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 PLÁNOVÁNÍ TRAS SERVISNÍCH PRACOVNÍKŮ

SYSTEM FOR SERVICE WORKERS' ROUTES PLANNING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MICHAL MLČOCH**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JIŘÍ HYNEK, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

146407

Ústav: Ústav informačních systémů (UIFS)
Student: **Mlčoch Michal**
Program: Informační technologie
Specializace: Informační technologie

Název: **Systém pro plánování tras servisních pracovníků**
Kategorie: Informační systémy
Akademický rok: 2022/23

Zadání:

1. Prostudujte koncept chytrých měst (*Smart City*). Zaměřte se na problematiku správy servisních požadavků a jejich plánování pro servisní pracovníky.
2. Prostudujte problematiku výpočtu trasy v mapách na základě různých parametrů (čas, délka apod.). Prozkoumejte existující řešení určené pro tento účel.
3. Analyzujte problém výpočtu trasy pro servisní pracovníky v projektu Smart City firmy Logimic. Definujte parametry, které mají vliv na určení optimální trasy.
4. Dle výsledků analýzy navrhněte vhodné řešení pro plánování tras servisních pracovníků.
5. Po konzultaci s vedoucím navržené řešení implementujte a integrujte do systému Smart City firmy Logimic.
6. Proveďte testování funkčnosti a použitelnosti implementace.

Literatura:

- Kirimtat, A., Krejcar, O., Kertesz, A., & Tasgetiren, M. F. (2020). Future trends and current state of smart city concepts: A survey. *IEEE access*, *8*.
- Toth, P., & Vigo, D. (Eds.). (2014). *Vehicle routing: problems, methods, and applications*. Society for Industrial and Applied Mathematics.
- Cordeau, J. F., Laporte, G., Savelsbergh, M. W., & Vigo, D. (2007). Vehicle routing. *Handbooks in operations research and management science*, *14*, 367-428.
- Interní dokumentace firmy Logimic.

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hynek Jiří, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 21.10.2022

Abstrakt

Tato bakalářská práce se zabývá problematikou servisních požadavků vznikajících v chytrých městech a jejich plánováním do tras servisních pracovníků. V práci je probrána problematika konceptu chytrých měst, optimalizace trasy a rozbor služeb poskytujících optimalizaci tras. Cílem práce je vytvořit systém pro plánování tras servisních pracovníků pro platformu chytrého města vyvíjenou firmou Logimic. Hlavní funkcionalitou je umožnění optimalizace trasy skrze služby třetích stran a je kladen důraz na to, aby bylo možné snadným způsobem změnit službu třetí strany poskytující optimalizaci.

Abstract

This bachelor's thesis deals with the issue of service requests arising in smart cities and their planning into the routes for service workers. The thesis discusses the issue of the concept of smart cities, route optimization and analysis of services providing route optimization. This work aims to create a system for planning the routes of service workers for the smart city platform developed by Logimic. The main functionality is to enable route optimization through third-party services, and the emphasis is on being able to easily change the third-party service providing the optimization.

Klíčová slova

chytrá města, servisní požadavky a pracovníci, optimalizace tras, TSP, VRP, TypeScript

Keywords

smart cities, service requests and workers, route optimalization, TSP, VRP, TypeScript

Citace

MLČOCH, Michal. *Systém pro plánování tras servisních pracovníků.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Hynek, Ph.D.

Systém pro plánování tras servisních pracovníků

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jiřího Hynka, Ph.D. Další informace mi poskytli pan Ing. Michal Valný, Ph.D., pan Ing. František Mikula a pan Ing. Petr John. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Michal Mlčoch
3. května 2023

Poděkování

Chtěl bych poděkovat především vedoucímu práce Ing. Jiřímu Hynkovi, Ph.D., za velmi přátelský přístup, za cenné rady, názory a za veškerou pomoc v průběhu psaní a realizace bakalářské práce. Dále bych chtěl poděkovat panu Ing. Michalu Valnému, Ph.D., panu Ing. Františku Mikulovi a panu Ing. Petru Johnovi z firmy Logimic za poskytnuté informace k systému a cenné rady.

Obsah

Kapitola 1

Úvod

V dnešní době přibývá měst, která se snaží využít výhody konceptu chytrých měst. Chytrá města využívají ve velké míře zařízení připojená do internetu věcí (IoT), a to např. kvalita ovzduší, chytré odpadkové koše, senzory na parkovacích místech, detekce úniků potrubí a mnoho dalších. Vzhledem k tomu, že tato zařízení pracují automaticky, je čas od času potřeba lidské práce, a to ať už pro instalaci nového zařízení, jejich opravu nebo např. výměnu článků, pokud zařízení není připojeno trvale do sítě. Dále ve městech mohou vznikat podněty od širší veřejnosti. Ty se mohou týkat jakékoliv stránky města, a tak lidé mohou upozornit na poškozený chodník, dát požadavek na zastřižení zeleně atp. Požadavky jednotlivých zařízení, která mohou vznikat automaticky, nebo podnětů od veřejnosti jsou sdružovány v systému chytrého města, kde jsou na základě těchto podnětů vytvářeny servisní požadavky.

Vzniklými servisními požadavky se dále správce zabývá, vyhodnocuje jejich závažnost a dále je přiděluje kompetentním lidem na vyřešení. Servisních požadavků přibývá postupně více a více a je zapotřebí plánovat jejich vyřešení. K tomu, aby bylo denně možné vyřešit co nejvíce požadavků, a to dle možností aktuální situace, je zapotřebí optimalizovat práci servisních pracovníků. Toho lze docílit vhodným plánováním pracovní doby servisního pracovníka, a to např. selekcí požadavků a následně v jakém pořadí by měly být řešeny. Proto je potřeba optimalizovat jeho cestu po servisních požadavcích tak, aby jich za svou pracovní dobu vyřešil co nejvíce a co nejefektivněji.

Optimalizace cesty není jednoduchý problém, který lze snadno řešit. Je zde mnoho proměnných, jež vstupují do problému, a je potřeba je zohlednit tak, aby všechny požadavky byly splněny a zároveň se jednalo o co nejvíce optimální řešení. Optimalizace cesty je již v historii definovaný pojem jako *problém směrování vozidel* a již existuje mnoho řešení, která se daný problém snaží řešit různými technikami.

Cílem práce je navrhnout systém pro správu tras servisních požadavků, který i umožňuje plánování tras, resp. jejich optimalizaci. Výstupem práce je návrh architektury systému pro správu tras a její implementace do již existující platformy chytrého města. Tato práce je řešena ve spolupráci s firmou Logimic pro jejich vyvíjenou platformu pro chytrá města.

Práce se skládá jak ze tří teoretických kapitol, kde jsou podrobněji rozebrána související témata, a čtyř praktických, ve kterých jsou aplikovány získané znalosti z teoretických kapitol. Kapitola 2 je věnována popisu konceptu chytrých měst, internetu věcí a servisním požadavkům. Kapitola 3 se zaobírá problematikou optimalizací cest, kde je nejprve představen problém a následně jakými způsoby jej lze řešit. Poslední 4. kapitola teoretické části se věnuje dostupným webovým řešením, která nabízejí řešení problému směrování vozidel. První praktická část je popsána v kapitole 5, kde je analyzován současný stav, uživatelé,

jejich potřeby a požadavky na řešení. Následující kapitola 6 se zabývá návrhem řešení, a to především architekturou, databází a definicí rozhraní. Kapitola 7 se věnuje implementaci navrženého řešení z předešlé kapitoly. Poslední 8. kapitola praktické části se zabývá testováním výsledného řešení. Kapitola 9 obsahuje celkové závěrečné zhodnocení práce.

Kapitola 2

Chytrá města

V této kapitole jsou představeny dva koncepty nad ICT[1]. Nejprve je představen *koncept chytrých měst* a poté následuje interpretace *internetu věcí*, který je využíván v konceptu chytrých měst. V konceptu chytrých měst se dále vyskytují *servisní požadavky*, které v něm hrají podstatnou roli.

2.1 Charakteristika

V dnešní době žije velké procento lidí ve městech a toto procento i do budoucna poroste. V USA v roce 2014 žilo ve městech 54 % populace a odhaduje se, že do roku 2050 by mohlo obývat města až 66 % [42]. V důsledku urbanizace je kladen vysoký důraz na požadavky měst, a to zejména na obytné zóny, zdravotnictví, vzdělání, práci a mnoho dalšího. Díky urbanizaci mají města více prostředků, a proto mohou růst a přinášet lidem lepší podmínky pro život. Na druhé straně to nejsou jen pozitiva, ale je zde mnoho výzev a problémů, na které lze narazit při urbanizaci. Některé problémy mohou být řešeny na poli ICT, a v důsledku toho se začíná objevovat koncepce chytrých měst [26, 42].

První zmínka o konceptu chytrých měst se objevila v 90. letech minulého století [42] a je spojena s procesem urbanizace myšlenkami inovace, výzkumu a technologií. Pro tento koncept existuje mnoho myšlenek a definic na základě úhlu pohledu. Lze jej například definovat jako „spojení mezi fyzickou, sociální, obchodní a ICT infrastrukturou pro chytřejší městské oblasti“ [19] nebo jako „moderní město, které musí využívat výhod z ICT a zlepšovat kvalitu života a služeb pro obyvatele města“ [27]. Práce Galan-Garcial et al. [16] uvádí model města se zaměřením na optimalizaci dopravy za použití chytrých semaforů a dalších prostředků, jež jsou zpracovávány celulárními automaty a neuronovými sítěmi. Předpokladem je i otevřenost lidí vůči tomuto konceptu, schopnost přijímat změny a navrhovat změny tak, aby město bylo tvořeno, jak sami chtějí. Další předpoklady jsou například vyspělá a neustále se zvětšující infrastruktura IoT zařízení, aplikační a služební vybavenost na poli ICT.

Chytré město může být dle [18] charakterizováno šesti body:

- **chytrá ekonomika** – konkurenceschopnost, produktivita, flexibilita trhu práce.
- **chytří lidé** – dobrá kvalifikace, flexibilita, kreativita, otevřenost k celoživotnímu vzdělávání.
- **chytrá vláda** – transparentní vláda, veřejné a sociální služby.

[1] ICT – Informační a komunikační technologie.

- **chytrá přeprava** – národní/mezinárodní dostupnost, ICT infrastruktura, inovativní a bezpečné způsoby dopravy.
- **chytré prostředí** – přírodní zdroje, nízké znečištění, vhodné přírodní podmínky.
- **chytré bydlení** – kvalita bydlení, možnost turistiky, dostupnost vzdělání, kultura.

V návaznosti na tyto charakteristiky lze určit hlavní oblasti výzev chytrých měst a to: (1) soukromí a bezpečnost, (2) chytrá vláda, (3) městská doprava, (4) energetika a prostředí a (5) zdraví a bydlení.

Jedna z výzev tohoto konceptu je městská doprava [42, 22]. Je zde mnoho dat, která lze získat pro lepší chápání dopravy ve městě, a to například z dopravních kamer, GPS polohy ve vozidlech, chytré poklady v MHD a mnoho dalšího. Tyto prvky se nazývají aktivními prvky (dochází k častým aktualizacím), ale také se zde vyskytují statické zdroje dat (aktualizace dat nejsou tak časté), jako jsou geografická data (GIS[2]), GPS data, body zájmu (POI[3]) apod. S daty lze pracovat jak historicky, tak i v reálném čase nebo kombinací obou metod. Dle historických dat je možné hledat různé vzory chování dopravy a možnost jim v budoucnu předcházet. Na základě dat lze v dopravě provádět optimalizaci městské dopravy, což může napomoct k lepší vytíženosti, snížení přetížení a efektivitě. Do výzvy městské dopravy je možné taktéž zařadit chytré parkování. Podle senzorů na parkovištích lze sledovat, zda jsou místa volná a kde se nacházejí. Informace mohou být například poskytnuty v mobilní aplikaci, která zjednoduší parkování a ušetří čas.

2.2 Internet věcí

Internet věcí *Internet of Things (IoT)* [29, 33] je technologie, která již v dnešní době hraje významnou roli v získávání dat z různých zařízení, a to jak na poli zdravotnictví, autoprůmyslu, automatizace, chytrých měst a v mnoho dalších. Tento pojem se začal objevovat na přelomu 20. a 21. století. Za poslední desetiletí rapidně vzrostl zájem o tento koncept. To je možné vidět i na číslech, kdy v roce 2021 bylo přes 12,2 miliardy [20] připojených aktivních zařízení do sítě IoT. I jeho definice se v čase měnila a není možné ji pro tento koncept jednoznačně určit.

Pro příklad lze uvést definici podle Tarkoma a Katasonov [40], která je poměrně detailní: „globální síť a infrastruktura služeb s proměnnou hustotou a konektivitou se schopnostmi autokonfigurace založené na standardních a vzájemně kompatibilních protokolech a formátech, které se skládají z heterogenních věcí, které mají unikátní identifikátor, fyzické a virtuální atributy a jsou bezproblémově a bezpečně integrovány do internetu."

Věci nebo zařízení mohou být zapojeny do internetu za předpokladu, že je lze digitálně identifikovat. Identifikaci může zajišťovat například RFID nebo NFC. K tomu, aby byla zajištěna jednodušší komunikace s koncovými zařízeními, je zde využíváno *middleware* a různých architektur. S IoT je úzce spjata technologie *cloud*, ať už jako úložiště nebo pro zpracování dat. Cloud umožňuje využívat sdílené zdroje a dosahovat velkého výkonu pro zpracování obrovského počtu dat z IoT zařízení. Tyto technologie nám umožňují shromažďovat, kontrolovat a analyzovat data v reálném čase [29, 33].

Internet věcí je jednou ze základních součástí konceptu chytrých měst [22, 39] a lze se s ním setkat v různých odvětvích, a to například:

[2]GIS – Geografický informační systém.
[3]POI – *Point of Interest* neboli bod zájmu.

- **chytré zemědělství** – v zemědělství se lze setkat s použitím senzorů ať už u rostlin, tak i na zemědělské půdě. Následně tak může být získána jejich prosperita nebo úpadek a mnoho dalších parametrů.
- **chytré zdravotnictví** – lze sledovat dostupnost a kvalitu na poli zdravotnictví. Na základě zařízení v IoT a mobilních zařízení je možné sledovat v reálném čase údaje, které mohou pomoci při identifikaci různých zdravotních problémů.
- **chytrá domácnost** – má pro lidi mnoho možností jak využívat technologie, ke kterým není potřebná odborná montáž, a z toho důvodu je přístupná mnoha lidem. V chytré domácnosti se mohou nacházet chytré žárovky, sledování spotřeby energií apod.
- **chytrý průmysl** – v moderním průmyslu jsou prvky IoT implementovány do jednotlivých přístrojů. Na základě získaných dat lze například lépe optimalizovat výrobní linky nebo za pomocí umělé inteligence zvyšovat podíl automatizace procesů.
- **chytrá doprava** – může se týkat jak hromadné dopravy, tak i jednotlivých aut. Lze např. sledovat vytíženost silnic, znečistění nebo obsazenost parkovacích míst.

Dalšími odvětvími mohou být chytrá infrastruktura města, energetika, služby atd.

Ve spojitosti s těmito odvětvími lze identifikovat různé typy senzorů [39]: **ambientní** (teplota, vlhkost apod.), **bio** (EEG, krevní tlak, puls atd.), **chemické** (detekce např. CO_2, kvalita vzduchu apod.), **elektrické** (spotřeba elektřiny), **hydraulické** (detekce úniků, průtok kapalin apod.), **pohybové/přítomnostní** (např. detekce pohybu na základě PIR senzoru[4]) a mnoho dalších druhů senzorů.

2.3 Servisní požadavky v chytrých městech

Ve městech mohou vznikat různé požadavky. Tyto požadavky mohou vznikat automaticky, resp. jsou vytvořeny IoT zařízením, nebo jsou zadány ručně, např. skrze webový portál nebo mobilní aplikaci. Vytvořit ručně je mohou i obyvatelé města, kteří chtějí upozornit na nějakou skutečnost, jež se týká fungování města. Mezi automaticky vytvářené lze zařadit např. nefunkční pouliční osvětlení, indikace překročení plnosti odpadkových košů a podobné senzory, které se mohou nacházet ve městě. Mezi manuální požadavky lze zařadit např. požadavek na ořezání stromů, opravu veřejné věci, údržbu ulice apod.

Záznam servisního požadavku by měl obsahovat alespoň tyto informace [30]: identifikátor, datum založení požadavku, datum uzavření požadavku, status, popis, kategorii (případně podkategorii), adresu a souřadnice. Do správy tohoto systému města ještě vstupuje správce, pod kterého požadavek patří a který bude zodpovídat za jeho splnění, tato osoba taktéž může určit prioritu požadavku a osobu, která daný požadavek bude řešit.

Osoby zabývající se splněním požadavků mají během své pracovní doby mnoho požadavků k řešení. Je tedy důležité, aby k řešení docházelo systematicky, a to zejména na základě priority a času, do kdy by měl být požadavek vyřešen. K tomu je zapotřebí správně požadavky řadit a optimalizovat cestu pracovníka, aby jeho pracovní doba byla co nejvíce využita a bylo splněno co nejvíce požadavků. Optimalizaci cesty a problémů spojených se směrováním vozidel bude věnována následující kapitola 3.

Implementaci servisních požadavků lze vidět v USA a Kanadě, kde se nachází linka *311* [41], která slouží pro nenouzové požadavky na službu a je opakem linky *911*. Tato služba

[4]PIR senzor – je senzor na bázi detekce infračerveného světla.

je dostupná v mnoha městech a slouží například pro ohlašování problémů nebo získání potřebných informací o službách města. Jedno z měst, které tuto službu implementuje, je Boston[5] a službu zpřístupňuje prostřednictvím hovoru, mobilní aplikace, Twitteru a webu.

