VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

NAVIGACE V MĚSTSKÉ HROMADNÉ DOPRAVĚ

NAVIGATION IN PUBLIC TRANSPORT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MATÚŠ TURIC**
AUTHOR

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

BRNO 2018

Abstrakt

Táto diplomová práca sa zaoberá problematikou cestovania v mestskej hromadnej doprave. V prvej časti autor informuje hlavne o podobných aplikáciách a teórií potrebnej k pochopeniu problematiky. V druhej časti autor popisuje návrh a fungovanie mobilnej a serverovej aplikácie vrátane použitých technológií. Vytvorená mobilná aplikácia má za úlohu navigovať užívateľa z jedného bodu do druhého, v čo najkratšom čase, pričom by malo byť možné nahradiť aktuálnu trasu rýchlejšou.

Abstract

This master's thesis deals with the issue of traveling in public transport. In the first part the author informs about similar applications and theories necessary for understanding the problems. In the second part, the author describes the design and behavior of the mobile and server application, including the used technologies. Created mobile application has the task of navigating the user from one point to another in the shortest possible time, while it should be possible to replace the current route with faster one.

Kľúčové slová

algoritmy, android, kotlin, spring boot, mobilná aplikácia, navigácia, teória grafov, webové služby.

Keywords

algorithms, android, kotlin, spring boot, mobile application, navigation, graph theory, web services.

Citácia

TURIC, Matúš. *Navigace v městské hromadné dopravě*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Navigace v městské hromadné dopravě

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Radka Burgeta, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Matúš Turic
19. mája 2018

Poďakovanie

Chcel by som sa poďakovať pánovi Ing. Radekovi Burgetovi Ph.D. za jeho odborné rady, cenné pripomienky a ľudský prístup pri vedení tejto diplomovej práce. Zároveň by som sa chcel poďakovať firme Kordis JMK a.s., ktorá poskytla dáta nevyhnutné k realizácii tejto diplomovej práce.

Obsah

Kapitola 1

Úvod

> Cestovanie je ako manželstvo. Určite sa pomýlite, ak si budete myslieť, že ho ovládate.
>
> John Steinbeck

V dnešnej dobe je dopravná infraštruktúra a služby, ktoré doprava poskytuje, neoddeliteľnou súčasťou každodenného života obyvateľov jednotlivých štátov. Infraštruktúra má vplyv na ekonomický rast, zamestnanosť a je kľúčová pre zahraničných investorov, rozvoj cestovného ruchu a znižovanie disparity medzi jednotlivými regiónmi.

Rozvoj dopravnej infraštruktúry je však finančne a technicky náročný proces, ktorý automaticky neznamená zjednodušenie alebo zrýchlenie cestovania pre ľudí. A práve čas je v dnešnom svete vzácnou komoditou, keďže ľudia sa neustále niekam ponáhľajú. Veľké množstvo času pri tom strávia cestovaním do svojej destinácie, a preto je vhodné tento čas nejakým spôsobom redukovať. To je jedným z hlavných dôvodov vzniku navigačných systémov, ktoré sú dnes hojne využívané v automobiloch, či mobilných telefónoch. Čo však mestská hromadná doprava?
Proces plánovania trasy pre automobily a mestskú hromadnú dopravu má značné rozdiely. Narozdiel od automobilovej dopravy, musí mestská hromadná doprava počítať s kombinovanou dopravou. Nevýhodou je tiež nemožnosť okamžitého cestovania, teda je potrebné navyše brať v úvahu aj jednotlivé časy odchodov a príchodov každých vozidiel, ktorými bude užívateľ cestovať. Na základe týchto faktov je potrebné namodelovať sieť predstavujúcu MHD ako hranovo ohodnotený orientovaný graf, v ktorom vrcholmi budú jednotlivé zastávky, a kde hrany budú predstavovať cesty, ktoré nimi prechádzajú. Cenou hrany je čas potrebný k prejdeniu z jednej zastávky na druhú. Najrýchlejšia cesta je potom vypočítaná pomocou algoritmu z teórie grafov.

