VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO ANALÝZU A VYHODNOCENÍ JÍZD AUTOŠKOLY

A SYSTEM FOR A DRIVING SCHOOL TRIP ANALYSIS AND EVALUATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MARTIN ŠOULÁK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/19420/2016/xsoula02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Šoulák Martin, Bc.**
Obor: Informační systémy
Téma: **Systém pro analýzu a vyhodnocení jízd autoškoly**
A System for a Driving School Trip Analysis and Evaluation
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s projektem DoAutoškoly.cz pro zvyšování kvality autoškol pomocí sdílení uživatelských zkušeností
2. Prostudujte dostupné systémy a nástroje pro ukládání geografických dat a jejich aplikační rozhraní.
3. Navrhněte databázi na straně serveru umožňující efektivní uložení GPS dat pořizovaných v reálném čase.
4. Navrhněte aplikaci pro analýzu nasbíraných GPS dat z cvičných jízd za účelem zkvalitnění informací v profilu autoškoly.
5. Po dohodě s vedoucím implementujte navrženou aplikaci na vhodné platformě.
6. Implementujte grafické zobrazení výsledků analýzy na WWW.
7. Proveďte testování aplikace a zhodnoťte dosažené výsledky.

Literatura:

- Gutmans, A., Rethans, D., Bakken, S.: Mistrovství v PHP 5, Computer Press, 2012
- Momjian, B.: PostgreSQL - Praktický průvodce, Computer Press, 2003
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 4

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Burget Radek, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Předmětem diplomové práce je návrh a realizace systému pro ukládání geografických dat v reálném čase získaných během cvičných jízd autoškoly a následné vyhodnocení jejich průběhu. Řešení této práce je přímým rozšířením rozvíjejícího se projektu DoAutoškoly.cz. Pro zavedení kontextu práce je věnována pozornost teoretickému úvodu do oblasti geografických dat a vhodných databázových systémů s nadstavbou prostorových operací. Text plynule přechází k návrhu předmětného systému, vysvětluje důvody využití dvou databázových systémů a popisuje postupy při implementačním řešení jednotlivých částí. Významnou oblastí z pohledu diplomové práce je specifikace metod pro vyhodnocení cvičných jízd a grafické zobrazení těchto výsledků v katalogu DoAutoškoly.cz. Závěr je věnován testování, zhodnocení výsledků a návrhům na další rozšíření.

Abstract

The objective of this master thesis is to design and develop a real-time storage system for geographic data from driving school trips. The system provides tools for analysis and evaluation of practice trips. This system is an extension of the DoAutoskoly.cz project which is described in the text. The next part contains an introduction to geographical data, spatial data and available databases with spatial extensions. The understanding to spatial databases is very important for the system design, an explanation of a solution for a database layer and implementation of major parts. Solution for a graphical view of the results and possible extensions of the system are described in the last part of this thesis.

Klíčová slova

projekt DoAutoškoly.cz, záznam geografických dat v reálném čase, vyhodnocení dat z prostorové databáze, NoSQL databáze, MongoDB, PostreSQL, PostGIS, Typescript

Keywords

project DoAutoskoly.cz, real-time record of geographic data, spatial databases and evaluation, NoSQL databases, MongoDB, PostreSQL, PostGIS, Typescript

Citace

ŠOULÁK, Martin. *Systém pro analýzu a vyhodnocení jízd autoškoly.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Burget Radek.

Systém pro analýzu a vyhodnocení jízd autoškoly

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vytvořil samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. Důležité informace k věcným požadavkům mi poskytl Ing. Jan Hrivnák, jako autor a vedoucí projektu DoAutoškoly.cz. Uvedl jsem všechny literární prameny, publikace a zdroje, ze kterých jsem čerpal.

.....
Martin Šoulák
15. května 2017

Poděkování

Rád bych poděkoval vedoucímu této práce Ing. Radku Burgetovi, Ph.D. za odborné vedení a vstřícné jednání během konzultací a řešení jednotlivých částí práce.

Obsah

1 Úvod **3**

2 Seznámení s projektem DoAutoškoly.cz **5**
2.1 Motivace a cíle projektu . . . 5
2.2 Struktura webového portálu projektu . . . 6
2.3 Použité technologie . . . 7

3 Geografická data **8**
3.1 Základní prostorové objekty . . . 8
3.1.1 Reprezentace prostorových dat . . . 9
3.2 Zdroje geografických dat . . . 10
3.3 Související pojmy . . . 10

4 Teorie prostorových databází **11**
4.1 Georelační algebra . . . 12
4.1.1 Datové typy . . . 12
4.1.2 Operátory . . . 13
4.2 Vybrané prostorové operace . . . 13

5 Prostorové databáze v prostředí webu **14**
5.1 Relační databáze . . . 14
5.1.1 Vybraní zástupci . . . 14
5.2 NoSQL databáze . . . 15
5.2.1 Rozdíly oproti relačním databázím . . . 16
5.2.2 Vybraní zástupci . . . 16

6 Návrh řešení aplikace **17**
6.1 Struktura navrhovaného systému . . . 17
6.2 Datový formát trasy . . . 19

7 Implementace serverové aplikace **21**
7.1 Použité technologie . . . 22
7.1.1 Typescript . . . 22
7.1.2 Node.js a Node package manager . . . 23
7.1.3 Mongoose ODM . . . 24
7.1.4 PG-promise . . . 24
7.1.5 Socket.io . . . 24
7.2 Architektura aplikace . . . 25

7.2.1 Obsluha vstupních požadavků . . . 26
7.2.2 Správa připojených klientů . . . 27
7.2.3 Přístup k databázím . . . 28
7.2.4 Periodická správa systému . . . 29
7.2.5 Logování stavů aplikace . . . 30
7.3 Prostředí vývoje systému . . . 30

8 Metody vyhodnocení prostorových informací 31
8.1 Schéma databáze . . . 32
8.2 Proces vložení nové trasy . . . 33
8.3 Vyhodnocení plynulosti jízdy . . . 34
8.4 Analýza frekvence opakovaných průjezdů . . . 36
8.4.1 Optimalizace bodů . . . 36

9 Aplikační rozhraní pro operační databázi 37
9.1 Architektura a struktura schémat . . . 38
9.2 Navázání spojení s databází . . . 39
9.3 Záznam trasy do databáze . . . 39
9.4 Získání uložených dat . . . 40
9.5 Správa uložených dat . . . 42
9.6 Sledování jízdy v reálném čase . . . 44

10 Grafické zobrazení výsledků 45
10.1 Použité technologie . . . 45
10.1.1 Leaflet . . . 45
10.1.2 Chartist.JS . . . 46
10.1.3 Webpack . . . 46
10.2 Aplikace pro zobrazení výsledků . . . 47
10.3 Grafické uživalelské rozhraní . . . 48

11 Testování a ověření korektnosti výsledků 52
11.1 Ověření korektnosti výsledků . . . 52
11.2 Výsledky testování . . . 53

12 Závěr 54

Literatura 55

Přílohy 57

A Obsah přiloženého CD 58

Kapitola 1

Úvod

Využívání moderních technologií v podobě chytrých telefonů je v aktuální době více než samozřejmostí. Pro každý typ chytrých telefonů existuje velká nabídka uživatelských aplikací různého zaměření. Rozšířením a lepší dostupností datových přenosů s připojením k internetu se tato nabídka ještě více rozrostla. Díky tomu lze výsledky operací mobilních aplikací využívat a analyzovat v reálném čase, byť tato příležitost není často efektivně využita. Jako konkrétní případ lze uvést aplikace využívající GPS modul pro zaznamenávání trasy. Potenciál hodnoty datové informace popisující určitý pohyb již částečně využily například aplikace pro sport či turistiku, které tyto záznamy vyhodnocují a uživateli poskytují zajímavé výsledky o nadmořské výšce, průměrné rychlosti nebo hodnoty celkových statistik. Informace tohoto typu však jsou pouze zájmového charakteru bez žádného dalšího hodnotného přínosu.

Předmětem diplomového projektu je přímé rozšíření portálu DoAutoškoly.cz[1], který se snaží podporovat zvyšování kvality autoškol s využitím sdílených zkušeností absolventů. Definování hlavní myšlenky autora a současný stav projektu jsou v tomto textu samozřejmě obsaženy, a to konkrétně v kapitole, která bezprostředně následuje za úvodním slovem. Právě na trhu autoškol se nabízí velká příležitost k využití a vyhodnocení reálných údajů z cvičných jízd, které mohou být při výběru autoškoly klíčové. Tato získaná data mohou nabídnout neomezenou analýzu odkrývající skutečnou úroveň cvičných jízd a tedy celé autoškoly, z čehož vyšla hlavní myšlenka této práce.

Motivace osobního přínosu k projektu DoAutoškoly.cz

V současné době je trh s nabídkami velmi přesycený a malé autoškoly často vědomě nedbají na kvalitu dovedností a znalostí absolventů s cílem menších nákladů a tedy vyššího zisku.

Rád bych proto touto prací poskytl nástroj pro zpřístupnění reálných údajů o autoškolách a především cvičných jízdách, neboť i z vlastní zkušenosti vím o situacích, kdy jsem jako žák autoškoly v rámci cvičné jízdy stál na parkovišti, než si instruktor nakoupil potraviny či vyřídil jiné osobní záležitosti. Další nevhodnou situací je trasování cvičné jízdy instruktorem bez jakéhokoliv zájmu se vyhnout přeplněným ulicím ve špičkách, kdy je hustota provozu nejvyšší. V takových situacích je předvídatelné, že v odpoledních hodinách nebude na hlavních městských tazích plynulý provoz. Během přiděleného času cvičné jízdě pak žák stráví nezanedbatelnou část popojížděním v koloně. Nekvalitní autoškoly pak tuto praktiku mohou vytvářet záměrně za cílem snížení spotřeby paliva vozidel, bohužel však na úkor času, za který by žák mohl získat při plynulé jízdě nové zkušenosti. Neplynulý provoz však

[1] URL adresa projektu: http://www.doautoskoly.cz

samozřejmě nelze vždy předvídat, neboť jsou často na vině dopravních nehody a jiné náhodné vlivy. Při dostatečném počtu vzorků projetých cvičných jízd a cíleném vyhodnocení se tyto výjimečné situace mohou stát zanedbatelnými a průměrnou dobu plynulé jízdy neovlivní. Cílovou skupinou jsou především zájemci o výuku autoškoly, kteří se z velké části pohybují v předmaturitním věku a výběr autoškoly řeší mezi vrstevníky a s rodiči. Právě v tomto okruhu se diskutuje i o absolvování zvláště důležitých dopravních situacích, které například na silnicích III. třídy nelze potkat.

Věřím, že tato práce má svůj význam ve prospěch motivace jednotlivých autoškol jako další prostředek pro zvýšení jejich kvality.

Je důležité zmínit, že současně s touto prací byla vyvíjena mobilní aplikace v rámci diplomové práce Bc. Jakuba Voneše, která má za cíl nabídnout uživateli možnost záznamu projeté trasy cvičné jízdy autoškoly v reálném čase a slouží tak jako majoritní zdroj dat pro systém navržený a implementovaný v této práci. Právě data z dokončených cvičných jízd mají velký potenciál jako množina unikátních údajů, které je možné cíleně vyhodnocovat, analyzovat a zpracovávat za již představeným účelem. Výsledky těchto operací jsou následně zpřístupněny v katalogu portálu DoAutoškoly.cz a rozšiřují tak portfolio autoškoly.

Mimo již zmíněné představení projektu DoAutoškoly.cz nabízí tento text strukturované uvedení do teoretických znalostí, jejichž osvojení je nutné pro správnou interpretaci terminologie v oblasti geografických dat a přiblížení obecných principů prostorových databází. Text se rovněž popisuje výčet zástupců databázových systému s nadstavbou pro uložení a zpracování prostorových dat, především v prostředí webu. Řešenou problematiku tato práce obsahuje v logicky oddělených kapitolách, které se postupně věnují návrhu systému, popisu implementačního řešení architektury serverové aplikace a aplikačnímu rozhraní pro zpřístupnění operační databáze. Samostatnou částí je oblast metod vyhodnocování prostorových dat a grafické zobrazení výsledků v katalogu DoAutoškoly.cz.

Kapitola 2

Seznámení s projektem DoAutoškoly.cz

V úvodu celé práce je nezbytné seznámení s projektem DoAutoškoly.cz, neboť následující text popisuje přímé rozšíření navazující na současný stav projektu. Ten byl realizován v roce 2016 v rámci diplomové práce *Zvyšování kvality autoškol pomocí sdílení uživatelských zkušeností* pana Ing. Jana Hrivnáka na Fakultě informačních technologií Vysokého učení technického v Brně. [22] Obsah této kapitoly zahrnuje nastudované poznatky ze struktury a cílů projektu, které jsou nutné pro návrh a implementaci rozšíření v podobě analýzy a vyhodnocení cvičných jízd autoškol.

2.1 Motivace a cíle projektu

Hlavní motivací pro vytvoření projektu byl pro Jana Hrivnáka velmi neuspokojivý přístup majitelů autoškol po celé České republice k výuce žáků, čímž často autoškoly produkují absolventy, coby velmi podprůměrné řidiče. Nezanedbatelná část majitelů autoškoly se dokonce netají, že obchodní strategie v boji s konkurencí má pro ně větší význam, než kvalita a řidičské schopnosti jejich absolventa a čerstvého držitele řidičského oprávnění jakéhokoliv ze skupin.

Cílem tedy bylo tento nevyhovující stav změnit webovým katalogem, který by na základě sdílených zkušeností absolventů poskytoval každé autoškole vlastní portfolio s hodnocením. Neméně důležitým krokem byla také nabídka podpory v podobě centrální webové služby pro samotné majitele a učitele autoškol. Jmenujme bodově nejdůležitější cíle z pohledu žáků a z pohledu autoškol, které Jan Hrivnák zmiňuje ve své práci:

1. Zvýšit povědomí veřejnosti, žáků a rodičů o rozdílu kvalitních a nekvalitních autoškol a zdůraznit důležitost úrovně teoretické výuky a především cvičných jízd.

2. Poskytnout podporu kvalitním autoškolám, které dbají na důležitost teoretické výuky a cvičných jízd vedením žáků ke správné defenzivní jízdě. Dále potom poukázat na případy, kdy autoškoly dostatečně nedbají na výuku poskytování první pomoci.

Vyšším cílem bylo tedy vybudování portálu, který by při úspěšném rozšíření mezi širokou veřejnost a autoškoly poskytoval globální informace o autoškolách a uživatelských zkušenostech s jednotlivými výukovými kurzy. Bezpochyby neméně důležitým cílem je i motivace vedoucích pracovníků autoškol k udržování kvalitní reprezentace profilu autoškoly v katalogu z pohledu aktuálnosti a přesnosti uvedených údajů. Jedná se tak o centralizované místo pro setkání nabídky kurzů autoškol s poptávkou ze strany potencionálních zákazníků, tedy žáků.

2.2 Struktura webového portálu projektu

Pro další vývoj rozšíření projektu je zapotřebí zmínit i realizovanou strukturu projektu, neboť především výstupy analýzy jízd autoškoly, které jsou předmětem této práce, musí korespondovat se zavedeným formátem projektu. Celý portál je vybudován na základě výsledků dotazování jednotlivých cílových skupin a použití nejmodernějších UX principů pro maximálně intuitivní a vhodné uživatelské rozhraní. Celý portál je zaměřen na 4 skupiny uživatelů, kteří do systému nezávisle na sobě vstupují. Jedná se o:

- uživatele, který anonymně vyhledává v katalogu,
- autentizovaného žáka autoškoly,
- správce profilu autoškoly,
- administrátora portálu.

Rozhraní žáka autoškoly

Po autentizaci žáků, kteří aktuálně absolvují libovolný kurz, je uživatelské rozhraní připraveno do dvousloupcové varianty, která nabízí přehledně vyobrazené informace o stavu a vývoji jejich kurzu. Lze předpokládat, že s postupným inkrementálním vývojem portálu budou jednotlivé nové funkčnosti a rozšíření řazeny dle priorit právě do těchto dvou sloupců. V současné době se zde nachází přehled procentuálního vyjádření splněné části celkového kurzu, ale i jednotlivé grafové prvky s výsledky výuky nebo cvičných jízd a další hodnotné informace z průběhu celého kurzu. Velmi užitečnou funkcí je plánování cvičných jízd studenta, což samozřejmě vyžaduje plné zapojení této administrativní činnosti ze strany autoškoly a instruktorů jízd.

Rozhraní správce profilu autoškoly

Jedná se o část portálu, kde jako správce profilu autoškoly vystupuje majitel, učitel či jiná pověřená osoba autoškoly. Počítačová a internetová gramotnost této skupiny lidí není očekávána na vysoké úrovni, proto je toto rozhraní navrženo co možná nejjednodušeji. Nejčastějšími úkony správce profilu autoškoly je editace údajů a informací, které jsou zobrazeny ve veřejném katalogu. Jako další důležitý prvek je nutné doplnit správu uživatelských účtů a přidělování různých úrovní práv dalším správcům pro editaci profilu autoškoly.

Katalogový profil autoškoly

Posledním zmíněným rozhraním portálu je samotný katalog, který nabízí v přehledném zobrazení detailní profily všech autoškol, možnost filtrování, seřazení autoškol dle jejich hodnocení od absolventů či jejich vzájemné srovnávání v nejdůležitějších parametrech. Především detail profilu autoškoly nabízí velké možnosti v dalším rozšíření o informace z vyhodnocených cvičných jízd a podobně.

2.3 Použité technologie

Představení a znalost technologií, které jsou využity pro implementaci celého projektu je z pohledu dalšího rozšíření a navázání na současný stav nezbytné. V následujících odstavcích této části textu jsou proto stručně popsány veškeré hlavní technologie, které jsou nutné pro vývojáře dalších rozšíření ovládat na velmi dobré úrovni.

Není překvapením, že pro klientskou část webových aplikací využil Jan Hrivnák kombinaci nejrozšířenějších technologií, tedy značkovací jazyk *HTML* ve verzi 5 a skriptovací jazyk *JavaScript* s využitím suverénně nejpoužívanější javascriptové knihovny *jQuery* pro usnadnění práce s objektovým modelem dokumentu (DOM). Byť pro jazyk *JavaScript* jsou v dnešní době dostupné i kvalitní frameworky různého zaměření, jako je například *AngularJS* či *React*, pro výstavbu klientské části tohoto portálu neměly žádný markantní potenciál. V budoucím rozšíření front-endu se možnost využití těchto nástrojů už nabízí více.

Serverová část webové aplikace je jádrem celého projektu a je implementována v jazyce *PHP*. Vzhledem k rozsahu logiky celého portálu bylo nutné využití objektově orientovaného přístupu společně s návrhovým vzorem *MVC (Model-View-Controller)*, který patří mezi velmi oblíbené, především díky svému jednoduchému řešení architektury pro odstínění práce s databázovou vrstvou, řídící logiky a uživatelského rozhraní.

Rozsáhlé aplikace si v dnešní době rovněž nepřímo říkají o implementaci využitím univerzálních *PHP* frameworků. Tento projekt není výjimkou a Jan Hrivnák pro jeho realizaci zvolil český framework *Nette*, především pro jeho rozsáhlou českou komunitu, která ho nadále rozvíjí. Mezi jeho hlavní výhody patří využití výše popsané architektury *MVC*, ale rovněž nabízí řešení zabezpečení proti chybám *XSS*, *CSRF* a podobně. Nelze opomenout velkou škálu pluginů, které lze do frameworku zapojit.

Jedinou databázovou službou, která je v současnosti v projektu využívána, je známé multiplatformní *MySQL*. Komunikaci s databází zajišťuje modul *Doctrine*, který je jedním z několika využitých pluginů v tomto projektu. Jedná se o framework pro mapování relačních závislostí databázového modelu na objekty a naopak. Práce s databází je tak usnadněna o sestavování dotazů *SQL*. V dalších rozšířeních je pomocí tohoto modulu možné pracovat i s nerelačními databázemi *NoSQL* či rovněž velmi rozšířeného *PostreSQL*.

