druh pozemní komunikace, Směrové poměry, Číslo pozemní komunikace, Dělení komunikace, Místo dopravní nehody, Situování nehody na komunikaci, Řízení provozu v době nehody, Místní úprava přednosti v jízdě.

Účastníci nehody

Do třetí skupiny jsou zařazeny atributy týkající se řidiče a dalších účastníků nehody. Obsahuje informace o počtu zraněných a usmrcených osob a dále pak tyto atributy: Způsob vyproštění osob z vozidla, Vlastník vozidla, Kategorie řidiče, Stav řidiče, Vnější ovlivnění řidiče, Alkohol u viníka nehody.

Vozidlo

Čtvrtá skupina popisuje zúčastněná vozidla a obsahuje tyto atributy: Počet zúčastněných vozidel, Druh vozidla, Výrobní značka motorového vozidla, Vozidlo po nehodě, Rok výroby vozidla, Únik provozních hmot, Škoda na vozidle, Celková hmotná škoda.

Počasí

Pátá skupina se skládá z těchto atributů: Stav povrchu vozovky v době nehody, Povětrnostní podmínky v době nehody a Viditelnost. Počasí má na dopravní ne-

hody významný vliv a nejdůležitějším atributem v této skupině jsou povětrnostní podmínky.

Příčina nehody

Do poslední skupiny byly zařazeny tyto atributy: Zavinění nehody, Druh nehody, Druh srážky jedoucích vozidel, Druh pevné překážky, Hlavní příčina nehody, Stav komunikace, Rozhledové poměry, Specifické objekty v místě nehody, Smyk, Druh povrchu vozovky, Směr jízdy nebo postavení vozidla.

1.2 Mapové systémy

Mapových systémů existuje celá řada, my se zaměříme na několik systémů, které jsou nejrozšířenější na internetu. Kromě systému poskytovaného třetí stranou je možnost vytvořit si mapový server vlastní. Realizace vlastního serveru byla otestována a oba přístupy jsou porovnány v závěru kapitoly.

1.2.1 Kritéria pro výběr mapového systému

Výběr správného mapového systému závisí na platformě, kde bude mapa využívána. Provozovatelé mapových systémů obvykle poskytují několik API rozdělených podle platformy. Základem je obvykle Javascript API pro web, které poskytují všichni zkoumaní provozovatelé. Dále pak aplikační rozhraní pro mobilní zařízení (operační systémy iOS a Android).

Druhým kritériem pro správný výběr je formát mapy. Existují dva formáty, vektorový a rastrový formát. Rastrová mapa je složena ze čtverečkových obrázků označovaných jako *Tiles*. Tyto čtverečky musí být vygenerovány pro každou úroveň přiblížení mapy (*zoom*). Klientovi jsou čtverečky servírovány ve formátu *jpeg* na základě přijatých parametrů o poloze a úrovni přiblížení.

Ve vektorové mapě jsou všechny objekty popsány matematicky. V tomto případě se obraz počítá vždy znovu, což přináší výhodu v tom, že mapa bude ostrá při libovolném přiblížení a nejsme závislí na předgenerovaných datech.

1.2.2 Google Maps

Společnost Google nabízí několik mapových API rozdělených podle platforem, kde budou mapy využívány - Android, iOS, Web a Webové služby. API pro webové služby umožňují využití pokročilých služeb, jako např. navigování, geolokace nebo sledování převýšení libovolného místa na Zemi.

Pro webové aplikace lze využít Google Maps Javascript API, které obstarají veškerou práci s mapu, od zobrazení až po vkládání objektů [3]. Využití tohoto aplikačního rozhraní je podmíněno registrací u společnosti Google a vygenerování soukromého API klíče, který je odeslán při každém požadavku na zobrazení mapy. Základní využití map od této společnosti je zdarma a do 25 000 načtení mapy denně si společnost nic neúčtuje [2]. Nevýhodou ovšem je, že společnost nabízí pouze rastrové mapy[1].

1.2.3 Open Street Map

Open Street Map (OSM) je iniciativa k vytváření a poskytování geografických dat - zdarma a pro všechny [4]. Geografická data poskytují sami uživatelé a každý má možnost do mapy přispět a aktualizovat ji. OSM poskytuje data pro vykreslování map, neposkytuje API pro jejich přímé použití jako např. Google Maps. Nicméně existuje spousta firem i nadšenců, kteří provozují mapové služby založené pouze na datech od OSM. Jedním ze zástupců využívajících data OSM je Mapbox popsaný níže.

1.2.4 Mapbox

Mapbox je platforma pro vývojáře a jedná se o podobnou platformu, jako jsou Google Maps. Liší se v tom, že využívá veřejná data od nadace Open Street Map a uvolňuje svůj software jako open source, který je dostupný v jejich GitHub repozitáři[2]. Od

[1] V době provádění rešerše.

[2] `https://github.com/mapbox/`

konkurence se odlišuje svým Mapbox studiem[3], umožňujícím přizpůsobit mapu svým potřebám. Další výhodou je to, že kromě rastrové mapy poskytuje API k vektorovým mapám pomocí JavaScriptového SDK Mapbox GL JS [6].

Cenová politika společnosti je podobná, jako v případě společnosti Google, do určitého počtu požadavků (50 000 načtení měsíčně) je zdarma [5]. Pro naše účely si vystačíme s variantou zdarma, proto se ceníkem prémiových balíčků nebudeme zabývat.

1.2.5 Vlastní mapový server

Existují případy, kdy je vhodné využít vlastní mapový server, např. v případě nějakých specifických požadavků na mapy. Při výběru této varianty vznikne další aplikace o kterou je nutné se starat, udržovat ji a aktualizovat mapové podklady.

U služeb od společností Google a Mapbox stačí registrace, vytvoření API klíče a mapy jsou připravené k použití. V případě provozování vlastního serveru je to pochopitelně složitější a pro porovnání je úvodní konfigurace mapového serveru popsána níže.

Mapové podklady

Výše už byla zmíněna nadace Open Street Map, která sbírá data pro mapové podklady od uživatelů a umožňuje jejich využití. Mapové podklady lze stáhnout na internetových stránkách společnosti Geofabrik[4]. Společnost umožňuje stáhnout data pro celou planetu i specifické oblasti, jako kontinenty, či jednotlivé státy.

V našem případě jsme využili možnosti stáhnout data pouze pro Českou republiku a to především kvůli požadavkům na výkon serveru. V případě mapy pro celou planetu, je potřeba server s minimální konfigurací 24 GB RAM a 512 GB SSD úložiště [7], protože data pro vygenerování mapy celé planety mají v roce 2017 velikost 256 GB [7]. Přímá URL pro stažení mapových podkladů pro ČR je `http://download.geofabrik.de/europe/czech-republic-latest.osm.pbf` (soubor má

[3] `https://www.mapbox.com/mapbox-studio/`

[4] `http://download.geofabrik.de/europe/czech-republic.html`

velikost přes 600 MB). Data pro celou planetu je možné získat i přímo ze serveru Open Street Map[5], pro naše účely je ale vhodné využít data pouze pro Českou republiku.

Konfigurace serveru

Instalace je popsána sadou bash příkazů, které jsou spouštěny v terminálu operačního systému Ubuntu. Předpokládá se čtenářova základní znalost tohoto prostředí.

```
sudo apt-get install software-properties-common
```

První příkaz instaluje základní softwarové závislosti, balíček software-properties-common obsahuje balíčky ca-certificates, gir1.2-glib-2.0, python-apt-common, python3, python3-dbus, python3-gi, python3-software-properties.

Následuje přidání repozitáře Open Street Map, který obsahuje softwarový balíček libapache-mod-tile.

```
sudo add-apt-repository ppa:kakrueger/openstreetmap
```

Po přidání repozitáře je nutné aktualizovat lokální seznam balíčků.

```
sudo apt-get update
```

Po aktualizaci seznamu se musí nainstalovat výše zmiňovaný balíček libapache-mod-tile a jeho závislosti. Během instalace je uživateli položeno několik dotazů ohledně konfigurace. V našem případě si vystačíme s výchozím nastavením, proto zde není potřeba nic měnit a stačí potvrdit výchozí hodnoty.

```
sudo apt-get install libapache2-mod-tile
```

[5] `http://wiki.openstreetmap.org/wiki/Planet.osm`

Import mapových podkladů

Stažení mapových dat proběhne pomocí příkazu níže.

```
wget http://download.geofabrik.de/europe/czech-republic-
latest.osm.pbf
```

Data se stáhnou do aktuální složky, ve které je otevřený terminál. Následuje import dat do PostgreSQL databáze pomocí příkazu osm2pgsql. Obě závislosti byly nainstalovány pomocí balíčku libapache2-mod-tile.

```
osm2pgsql --slim -C 36000 --number-processes 4 czech-republic-
latest.osm.pbf
```

Parametry příkazu osm2pgsql jsou závislé na hardwaru serveru.

- Parametr –slim nastavuje ukládání dočasných dat do databáze. Bez použití tohoto módu jsou všechna dočasná data uložena v paměti RAM a v případě nedostatku paměti skončí import dat chybou. Použití tohoto parametru také umožňuje databázi průběžně aktualizovat pomocí datového souboru se změnami. Konfigurací průběžné aktualizace se ale zabývat nebudeme.
- Parametr -C se používá společně s parametrem slim a nastavuje velikost cache v paměti RAM (v MB). V tomto případě se použije 3 GB RAM. Při importu map pro celou planetu se udává minimální doporučená hodnota 17 000 [8].
- Parametr –number-processes udává počet paralelních procesů zpracování dat. V případě, že server obsahuje SSD disk a vhodný procesor, může tento parametr výrazně zrychlit import dat
- Poslední parametr je cesta k souboru PBF s daty

Doba importu je závislá na výkonu serveru. V případě dat pro Českou republiku může tato doba být v řádu minut u nejvýkonnějších serverů, až v řádu hodin u pomalejších strojů.

Po proběhnutí importu dat do databáze je nutné import dokončit spuštěním příkazu:

```
sudo touch /var/lib/mod_tile/planet-import-complete
```

Příkaz touch změní *timestamp* (časovou stopu) souboru planet-import-complete a díky tomu program mod_tile, který servíruje vyrenderované čtverečky s mapou zjistí, že je import kompletní.

Dokončení instalace proběhne restartováním démona renderd.

```
sudo /etc/init.d/renderd restart
```

Pokud vše proběhlo bez problémů, měla by fungovat následující URL: `http://localhost/osm/slippymap.html`, viz. obrázek 1.2.