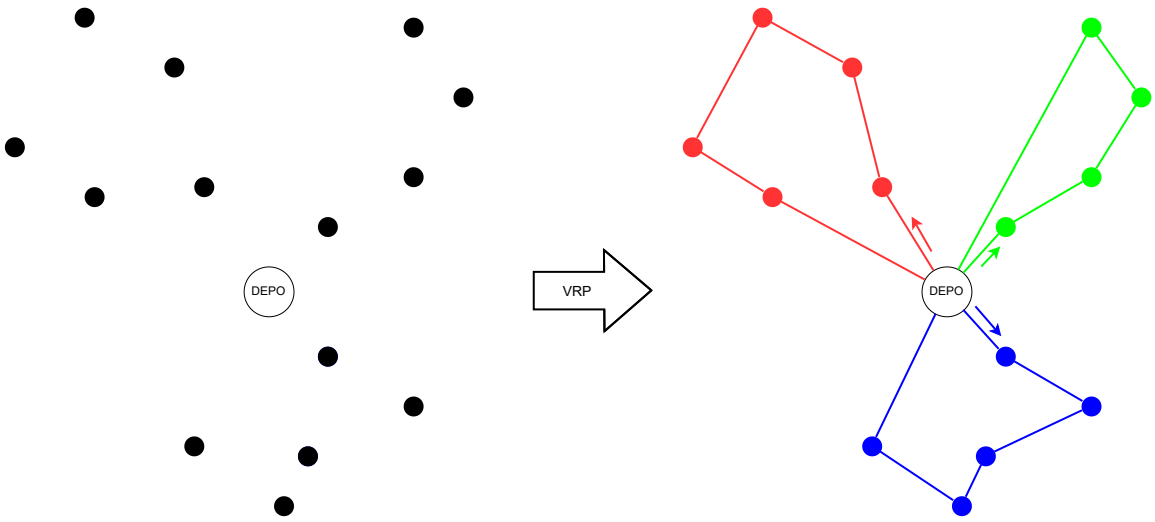
Jako názorný příklad lze uvést vědeckou studii [30] zabývající se servisními požadavky (line *311*) v kanadském městě Edmonton, jež mělo v roce 2014 877 926 obyvatel [1] a v němž bylo mezi roky 2013 až 2015 nahlášeno 178 691 požadavků. Lze říci, že toto číslo reprezentuje průměrně 163 požadavků denně. Pro lepší představu je dále číslo převedeno vzhledem k počtu obyvatel. Po přepočtu se lze dostat na 19 požadavků denně na 100 000 obyvatel již kolem roku 2014, proto je pro dnešní dobu nutné počítat s mnoho procentním nárůstem.

[5] https://311.boston.gov/

Kapitola 3

Problém směrování vozidel

Jedná se o kombinatorický problém optimalizace trasy, v němž se nachází body (lokace), které reprezentují činnost, a to ať dodání, vyzvednutí nebo například výkon servisního úkonu. Dále zde figurují vozidla, jež mají danou počáteční a cílovou lokaci, nebo jsou všechna umístěna v depu, odkud jsou vysílána. Na obrázku 3.1 lze vidět optimalizaci několika bodů a jednoho depa, které má dostupná tři vozidla, a to vzhledem k co nejmenší celkové ujeté vzdálenosti všech vozidel. Danzig a Ramser [12] definovali již v roce 1959 *Truck Dispatching Problem*, ve kterém je řešeno optimální nasměrování vozidel na dodávku benzínu mezi velkokapacitním zásobníkem a jednotlivými čerpacími stanicemi. Následně v roce 1964 Clarke a Wright [32] problém zobecnili pro použití v dopravě a logistice, díky čemuž se tento problém stal známým jako *Vehicle Routing Problem (VRP)* neboli problém směrování vozidel. Problém směrování vozidel se opírá o problém obchodního cestujícího (*Traveling Salesman Problem (TSP)*), který byl definován Dantzig, Fulkerson a Johnson již v roce 1954 [11]. Jedná se o kombinatorický optimalizační problém, který má za úkol nalezení nejkratšího propojení všech bodů v grafu.

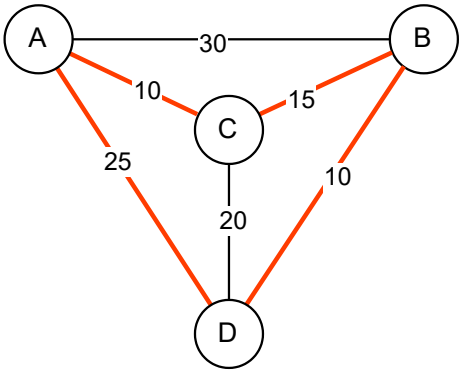


Obrázek 3.1: Ukázka VRP s jedním depem a třemi vozidly.[1]

[1] Převzato z: https://upload.wikimedia.org/wikipedia/commons/9/94/Vrp_esquema.png

3.1 Problém obchodního cestujícího

Tento problém je v angličtině označován jako *Traveling Salesman Problem (TSP)* a zaobírá se obchodním cestujícím, který chce navštívit všechna města, a to nejkratší možnou cestou. Jedná se o grafový problém, ve kterém jsou vrcholy města a neorientované ohodnocené hrany jsou cesty mezi městy. Pokud je graf úplný, tak je vždy možné v něm najít Hamiltonovskou kružnici a je ji možné sestavit $(n-1)!/2$ způsoby, kde n je počet vrcholů. Toto řešení má tedy složitost $O(n!)$ a řadí se mezi polynomiální složitost. V důsledku toho je tento problém řazen mezi NP-úplné, tzn. nelze ho řešit v lepším než polynomiálním čase, a pokud by někdo našel lepší řešení, mělo by to za důsledek to, že by bylo možné najít řešení i pro ostatní problémy v této třídě, a to za pomoci redukce [14]. Na obrázku 3.2 lze vidět řešení TSP mezi čtyřmi městy.



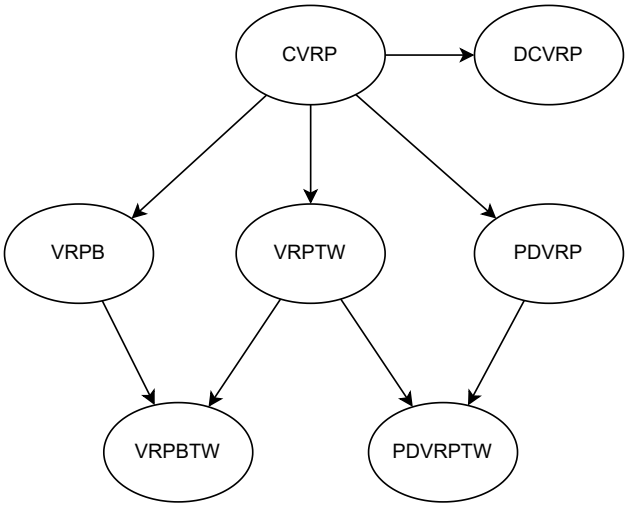
Obrázek 3.2: Graf o čtyřech městech (řešení TSP: $A \to C \to B \to D \to A$).

Modifikací TSP lze dospět k různým variantám [13, 25] a některé z nichž zde budou představeny. Nejprve je potřeba definovat graf $G = (V, E)$, kde V jsou vrcholy (města), E jsou hrany (cesty), přičemž $E = \{(a, b) : a, b \in V\}$ a d_{ab} specifikují vzdálenost mezi body, kde $(a, b) \in E$. Graf G nemusí být úplný a na základě toho může docházet k tomu, že některým vrcholem bude potřeba projít vícekrát pro dosažení nejkratší Hamiltonovské kružnice. **sTSP** *symmetric* nebo také **gTSP** *graphical* je varianta, při které se jedná o symetrické vyjádření hran, tzn. $d_{ab} = d_{ba}$, a zaměřuje se na hledání nejkratší cesty. **aTSP** *asymmetric* má stejný cíl jako symetrické, ale graf může obsahovat hrany, které $d_{ab} \neq d_{ba}$. **mTSP** *multi-salesman* obsahuje více než jednoho obchodníka, zde se nejkratší vzdálenost kalkuluje jako suma vzdáleností jednotlivých cest obchodníků. Tuto variantu lze rozdělit na dva různé pohledy, a to, že obchodníci začínají ve stejném bodě, nebo každý začíná v jiném bodě.

S TSP se lze setkat v různých aplikacích, a to např. vrtání tištěných desek, použití rentgenu v krystalografii, vybírání balíků ve skladech, směrování vozidel, plánování pohovorů apod. [13]

3.2 Problém směrování vozidel

Vehicle Routing Problem (VRP) je generalizací problému obchodního cestujícího a je spojován hlavně s logistikou a plánováním cest různých dopravních prostředků. Dále VPR se může dělit na podproblémy, které jsou hierarchicky řazeny (viz hierarchie 3.3).



Obrázek 3.3: Hierarchie podproblémů VRP.

CVRP

Capacited VRP [4] se zabývá nalezením optimální cesty pro jedno nebo více vozidel, která jsou vysílána z jednoho místa a mají předem určenou jednotnou kapacitu. Úkolem je minimalizace trasy tak, aby nebyla překročena kapacita vozidla a byly obslouženy všechny body (zákazníci).

VRPB

VRP with Backhauls [21] je problém, kde se vyskytují body doručení, tzv. *Linehauls*, a body vyzvednutí, tzv. *Backhauls*, a jeden bod skladu. Tento problém je zaměřen na hledání cesty tak, aby byly nejdříve obslouženy všechny body doručení a až poté všechny body vyzvednutí. Tento předpoklad vychází z toho, že pokud by byly body řazeny kombinovaně, tak by muselo dojít k přeuspořádání zboží ve vozidle, což je považováno za ekonomicky náročné nebo neproveditelné. Je zde využíváno problému CVRP a můžeme se setkat i s kombinací s VRPTW.

VRPTW

VRP with time windows obsahuje časová okna, kdy je možné daný úkol splnit. Je rozšířením podproblému CVRP. Dle zvoleného modelu pak lze určit, jaké preference jsou důležité a které méně. Například model uvedený v jednání [43] má za úkol minimalizaci celkových nákladů za předpokladu minimalizace množství vozidel, tzn. delší cesta může být výhodnější než vypravit nové vozidlo za splněním tohoto úkolu.

PDVRP

Pickup and Delivery VRP [34] je rozšířením problému *Pickup and Delivery Problem (PDP)* [37] a lze ho rozdělit do dvou podproblémů. První je rozvoz, nebo svoz zboží na jedno určené místo (např. sklad). Druhým podproblémem je seskupení úkolů, které jsou definovány vyzvednutím zboží na určeném místě a složením zboží na jiném místě. V obou problémech lze rozlišit dvě situace: (1) jedno místo vyzvednutí, jedno místo doručení, (2) jedno

místo vyzvednutí, více míst doručení. PDVRP je postaveno nad CVRP a často se vyskytuje ve spojení s VRPTW.

Dále se lze setkat s *Heterogenous VRP (HVRP)*, *Multi-depot VRP (MDVRP)* a různými kombinacemi výše uvedených podproblémů, a to s: *Distance-constrained capacitated VRP (DCVRP)*, *VRP with Backhauls and time windows (VRPBTW)* a *Pickup and Delivery VRP with time windows (PDVRPTW)*. Přehled dalších možných podproblémů VRP je možné najít v přehledu literatury od Kris Braekers et al. [9]. Lze zde také zahrnout problém *The team orienteering problem (TOP)* [10] a jeho kombinace s časovými okny nebo kapacitou. Tento problém se zaobírá problematikou orientačního běhu.

3.3 Metody řešení

Jak je již zmíněno výše, tak TSP/VRP má polynomiální časovou náročnost výpočtu, a proto existuje více různých metod k výpočtu tohoto problému. Metody je možné klasifikovat dle přístupu: *přesné* a *aproximační*. Metody lze rozřadit do tří skupin: *exaktní*, *heuristiky* a *meta-heuristiky*, přičemž heuristiky a meta-heuristiky řadíme mezi aproximační přístupy.

V souvislosti s různými metodami řešení je potřeba definovat formální model VRP. Ve většině publikací se lze setkat s matematickou formulací problému. Matematickou formulaci můžeme rozdělit na dvě části. První z nich je minimalizační výraz, v němž se nachází výraz, který definuje, co má být předmětem minimalizace. Druhá část jsou ostatní podmínky, kde jsou obsaženy další definice problému a omezující podmínky. V článku od Zirour et al. [44] je možné najít jednotlivé matematické formulace nejběžnějších problémů (PDVRP, VRPTW, CVRP).

V následující části jsou rozebrány různé metody řešení pro obecné TSP/VRP. Tyto metody jsou obecné a mohou být použity i při řešení komplikovanějších typů TSP/VRP za předpokladu zohlednění daných podmínek problému v metodách.

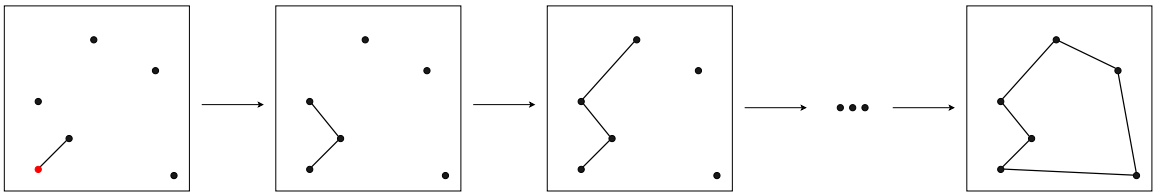
Exaktní metody

Tyto metody získávají a garantují optimální řešení. V důsledku toho, že najdou vždy optimální řešení, je potřeba počítat s větší časovou náročností výpočtu. Z toho důvodu jsou tyto metody určeny především pro málo obsáhlé problémy. Taktéž se s nimi lze setkat u heuristické metody *Local search*, kde mohou provádět některé části výpočtu. S exaktními metodami se můžeme například setkat v *Branch and Bound/Cut/Price* algoritmech, lineárním programování, dynamickém programování apod. [24] V publikaci od Gilbert Laporte [28] je možné najít upravené modely ve formě matematických formulací pro obecné řešení VRP a různé exaktní algoritmy.

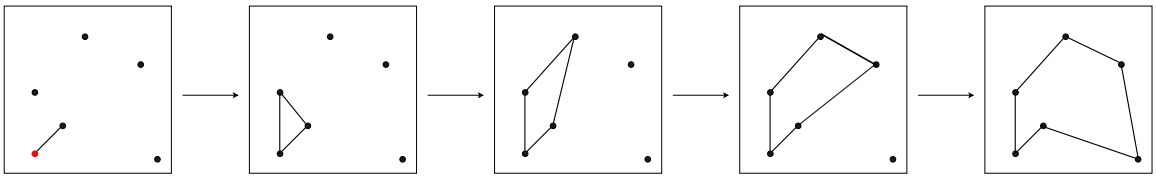
Heuristické metody

Heuristické metody fungují na principu aproximace a jejich řešení tedy nemusí být stoprocentně přesné nebo optimální. Používají se v případech, kdy je požadováno urychlení výpočtu a kde jsou exaktní metody příliš časově náročné nebo neřešitelné v rozumném časovém horizontu. Heuristiky jsou vždy založeny na nějaké myšlence výpočtu, který vede k uspokojivému výsledku, přičemž neprohledávají všechna možná řešení, ale pouze jejich část. Mezi heuristické metody lze zařadit *Nearest Neighbor/Insertion*, *k-Optimal algorithm*, *Savings algorithm* a mnoho dalších.

Nearest Neighbor/Insertion – jedná se o jedny z nejjednodušších metod. Jsou postaveny na základě vyhledávání nejbližších sousedících bodů. *Nearest Neighbor* [35] začíná prohledávat od startovního bodu a vždy vybere nejbližší bod, tak postupuje, dokud všechny nespojí (viz 3.4). *Nearest Insertion* [23] funguje na principu kliky grafu, tzn. do kliky je vždy přidán nejbližší bod a je začleněn do hrany, která je bodu nejblíže. Končí pokud se z kliky stane Hamiltonovská kružnice (viz 3.5).

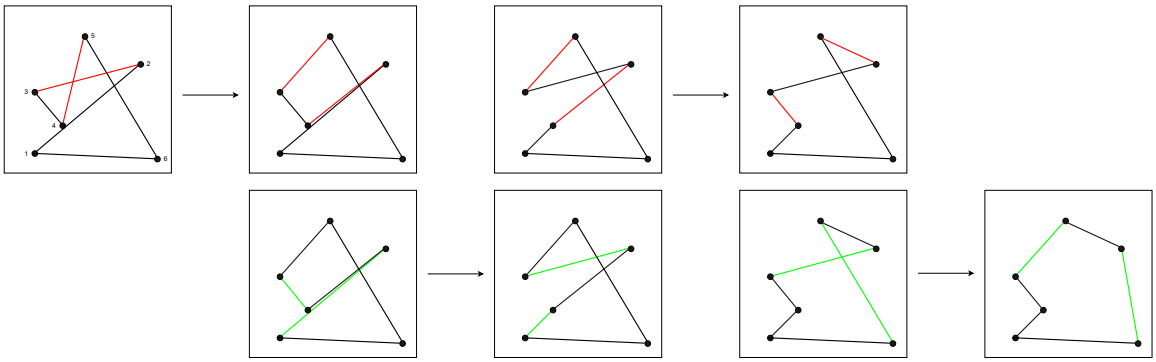


Obrázek 3.4: Vizualizace metody *Nearest Neigbor.*



Obrázek 3.5: Vizualizace metody *Nearest Insertion.*

k-Optimal algorithm [7, 15] – nejčastěji se lze setkat s *2-opt* nebo *3-opt* verzemi. Tato metoda je založena na výměně až k hran, kde se porovnává cena hran před změnou a po změně. Na začátku je vytvořena libovolná Hamiltonovská kružnice, nebo lze vyžít například *Nearest Insertion* metody pro její vytvoření. Na vytvořené kružnici jsou vyměňovány hrany. Pokud nelze již žádných k hran vyměnit tak, že by měly nižší ohodnocení, je získáno výsledné řešení. Vizualizaci algoritmu ve verzi *2-opt* je možné vidět na ilustraci 3.6.



Obrázek 3.6: Vizualizace metody *2-opt algorithm.*

Savings algorithm [32] je jeden z nejznámějších heuristických algoritmů pro VRP. Byl definován Clarkem a Wrightem v roce 1964. Algoritmus funguje na principu slučování hran, a to tím způsobem, že pokud lze najít dvě hrany, které mohou být sloučeny, dojde k jejich sloučení, čímž dochází k získání nové hrany, která má nižší ohodnocení než součet ohodnocení původních hran.

Meta-heuristické metody

Jedná se o pokročilejší přístupy oproti heuristickým metodám a obecně generují lepší výsledky, ale mají stejná omezení jako heuristické metody, a to, že nemusí vždy vést k nejoptimálnějšímu řešení. Většinou jsou meta-heuristické metody popsány pomocí stochastických algoritmů, což znamená, že se v algoritmu mohou objevovat různé druhy náhodnosti [17]. Lze sem zařadit metody jak inspirované přírodou, a to ať inspirované fyzikálními zákony (např. simulované žíhání) nebo inspirované evolucí (např. genetikcé/evoluční algoritmy), tak i neinspirované přírodou (např. *Tabu search*) [8]. Níže jsou představeny dvě metody, a to *Tabu search* a *Ant colony optimization*.