Stručně lze doplnit i řešení bezpečnosti webové aplikace, což je v současné době velmi diskutované téma s ohledem na prevenci proti napadení aplikace či krádeže citlivých dat. Portál DoAutoškoly.cz využívá zabezpečeného přenosového protokolu *HTTPS*.

Kapitola 3

Geografická data

Základním stavebním kamenem všech systému, které se zabývají zpracováním a analýzou prostorových dat ve spojení s popisem zemského povrchu, jsou geografická data. Jedná se o specializovaný typ prostorových dat, které jsou vázané ke konkrétní poloze a obsahují další neprostorové atributy k danému bodu. [29]

Geografická data obsahují následující typy informací:

- *prostorové* – jedná se o aspekt popisující polohu objektu, která musí být jednoznačně určena. Tato poloha je definována souřadnicovým systémem, který je pro daný systém využíván a je nezbytně nutné, aby všechna související data byla určena v jednom konkrétním souřadnicovém systému,
- *popisné* – důležitou informací pro geografické data jsou atributy (metadata), které plní roli popisu unikátních vlastností konkrétního objektu. Tyto atributy mohou být strukturovány do tabulek atributů. Každý z atributů poté obsahuje hodnotu, kterou může běžně být: délka, plocha, obvod, nadmořská výška apod.,
- *časové* – podobně jako u popisné informace se jedná o neprostorovou složku dat. Geografická data jsou z pravidla vždy vztažena k časové informaci, která umožňuje zpětně analyzovat historický vývoj objektu nebo kategorizaci objektů v daném časovém intervalu či okamžiku.

Terminologická komise ČÚZK uvádí pro pojem *Geografická data* ekvivalentní zkrácený termín *geodata*, který je v české i zahraniční literatuře využíván mnohem častěji. [30] S tímto souvisí i termín *geoobjekt*, který reprezentuje základní prostorový objekt, který je atomický a tedy dále nedělitelný na prvky stejného či dokonce jiného typu.

3.1 Základní prostorové objekty

Nejčastějším výčtem základních prostorových objektů je trojice bod – linie – plocha. Tedy prvky, kterými lze kompozicí popsat jakýkoliv objekt reálného světa v geografickém systému: [21]

- *bod (angl. point)* – objekt, který nenese žádnou rozměrovou informaci nebo vzhledem k použitému měřítku nelze zobrazit jako plochu,

- *linie (angl. line)* – reprezentuje objekty, které jsou popsány pouze svojí délkou bez definované šířky. V praxi je to například silnice nebo vedení vysokého napětí,
- *plocha (angl. area)* – každý objekt je plochou, pokud je ohraničen uzavřenou linií. Jedná se tedy o tvary reprezentující vymezené území na zemském povrchu.

Zatímco předchozí dělení popisuje základní prostorové objekty spíše neformálně, podle [29] lze rovněž prostorové objekty kategorizovat matematicky podle jejich dimenze, tedy počtu rozměrů:

- *bezrozměrné* – body, které jsou definované pouze svojí polohou a nemají žádný rozměr popisující jejich velikost, délku nebo jinou rozměrovou vlastnost. Mluvíme tedy o konkrétních atomických bodech v prostoru označené souřadnicemi *[x,y]*.
- *jednorozměrné* – objekty, které mají konečnou délku a nulovou plochu. Jedná se tedy o linie zapsané posloupností souřadnic *[x,y]*.
- *dvojrozměrné* – do této kategorie spadají veškeré definice ploch, tedy objektů, které jsou ohraničeny alespoň třemi jednorozměrnými objekty, resp. jsou definovány uzavřenou posloupností souřadnic *[x,y]* s určením počátečního a koncového bodu.
- *trojrozměrné* – objekty, jenž jsou nejméně využívaný reprezentant v základních geografických systémech. Takové objekty jsou ohraničeny čtyřmi dvojrozměrnými objekty a mají tedy vlastnost objemu.

3.1.1 Reprezentace prostorových dat

V současné době lze digitální podobu prostorových dat reprezentovat dvěma způsoby: rastrově a vektorově. [24]

Rastrová reprezentace prostorových dat

Základem rastrové reprezentace je překrytí prostoru mřížkou neboli rastrem. Každý prvek v tomto prostoru je poté identifikován diskrétními hodnotami, které popisují pozici vůči rastru.

Tvar buněk tohoto rastru je vždy předem definovaný a mimo standardní čtvercový tvar lze využít i šestiúhelníkový či trojúhelníkový tvar. V praxi se však využívá nejčastěji mřížka čtverců s vazbou na kartézskou soustavu souřadnic pro určení polohy prvku.

Vektorová reprezentace prostorových dat

Hlavním předpokladem tohoto přístupu je spojitost prostoru, ve kterém se všechny prvky vyskytují. Vektorová reprezentace je založena na objektovém přístupu, kde každý objekt obsahuje identifikaci, popisující atributy a vztahy k ostatním objektům v prostoru. Výsledné vyjádření prostoru je možné realizovat vrstvami jednotlivých objektů nebo hierarchickým seskupením tříd objektů. Nejčastější využití vektorové reprezentace lze potkat v mnoha odvětvích Geografických informačních systémů (GIS) pro popis a analýzu prostoru.

3.2 Zdroje geografických dat

Získávání geografických dat z krajinné sféry je specifickou oblastí geografických informačních systému. Jako primární jsou označovaná data z terénních průzkumů, dálkového průzkumu země (letecké a družicové snímkování) a geodetického nebo statistického měření. Do této kategorie spadá i získání dat z Globálního pozičního systému (GPS), který je pro tuto práci velmi významný. Pojem GPS je stručně definován společně s ostatními pojmy na konci této kapitoly. Za sekundární geografická data jsou potom označovány případy získání informací o geografickém objektu z analogových nebo historických záznamů.

3.3 Související pojmy

- **Geoinformatika** – je interdisciplinární věda, která ve své oblasti spojuje geografii, kartografii, informatiku a geodézii. Z několika zaměření geoinformatického výzkumu můžeme vybrat například získávání digitálních geografických dat, globální polohové systémy, geografické informační systémy, simulační modely a vizualizace. [24]
- **GIS - Geografický informační systém** – *„Organizovaný soubor počítačového hardware, software, geografických údajů a personálu určený k efektivnímu sběru, uchovávání, obnovování, manipulaci, analýze a zobrazování všech geograficky vztažených informací; používaná zkratka je GIS.“* [15]
- **Dálkový průzkum Země** – je specializovaná oblast geoinformatiky, která obsahuje soubor metod a postupů pro získávání digitálních údajů a měření objektů reálného světa na zemském povrchu. Získávání dat probíhá v nadpovrchových vrstvách, tedy leteckým snímkováním nebo z družice. [16]
- **Souřadnicový systém** – jedná se o definici referenčního systému, k němuž jsou vztaženy veškeré výpočty v prostoru. Definování takového systému je nutné pro jednoznačné numerické určování polohy, délky nebo plochy v prostoru. [26]
- **WGS-84** – světově uznávaný globální souřadnicový systém, který je v civilním využití spojen především s technologií globálních pozičních a navigačních systémů známých pod zkratkou GPS. Systém WGS-84 je geocentrický pravoúhlý pravotočivý systém, kde tvar souřadnic vychází z desetinného zápisu běžných zeměpisných souřadnic, tedy zeměpisné délky a zeměpisné šířky. [14]
- **GPS – Globální poziční systém** – v originálním anglickém názvu *Global Positioning System*, je polohovací systém provozovaný Ministerstvem obrany USA, díky němuž je možné v jakémkoliv okamžiku určit polohu objektu na zemském povrchu. Systém GPS zahrnuje 24 navigačních družicových satelitů, které obíhají planetu Zemi a jsou vzájemně časově synchronizovány. Uživatelské zařízení s GPS přijímačem pasivně získává datovou informaci z družic, ze kterých probíhá výpočet mj. údajů o poloze v souřadnicovém systému WGS-84. Přesnost údajů je závislá na počtu družic, které jsou pro GPS zařízení v daném okamžiku na obloze dostupné. [26]
- **Formát GPX** – (angl. *The GPS eXchange format*) je standardizovaný formát geografických dat z GPS přístrojů. Jedná se o XML standard pro GPS zařízení, který byl zavedený v roce 2002. V roce 2004 byla vydána aktuálně používaná verze standardu GPX 1.1. [2]

Kapitola 4

Teorie prostorových databází

V předchozí kapitole jsou popsány různé typy datových informací, které jsou jednotně reprezentovány pojmem *Geografická data*. Jak již bylo uvedeno, zpracování těchto dat je především v roli Geografických informačních systémů (GIS), ale také specializovaných aplikací, které obsahují konkrétní specifické analýzy a operace. Nejenom pro tuto potřebu je nutné mít možnost geografická data a výsledky operací nad nimi uchovat pro budoucí zpřístupnění. Za tímto účelem je výhodné použít vhodný databázový systém s rozšířením podpory prostorových dat. V tomto rozšíření je zahrnuta především podpora využití geometrických objektů, integrace aplikovatelných geometrických funkcí a efektivní uložení fyzických dat. Zkráceně se tyto systémy nazývají *Prostorové databáze*. Podobně jako další typy databázových systémů podléhají i prostorové databáze obecnému přístupu Systému řízení báze dat (SŘBD). Tento termín je v anglickém jazyce známý jako Database Management System, zkráceně DBMS.

Database Management System (DMBS) je podle [27] kolekce programů, která univerzálně popisuje mezivrstvu mezi uloženými reálnými daty a aplikacemi, jenž databáze využívají. Vzhledem k velkému množství dat, které se v databázích nacházejí, je stanovení systému správy dat nezbytné především z důvodu kontroly přístupu. Jádrem každého DBMS jsou následující procesy:

1. *Definice databáze* jako specifikace struktury a datových typů jednotlivých položek a jejich omezení.
2. *Konstrukce databáze* probíhající uložením samotných fyzických dat do úložiště, které splňuje důležitou podmínku perzistence, tedy zachování trvanlivosti svého stavu v průběhu času.
3. *Manipulace s databází* zahrnující široké spektrum úkonů, které lze nad databázovým systémem provést.
4. *Dotazování*, které je důležitou sadou operací sloužící pro získání specifických dat z databáze na základě konkrétního vyjádření databázového dotazu.
5. *Změna databáze* nastávající při potřebě modifikace uložených dat.

Database Management System je rozdělen do dvou částí. První z nich je část pro zpracování uživatelských dotazů, tou druhou je poté samotný přístup k datové vrstvě uložených informací a k metadatům, které popisují definovanou strukturu databáze.

DBMS závisí na základní koncepci závislosti dat (*data independence*), což znamená, že uživatelé pracují s reprezentací dat nezávisle na jejich fyzickém uložení. DBMS tedy zajišťuje převedení manipulace s daty na straně uživatele do specifických a efektivních operací přímo ve fyzickém uložení datových struktur. V porovnání se souborovými systémy se tedy jedná o naprosto odlišnou strategii přístupu k datům.

Jako doplnění je vhodné uvést, že pro databázové systémy s relačním modelem dat se v literatuře používá označení Relational DBMS (RDBMS). V oblasti zpracování prostorových dat je prostor definován jednotlivými relacemi, ve kterých každý řádek reprezentuje geografický objekt a jednotlivé sloupce jeho atributy. [27]

Předtím, než budou zmíněny konkrétní varianty prostorových databázových systémů, je vhodné věnovat pozornost také popisu vlastností, které jsou pro tuto kategorii databází typické.

4.1 Georelační algebra

Algebry v prostorových databázích mají významnou roli pro matematickou a geometrickou definici prostorových datových typů a operací, které jsou nad nimi prováděny. Jedná se o důležitý nástroj, který je nezávislý na použitém DBMS a formálním popisem slouží pro pochopení operací a vztahů na elementární úrovni. Jednou ze základních algeber je georelační algebra, jejíž základní pojmy jsou uvedeny v následující části textu. Zdroj definovaných vztahů a rozsáhlejší popis algebry lze dohledat v článku Ralfa Hartmuta Gütinga [20].

4.1.1 Datové typy

Tak jako v obecných algebrách jsou i v georelační verzi využity základní datové typy, především pro popis hodnot atributů jednotlivých objektů a výsledky aplikace operátorů. V notaci georelační algebry jsou označeny jako `NUM` pro vyjádření číselné hodnoty, `STR` reprezentující řetězec znaků a `BOOL` nabývající logických hodnot `true` nebo `false`. Datové typy, jejichž definice je doplněna v georelační algebře jsou následující:

- `POINT` – hodnota tohoto typu je bod ve smyslu místa průsečíku souřadnic x a y, tedy bod $p = (x, y)$ kde x, y jsou hodnoty kartézského souřadného systému kolmých os,
- `LINE` – reprezentuje geometrický objekt, který je lineárním spojením dvou libovolných objektů `POINT`. Pokud koncový bod linie je současné počátečním jiné linie, jedná se o tzv. segment linie. Takový případ je nazýván *řetězem segmentů*. Řetěz je tzv. jednoduchý, pokud žádný z množiny bodů není koncovým bodem více než dvou segmentů,
- `PGON` – v případě, že řetěz liniových segmentů je uzavřen, tedy v grafové terminologii mají stupeň 2, je geometrický objekt nazván polygonem,
- `AREA` – stejně jako u typu `PGON` se jedná o uzavřený řetěz segmentů bez děr, avšak v případě typu `AREA` se jedná o průnik množiny polygonů. Tím je v praxi možné například efektivně popsat pozemky v katastrální mapě a podobně.

4.1.2 Operátory

Před zmíněním nejdůležitějších operátorů zahrnutých v georelační algebře, je nutné definovat i notaci `TYPE*`, která se často využívá a reprezentuje množinu objektů datového typu `TYPE`. Následující tabulka již formálně popisuje nejdůležitější třídy operátorů. Záměrně nejsou v tabulce uvedeny základní matematické a logické operátory, jejichž znalost se předpokládá. Stejně jako popis datových typů v předchozí části pochází notace zápisu operátorů z [20].

Operátor	**Výsledek**	**Slovní přepis operátoru**
$POINT \times POINT$	$BOOL$	rovnost / nerovnost
$LINE \times LINE$	$BOOL$	
$POINT \times POINT$	NUM	vzdálenost
$LINE$	NUM	délka
$AREA \times AREA$	$BOOL$	sousedí s
$LINE* \times LINE*$	$POINT*$	množina bodů protnutí (průsečíků)
$LINE* \times REG*$	$LINE*$	množina linií protnutí
$AREA* \times AREA*$	$AREA*$	překrytí

Tabulka 4.1: Nejdůležitější třídy operátorů [20]

4.2 Vybrané prostorové operace

Pro tuto práci není důležité rozebírat veškeré netriviální prostorové operace, které vycházejí z georelační algebry. Zmíněna bude pouze ta, která má pro práci a její výsledky zásadní význam.

Intersection

Jedním z nejdůležitějších operátorů pro tuto práci je protnutí neboli průsečík linií (angl. *intersection*). Zápis této operace dle Georelační algebry je následující:

$$LINE* \times LINE* \rightarrow POINT* \tag{4.1}$$

Tento operátor je aplikován na dvě množiny linií, kde výsledkem zpracování operace jsou všechny body průniku typu `POINT` mezi liniemi prvního a druhého operandu. Je důležité zmínit, že vybraná linie náležící do množiny z prvního operandu může vytvořit průsečík s jednou konkrétní linií z druhého operandu na několika místech. [20]

Častým omylem z důvodu podobnosti termínů bývá záměna s operátorem testu existence průsečíku (angl. *intersects*), jehož výsledkem je pouze logická hodnota s významem, zda pro oba operandy existuje společné místo průniku či nikoliv.

Kapitola 5

Prostorové databáze v prostředí webu

Velmi důležitou částí této práce je řešení principu uložení, správy a analýzy prostorových dat. V předchozím textu byla tato problematika rozebírána především formálně. Pro další vývoj řešení je však nutné prostudovat dostupné databázové systémy, které nabízejí podporu zpracování prostorových dat. S ohledem na fakt, že projekt DoAutoškoly.cz je webovou službou, je nutné prostorové databáze zkoumat se zaměřením pro toto webové využití. Za důležitý faktor studia dostupných databázových systémů lze považovat i potřeba nekomerčního řešení. V současné době však tento požadavek není nikterak limitující, neboť v nekomerčním prostředí se vyskytuje velké množství DBMS s různorodou škálou zaměření. Veškeré systémy pro realizaci databázových vrstev v prostředí webu lze rozdělit na skupinu *relačních databází* a skupinu *NoSQL databází*.

5.1 Relační databáze

Základní princip relačních databází není třeba důkladně popisovat, neboť se jedná o velmi rozšířené principy databázového přístupu, které patří k základním znalostem každého vývojáře. Pro čtenáře tohoto textu, kteří se v této oblasti nepohybují, lze *relační model* představit jako kolekci logicky oddělených tabulek, v jejichž řádcích jsou uloženy jednotlivé záznamy. Název těchto databází napovídá jistou souvislost s pojmem matematické relace, což je vztah mezi dvěma množinami. V prostředí databází je možné tyto vztahy definovat pomocí primárních a cizích klíčů.

Nemalé množství relačních databází poskytuje prostorovou nadstavbu, která k základním datovým typům nabízí navíc prostorové. V drobných odchylkách v závislosti na typu DBMS jsou to především typy `POINT`, `LINESTRING`, `POLYGON`, `MULTIPOINT`, `MULTILINESTRING` a další. Nejvýznamnější představitele těchto databází představuje následující text.

5.1.1 Vybraní zástupci

Škálu prostorových funkcí aplikovatelných na data nabízí řada relačních databázových systémů. V tomto textu je důležité zmínit ty reprezentanty, které mají s ohledem na využitelnost při implementaci vyhodnocení dat v rámci této práce největší potenciál.