Hlavným cieľom tejto diplomovej práce je vytvoriť mobilnú aplikáciu na platforme Android, ktorá bude schopná navigovať užívateľa do destinácie v čo najkratšom čase. V prípade nájdenia rýchlejšej trasy by mal byť užívateľ oboznámený, a zároveň by mal mať možnosť nahradiť súčasnú trasu rýchlejšou. Súčasťou riešenia je aj vytvorenie servera, ktorý bude poskytovať potrebné dáta. Výsledná aplikácia by mala mať prínos hlavne pre ľudí, ktorí využívajú mestskú hromadnú dopravu v Brne s ohľadom na aktuálnu dopravnú situáciu.

K popisu rôznych problémov, faktov, či implementačných detailov v rámci tejto diplomovej práce sa venuje nasledujúcich deväť kapitol.

V kapitole 2 sú porovnané ako existujúce riešenia dostupné z obchodu Google Play, tak aj riešenia vypracované na fakulte informačných technológií VUT v Brne.

Kapitola 3 sa venuje rozboru problémov spojených s cestovaním v MHD vrátane informácií o spôsobe, akým by sa dali tieto problémy riešiť.

Kapitola 4 obsahuje teoretické informácie o grafoch. Súčasťou tejto kapitoly je popis jednotlivých typov grafov, problému najkratšej trasy a tiež popis algoritmov, ktoré sa viažu s týmto problémom.

V kapitole 5 sa nachádzajú teoretické informácie popisujúce webové služby. Súčasťou tejto kapitoly je aj predstavenie poskytovateľa dát, ktorého dáta sú nevyhnutné k technickej realizácií tejto diplomovej práce.

Kapitola 6 popisuje návrh riešenia. V tejto kapitole sa okrem návrhu serverovej a klientskej časti nachádza aj popis architektúry riešenia.

Kapitola 7 popisuje najdôležitejšie technológie použité v rámci tejto diplomovej práce. Obsahuje ako technológie použité na vytvorenie klienta, tak aj technológie použité v mobilnej aplikácií.

Kapitola 8 sa venuje samotnej implementácii serverovej časti a mobilnej aplikácie.

Kapitola 9 sa zaoberá testovaním výsledného riešenia. V rámci tejto kapitoly sú k dispozícií spôsoby a výsledky testovania.

Poslednou kapitolou je kapitola 10, ktorá obsahuje možné rozšírenia do budúcnosti.

Kapitola 2

Analýza existujúcich mobilných aplikácií

Táto kapitola sa sústreďuje na popis funkčnosti a užívateľského rozhrania už existujúcich mobilných aplikácií, ktoré sú podobné vyvíjanej aplikácii. Kapitola sa bude tiež venovať analýze predchádzajúcich riešení vyvíjaných na fakulte informačných technológií VUT v Brne.

2.1 Oficiálne mobilné aplikácie

Existuje mnoho aplikácií, ktoré sa zaoberajú poskytovaním informácií o verejnej doprave. V rámci tejto práce budú rozobraté najmä najpoužívanejšie aplikácie, ktoré takéto informácie poskytujú v meste Brno.

2.1.1 IDOS

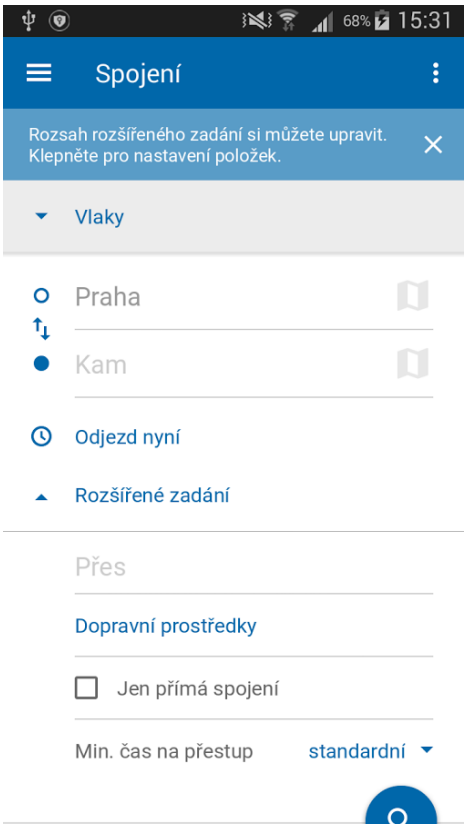
Aplikácia IDOS je oficiálnou aplikáciou Jízdnych řádů a slúži k jednoduchému vyhľadávaniu v cestovných poriadkoch vlakov a autobusov. Aplikácia tiež aktuálne podporuje zobrazenie MHD vo viac ako 110-tich českých mestách, pričom obsahuje pokročilejšie funkcie ako inteligentný našepkávač zastávok, notifikácie na príjazd a odjazd, offline prehliadanie vyhľadaných spojení, možnosť zakúpenia lístku cez SMS, alebo zobrazenie spoja na mape.