Tabu search (TS) [5] se řadí mezi iterativní metody. Algoritmus funguje na principu prohledávání širšího okolí, aby se vyhnulo uvíznutí v lokálním optimu. Z tohoto důvodu může dojít i k vybrání takového souseda, který nemusí být nejoptimálnější z pohledu optimalizačního parametru. V každé iteraci dojde k vygenerování okolí a je vybráno nejlepší řešení, a to je vloženo do množiny *tabu* (tzn. zakázaná množina), aby se zamezilo cyklení, a do množiny aktuálního řešení. Aktuální řešení je dále porovnáno s globálním řešení, pokud je aktuální lepší, tak se stává globálním. Algoritmus iteruje dokud nekonverguje ke globálnímu nejlepšímu řešení.

Ant colony optimization (ACO) [6, 38] je metoda inspirovaná chováním mravenců v přírodě. Mravenci po cestě vylučují feromony a čím více jich projde stejnou cestou, tak tím na ní zanechají více feromonů, a tím se pro ně stává atraktivnější. V kontextu TSP/VRP jsou na začátku mravenci vsazeni do různých vrcholů. Poté každý mravenec prochází vrcholy na základě předem daných pravidel (např. nesmí navštívit stejný bod dvakrát, preferují kratší cestu do dalšího vrcholu apod.) a při své cestě zanechávají na cestě feromony, dokud neprojdou všechny vrcholy. Tento postup se opakuje několikrát, přičemž po každé iteraci se snižuje koncentrace feromonů na jednotlivých cestách. V důsledku toho nám po několika interacích vzniká jedna cesta s největším počtem feromonů, která určuje finální řešení.

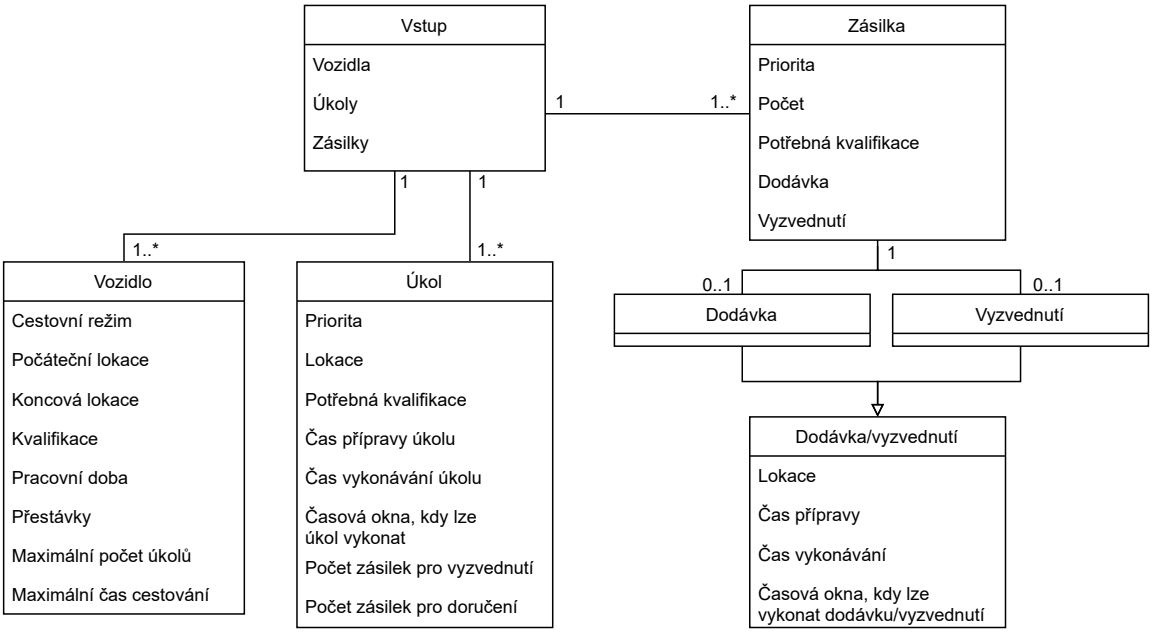
Kapitola 4

Dostupná webová řešení VRP

Tato kapitola je zaměřena na přehled dostupných webových řešení (API), která umožňují řešit VRP a jeho modifikace. Je zde uvedeno několik nejznámějších platforem/služeb. Dále se kapitola zabývá vhodným výběrem platformy pro určité modelové situace.

4.1 Definice vstupních a výstupních dat

V následujících zjednodušených grafech je znázorněna struktura vstupních (obr. 4.1) a výstupních (obr. 4.2) dat. Jedná se o souhrnnou šablonu všech služeb, nicméně není nijak závazná a služby se mohou lišit v její implementaci a mohou obsahovat některé informace navíc, případně informace i postrádat. U všech služeb jsou tato data uváděna ve formátu JSON.

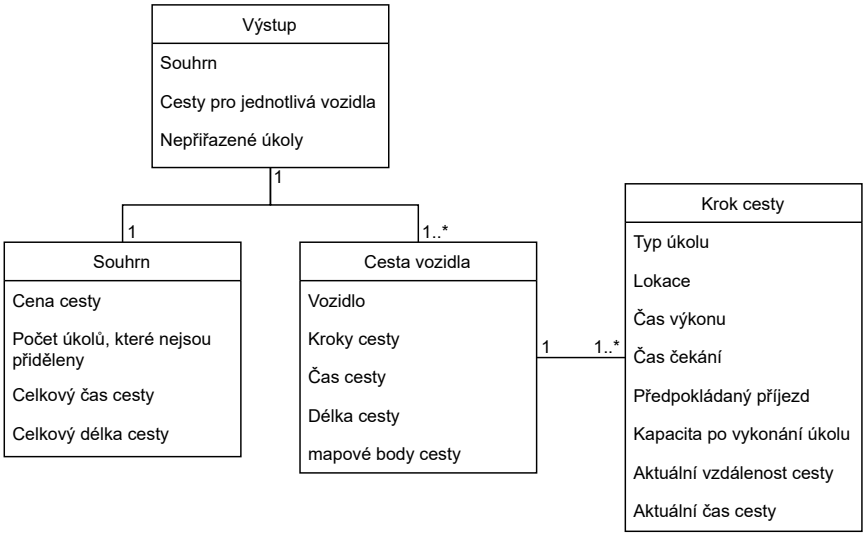


Obrázek 4.1: Vizualizace vstupních dat.

Hlavní část vstupních dat (obr. 4.1) je členěna na tři části:

- **vozidla** – jsou v nich obsaženy rámcové informace o daném vozidle/pracovníkovi.
- **úkoly** – popisují možné zastávky vozidel, přičemž obsahují náležité informace (lokace, priorita, čas trvání, apod.).
- **zásilky** – fungují podobně jako úkoly, ale nepopisují činnost, ale pouze dodávku, nebo vyzvednutí zásilky, ty jsou definovány podobně jako úkoly.

Výstupní data (obr. 4.2) jsou členěna na dvě části. První část obsahuje obecné a souhrnné informace pro všechny cesty vozidel. Druhá část obsahuje cesty vozidel, které jsou tvořeny jednotlivými kroky cesty, a ty dále obsahují informace o daném úkolu nebo zásilce ze vstupních dat.



Obrázek 4.2: Vizualizace výstupních dat.

4.2 Přehled dostupných řešení

V této části jsou rozebrány jednotlivé služby zabývající se řešením VRP a jeho modifikacemi. V popisu jednotlivých služeb bude posouzeno pokrytí různých modifikací VRP, škálovatelnost, kvalita dokumentace a ekonomická stránka služby.

Pro všechny zmíněné služby je potřeba počítat s občasnými odchylkami vůči reálnému světu. Tímto je myšleno, že služba může nabídnout cestu, která je v realitě nemožná z důvodů například jednosměrné silnice, pěší zóny apod. Toto je způsobeno odlišnými mapovými podklady, které služby používají, a v různých státech se mohou více či méně tyto anomálie objevovat.

4.2.1 Geoapify

Firma Geoapify nabízí službu *Route Planner API*[1], taktéž i ostatní mapové API jako geokódování, trasy apod. Služba je poskytována skrze webové API. Služba podporuje všechny typy modifikací VRP a v tabulce 4.1 jsou uvedeny ostatní podporované parametry.

[1] https://www.geoapify.com/route-planner-api

Přestávky	Pracovní doba	Priority	Zkušenosti	Teritoria	Vztahy bodů
✓	✓	✓	✓	×	×

Tabulka 4.1: Geoapify – Další parametry.

Platforma funguje na kreditním systému, což znamená, že každý den dochází k obnově celkového počtu kreditů dle zvoleného plánu. Jednotlivé služby (API) platformy mají svá kreditní ohodnocení, tzn. je definován výpočet, který udává, kolik kreditů stojí jeden požadavek na danou službu. V případě *Route Planner API* se cena odvíjí od počtu lokací v jednom požadavku (dle tabulky 4.2). Dále API neobsahuje další restrikce požadavků.

Počet lokací	Výpočet ceny kreditů
≤ 10	`počet lokací` $\times$ `počet lokací`
>10	`počet lokací` $\times 10$

Tabulka 4.2: Geoapify – Výpočet ceny kreditů jednoho požadavku na *Route Planner API.*

V tabulce A.1 se nachází jednotlivé cenové plány platformy, přičemž platforma nabízí i roční účtování, které je výhodnější o ≈ 16 %. Všechny placené plány obsahují SLA s pokrytím $99,5$ %.

Služba má velkou škálovatelnost díky kreditnímu systému ceníku a absenci dodatečných restrikcí. Platforma nabízí dokumentaci, která je poměrně stručná, ale obsahuje všechny potřebné náležitosti k práci se službou.

4.2.2 Graphhopper

Firma *Graphopper* se specializuje na mapové API jako geokódování, trasy nebo optimalizace trasy. Dále je podporovatel *open source* projektu *OpenStreetMap*[2]. Taktéž poskytuje službu *Route Optimization API,*[3] což je webové API zabývající se VRP. Služba podporuje všechny typy modifikací VRP a v tabulce 4.3 jsou uvedeny informace dalších parametrů.

Přestávky	Pracovní doba	Priority	Zkušenosti	Teritoria	Vztahy bodů
✓	✓	✓	✓	×	✓

Tabulka 4.3: Graphhopper – Další parametry.

Mezi další funkce lze zařadit možnost specifikace průjezdu bodu zprava/zleva vůči řidiči vozidla, zvážení dopravního vytížení, možnost výběru cíle algoritmu (např. minimalizace počtu vozidel, doba přepravy atd.), nebo specifikace vlastní matice časů a vzdáleností mezi body.

[2] https://www.openstreetmap.org/
[3] https://www.graphhopper.com/route-optimization/

Platforma funguje na kreditním systému a ceník plánů (viz tabulka A.2) je pro platformu jednotný. Kredity jsou obnovovány na denní bázi. V jednotlivých plánech dochází k limitacím, ať už maximálního počtu vozidel, nebo lokací, v rámci jednoho požadavku. Měsíční účtování oproti ročnímu vychází o ≈ 19 % levněji. Každý plán obsahuje SLA s pokrytím $99,5$ %. Služba *Route Optimization API* je kreditově ohodnocena pro jeden požadavek dle výpočtu:

$$\texttt{počet vozidel} \times \texttt{počet lokací}.$$

Služba je díky kreditnímu systému do jisté míry dobře škálovatelná, avšak (1) standardní plány obsahují limitace na počet vozidel a lokací, (2) výpočet ceny kreditů je závislý na dvou veličinách. Platforma nabízí uspokojivou dokumentaci, která je především orientována na jednotlivé parametry požadavků.

4.2.3 Openrouteservice

Jedná se *opensource* projekt spravovaný skupinou *Heidelberg Institute for Geoinformation Technology (HeiGIT)*. Jedná se o projekt poskytující mapové služby jako výpočet tras, geokódování, hledání bodů zájmů apod. Platforma nabízí webové API, ale taktéž i *Python, R* a *JavaScript SDK*. Taktéž nabízí službu *Optimization*[4], která nabízí řešení VRP. Tato služba je implementována *opensource* projektem *VROOM-Project*[5]. Služba nabízí řešení všech modifikací VRP s výjimkou VRPB a přehled dalších podporovaných parametrů nalezneme v tabulce 4.4.

Přestávky	Pracovní doba	Priority	Zkušenosti	Teritoria	Vztahy bodů
✓	✓	✓	✓	×	×

Tabulka 4.4: Openrouteservice – Další parametry.

Platforma nabízí tři druhy plánů (viz tabulka 4.5), které jsou zdarma, ale jsou odlišně ohodnoceny denními/minutovými dotazovými limity (viz A.6) na jednotlivé služby a podmínkami použití. Služba pro optimalizaci VRP v plánech *Standart* a *Collaborative* limituje počet vozidel na 3 a počet lokací na 50 v rámci jednoho požadavku. Platforma negarantuje SLA ani podporu.

Název plánu	Popis
Standard	volné použití pro všechny
Collaborative	slouží pro akademické, humanitární, vládní a nevýdělečná použití
On-Premise	vlastní instalace platformy

Tabulka 4.5: Openrouteservice – Popis plánů.

Platforma ve využití formou dostupného API není nijak škálovatelná. S velkou škálovatelností se setkáváme u vlastní instance platformy, kde je možné platformu nastavit

[4] https://openrouteservice.org/services/
[5] https://github.com/VROOM-Project/vroom

dle vlastních požadavků a jedinou limitací je zde server, na němž bude služba spuštěna. Projekt zabývající se VRP obsahuje stručnou dokumentaci, která je orientována na popis struktur, a jednoduchý popis fungování jednotlivých objektů.

4.2.4 Routific

Platforma *Routific* se zaměřuje pouze na optimalizační API *Routific [Engine]*[6] a vlastní aplikaci pro plánování rozvážky. Nabízí řešení všech modifikací VRP s výjimkou VRPB a možnosti dalších parametrů, viz 4.6.

Přestávky	Pracovní doba	Priority	Zkušenosti	Teritoria	Vztahy bodů
✓	✓	✓	×	×	×

Tabulka 4.6: Routific – Další parametry.

Platforma funguje na principu účtování dle počtu lokací za měsíc, který je dán cenovým plánem, viz A.3, přičemž pokud je stejná lokace zadána vícekrát v rámci 24 hodin, tak je účtována pouze jedenkrát. Maximální počet lokací nebo aut není nijak omezen. Součástí plánu *Enterprise* je i SLA s minimální hodnotou 99 %, přičemž *Routific* musí odsouhlasit, že zákazníkovi poskytne SLA.

Služba nabízí velkou škálovatelnost pouze za předpokladu vysokého procenta stejných adres v rámci jednoho dne. Platforma nabízí kvalitní dokumentaci s rozsáhlejšími popisy jednotlivých vstupních parametrů.

4.2.5 Here

Platforma *Here* se zabývá mapovými službami, mezi něž patří například plánování cesty, trasování, zjišťování polohy uvnitř objektu a optimalizace cesty. Služba *HERE Tour Planning*[7] poskytuje webové API pro řešení VRP. Služba nabízí řešení všech modifikací VRP s výjimkou VRPB a doplňujících parametrů, viz tabulka 4.7

Přestávky	Pracovní doba	Priority	Zkušenosti	Teritoria	Vztahy bodů
✓	✓	✓	✓	✓	×

Tabulka 4.7: Here – Další parametry.

Platforma nabízí ohodnocení každé služby zvlášť, a to skrze počet transakcí za měsíc. U služby *HERE Tour Planning* se počet transakcí vypočítá následujícím výpočtem:

$$\sum \texttt{lokace v~požadavku.}$$

Pro službu platí cenové ohodnocení dle tabulky A.4. Dále platforma nabízí plány podpory samostatně od účtování jednotlivých služeb. Nejdůležitější položky plánu podpory jsou

[6] https://dev.routific.com/

[7] https://developer.here.com/products/tour-planning

uvedeny v tabulce A.5, přičemž SLA je definované pro každou službu zvlášť a konkrétně pro *HERE Tour Planning* je $98,5$ %.

Služba díky měsíčnímu účtování transakcí nenabízí velkou škálovatelnost, avšak počet transakcí je pouze závislý na počtu lokací. Platforma nabízí kvalitní dokumentaci, přičemž popis struktur je stručnější, avšak dále obsahuje rozsáhlý počet ilustrativních řešení s popisem uvádějícím do problematiky.

4.2.6 Shrnutí

V následujících tabulkách je možné vidět porovnání jednotlivých služeb, a to: (1) ekonomicky nebo dle omezení (viz 4.8), (2) dle podporovaných modifikací VRP (viz 4.9), (3) dle podporovaných parametrů (viz 4.10).

Služba	Systém účtování	Obnova (transakcí/kreditů)	Omezení počtu vozidel	Omezení počtu lokací
Geoapify	kreditní	denní	×	×
Graphhopper	kreditní	denní	✓	✓
Openrouteservice	transakce (požadavky)	denní	✓	✓
Openrouteservice (On-Premise)	-	-	×	×
Routific	transakce (lokace)	měsíční	×	×
Here	transakce (lokace)	měsíční	×	×

Tabulka 4.8: Shrnutí služeb.

Služba	TSP	CVRP	VRPB	VRPTW	PDVRP	HVRP	MDVRP
Geoapify	✓	✓	✓	✓	✓	✓	✓
Graphhopper	✓	✓	✓	✓	✓	✓	✓
Openrouteservice	✓	✓	×	✓	✓	✓	✓
Routific	✓	✓	×	✓	✓	✓	✓
Here	✓	✓	×	✓	✓	✓	✓

Tabulka 4.9: Shrnutí podporovaných modifikací služeb.

Služba	Přestávky	Pracovní doba	Priority	Zkušenosti	Teritoria	Vztahy bodů
Geoapify	✓	✓	✓	✓	×	×
Graphhopper	✓	✓	✓	✓	×	✓
Openrouteservice	✓	✓	✓	✓	×	×
Routific	✓	✓	✓	×	×	×
Here	✓	✓	✓	✓	✓	×

Tabulka 4.10: Shrnutí parametrů služeb.

4.3 Zhodnocení služeb z ekonomického hlediska

V této části jsou služby zhodnoceny na základě ekonomického hlediska. To lze hodnotit jak na základě cenových plánů, které služby nabízí, tak i z pozice kalkulace ceny/kreditů. Způsob kalkulace je velmi důležitý parametr pro hodnocení, který může být závislý na některých parametrech jako počet vozidel a lokací v jednom požadavku a na počtu požadavků v rámci dne.

Služby, které jsou závislé na jakémkoliv parametru a mají měsíční účtování, se stávají méně výhodnými oproti službám s denním účtováním, pokud služba má být často využívána. Z pohledu denního účtování poté rozhoduje čistě způsob výpočtu ceny dotazu. Tyto služby je poté třeba rozlišovat na základě plánovaného způsobu využití. Pokud řešení vyžaduje optimalizaci pouze pro jedno vozidlo, tak se služby nijak neliší, protože v každé je kalkulováno s počtem lokací v požadavku (vyjma *Openrouteservice*). Pokud je ale vyžadováno více vozidel, tak je potřeba brát v potaz, že služby, které ve svém výpočtu násobí lokace počtem vozidel, tak cena za jeden požadavek může násobně vzrůst.