- **MySQL** je bezkonkurenčně nejrozšířenější databázový multiplatformní systém pro dotazování pomocí jazyka SQL pro široký záběr využití. Současné verze obsahují vhodné rozšíření pro prostorová data, které je navrženo dle specifikace *OGC (Open Geospatial Consortium)*. Pod tímto názvem vystupuje mezinárodní standardizační organizace zabývající se procesem návrhu a implementace standardů pro geodata. Ve starších verzích bylo prostorové rozšíření dostupné pouze pro typovou definici tabulek *MyISAM*. Aktuálně je však možné rozšíření prostorových datových typů a funkcí využít například i pro známý typ *InnoDB*. Označení současné verze je *MySQL 5.7*. [3]
- **PostgreSQL** v kombinaci s prostorovou nástavbou **PostGIS** je pravděpodobně nejznámější nekomerční databázových systém pro správu, uložení a zpracování prostorových dat. Stejně jako u *MySQL* se využívá popis geografických prvků dle norem konsorcia *OGC*, čímž je zajištěna standardizovaná reprezentace geodat. Prostorové funkce průniků, analýzy překryvů či vzdálenosti jednotlivých prostorových objektů jsou zahrnuty v rámci základní sady běžně dostupných operací v rámci tohoto systému. Nabídku výhod rozšiřuje i možnost využití prostorových indexů *R-tree* neboli *R-stromu*. Velkým přínosem pro využití rozšíření *PostGIS* je také velmi rozsáhlá základna vývojářů, kteří se věnují vývoji různých geografických informačních systémů a mapových portálů využitím právě databáze *PostgreSQL*. Online dokumentace jsou dostupné z [7, 6].
- **SQLite** je posledním zástupcem, který bude zmíněn. Jedná se o řešení alternativního přístupu k předchozím, neboť se jedná pouze o knihovnu, která je připojena k aplikaci. Celá databáze je potom uložena v jediném souboru formátu *DBM* a díky své malé robustnosti je její využití velmi výhodné v situacích, kde není možné využít serverovou aplikaci nebo její využití je zbytečné. Pro prostorový přístup je dostupné rozšíření **SpatiaLite**, které umožňuje správu geografických dat podobně jako *PostGIS*. Nabídka složitějších prostorových operací však neumožňuje rozsáhlé zpracování geodat a proto je využití výhodné především pro uložení statických údajů, popřípadě multiplatformní transport dat v jediném souboru. [9]

5.2 NoSQL databáze

Pro termín *NoSQL* v současné době stále neexistuje konkrétní definice, která by mohla být citována pro vysvětlení těchto technologií. Ve skutečnosti se principy *NoSQL* databází popisují výčtem vlastností a také odlišností od databází relačních. Základní představení poskytuje i význam zkratky *NoSQL*, která je často nesprávně interpretovaná ve smyslu *„No SQL – žádné SQL“*, tedy striktní postavení proti *SQL* databázím. Ve správném výkladu se však jedná o *„Not only SQL“*, tedy alternativní databáze k relačnímu přístupu. [18]

Ve výčtu nejdůležitějších vlastností *NoSQL* je společným znakem nabídnutí možnosti práce s databází v jiném pojetí, než využitím známých principů relačních databází. Z toho plyne, že data v *NoSQL* databázích nemají žádnou pevně strukturu, která by byla nutně předem známá. Tato vlastnost je velmi užitečná v situaci, kdy data uložená v databázi nemají stejný počet atributů, byť spolu logicky souvisí a v relačním modelu dat by náležely do jedné relace. V praxi tedy může nastat situace, kdy k jedné jediné položce jsme nuceni přidat unikátní atribut, který je pro ostatní nevýznamný. V *NoSQL* proto může mít každý prvek v databázi jiný počet a typ atributů, aniž by ovlivnil další existující data, která jsou obsažena v související datové struktuře. Nelze opomenout vlastnost vertikální a horizontální škálovatelnosti, specifický přístup k indexování a možnost jednoduché replikace dat.

5.2.1 Rozdíly oproti relačním databázím

V předchozím textu byly nastíněny vlastnosti, kterými se vyznačují *NoSQL* databáze. Pokud bychom tedy chtěli jmenovat konkrétní rozdíly mezi relačním modelem dat a nerelační strukturou *NoSQL* databází, mohli bychom většinu z předních vlastností *NoSQL* zopakovat, neboť tento databázový přístup vznikl především z důvodu potřeby databází, které nejsou striktně svázány svým schématem. Inspirací bylo rovněž doplnění chybějících operací. Další velkým rozdílem je možnost zanořování datových struktur v *NoSQL*, které v určitých případech mohou velmi efektivně vynahrazovat konstrukci výkonově náročných spojovacích dotazů s klauzulí `JOIN`, bez kterých se relační databáze téměř neobejdou.

Významným znakem *NoSQL* databází, který ještě v tomto textu nebyl zmíněn, je jejich textový formát, ve kterém jsou data uloženy. V případě dokumentově orientovaných *NoSQL* databází jsou formátem tohoto textového dokumentu často varianty *JSON* nebo varianty *XML*, což v případě velkého množství dat může způsobit jistou míru redundantních dat, neboť názvy atributů ve strukturách a kolekcích se pro každý záznam opakují.

Na rozdíl od relačních databází ovšem mají *NoSQL* databáze výhodu v efektivitě práce s databázovým serverem, neboť v případě relačních databází je vždy poskytována celá škála funkčnosti RDBMS, které ve výsledku nemusí být vůbec využity. *NoSQL* databáze je proto vhodné použít v takových případech, kdy rychlost zpracování je velmi důležitá, data jsou v jednoduché formě a není potřebné využití složitých operací nad relačním modelem dat.

5.2.2 Vybraní zástupci

V současné době již existuje více než 200 různých *NoSQL* databází. [19] Pro nás jsou však nejdůležitější zástupci, kteří poskytují prostorové rozšíření a práci s geografickými daty.

- **MongoDB** je pravděpodobně nejrozšířenější z vybraných zástupců. Jedná se dokumentově orientovanou open-source *NoSQL* databázi. *MongoDB* umožňuje libovolné zanoření dokumentů bez předem dané struktury. Každý je opatřen identifikátorem `_id`, jehož unikátnost je zajištěna na úrovni databázového systému. Všechny dokumenty jsou popsány v textovém formátu *JSON*, jehož interní reprezentace je binárně zakódovaná do formátu *BSON*. Dokumentace je online dostupná z [10].

  V této části textu je možné prozradit, že právě *MongoDB* byla zvolena jako technologie pro realizaci databázové vrstvy pro uložení real-time geografických dat v této práci.

- **GeoCouch** je prostorové rozšíření pro dokumentově orientovanou databázi *CouchDB*. Stejně jako u *MongoDB* je tato technologie open-source a data jsou reprezentována textovým formátem *JSON*. Právě rozšíření *GeoCouch* přidává k základní verzi možnosti prostorové indexace. Dokumentace této technologie je online přístupná z [1].

- **Neo4j** je nejznámější zástupce grafových *NoSQL* databází. Hlavní myšlenkou těchto databází je pojetí databáze jako grafu, který obsahuje uzly s atributy a vzájemným provázáním. Prostorové rozšíření *Neo4j* obsahuje podporu všech základních geometrických typů a geometrických operací. Databáze je implementována a navržena pro použití v jazyce *Java*. [4, 11]

Kapitola 6

Návrh řešení aplikace

Úvodní kapitola této práce stručně naznačila cíl práce, který je nyní nutné podrobně dekomponovat na jednotlivé problémy a navrhnout k nim nejvhodnější řešení. Pro lepší zasazení této kapitoly do obecného kontextu práce je vhodné zopakovat, že hlavní motivací bylo navrhnout a implementovat databázovou vrstvu pro real-time záznam trasy a následně nad těmito trasami na úrovni databázového serveru provádět jednotlivé analýzy a kroky vyhodnocení.

Prvotním základním řešením, které se nabízelo, byla jediná aplikace poskytující funkcionalitu pro záznam trasy v reálném čase i kompletní vyhodnocení jízd současně. Při bližším pohledu je zřejmé, že toto řešení by bylo nevyhovující, především s přihlédnutím na dva faktory. Databázový systém využitý pro analýzu a vyhodnocení jízd bude zpracovávat výkonově náročné dotazové operace a je nutná odpovídající robustnost a izolovanost takového systému, neboť bude pracovat nad velkým množstvím datových údajů. V některých případech se může jednat i o výpočty, které budou prováděny s pravidelným časovým intervalem na pozadí aplikace. Naproti tomu zápis jednotlivých bodů trasy v reálném čase bude realizovat mobilní aplikace, pro kterou je klíčová rychlá komunikace a spolehlivost přenosu dat. V tomto případě jsou hodnotou přenášených dat jednotlivé body z GPS zařízení mobilního telefonu, které ihned po získání jsou odeslány pro zpracování a uložení v databázi.

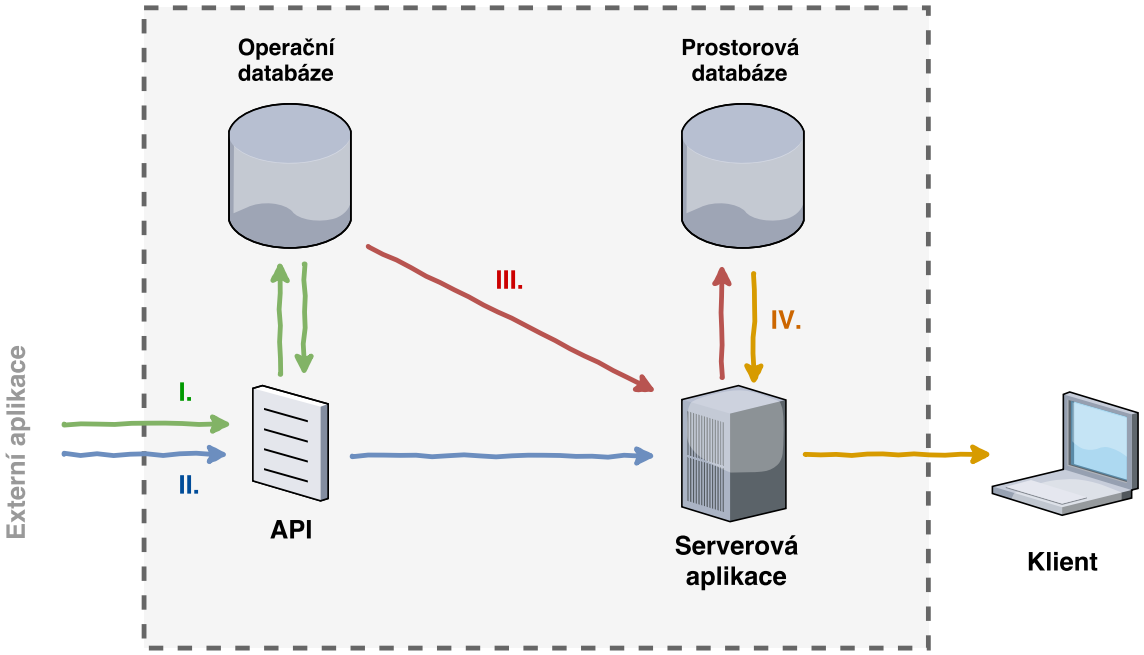
Z toho vyplývá nutná potřeba oddělení těchto dvou částí a tedy nezbytného návrhu komunikačního rozhraní mezi nimi. V následujících částech této kapitoly se objeví návrh struktury systému a podrobnější rozbor významných částí.

6.1 Struktura navrhovaného systému

Návrh struktury tohoto systému lze lépe popsat se schématem z obrázku 6.1, které za pomoci využití orientovaných spojení lépe představuje tok informací a komunikaci v navržené aplikaci mezi jednotlivými částmi. Z pohledu řízení toku dat se v systému nachází aplikační rozhraní (API) pro umožnění záznamu trasy v reálném čase externí službou do operační databáze a serverová aplikace, jejíž činnost řídí přenos zaznamenané trasy do prostorové databáze pro následné vyhodnocení průběhu.

Je samozřejmostí, že v reálném využití bude v aktuálním čase neomezený počet aktivních a vzájemně nezávislých fází systému. Tedy několik současných izolovaných záznamů jízd současně s dostupným vyhodnocením již dokončených tras. Kompletní vývoj

stavů systému v čase je pro názornost popsán pro jediný konkrétní případ a je rozdělen do následujících čtyř fází, které jsou na obrázku 6.1 barevně odlišeny.



Obrázek 6.1: Schéma navrhovaného systému

Fáze I. – Záznam trasy v reálném čase

Tato vstupní úvodní fáze, která je označena zelenou barvou, reprezentuje využití aplikačního rozhraní (API) externí aplikací, které poskytuje možnost vytvoření nové trasy při začátku záznamu a následné ukládání projetých bodů v reálném čase. Body jsou reprezentovány zeměpisnými souřadnicemi a dále doplněny především o časový údaj získání bodu GPS zařízením.

Fáze II. – Dokončení záznamu trasy

Akce spojená se skončením cvičné jízdy autoškoly je ve schématu zvýrazněna modře. V této situaci externí aplikace využívá funkčnost aplikačního rozhraní pro ukončení trasy, které tuto informaci deleguje do serverové aplikace. Současně tato situace může znamenat i zprávu z externí aplikace o dokončené editaci konkrétní trasy. Tyto informace serverová aplikace zpracuje v rámci následující třetí fáze.

Fáze III. – Přenesení trasy mezi databázemi

Červená barva reprezentuje reakci systému, která je vyvolána událostmi z předchozí fáze. V závislosti na situaci provádí serverová aplikace přenos dokončené nebo editované trasy do prostorové databáze pro následné vyhodnocení. V této chvíli je důležitá kontrola kompletnosti a správnosti formátu dat popisujících dokončenou trasu.

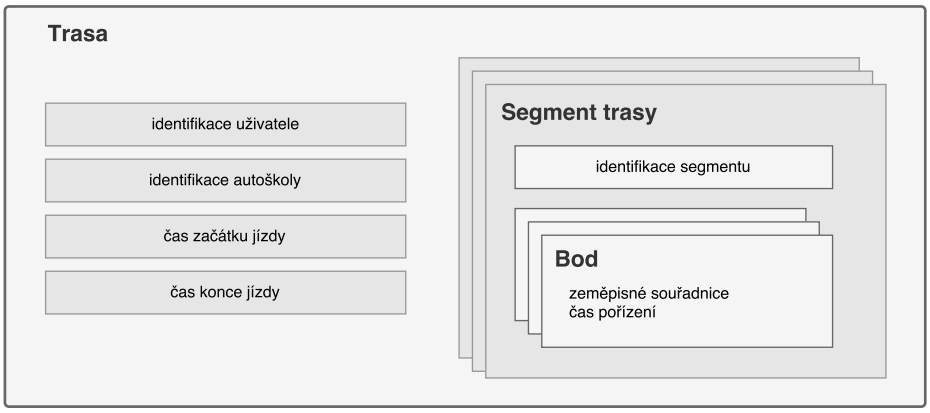
Fáze IV. – Poskytnutí vyhodnocení klientské straně

Závěrečná „žlutá“ fáze už není přímo navazující k předchozím, neboť obsahuje především vyhodnocení uložených tras v databázi. Tato analýza může probíhat až po požadavku z klientské strany, kde je výsledek zobrazen ve webovém rozhraní portálu DoAutoškoly.cz. Složitější operace pro vyhodnocení mohou probíhat v pravidelných intervalech nezávisle na aktuálním chodu systému. Uživateli jsou potom poskytnuty poslední kompletní získané výsledky.

6.2 Datový formát trasy

V jednotlivých fázích systému se pracuje se záznamem trasy, který ze zřejmých důvodů musí mít předem definovanou strukturu pro formát těchto dat. Ilustrační znázornění datového formátu trasy na obrázku 6.2 popisuje obsah tohoto záznamu.

Mimo atomické údaje reprezentující identifikaci uživatele, autoškoly a časové údaje obsahuje především kolekci segmentů trasy. Tyto segmenty zastupují posloupnost libovolně rozsáhlých částí trasy a obsahují svojí identifikaci a případně další položky informativního charakteru. Mnohem větší významová důležitost náleží sadě samotných bodů, které nesou informaci o zeměpisných souřadnicích a času pořízení GPS zařízením. Z této množiny bodů je potom celá trasa sestavena pro následnou analýzu.



Obrázek 6.2: Datový formát trasy

Serializace trasy

K přenosu trasy v rámci navrhovaného systému je nutné využít serializaci do textového formátu. Bezpochyby se nabízí běžně využívané standardy *XML* a *JSON*. Jelikož je již v textu dříve uvedeno, že pro řešení operační databáze je využita databáze *MongoDB*, která data ukládá právě v *JSON*, nebyl jediný důvod tento formát serializace dat nevyužít.

Pro oblast geografických dat dokonce existuje speciálně připravený formát *GeoJSON*, který je popsaný v normě `RFC 7946` a nabízí připravené datové typy pro popis prostorových objektů. Byť se užití v této práci zdá být přínosné, formát *GeoJSON* není využitý z následujících důvodů, které pro tuto práci vyhodnocují formát jako nevhodný:

1. *redundance dat* – jelikož každý typ datové struktury dle zmíněného standardu obsahuje jako jeden z atributů svůj název, je zřejmé, že využití ve velkém množství stejných typů přináší výraznou redundanci. Pro účely tohoto projektu se jedná především o datovou strukturu `Point`, která ve své definici obsahuje atributy `"type": "Point"` a `"coordinates": [Number, Number]`. Atribut `type` je naprosto zbytečný, neboť pro celou cvičnou jízdu autoškoly, která může obsahovat několik stovek až tisíců bodů, přináší nezanedbatelnou položku nadbytečných dat, které by bylo nutné přenášet, aniž by nesly významově důležitou hodnotu.

2. *nekomplexnost* – formát *GeoJSON* ve své definici neposkytuje komplexnější definici. Dodržování normy by vedlo ke zbytečnému zanořování datové struktury, neboť pro tento projekt je ke každému bodu trasy nezbytné doplňovat i čas pořízení a další atributy, které v typu `Point` formátu *GeoJSON* nejsou obsaženy.

3. *nejednoznačnost* – v prvním jmenovaném bodu nevýhod byla zmíněna reprezentace souřadnic bodu v atributu `coordinates` jako kolekce dvou číselných hodnot. Na první pohled však není z tohoto popisu zřejmé, která ze souřadnic popisuje zeměpisnou šířku, resp. zeměpisnou délku. Byť existují normy pro pořadí těchto hodnot, toto vyjádření je velmi nejednoznačné a bezpochyby lepším řešením je využití popisu jednotlivými atributy pro tyto dvě hodnoty.

Po shrnutí těchto nevýhod formátu *GeoJSON* lze říci, že serializace v řešeném systému je částečně ovlivněna i tímto geografickým formátem, ale reálně je využit obecný formát *JSON*.

Kapitola 7

Implementace serverové aplikace

V návaznosti na předchozí kapitolu obsahující návrh systému je zřejmé, že právě serverová aplikace je nejdůležitějším a řídícím členem celého systému pro postupný proces uložení a následné vyhodnocení tras. Rovněž se jedná o prvek reagující na veškeré vstupní požadavky z klientské strany portálu DoAutoškoly.cz na získání výsledných vyhodnocených údajů. Poskytnutí výsledků však předchází řada netriviálních operací při zpracování dat z externí aplikace a jejich postupná analýza.

Z implementačního pohledu serverová aplikace představuje program napsaný v jazyce *Typescript*, který je spuštěn v serverovém prostředí. V tradičním postavení komunikace typu klient-server zpracovává a odpovídá na vstupní požadavky. Do reálného systému z klientské strany vstupují dva typy uživatelů. První zastupuje návštěvníka katalogu DoAutoškoly.cz a druhý je připraven pro budoucí potřeby pozice administrátora portálu. Běžný uživatel při načtení katalogového profilu konkrétní autoškoly ze strany serveru získá informace o vyhodnocení jízd, které jsou graficky v profilu zobrazeny.

Specifickým případem klienta v komunikaci se serverem je externí aplikace, která poskytuje informaci o právě skončené trase, čímž oznamuje dostupnost geografických dat z průběhu trasy v operační databázi.

Tímto stručným popisem však nelze postihnout a reprezentovat veškeré implementační postupy, a proto jsou v této kapitole představeny podrobněji použité technologie a následně i samotné řešení v průběhu vývoje celé serverové aplikace.

Obrázek 7.1: Loga hlavních implementačních technologií [5, 8, 12]

7.1 Použité technologie

Veškeré technologie použité k vývoji serverové aplikace pro vyhodnocení cvičných jízd souvisí s možnostmi využití klientské formy javascriptového kódu na serverové straně. Díky rozsáhlé komunitě, která v open-source licencích vyvíjí knihovny podporující například i komunikaci s databázovými vrstvami, je možné vystavět velmi komplexní aplikace. V konkrétním případě této práce se jedná o využité databáze *MongoDB* pro záznam trasy v reálném čase a relačního databázového systému *PostreSQL* s prostorovým rozšířením *PostGIS*. Výčet konkrétních verzí klíčových technologií použitých během vývoje je uvedený v tabulce 7.1.

Název	**Typ**	**Verze**
Typescript	implementační jazyk	v2.1.0
Node.js	framework	v6.2.1
npm	správce balíčků (package manager)	v3.9.3
socket.io	knihovna	v1.5.0
Mongoose ODM	knihovna	v4.6.4
PG-promise	knihovna	v5.5.0

Tabulka 7.1: Verze použitých technologií

7.1.1 Typescript

Jak již bylo předesláno, hlavním implementačním jazykem celé serverové aplikace je jazyk *Typescript*, který je nadmnožinou standartního klientského jazyka *Javascript*. Je to open-source programovací jazyk vyvíjený společností Microsoft, který *Javascript* rozšiřuje o zásadní implementační možnosti. Především přináší silnou typovost a objektově orientovaný přístup k návrhu systému. V současném trendu a velkém nárůstu množství javascriptového kódu v rozsáhlých aplikací se proto přímo nabízí využití objektového návrhu. *Typescript* ve svém rozsahu nabízí mimo jiné standartní definování tříd, dědičnost, generické datové typy nebo implementaci rozhraní.