Aplikácia je zložená z toolbaru a obsahovej časti, ktorá sa mení v závislosti na aktívnej sekcií v postrannom panely. Toolbar slúži predovšetkým ako prístup k postrannému panelu, ktorý predstavuje navigáciu v aplikácii. Okrem toho toolbar obsahuje možnosti ako napríklad vymazanie formulára. Po zapnutí aplikácie sa automaticky zobrazuje sekcia spojení. Táto sekcia obsahuje jednoduchý formulár pre zadanie počiatočného miesta a destinácie. Tiež umožňuje zmeniť čas a detailnejšie upraviť trasu. Po rozkliknutí výsledku sa zobrazia bližšie informácie o spojení a možnosť zobraziť spojenie na mape.

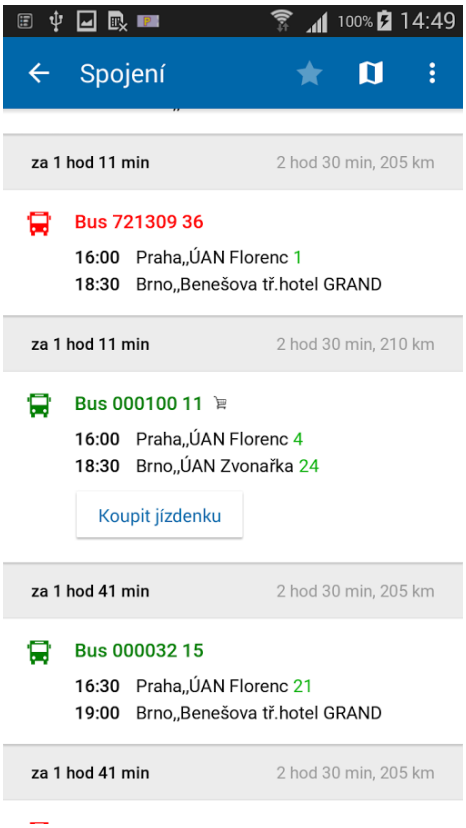
Sekcia s odchodmi obsahuje jednoduchý formulár, v ktorom je potrebné zadať počiatočné miesto a čas. Výsledkom je zoznam odchodov všetkých vyhovujúcich vozidiel zo zadaného miesta. Po rozkliknutí niektorého z výsledkov sa zobrazia nadchádzajúce zastávky a poznámky k trase. Poslednou sekciou je SMS lístok. V tejto sekcií je možné si zakúpiť lístok k MHD. Po kliknutí na niektorý z lístkov aplikácia otvorí SMS aplikáciu v telefóne a vyplní potrebné detaily. Túto SMS následne stačí už len odoslať.

Hlavnou výhodou aplikácie je široká podpora spojov. Aplikácia tiež podporuje online vyhledávanie, ktoré poskytuje aktuálne a oficiálne informácie vrátane oneskorenia a výluk.

Aplikácia má však aj svoje problémy. Je ju síce možné získať bezplatne prostredníctvom obchodu Google Play [15], ale obsahuje reklamy, ktorých je možné sa zbaviť len zaplatením poplatku na jeden rok alebo vyššieho poplatku natrvalo. Tieto reklamy pôsobia veľmi rušivo a kazia jednoduchý dizajn aplikácie. Aplikácia niekedy pôsobí až príliš spomalene a tiež nepracuje detailnejšie s aktuálnou polohou vozidiel.



Obr. 2.1: Vyhľadanie spojenia.



Obr. 2.2: Zoznam spojov.