Demonstrace vývoje cen služeb

Na základě zhodnocení služeb v předchozí části byly definovány hlavní ekonomické rozdíly služeb, které mohou mít vliv na vývoj ceny. K tomu, aby bylo možné služby porovnat na základě jejich cenových plánů, tak je třeba stanovit hodnoty parametrů ovlivňující výpočet ceny. Ty jsou pro porovnání nastaveny tak, aby výpočet ceny nepřesahoval do cenového plánu, který je definován na míru zákazníkovi.

Z toho důvodu jsou situace rozděleny na dvě části: (1) s jedním vozidlem a (2) s deseti vozidly. Tyto hodnoty byly zvoleny tak, aby byly v co největší míře vidět rozdíly mezi použitím jednoho vozidla a více vozidel v jednom požadavku. Dále jsou pro obě situace rozlišeny počty lokací v jednom požadavku, a to: (1) patnáct, (2) padesát a (3) sto lokací v jednom požadavku, čísla jsou zvolena tak, aby bylo možné sledovat různá omezení jednotlivých plánů služeb (pokud nějaké omezení obsahují). V poslední řadě je sledován počet denních dotazů na službu.

Pro toto porovnání je potřeba službu *Openrouteservice* (sekce 4.2.3) rozdělit na dvě části, a to dle plánů: (1) *Standart* a (2) *On-Premise*. Pro (2) je potřeba definovat cenu a ta bude stanovena cenou hostingu *VPS*[8]. Dle systémových požadavků [2] pro celou planetu je zapotřebí, aby server obsahoval okolo 128 *GB* paměti RAM. Jako referenční služba posky-

[8]VPS – Virtuální privátní server.

tující VPS bude *VPSSERVER*[9] a konkrétně se bude jednat o server z kategorie *Memory optimized* a plán *VPS 128 GB* za $\doteq$ 380 €/měsíc.

Vyhodnocení demonstrace vývoje cen služeb

V příloze B lze najít odpovídající grafy k následujícímu zhodnocení ekonomické stránky služeb. V grafech sledujících situaci s jedním vozidlem (a max. 80 lokacemi) v požadavku, lze zahrnout i službu *Openrouteservice* v plánu *Standart*, která je v rámci sledovaného počtu denních dotazů taktéž vyhovující. Ostatní služby poskytující plány zdarma zde nejsou zahrnuty z důvodu limitovaného komerčního použití.

Z obou situací (1 a 10 aut) lze vidět, že hraje velkou roli, na jakém systému účtování služba funguje. Služby používající denní obnovu jsou výrazně ekonomicky výhodnější než služby s měsíční obnovou. Při zaměření na služby poskytující denní obnovu, tak lze pozorovat hlavní rozdíl služeb, a to ve výpočtu ceny jednoho požadavku. V případě použití více aut lze vidět výhodu služby *Geoapify*, jejíž výpočet se pouze odvíjí od počtu lokací v požadavku, na rozdíl od služby *Graphhopper*, která má ve výpočtu zahrnut i počet aut. U služby *Graphhopper* lze dále sledovat limitaci aut a lokací v rámci jednoho požadavku, kde i při nižších počtech dotazů je zapotřebí zvolení vyššího plánu, který je méně limitující.

Na základě těchto informací lze říci, že pokud je vyvíjena aplikace, která by měla zvládat větší počet denních požadavků, tak jsou výhodnější služby nabízející denní obnovu kreditů/transakcí, což jsou služby *Geoapify*, *Graphhopper* a *Openrouteservice*. V opačném případě, pokud jsou denní či týdenní počty požadavků v nízkých číslech, tak se mohou stávat výhodnější služby s měsíční obnovou kreditů/transakcí, a to především služba *Here*. Řešení od firmy *Routific* přichází poměrně se specifickým účtováním, kde je zapotřebí mít velké procento stejných adres v požadavcích v rámci dne. K předešlému vyhodnocení je potřeba následně zvážit konkrétní požadavky systému ať technické nebo obecné parametry služeb a zejména parametry jednotlivých plánů služeb.

[9] https://www.vpsserver.com/

Kapitola 5

Analýza

Tato kapitola obsahuje rozbor současného stavu, analýzu potřeb uživatelů, celkových požadavků na optimalizaci cesty a vznikajících servisních požadavků pro servisního pracovníka. Jednotlivé části jsou rozebrány v kontextu firmy Logimic[1], která vyvíjí platformu pro chytrá města.

Firma Logimic se zabývá vývojem a následnou implementací IoT zařízení do měst či různých objektů (areály, budovy apod.). Firma nabízí svou platformu řešení chytrých měst, pro kterou bude navrhnut a implementován systém pro plánování tras servisních pracovníků. *Logimic* poskytuje řadu produktů pro chytrá města, mezi něž patří například řízení veřejného osvětlení, měření kvality ovzduší, monitorování odpadu nebo parkovacích míst.

5.1 Současný stav

Platforma nabízí dva moduly, kde se lze setkat se servisními požadavky. Jeden z modulů je *RAH* neboli *Report A Hazard*. Tento modul především slouží pro širší veřejnost a je zaměřen na podání podnětu správci města k vybranému úkonu, a to např. o přeplněném odpadkovém koši, poškozené části chodníku apod. Druhým modulem jsou *servisní požadavky*, které jsou dále využívány správou města.

Servisní požadavky lze tvořit několika způsoby. Prvním z nich je prostřednictvím modulu *RAH*, a to tak, že ze záznamu *RAH* je vytvořen servisní požadavek, z něhož jsou předvyplněny některé hodnoty jako kategorie, podkategorie, detail, přiřazená osoba a priorita, a to za předpokladu, že byly vyplněny. Další možností je přímé vytvoření servisního požadavku, při kterém se na pozadí vytvoří záznam *RAH*, který je dále editovatelný. Rovněž je důležitý proto, že uchovává lokaci, kde má dojít k vykonání dané činnosti. V poslední řadě lze vytvořit servisní požadavek pro konkrétní zařízení. V tomto případě je potom lokace vykonání servisního úkonu určena polohou daného zařízení.

Technické řešení platformy

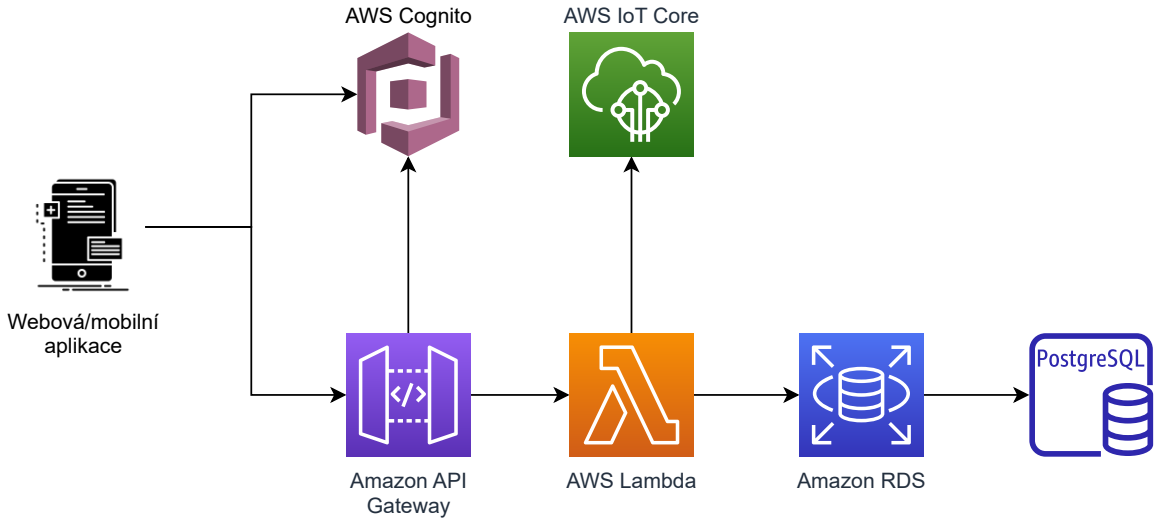
Logimic pro svou platformu chytrého města využívá cloud technologii (*AWS* neboli *Amazon Web Services*[2]) poskytovanou od firmy Amazon. Hlavní funkcí a základním pilířem platformy je služba *AWS Lambda*. Pro použití *AWS Lambda* je využíváno služby *Amazon API Gateway* [36], která slouží ke tvorbě *HTTP endpoints* pro jednotlivé funkce *AWS Lambda*. O zabezpečení a autentizaci uživatelů se stará služba *AWS Cognito*, která nabízí i použití

[1] https://www.logimic.com/
[2] https://aws.amazon.com/

přihlášení přes různé organizace (Google, Apple apod.). Dále jsou v systému využívány další služby *AWS*, a to například *AWS IoT Core*, *Amazon Cloud watch* nebo *Amazon S3*. Pro ukládání dat je využito databázových služeb *Amazon RDS* a Logimic využívá pro uchování dat relační databázi *PostgreSQL*. Na obrázku 5.1 lze vidět část systémové architektury, která je důležitá pro tuto práci.

AWS Lambda [3] je služba poskytující bez serverový (*serverless*) přístup k operacím. Na základě událostí/požadavků spouští jednotlivé funkce (části kódu, které definují funkce), přičemž služba již automaticky spravuje výpočetní zdroje za nás. Tento přístup dovoluje vypustit další náklady na správu vlastních serverů, a v důsledku automatické správy výpočetních zdrojů je toto řešení vysoce škálovatelné i na základě toho, že cena služby se odvíjí od výpočetního času, který jednotlivé funkce potřebovaly.

Amazon RDS (*Amazon Relational Database Service*) [31] je služba, která pracuje nad jednotlivými databázemi. Podporuje řadu databázových nástrojů, mezi které lze například zařadit migraci dat, replikaci, zálohování, obnovy i opravy databází. Službu lze napojit na šest typů databázových strojů: (1) Amazon Aurora (proprietární stroj od AWS, který podporuje MySQL a PostgreSQL), (2) MariaDB, (3) MySQL, (4) Oracle Database, (5) PostgreSQL a (6) SQL Server (Microsoft SQL Server).



Obrázek 5.1: Část architektury systému Logimic.

5.2 Definice uživatelů

V kontextu servisních požadavků se vykytují primárně dva typy uživatelů, a to (1) servisní pracovník a (2) vedoucí (*supervisor*). Popis jednotlivých uživatelů je zaměřen na systematiku plánování tras servisních požadavků.

- **Servisní pracovník** – disponuje určitými schopnostmi (opravář, údržbář apod.), je poskytován městem, nebo jinou organizací zajišťující servisní pracovníky. Jeho úkolem během pracovní doby je vykonávat přiřazené úkoly (servisní požadavky), ke kterým má přístup skrze aplikaci chytrého města. Úkoly mohou být zobrazeny jako výčet nebo v grafické podobě na mapě.

V souvislosti s plánováním trasy by měl mít možnost vybrat úkoly, které chce zahrnout do plánování, vybrat pracovní dobu, případně přestávky a počáteční úkoly, jež nemají být zahrnuty do optimalizace. Na základě těchto parametrů bude mít možnost vygenerovat optimální trasu, resp. optimální seřazení úkolů. Toto seřazení bude mít možnost dále editovat (verzovat), a to případným odebráním nebo přidáním úkolů.

- **Vedoucí** – jeho úkolem je řídit proces zpracování servisních požadavků (přiřazení požadavku servisnímu pracovníkovi, schválení ukončení apod.) a taktéž má zodpovědnost za jejich správné vyřešení.

  V kontextu plánování tras bude mít možnost naplánovat trasy více servisním pracovníkům současně, a to na základě informací o servisních pracovnících a schopnosti definující servisní požadavek. Dále by měl disponovat možností schvalování vygenerovaných tras a přiřazením servisních požadavků servisním pracovníkům. Schvalování tras zvyšuje kontrolu nad jednotlivými servisními pracovníky a jejich trasami, přičemž servisní pracovníci je mají taktéž možnost upravovat (verzovat). V poslední řadě by měl mít přístup k aktuálním i historickým trasám servisních pracovníků.

5.3 Požadavky na plánování trasy

Pro plánování trasy servisních požadavků je nezbytné určit hlavní parametry, dle kterých bude docházet k plánování trasy. Parametry lze určit na základě informací o servisním požadavku a servisním pracovníkovi.

Servisní požadavek obsahuje **(1) prioritu**, s níž je potřeba kalkulovat v rámci plánování trasy, tzn. požadavky s vyšší prioritou budou upřednostněny před požadavky s nižší prioritou. Na základě typu servisního požadavku je dále určena potřebná **(2) schopnost** pro její splnění.

Servisní pracovník má svou **(3) pracovní dobu** a **(4) přestávky**, které je potřeba zohlednit při plánování trasy. Taktéž je potřeba zohlednit jeho **(5) schopnosti**, aby nedošlo k tomu, že by trasa servisního pracovníka obsahovala nesplnitelné požadavky vzhledem k jeho schopnostem.

Trasa by měla být zkonstruována tak, aby byly všechny výše uvedené podmínky splněny a zároveň byla minimalizována vzdálenost trasy servisního pracovníka.

5.4 Požadavky na řešení

Platforma pro chytrá města od firmy Logimic nenabízí možnost plánování a správu tras jednotlivých servisních pracovníků. Z toho důvodu bude cílem navrhnout řešení, které tento nedostatek bude řešit. Řešení by mělo umožňovat následující body v souvislosti s definovanými uživateli v sekci 5.2 a definovanými požadavky na plánování trasy v sekci 5.3:

- definování servisního pracovníka.
- definování schopností pracovníků a typy servisních požadavků.
- vytvoření tras/y pro servisní/ho pracovníky/a ze servisních požadavků.
- možnost definovat pevný začátek trasy (počátečních několik bodů).
- možnost verzovat již vytvořené trasy.

- evidování vytvořených tras.
- schvalování vytvořených tras.

Dalším požadavkem je vybrání vhodné služby, která bude zajišťovat optimalizaci trasy. K tomuto tématu je již vypracován přehled dostupných řešení v kapitole 4. V důsledku toho, že doposud nebyly na platformě řešeny trasy servisních pracovníků, je potřeba z počátku počítat s menším vytížením služby a pokud možno co nejvíce snížit ekonomické náklady na tuto službu. V této situaci je potřeba počítat s možností, která umožňuje změnit poskytovatele služby pro optimalizaci, a to dle potřeby vytížení nebo z ekonomického hlediska. Tato možnost zahrnuje i mít možnost definování služby pro každého zákazníka platformy zvlášť.

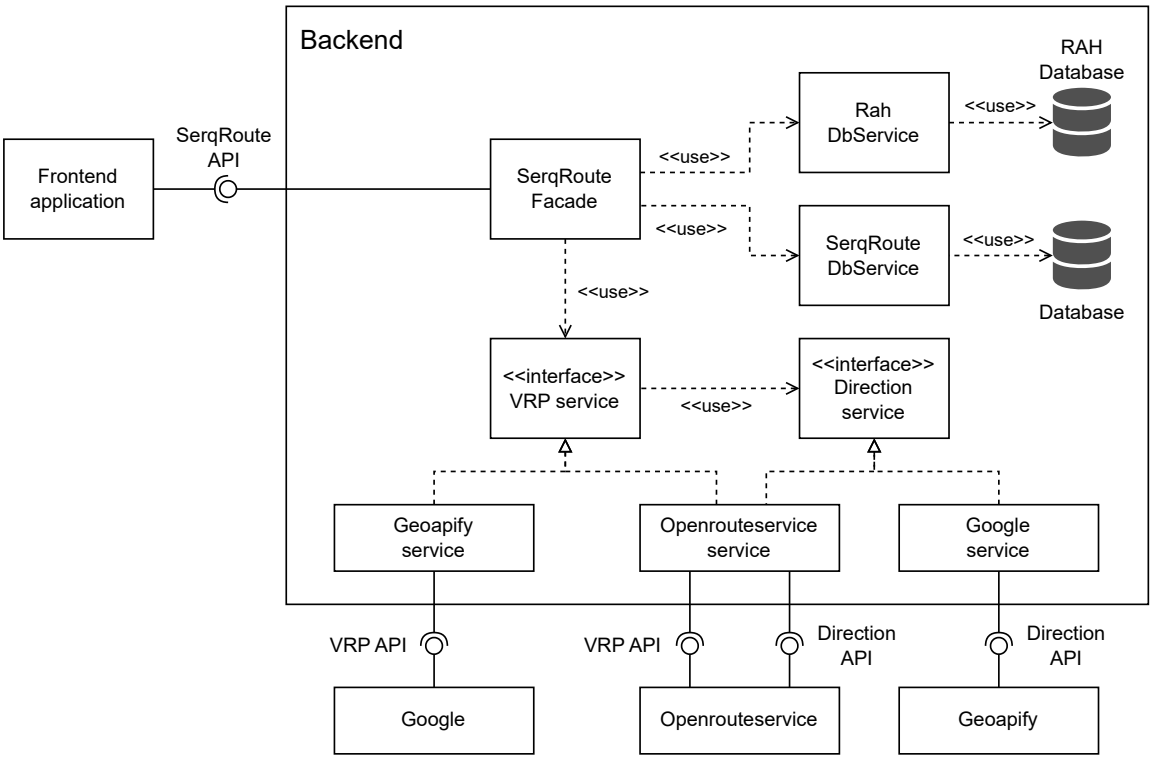
Kapitola 6

Návrh řešení

Tato kapitola je věnována návrhu řešení. V první části (sekce 6.1) je popsána architektura včetně dalších informací k řešení vyplývajících z analýzy. Následně je definován návrh databáze (sekce 6.2), který bude zaměřen na význam jednotlivých tabulek a případných problémů vzniklých s implementací možnosti návrhu tras. Poslední část se věnuje popisu rozhraní (sekce 6.3), které bude mít uživatel dostupné.

6.1 Architektura řešení

Cílem řešení je navrhnutí *backend* funkcionality tak, aby bylo umožněno plánování a správa tras servisních pracovníků. Návrh architektury je členěn na jednotlivé servisní třídy, které zajišťují určité funkce.



Obrázek 6.1: Návrh architektury řešení.