Zdrojový kód v jazyce *Typescript* je transpilován do jazyka *Javascript*. Nejedná se tak o klasickou kompilaci nebo interpretaci zdrojového kódu, ale pouze o transformaci. Dále už je javascriptový kód běžným způsobem interpretován v klientských prohlížečích. Díky této transpilaci je například možné v metodách typescriptových tříd používat standartní javascriptové konstrukce a knihovny, u kterých je však nutné dbát na správné dodržení kontextu a rámce rozsahu dostupnosti proměnných a funkcí (angl. *scope*).

S využitím tohoto jazyka úzce souvisí modulové specifikace *CommonJS* a *AMD*, které umožňují definovat použití externích modulů ve vyvíjené aplikaci. Těmito standardy lze dosáhnout nejenom efektivnějšího vývoje, ale i zvýšení přehlednosti zdrojového kódu. Rozdílem mezi zmíněnými specifikacemi je především způsob načítání modulů. Zatímco u *CommonJS* probíhá načítání synchronně a je výhodnější pro využití u aplikací běžících na serveru, u *AMD* je způsob definice modulů obsažen přímo ve významu písmen v názvu AMD *(Asynchronous Module Definition)*. Díky asynchronnímu průběhu je především

v klientských aplikacích možné zoptimalizovat a zrychlit načítání zdrojových skriptů. V praxi to znamená načítání modulů javascriptového kódu až ve chvíli, kdy jsou nutné pro běh celé aplikace nebo její části. Při implementaci předmětné serverové aplikace se však nabízí využití specifikace *CommonJS*, zejména díky úzké souvislosti s javascriptovým frameworkem *Node.js*, kterému se tento text věnuje v následujícím bloku.

Jelikož tento jazyk není tak rozšířený a v následujícím textu se objevují ukázky zdrojového kódu, je vhodné představit základní syntaktické konstrukce, které jsou pro *Typescript* typické. Oproti jiným jazykům je zde odlišnost v definování datového typu proměnné, který je uveden vždy až za názvem proměnné, od kterého jej odděluje znak dvojtečky. Při deklaraci proměnné využívá klíčové slovo `let`. Jako demonstraci můžeme uvést například zápis deklarace číselné proměnné: `let pocet: number;`. Stejně tak je tato syntaxe využita při definici metod a jejich parametrů, kde datový typ návratové hodnoty je uveden až za samotnou hlavičkou metody. Tedy například: `public addMessage(msg: string):void`. Základními datovými typy, které *Typescript* nabízí, jsou `number`, `string` a `boolean`. Konstrukce související s objektově orientovaným zápisem, předpisem tříd, vytváření instancí nebo volání metod se už příliš neliší například od syntaxe jazyku *Java*, stejně tak jako řídící konstrukce, komentáře a další bloky zdrojového kódu.

7.1.2 Node.js a Node package manager

V souvislosti s technologií *Node.js* hovoříme o běhovém, událostmi řízeném frameworku, který umožňuje vývoj a využití javascriptových aplikací v prostředí serveru. Tento framework poskytuje neblokující princip zpracování operací, které v hlavní smyčce po detekci událostí provádějí odpovídající obslužné procesy.

Právě k *Node.js* neodmyslitelně patří správce balíčků *npm*, který přináší zvýšení efektivity a rychlosti vývoje. Zkratka pochází z anglického názvu *Node package manager*, který sám o sobě vypovídá o významu. Cílem komunity kolem *npm* je poskytnutí nástrojů a služeb v podobě modulů (balíčků) pro volné vývojářské využití. Každá aplikace v kořenovém adresáři obsahuje konfigurační soubor s názvem `package.json`, který obsahuje ve stanoveném formátu mimo jiné definici všech balíčků a jejich verzí pro danou aplikaci. Právě tyto jmenované balíčky včetně jejich závislostí jsou příkazem `npm install` s příslušnými parametry nainstalovány do adresáře `\node_modules` ve struktuře konkrétní aplikace. Je důležité zmínit, že balíčky podléhající frameworku *Node.js* respektují specifikaci modulů *CommonJS*.

V následujícím seznamu jsou jmenovány další balíčky, které nejsou explicitně v textu zmíněny a jsou využité při implementaci předmětné serverové aplikace:

- `express` – minimalistický framework pro vytvoření serverového prostředí,
- `node-schedule` – plánovač úloh,
- `os` – modul pro zjištění informací o serveru,
- `http` – modul pro podporu využití protokolu *HTTP*,
- `moment` – modul nabízející podporu při práci s aktuálním časem.

7.1.3 Mongoose ODM

Volba dokumentové databáze *MongoDB* pro využití při ukládání záznamu trasy cvičné jízdy v reálném čase již byla zmíněna. Nyní je důležité představit způsob připojení a komunikace s touto databází. V této souvislosti je nutné zavést termín ODM (*Object Document Mapper*), který transparentně mapuje dokumenty *NoSQL* databáze, v našem případě *MongoDB*, na objektové reprezentanty pro pohodlnější zpracování.

Z dostupných řešení ODM pro databázi *MongoDB* padla volba na hojně rozšířenou knihovnu *Mongoose*, která zavádí převod dokumentů na objektová schémata, které tyto data reprezentují a popisují jejich strukturu a datové typy. Umožňují tak k datům elegantně a především efektivně přistupovat. K využití jsou v nabídce této knihovny základní CRUD operace, ale i složitější konstrukce pro sestavení netriviálních dotazů.

Mongoose v této práci bylo využito především pro vývoj aplikačního rozhraní pro zpřístupnění operační databáze na straně externí aplikace, která v reálném čase zaznamenává průběh cvičné jízdy do připravených dokumentových schémat. Realizaci tohoto aplikačního rozhraní se dále podrobně věnuje kapitola 9, která popisuje i netriviální využití této knihovny v této práci.

Využití si však tato knihovna našla i v serverové aplikaci, kde stejně jako v aplikačním rozhraní umožňuje práci s databází *MongoDB*. V tomto případě v situaci získání záznamu trasy cvičné jízdy pro přenos do prostorové databáze k dalšímu zpracování.

7.1.4 PG-promise

Podobnou službu jako plní *Mongoose* z předchozího odstavce, nabízí knihovna *PG-promise* pro relační databázi *PostgreSQL*. Jedná se o neblokujícího klienta pro připojení k databázi pro *Node.js* aplikaci, který nabízí automatické udržování spojení s databází, flexibilní formátování a sestavování SQL dotazů. Velkou předností je rovněž řízení databázových transakcí a úloh nebo obsluha chybových stavů při provádění dotazů, případně při nedostupnosti připojení k databázovému serveru. Právě oblast správy transakčních dotazů byla během vývoje často využívána, zejména u sekvence dotazů pro korektní přenesení a vložení geografických informací o proběhlých cvičných jízdách nebo pro postupné vyhodnocení a analýzy těchto dat.

7.1.5 Socket.io

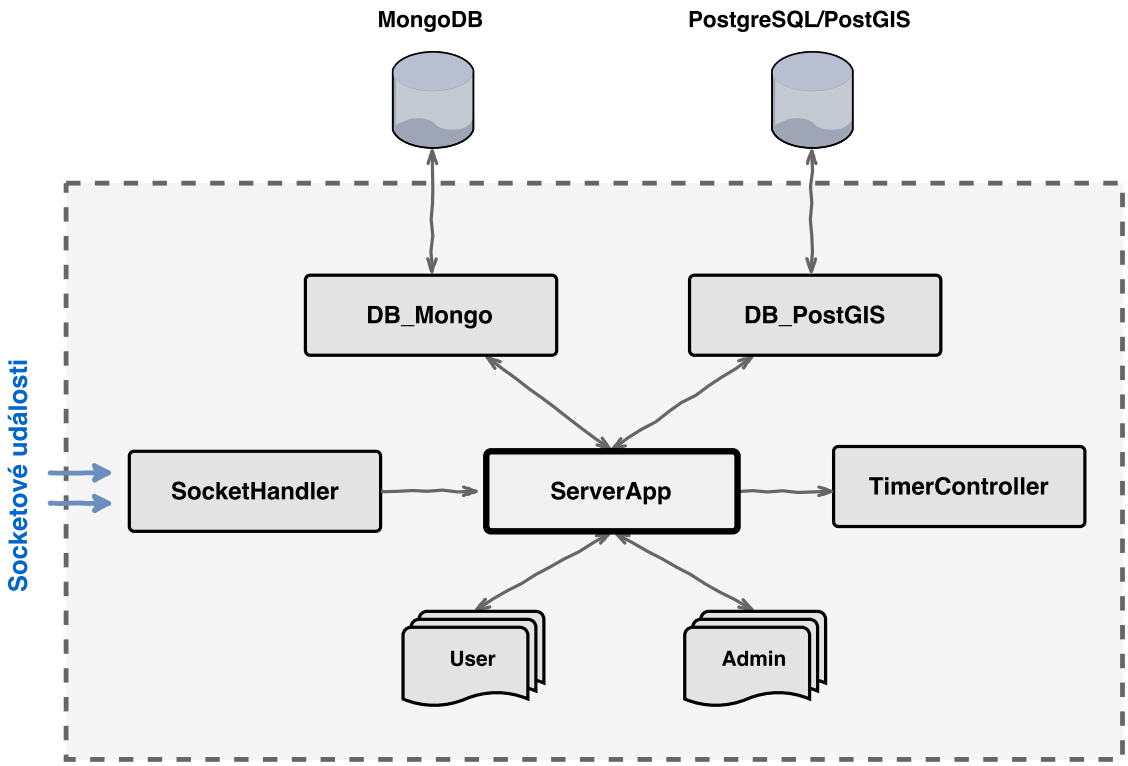
Poslední specifickou oblastí, pro kterou nebylo doposud představené technologické řešení, je komunikace serverové aplikace s klientem. Pro řešení komunikace typu klient-server u systémů vyvinutých nad platformou *Node.js* je připravena velmi rozšířená knihovna *Socket.io*.

Jedná se o událostmi řízenou javascriptovou knihovnu pro realizaci obousměrné komunikace webových aplikací se serverem v reálném čase. Pro obě strany komunikace klient-server poskytuje jinou úpravu knihovny, tedy variantu pro klientský webový prohlížeč a technologické řešení pro serverovou aplikaci vystavěnou využitím frameworku *Node.js*. V obou případech však knihovna nabízí identické API, pro využití vytváření a přijímání soketových událostí, které reprezentují výměnu zpráv mezi klientem a serverem. Základní stavební jednotkou je objekt třídy `Socket`, jejíž předpis je představený v tabulce 7.2.

7.2 Architektura aplikace

Na přiloženém obrázku 7.2 je zjednodušený diagram, který zahrnuje nejvýznamnější logické celky a jejich vzájemné závislosti architektury aplikace. Pojmenování jednotlivých bloků v diagramu odpovídá názvům příslušných implementovaných tříd. Právě jim jsou jednotlivě postupně věnovány následující podkapitoly, které přibližují jejich význam, funkci a zodpovědnosti v architektuře aplikace.

Ústředním bodem celé aplikace je třída `ServerApp`. Její zodpovědností je poskytnutí funkčních metod pro všechny vstupní požadavky v plném rozsahu, správa připojených uživatelů a vytvoření instancí pro přístup k oběma databázovým systémům. V kontextu obsluhy vstupních požadavků se jedná o sadu metod, které jsou implementované v třídě `SocketHandler` se závislostí na příchozí události z klientské strany. Ve zdrojovém kódu jsou vždy označeny prefixem, který na první pohled jasně specifikuje, kterému typu uživatele je metoda přiřazena.



Obrázek 7.2: Architektura serverové aplikace

7.2.1 Obsluha vstupních požadavků

Za vstup do celé serverové aplikace lze označit třídu `SocketHandler`, která svojí logikou představuje obsluhu vstupních požadavků a následné nasměrování k dalšímu zpracování. Jediným parametrem konstruktoru při vytváření instance této třídy je objekt knihovny `SocketIO.Server`, který naslouchá na předem definovaném portu serverového prostředí příchozím událostem z klientské strany. Je důležité zmínit, že během následujícího textu je v různých formách užívaný termín *soket* (angl. *socket*), který však nijak s architekturou *TCP/IP* nesouvisí. V kontextu knihovny `Socket.io` se jedná o třídu, která je stavebním kamenem při komunikaci klienta se serverem a nabízí atributy a metody představené v tabulce 7.2.

Atributy	
`id: string`	unikátní identifikátor soketu klienta
`rooms: Object`	struktura místností, ve kterých je klient připojen
`client: Client`	reference na objekt připojeného klienta
`conn: engine.Socket`	přístup k připojení klienta
Vybrané metody	
`emit(eventName[, args][, ack])`	metoda pro odeslání události identifikované jejím názvem, ke které je možné přiložit libovolné serializované data určené pro přenos, popřípadě *callback* funkci pro potvrzení doručení
`on(eventName, callback)`	metoda pro registraci obslužné *callback* funkce při doručení pojmenované události
`join(room[, callback])`	metoda pro připojení klienta do místnosti, reprezentující logickou oblast působení klienta
`leave(room[, callback])`	metoda pro přímé opuštění místnosti
`disconnect(close)`	metoda pro přímé ukončení spojení soketu, kdy dochází defaultně i k opuštění místnosti

Tabulka 7.2: Objekt třídy `Socket`

Obsluha žádosti o připojení klienta

V rámci samotného připojení se klient hlásí událostí `connection`, jejíž obsluha je v třídě `SocketHandler` reprezentována metodou `onConnection(socket: SocketIO.Socket)`. Parametrem je právě zmiňovaný soket zastupující klientskou stranu. K ustanovení spojení je klientem sestavený `connectionQuery`, což je řetězec obsahující serverovou adresu a parametrické proměnné oddělené znakem `&`, kterými se klient identifikuje. V předmětné aplikaci se jedná o číselné proměnné `type` definující typ připojovaného uživatele a `aid` zastupující identifikátor zobrazeného profilu autoškoly v katalogu.

V této chvíli je nutné definovat jednotlivé typy uživatelů, kteří do komunikace klient-server a tedy do systému vstupují:

- `USER` – Prvním typem je klient reprezentovaný webovým prohlížečem s načteným obsahem katalogu DoAutoškoly.cz, resp. zobrazeným detailem konkrétní autoškoly, odkud pochází zmíněná hodnota `aid` v konfiguračním řetězci.
- `EXTERNAL_MOBILE_SERVER` – Druhý typ je specifický, neboť se jedná o externí serverovou aplikaci, která při komunikaci s popisovanou serverovou aplikací plní funkci klientské strany a informuje o dostupnosti nově vloženého záznamu cvičné jízdy. V tomto případě se jedná o výhradní postavení klienta tohoto typu v komunikaci, jenž musí být právě jediným tímto typem, který je připojený.
- `ADMIN` – Posledním typem je uživatel s právy administrátora, jehož funkčnost je rozšířena o další běžně neveřejné operace a získání výsledků. Tento typ je připraven především pro další rozvoj portálu, kde je pozice administrátora již plánována.

Obsluha příchozích událostí

Pro každý z výše jmenovaných typů je vytvořena sada pojmenovaných událostí knihovny *socket.io*, které umožňují přesně identifikovat v prostředí serveru typ požadavku klientské strany. Pro rozpoznání příchozích událostí je využita metoda `on(eventName, callback)` z instance připojeného soketu. V případě zaregistrování konkrétní události je obsluhou v rámci anonymní funkce zvolena odpovídající metoda řídící třídy `ServerApp` pro následné zpracování.

7.2.2 Správa připojených klientů

Každý z dříve jmenovaných typů uživatele, který se úspěšně připojí k serveru, je reprezentován předpisem odpovídající třídy. V implementaci aplikace jsou pojmenované jako `User`, `Admin` a `ExternalMobileServer`. U prvních dvou jmenovaných se jedná o uživatele, kteří se připojují z katalogu DoAutoškoly.cz spuštěném ve webovém prohlížeči a výběrem autoškoly zobrazí její detail. Uživatelský typ administrátora portálu disponuje oproti běžnému uživateli pouze větším rozsahem poskytovaných dat a funkcí. Z tohoto důvodu v implementaci existuje základní předpis `CatalogUser`, který generalizuje společné atributy a metody obou těchto uživatelů. Jedná se především o přístupové metody k jejich atributům `id:string` a `socket:SocketIO.Socket` využité při správě všech připojených uživatelů.

Je důležité doplnit, že pro zmíněnou správu všech připojených uživatelů jsou implementovány třídy `UsersList` a `AdminsList`, které ve svých atributech obsahují mimo jiné pole objektů `User`, resp. `Admin`. Unikátní hodnotou identifikující objekty v poli

je řetězcová hodnota `id`, obsahující hodnotu identifikátoru připojeného soketu (`socket.id`), kterým uživatel v serverové aplikaci vystupuje. Uživatel je v tomto poli registrován po celou dobu jeho připojení k serveru, tedy do soketové události `disconnect`. Díky tomu je možné k němu kdykoliv přistupovat a skrz jeho soket s ním komunikovat.

V případě třídy `ExternalMobileServer` je využit princip návrhového vzoru Jedináček (angl. *Singleton*), neboť může při komunikaci existovat právě jediná instance této třídy.

Obrázek 7.3: Typy uživatelů

7.2.3 Přístup k databázím

Ze schématu 7.2 je zřejmé, že pro každý databázový systém je vytvořen nezávislý blok pro přístup a dotazování nad daty. Hlavní řízení obsluhy databází je zodpovědností tříd `DB_Mongo` a `DB_PostGIS`, které zajišťují připojení a udržování spojení s databází, provádění dotazovacích operací nad daty v databázi a následné získání a poskytnutí výsledků. Během inicializačního procesu po spuštění serverové aplikace se v obou případech sestavuje konfigurační řetězec pro připojení k databázovému serveru.

Princip vytvoření a navázání spojení s databází *MongoDB* ve třídě `DB_Mongo` a rovněž inicializace dokumentových schémat je identická jako u vyvinutého aplikačního rozhraní pro externí mobilní server. Právě tomu se detailně věnuje celá kapitola 9. Podobně je průběh připojení k databázi *PostgreSQL* v rámci třídy `DB_PostGIS` zmíněn v kapitole 8, kde je doplněn i o relační schéma prostorové databáze.

Obrázek 7.4: Loga databázových technologií [10, 6, 7]

7.2.4 Periodická správa systému

Jelikož běžící serverová aplikace v reálném čase zpracovává velké množství dat, je nutné periodicky kontrolovat režijní průběh aplikace a především stav obou databázových systémů. Tuto funkci zajišťuje třída `TimerController`, která nezávisle rozhoduje o spouštění jednotlivých údržbových operací. Hlavním plánovacím atributem je v současné době časový údaj pro opakování operace.

První variantou je časová perioda, tedy pravidelný interval, na základě kterého je spouštěna příslušná operace. Její řešení spočívá v prostém využití javascriptové funkce `setInterval(function, milliseconds)`, kde parametr `milliseconds` definuje délku periody provádění v milisekundách a parametr `function` popisuje prováděné operace.