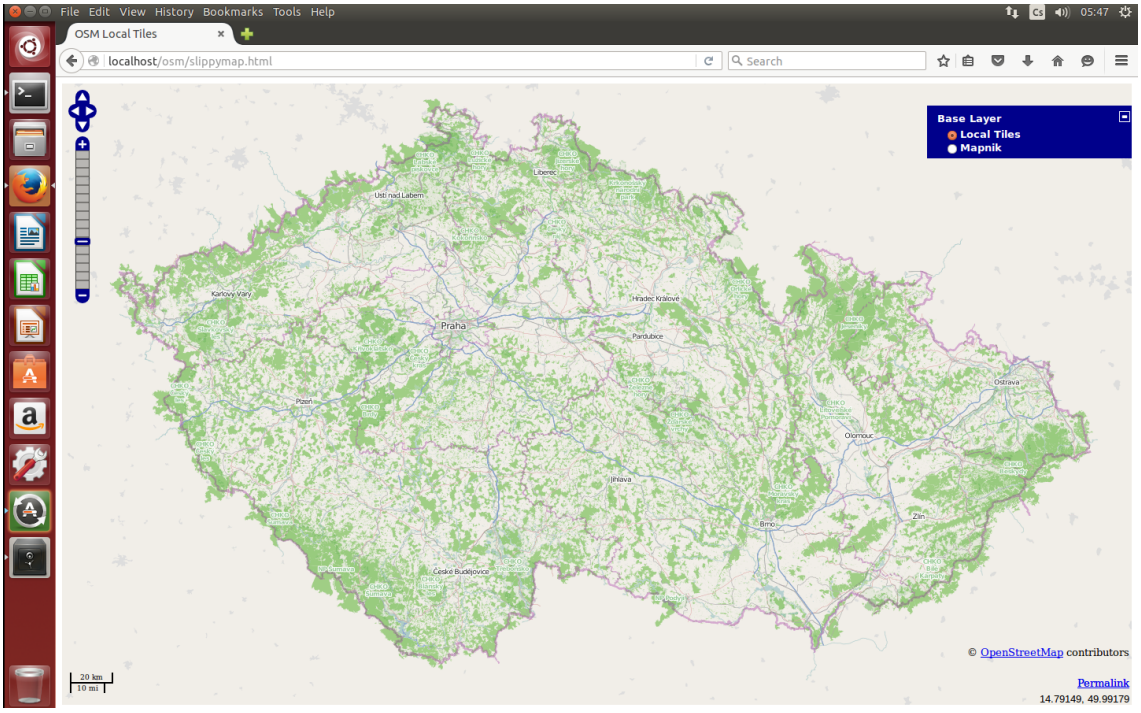
Použití mapy v aplikaci

Mapový server je nyní k dispozici. Pro jednoduchost budeme uvažovat, že mapový server a aplikace budou spuštěny na stejném stroji. Z textu výše vyplývá, že společnosti poskytující mapové systémy zároveň poskytují SDK pro práci s mapou. My máme nyní nakonfigurovaný vlastní mapový server a je potřeba vyřešit, jak s mapou pracovat. Pro použití mapy ve webové aplikaci lze využít framework Leaflet.js [9].

Jako reprezentativní aplikace nám poslouží jednoduchá HTML stránka, do které umístíme kód z ukázky 1.1, kde je provedena inicializace mapy a její základní nastavení pomocí frameworku Leaflet. To umožní uživateli mapu načíst a prohlížet ji.

```
1  // Kontejner mapy
2  <div id="map-container"></div>
3  // JS kód inicializace mapy
4  <script type="text/javascript">
5      var map = L.map('map-container').setView([49.7437500, 15.3386389],
           8);
6      L.tileLayer('http://localhost/osm/{z}/{x}/{y}.png', {
7          maxZoom: 18,
8          minZoom: 7,
9          reuseTiles: true,
10         dragging: true
11     }).addTo(map);
12 </script>;
```

Ukázka kódu 1.1: Inicializace mapy distribuované z vlastního serveru



Obrázek 1.2: Výchozí mapa dostupná z `http://localhost/osm/slippymap.html`

1.2.6 Výběr mapového systému pro aplikaci

Otestovali jsme si oba přístupy, v případě vlastního řešení obnáší zprovoznění mapového systému mnohem více práce, než využití Google Maps nebo Mapboxu, kde si stačí vygenerovat soukromý API klíč a začít službu využívat pomocí dodávaného SDK. Dále je potřeba server pravidelně aktualizovat a provádět jeho údržbu.

Tento problém u řešení od třetí strany odpadá a vzhledem k tomu, že pro požadavky naší aplikace bude postačovat jakákoliv varianta zdarma, byl vybrán systém od společnosti Mapbox a to především protože nabízel využití vektorových map. Dále poskytuje nástroj Mapbox studio, ve kterém lze mapy snadno upravit a přizpůsobit tak svým potřebám.

V případě že je vybírán systém pro nasazení ve velké aplikaci, je nutné dobře promyslet veškeré náklady spojené s pořízením a provozováním vlastního serveru. Přestože se tato varianta může jevit jako výhodnější, nemusí tomu tak úplně být. Systémy třetích stran navíc poskytují mnoho navazujících užitečných API, která lze

využít při rozšiřování aplikace. Lze pak velmi jednoduše implementovat například navigování, výpočty vzdáleností, převýšení nebo převod GPS souřadnic na adresu. V případě vlastního řešení bychom tyto funkce museli implementovat sami.

1.3 Shrnutí

V kapitole byly představeny jednotlivé atributy dopravní nehody a jejich rozřazení do logických skupin. Atributy vycházejí z číselníků Policie ČR a při jejich podrobném zkoumání autor narazil na duplicitní informace. Například v případě srážky se zvěří tuto informaci nese atribut Zavinění nehody i Druh nehody, což by šlo sjednotit. Tento případ není v datech ojedinělý, číselníky jsou ale spravovány Ministerstvem dopravy a kdybychom do nich zasahovali, data bychom poškodili a znemožnili import dat v budoucnosti.

Dále byly diskutovány mapové systémy, včetně vlastní implementace a pro účely aplikace byl nakonec vybrán mapový systém Mapbox.

2 Návrh systému

Následující kapitola představí čtenáři architekturu systému, následně ho provede návrhem jednotlivých částí a seznámí ho s jejich use-casy a funkcionalitou.

2.1 Požadavky na aplikaci

Práce s aplikací by měla vypadat tak, že je uživateli zobrazena mapa České republiky s možností filtrování dopravních nehod na základě atributů nehody. Vyfiltrovaná data se zobrazí na mapě a uživatel s nimi může dále pracovat. Uživatel uvidí souhrnné informace o dopravních nehodách a má možnost data analyzovat nebo exportovat pro použití v desktopových dataminingových aplikacích.

Aplikace by měla splňovat následující požadavky:

- Zobrazit data o dopravních nehodách na mapě
- Filtrovat data na základě uživatelem zadaných kritérií
- Provést jednoduchou analýzu nad vyfiltrovanými daty
- Přes administrační rozhraní vkládat nová data do databáze
- Poskytnout API pro klientské aplikace
- Exportovat data pro pro datamining nástroje a další mapové systémy

Na základě požadavků, je vhodné aplikaci rozdělit do třech logických celků, kde každý celek bude samostatná aplikace a dohromady budou tvořit webový portál.

Aplikace bude pracovat s daty uloženými v SQL databázi a uživatel bude pracovat s HTML stránkou vyrenderovanou webovým prohlížečem, ve které bude načtena mapa. Uživatelské požadavky bude zpracovávat Javascript, který bude zároveň komunikovat s mapovým API a obstarávat veškerou práci s mapou. Serverová část bude napsána v jazyce PHP7 za pomoci některého PHP frameworku.

2.2 Databáze

Místo MySQL databázového serveru byl vybrán MariaDB server, je to jeden z nejpopulárnějších databázových nástrojů na světě a byl vytvořen přímo tvůrci MySQL [10]. MariaDB podporuje více úložných systémů a je optimalizována pro vyšší rychlost než MySQL [11].

2.2.1 Optimalizace databáze

V aplikaci se bude se všemi číselníky pracovat stejným způsobem. Z tohoto důvodu se vyplatí, aby se názvy sloupců ve všech tabulkách číselníků shodovaly. V původní databázi na to není brán zřetel a sloupce hodnot v tabulkách mají různé názvy. U tabulky „mesta“ se sloupec s hodnotami jmenuje „jmeno“ u tabulky kraj se sloupec jmenuje „nazev“. Všechny číselníkové tabulky byly převedeny na formát o dvou sloupcích „id“ a „hodnota“, kde „id“ je unikátní identifikátor původní číselníkové hodnoty Ministerstva dopravy.

Dalším nedostatkem v konvenci pojmenování je, že v jednom případě je název tabulky zkrácen, např. „charakt_vlastnika_vozidla“ a v jiných případech je název kompletní např. „druh_pevne_prekazky“. Vzhledem k délkám názvů jiných tabulek nemá smysl některé názvy zkracovat, naopak při nasazení databáze v aplikaci vzniknou problémy s konvencí pojmenování v aplikaci.

Další nedostatky se týkají datových typů sloupců. Sloupec ID dopravní nehody (primární klíč) je nastaven na datový typ *char* obsahující 12 číslic. Tento způsob byl pravděpodobně zvolen protože, takto byla označena data stahovaná z webu Ministerstva a hodnotu ID je nutné držet kvůli přehledu o tom, které dopravní nehody už

byly staženy. Tento datový typ je ale nevhodný pro použití jako primární klíč, byl proto vytvořen nový sloupeček s datovým typem *integer* a AUTO_INCREMENT[1] nastavením.

Případů, kdy byl na čísla použit datový typ *char* bylo více. Aplikace bude na databázi spouštět náročné filtrovací dotazy, řadit a seskupovat data a na to je datový typ *char* oproti datovém typu *integer* nevýhodný, protože SŘBD musí nejprve řetězec rozparsovat, převést na *integer* a až poté provede požadovanou operaci.

Tabulka dopravních nehod navíc neobsahuje žádné indexy, pouze cizí klíče, takže filtrovací dotazy jsou velmi pomalé. Všechny sloupce v tabulce „dopravni_nehoda“ proto byly zaindexované, protože bude možné filtrovat na základě všech atributů dopravní nehody.

Databáze dále neobsahovala datové propojení měst s kraji a okresy. To způsobilo nekonzistence v datech, protože stahovací skript pak nemohl správně přiřadit města, která mají stejný název.