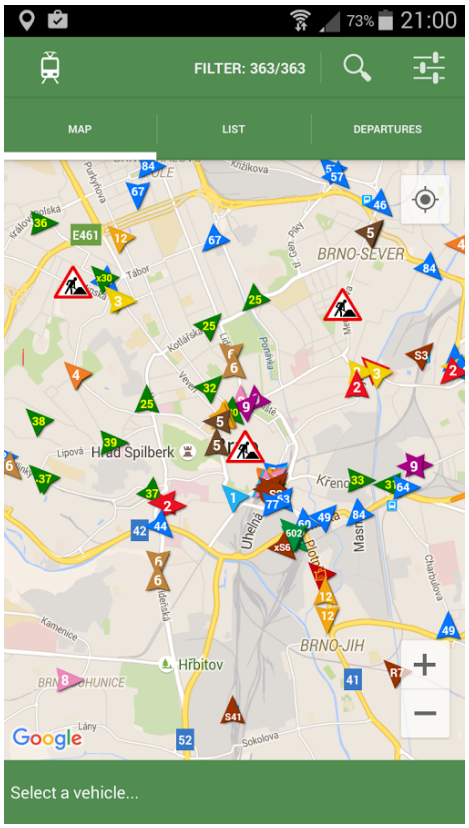
2.1.2 iRIS

Aplikácia iRIS slúži k zobrazeniu aktuálnej polohy všetkých vozidiel nachádzajúcich sa v systéme IDS JMK. Tento systém zahŕňa verejnú dopravu v Brne, Blansku, Kyjove, Břeclave, Vyškove, Adamove, Hodoníne a Znojme. Zahrnuté sú aj regionálne autobusy a vlaky. Systém tiež ponúka ďalšie rôzne informácie o vozidlách ako oneskorenie alebo typ vozidla. Poloha vozidiel je prenášaná cez dopravné centrá CEDRIS a RIS. Táto poloha je aktualizovaná každých 20 sekúnd.

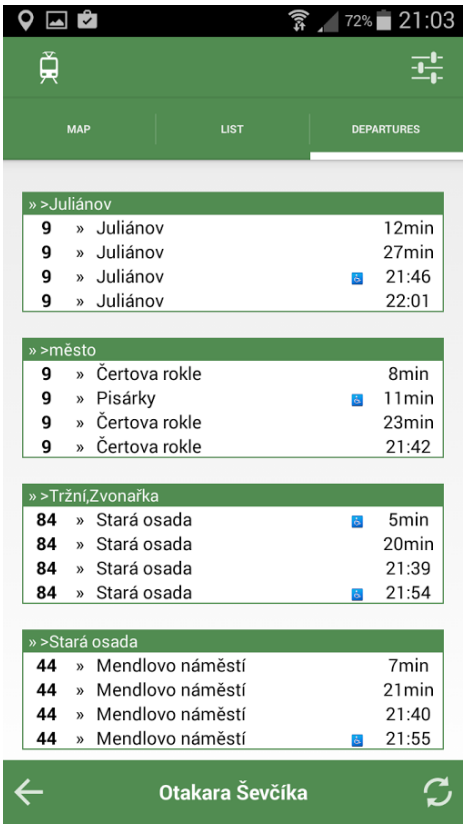
Aplikácia je zložená z toolbaru a troch tabov. V toolbare sa nachádzajú možnosti pre úpravu obsahu, ktoré sa menia s práve aktívnym tabom. Prvý tab obsahuje mapu so všetkými vozidlami a dolný toolbar s nápisom vyzývajúci užívateľa vybrať vozidlo. Po kliknutí na vozidlo sa zobrazia jeho podrobnosti a v dolnom toolbare sa okrem textu o vybratom vozidle zobrazia aj ďalšie možnosti. Druhý tab tvorí zoznam všetkých vozidiel. Po kliknutí na vozidlo v zozname sa toto vozidlo označí na mape v prvom tabe. Posledný tab obsahuje všetky najbližšie odjazdy z jednotlivých zastávok.

Medzi hlavné výhody aplikácie patria možnosti pre úpravu informácií zobrazených na mape a bezplatná dostupnnosť prostredníctvom obchodu Google Play [14]. Z technického

hľadiska teda aplikácia pôsobí pomerne dobre, avšak z pohľadu užívateľa to je už horšie. Vzhľadom na veľký počet vozidiel začína byť mapa na menšom displeji chaotická. Dolný toolbar zbytočne vyzýva užívateľa na vykonanie jednoduchého inštinktívneho rozhodnutia. Po vybratí vozidla zdieľa dolný toolbar duplicitné hodnoty s informáciami o vybratom vozidle. Pri niektorých použitých ikonách si užívateľ nie je istý, akú akciu budú vykonávať.



Obr. 2.3: Mapa s vozidlami.