Návrh architektury řešení lze vidět na obrázku 6.1. Možnost jednoduché záměny služby třetí strany bude zajištěna tak, že každá služba bude mít vlastní implementaci služby (tzv. *service*) a bude navazovat na sdílené rozhraní, které zprostředkovává funkcionalitu nezávisle na konkrétní službě třetí strany. Její funkcionalita bude zajišťovat veškerou komunikaci se službou a správnou transformaci dat mezi službou a interními strukturami. Interní struktury dat budou kopírovat generalizovanou definici vstupních a výstupních dat pro služby (sekce 4.1), přičemž nedůležitá data budou vynechána. Pro zajištění možnosti určení počátečních pevných lokací bude sloužit rozhraní *Direction service* z toho důvodu, že služba VRP tuto akci nepodporuje. Manipulaci s daty budou zajišťovat databázové služby. Jim nadřazená služba (fasáda) bude obsahovat již celou logiku související s tvorbou tras. Jednotlivé koncové body API boudou popsány HTTP protokolem (součástí *Amazon API Gateway*), ty budou následně využívat jednotlivé funkce (*AWS Lambda*).

Výběr vhodné webové služby zajišťující VRP

Z dostupných služeb uvedených v kapitole 4 a na základě požadavků v sekci 5.4 byla pro aktuální řešení vybrána služba ***Openrouteservice***. Služba je pro aktuální použití výhodná z ekonomického hlediska, a to proto, že je zdarma a přitom nijak neomezuje možnost komerčního použití. Bezplatné používání služby je poskytováno pod licencí *CC BY 4.0*[1]. Její základní limity jsou pro aktuální řešení dostatečné, přičemž je dále možné pro každého zákazníka zřídit vlastní *API token* pro službu. Pokud by Logimic v budoucnu uvažoval o některých jiných (placených) službách poskytujících řešení VRP, tak se nabízí ta řešení, která mají denní kreditní systém (např. *Geoapify*).

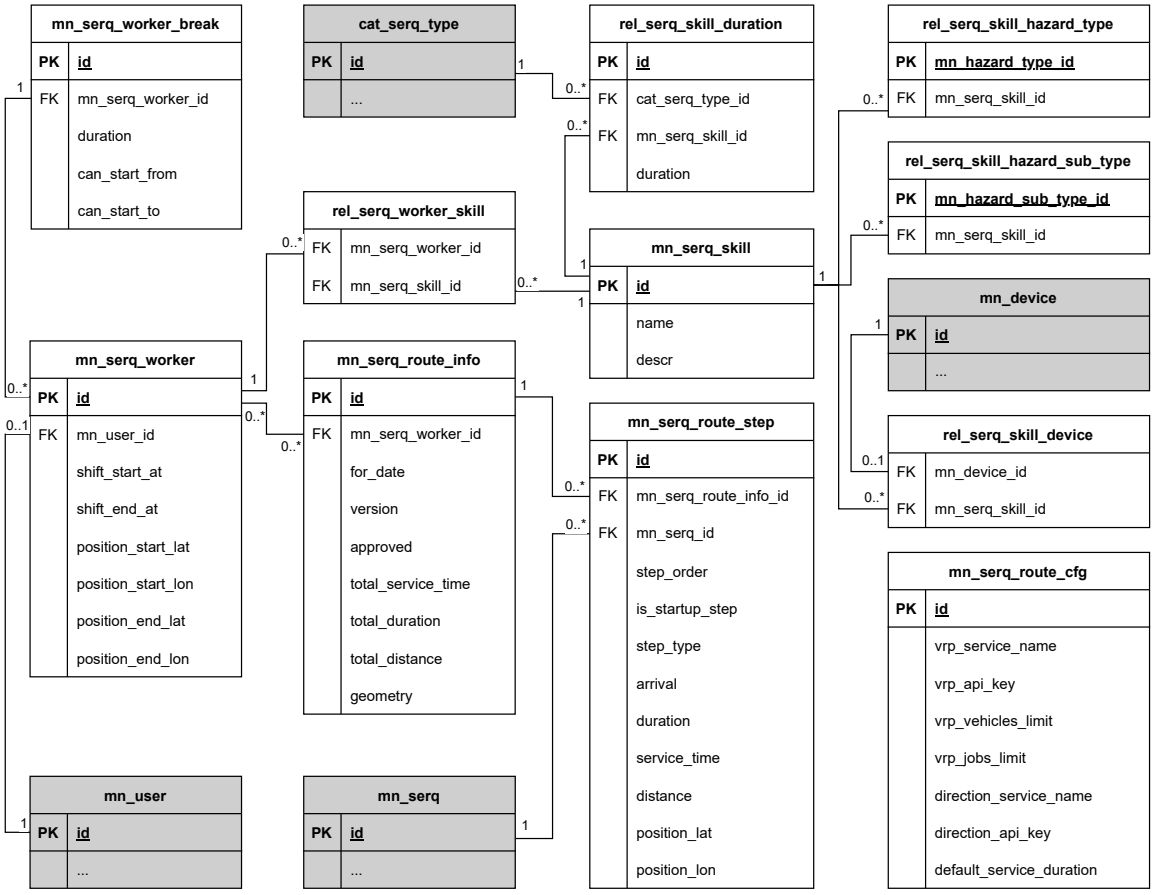
Služba *Openrouteservice* může být výhodná z hlediska budoucnosti, protože nabízí možnost vlastního nasazení služby[2]. Službu lze nasadit jak z přístupných zdrojových souborů, tak pomocí *Docker*[3] kontejneru. Toto by umožnilo jednoduché budoucí nasazení přes službu *AWS ECS (Elastic Container Service)*[4] do *cloud* platfomy od firmy Amazon, na které běží i zbytek platformy pro chytrá města od Logimic. Vlastní nasazení služby taktéž dovoluje určování územních celků, které bude služba využívat. Na základě toho lze korigovat paměťové nároky a snížit ekonomické náklady.

Mapové podklady jsou volně dostupné ze serveru *Geofabrik*[5]. Územní celky jsou základně rozděleny dle kontinentů a následně pak kontinent může obsahovat další celky (státy, státní celky). Služba nabízí data ve formátu *.osm.pbf (OpenStreetMap.Protocolbuffer Binary Format)*, přičemž jsou denně aktualizována. Pokud by bylo potřeba používat více územních celků naráz (např. Evropa a USA), lze využít nástroj *Osmconvert*[6]. Pro aktualizaci mapových souborů lze využít nástroj *Osmupdate*[7], přičemž pro Logimic by bylo vhodné provádět aktualizaci v intervalech o délce minimálně jeden měsíc, a to z důvodu náročnosti, protože v kratších intervalech by byly rozdíly zanedbatelné.

[1] https://creativecommons.org/licenses/by/4.0/
[2] https://giscience.github.io/openrouteservice/installation/Installation-and-Usage.html
[3] https://www.docker.com/
[4] https://aws.amazon.com/ecs/
[5] https://download.geofabrik.de/
[6] https://wiki.openstreetmap.org/wiki/Osmconvert#Parallel_Processing
[7] https://wiki.openstreetmap.org/wiki/Osmupdate

6.2 Návrh databáze

Návrh databáze se odvíjí od již existující databáze Logimic, do které bude potřeba doplnit nové tabulky a případně doplnit informace do již existujících. Na obrázku 6.2 je znázorněn návrh databáze, kde jsou znázorněny všechny entity, které souvisí s řešením. Šedě podbarvené entity jsou již existující v databázi a nijak do nich nebude zasahováno, i z tohoto důvodu není třeba uvádět jejich další informace. Entity bez podbarvení jsou nové, které vzniknou pro řešení. Jména tabulek již budou korespondovat s aktuální systematikou názvů v řešení Logimic a taktéž budou následně použity pro implementaci (přičemž tabulky s prefixem „mn“ jsou hlavní, „rel“ jsou relační a „cat“ jsou kategorické).



Obrázek 6.2: Návrh/Úprava databáze řešení.

6.2.1 Definice tabulek

Servisní pracovník (mn_serq_worker)

Jedná se o tabulku definující servisního pracovníka a obsahující obecné informace k jeho práci. Tyto informace jsou výchozí, přičemž pro následnou konkrétní optimalizaci trasy je bude možné upřesnit. Servisní pracovník je vždy přiřazen k jednomu uživatelskému účtu, protože jeden uživatel nemůže reprezentovat více různých servisních pracovníků.

Přestávka servisního pracovníka (mn_serq_worker_break)

Jde o tabulku, která specifikuje přestávky pro servisního pracovníka, které mají být vykonány během jeho pracovní doby.

Schopnost (mn_serq_skill)

Tato tabulka obsahuje uživatelsky definované schopnosti pro jednotlivé typy a podtypy nebezpečí související s konkrétním úkonem (únik vody, přeplnění koše, údržba zařízení apod.) a pro jednotlivá zařízení. Na provázení s typem a podtypem nebezpečí slouží relační tabulky s prefixem „*rel*“. Tyto relační tabulky jsou vytvořeny pro jednodušší manipulaci s relacemi, protože data s typy a podtypy nebezpečí jsou součástí samostatné databáze pro modul *RAH*. Zařízení jsou přímo navázána rovněž přes vazební tabulku, protože firma Logimic se snaží mít v zařízení jen nejdůležitější informace. Servisní pracovník může disponovat několika schopnostmi, což řeší vazební tabulka *rel_serq_worker_skill*.

Délka výkonu servisního úkonu (rel_serq_skill_duration)

Jedná se o tabulku, která bude uvádět, kolik času trvá vyřešit daný servisní požadavek. Časový údaj bude uživatelsky definován na základě schopnosti a typu servisního požadavku (inspekce, instalace, oprava, výměna).

Trasa (mn_serq_route_info)

Je základní tabulkou pro uchování plánovaných tras. Obsahuje souhrnné informace o konkrétní trase, včetně geometrie pro vizualizaci, přičemž bude uložena v řetězcové notaci lomené čáry (tzn. po dekódování řetězce vznikají body trasy). Trasa dále obsahuje jednotlivé zastávky.

Krok trasy (mn_serq_route_step)

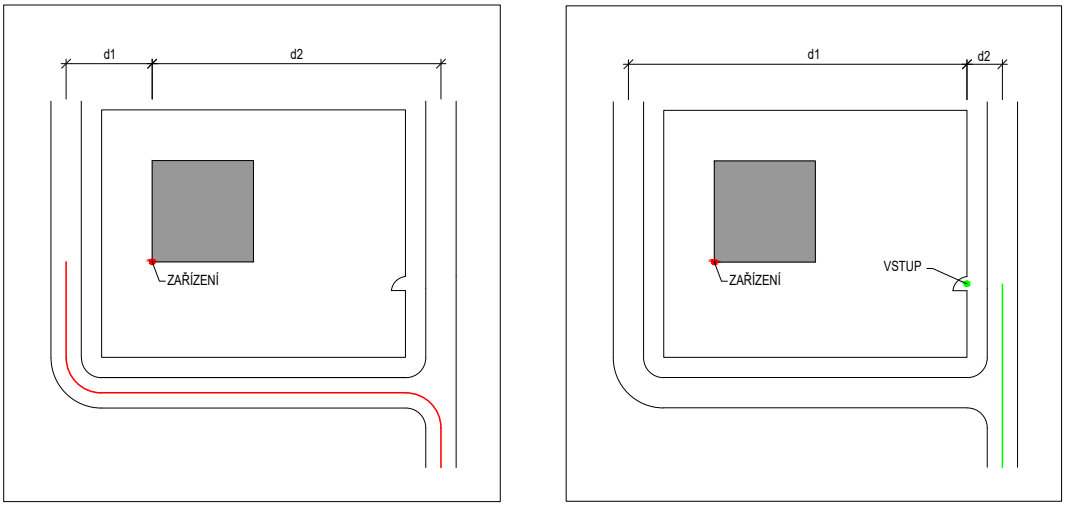
Jedná se o tabulku navazující na tabulku trasy. V kroku trasy jsou uchovány informace k zastávkám. Každá zastávka reprezentuje konkretní servisní požadavek.

Konfigurace (mn_serq_route_cfg)

Konfigurace určuje služby třetích stran, které budou využívány. Dále se zde nacházejí základní hodnoty pro plánování tras a limity pro požadavek na optimalizaci skrze službu třetí strany. Tato tabulka nebude nijak uživatelsky definovatelná a bude přístupná pouze přes databázi. Toto je z důvodu, aby nedošlo k nesprávnému definování dat, a z důvodu, že tato data mají návaznost na implementaci jednotlivých tříd komunikujících se službou třetí strany.

6.2.2 Úprava lokace zařízení

Každé zařízení má v systému definovanou svou polohu, avšak pro generování trasy k zařízení nemusí být tato lokace správná. Při plánování trasy je k zadanému bodu vždy vyhledán nejbližší bod pozemní komunikace, ke kterému je trasa následně generována. V následujících obrázcích jsou sledovány vzdálenosti $d1$ a $d2$, které ukazují, která pozemní komunikace je blíže a která bude použita pro generování trasy.



(a) před korekcí (b) po korekci

Obrázek 6.3: Korekce polohy zařízení.

Na obrázku 6.3a je pouze základní lokace zařízení. Pokud je k němu naplánovaná trasa, jak lze vidět u této konkretní situace, že vchod na pozemek je z opačné strany, může dojít ke špatnému navedení servisního pracovníka k zařízení. Tuto situaci lze eliminovat přidáním nové lokace do zařízení, která bude reprezentovat bod vstupu k zařízení (bod „VSTUP“ na obrázku 6.3b). Tento bod zajistí správný konec pro generování trasy. Lokace vstupu bude v zařízení nepovinná a bude sloužit pro zařízení se zhoršeným přístupem a pokud nebude zadána, tak bude trasa standardně generována k pozici zařízení.

6.3 Definice rozhraní (API)

Definici rozhraní je třeba definovat z toho důvodu, že přes ně bude uživatel využívat veškerou funkcionalitu řešení. Pro většinu požadavků budou definovány tři operace: *GET* (čtení), *POST* (nový záznam) a *PUT* (úprava záznamu). Požadavek typu *DELETE* (smazání) nebude definován u většiny požadavků, a to z důvodu, že je potřeba uchovávat historii dat. Pokud by došlo ke smazání některého záznamu, mohlo by dojít ke smazání dalších závislých dat, a v důsledku toho by mohla být smazána historická data, která by měla být i nadále dostupná. V důsledku uchování historie vyvstává otázka řešení problému zahlcení databáze daty. Tuto otázku není třeba řešit, a to z důvodu, že každé město má svou databázi a data nevznikají periodicky v nízkých intervalech (vteřiny, minuty), tudíž nevznikne tolik dat, aby mohlo po nějakém rozumném čase dojít k přehlcení. Vstupní i výstupní data požadavků budou ve formátu *JSON*.

/sreq_routes/optimalize

- **/single *POST*** – Jedná se o požadavek, jehož účelem je vytvoření optimalizované cesty pro pracovníka. Obsahem požadavku jsou potřebné informace pro optimalizaci a to: *id servisního pracovníka*, *id servisních požadavků*, které mají být předmětem optimalizace, pro které *datum* má být trasa naplánována a *id servisních požadavků*,

které mají být vykonány na začátku trasy, a tak nebudou součástí optimalizace. Dále v požadavku mohou být informace o servisním pracovníkovi, které budou redefinovat uložené informace v databázi. Pokud tyto informace nebo některé části nebudou součástí, tak jsou automaticky přebrány ze servisního pracovníka uloženého v databázi. Požadavek bude vracet vytvořené *id trasy* a případné chybové hlášky, jež vznikly během plánování trasy. Pokud již existuje trasa naplánovaná pro tento den, tak dojde k jejímu verzování (nová trasa bude obsahovat vyšší číslo verze).

- **/multiple *POST*** – Vytvoří optimalizované cesty pro více pracovníků. Obsahem požadavku budou *id servisních pracovníků*, *id servisních požadavků* a pro které *datum* má být trasa naplánována. Informace o servisních pracovnících budou striktně převzaty z databáze a nebude možné nijak předefinovat jejich hodnoty v rámci požadavku. Pokud některý servisní pracovník již má definovanou trasu pro konkrétní datum, tak dojde k verzování trasy.

/serq_routes/route

- **/{*id*} *GET*** – Získá cestu dle *id* včetně jednotlivých kroků cesty.
- **/list *GET*** – Získá cesty dle zvolených parametrů („filtru“):*id trasy*, *id servisního pracovníka*, *id vedoucího (supervisor) pracovníka*, *poslední schválená verze*, *specifická verze*, *od data* a *po datum*.
- **/approve *POST*** – Schválí cesty specifikované dle *id cest* v těle požadavku. Dále je možné v parametru specifikovat, zda má při schvalování dojít i k přiřazení servisních požadavků na servisního pracovníka.

/serq_routes/worker

- **/{*id*} *GET*** – Získá servisního pracovníka dle *id* včetně jeho schopností a přestávek.
- **/list *GET*** – Získá servisního pracovníka dle parametrů: *id* servisního pracovníka a *id* vedoucího (*supervisor*).
- ***POST*** – Vytvoří nového servisního pracovníka na základě *id uživatele*, přičemž v rámci uživatele jsou spravovány i jeho schopnosti a přestávky.
- ***PUT*** – Editace již vytvořeného servisního pracovníka.

/serq_routes/skill

- **/{*id*} *GET*** – Získá schopnost dle *id* včetně délek výkonů servisních požadavků a relací na typ a podtyp nebezpečí (*hazards*) a zařízení.
- **/list *GET*** – Získá všechny schopnosti pouze se základními informacemi a bez navázaných dalších objektů.
- ***POST*** – Vytvoří novou schopnost včetně délek výkonů servisních požadavků a návaznostmi na typy a podtypy nebezpečí (*hazards*) a zařízení.
- ***PUT*** – Editace již vytvořené schopnosti.

- ***/{id} DELETE*** – Odebere schopnost dle *id* včetně navázaných délek výkonů servisních požadavků a relací typu a podtypu nebezpečí (*hazards*) a odebere navázaná zařízení.

/serq_routes/unused_rah_types

- ***GET*** – Získá typy a podtypy nebezpečí (*hazards*), které nejsou součástí žádné schopnosti, přičemž v parametru bude uvedeno, pro které město mají být typy získány.

Kapitola 7

Implementace

Tato kapitola je věnována implementaci řešení, které bylo navrženo v kapitole 6. V části 7.1 jsou popsány použité technologie, které byly použity pro implementaci. Následující část 7.2 popisuje jednotlivé komponenty řešení a jejich účel. V poslední části 7.3 je vysvětlen proces tvorby cest.