Pro specifičtější plánování spouštění úloh je připraven pro využití modul `node-schedule`, který přináší elegantní řešení plánovače. Jeho použití přináší možnosti plánování pro obecné časové periody. Konkrétním případem může být například požadavek na provedení operace každý druhý den v týdnu a podobně. Hlavní metoda modulu `scheduleJob(period, function)` přijímá jako druhý parametr anonymní funkci popisující prováděné operace při spuštění úlohy. Prvním parametrem je určen interval pro plánování provádění události ve speciálním formátu řetězce znaků specifického významu odděleného mezerou. Zástupným znakem při nevyužité volbě některého ze znaků daného formátu je znak hvězdičky „*“. Vysvětlení jednotlivých znaků formátu je převzato a přeloženo z oficiálního repozitáře modulu [25]:

```
*    *    *    *    *    *
|    |    |    |    |    |
|    |    |    |    |    den v týdnu (0 - 7) (0 a 7 reprezentují neděli)
|    |    |    |    měsíc (1 - 12)
|    |    |    den v měsíci (1 - 31)
|    |    hodiny (0 - 23)
|    minuty (0 - 59)
sekundy (0 - 59, volitelné)
```

Následující demonstrační příklad zajišťuje naplánování úlohy pro vypsání textu do konzole aplikace každé úterý, tedy druhý den v týdnu a každou hodinu na začátku 32.minuty. Tedy pravidelně v 0:32, 1:32, 2:32, ..., 23:32 hodin.

```
schedule.scheduleJob('* 32 * * * 2', function () {
     console.log("Test node-schedule");
});
```

Veškerá logika tohoto bloku je připravena pro budoucí vývoj serverové aplikace, kdy plánování periodických úloh bude velmi důležité pro další operace vyhodnocování dat z cvičných jízd. V rozsahu diplomové práce obsahuje pouze jedinou úlohu, která zajišťuje každý den v čase 00:00:00 selekci a následné odstranění korektně neukončených tras v operační databázi *MongoDB* při real-time sledování.

7.2.5 Logování stavů aplikace

Pro podporu při vývoji a rovněž jako nástroj zpětné analýzy průběhu provádění veškerých operací v rámci běhu serverové aplikace je implementována třída `Logger`. Ta slouží pro sestavení zprávy a následné odeslání podle typu informace na standartní výstup, popř. standartní chybový výstup. Při vytváření instance této třídy je konstruktoru předán jediný řetězcový parametr, který obsahuje informaci o názvu třídy nebo logického bloku, kde je právě využíván. Ve zdrojovém kódu třídy `Logger` je tento údaj označený atributem `caller:string`. Každé zprávě předchází prefix sestavený z aktuálního času ve formátu `YYYY-MM-DDHH:mm:ss.SSS` a právě hodnoty atributu `caller`. Za těmito informacemi následuje neomezený počet „štítků“ (v kódu reprezentovány klíčovým slovem *tag*), které jsou sestavovány pomocí objektového zřetězení libovolně za sebe. Poslední částí je dle povahy vytvářené zprávy výběr metody `log(msg:string):void`, `warning(msg:string):void` nebo `error(msg:string):void`. Tímto způsobem jsou tak například logovány události připojení a odpojení klientů na serveru, stav připojení databází, průběh provádění klíčových funkčních bloků a podobně. Díky tomu je možné v případě chybového stavu jednoduše analyzovat stav těsně před vznikem chyby. V následujícím krátkém kódu je ukázka použití třídy `Logger` a jemu odpovídající výstup. Tuto třídu je jako modul možné použít i nezávisle na této práci.

```
/* Ukázka použití třídy Logger */
let logger = new Logger("TestApp");

logger.tag("TAG 1").tag("TAG 2").log("Lorem ipsum dolor sit amet");
logger.warning("warning text");
logger.tag("TAG 3").error("error text");

/* Výsledek na STDOUT | STDERR */
[2017-04-06 17:49:04.362 > TestApp]    [ TAG 1 ][ TAG 2 ]  Lorem ipsum
[2017-04-07 08:16:13.163 > TestApp]    [ WARNING ]  warning text
[2017-04-07 12:13:13.008 > TestApp]    [ ERROR ][ TAG 3 ]  error text
```

7.3 Prostředí vývoje systému

V krátkém představení je vhodné se v textu zmínit i o vývojovém prostředí, které bylo využito. V základu se jedná o IDE společnosti *JetBrains*, která nabízí celou sadu specializovaných prostředí pro vývoj programových výstupů v *Javě*, *Pythonu*, mobilních aplikací a podobně. Byť pro systémy s javascriptovým základem je připraveno prostředí *WebStorm*, z vlastních dřívějších potřeb mimo rozsah diplomové práce jsem zvyklý na práci v IDE s názvem *PhpStorm*. Ten sice nabízí více podporu pro webové aplikace psané v *PHP*, ale díky komplexnosti všech produktů společnosti *JetBrains* lze dodatečnou instalací pluginů, především pro *Typescript* a *Node.js*, plnohodnotně nahradit nabízený *WebStorm*. Využitá verze vývojového prostředí *PhpStorm* během implementace diplomové práce je výrobci označena jako *2016.1*. Pro studijní účely je možné tento software využívat neomezeně.

Simulace běhu serverové aplikace po transpilaci typescriptového kódu lze provést v příkazovém řádku lokálně nainstalované instance frameworku *Node.js* a databází *MongoDB*, resp. *PostreSQL*.

Kapitola 8

Metody vyhodnocení prostorových informací

Název této kapitoly napovídá, že její obsah se zabývá implementačním řešením hlavního cíle této práce, tedy vyhodnocení prostorových údajů zaznamenaných během cvičných jízd autoškol. Tato část celého systému je klíčová pro získání analýzy a výsledků, které jsou dále graficky zobrazeny v katalogovém profilu autoškoly. Metody vyhodnocení v rámci rozsahu diplomové práce nabízejí dvě důležité oblasti analýzy zaznamenaných dat. První v této kapitole je představen postup řešení vyhodnocení plynulosti a čistého času cvičné jízdy. Tedy uplynulá doba, kdy vozidlo během cvičné jízdy bylo v pohybu. Druhým výsledkem analýzy dat je nalezení a optimalizace protnutí všech jízd vybrané autoškoly, které při dostatečném vzorku dat nabízí náhled na frekvenci opakovaných průjezdů stejných křižovatek a ulic.

Jako hlavní úložiště záznamů tras je databázový relační systém *PostgreSQL*, který v současné době díky široké komunitě vývojářů nabízí pravděpodobně nejlepší služby a výkon pro řešení relační databáze. V kontextu diplomové práce byla jeho volba navíc umocněna prostorovým rozšířením *PostGIS*, které tento databázový systém nabízí. Stejně jako v případě operační databáze byly verze těchto technologií omezené možnostmi produkčního databázového serveru. Z tohoto důvodu jsou pro vývoj systému využité verze *PostgreSQL 9.4* a prostorové rozšíření *PostGIS 2.3.1*. V tomto případě nebylo omezení zásadním problémem při řešení.

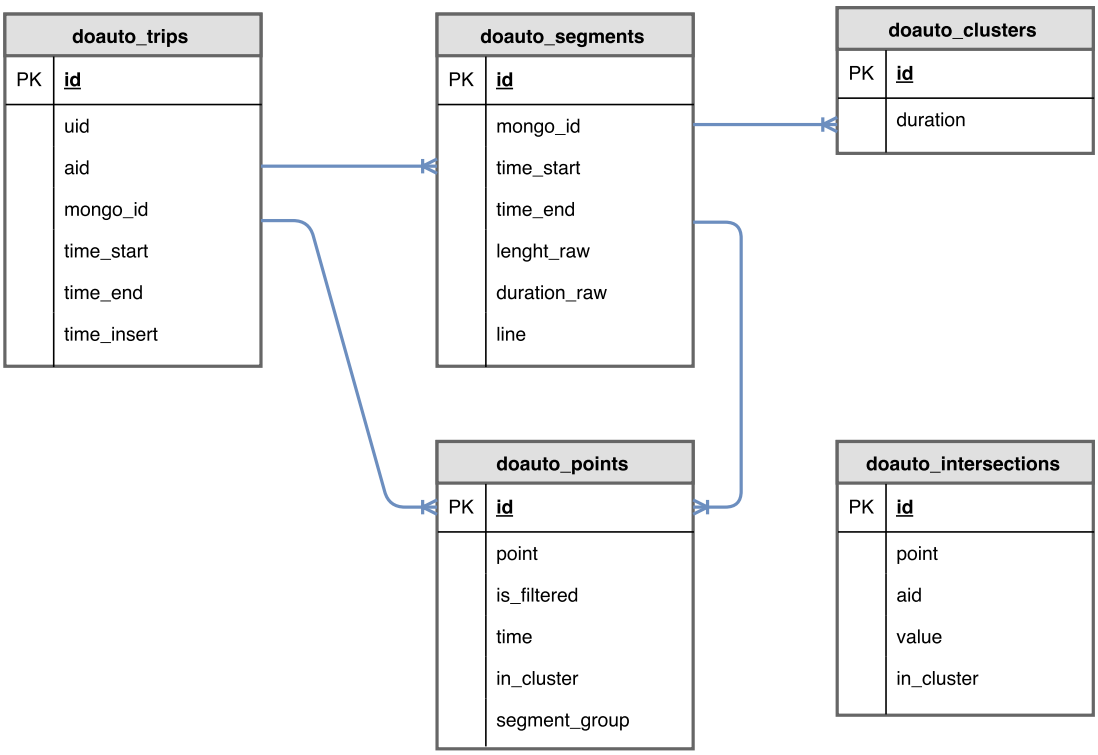
Pro samotnou implementaci je v úvodu této kapitoly důležité znovu připomenou využití knihovny *PG-promise*, která poskytuje metody pro připojení k databázi a správu databázových transakcí a dotazů. Právě provádění transakčních posloupností je základním stavebním kamenem při řešení netriviálních kroků vyhodnocení. Ty nejvýznamnější jsou v této kapitole důkladněji rozebrány.

8.1 Schéma databáze

Na obrázku 8.1 je modelováno relační schéma prostorové databáze pro vyhodnocení dat. Jelikož výsledky obou analýz spolu nijak nesouvisí, nevzniká mezi těmito tabulkami žádná závislost. Jedná se především o nezávislou tabulku *doauto_intersections*, která obsahuje výsledky protnutí všech jízd autoškol. Každý záznam tak nese pouze prostorovou informaci datového typu `POINT`, číselný identifikátor autoškoly `aid`, hodnotu mohutnosti bodu `value` a booleovský příznak příslušnosti ke shluku v rámci okolních bodů pojmenovaný `in_cluster`. Význam těchto sloupců souvisí s postupem vyhodnocení v podkapitole 8.4.

Podstatnou část databázového schématu však zaujímají relace pro popis trasy cvičné jízdy. Jedná se o tabulky *doauto_trips*, *doauto_segments* a *doauto_points*, které svými relačními závislostmi popisují formát datové struktury cvičné jízdy z operační databáze. Jedinou přidanou závislostí, která z datové struktury jízdy nevyplývá, je vztah mezi tabulkami *doauto_trips* a *doauto_points*. Ta byla přidána v průběhu vývoje metod vyhodnocení z důvodu výrazného zjednodušení databázových dotazů i za cenu přidání druhého cizího klíče k entitě *doauto_points*.

Poslední nezmíněnou tabulkou je *doauto_clusters*, která obsahuje výsledky shlukování bodů, které jsou klíčové pro vyhodnocení plynulosti cvičných jízd.



Obrázek 8.1: Schéma prostorové databáze

Navázání spojení s databází

Navázání a automatické udržování spojení s databází *PostreSQL* je plně v režii knihovny *PG-promise*. Při inicializaci spojení pouze vyžaduje konfigurační řetězec pro připojení, který je ve tvaru `pg://[user]:[password]@[host]:[port]/[nazev_db]`. Zodpovědnost za připojení k databázi má v architektuře serverové aplikace třída `DB_PostGIS`, která v konstruktoru volá metodu své instance `makeConnection()`. Ta vytvoří klientské připojení k databázovému serveru, které je po celou dobu běhu aplikace přístupné v atributu `clientDB`. Konstrukce ustanovení spojení poskytovaná knihovnou *PG-promise* je ve zjednodušeném podání demonstrována následujícím příkladem:

```
this.clientDB = this.pgp("konfiguracni_retezec");
this.clientDB.connect()
    .then(function (obj) {
        obj.done(); // success, release the connection;
        logger.log("CONNECT Success");
    })
    .catch(function (error) {
        logger.error("Can't connect to DB server: " + error);
        process.exit(1);
    });
```

8.2 Proces vložení nové trasy

Průběh procesu vložení nové trasy začíná u obsloužení události, která ze strany externí serverové aplikace označí záznam aktuální trasy za kompletní. V tento okamžik zajistí třída `ServerApp` získání datové struktury jízdy z operační databáze a předává řízení operace pro vložení nové trasy instanci třídy `DB_PostGIS`. Ke každé vkládané trase je přistupováno izolovaně jako k objektu třídy `PG_NewTrip` a je tedy možné provádět tento proces pro více jízd zároveň, což je situace, která v reálném využití může kdykoliv nastat. V rámci každé této instance je udržován kontext konkrétní jízdy po celou dobu trvání procesu vkládání.

Celý průběh této rozsáhlé operace lze popsat jako posloupnost databázových transakcí, kde vždy následující transakční sestava dotazů je závislá na dokončení té předchozí. Tato závislost vzniká především na identifikátoru databázového záznamu, který pro další krok posloupnosti nese hodnotu cizího klíče v relační databázi. Posloupnost bloků transakcí je rozdělena do metod podle jejich příslušnosti:

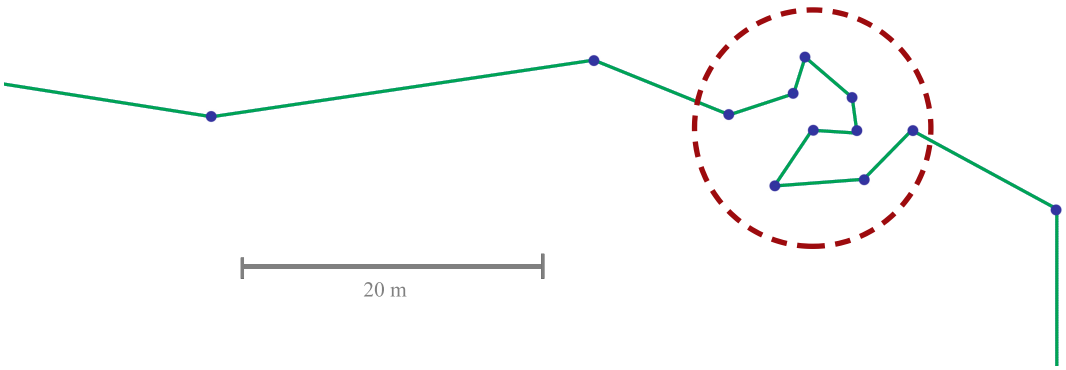
```
insertNewTrip
      |__  insertSegments
                  |__ insertPoints
                              |__ makeLinesFromPoints
```

První tři zástupci této posloupnosti rovněž reprezentují hierarchii datové struktury pro popis cvičné jízdy, tedy náležitost bodů v segmentech jízdy. Poslední metoda `makeLinesFromPoints` sestavuje kolekci databázových dotazů pro vytvoření prostorového objektu `LINE` z bodů každého nově vloženého segmentu. Opět je tato sada dotazů zastřešena transakčním řízením proti vzniku nekonzistentního stavu databáze. Z knihovny *PG-promise* jsou transakce implementovány funkcí `tx`, která anonymní funkcí poskytuje kontext pro sestavení transakčních dotazů, jejich provedení a získání výsledků či chybových stavů.

8.3 Vyhodnocení plynulosti jízdy

Ve chvíli, kdy je ukončen proces vkládání záznamu nové jízdy, je kontext databázového připojení předán nově vytvořené instanci třídy `PG_EvalTrip`, která již svým názvem prozrazuje svoji zodpovědnost při vyhodnocení nově vložených prostorových dat.

Hlavní myšlenkou pro vyhodnocení plynulosti jízd je analýza vzájemné vzdálenosti jednotlivých sousedících bodů při průběhu trasy. Při přerušení plynulé jízdy je GPS přijímač ve statické poloze nebo jen v minimálním pohybu a tedy zaznamenané body se i s ohledem na přesnost GPS souřadnic pohybují jen na velmi malém prostoru. I v této situaci jízdy zodpovídá externí mobilní aplikace za pravidelné zaznamenávání aktuálních GPS souřadnic. Otázkou je tedy princip rozeznání těchto oblastí a jejich časový rozsah. V textu a zdrojových kódech se v této souvislosti dále objevuje pojem *shluk* (angl. *cluster*) a jemu slova odvozená. Situaci řešeného problému ilustruje obrázek 8.2, kde je vyznačen předmětný shluk bodů.



Obrázek 8.2: Ilustrace případu shluku bodů při přerušení jízdy

Známým algoritmem shlukové analýzy bodů je například *K-means*, který spočívá v předem definovaném počtu náhodně rozmístěných k oblastí, do kterých třídí jednotlivé body podle daných parametrů. Tím je v tomto případě vzdálenost od středu oblasti. Algoritmus iterativním postupem upravuje středy oblastí s cílem dosažení co nejmenších hodnot vzdáleností bodů. [28] Velkou nevýhodou pro využití v této práci je však nutnost inicializace algoritmu pro pevně daný počet oblastí, který pro unikátní záznam cvičné jízdy není předem známý. Rovněž je problémem vyžadovaná příslušnost každého bodu do právě jednoho shluku. Cílem vyhodnocení jízdy je však nalezení pouze shluků, které reprezentují přerušenou jízdu, a tedy body zaznamenané během plynulého pohybu nejsou ve výsledku vůbec zahrnuty.

Řešením této situace je využití prostorové agregační funkce `ST_ClusterWithin` přímo v databázovém dotazu pro vyhodnocení shluků. Prostorové rozšíření *PostGIS* tuto funkci nabízí od verze *2.2.0* a jejím výsledkem je pole typů `GeometryCollection`, které definují sady prostorových objektů na základě určené prahové vzdálenosti těchto objektů mezi sebou.

Algoritmus nalezení a vyhodnocení shluků

1. *Nalezení shluků v databázi*

   Základem pro implementační řešení algoritmu je databázový dotaz pro nalezení shluků využitím funkce `ST_ClusterWithin` nad všemi body vybraného segmentu. Jeho výsledek obsahuje kolekci shluků s náležejícími body, které jsou zdrojem záznamů pro následující zpracování.

2. *Předzpracování shlukových kolekcí*

   V tomto kroku je nutné získané kolekce shluků z předchozího kroku připravit k dalšímu zpracování. Jedná se především o rozložení geometrických kolekcí do formátu jednotlivých bodů, které reprezentuje jejich *id* a příslušnost ke shluku *cluster_index*. Důležitou částí v tomto kroku je i odfiltrování shluků, jejichž rozsah obsahujících bodů nevyhovuje stanovené úrovni.

3. *Databázové spojení s konkrétními body*

   V této fázi je provedeno databázového spojení s tabulkou *doauto_points*, která na základě cizího klíče poskytuje rozšiřující informace každého bodu o časovou hodnotu pro další zpracování. Samozřejmostí je i seřazení všech bodů podle jejich času pořízení.

4. *Vyhodnocení databázových dotazů*

   Předchozí jmenované kroky jsou vyhodnoceny v rámci jediného složeného dotazu. Výsledek provedeného dotazu je poskytnutý k programovému zpracování v dalším kroku.

5. *Analýza doby trvání shluků*

   Získaná data z předchozího kroku jsou standartním cyklickým průchodem zpracovány s cílem analýzy časů pořízení krajních bodů ve shluku. Výsledkem je časový údaj reprezentující dobu trvání přerušené jízdy, kterou popisuje shluk nalezených bodů.

6. *Uložení výsledků*

   Posledním krokem je uložení všech vyhodnocených výsledků do databázové tabulky *doauto_cluster*, která obsahuje záznamy s číselnou hodnotou doby trvání v sekundách a identifikátor odpovídajícího segmentu, ve kterém se shluk nachází. Záznamy této tabulky jsou základem pro poskytnutí dat při grafickém zobrazení výsledků.