Testování databáze probíhalo spouštěním SQL dotazu, který bude využit v aplikaci. Jako referenční byl zvolen dotaz 2.1, který vybere z tabulky „dopravni_nehoda“ 100 dopravních nehod napojených na všechny číselníky.

```
1      SELECT d.id
2        FROM dopravni_nehoda d
3  INNER JOIN mesto m
4          ON (m.id = d.mesto_id)
5  INNER JOIN pozemni_komunikace pk
6          ON (pk.id = d.pozemni_komunikace_id)
7
8  /* INNER JOIN na zbývajících 34 tabulek */
9  LIMIT 100;
```

Ukázka kódu 2.1: Dotaz pro výběr dat s napojením číselníků

Server tento dotaz na původní databázi nedokázal ani zpracovat. Po optimalizacích popsaných výše trvalo spuštění dotazu 100ms.

[1]ID se k přidanému řádku do tabulky přidá automaticky a navýší se jeho hodnota pro další záznam

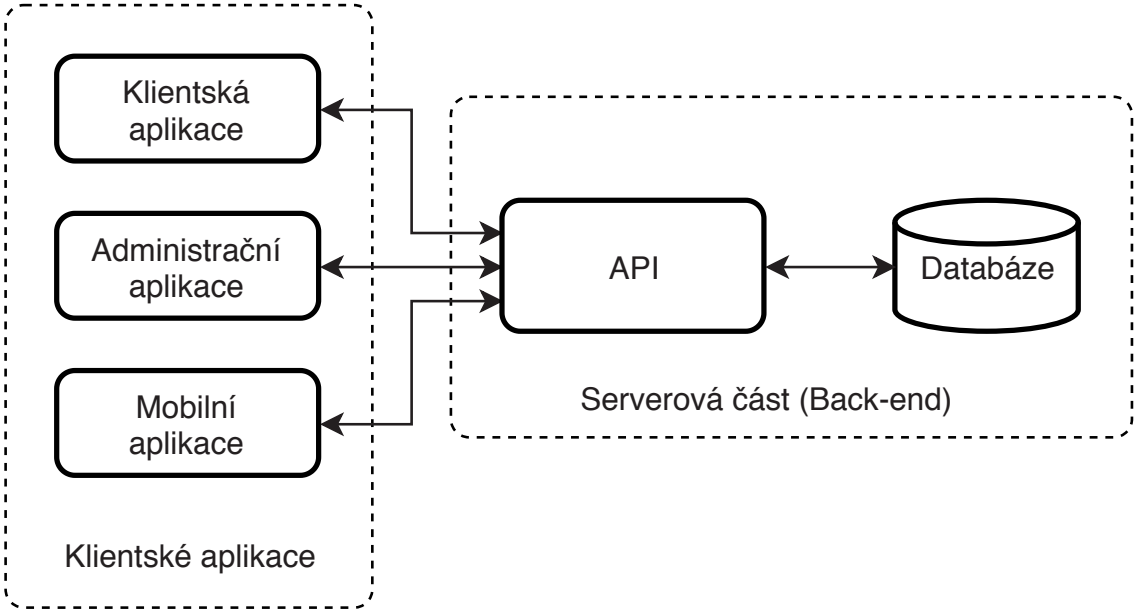
2.2.2 Hustota dopravy

Autor od vedoucího práce obdržel vedle databáze dopravních data o hustotě dopravy datované k roku 2016. Databáze se skládá ze dvou tabulek, „hustota“ a „hustota_max“, „hustota_max“ párovala data na jednotlivé nehody.

Abychom mohli hustotu dopravy využít k filtrování nehod, byla na základě těchto dat rozšířena tabulka „dopravni_nehoda“ o sloupec „hustota_dopravy“, kam byla data exportována.

2.3 Architektura systému

Architekturu budou tvořit 4 části, API napojené na databázi, které bude servírovat data a tři klientské aplikace. Architektura je zachycena na obrázku 2.1.



Obrázek 2.1: Schéma systému na správu dopravních nehod

Klientská aplikace

Klientská aplikace je určená uživatelům, kteří budou data o dopravních nehodách konzumovat. Aplikace umožní uživatelům data filtrovat, zobrazit dopravní nehody

na mapě a následně nad daty provádět základní analýzu.

Administrační aplikace

Administrační aplikace slouží jako správce databáze a uživatelů. Tento modul je zodpovědný za kompletní správu dat, lze v něm upravovat textace číselníků a přidávat a mazat dopravní dopravní nehody. Dále je modul zodpovědný za správu uživatelů.

Mobilní aplikace

Mobilní aplikace pouze umožní vložit dopravní nehodu. Slouží jako demonstrace vložení vložení dat do databáze od třetí strany.

Serverová část

Serverová část bude sloužit jako vrstva mezi klientskými aplikacemi a databází. Databázi zpřístupní jako REST API a bude zodpovědná za správné filtrování a následné servírování dat klientským aplikacím.

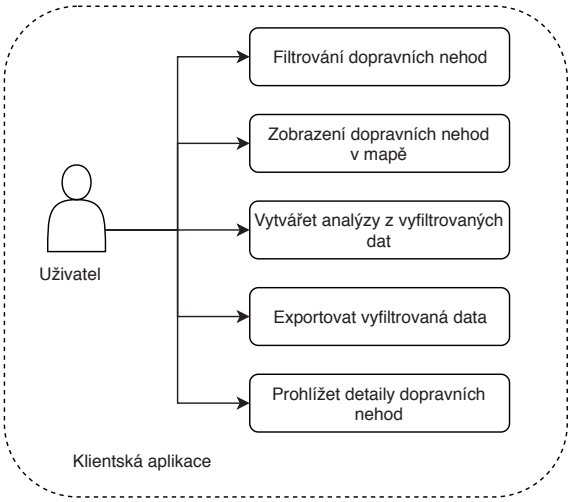
2.4 Klientská aplikace

Klientská aplikace je hlavní částí celého systému. Zde bude uživateli zobrazena mapa, ve které se budou zobrazovat dopravní nehody.

Po vyfiltrování dopravních nehod budou data zobrazena v mapě, uživatel si bude moct prohlédnout detaily konkrétních nehod, vytvářet z dat grafy a zobrazit si přehledy o vyfiltrovaných nehodách. Konkrétní use-casy aplikace jsou znázorněny na obrázku 2.2.

2.4.1 Mapa

Mapa bude klíčovou komponentou aplikace a pro tyto účely byla vybrána knihovna Mapbox GL JS, důvody byly popsány v sekci 1.2.6. Knihovna je poháněna technologií WebGL, která je podporována ve všech současných verzích webových prohlížečů,



Obrázek 2.2: Use-case diagram klientské aplikace

kromě Opera mini [2], pro správné fungování aplikace se tedy předpokládá, že uživatel disponuje webovým prohlížečem, který touto technologií disponuje [6].

Mapa bude vhodně zobrazovat dopravní nehody na základě GPS souřadnic a umožní uživateli po kliknutí na jednu konkrétní nehodu zobrazit její kompletní detail v dialogovém okně.

2.4.2 Filtrování dopravních nehod

Filtrování bude uživateli prezentováno jako vícekrokový formulář v samostatném dialogovém okně. Jednotlivé kroky budou rozděleny do skupin atributů, které byly popsány v sekci 1.1.

Formulář je rozdělen do kroků především z toho důvodu, že lze filtrovat na základě všech 44 atributů dopravní nehody a uživatel se tak může ve filtru lépe orientovat.

Kategoriální atributy budou uživateli prezentovány ve formě *select boxu*, kde bude moct vybrat více hodnot z konkrétního číselníku, uživateli se bude zobrazovat číselníková hodnota a technicky bude výběr proveden na základě unikátního identifikátoru. Pro atribut alkohol, může uživatel vybrat hodnoty „ano, obsah alkoholu

[2] `https:\caniuse.com/#feat=webgl`

v krvi od 0,5‰ do 0,8‰“ a „pod vlivem drog a alkoholu“, atd. Na tento výběr bude aplikováno pravidlo *OR* (nebo). V případě, že uživatel do výběru zahrne ještě například město. Pak bude ta atributy alkohol a město aplikováno pravidlo *AND* (a zároveň).

Nominální atributy týkajících se zraněných a usmrcených osob, počtu vozidel a napáchaných škod budou omezené na rozsahy nominálních hodnot, protože např. filtrování na základě celkové hmotné škody nastavené přesně na určitou částku je podle autora irelevantní.

Uživatel bude mít možnost procházet si historii filtrování i po tom, co web opustí a vrátí se později. Na základě historie si poté může data vyfiltrovat znovu aniž by znovu vyplňoval celý filtrovací formulář.

2.4.3 Vytvoření analýzy z vyfiltrovaných dat

Po vyfiltrování dopravních nehod bude mít uživatel k dispozici dialogové okno s přehledem dat o vyfiltrovaných nehodách. Data aplikaci poskytne API na základě unikátních identifikátorů dopravních nehod, které data vytvoří přímo na úrovni databáze.

Dialogové okno s analýzou obsahuje několik záložek - Základní přehled, Grafy, Seznam nehod, DBScan a Export dat.

Základní přehled obsahuje informace o počtu dopravních nehod vyhovujících filtrovacím kritériím, informace o celkových škodách a počtu zraněných a usmrcených osob. Tyto informace budou viditelné nejen v dialogovém okně ale i na hlavní stránce s mapou, aby měl uživatel přehled o datech, i když zrovna pracuje s mapou a nemá otevřené dialogové okno s analytikou.

Dále budou v této záložce vytvořeny přehledy o rozložení dopravních nehod po krajích a okresech, které jsou uživateli prezentovány formou sloupcových grafů.

V záložce grafy bude mít uživatel možnost vytvořit si graf pro všechny kategoriální atributy dopravní nehody. Pro všechny atributy půjde nezávisle vybrat, jaký typ grafu chce uživatel zobrazit a to sloupcový, koláčový nebo radarový graf.

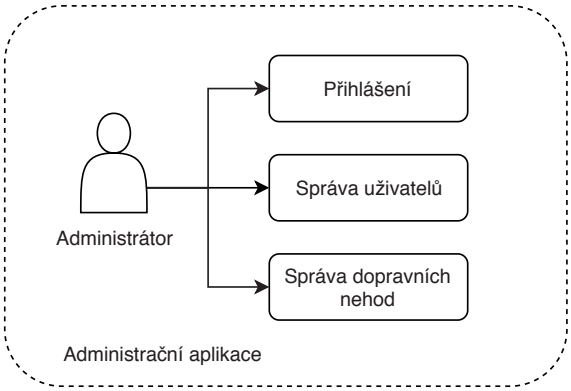
Další funkcionalitou je sestavení vlastního grafu. To uživateli umožní vybrat

si hodnoty napříč všemi kategoriálními atributy a vytvořit z nich graf. Není tak omezený pouze na grafy s hodnotami jednoho konkrétního číselníku, ale může si graf přizpůsobit svým potřebám.