Obr. 2.4: Odchody zo zastávky.

2.1.3 DPMBinfo

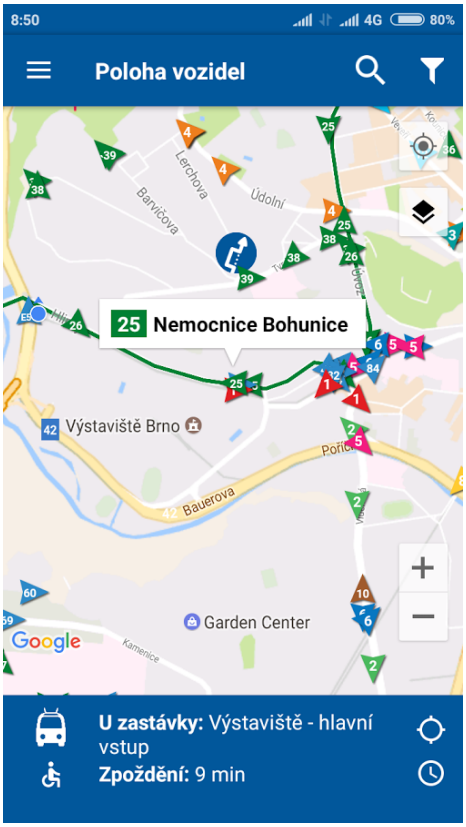
Nová aplikácia priamo od dopravného podniku mesta Brno. Ponúka aktuálne informácie o plánovaných zmenách v prevádzke MHD mesta Brno a verejnej dopravy v regióne. Tiež obsahuje informácie o polohe vozidiel, ich odchodoch, či prípadných mimoriadnych udalostiach v reálnom čase.

Podobne ako aplikácia iRIS, aj aplikácia DPMBinfo je zložená z toolbaru a obsahovej časti, ktorá sa mení v závislosti na aktívnej sekcií v postrannom panely. Po zapnutí aplikácie sa automaticky zobrazuje zoznam všetkých sekcií. Prvou sekciou sú aktuálne odjazdy z jednotlivých zastávok. Druhou sekciu je poloha vozidiel. Táto sekcia zobrazuje všetky vozidlá na mape spolu s relevantnými informáciami k vozidlám. Ďalšími sekciami sú vyhľadanie spojenia, zoznam vozidiel a zakúpenie lístka pomocou SMS.

Hlavnú výhodu aplikácie tvoria sekcie informujúce o zmenách v doprave, mimoriadnych udalostiach, novinkách a obsahu jednotlivých tarifov. Výhodou je tiež možnosť zobraziť schémata liniek, príjemné užívateľské rozhranie a bezplatné získanie cez obchod Google Play [13]. Podobne ako predchádzajúce aplikácie, ani táto aplikácia neponúka bližšiu formu

navigácie v doprave. Nevýhodou je tiež sekcia parkoviská, ktorá užívateľa presmeruje priamo do webového prehliadača.

Obr. 2.5: Informácie o zmenách v doprave.



Obr. 2.6: Mapa s vozidlami.

2.2 Riešenia FIT VUT

V rámci štúdia alebo práce na fakulte informačných technológií VUT v Brne sa človek prakticky neustále stretáva s nejakou problematikou zo sveta IT, ktorú je možné pokúsiť sa vyriešiť prostredníctvom diplomovej práce. Táto sekcia je venovaná zadaniam, ktoré v minulosti podobnú tému už riešili.