8.4 Analýza frekvence opakovaných průjezdů

Druhou oblastí vyhodnocení dat je poskytnutí přehledu frekvence opakovaně projížděných křižovatek a ulic. V kapitole 8.2 byla jako posledním krokem vložení nové trasy označena metoda `makeLinesFromPoints`. Jejím výstupem je prostorový objekt typu `LINE`, který spojitě reprezentuje celou projetou trasu každého segmentu cvičné jízdy. Principem této analýzy je získání průsečíků těchto linií pro detekci průjezdu křížení komunikací. Díky drobným nepřesnostem při získávání GPS souřadnic z družic lze tímto postupem detekovat i opakované průjezdy konkrétních silnic, neboť vytvořená linie z GPS bodů cvičné jízdy nikdy nekopíruje trasu silnice přesně. Vypovídající hodnota výstupu této analýzy je velmi závislá na počtu zahrnutých jízd, neboť s rostoucím množstvím zaznamenaných tras jsou místa opakovaných průjezdů mnohem zřetelnější od ostatních.

Implementační řešení spočívá ve složeném databázovém dotazu, který využívá prostorovou funkci `ST_Intersection` z databázového systému, která na základě dvou vstupních objektů typu `LINE` vrací kolekci objektů typu `POINT`, tedy bodů popisujících místa protnutí linií. Tyto body jsou ihned v rámci stejného dotazu vloženy jako nové záznamy tabulky *doauto_intersections*. Celému tomuto procesu předchází selekce pouze těch linií, které náleží jízdám přiřazeným vybrané autoškole. Cílem tedy není získat přehled opakovaných průjezdů v rámci všech autoškol, ale vyhodnotit pohyb v rámci cvičných jízd každé autoškoly zvlášť.

8.4.1 Optimalizace bodů

S přibývajícím počtem zaznamenaných jízd souvisí i nárůst množství bodů protnutí jednotlivých linií. Nejenom pro redukci těchto dat, ale i pro větší vypovídající hodnotu záznamů byl navržen a implementován systém optimalizace dat, který zavádí v kontextu této práce termín *mohutnost bodu*. Jedná se o číselnou hodnotu množství bodů, které předmětný bod zastupuje. Inicializační hodnotou při vkládání nového bodu protnutí je samozřejmě hodnota „jedna“, tedy bod reprezentující sám sebe. Smyslem této optimalizace je nahradit body protnutí v dostatečně blízké vzdálenosti za jediný zástupný bod, jehož poloha je specifikována geometrickým středem těchto bodů (angl. *centroid*) a hodnota mohutnosti obsahuje součet mohutností jím nahrazených bodů.

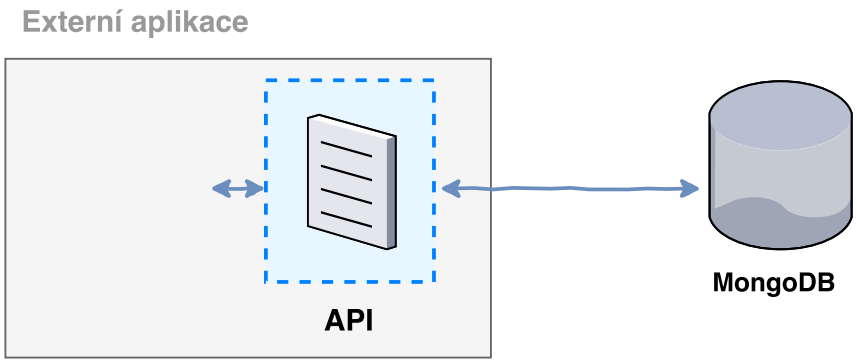
Implementačně se jedná o sekvenci databázových dotazů v rámci jednoho transakčního průběhu. Princip optimalizace spočívá ve stejném výpočtu bodových shluků jako při vyhodnocení plynulosti jízdy. V tomto případě je ale za shluk prohlášena i dvojice bodů, která není od sebe dále než pět metrů. Tato vzdálenost byla zvolena při uvážení šířky silnic a plochy zaujímané křížením komunikací. Všechny body libovolně velkého shluku jsou v databázi nahrazeny jediným, jehož hodnota mohutnosti je součet mohutností nahrazených bodů vypočítaná agregační funkcí `SUM`. Pro geometrický střed shluku a tedy prostorové souřadnice tohoto zástupného bodu je nad celým objektem shluku aplikována prostorová funkce `ST_Centroid`, jejímž výsledkem je právě jediný objekt typu `POINT`.

Kapitola 9

Aplikační rozhraní pro operační databázi

Vývoj aplikačního rozhraní k operační databázi pro záznam cvičné jízdy v reálném čase probíhal odděleně od serverové aplikace a jeho hlavním významem je poskytnout mezivrstvu se sadou metod pro práci s databází *MongoDB*. Celou tuto kapitolu lze považovat i za popisný manuál použití aplikačního rozhraní jako modulu pro přístup k databázi *MongoDB* na straně externí serverové aplikace. Ta byla implementována souběžně k této práci Bc. Jakubem Vonešem a jeho zpětnými vazbami bylo aplikační rozhraní inkrementálně rozšiřováno. Jednotlivé logické celky jsou v této kapitole podrobně představeny i se sadou odpovídajících metod.

Do této doby nebyla v textu práce specifikována použitá verze operační databáze *MongoDB*. S ohledem na možnosti databázového serveru v produkčním prostředí byl od začátku celý systém vyvíjen pro verzi *2.4.10*. Tato verze nabízí běžné operace nad databází *MongoDB*, ale pro složitější problémy by bylo možné využít elegantnější řešení vyšších verzí databáze. Instalace vyšší verze na produkčním serveru bohužel v době vývoje diplomové práce nebyla ze strany poskytovatele serverových služeb možná a proto i v testovacím prostředí byla využita tato verze.



Obrázek 9.1: Modul aplikačního rozhraní v externí serverové aplikaci

9.1 Architektura a struktura schémat

Stejně jako v případě serverové aplikace je implementačním jazykem aplikačního rozhraní *Typescript*. Díky tomu je možné toto rozhraní považovat za modul, který je možné importováním využít kdekoliv v jiné aplikaci. Architektura tohoto modulu obsahuje významnou třídu `DoAuto_MongoDB` a doplňující části v adresáři `/config`. Ten obsahuje soubor `ConfigDB.ts` s konfigurací přístupových údajů k databázím, dále třídu `Logger` s identickým způsobem logování stavů modulu jako u serverové aplikace a soubor `interfaces.d.ts`, který definuje rozhraní datových struktur pro ukládání a získávání dat z databáze. Tyto rozhraní, které se objevují v metodách v různých souvislostech, vždy předepisují formát datové struktury a je důrazně doporučeno tyto rozhraní implementovat i při předávání parametrů metodám aplikačního rozhraní z externích aplikací, a to především z důvodu korektního formátu dat.

Právě jmenovaná třída `DoAuto_MongoDB` obsahuje veškerou poskytovanou logiku. Při vytváření instance této třídy probíhá mimo jiné i inicializace dokumentových schémat, které reprezentují zanořené záznamy v databázi a jsou tak předpisem pro objektové mapování již představené knihovny *Mongoose*. Každé schéma je instancí třídy `mongoose.Schema`, jejíž konstruktor přijímá jediný parametr v podobě struktury specifikující formát *NoSQL* dokumentu nebo subdokumentu. V následující ukázce je demonstrován princip vytvoření zjednodušeného schématu `tripSchema` pro cvičnou jízdu a dvě podschémata `segmentSchema` a `pointSchema`, které vytvářejí hierarchickou reprezentaci databázového dokumentu.

```
/* souradnice a cas bodu cvicne jizdy */
let pointSchema = new this.mongoose.Schema({
    lat: Number,
    lon: Number,
    time: Number
});

/* segment cvicne jizdy s casem zacatku a konce segmentu */
let segmentSchema = new this.mongoose.Schema({
    points: [pointSchema],
    timeStart: Number,
    timeEnd: Number
});

/* hlavni schema cvicne jizdy */
let tripSchema = new this.mongoose.Schema({
    uid: Number,                // identifikator studenta
    aid: Number,                // identifikator autoskoly
    segments: [segmentSchema],
    timeStart: Number,
    timeEnd: Number
});
```

9.2 Navázání spojení s databází

Knihovna *Mongoose* pro objektové mapování *NoSQL* dokumentů poskytuje rovněž obsluhu navázání a udržování spojení s databází. Samotnému navázání spojení předchází sestavení konfiguračního řetězce pro připojení. V aplikačním rozhraní se tak děje automaticky v závislosti na výběru jedné z metod z tabulky 9.1. Přístupové údaje ke všem variantám připojení jsou připraveny v souboru `config/ConfigDB.ts`. V přiložených zdrojových kódech na CD nejsou vyplněny přístupové údaje k oběma serverům z důvodu ochrany dat portálu DoAutoškoly.cz.

`connectToLocalhost()`
Popis: Zajišťuje navázání spojení s lokálním úložištěm. Vrací instanci třídy `DoAuto_MongoDB` s ustanoveným spojením k databázi.
`connectToTest()`
Popis: Zajišťje navázání spojení s databází na testovacím serveru. Vrací instanci třídy `DoAuto_MongoDB` s ustanoveným spojením k databázi.
`connectToServer()`
Popis: Zajišťuje navázání spojení s produkčním databázovým serverem. Vrací instanci třídy `DoAuto_MongoDB` s ustanoveným spojením k databázi.

Tabulka 9.1: Metody pro navázání spojení

9.3 Záznam trasy do databáze

Jednou z nejdůležitějších částí celého aplikačního rozhraní jsou metody pro záznam průběhu cvičné jízdy. Posloupnost průběhu záznamu začíná vytvořením nové instance schématu `tripSchema`, které obsahuje veškeré položky potřebné pro specifikaci každé jízdy. Navíc obsahuje několik statistických parametrů, které byly do schématu přidány na vyžádání Jakuba Voneše, neboť operační databáze je pro jeho systém a mobilní aplikaci hlavním zdrojem dat pro zpětnou správu zaznamenaných jízd. Metoda `createNewTrip` přijímá inicializační data položek schématu v prvním parametru, který specifikuje typ `ITrip` definovaný společně s dalšími v souboru `config/interfaces.d.ts`. O úspěšném vytvoření záznamu cvičné jízdy v databázi je externí aplikace informována hodnotou unikátního identifikátoru v databázi. Samotné vložení záznamu do databáze je realizováno využitím knihovní funkce `save` volanou nad vytvořeným schématem knihovny *Mongoose*.

V posloupnosti zaznamenávání trasy dále následuje neomezený počet vložení izolovaných segmentů, tedy libovolně dlouhých úseků cvičné jízdy, které obsahují i klíčové pole bodů jízdy `points`. Celá vkládaná datová struktura je opět specifikována vytvořeným rozhraním, v tomto případě pojmenovaným jako `ISegment`. Každý segment obsahuje i položku `order`, která číselně reprezentuje pořadí vytvořeného segmentu na straně mobilní aplikace a je důležitá pro pozdější využití.

Posledním krokem je vzestupné seřazení segmentů metodou `sortSegments`. Nutnost řešit řazení segmentů vznikla během testování v průběhu vývoje aplikačního rozhraní. Technicky je řazení nad uloženými daty realizováno funkcí `update()`, která využívá operátor `$sort` ze specifikace *MongoDB*. Tímto seřazením metoda i formálně ukončuje záznam celé cvičné jízdy.

Předmětné metody pro záznam trasy jsou přehledně specifikovány v tabulce 9.2.

`createNewTrip(data, callback)`
Popis: Vytvoří záznam dle schématu `Trip` specifikující popisné atributy cvičné jízdy.
Parametry:
`data:ITrip` | datová struktura vkládaných dat cvičné jízdy
`callback:(_id, time)` | *callback* funkce s argumenty `_id` a času vložení trasy
`insertSegment(_id, data, callback)`
Popis: Vloží nový segment do kolekce segmentů v záznamu jízdy.
Parametry:
`_id:string` | identifikátor záznamu jízdy v databázi
`data:ISegment` | datová struktura vkládaného segmentu
`callback:(result)` | *callback* funkce s argumentem `true` jako výsledek úspěšné operace, jinak `false`
`sortSegments(_id, edited, callback)`
Popis: Provede vzestupné seřazení segmentů jízdy podle pořadí vytvoření.
Parametry:
`_id:string` | identifikátor záznamu jízdy v databázi
`edited:boolean` | hodnota `true` určuje zda se mají řadit editované segmenty
`callback:(result)` | *callback* funkce s argumentem `true` jako výsledek úspěšné operace, jinak `false`

Tabulka 9.2: Metody pro záznam trasy

9.4 Získání uložených dat

Pro zobrazení zaznamenaných jízd pro další využití vyžaduje externí mobilní aplikace získání těchto dat. Sada metod z tabulky 9.3 nabízí různé způsoby získání záznamů z databáze dle jejich určení a významu. U prvních dvou se v implementaci jedná o triviální využití *Mongoose* funkce `findById`, která na základě identifikátoru vrací odpovídající dokument z databáze.

Implementační řešení metody `getUserLastTrips` pro výběr tras uživatele s časovým omezením je vhodné pro demonstraci zápisu netriviálního dotazu pro selekci dat. V komentáři ukázky zdrojového kódu je uveden SQL formát zápisu stejné podmínky pro porovnání se zápisem dotazu při práci s databází *MongoDB*. Využitý operátor `$gt` je ekvivalentní relačnímu operátoru „*je větší* >“.

```
// WHERE uid = x AND ( timeLastUpdate > y OR timeDelete > z)
this.Trip.find({uid: x})
   .and([
     {$or: [
        {timeLastUpdate: {$gt: y}},
        {timeDelete: {$gt: z}}
        ]
     }
   ]).exec(function (err, result) {/*...*/});
```

Velmi zajímavou z pohledu implementace je rovněž metoda `getFilteredTripById`, která vrací trasu ve standartním schématu, avšak obsahuje pouze body, které nejsou externí mobilní aplikací při záznamu trasy označeny jako filtrované. Pokud bychom použili tzv. tečkovou notaci pro specifikaci umístění tohoto atributu, jednalo by se o zápis: `trip.segments.points.isFiltered`, tedy trojnásobné zanoření ve schématu dokumentu. Pro řešení tohoto problému je využit agregační framework přímo z nabídky funkcí databázového systému *MongoDB*. Byť se v tomto případě nejedná o agregační situaci, jednotlivé operátory dělají tento framework využitelný i v jiných případech dotazování. Framework funguje na principu proudové transformace databázových dokumentů podle operátorů `$match` pro výběr prvků a `$group` pro následné seskupení. Jejich použití může být libovolné. Pro řešení zmíněné filtrace bodů je po výběru záznamu trasy dle jejího `_id` provedeno „rozvinutí“ kolekcí dokumentu operátorem `$unwind` konstrukcí:

```
/*...*/

{"$unwind": "$segments"},
{"$unwind": "$segments.points"},

/*...*/
```

Výsledkem těchto dvou operací je nahrazení kolekcí `segments`, resp. `points` za jednotlivé záznamy dokumentu na nejvyšší úrovni hierarchie schématu. Ve schéma `Trip` tedy každý řádek reprezentuje unikátní bod z kolekce `points` náležící do odpovídajícího záznamu z nadřazené kolekce `segments`. Na této úrovni je poté možné využít znovu operátor `$match` pro výběr řádků odpovídající podmínce `"segments.points.isFiltered": false`. V postupu řešení zbývá po vyfiltrování už jenom dvojí použití operátoru `$group` pro seskupení rozvinutých záznamů zpět do kolekcí v opačném pořadí. Tedy nejprve pro kolekci `points` a následně `segments`. Posloupnost těchto operátorů je použita jako kolekce v parametru funkce `aggregate` vracející požadovaný výsledek.

`getTripById(_id, callback)`
Popis: Poskytuje záznam jízdy z databáze dle jeho identifikátoru `_id`.
Parametry:
`_id:string` | identifikátor záznamu jízdy v databázi
`callback:(result)` | *callback* funkce s argumentem získaného záznamu jízdy

`getEditedTripById(_id, callback)`
Popis: Poskytuje záznam editované jízdy z databáze dle jeho `_id`.
Parametry:
`_id:string` | identifikátor záznamu jízdy v databázi
`callback:(result)` | *callback* funkce s argumentem získaného záznamu jízdy

`getFilteredTripById(_id, callback)`
Popis: Poskytuje záznam jízdy obsahující pouze body označené jako filtrované.
Parametry:
`_id:string` | identifikátor záznamu jízdy v databázi
`callback:(result)` | *callback* funkce s argumentem získaného záznamu jízdy

`getUserNewTrips(where, callback)`
Popis: Poskytuje záznamy jízd konkrétního uživatele vyhovující časové podmínce.
Parametry:
`where:IQueryWhereUserTrips` | položky časové podmínky pro vyhledání
`callback:(result)` | *callback* funkce s argumentem kolekce získaných záznamů

Tabulka 9.3: Metody pro získání uložených dat

9.5 Správa uložených dat

V závislosti na potřebách ze strany vývoje externí serverové a mobilní aplikace nabízí aplikační rozhraní i metody pro správu uložených záznamů jízd. Jednou z nich je metoda `isExistTrip`, která vrací výsledek testu na existenci záznamu dle vstupních atributů. Ty jsou specifikovány typem `IQueryWhereIsExist`. Záznam je označen za již existující, pokud v databázi vyhovuje podmínce shody `uid` uživatele a času začátku a konce trasy (`timeStart`, `timeEnd`). V rámci správy uložených dat jsou poskytovány i metody pro editaci záznamů a jejich trvalé odstranění.

Knihovna *Mongoose* pro editaci záznamu nabízí dva způsoby řešení. Prvním je funkce `update`, která je přímo určena pro toto využití. Editaci záznamu v databázi lze však provést i způsobem s funkcí `findOne` nebo `findById`, které ve svém výsledku v kontextu anonymní funkce vrací selektovaný dokument. Přímo k němu lze pak přistupovat a měnit jeho hodnoty. Akce uložení změněného dokumentu je vyvolána knihovní funkcí `save` nad upravovaným dokumentem. Seznam všech metod aplikačního rozhraní z této funkční oblasti je jmenován i s popisem v tabulce 9.4.

`isExistTrip(where, callback)`
Popis: Poskytuje informaci, zda záznam trasy dle vstupních podmínek již existuje.
Parametry:
`where:IQueryWhereIsExist` | položky podmínek pro vyhledání
`callback:(isExist, data)` | *callback* funkce s argumenty `true` a získaného záznamu trasy pokud existuje, jinak `(false, null)`
`markAsDeletedTripById(_id, callback)`
Popis: Označí záznam trasy jako smazaný.
Parametry:
`_id:string` | identifikátor záznamu jízdy v databázi
`callback:(deleteTime)` | *callback* funkce s argumentem času smazání záznamu
`updateSegmentsEdit(_id, data, distance, callback)`
Popis: Edituje kolekci segmentů a údaj celkové délky jízdy.
Parametry:
`_id:string` | identifikátor záznamu jízdy v databázi
`data:ISegmentEdit` | datová struktura editovaných segmentů
`distance:number` | hodnota délky trasy po editaci
`callback:(result)` | *callback* funkce s argumentem `true` jako výsledek úspěšné operace, jinak `false`
`deletePermanentlyTripById(_id, callback)`
Popis: Provede odstranění záznamu jízdy z databáze.
Parametry:
`_id:string` | identifikátor záznamu jízdy v databázi
`callback:(result)` | *callback* funkce s argumentem `true` jako výsledek úspěšné operace, jinak `false`

Tabulka 9.4: Metody pro správu uložených dat

9.6 Sledování jízdy v reálném čase

Jako poslední část aplikačního rozhraní zbývá představit sadu metod pro sledování jízdy v reálném čase. Tyto metody spojuje zjednodušené schéma dokumentu `realtime_Trip`, které je vytvořené cíleně pro využití v reálném čase a oproti schématu `Trip` neobsahuje pole segmentů trasy, ale přímo kolekci geografických bodů typu `pointSchema`. Průběh každé jízdy je tímto schématem reprezentován v databázi a zpětně tak poskytuje geografická data pro zobrazení projeté trasy, avšak pouze po dobu této jízdy.