Záložka seznam nehod reprezentuje tabulku, s detaily k jednotlivým dopravním nehodám. Uživatel zde bude moct procházet a prohlížet jejich detaily. Tabulka bude mít stránkování, aby se předcházelo načítání velkého množství dat a uživatel si bude moct navolit počet zobrazených záznamů na stránku.

2.5 Administrační aplikace

Administrační aplikace bude mít na starosti správu všech systémových dat. Bude v ní možné upravovat a revidovat data o dopravních nehodách a spravovat uživatele. Use-casy aplikace jsou znázorněny na obrázku 2.3



Obrázek 2.3: Use-case diagram administrátorské aplikace

2.5.1 Přihlášení uživatele

Přístup do aplikace bude omezen pouze přihlášeným uživatelům. Uživatel se bude přihlašovat přes přihlašovací formulář pomocí uživatelského jména a hesla.

Po vyplnění přihlašovacích údajů bude odeslán požadavek na API, které se postará o ověření uživatele a vrátí aplikaci odpověď, zda-li má uživatel požadované oprávnění nebo ne.

2.5.2 Správa uživatelů

Přidávání nových uživatelů bude fungovat na principu odeslání pozvánky na email s unikátním odkazem, který bude mít dočasnou platnost.

Pozvaný uživatel si zvolí heslo a následně se bude moct přihlásit do systému. Zároveň se uživateli aktivuje přístup do mobilní aplikace, ze které bude možné přidávat dopravní nehody.

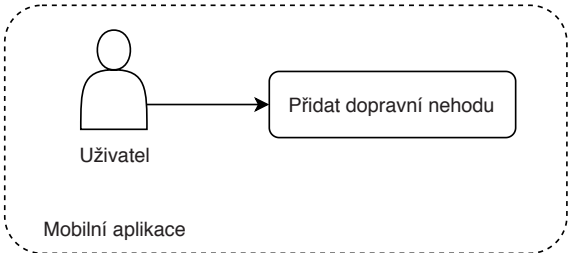
Pozvánku může odeslat jakýkoliv přihlášený uživatel a to tak, že na stránce „Pozvat nového uživatele“ vyplní jméno a email člověka, kterého chce pozvat do systému.

2.6 Mobilní aplikace

Mobilní aplikace bude sloužit k pohodlnějšímu přidání dopravní nehody do systému.

Přidání dopravní nehody bude podmíněno přihlášením do systému. Přidat dopravní nehodu bude moci pouze člověk, který byl do systému pozvánku některým z administrátorů, jak bylo řečeno v sekci 2.5.2.

Aplikace umožní uživateli vyplnit všechny atributy dopravní nehody a pro zjednodušení vyplňování dat v terénu by měla umět načíst GPS lokaci zadavatele. Tato funkcionalita najde využití v případě vyplňování dat přímo na místě, kde se nehoda stala.



Obrázek 2.4: Use-case diagram mobilní aplikace

3 Serverová část aplikace

Serverová část je aplikační rozhraní (API), které klientům poskytuje data ve formátu JSON se specifikací ECMA-404 [19]. API je napsané v programovacím jazyce PHP7 za pomoci frameworku Symfony.

3.1 Technologie

V současné době se webové aplikace nepíší v čistém PHP, ale využívá se PHP frameworků. Jejich předností je to, že programátora vedou ke správnému návrhu aplikace a kladou důraz na bezpečnost, protože eliminují výskyt bezpečnostních děr a jejich zneužití (např. XSS - Cross Site Scripting, SQL injection) [12].

Frameworků existuje celá řada, jmenovitě např. Nette, Laravel, CakePHP a výše zmíněné Symfony.

3.1.1 Symfony framework

Symfony je open source framwerok pro vytváření webových aplikace. Využívá návrhového vzoru Dependency Injection a MVC - (Model - View - Controller), který aplikaci rozděluje na tři logické části, které lze upravovat nezávisle na sobě [13].

Model

Model je funkční a datový základ aplikace, obsahuje aplikační logiku. Jakákoliv uživatelská akce (zobrazení dat z databáze, přihlášení uživatele) je akcí modelu. Model poskytuje rozhraní, který umožňuje zjišťovat a měnit jeho stav, to obvykle probíhá v Controlleru. O Controlleru ani o View Model neví [12].

View

View je vrstva aplikace, která má na starosti jen a pouze zobrazení výsledku požadavku uživateli. View může využívat šablonovací systém, který usnadňuje práci oproti klasickému výpisu z PHP. Ve frameworku Symfony je to šablonovací systém Twig, ten v našel případě nevyužijeme, protože programujeme API, které poskytuje data klientským aplikacím ve formátu JSON [13].

Controller

Controller zpracovává požadavky od uživatele a následně volá patřičnou aplikační logiku v modelu. Po vykonání požadavku předá získaná data View (šabloně), které data vykreslí nebo odešle klientovi odpověď ve formátu JSON [12].

Dependency injection

Dependency injection (DI) je návrhový vzor, jehož cílem je odebrat třídám zodpovědnost za získávání závislostí, tj. objektů, které potřebují ke své činnosti. Místo toho jim DI předá závislosti přímo při jejich vytváření [12].

Vytváření závislostí se děje ještě před spuštěním aplikace a všechny instance jsou uloženy v dalším objektu - DI kontejneru. Jednou z výhod tohoto přístupu je lepší čitelnost kódu, je totiž jasně vidět, na jakých třídách je komponenta závislá, zároveň je kód testovatelný a lépe použitelný. Programátor také nemusí řešit, odkud závislost dostane, postará se o to DI kontejner [12].

Výhodou je to, že správu instancí tříd za nás spravuje kontejner a my jako vývojáři se o to nemusíme starat. Instance třídy se zároveň vytváří až ve chvíli, kdy je opravdu potřeba [12].

3.1.2 Object Relational Mapper

Objektově relační zobrazení (ORM) složí k automatické konverzi dat z relační databáze do objektově orientovaného programovacího jazyka [14]. Jedna dopravní nehoda je v databázi definována jako řádek tabulky. V rámci aplikace ale chceme

s dopravní nehodou pracovat jako s objektem, který bude instancí třídy DopravniNehoda, která bude obsahovat metody pro získávání a nastavování jednotlivých atributů.

O to, že budeme moci pracovat s objektem DopravniNehoda se postará knihovna Doctrine, která za nás obstará synchronizaci s databází. Princip je takový, že pro každou tabulku v databázi se napíše třída, která bude obsahovat *gettery* a *settery* na její atributy a synchronizaci s databází se nemusíme starat. Na takto vytvořené objekty budeme odkazovat jako na Entity [14].

3.1.3 Kontrola kvality kódu

Pro udržení kvality kódu se využívá nadefinovaného coding standardu. Typicky se toho využívá v týmech, aby kód vyprodukovaný různými vývojáři vypadal vždy stejně. Autor v práci využil kombinaci standardů Consistence Coding Standard a Slevomat coding standard [16], [17].

Zahrnutí kontroly coding standardu při pushnutí[1] do GIT repozitáře lze zaručit jednotnost kódu nezávisle na tom, jaký vývojář kód napsal. Ke kontrole dodržování standardu byl použit nástroj PHP Code Sniffer [15].

Dalším užitečným nástrojem pro kontrolu kvality kódu je PHPStan, který provádí statickou analýzu kódu a lze díky němu předcházet nepředvídatelnému chování aplikace [18].

PHPStan má definovaných sedm úrovní striktnosti analýzy a pro vývoj byla využita nejvyšší úroveň.

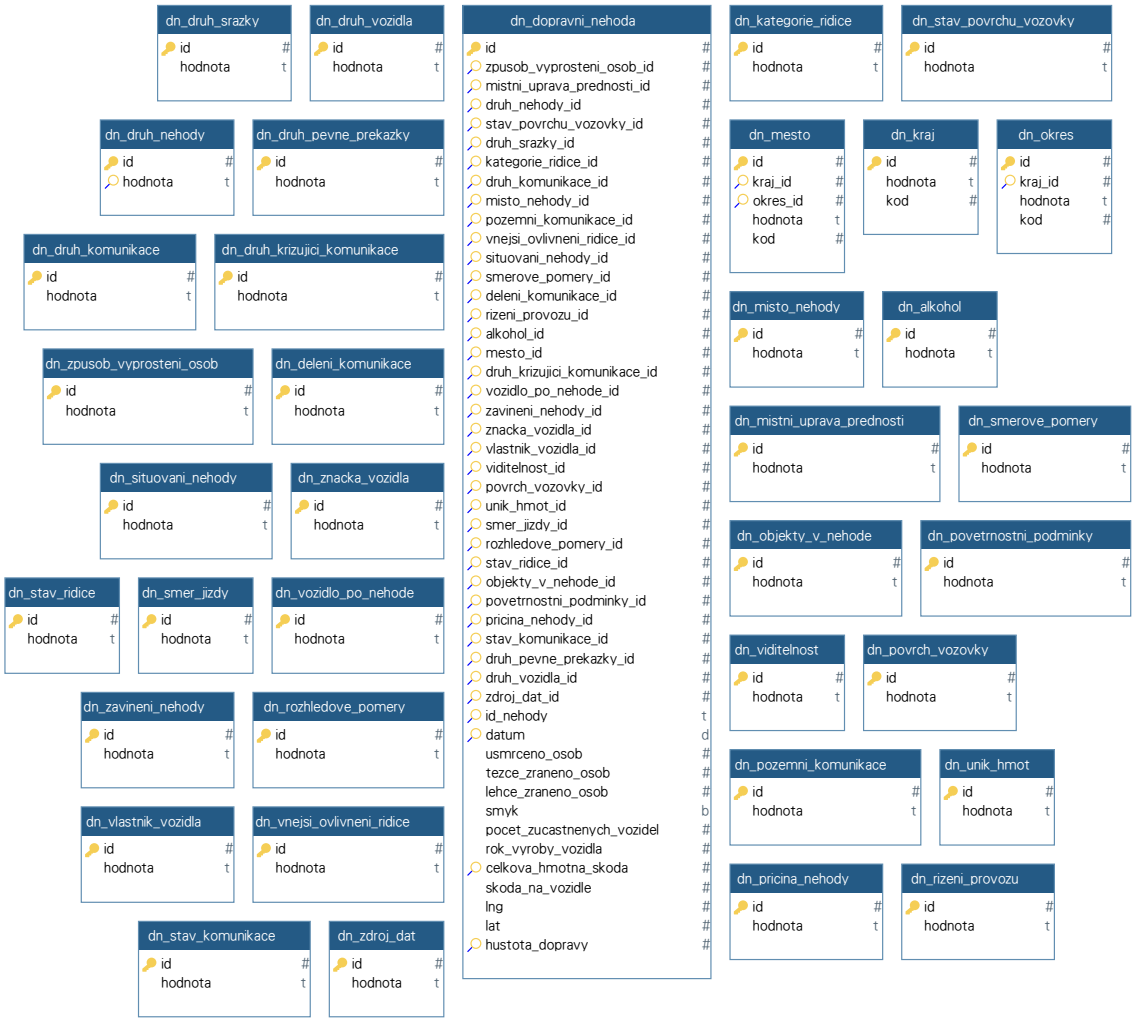
3.2 Model

Model byl definován databází, která byla obdržena od vedoucího práce. Protože byly provedeny optimalizace popsané v sekci 2.2, struktura databáze byla pozměněna. Nejviditelnějším zásahem bylo přidání prefixu „dn“ ke všem tabulkám týkajících se dopravní nehody. Prefix na první pohled oddělí tabulky, které se netýkají doprav-

[1] Promítnutí změn

ních nehod. Nová struktura je zachycena na obrázku 3.1. Pro přehlednost musely být odstraněny vazby cizích klíčů, ale všechny číselníkové tabulky jsou propojeny s tabulkou „dopravni_nehoda“.