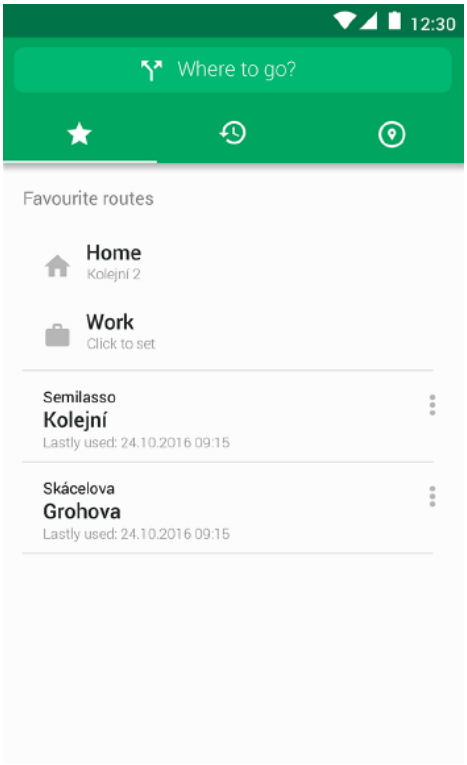
2.2.1 Sledování pohybu v MHD

Jedná sa o prvú z aplikácií, ktoré boli vyvíjané ako diplomová práca na fakulte informačných technológií VUT v Brne [27]. Podobne ako predchádzajúce aplikácie, aj táto obsahuje aktuálne informácie o polohe vozidiel, ich odchodoch a trasách.

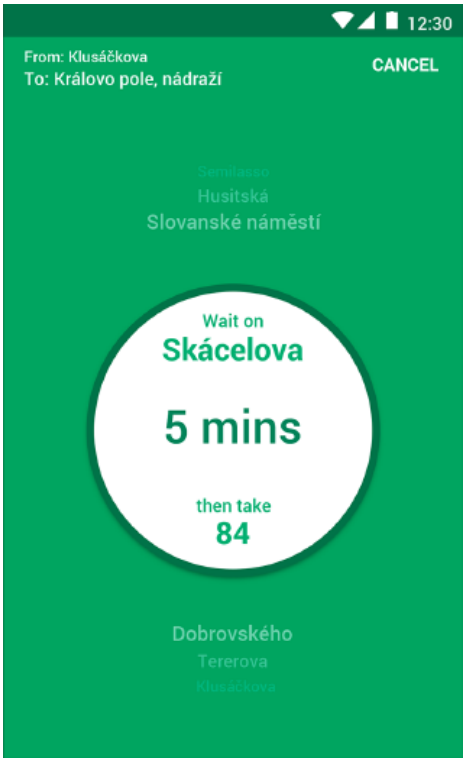
Hneď po spustení aplikácie sa užívateľovi zobrazí úvodná obrazovka pozostávajúca z poľa pre vyhľadávanie a troch tabov, ktoré slúžia ako navigácia. Prvý tab obsahuje zoznam obľúbených trás. Druhý tab zobrazuje zoznam histórie plánovania. Posledný tab ponúka prehľad odchodov jednotlivých spojov v rámci okolia, v ktorom sa užívateľ nachádza. Po vyplnení poľa destináciou začne proces plánovania trasy, s ktorým sa zmení aj obrazovka. Pomocou riadiacich prvkov na obrazovke má užívateľ možnosť presne špecifikovať jednotlivé parametre vyhľadávania ako napríklad počiatočnú zastávku, cieľovú zastávku, deň a

čas trasy alebo počet prestupov, či čas potrebný k jednotlivým prestupom. Po kliknutí na výsledok sa na novej obrazovke zobrazí detail trasy, ktorý obsahuje funkciu live directions. Túto funkciu je možné aktivovať už zo zoznamu výsledkov vyhľadávania. Po zapnutí tejto funkcie sa zobrazí posledná obrazovka, kde sa v hornej časti nachádza stručný popis trasy. Napravo od popisu je tlačidlo, ktoré funkciu ukončí. Táto obrazovka tiež obsahuje kruhový indikátor, ktorý popisuje práve prebiehajúcu akciu. Pod indikátorom sa nachádza zoznam troch predchádzajúcich zastávok. Nad indikátorom sa zas nachádza zoznam troch nadchádzajúcich zastávok. Pri prestupe medzi zastávkami sa v kruhovom indikátore navyše zobrazí šípka smerujúca k požadovanej zastávke. Táto šípka teda slúži ako kompas.

Hlavnú výhodu aplikácie tvorí funkcia live directions, ktorá prestavuje navigáciu. Výhodou je tiež intuitívny dizajn a použitie grafov pre výpočet minimálnej cesty. Výsledné riešenie je použiteľné v praxi, no pridaná hodnota nie je príliš veľká. Aplikácií by prospela ako podpora offline plánovania, tak aj indikácia nedostupnosti servera.



Obr. 2.7: Hlavná obrazovka.



Obr. 2.8: Režim navigácie.