7.1 Použité technologie

Pro implementaci řešení je využit jazyk *TypeScript*[1], který je staticky typovaný. Oproti jazyku *JavaScript* nabízí větší modularitu, která je zajištěna podporou tříd, rozhraní, modulů a jmenných prostorů. Pro komunikaci se službami třetích stran je použita knihovna *Axios*[2]. Jedná se o knihovnu, která nabízí komplexnější práci s dotazy. Zpřístupnění rozhraní řešení plánování tras servisních pracovníků zajišťují *Serverless*[3] funkce, které jsou následně vloženy do systému *AWS Lambda*[4]. Ukládání dat zajišťuje databáze *PostgreSQL*[5] ve spolupráci s knihovnou *TypeORM*[6], která zpřístupňuje a abstrahuje databázi pro *TypeScript* a dovoluje přístup *Code-First*[7], který je již využíván v aplikaci. Výběr uvedených technologií byl závislý na již používaných technologiích firmy Logmic pro platfomu chytrých měst.

7.2 Vrstvy řešení

V této části jsou popsány jednotlivé vrstvy řešení, které jsou použity pro docílení požadované funkcionality uvedené v kapitole 5 a reflektují návrh řešení z kapitoly 6. Implementaci lze rozdělit do tří vrstev: (1) databázová, (2) logická a (3) funkce pro *Serverless*. Vrstvy používají vždy pouze vrstvu, která je jim nadřazená.

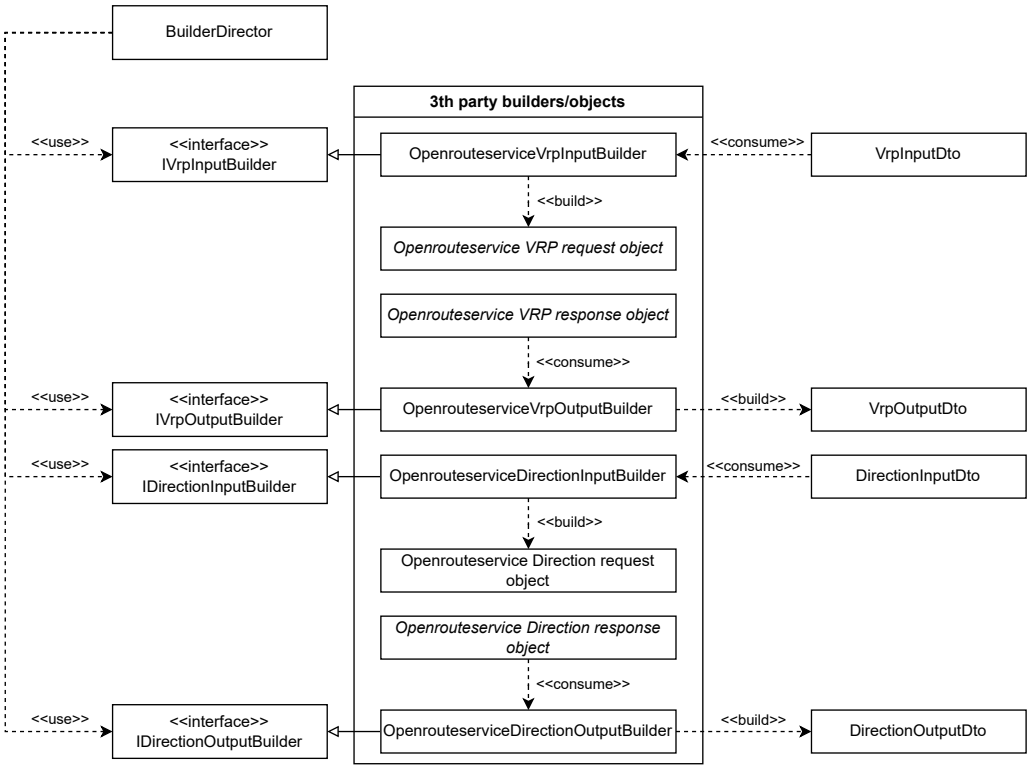
[1] https://www.typescriptlang.org/
[2] https://axios-http.com/docs/intro
[3] https://www.serverless.com/
[4] https://aws.amazon.com/lambda/
[5] https://www.postgresql.org/
[6] https://typeorm.io/
[7] *Code-First* – Jedná se o přístup, kde je databáze vytvářena na základě modelů popsaných v kódu.

7.2.1 Databázová vrstva

Obsahuje pouze jednu třídu služeb `SerqRoutingDbService`, která poskytuje potřebné funkce pro přístup k databázi, a třídy filtrů sloužících k filtrování výběru dat. Jsou zde implementovány funkce pro získání konkrétní entity, seznamu entit dle filtru, uložení entity a případné smazání. V rámci této části jsou definovány jednotlivé databázové modely (entity). Ty jsou uloženy v souboru `SerqRoutingEntities.ts`, ten se nachází mimo adresář databázového projektu pro plánování tras, ale je součástí adresáře `entities`, kde se nachází i další definice databázových modelů. Mapování entit z databáze a manipulace s databázovými daty je zprostředkována ORM[8] framework *TypeORM*.

7.2.2 Logická vrstva

Jedná se o vrstvu, kde se nachází veškerá logika plánování tras. Hlavní třídou poskytující veškerou funkcionalitu je `SerqRoutingFacade`. Ta využívá dalších komponent, které jsou rozděleny do adresářů: `builders`, `models` a `services`.



Obrázek 7.1: Diagram návrhového vzoru *Builder*.

Builders

Tento adresář je věnován návrhovému vzoru *Builder*, který zajišťuje konstrukci datových modelů pro služby třetích stran. Obrázek 7.1 lze rozdělit na dvě části: (1) interní datové

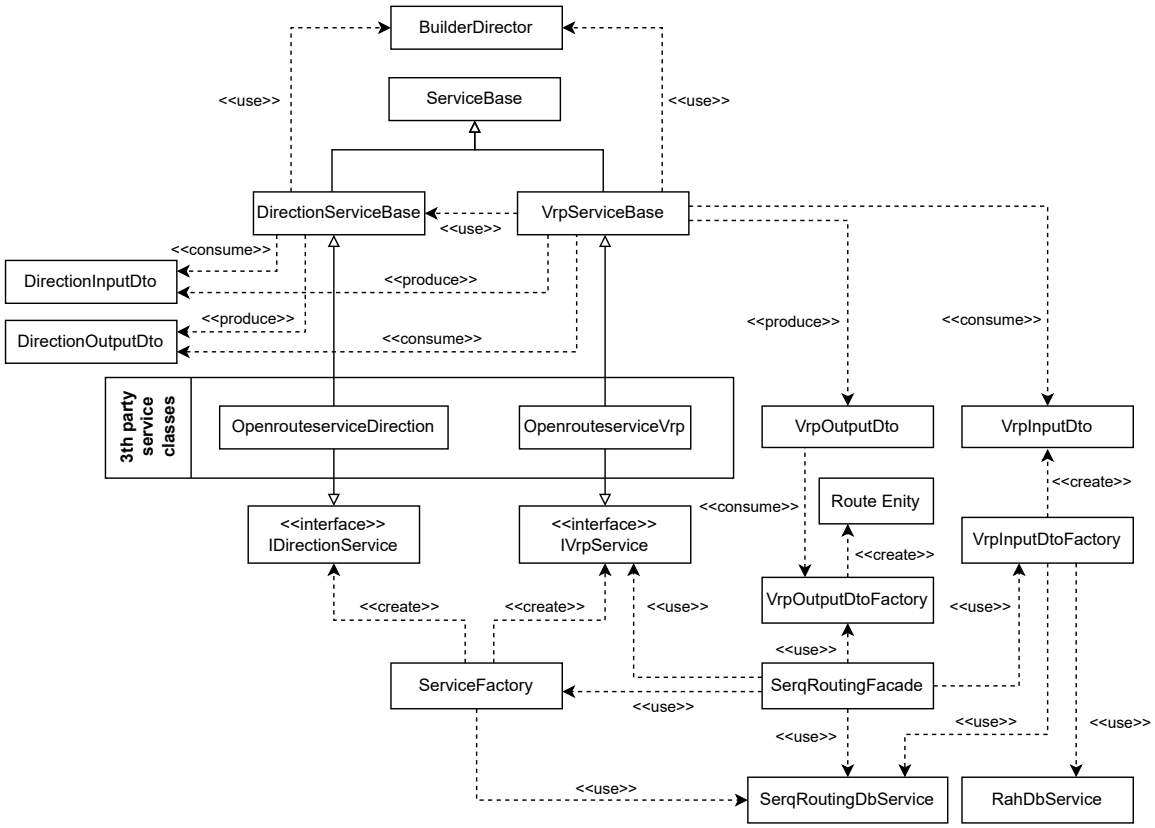
[8] ORM – *Object Relational Mapping* neboli Objektově relační zobrazení, které konvertuje data mezi relační databází a OOP jazykem.

struktury a (2) externí datové struktury, které se nacházejí v části označené *„3th party builders/objects“*. Interní část je navržena tak, aby celá funkcionalita plánování tras byla odstíněna od datových struktur, které jsou využívány službami třetích stran. K tomu, aby řešení podporovalo některou další externí službu, je potřeba definovat další konkrétní *builders*, aby došlo ke správnému převedení dat z interních struktur na externí a opačně. Následná konstrukce objektů je zprostředkována pomocí funkcí umístěných ve třídě `BuilderDirector`.

Modely

Adresář `models` obsahuje veškeré interní datové modely používané v rámci celého procesu tvorby tras. Nachází se zde DTO[9] modely používané v `builders`, které jsou výše popsané. Jsou zde taktéž popsány vstupní a výstupní datové modely dotazů pro optimalizaci trasy.

Dále jsou zde obsaženy třídy návrhového vzoru *Factory*, které se starají o práci s interními datovými modely používanými v `builders`. Tento návrhový vzor zajišťuje správné zkonstruování dat, a to (1) vytvoření vstupního DTO modelu na základě dat uložených v databázi a (2) vytvoření entity z výstupního DTO modelu pro následné uložení do databáze. Závislosti jednotlivých tříd jsou graficky znázorněny v pravé části obrázku 7.2.



Obrázek 7.2: Diagram závislostí tříd.

[9]DTO – Data transfer object je objekt přenášející data mezi dvěma procesy.

Služby

Adresář `services` obsahuje třídy zajišťující konstrukci tras a volání služeb třetích stran zajišťující optimalizaci. Na obrázku 7.2 je tato část vyobrazena v levé části a taktéž lze rozdělit na dvě části, a to interní třídy a externí, které jsou v části označené *„3th party service classes“*.

Externí část je věnovaná specifickému nastavení služby třetí strany, tzn. nastavení `builders`, `URL` služby, `API` klíče (autorizace) a specifickému nastavení požadavku. Z toho důvodu jsou tyto třídy co nejjednodušší, aby bylo snadné přidat podporu dalších služeb.

Interní část zajišťuje veškerou funkcionalitu, jedná se především o bázové třídy, které obsahují tuto funkcionalitu. Externí třídy služeb jsou vytvářeny pomocí návrhového vzoru *Factory* a jsou zkonstruovány na základě konfigurace uložené v databázi. Třída je vracena v souladu s jejím rozhraním tak, aby byla zajištěna uvedená škálovatelnost externích služeb.

Třídy služeb jsou navrženy tak, aby bylo možné co nejvíce využít dědičnosti a zapouzdření. Toto zajišťuje jednoduchou implementaci dalších služeb třetích stran.

7.2.3 *Serverless* vrstva

Tato vrstva slouží k obalení logické vrstvy do *Serverless* funkcí. Funkce z logické vrstvy jsou provolávány skrze soubor `main.ts`, kde jsou popsány všechny funkce dostupné pro aplikační rozhraní. Definice funkcí pro *Serverless* jsou popsány v souboru `serverless.yml`.

7.3 Proces tvorby a optimalizace tras

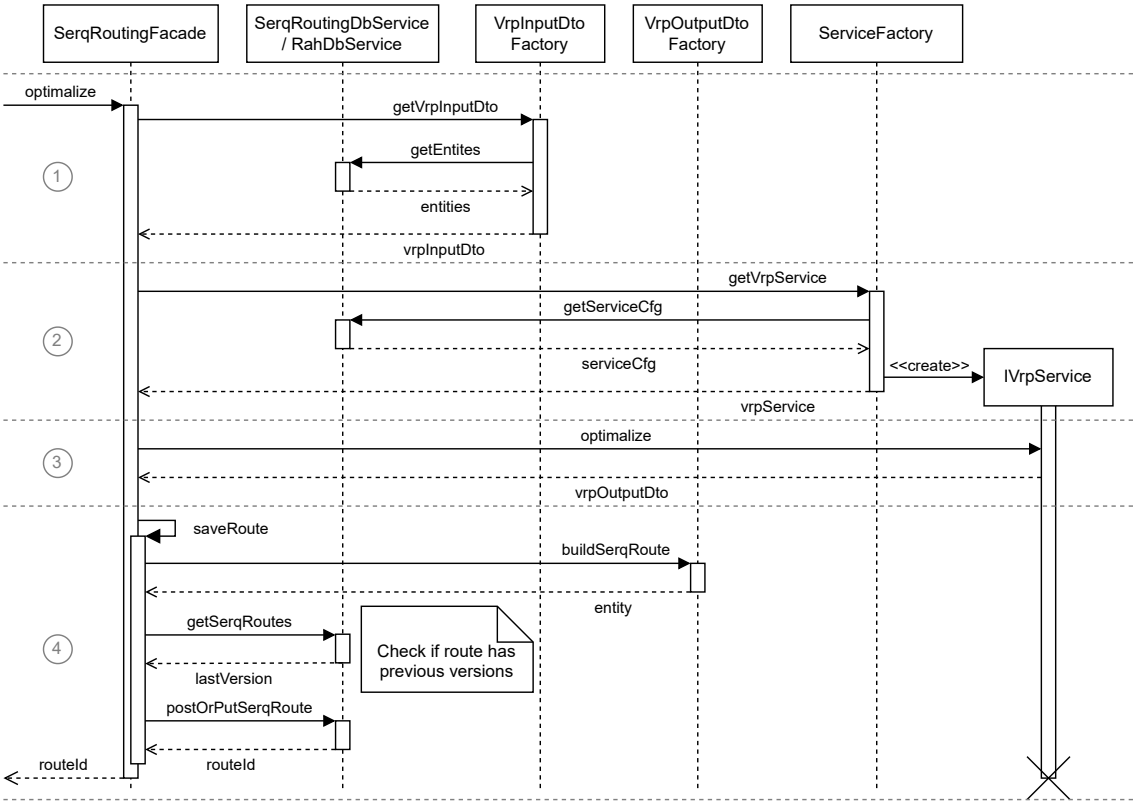
V této části je detailněji popsán celý proces tvorby trasy, který je dále vizualizován na obrázcích 7.3 a 7.4, a to pomocí sekvenčního diagramu. Digramy jsou ve zjednodušené formě, tzn. jsou vynechány parametry funkcí a nejsou zde obsaženy méně důležité funkce, což by mohlo způsobovat špatnou čitelnost grafů. Především je kladen důraz na důležité rámcové funkce, které zajišťuji hlavní funkcionalitu.

Na obrázku 7.3 je diagram zaměřen na rámcovou funkcionalitu pro optimalizaci trasy. Diagram je rozčleněn do čtyř sekcí, které rozčleňují optimalizační metodu na funkční bloky, které jsou dále rozepsány.

1. V této části dochází k zavolání optimalizace, která je dále implementačně rozdělena do dvou funkcí pro *single* a *multiple* optimalizaci, což kopíruje popsanou definici API v části 6.3. V diagramu jsou obě metody abstrahovány do metody `optimalize`, a to na základě toho, že metody nejsou nijak odlišné v jejich chování, ale pouze v obsahu dat.

   Po zavolání optimalizace dochází k vytvoření `VrpInputDto`, který v sobě obsahuje veškeré informace pro následnou optimalizaci. Tento objekt je vytvořen pomocí třídy `VrpInputDtoFactory`, která implementuje návrhový vzor *Factory*. Tato třída obsahuje analogicky dvě metody, a to pro *single* a *multiple* (v diagramu abstrahováno do metody `getVrpInputDto`), pro vytvoření vstupních DTO pro optimalizaci. Tyto metody zpracují stručné informace z požadavku a na jejich základě dohledají v databázi všechna potřebná data, která budou potřeba pro plánování a optimalizaci trasy.

2. Po získání vstupního DTO a jeho kontrole, zda se nevyskytly kritické chyby při jeho vytváření, dochází k vytvoření služby zajišťující optimalizaci trasy. Služba je vytvo-



Obrázek 7.3: Sekvenční diagram procesu tvorby tras z pohledu hlavní služby.

řena na základě konfigurace, která se nachází v databázi. Objekt služby je získán pod rozhraním `IVrpService`, které implementuje funkci pro optimalizaci trasy.

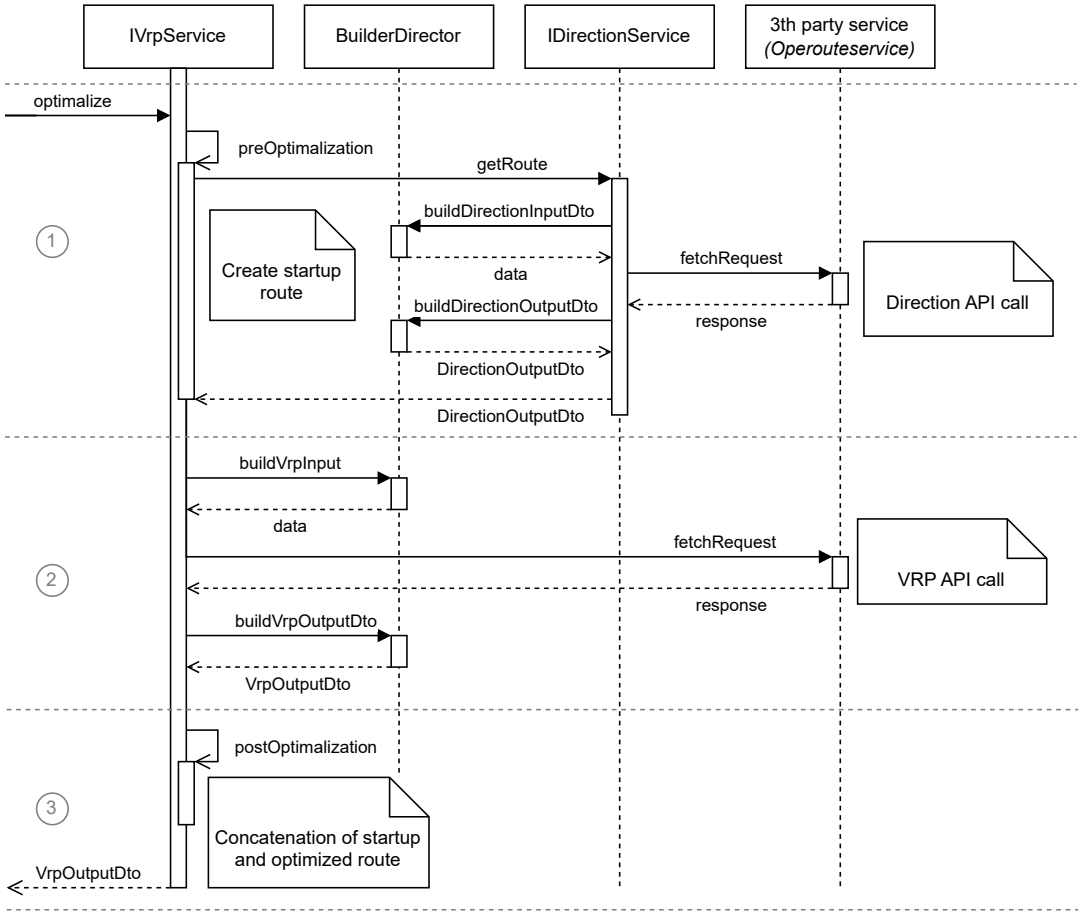
3. Následně dochází k zavolání metody `optimalize`, která slouží pro získání optimalizované trasy. Tato metoda vrací `VrpOutputDto`, kde jsou obsaženy všechny potřebné informace o optimalizovaných trasách. Detailnímu vysvětlení fungování této metody je věnován popis diagramu na obrázku 7.4.