Jak bylo řečeno v sekci 3.1.2, s jednotlivými tabulkami se v kódu pracuje jako s entitami a za běhu aplikace jsou reprezentovány instancí třídy.



Obrázek 3.1: Schéma modelu dopravní nehody

3.3 Serializace dat

Server data distribuuje ve formátu JSON, je tedy nutné zajistit konverzi PHP objektů do serializované podoby. O to se ve frameworku Symfony starají normalizery, což je třída definující, jaké atributy chceme serializovat a odesílat na klienta [13].

Symfony si automaticky načte třídy normalizerů do DI kontejneru a data serializuje podle jejich předpisu. Do jisté míry lze využívat obecných normalizerů dostupných přímo ve frameworku, ale v případě, že potřebujeme data serializovat do specifické struktury, musíme si napsat vlastní. Ukázka 3.1 definuje předpis struktury dat pro grafy zobrazované v klientské části.

```
1  class ChartNormalizer implements \Symfony\Component\Serializer\
       Normalizer\NormalizerInterface
2  {
3      public function normalize(App\Model\Chart $object, $format = null,
           array $context = []): array
4      {
5          $normalizedData = [];
6
7          foreach ($object as $chartItem) {
8              $normalizedData[] = [
9                  'label' => $chartItem->getLabel(),
10                 'total' => $chartItem->getTotal(),
11             ];
12         }
13
14         return [
15             'name' => $object->getName(),
16             'data' => $normalizedData
17         ];
18     }
19
20     public function supportsNormalization($data, $format = null): bool
21     {
22         return $data instanceof Chart;
23     }
24
25 }
```

Ukázka kódu 3.1: Normalizer objektu Chart reprezentující data pro vykreslení grafu

3.4 Testování

V Symfony lze jednoduše testovat pomocí knihovny PHPUnit [13]. Testy jsou nastavené tak, že proběhnou při každém pushnutí do GIT repozitáře a změny se do hlavní větve zapíšou pouze v případě, že všechny testy proběhnou úspěšně.

Testována je především funkčnost API a to na úrovni Controllerů, jednotkové testy v našem případě nemají žádný smysl, protože API je pouze prostředníkem mezi klientem a databází. Zajímá nás tedy pouze to, zda API vrací správná data.

4 Klientská a administrační část

Tato kapitola popisuje koncepty využité při vývoji klientské a administrační části systému. Obě aplikace jsou postaveny na stejných principech, ale jsou odděleny, aby mohly být provozovány nezávisle na sobě.

4.1 Návrh aplikace

Jelikož se jedná o aplikaci, která by do jisté míry měla nahradit desktopovou aplikaci, byla využita technologie Single Page Application (SPA).

Uživatel bude primárně pracovat s mapou, kde budou reprezentována data o dopravních nehodách, filtrovat data a vytvářet grafy. Podrobněji se tomuto tématu věnuje sekce 2.4. Z těch to důvodů byla zvolena technologie React, což je JavaScriptová knihovna pro vytváření uživatelských rozhraní společně s knihovnou Redux, která spravuje globální stav aplikace.

4.2 Single page aplikace

Single page aplikaci reprezentuje jedna stránka, která reprezentuje celou aplikaci. Při prvním načtení stránky se přenesou všechny zdrojové kódy aplikace, které jsou potřeba pro její chod a na základě interakcí uživatele jsou pak dynamicky načítány ostatní zdroje. V našem případě pouze data.

První načtení stránky trvá trochu déle než u statické HTML stránky, ale veškerá další odezva je mnohem rychlejší, protože HTML kód se generuje přímo na straně klienta.

Nevýhodou je pak to, že uživatel musí mít pro běh aplikace zapnutý JavaScript a horší indexace stránek internetovými vyhledávači, což negativně ovlivňuje SEO - Search Engine Optimalization. Ale zařazení aplikace do vyhledávacích výsledků na základě obsahu není pro aplikaci důležité.

4.3 Komunikace se serverem

Klient komunikuje se serverem pomocí technologie AJAX (Asynchronous JavaScript and XML). AJAX umožňuje načíst data ze serveru, aniž by bylo nutné stránku znovu načíst a poté ji aktualizovat. Takže lze data načítat na pozadí, aniž by uživatel byl nějak omezen v dalším používání aplikace.

4.4 Design

Design aplikace vychází z Material Design systému navrženého společností Google [20]. Cílem systému je přinést uživateli konzistentní design, který bude funkční na všech typech zařízení.

Za tímto účelem byla využita knihovna Material UI, která je postavena na technologii CSS in JS. Design tedy není psán pomocí klasického CSS, ale přímo v JS souborech, ze kterých je pak CSS vygenorováno a vloženo přímo do výsledné HTML stránky.

4.5 Vývojové prostředí

Pro kvalitní a pohodlný vývoj SPA je velmi klíčová správná konfigurace vývojového prostředí. Hlavními požadavky jsou předzpracování kódu, automatické obnovení stránky po provedení změn a uložení souboru - Hot Module Reload (HMR) [21].

4.5.1 TypeScript

TypeScript je nadstavbou nad jazykem JavaScript a rozšiřuje ho o statické typování, třídy, moduly a další atributy známé z klasických objektově orientovaných programovacích jazyků [22]. Aby výsledný kód mohl běžet v prohlížečích, kompiluje se do JavaScriptu.

4.5.2 Webpack

Webpack je nástroj pro prostředí NodeJS, jeho účelem je vytvořit námi požadované vývojové prostředí a vytvořit funkční JS balíček námi napsaného modulárního kódu.

Tím jak je webpack vytvořen umožňuje transformovat téměř jakýkoliv druh assetu do balíčku spustitelného webovým prohlížečem. Za assety jsou považovány například obrázky nebo styly, ať už v CSS nebo přímo v JavaScriptu. Pro zpracování assetů webpack využívá loadery a pluginy [21].

Konfiguruje se pomocí konfiguračních souborů, kde je využita rozdílná konfigurace pro vývoj a pro produkci. Při vývoji požadujeme funkcionalitu automatického obnovení stránky, zároveň jako vývojáři chceme vidět *source mapu* kódu, pro ladění v prohlížeči, zatímco na produkčním prostředí chceme tuto funkcionalitu vypnout a kód minifikovat pro optimalizaci rychlosti načítání stránky [21].

4.6 Knihovna React

Knihovna React je rozšířená open source JS knihovna pro vytváření webových komponent vyvinutá firmou Facebook. Její předností je efektivní aktualizace a renderování pouze těch komponent, kde proběhla změna dat.

Knihovna vytváří abstraktní vrstvu nad DOM (Document Object Model) dokumentem, kde zpracovává změny od uživatele. Díky tomu si mohou uzly udržovat vnitřní stav a v případě změny stavu pak React efektivně promítne změny do DOM dokumentu.

4.6.1 JavaScript XML - JSX

JSX je rozšířením JavaScriptové syntaxe umožňující kompilovat stromovou strukturu podobnou XML do objektů standardu ECMAScript [19]. React komponenty se píší pomocí JSX a výsledná struktura renderovací funkce je pak velmi podobná HTML [23]. Podrobněji komponenty popisuje následující sekce.

4.6.2 Komponenty

Pomocí komponent lze prezentační vrstvu rozdělit do logických celků. Jednotlivé komponenty lze zanořovat do sebe a tím vytvářet složitější uživatelské rozhraní.

Komponenta dědí od třídy *Component* či *PureComponent* a musí implementovat funkci render, která vrací uzel JSX.

Důležitými atributy komponenty jsou *props* a *state*. Jako *props* jsou označované atributy předané komponentě z rodičovské komponenty a *state* obsahuje stavové objekty komponenty.

Komponenta disponuje funkcí *setState* a každé zavolání této funkce změní její vnitřní stav a vynutí přerenderování komponenty.

Překreslení komponenty způsobí i změna *props* předávaných z rodičovské komponenty. Zde se dostáváme k rozdílu mezi *Component* a *PureComponent*. Komponenta vycházející z třídy *Component* se překreslí při každé změně *props* zatímco *PureComponent* provede mělké porovnání všech atributů a překreslí komponentu pouze pokud se některý z atributů změnil [24].

```
1  class BasicDialog extends React.PureComponent<Props> {
2      public static defaultProps: Partial<IProps> = {
3          maxWidth: "xs",
4          fullWidth: false,
5      };
6
7      private handleClose = () => {
8          const { closeDialog, name } = this.props;
9          closeDialog(name);
10     };
11
12     public render(): React.ReactNode {
13         const { title, open, children, name, fullWidth, maxWidth } =
               this.props;
14         return (
15             <Dialog open={open} fullWidth={fullWidth} onClose={this.
                   handleClose} maxWidth={maxWidth}>
```

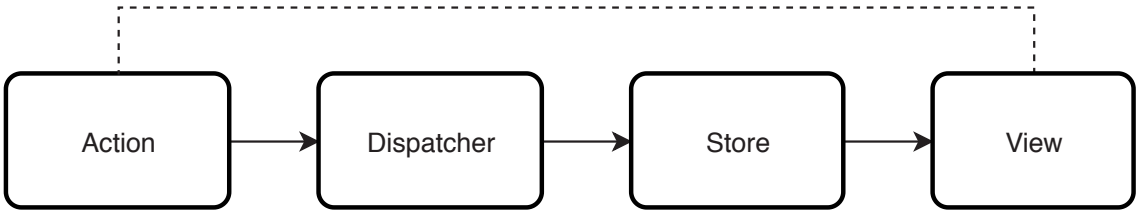
```
16                  {title && <DialogTitle id={`${name}-title`}>{title}</
                        DialogTitle>}
17                  <DialogContent>{children}</DialogContent>
18                  <DialogActions>
19                      <Button onClick={this.handleClose} color="primary">
20                          OK
21                      </Button>
22                  </DialogActions>
23              </Dialog>
24          );
25      }
26  }
```

Ukázka kódu 4.1: Příklad komponenty pro zobrazení dialogového okna

4.7 Návrhový vzor Flux

Návrhový vzor Flux má tři hlavní části, *store* (datové úložiště), *dispatcher* a *view*. Oproti vzoru MVC se liší tím, že Controller a View jsou reprezentovány jednou komponentou, která obdrží data z úložiště a dál je distribuuje svým potomkům, dalším UI komponentám [25].