Všechny vybrané metody jmenované v tabulce 9.5 související s touto podkapitolou není nutné z pohledu implementačního řešení důkladněji představovat. Operace vytvoření trasy, vložení bodu, získání trasy dle jejího `_id` nebo smazání z databáze jsou svým principem realizace shodné jako dříve představené řešení záznamu a správy tras.

`realtime_createNewTrip(data, callback)`
Popis: Vytvoří záznam nové jízdy v databázi.
Parametry:
`data:IRealtimeTrip` | datová struktura sledované jízdy v reálném čase
`callback:(result)` | *callback* funkce s argumentem `_id` vložené trasy
`realtime_insertPoint(_id, data, callback)`
Popis: Vloží nový bod do záznamu jízdy.
Parametry:
`_id:string` | identifikátor záznamu jízdy v databázi
`data:IPoint` | datová struktura vkládaného bodu
`callback:(result)` | *callback* funkce s argumentem `true` jako výsledek úspěšné operace, jinak `false`
`realtime_getTripById(_id, callback)`
Popis: Poskytuje záznam jízdy z databáze dle jeho identifikátoru `_id`.
Parametry:
`_id:string` | identifikátor záznamu jízdy v databázi
`callback:(result)` | *callback* funkce s argumentem získaného záznamu jízdy
`realtime_getActiveTripsByAid(aidr, callback`
Popis: Poskytuje právě probíhající jízdy autoškoly.
Parametry:
`aid:number` | identifikátor autoškoly
`callback:(result)` | *callback* funkce s argumentem kolekce získaných záznamů

Tabulka 9.5: Metody pro real-time sledování

Kapitola 10

Grafické zobrazení výsledků

Neméně důležitou částí této práce bylo zobrazení výsledků v katalogu DoAutoškoly.cz tak, aby přínos vyhodnocených záznamů byl maximálně využit pro srovnání autoškol. Je však důležité doplnit, že v této situaci musí být na prvním místě objektivita zobrazených informací pro porovnávání v rámci katalogu. Právě objektivita byla v této části velmi diskutovaným tématem, neboť nastavení kritérií pro zobrazení dat v katalogu je nutné plánovat tak, aby žádná z autoškol nebyla zvýhodněna nebo naopak poškozena. Hlavním měřítkem je samozřejmě počet zaznamenaných jízd. Ten je závislý na množství žáků autoškoly, kteří si nainstalují a využijí připravenou mobilní aplikaci. Právě proto je nutné potřebnou dobu nejprve sbírat data, vyhodnocovat je odděleně na úrovni každé autoškoly a až následně zobrazit výsledek.

I přestože není z tohoto důvodu možné ihned zobrazit vyhodnocené výsledky v produkčním prostředí portálu, je grafické zobrazení výsledků v rámci této práce připravené k okamžitému použití. Pro každou z prováděných analýz je navrženo grafické uživatelské rozhraní, které intuitivně a přehledně zobrazuje konkrétní data. Každé z těchto analýz je vyhrazena oddělená podkapitola pro popis návrhu uživatelského rozhraní a jeho implementačního řešení.

10.1 Použité technologie

Stejně jako v předchozích případech je hlavním implementačním jazykem pro svoji komplexnost *Typescript*. Rozdílem je pouze využití asynchronní specifikace modulové architektury *AMD*. Stejně tak je pro komunikaci se serverovou aplikací využita již představená knihovna *Socket.io*. Zbývá tedy představit technologie, které jsou určeny pro specifické využití ve webových aplikacích.

10.1.1 Leaflet

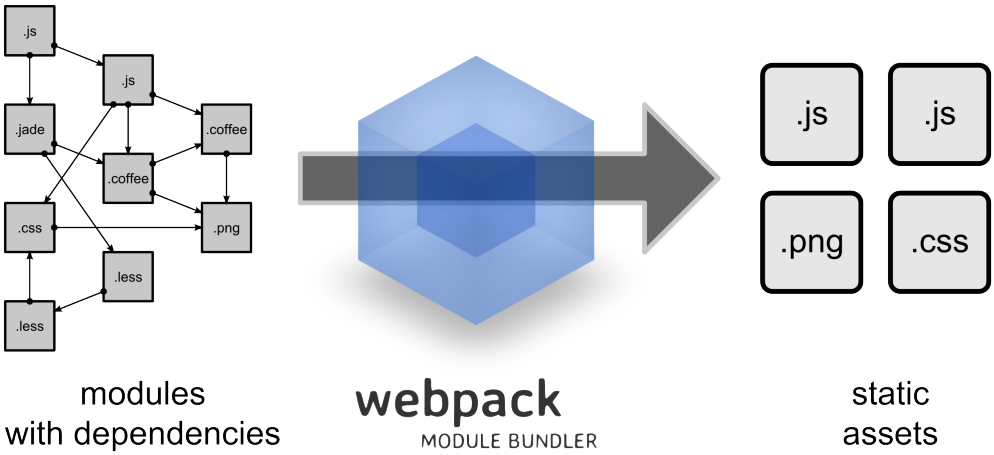
Leaflet je javascriptová open-source knihovna pro tvorbu interaktivních map. [17] Nabízí velmi jednoduché použití a velkou výkonnost vykreslování mapových vrstev i v případě velkého počtu objektů. Jako mapový podklad jsou využité dlaždice velmi rozšířeného projektu *OpenStreetMap*. Tato knihovna rovněž nabízí velkou sadu možných rozšíření základní verze. Jediným rozšířením využitým pro zobrazení frekvence průjezdů cvičných jízd je plugin *Heatmaps*, který umožňuje mapové podklady překrýt vrstvou vektorových dat pro zobrazení teplotní mapy, ve které je každá hodnota reprezentována barvou určitého spojitého barevného spektra.

10.1.2 Chartist.JS

V open-source prostředí existuje mnoho knihoven pro vykreslení grafů v prostředí webových stránek. Pro účely diplomové práce byla zvolena knihovna *Chartist.JS*, která poskytuje responzivní řešení grafů s velkou škálou možností přizpůsobení. Právě možnost vlastních úprav podoby grafových prvků byla velké kritérium, neboť výsledky jsou zobrazeny v katalogovém profilu, který má v rámci celého portálu určený grafický styl a je nutné ho dodržovat. Základní verzi lze rozšiřovat pomocí pluginů, čehož bylo využito i v této práci. Konkrétně rozšířením o pluginy *Tooltip plugin* a *Axis title plugin*. Jejich význam vyplývá z jejich anglických názvů, tedy rozšíření pro možnost rychlých vysvětlivek (angl. *tooltip*) a popisků obou os grafu (angl. *axis title*). Dokumentace a popis knihovny je dostupný z [23].

10.1.3 Webpack

Hlavní motivací pro využití knihovny *Webpack* je co možná nejvíce urychlit načítání javascriptového kódu ve webovém prohlížeči. Díky této knihovně je možné transpilovat zdrojový kód do jediného výsledného javascriptového souboru, který obsahuje efektivní definici všech tříd jako modulů. Při transpilaci je v klientském prostředí využito asynchronní načítání modulů dle specifikace *AMD*, které je pro tuto knihovnu ideální. Přináší výhody v načítání modulů až ve chvíli, kdy jsou potřebné pro běh klientské aplikace. Konfigurace pro práci knihovny, vstupní a výstupní soubory a další atributy jsou specifikovány v kořenovém adresáři aplikace v souboru `webpack.config.js`. Základní verzi knihovny lze samozřejmě dále rozšiřovat. V této práci byl využit plugin *UglifyJS* pro výsledné minimalizování javascriptových konstrukcí. Efektivitu pluginu v případě této práce potvrzuje téměř pětinásobné zmenšení velikosti zdrojového kódu. Celý princip nejlépe demonstruje obrázek převzatý z oficiální online dokumentace knihovny. [13]



Obrázek 10.1: Princip funkce knihovny *Webpack*.

10.2 Aplikace pro zobrazení výsledků

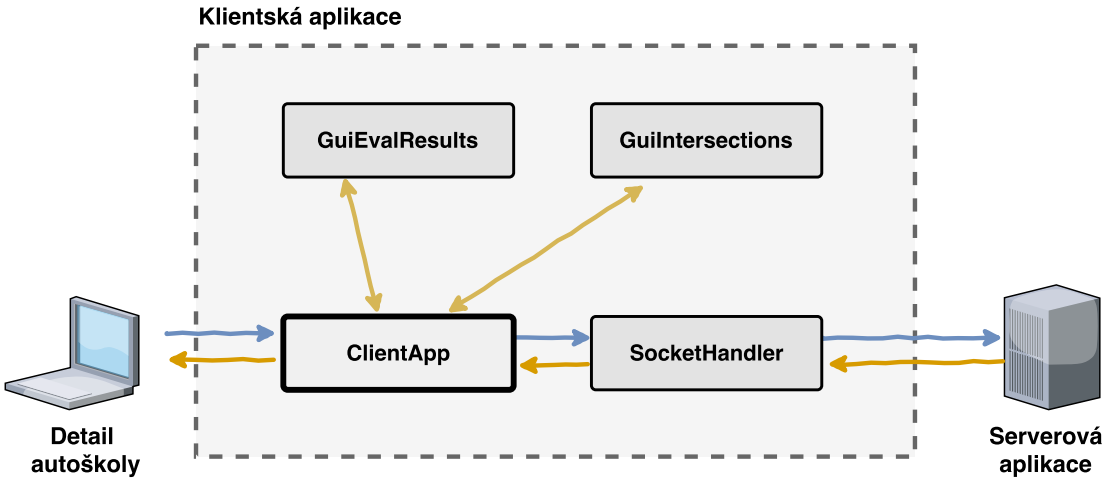
Zodpovědnost veškerého zobrazení výsledků v grafické podobě má aplikace vytvořená v jazyce *Typescript*. Z pohledu budoucího zvyšování rozsahu poskytovaných informací je její návrh řešen komplexním způsobem tak, aby přidání dalšího bloku s vyhodnocenými daty byl maximálně přímočarý a nijak neovlivnil stávající prvky. Každý takový blok v aplikaci vystupuje jako oddělený modul.

Řídícím členem v implementaci je třída `ClientApp`, jejíž hlavní zodpovědností jsou následující:

- plnit úlohu vstupního bodu aplikace ze strany webového portálu,
- zajistit obsluhu požadavků a komunikaci se serverem,
- spravovat instance jednotlivých tematických bloků pro zobrazení dat.

Inicializace této aplikace probíhá při načítání každého detailu autoškoly v prohlížeči. V tento okamžik se na základě sady vstupních parametrů ustanoví a dále udržuje spojení se serverovou aplikací pomocí knihovny *Socket.io*. Komunikace se serverovou aplikací je odstíněna do třídy `SocketHandler`, která ve svém konstruktoru obsahuje proces připojení k serverové aplikaci a navázání obslužných funkcí k jednotlivým soketovým událostem. Tento celý průběh je na obrázku 10.2 znázorněn pomocí modrých šipek. Žlutá barva orientovaných spojení znázorňuje zpracování odpovědi požadavku na získání dat ze strany serveru.

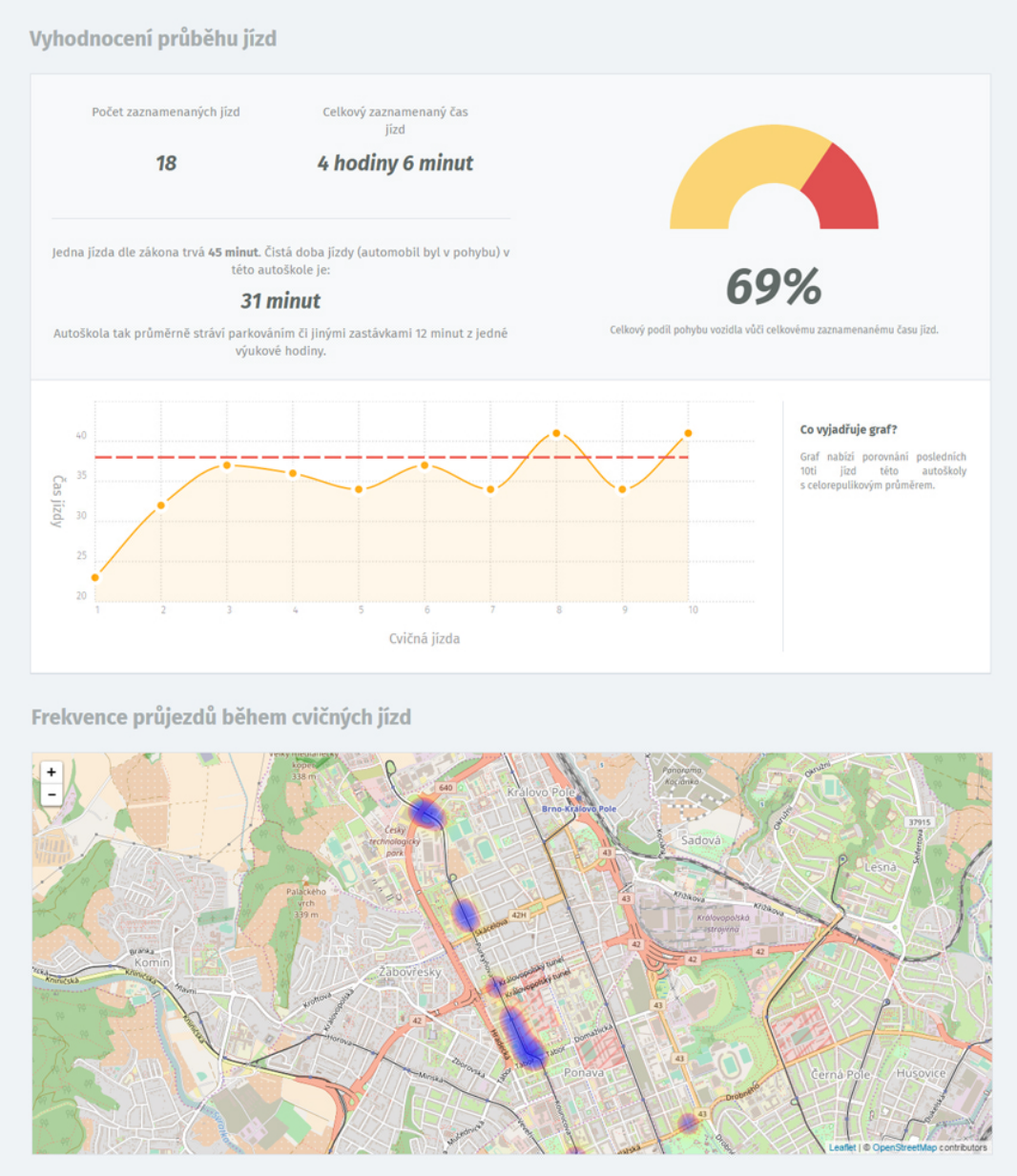
Každý tematicky oddělený blok zobrazení datových informací je v třídě `ClientApp` zastoupený instancí svého předpisu, která poskytuje sadu metod pro výslednou prezentaci ve webovém portálu. V architektuře klientské aplikace se jedná o třídy `GuiEvalResults` a `GuiIntersections`.



Obrázek 10.2: Princip funkčnosti klientské aplikace

10.3 Grafické uživalelské rozhraní

V diplomové práci Ing. Jana Hrivnáka [22] je detailně popsán návrh grafického stylu, význam jednotlivých prvků z pohledu UX a klíčové vlastnosti uživatelského rozhraní portálu DoAutoškoly.cz. V rámci této části práce je nutné při zobrazení všech výsledků analýz cvičných jízd dodržovat tento grafický styl. Samotné umístění grafické reprezentace vyhodnocených dat rozšiřuje portfolio každé autoškoly v jejím katalogovém profilu. Na obrázku 10.3 je výsledné rozmístění dvou tematických bloků na stránce. Detailní přehled výsledného grafického rozhraní přináší následující kapitoly, které postupně obsahují důležité výřezy s vysvětlením významu.



Obrázek 10.3: Celkový náhled zobrazených výsledků

Vyhodnocení průběhu jízd

Nejdůležitějším blokem graficky zobrazených výsledků vyhodnocení je téma plynulosti cvičných jízd, které je stěžejním nástrojem pro porovnání autoškol v katalogu. Z hlediska implementace má zodpovědnost za zobrazení těchto dat třída `GuiEvalResults`, která poskytuje sadu metod pro zobrazení textových hodnot a všech grafových prvků využitím představené knihovny *Chartist*.

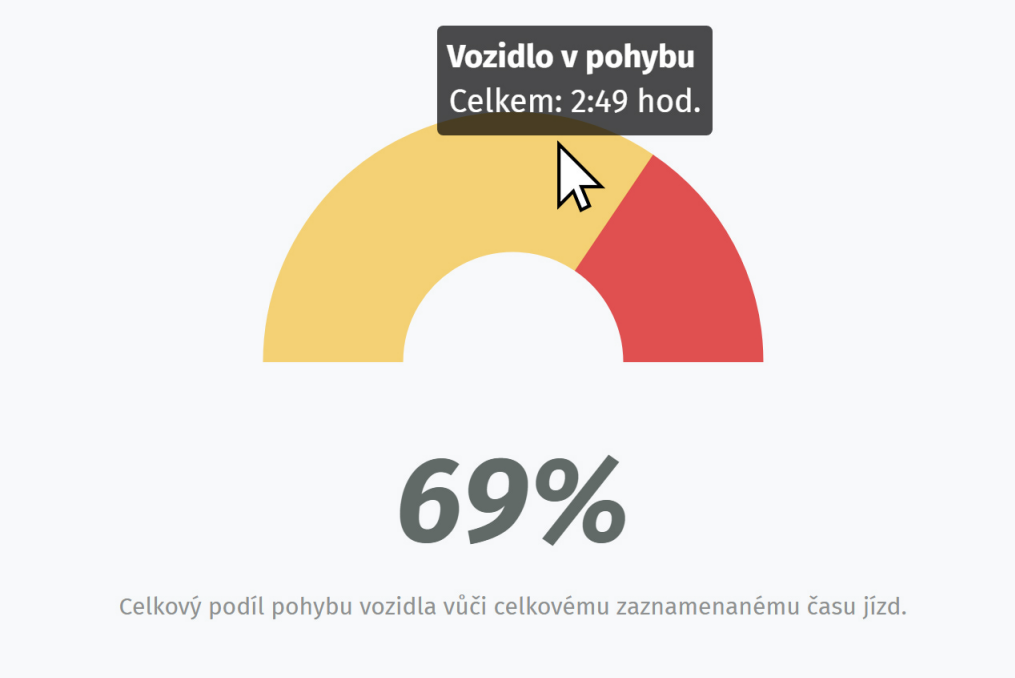
Obrázek 10.4: Oblast I. - vyhodnocení průběhu jízd

Z pohledu grafického uživatelského rozhraní se v tomto panelu jedná o tři oblasti, které reprezentují různou formu výsledků. Tou první, jejíž výřez je na obrázku 10.4, je prostor, který nabízí v horní části statistické informace o zaznamenaných jízdách. Mnohem důležitější informaci z pohledu srovnání autoškol však přináší prostor spodní části.

Myšlenka objektivity: V rámci jednotlivých autoškol bude neustále odlišný počet zaznamenaných a vyhodnocených cvičných jízd, čímž mohou být v některých případech autoškoly znevýhodněny vůči jiným. Stejně tak může být průběh cvičných jízd v jednotlivých autoškolách odlišný. Především doby stání za účelem technické výuky či parkování mohou velmi ovlivnit vyhodnocení plynulosti jízdy.