4. Po získání výstupního objektu (`VrpOutputDto`) dochází k uložení jednotlivých tras pomocí metody `saveRoute`. K tomu, aby bylo možné uložit data do databáze, je nejprve potřeba transformovat výstupní objekt na objekt databázového modelu (entity) trasy. K tomu slouží metoda `buildSerqRoute`, která je součástí `VrpOutputDtoFactory` třídy, která implementuje návrhový vzor *Factory*.

   Před uložením získaného databázového modelu trasy je potřeba ověřit, jestli nedochází k verzování trasy. To je uskutečněno na základě metody `getSerqRoutes`, jíž jsou předány informace o trase. Pokud již trasa existuje, tak pro novou trasu dochází k inkrementaci verze, v opačném případě je nastavena trasa s verzí jedna.

   Po přiřazení verze dochází k uložení trasy pomocí metody `postOrPutSerqRoute`, která vrací identifikátor nově vytvořené trasy v databázi.

Dále je diagram na obrázku 7.4 věnován detailnímu popisu fungování optimalizace a je rozdělen do tří částí, jež definují funkční bloky.



Obrázek 7.4: Sekvenční diagram procesu tvorby tras, z detailního pohledu služby VRP.

1. Optimalizace (metoda `optimalize`) je vykonána na základě vstupního interního DTO (`VrpInputDto`). Na začátku optimalizace dochází ke kontrole, zda jsou některé servisní požadavky označeny jako fixní, protože fixní požadavky mají být vykonány na začátku trasy a nemají být součástí optimalizace. Pokud nejsou definovány fixní servisní požadavky, dochází k přeskočení funkcionality tohoto bod a bodu č. 3 (resp. funkce `postOptimalization`). Dále je pokračováno bodem č. 2.

   Je-li označen alespoň jeden servisní požadavek jako fixní, dochází k vykonání metody `preOptimalization`. Tato funkce zajišťuje získání trasy pro počáteční fixní body trasy. Pro tuto funkcionalitu je využívána služba pod rozhraním `IDirectionService`.

   Nejprve jsou vybrány ty servisní požadavky, které mají být součástí fixního začátku trasy a s nimiž je následně volána metoda `getRoute`. Zde dojde k vytvoření vstupního objektu pro službu tras (*Direction service*) za pomoci `BuilderDirector` a metody `buildDirectionInputDto`. Tento objekt je zkonstruován na základě interního DTO (`DirectionInputDto`). Po získání trasy pomocí `fetchRequest` dochází k transformaci dat metodou `buildDirectionOutput` na interní DTO (`DirectionOutputDto`).

   Výsledkem těchto operací je uchování fixního začátku tras/y.

2. Tato část je věnována optimalizaci servisních požadavků, které mají být předmětem optimalizace. Nejprve dochází ke zkonstruování vstupního objektu pro konkrétní

optimalizační službu na základě interního DTO (`VrpInputDto`). Objekt je vytvořen pomocí metody `buildVrpInput`, která je obsažena v `BuilderDirector`. Po získání vstupního objektu je zavolána služba třetí strany pro optimalizaci (metoda `fetchRequest`). Odpověď je zpět transformována na interní DTO (`VrpOutputDto`) pomocí funkce `buildVrpOutput` v `BuilderDirector`.

3. V poslední části dochází k případnému spojení tras, pokud byl v optimalizaci specifikován fixní začátek. Spojení fixní části a optimalizované části trasy je realizováno metodou `postOptimalization`. Následně je vracena výsledná trasa.

Kapitola 8

Testování

Testování je důležitou součástí vývoje, při kterém dochází ke kontrole vyvíjené funkcionality. Jedná se o proces, v němž se odhalují možné nedokonalosti a chyby, dochází ke kontrole, zda jsou splněny požadavky na řešení.

8.1 Implementace

Pro navrhované a implementované řešení je třeba k testování přistupovat tak, aby bylo možné ověřit funkčnost a správnost jednotlivých částí. Z tohoto důvodu je v první části zvolen přístup jednotkových testů neboli *Unit Testing*. Je to postup, při kterém dochází k testování samostatných částí kódu a dochází k odhalování chyb implementace. Pro testování byl zvolen framework *Mocha*[1], resp. *TS-Mocha,*[2] aby bylo možné *framework* využít pro testování kódu v jazyce *TypeScript*.

Toto testování se zaměřuje na správnost implementace jednotlivých funkcí a je rozděleno do dvou částí.

- První část je věnována testům, které kontrolují správnou funkcionalitu ukládání dat. To zahrnuje ověření správnosti definic jednotlivých databázových modelů (entit), protože v řešení je použit *Code-first* přístup. Dále dochází ke kontrole funkcí, které jakkoliv manipulují s databázovými daty, tzn. ukládání a výběr dat. Testy této části jsou popsány v testovacím skriptu `SerqRouteDbService`.
- Druhá část se věnuje testům implementace procesu tvorby a optimalizace tras. Zde jsou zahrnuty testy, které ověřují správnou transformaci dat mezi interními a externími datovými modely, tzn. ověření funkčnosti tříd implementujících návrhové vzory *Factory* a *Builder*. Dále se zde vyskytují testy implementace, resp. správnosti získávání dat ze služeb třetích stran. V poslední řadě je otestována celková funkcionalita, kde jsou spojeny všechny předešlé části dohromady do požadované funkcionality řešení. Tyto testy jsou součástí testovacího skriptu `SerqRoutingFacade`.

Tyto testy taktéž zabezpečují správnou funkcionalitu a stejné chování i za předpokladu, že bude dále docházet k vývoji nebo změně v systému správy tras.

[1] https://mochajs.org/
[2] https://www.npmjs.com/package/ts-mocha

8.2 Návrh architektury

Tato část testování navazuje na požadavek řešení (sekce 5.4), který definuje, že řešení by mělo obsahovat jednoduchý systém záměny služeb třetích stran.

Návrh architektury a její implementaci nelze otestovat automatickým nástrojem, a proto je tato část posouzena na základě manuálního testování. Tuto část lze otestovat tak, že ke službě *Openrouteservice*, která je součástí řešení, bude přidána nová služba. Implementace nové služby bude posouzena na základě náročnosti implementace a funkčnosti.

Vybrána byla služba od firmy *Geoapify*, pro kterou byly implementovány jednotlivé *Builders*, a služba zajišťující komunikaci. Časová náročnost implementace jednotlivých tříd nebyla vysoká a trvala okolo tří hodin, v nichž je i započítán čas potřebný pro přečtení a pochopení dokumentace ke službě. Při implementaci byla zjištěna nedostatečná funkcionalita pro získání geometrie tras. To bylo způsobeno tím, že tato služba v rámci optimalizačního požadavku negeneruje geometrii tras v žádné podobě, ale generuje pouze seřazení bodů (servisních požadavků), v jakém pořadí mají být vykonány. Z tohoto důvodu je dále potřeba po získání seřazení získat jednotlivé geometrie tras. Získání tras již zabezpečuje služba tras (*Direction service*). Tento nedostatek byl opraven tak, aby bylo možné po optimalizaci získat geometrii tras, pokud to služba v základu neumožňuje.

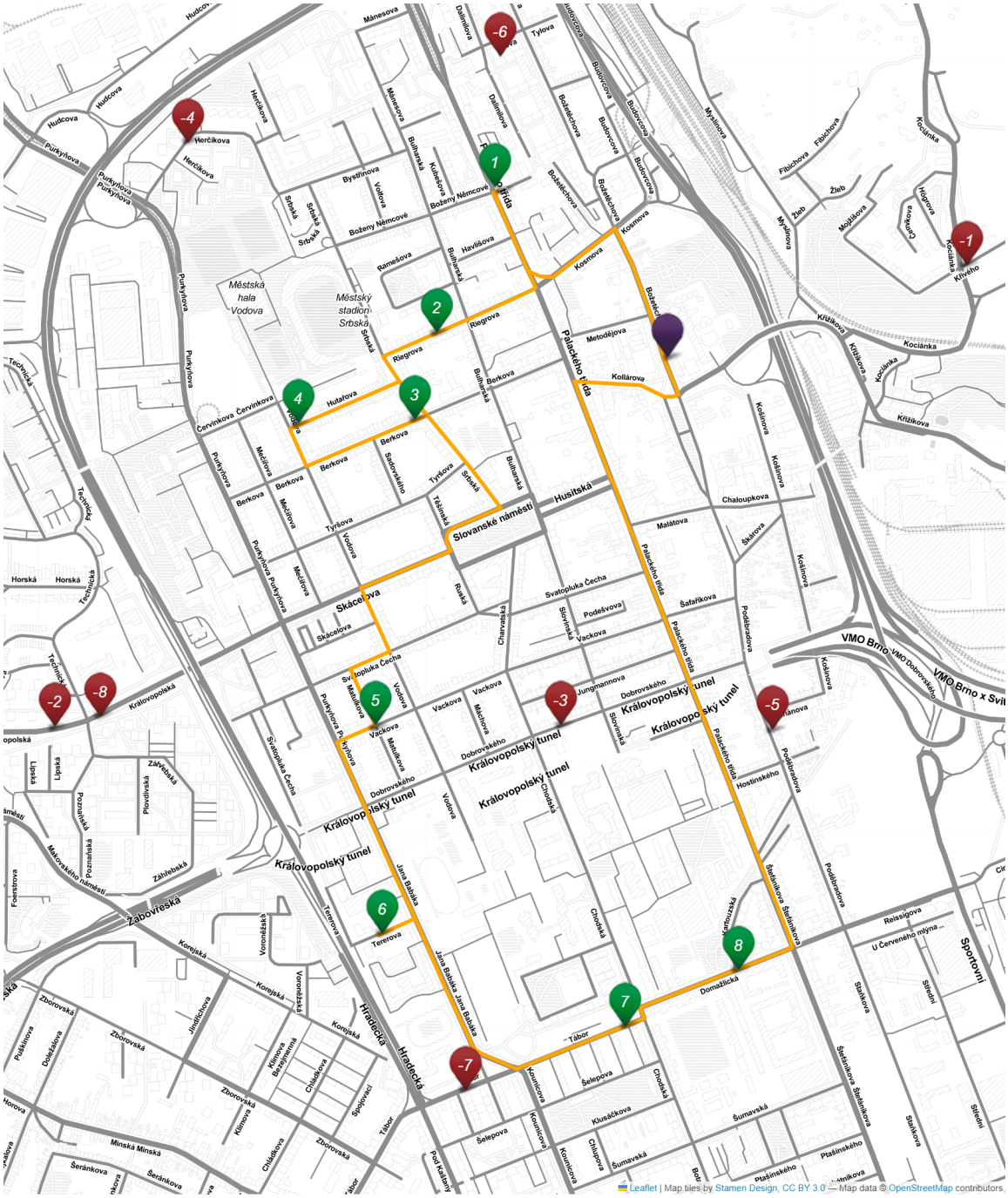
Na základě výše uvedených informací lze říci, že tento požadavek na řešení byl splněn, a to na základě malého počtu tříd, které je potřeba implementovat pro přidání další služby, a malé časové náročnosti.

8.3 Zhodnocení použitelnosti

V rámci testování je potřeba zhodnotit i výsledné výstupy, tzn. jestli vygenerovaná (optimalizovaná) trasa má potenciál být i vhodnou trasou v reálném světě.

Na obrázku 8.1 lze vidět vygenerovanou trasu pro servisního pracovníka. Začátek a konec trasy je vyznačen fialovou značkou, body označené zeleně jsou body trasy s příslušným pořadím a červeně označené body jsou ty, které byly součástí vstupu pro optimalizaci, avšak nebylo je možné v daném časovém úseku splnit. Každý bod servisního požadavku na mapě má časové ohodnocení dvacet minut a servisní pracovník má tří hodinovou pracovní dobu na jejich vyřešení.

Na obrázku je možné vidět, že výsledná cesta je v pořádku, protože úkolem optimalizace je minimalizovat ujetou vzdálenost. (Bod 1 je upřednostněn, protože obsahuje kratší trasu oproti bodu -7. Bod -7 se nachází na straně pozemní komunikace, pro kterou je trasa delší kvůli jednosměrným ulicím v blízkém okolí.) Je však možné polemizovat, zda v reálném použití bude trasa nejvhodnější. To reflektuje různé aktuální situace, které při plánování není možné úplně zhodnotit a zohlednit. Proto je potřeba, aby i uživatel byl schopen zasáhnout do trasy a individuálně ji mohl upravit dle potřeb. Taktéž se ve velkých městech ve velké míře objevují jednosměrné ulice, které mohou plánování tras komplikovat. Z toho plyne, že generování optimalizace trasy má smysl a může výrazně zjednodušit práci a celkové plánování, avšak je třeba dbát na to, že ne vždy může být nejlepší a nejoptimálnější.



Obrázek 8.1: Ilustrace trasy servisního pracovníka.

Kapitola 9

Závěr

Cílem této práce bylo navrhnout a implementovat systém pro plánování (optimalizaci) a správu tras servisních požadavků v chytrých městech pro platformu vyvíjenou firmou Logimic. Optimalizace je důležitá část tohoto řešení, je zprostředkována službami třetích stran a byla důležitou částí správného návrhu a následné implementace do systému.

V rámci práce byla vytvořena nová část systému, která napojuje již existující servisní požadavky na systém umožňující z nich vytvářet optimalizované trasy. V tomto řešení je kladen důraz zejména na škálovatelnost, resp. možnost měnit služby třetích stran v relativně krátkém čase a bez větší nutnosti měnit značnou část kódu aplikace. Z toho důvodu implementovaná funkcionalita ve velké míře využívá návrhových vzorů, které zajišťují vhodnou architekturu řešení pro měnění služeb.

Hlavním výstupem je jedna třída („fasáda“), která v sobě obsahuje veškerou aplikační logiku potřebnou pro správu tras a která je dále využívána pro koncové body rozhraní. Aplikační logika aktuálně obsahuje implementaci služby *Openrouteservice* zajišťující optimalizaci trasy, která je pro aktuální použití nejvýhodnější (zdarma s některými omezeními).

Do budoucna je možné řešení dále přizpůsobovat požadavkům, které vzniknou až v reálném provozu. Po nasazení do reálného provozu je potřeba taktéž počítat s ekonomickou náročností a je nutné vhodně zvolit službu třetí strany dle nabízených cenových plánů. Dále může dojít k přidání nové funkcionality, a to tzv. navigace, která bude využívat již existující trasy a která bude servisnímu pracovníkovi nabízet navigaci mezi jednotlivými servisními požadavky nebo např. informovat a ukazovat důležité informace pro jeho práci během jeho aktuální trasy.

Literatura

[1] *Population History | City of Edmonton* [online]. [cit. 2022-07-12]. Dostupné z: https://www.edmonton.ca/city_government/facts_figures/population-history.

[2] *System Requirements | Openrouteservice* [online]. [cit. 2023-01-21]. Dostupné z: https://giscience.github.io/openrouteservice/installation/System-Requirements.html.

[3] *What is AWS Lambda?* [online]. [cit. 2023-02-15]. Dostupné z: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html.

[4] Augerat, P., Belenguer, J.-M., Benavent, E., Corbéran, A. a Naddef, D. Separating capacity constraints in the CVRP using tabu search. *European Journal of Operational Research*. Elsevier. 1998, sv. 106, 2-3, s. 546–557.

[5] Barbarosoglu, G. a Ozgur, D. A tabu search algorithm for the vehicle routing problem. *Computers & Operations Research*. Elsevier. 1999, sv. 26, č. 3, s. 255–270.

[6] Bell, J. E. a McMullen, P. R. Ant colony optimization techniques for the vehicle routing problem. *Advanced engineering informatics*. Elsevier. 2004, sv. 18, č. 1, s. 41–48.

[7] Blazinskas, A. a Misevicius, A. Combining 2-opt, 3-opt and 4-opt with k-swap-kick perturbations for the traveling salesman problem. *Kaunas University of Technology, Department of Multimedia Engineering, Studentu St.* 2011, s. 50–401.

[8] Blum, C. a Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*. Acm New York, NY, USA. 2003, sv. 35, č. 3, s. 268–308.

[9] Braekers, K., Ramaekers, K. a Van Nieuwenhuyse, I. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*. Elsevier. 2016, sv. 99, s. 300–313.

[10] Chao, I.-M., Golden, B. L. a Wasil, E. A. The team orienteering problem. *European Journal of Operational Research*. 1996, sv. 88, č. 3, s. 464–474. DOI: https://doi.org/10.1016/0377-2217(94)00289-4. ISSN 0377-2217.

[11] Dantzig, G., Fulkerson, R. a Johnson, S. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*. INFORMS. 1954, sv. 2, č. 4, s. 393–410. ISSN 00963984.

[12] Dantzig, G. B. a Ramser, J. H. The truck dispatching problem. *Management science*. Informs. 1959, sv. 6, č. 1, s. 80–91.

[13] Davendra, D. *Traveling salesman problem: Theory and applications*. BoD–Books on Demand, 2010. 1–5 s.

[14] Davis, M. What is Turing reducibility. *Notices of the AMS*. 2006, sv. 53, č. 10, s. 1218–1219.

[15] Engels, C. a Manthey, B. Average-case approximation ratio of the 2-opt algorithm for the TSP. *Operations Research Letters*. Elsevier. 2009, sv. 37, č. 2, s. 83–84.

[16] Galán García, J. L., Aguilera Venegas, G. a Rodríguez Cielos, P. An accelerated-time simulation for traffic flow in a smart city. *Journal of Computational and Applied Mathematics*. Elsevier. 2014, sv. 270, s. 557–563.