Interakce uživatele s aplikací vyvolává akce, která je spuštěna pomocí dispatcheru, následně se data zpracují, uloží do *storu* a UI komponenta se překreslí. Oproti MVC se jedná o jednosměrný datový tok. Princip návrhového vzoru je zachycen na obrázku 4.1.



Obrázek 4.1: Návrhový vzor Flux

4.7.1 Knihovna Redux

Redux je knihovna, která nabrala na popularitě v JS komunitě především díky své jednoduchosti [25]. Klíčovými myšlenkami knihovny jsou:

- Všechna data aplikace jsou v jedné datové struktuře, nazývané *state*, která je uložena ve *storu*
- Aplikace si načítá *state* se *storu*
- *Views* (React komponenty) vytvářejí akce, definující, co se stalo
- Nový *state* je vytvořen zkombinováním starého stavu a akcí pomocí funkce nazývané *reducer*

4.7.2 State aplikace

Celková datová struktura aplikace je tvořena z devíti menších datových struktur a je znázorněna na ukázce 4.2. Datové struktury pro menu a dialog definují, jestli je menu respektive některé dialogové okno otevřené nebo zavřené.

Za zmínku stojí struktura *ICodeBookState*, které obsahuje data všech číselníků k dopravním nehodám. Všechny číselníky se stáhnou z API při inicializaci aplikace a jsou využívány pro zobrazení možností filtrování.

```
1  export interface IReduxState {
2      form: FormStateMap;
3      menu: IMenuState;
4      dialog: IDialogState;
5      filter: IFilterState;
6      api: IApiState;
7      chart: IChartState;
8      carAccidents: ICarAccidentState;
9      codeBooks: ICodeBookState;
10     notifications: INotificationsState;
11 }
```

Ukázka kódu 4.2: Redux state aplikace

Nejdůležitější strukturou je *ICarAccidentState*, která nese data o vyfiltrovaných dopravních nehodách. Zajímavá struktura je i *ApiState*, obsahující informace o tom, zda se stahují data z API a jestli požadavek skončil úspěšně či nikoliv. Na základě toho, pak lze uživatele notifikovat, že data jsou kompletně stažena nebo se se někde stala chyba.

5 Mobilní aplikace

Jedním z požadavků na systém bylo vytvoření mobilní aplikace pro operační systém Android. Aplikace slouží jako reprezentativní příklad přidání dopravní nehody přes API a postup implementace aplikace shrnuje tato kapitola.

5.1 React Native

Pro vývoj aplikace byla zvolena technologie React Native a to z toho důvodu, že aplikaci je možné napsat v TypeScriptu.

Hlavní myšlenkou je co nejvíce přiblížit vývoj mobilní aplikace k webovému vývoji, aby vzájemně mohly sdílet zdrojový kód. Z názvu je patrné, že technologie vychází z Reactu, kterému jsme se věnovali v sekci 4.6. Rozdíl je v tom, že nepracujeme se stránkami, ale s obrazovkami mobilního telefonu. Hlavní rodičovskou komponentou tedy nebude *Component*, ale *View*. S komponentami se pak pracuje stejně jako v knihovně React a lze využít i knihovnu Redux pro správu aplikačních dat.

Největším přínosem této technologie je bezesporu to, že napsaný kód lze zkompilovat pro platformu Android i iOS, které jsou v současné době nejrozšířenější.

Kompilace probíhá pro každou platformu zvlášť a výsledkem je opravdová mobilní aplikace nikoliv zabalený web nebo hybridní aplikace, jak by se mohlo na první pohled zdát.

5.2 Návrh aplikace

Oproti původnímu zadání z analýzy byla aplikace rozšířena o přihlášení uživatele. Do aplikace se dostane uživatel, který obdržel pozvánku do administrace systému.

Aplikace se skládá z formuláře, kde uživatel vyplní všechny údaje o dopravní nehodě na základě číselníků. Díky zvolené technologii, bylo možné využít část zdrojových kódů z klientské aplikace.

Takto byl využit API klient pro komunikaci se serverem, Redux *state* a funkce pro práci s číselníky.

5.3 Implementace aplikace

Při implementaci aplikace jsme mohli využít stejné koncepty jako při vývoji klientské a administrační části systému. Pro aplikaci byl využit koncept Material Designu, stejně jako u klientské a administrační části.

Aplikace byla zkompilována pouze pro operační systém Android, protože umožňuje nainstalovat aplikaci, která nebyla stažena z oficiálního obchodu.

Společnost Apple toto na svém operačním systému iOS neumožňuje a aplikace lze distribuovat pouze přes jejich oficiální obchod, kde je nutné mít placený účet. Aplikace byla na tomto systému otestována, ale zkompilovanou verzi není možné nainstalovat jinam, než na zařízení vývojáře.

Závěr

V rámci této diplomové práce vznikl systém na správu dat o dopravních nehodách. Systém se skládá z několika dílčích aplikací, které navenek působí jako jeden webový portál.

Bylo vytvořeno administrační rozhraní pro kompletní správu dat, které je navrženo takovým způsobem, aby se přizpůsobilo jakýmkoliv budoucím změnám ve struktuře dat, či vzniku nových tabulek, v případě dalšího vývoje aplikace.

Klientská část systému poskytuje rozhraní pro filtrování dopravních nehod, které jsou následně přehledně zobrazené v mapě. Z těchto vyfiltrovaných dat si uživatel může ručně vybrat oblast, která ho zajímá a vytvořit z nich datovou vrstvu, kterou bude analyzovat.

Aplikace vytvoří uživateli statistiku ze všech atributů nehody, která je prezentována formou grafů, které si uživatel může přizpůsobit svým potřebám. Byla vyzkoušena i metoda hledání asociačních pravidel mezi nehodami pomocí algoritmu Apriori, ale ukázalo se, že tato metoda není vhodná pro vyhodnocování velkého množství dat pomocí webové aplikace.

K realizaci výpočtů bylo nutné stahovat velké množství dat, i několik stovek MB, v případě filtrování stovek tisíc nehod a výpočty trvaly dlouhou dobu. Z tohoto důvodu aplikace umožňuje data exportovat ve formátu *csv*, aby data mohla být zpracována pomocí desktopových data miningových nástrojů. Jako největší přínos práce autor vidí možnost vizualizace dat v mapě a následný výběr oblasti z mapy, která ho zajímá.

Dále vznikla mobilní aplikace pro operační systém Android, která umožňuje do systému vložit dopravní nehodu.

Při vývoji systému se autor potýkal především s problematikou velkých datových přenosů a nepřiměřeného využití paměti na serveru, při zpracování klientských požadavků, tyto problémy byly nakonec vyřešeny a systém zvládá filtrování i zobrazování několika stovek tisíc dopravních nehod.

Literatura

[1] Ministerstvo dopravy. *Jednotná dopravní vektorová mapa* [online] 2017 [cit. 2017-01-21]. Dostupné z: `http://www.jvdm.cz/`

[2] Google Developers. *Google Maps Pricing and plans* [online] 2017 [cit. 2017-01-21]. Dostupné z: `https://developers.google.com/maps/pricing-and-plans/`

[3] Google Developers. *Google Maps JavaScript API Documentation* [online] 2017 [cit. 2017-01-21]. Dostupné z: `https://developers.google.com/maps/documentation/javascript/`

[4] OpenStreetMap Foundation. *OpenStreetMap* [online] 2017 [cit. 2017-01-21]. Dostupné z: `http://wiki.osmfoundation.org/wiki/Main_Page`

[5] Mapbox. *Plans & Pricing* [online] 2017 [cit. 2016-01-21]. Dostupné z: `https://www.mapbox.com/pricing/`

[6] Mapbox. *Mapbox GL JS* [online] [cit. 2017-01-21]. Dostupné z: `https://www.mapbox.com/mapbox-gl-js/api/`

[7] OpenStreetMap and contributors. *Building a tile server from packages. Switch2osm.org* [online] 2017 [cit. 2017-01-21]. Dostupné z: `https://switch2osm.org/serving-tiles`

[8] PUTZO Andreas. *Manuál softwarového balíku osm2pgsql* [online] 2017 [cit. 2017-03-15]. Dostupné z: `http://man.devl.cz/man/1/osm2pgsql`

[9] AGAFONKIN Vladimir. *Dokumentace JS frameworku Leaflet.js. Leafletjs.com* [online] 2017 [cit. 2017-03-15]. Dostupné z: http://leafletjs.com/reference.html

[10] The MariaDB Foundation. *About MariaDB* [online] 2017 [cit. 2017-03-30]. Dostupné z: https://mariadb.org/about/.

[11] The MariaDB Foundation. *MariaDB versus MySQL - Features* [online] 2017 [cit. 2017-03-30]. Dostupné z: https://mariadb.com/kb/en/mariadb/mariadb-vs-mysql-features/.

[12] Nette Foundation. *Dokumentace PHP frameworku Nette. Nette.org* [online] 2017 [cit. 2017-04-25]. Dostupné z: https://doc.nette.org/cs/2.4/

[13] Symfony SAS. *Symfony documentation* [online] 2017 [cit. 2017-04-25]. Dostupné z: https://symfony.com/doc/current/index.html#gsc.tab=0

[14] Doctrine. *Doctrine documentation* [online] 2017 [cit. 2017-04-25]. Dostupné z: https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/index.html

[15] Squiz Labs. *PHP Code Sniffer* [online] 2017 [cit. 2017-04-25]. Dostupné z: https://github.com/squizlabs/PHP_CodeSniffer

[16] Consistence. *Consistence Coding Standard* [online] 2017 [cit. 2017-04-25]. Dostupné z: https://github.com/consistence/coding-standard

[17] Slevomat.cz, s.r.o. *Slevomat Coding Standard* [online] 2017 [cit. 2017-04-25]. Dostupné z: https://github.com/slevomat/coding-standard

[18] Ondřej Mirtes. *PHP Static Analysis Tool* [online] 2017 [cit. 2017-04-25]. Dostupné z: https://github.com/phpstan/phpstan

[19] Ecma International. *ECMAScript® 2015 Language Specification* [online] 2017 [cit. 2017-06-28]. Dostupné z: http://www.ecma-international.org/ecma-262/6.0/index.html.