Řešení: Jedna cvičná jízda má dle zákona stanovenou délku 45 minut. Znormováním výsledků jednotlivých autoškol právě délkou jedné cvičné jízdy a následným průměrem všech autoškol lze získat hranici, která určuje průměrnou hodnotu čistého času jízdy všech autoškol. Právě s touto hodnotou lze již pracovat pro mnohem objektivnější srovnání.

Druhá část horního panelu (výřez na obrázku 10.5) obsahuje půlkruhový výsečový graf, který zobrazuje celkový poměr čistého času, kdy je vozidlo během cvičné jízdy v pohybu, a času, kdy je jízda z jakéhokoliv důvodu přerušena. Procentuální údaj doplňuje zobrazení rychlé nápovědy při najetí myši nad vybranou část grafu, kde se nachází doplňující a vysvětlující informace o vybrané výseči.

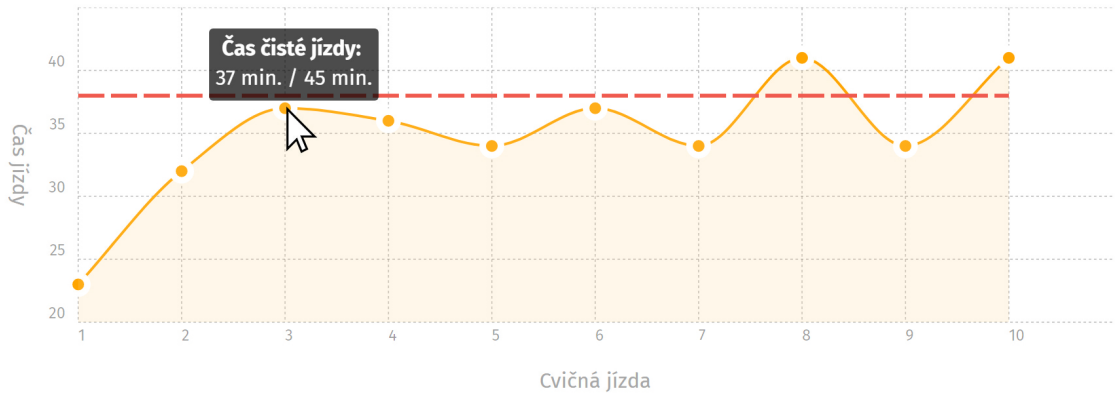


Obrázek 10.5: Oblast II. – vyhodnocení průběhu jízd

Téma objektivity bylo velmi diskutované i s Ing. Janem Hrivnákem, jehož požadavky z pohledu autora a vedoucího projektu DoAutoškoly.cz určily hlavní směr pro grafické zobrazení výsledků. Veškeré návrhy zobrazených informací byly prodiskutovány s ohledem na smysluplnost, korektnost i obchodní záměr, který je pro budoucnost katalogu plánován.

S tím souvisí i poslední oblast tohoto panelu, která je pro „čistotu“ načítaného detailu autoškoly skryta, ale intuitivní prvky vedou k jejímu snadnému zobrazení. Jejím obsahem je liniový graf, který zobrazuje spojitou křivkou vyhodnocené výsledky z posledních deseti jízd. Ke každé této hodnotě je připojena opět událost po najetí myši pro zobrazení rychlé nápovědy. Svislá osa grafu reprezentuje hodnotu čistého času pohybu vozidla během jízdy v minutách, která je opět normovaná zákonnou délkou cvičné jízdy (45 minut). S vodorovnou osou jsou spojeny jednotlivé položky jízd.

Ve výřezu této oblasti (obrázek 10.7) je v grafu výrazná přerušovaná červená vodorovná linie, která reprezentuje klíčovou hodnotu pro srovnání autoškol. Jedná se o úroveň vymezující průměrnou hodnotu čistého času jízd všech autoškol. Na první pohled je tedy zřejmé, zda aktuálně zobrazená autoškola se v posledních 10-ti cvičných jízdách pohybuje nad touto hranicí a nebo čistý čas pohybu vozidla byl nižší, než celorepublikový průměr.

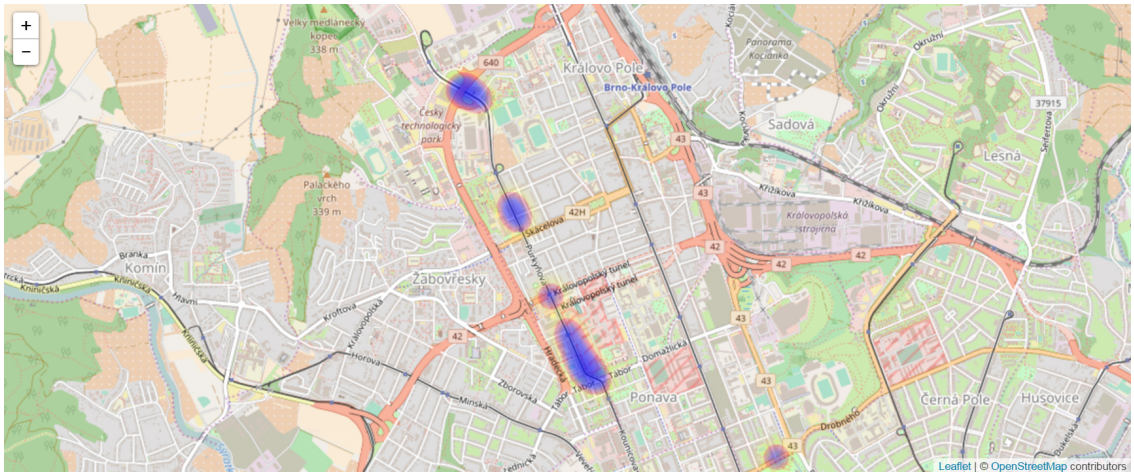


Obrázek 10.6: Oblast III. — vyhodnocení průběhu posledních 10-ti jízd

Frekvence průjezdů během cvičných jízd

Druhou tematicky oddělenou oblastí vyhodnocených výsledků je zobrazení frekvence průjezdů během cvičných jízd. Jedná se o mapový objekt s překryvnou vrstvou, která zobrazuje teplotní mapu pomocí knihovny *Leaflet* a jejího rozšíření *Heatmaps*. Spojité spektrum barev v této vrstvě rozlišuje v různých hladinách mapového přiblížení místa, která autoškola během vyhodnocených jízd nejčastěji projížděla. Tento výsledek vyhodnocení přináší další rozšíření portfolia autoškoly, které může více přiblížit trasování cvičných jízd.

Implementační řešení je obsaženo v třídě `GuiIntersections`, která poskytuje metody pro inicializaci mapových podkladů a následné překrytí teplotní mapou na základě dat získaných z prostorové databáze.



Obrázek 10.7: Mapové zobrazení frekvence průjezdů

Kapitola 11

Testování a ověření korektnosti výsledků

Testování celého systému pro vyhodnocení tras probíhalo prakticky po celou dobu jeho vývoje. Na počátku realizace se jednalo spíše o přesně směrované testování jednotlivých komponent a funkcí serverové aplikace nebo aplikačního rozhraní. Cílem bylo odstranit především počáteční chyby v rámci zpracování vstupních dat a vzájemné komunikace komponent systému. Význam jednotky testovacích dat lze v kontextu této práce chápat jako reprezentaci jedné zaznamenané trasy z cvičné jízdy, která odpovídá definovanému formátu.

Změnu principu testování během vývoje přineslo nejen postupné rozšiřování aplikace, ale především dostupnost reálně naměřených geografických dat z GPS zařízení. Zdrojem byla již několikrát v textu zmíněná mobilní aplikace v rámci současně vytvářené diplomové práce Bc. Jakuba Voneše. S tím přišla i nutnost implementace jednoduché testovací aplikace pro správu zaznamenaných jízd, řízení událostí pro přenesení jízd do prostorové databáze a později zobrazení dílčích výsledků vyhodnocení. Tato testovací klientská aplikace byla v závěru vývoje upravena pro potřeby demonstrace výsledků. Pro budoucí vývoj tohoto systému má další potenciál pro poskytování funkčního rozsahu uživateli s právy administrátora portálu.

Přínosem pro vývoj systému byly i poznatky z reálného provozu cvičných jízd od zástupců autoškol: *Autoškola SBS, Šumperk* a *Autoškola Martin Náplava, Šumperk*.

11.1 Ověření korektnosti výsledků

Především pro oblast vyhodnocení čistého času pohybu vozidla během cvičné jízdy bylo nutné výsledky ověřit tak, aby výstupy analýz byly korektní a odpovídaly skutečnému průběhu zaznamenané jízdy. Za tímto účelem byl v několika případech proveden cílený sběr dat při jízdě osobním automobilem s plánem simulace různých situací přerušení jízdy. Trasování i doba jízdy byly záměrně ovlivněny na základě poznatků od zástupců jmenovaných autoškol a vlastních zkušeností z jízdního výcviku při absolvování kurzu autoškoly. Celý průběh těchto jízd byl pro zpětnou kontrolu vyhodnocených hodnot mapován principem zápisu do časové osy.

V této fázi vývoje se velmi osvědčily i data získaná z pohybu při využití osobních vlaků, byť drážní vozidla s cílem diplomové práce vůbec nesouvisí. Hlavním přínosem těchto dat byla transparentnost průběhu pohybu, neboť nalezené shluky geografických bodů nutně musely korespondovat s nácestnými zastávkami a stanicemi, kde vlaková souprava přerušila svou jízdu.

11.2 Výsledky testování

Velké množství unikátních testovaných záznamů nabídlo rozsáhlou škálu situací, které mohou během cvičných jízd nastat. V několika případech došlo i k odhalení nekorektních stavů a výsledků, které při prvotním návrhu postupu vyhodnocení nebyly uvažovány. Jednou z těchto situací, které způsobily nesprávné výsledky, byla například trasa, která zahrnovala přerušení prohybu vozidla v místech, kde již v předcházející části jízdy k zastavení došlo.

Dalším přínosem velkého počtu vzorků byl základ pro postupné nastavení úrovně parametrů, na základě kterých je u geografických bodů trasy prohlášena shluková příslušnost či nikoliv. Tím se podařilo například odfiltrovat krátká stání na křižovatkách řízených světelnou signalizací a podobně.

S každou zaznamenanou jízdou se rychle zvyšuje množství uložených dat v prostorové databázi. Průběh projeté trasy standartní cvičné jízdy je reprezentován průměrně 3000-4000 geografickými body. S cílem testování výkonnosti metod pro vyhodnocení byly záměrně vytvářeny trasy, které obsahovaly řádově dvojnásobný až trojnásobný počet bodů. Díky těmto „dlouhým jízdám“ byly některé řídící konstrukce a části databázových dotazů nahrazeny efektivnějšími postupy.

Na závěr této kapitoly je zajímavé zmínit hodnoty množství testovaných geografických informací. Celkově testovací databáze při ukončení vývoje obsahovala 472 záznamů cvičných jízd, kterým náleží 1 524 163 geografických bodů pro popis jejich trasy. Tyto čísla pocházejí z identifikátorů záznamů v relační databázi, které jsou automaticky inkrementovány s každou novou operací vkládání a reprezentují tedy množství vložených dat. Začátek této číselné řady identifikátorů pochází z fáze implementace vyhodnocování tras a nejsou tak v této sumě zahrnuty záznamy během vývoje automatického přesunu dat z operační do prostorové databáze. Je nutné zdůraznit, že výčet nereprezentuje unikátní záznamy, neboť během testování byly některé specifické trasy vyhodnocovány záměrně vícekrát.

Kapitola 12

Závěr

Všechny předchozí kapitoly popisují vymezenou část diplomové práce a její řešení. Podrobné seznámení s katalogem DoAutoškoly.cz ihned v úvodu řešení ještě více podpořilo aktivní přístup k problému, neboť samotný projekt Ing. Jana Hrivnáka je velmi kvalitně navržený a prosazuje myšlenku pro obecný prospěch. Jelikož se webový katalog DoAutoškoly.cz nachází v počáteční fázi rozvoje, příležitost spoluúčasti v podobě implementace rozšíření stávajícího stavu projektu byla pro mě velkou motivací.

Jsem velmi rád, že jsem v rámci první části této práce poznal důkladně oblast *NoSQL* databází, kterou jsem ovládal do začátku řešení práce jen okrajově. Společně se studiem relačních databázových systémů s prostorovou nadstavbou tyto nové informace podpořily volbu technologií a návrh nejvhodnějšího řešení pro ukládání geografických dat v reálném čase a jejich následné nezávislé zpracování a vyhodnocení.

Spolu související kapitoly 7 a 8 přináší popis postupu vývoje serverové aplikace pro zpracování cvičných jízd. Kapitola 8, obsahující popis metod vyhodnocení, je záměrně oddělena od přehledového popisu realizace architektury serverové aplikace. Hlavním důvodem rozdělení je zdůraznění významu obsahu této kapitoly jako stěžejní části serverové aplikace a celého systému. Mimo serverovou aplikaci probíhal vývoj aplikačního rozhraní pro přístup k operační databázi, především s cílem poskytnutí databázových služeb pro záznam trasy v reálném čase. Jelikož je toto aplikační rozhraní využíváno externí aplikací, obsahem kapitoly 9 je mimo zmínění netriviálních implementačních postupů i popisný manuál pro jeho použití.

Výsledek implementačního řešení splňuje všechny motivační i formální cíle, které byly důležitým faktorem pro cílevědomý a zodpovědný přístup k realizaci tohoto systému. V závislosti na rozhodnutí autora projektu DoAutoškoly.cz bude výsledek této práce spuštěn do reálného provozu s velkým potenciálem přínosu pro hlavní myšlenku celého projektu i této práce. Stavu úspěšného dokončení a připravenosti ke spuštění do produkčního prostředí tohoto rozšíření nebrání ani grafické zobrazení získaných výsledků. Tento text se řešením zabývá v kapitole 10, kde specifikuje význam jednotlivých prvků grafického uživatelského rozhraní s ohledem na vzhled celého portálu.

Celý systém je navržený pro další průběh rozšiřování o další funkční bloky pro vyhodnocení zaznamenaných jízd cvičné autoškoly. Lze navrhnout například možnost porovnání autoškol při zohlednění velikosti obce, ve které sídlí nebo zmiňované rozhraní administrátora portálu. Jsem přesvědčený, že portál DoAutoškoly.cz se bude dále rozvíjet a podporovat prvotní myšlenku s cílem zvyšování kvality autoškoly v České republice.

Literatura

[1] *Apache CouchDB 2.0 Documentation*. [Online; navštíveno 18.11.2016].
URL http://docs.couchdb.org/en/2.0.0/

[2] *GPX: the GPS Exchange Format*. [Online; navštíveno 4.11.2016].
URL http://www.topografix.com/gpx.asp

[3] *MySQL 5.7 Reference Manual - Extensions for Spatial Data*. [Online; navštíveno 25.11.2016].
URL http://dev.mysql.com/doc/refman/5.7/en/spatial-extensions.html

[4] *Neo4j Spatial*. [Online; navštíveno 18.11.2016].
URL https://github.com/neo4j-contrib/spatial

[5] *Node.js*. [Online; navštíveno 5.12.2016].
URL https://nodejs.org/en/

[6] *PostGIS 2.1.9dev Manual*. [Online; navštíveno 25.11.2016].
URL http://postgis.net/docs/manual-2.1/

[7] *PostgreSQL 9.4.10 Documentation*. [Online; navštíveno 25.11.2016].
URL https://www.postgresql.org/docs/9.4/static/index.html

[8] *Socket.IO*. [Online; navštíveno 5.12.2016].
URL https://socket.io/

[9] *SpatiaLite 4.1.0 Manual*. [Online; navštíveno 25.11.2016].
URL https://www.gaia-gis.it/fossil/libspatialite/index

[10] *The MongoDB 3.2 Manual*. [Online; navštíveno 18.11.2016].
URL https://docs.mongodb.com/manual/

[11] *The Neo4j Manual v2.1.5*. [Online; navštíveno 18.11.2016].
URL http://neo4j.com/docs/2.1.5/what-is-a-graphdb.html

[12] *TypeScript - JavaScript that scales*. [Online; navštíveno 5.12.2016].
URL https://www.typescriptlang.org/

[13] *Webpack module bundler*. [Online; navštíveno 17.2.2017].
URL https://webpack.github.io/

[14] *WGS84 - Školení Úvod do (Open Source) GIS*. [Online; navštíveno 4.11.2016].
URL
http://training.gismentors.eu/open-source-gis/soursystemy/wgs84.html

[15] *Terminologický slovník geodézie, kartografie a katastra*, Úrad geodézie, kartografie a katastra SR a Český úřad zeměměřický a katastrální, 1998, ISSN 80-88716-36-5.

[16] *Terminologický výkladový slovník pojmů z oblasti geoinformací*, Úřad pro veřejné informační systémy, 2001, ISSN 1213-225X.

[17] Agafonkin, V.: *Leaflet - a JavaScript library for interactive maps.* [Online; navštíveno 17.2.2017].
URL http://leafletjs.com/

[18] Celko, J.: *Joe Celko's complete guide to NoSQL.* Morgan Kaufmann Publishers, 2014, ISBN 978-0-12-407192-6.

[19] Edlich, S.: *NOSQL Databases.* [Online; navštíveno 13.11.2016].
URL http://nosql-database.org/

[20] Güting, R. H.: *Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems.* In *CG '88 Proceedings of the International Workshop on Computational Geometry on Computational Geometry and its Applications*, 1988, ISBN 3-540-50335-8.

[21] Heywood, I.; Cornelius, S.; Carver, S.: *An introduction to geographical information systems.* Pearson Education, 2006, ISBN 978-0-13-129317-5.

[22] Hrivnák, J.: *Zvyšování kvality autoškol pomocí sdílení uživatelských zkušeností.* Diplomová práce, Fakulta informačních technologií, VUT Brno, 2016.

[23] Kunz, G.: *Chartist - Simple responsive charts.* [Online; navštíveno 17.2.2017].
URL https://gionkunz.github.io/chartist-js/

[24] Oršulák, T.; Pacina, J.: *Geoinformatika.* CEVRAMOK, 2012, ISBN 978-80-904927-5-2.

[25] Patenaude, M.: *GitHub - node-schedule/node-schedule: A cron-like and not-cron-like job scheduler for Node.* [Online; navštíveno 7.4.2017].
URL https://github.com/node-schedule/node-schedule

[26] Rapant, P.: *Družicové polohové systémy.* VŠB - TU Ostrava, 2002, ISBN 80-248-01248.

[27] Rigaux, P.; School, M.; Voisard, A.: *Spatial Databases With Application to GIS.* Morgan Kaufmann Publishers, 2002, ISBN 1-55860-588-6.

[28] Steinbach, M.; Tan, P.-N.; Kumar, V.: *Introduction to Data Mining.* Addison-Wesley Longman Publishing Co., Inc. Boston, 2005.

[29] Voženílek, V.: *Geografické informační systémy I. - pojetí, historie, Základní kompomenty.* Vydavatelství Univerzity Palackého, 1998, ISBN 80-7067-802-X.

[30] VÚGTK: *Terminologický slovník zeměměřičství a katastru nemovistostí.* [Online; navštíveno 4.11.2016].
URL https://www.vugtk.cz/slovnik/1070_geograficka-data--geodata--geoprostorova-data

Přílohy

Příloha A

Obsah přiloženého CD

- `\clientApp\` - zdrojové soubory ke klientské aplikaci pro grafické zobrazení výsledků
- `\serverApp\` - zdrojové soubory k serverové aplikaci
- `\doauto_mongodb\` - zdrojové soubory aplikačního rozhraní pro operační databázi
- `\thesis\` - zdrojové soubory technické zprávy diplomové práce
- `\dp_xsoula02.pdf` - technická zpráva ve formátu PDF
- `\README.txt` - textový soubor s popisem obsahu CD