[17] Gandomi, A. H., Yang, X.-S., Talatahari, S. a Alavi, A. H. *Metaheuristic applications in structures and infrastructures*. Newnes, 2013.

[18] Giffinger, R., Fertner, C., Kramar, H., Kalasek, R., Milanović, N. et al. *Smart cities - Ranking of European medium-sized cities*. Leden 2007.

[19] Harrison, C., Eckman, B., Hamilton, R., Hartswick, P., Kalagnanam, J. et al. Foundations for smarter cities. *IBM Journal of research and development*. IBM. 2010, sv. 54, č. 4, s. 1–16.

[20] Hasan, M. *State of IoT 2022: Number of connected IoT devices growing 18% to 14.4 billion globally* [online], 18. května 2022 [cit. 2022-07-12]. Dostupné z: https://iot-analytics.com/number-connected-iot-devices/.

[21] Jacobs Blecha, C. a Goetschalckx, M. The vehicle routing problem with backhauls: properties and solution algorithms. 1992.

[22] Janani, R., Renuka, K., Aruna, A. et al. IoT in smart cities: A contemporary survey. *Global Transitions Proceedings*. Elsevier. 2021, sv. 2, č. 2, s. 187–193.

[23] Joshi, S. a Kaur, S. Nearest neighbor insertion algorithm for solving capacitated vehicle routing problem. In: IEEE. *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*. 2015, s. 86–88.

[24] Jourdan, L., Basseur, M. a Talbi, E.-G. Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*. Elsevier. 2009, sv. 199, č. 3, s. 620–629.

[25] Jünger, M., Reinelt, G. a Rinaldi, G. The traveling salesman problem. *Handbooks in operations research and management science*. Elsevier. 1995, sv. 7, s. 225–330.

[26] Kirimtat, A., Krejcar, O., Kertesz, A. a Tasgetiren, M. F. Future trends and current state of smart city concepts: A survey. *IEEE access*. IEEE. 2020, sv. 8, s. 86448–86467.

[27] Kondepudi, S., Ramanarayanan, V., Jain, A., Singh, G., Nitin Agarwal, N. et al. Smart Sustainable Cities: an Analysis of Definitions; the ITU-T Focus Group for Smart Sustainable Cities. *International Telecommunication Union (ITU): Geneva, Switzerland*. 2014.

[28] Laporte, G. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research*. Elsevier. 1992, sv. 59, č. 3, s. 345–358.

[29] Lee, I. a Lee, K. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Business horizons*. Elsevier. 2015, sv. 58, č. 4, s. 431–440.

[30] Lu, Q. a Johnson, P. A. Characterizing new channels of communication: A case study of municipal 311 requests in Edmonton, Canada. *Urban Planning*. MISC. 2016, sv. 1, č. 2, s. 21.

[31] Lutkevich, B. *Amazon RDS (Relational Database Service): What is Amazon RDS?* [online]. Dostupné z: https://www.techtarget.com/searchaws/definition/Amazon-Relational-Database-Service-RDS.

[32] Lysgaard, J. Clarke & Wright's savings algorithm. *Department of Management Science and Logistics, The Aarhus School of Business*. 1997, sv. 44.

[33] Madakam, S., Lake, V., Lake, V., Lake, V. et al. Internet of Things (IoT): A literature review. *Journal of Computer and Communications*. Scientific Research Publishing. 2015, sv. 3, č. 05, s. 164.

[34] Min, J. a Jin, C. A two-phase greedy strategy in one to many PDVRP. In: IEEE. *2016 International Conference on Logistics, Informatics and Service Sciences (LISS)*. 2016, s. 1–5.

[35] O'Neil, D. Nearest Neighbors Problem. In: Shekhar, S. a Xiong, H., ed. *Encyclopedia of GIS*. Boston, MA: Springer US, 2008, s. 783–787. ISBN 978-0-387-35973-1. Dostupné z: https://doi.org/10.1007/978-0-387-35973-1_869.

[36] Pýrek, F. *Úvod do serverlessu: 3. Přehled základních AWS služeb pro Serverless aplikace* [online]. Dostupné z: https://blog.purple-technology.com/cs/uvod-do-serverlessu-prehled-zakladnich-aws-sluzeb-pro-serverless-aplikace/.

[37] Savelsbergh, M. W. a Sol, M. The general pickup and delivery problem. *Transportation science*. INFORMS. 1995, sv. 29, č. 1, s. 17–29.

[38] Stützle, T., Dorigo, M. et al. ACO algorithms for the traveling salesman problem. *Evolutionary algorithms in engineering and computer science*. 1999, sv. 4, s. 163–183.

[39] Syed, A. S., Sierra Sosa, D., Kumar, A. a Elmaghraby, A. IoT in smart cities: a survey of technologies, practices and challenges. *Smart Cities*. MDPI. 2021, sv. 4, č. 2, s. 429–475.

[40] Tarkoma, S. a Katasonov, A. Internet of Things Strategic Research Agenda. Finnish Strategic Centre for Science. *Technology and Innovation. Retrieved from http://www. internetofthings. fi/Turner, JH (1988). A theory of social interaction. Stanford: California Stanford University Press*. 2011, s. 6.

[41] Wood, C. *What Is 311?* [online], 2. srpna 2016 [cit. 2023-01-12]. Dostupné z: https://www.govtech.com/dc/what-is-311.html.

[42] Yin, C., Xiong, Z., Chen, H., Wang, J., Cooper, D. et al. A literature survey on smart cities. *Science China Information Sciences*. Springer. 2015, sv. 58, č. 10, s. 12.

[43] Zhong, Y. a Pan, X. A hybrid optimization solution to VRPTW based on simulated annealing. In: IEEE. *2007 IEEE International Conference on Automation and Logistics*. 2007, s. 3113–3117. DOI: 10.1109/ICAL.2007.4339117.

[44] Zirour, M. Vehicle routing problem: models and solutions. *Journal of Quality Measurement and Analysis JQMA*. 2008, sv. 4, č. 1, s. 205–218.

Příloha A

Cenové plány služeb

V následující části jsou popsány jednotlivé plány služeb s nejdůležitějšími informacemi o nich. Ceny jsou uvedeny v době Q1-Q2 2023.

Název plánu	Kredity	Cena (měsíční)	Komerční použití	limit req/sec	Podpora	SLA
Free	3 000	0 €	limitované	5 req/sec	×	×
API 10	10 000	49 €	✓	12 req/sec	Email	✓
API 25	25 000	89 €	✓	15 req/sec	Email	✓
API 50	50 000	149 €	✓	20 req/sec	Prioritní	✓
API 100	100 000	249 €	✓	25 req/sec	Prioritní	✓
API 250	250 000	499 €	✓	30 req/sec	Prioritní	✓
Custom	-	od 700 €	✓	-	Prioritní	✓

Tabulka A.1: Geoapify – Cenové plány

Název plánu	Kredity	Cena (měsíční)	Komerční použití	max. vozidel	max. lokací	Podpora
Free	500	0 €	×	1	5	fórum
Basic	5 000	69 €	✓	2	30	fórum
Standart	15 000	199 €	✓	10	80	✓
Premium	50 000	479 €	✓	20	150	✓
Custom	-	-	✓	až 200	až 10K	✓

Tabulka A.2: Graphhopper – Cenové plány

Název plánu	Počet lokací (měsíc)	Cena (měsíční)	Podpora 24/7	SLA
PRO Tier 1	1 000	150 $ (≐ 137 €)*	×	×
PRO Tier 2	2 000	275 $ (≐ 253 €)*	×	×
PRO Tier 3	5 000	600 $ (≐ 552 €)*	×	×
PRO Tier 4	15 000	1 500 $ (≐ 1379 €)*	×	×
Enterprise	>15 000	≈ 0.05 $ (0,046 €)* / lokace	✓	✓

*přepočet 1$ = 0.92 €

Tabulka A.3: Routific – Cenové plány

Transkace/měsíc	cena za 1 000 transakcí
0 - 500	0 €
501 - 40 000	24 €
40 001 - 200 000	19.20 €
>200 001	-

Tabulka A.4: HERE – Cenové plány

Název	Měsíční poplatek	Webový portál podpory	Čas odpovědi	SLA
Free	-	×	-	×
Developer	36 €	✓	do 72h	×
Essential	360 €*	✓	do 12h	✓
Advanced	1 600€*	✓	do 6h	✓
Premium	- *	✓	do 4h	✓
Premium Success	- *	✓	do 4h	✓

*nebo 9% licenčních poplatků

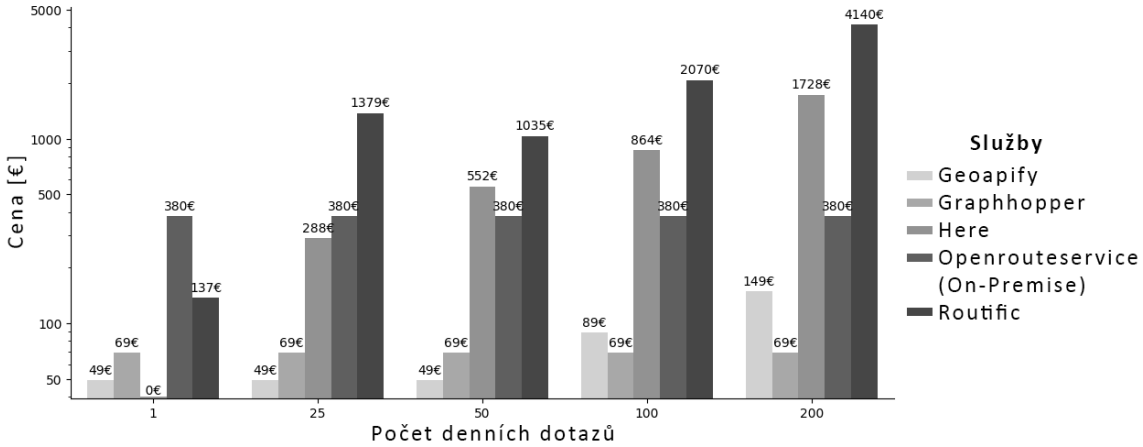
Tabulka A.5: HERE – Cenové plány podpory

Název plánu	limity pro optimalizaci	
	denní	minutový
Standart	500	40
Collaborative	2 500	40
On-Premise	-	-

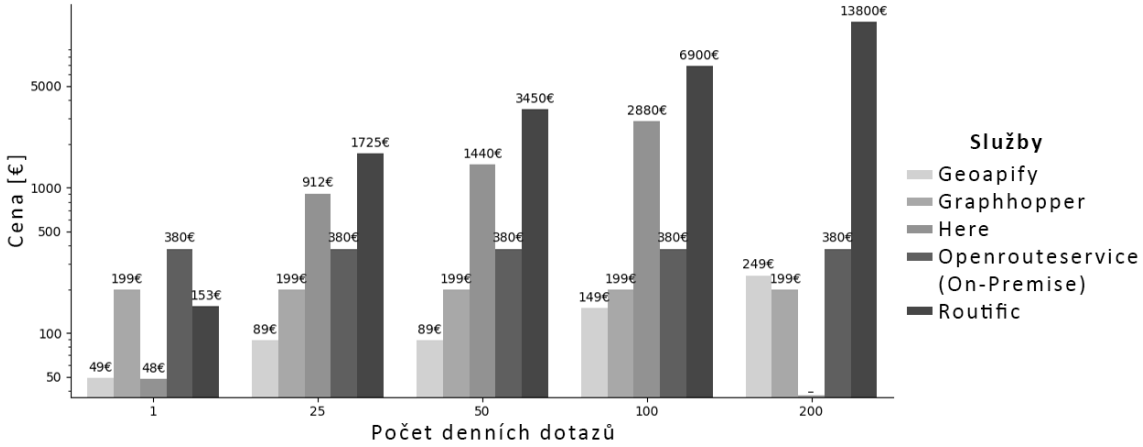
Tabulka A.6: Openrouteservice – Plány

Příloha B

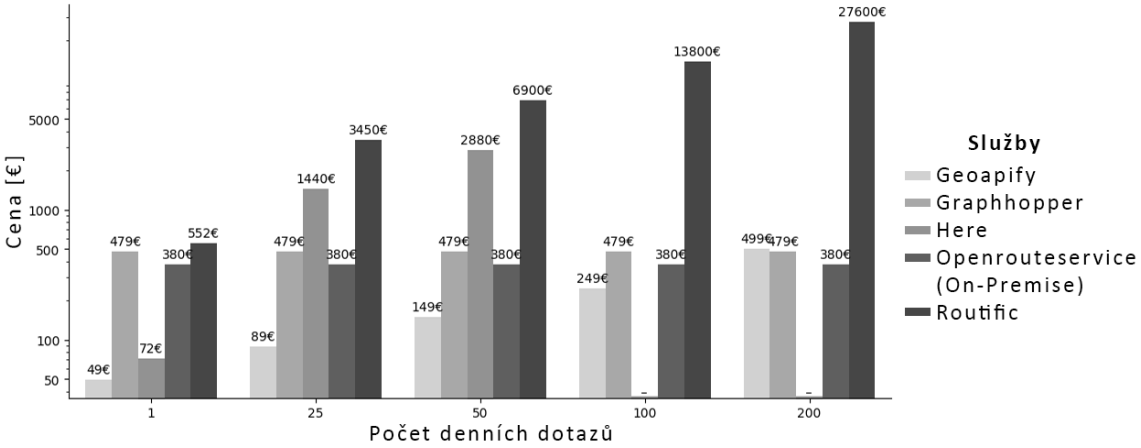
Cenové porovnání služeb



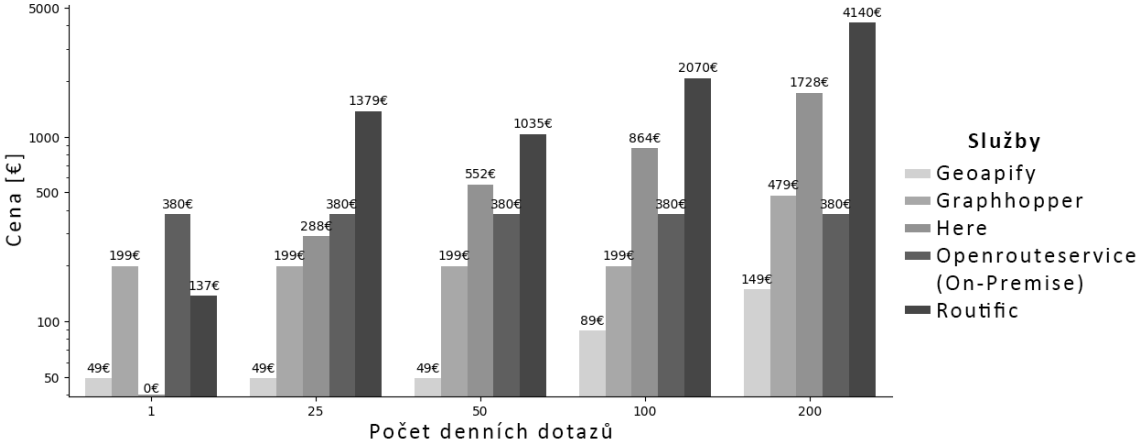
Obrázek B.1: 1 auto a 15 lokací v dotazu (*request*)



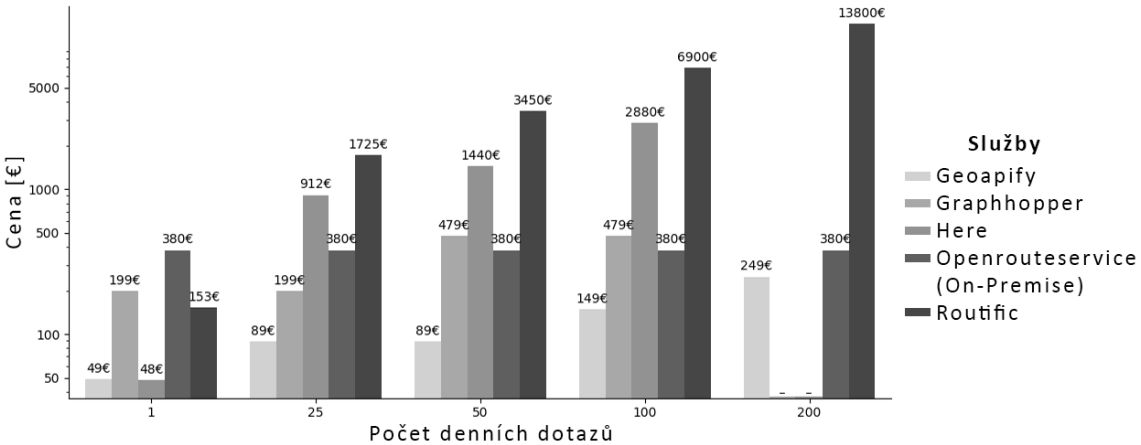
Obrázek B.2: 1 auto a 50 lokací v dotazu (*request*)



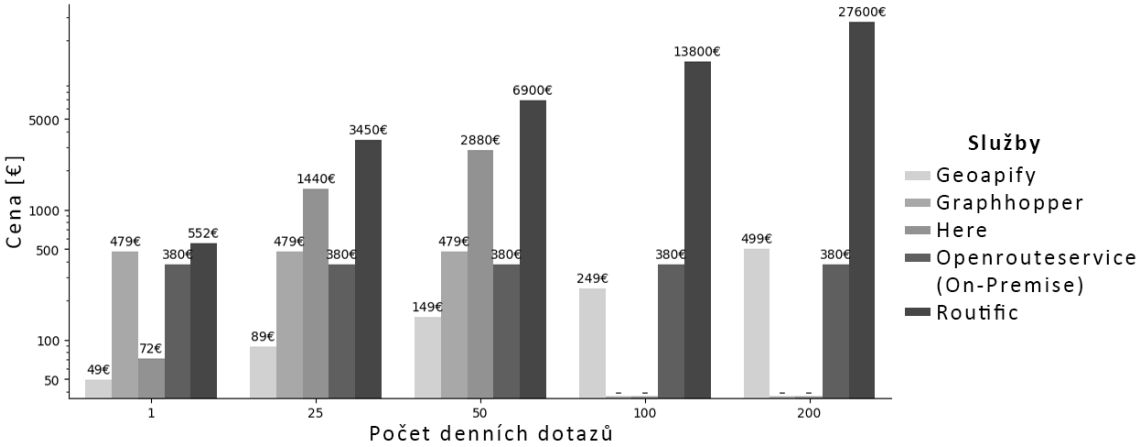
Obrázek B.3: 1 auto a 100 lokací v dotazu (*request*)



Obrázek B.4: 10 aut a 15 lokací v dotazu (*request*)



Obrázek B.5: 10 aut a 50 lokací v dotazu (*request*)



Obrázek B.6: 10 aut a 100 lokací v dotazu (*request*)