[20] Google Inc. *Material Design* [online] 2017 [cit. 2017-06-28]. Dostupné z: https://material.io/design/.

[21] Webpack. *Webpack Concepts* [online] 2017 [cit. 2017-06-28]. Dostupné z: https://webpack.js.org/concepts.

[22] Microsoft Inc. *Typescript* [online] 2017 [cit. 2017-06-28]. Dostupné z: https://www.typescriptlang.org/.

[23] Facebook Inc. *jsX In Depth* [online] 2017 [cit. 2017-06-28]. Dostupné z: https://facebook.github.io/react/docs/jsx-in-depth.html.

[24] Facebook Inc. *React Components and Props* [online] 2017 [cit. 2017-06-28]. Dostupné z: https://facebook.github.io/react/docs/components-and-props.html.

[25] FullStack React. *Intro to Flux and Redux* [online] 2017 [cit. 2017-06-28]. Dostupné z: https://www.fullstackreact.com/p/intro-to-flux-and-redux/.

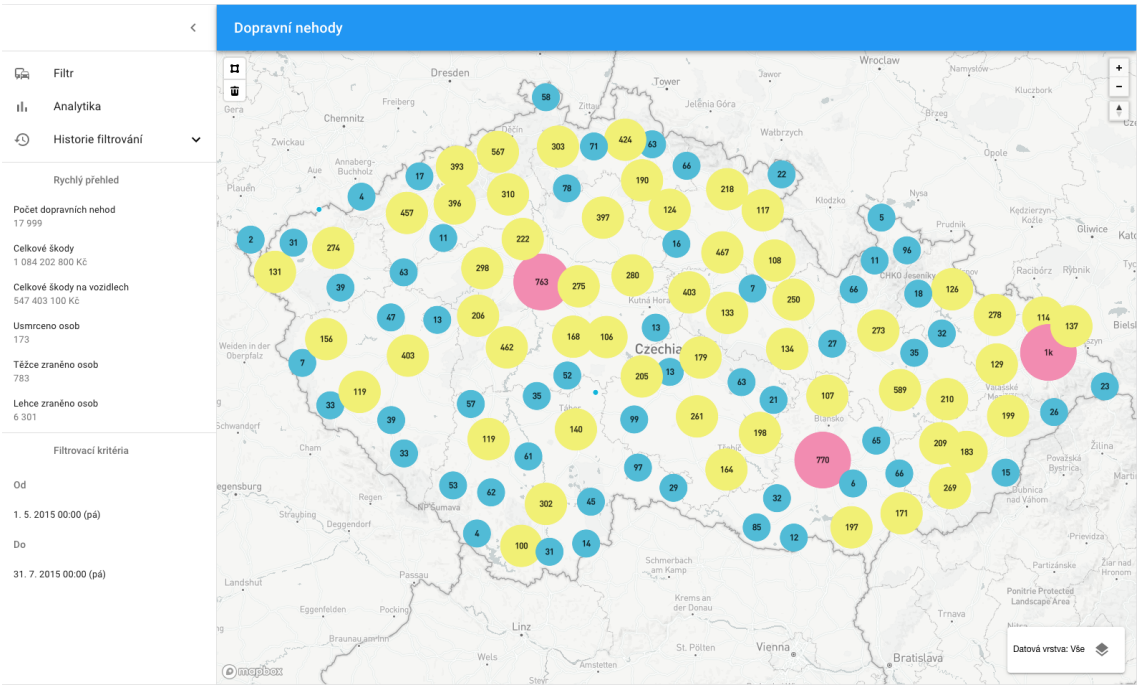
A Ukázky zdrojových kódů

Kompletní zdrojové kódy všech aplikací jsou dostupné včetně instalačních návodů na: `https://gitlab.robindvorak.org/robin/thesis`

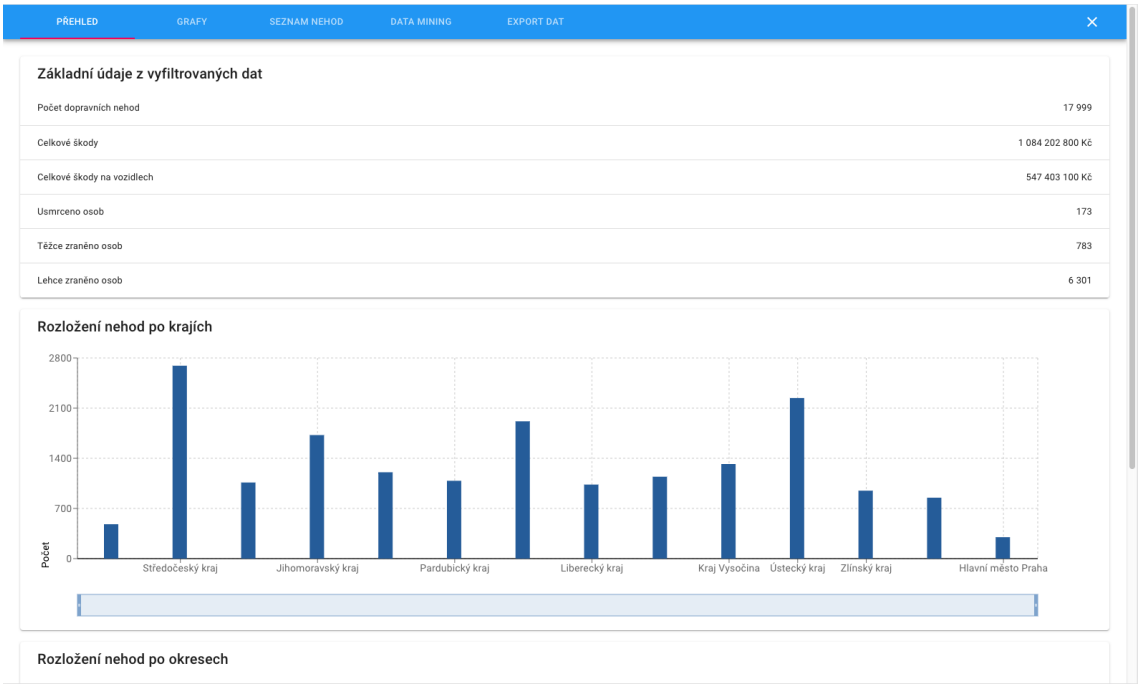
```
export const filterCarAccidentDataThunk = (formData: IFilterFormData)
    => (dispatch: IAppThunkDispatch) => {
        const summarySuccess = (layerId: number) => (summaryResponse:
            ISummary) =>
            saveSummaryForLayer(summaryResponse, layerId);
        const data = processCarAccidentList(response);
        if (response.length > 0) {
            const layer = layerFactory("Vse", data.featureCollection,
                data.ids);
            dispatch(saveCarAccidentLayer(layer));
            dispatch(getSummary(data.ids, summarySuccess(layer.id)));
            dispatch(getChatDataForLayerThunk(data.ids, layer.id));
        } else {
            dispatch(
                createNotification({
                    message: "Nic nenalezeno.",
                    options: {
                        variant: "warning",
                    },
                }),
            );
        }
    };

    dispatch(filterCarAccidents(formData,
        handleFilterCarAccidentsSuccess));
    dispatch(onFilter(formData));
    dispatch(destroy("filterForm"));
};
```

Ukázka kódu A.1: Funkce filtrování dopravních nehod v klientské aplikaci

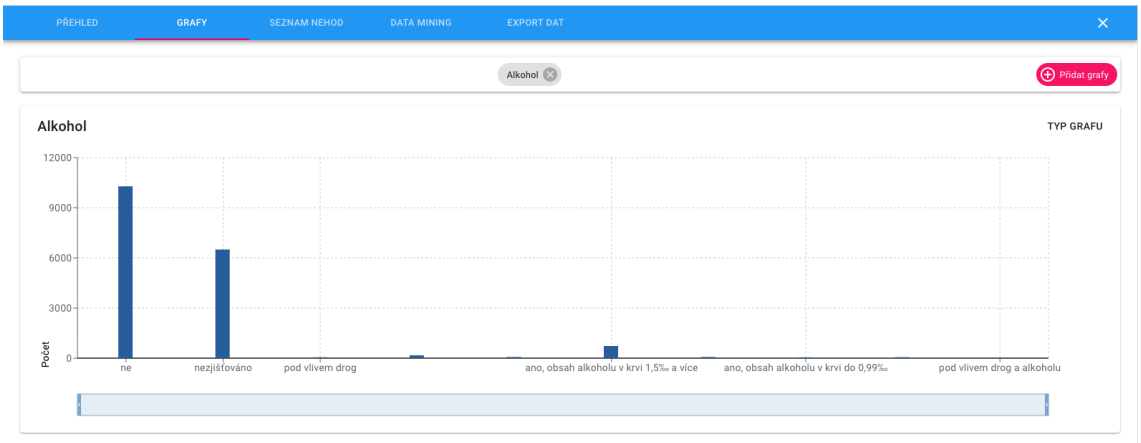
B GUI klientské aplikace



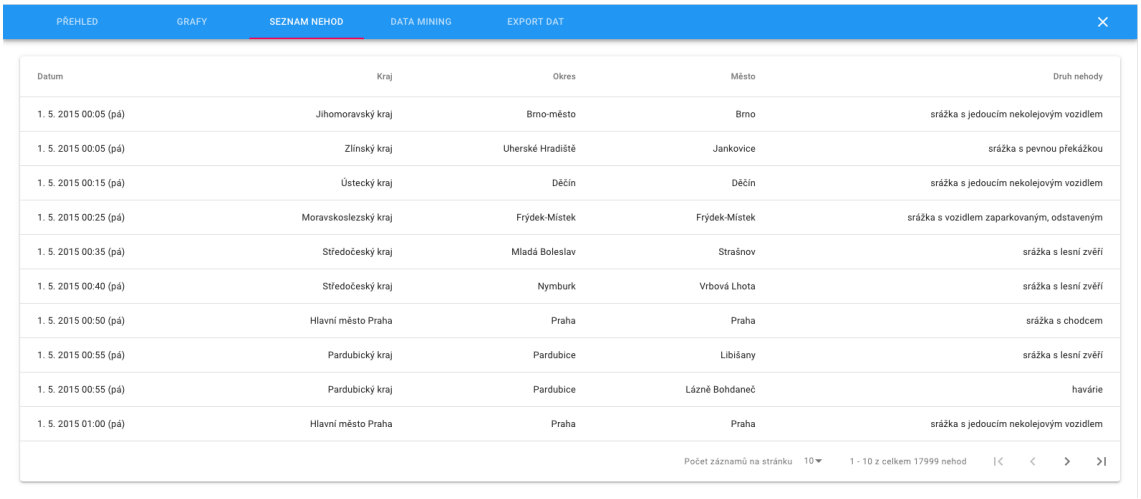
Obrázek B.1: Vizualizace dopravních nehod



Obrázek B.2: Přehled o vyfiltrovaných nehodách



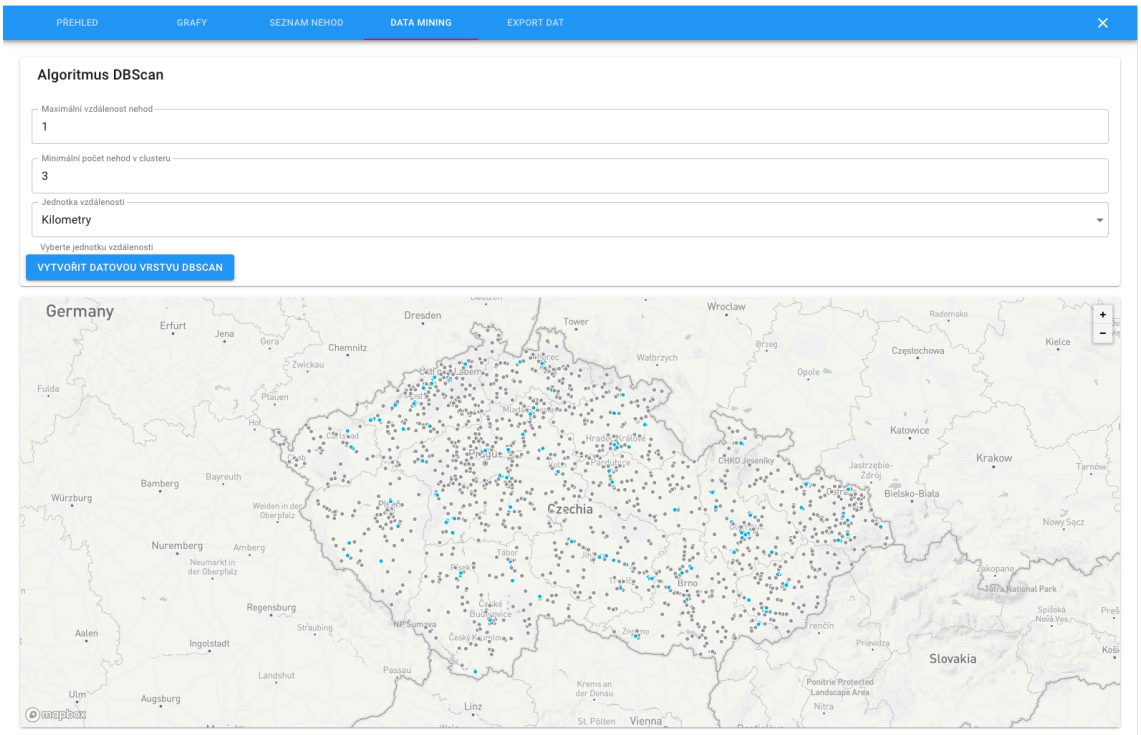
Obrázek B.3: Sloupcový graf počtu dopravních nehod pro číselník Alkohol



Datum	Kraj	Okres	Město	Druh nehody
1. 5. 2015 00:05 (pá)	Jihomoravský kraj	Brno-město	Brno	srážka s jedoucím nekolejovým vozidlem
1. 5. 2015 00:05 (pá)	Zlínský kraj	Uherské Hradiště	Jankovice	srážka s pevnou překážkou
1. 5. 2015 00:15 (pá)	Ústecký kraj	Děčín	Děčín	srážka s jedoucím nekolejovým vozidlem
1. 5. 2015 00:25 (pá)	Moravskoslezský kraj	Frýdek-Místek	Frýdek-Místek	srážka s vozidlem zaparkovaným, odstaveným
1. 5. 2015 00:35 (pá)	Středočeský kraj	Mladá Boleslav	Strašnov	srážka s lesní zvěří
1. 5. 2015 00:40 (pá)	Středočeský kraj	Nymburk	Vrbová Lhota	srážka s lesní zvěří
1. 5. 2015 00:50 (pá)	Hlavní město Praha	Praha	Praha	srážka s chodcem
1. 5. 2015 00:55 (pá)	Pardubický kraj	Pardubice	Libišany	srážka s lesní zvěří
1. 5. 2015 00:55 (pá)	Pardubický kraj	Pardubice	Lázně Bohdaneč	havárie
1. 5. 2015 01:00 (pá)	Hlavní město Praha	Praha	Praha	srážka s jedoucím nekolejovým vozidlem

Počet záznamů na stránku 10 1 - 10 z celkem 17999 nehod

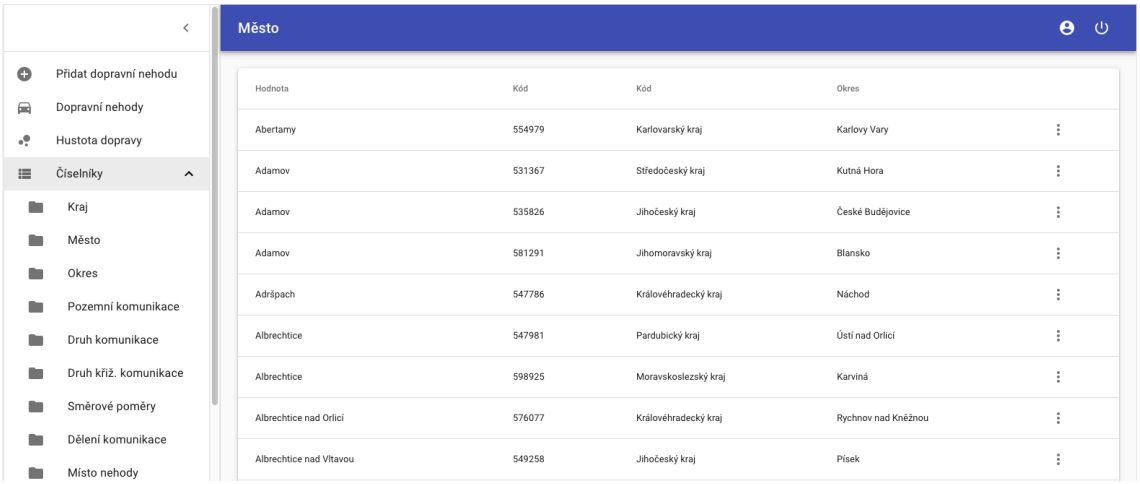
Obrázek B.4: Seznam dopravních nehod



Obrázek B.5: Vizualizace algoritmu DBscan

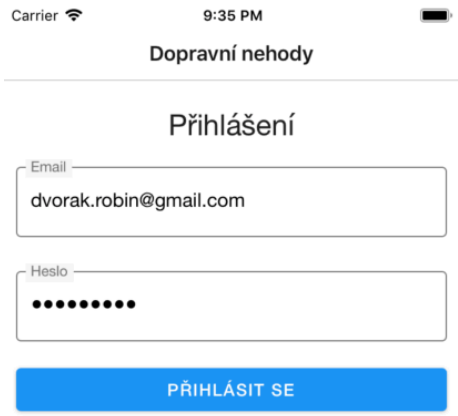
C GUI administrační aplikace

Obrázek C.1: Přihlášení do administrační aplikace

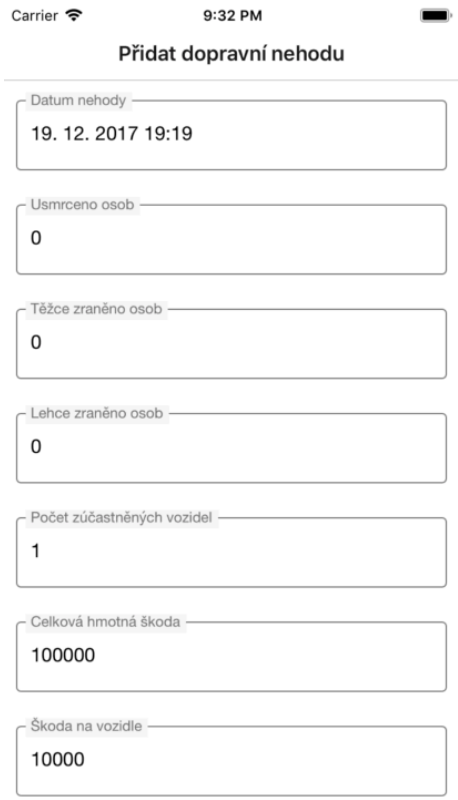


Obrázek C.2: Tabulka v administrační aplikaci

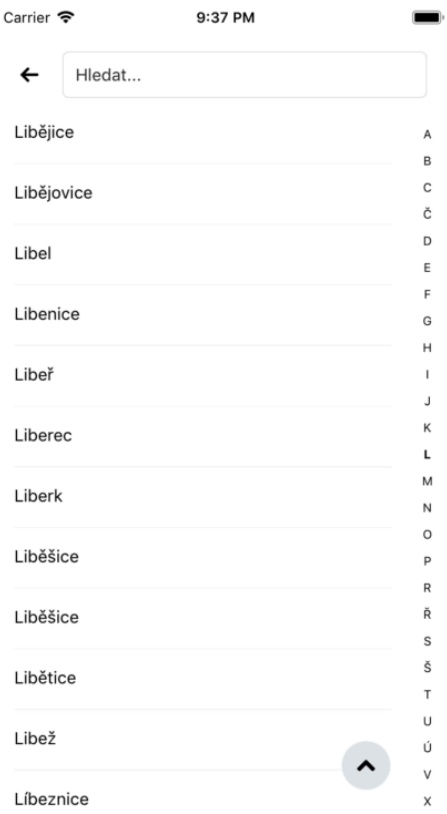
D GUI mobilní aplikace



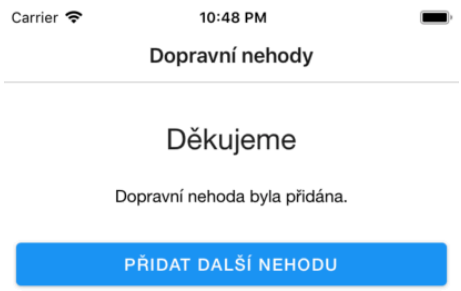
Obrázek D.1: Přihlášení do aplikace



Obrázek D.2: Zadávání dopravní nehody



Obrázek D.3: Výběr města



Obrázek D.4: Obrazovka s poděkováním