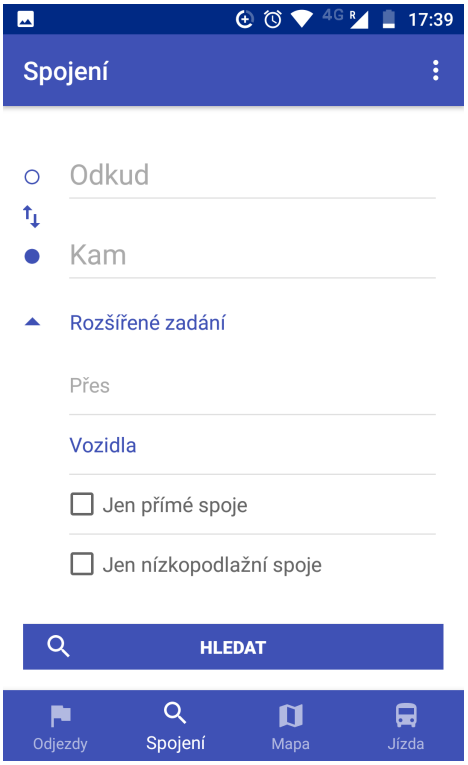
2.2.2 Mobilní asistent pro cestování v MHD

Mobilný asistent pre cestovanie v MHD je druhou aplikáciou, ktorá bola vyvíjaná ako diplomová práca na fakulte informačných technológií VUT v Brne [32]. Aplikácia ponúka aktuálne informácie o polohe vozidiel, ich odchodoch a trasách.

Narozdiel od predchádzajúcich riešení, vsadila aplikácia na použitie navigácie v toolbare, ktorý bol umiestnený na spodku. Po zapnutí aplikácie sa automaticky zobrazí sekcia pre vyhľadávanie odchodov. Táto sekcia obsahuje len textové pole pre výber zastávky a vysúvacie menu s možnosťami výberu medzi vyhľadávaním podľa klasického cestovného poriadku alebo podľa vozidiel. Druhá sekcia slúži k vyhľadaniu podľa spojenia. Sekcia obsa-

huje textové polia pre zadanie počiatočnej zastávky a destinácie. Súčasťou je tiež tlačidlo pre rozbalenie ponuky s ďalšími možnosťami pre špecifikáciu vyhľadávaného spojenia. Treťou sekciou je mapa s vozidlami. Po kliknutí na ikonu vozidla sa zobrazí informačné dialógové okno so základnými informáciami o vozidle a následne je jeho trasa vykreslená na mape. Poslednou sekciou je režim jazdy. Po vybratí najbližšieho vozidla v okolí sa sleduje jeho trasa vo forme zobrazenia predošlej, aktuálnej a následujúcej zastávky. Sekcia tiež zobrazuje odchody z následujúcej zastávky. Pri zadaní cieľovej stanice sa naplánuje optimálna trasa. Informácie sa aktualizujú v závislosti na zmene pozície alebo každých desať sekúnd. Pri nezhode pozície vozidla a užívateľa sa zobrazí dialógové okno s varovaním.

Hlavnú výhodu aplikácie tvorí dizajn, ktorý je síce veľmi jednoduchý, ale bol upravovaný na základe výhrad testerov. Výsledné riešenie je ale pomerne jednoduché. K hľadaniu nadväzujúcich spojení autor nevyužíva grafové algoritmy a používa teda skôr naivný prístup. Spoj je možné vyhľadať len od aktuálneho času. Programové riešenie je síce použiteľné v praxi, no výsledná aplikácia prakticky neprináša nič, čím by sa odlíšila od už existujúcich. Narozdiel od predchádzajúceho riešenia, aplikácia informuje o nedostupnosti internetu alebo lokalizačných služieb. Bohužiaľ však túto informáciu zobrazuje len vo forme dialógového okna pri spustení aplikácie, a to aj v prípade, keď sú obe služby dostupné.



Obr. 2.9: Vyhľadanie spojení.

Obr. 2.10: Odchody zo zastávky.

Kapitola 3

Analýza problému cestovania s MHD

Táto kapitola sa venuje problematike cestovania v MHD, ktorej diagram je uvedený na obrázku 3.1. Kapitola sa tiež zaoberá riešením rôznych problémov v rámci tohto cestovania.

3.1 Problém polohy užívateľa a zastávky

Riešenie tohto problému je pomerne jednoduché. Mobilná aplikácia porovná súradnice z GPS senzora v mobilnom telefóne so súradnicami zastávky poskytnutých DPMB. V prípade, že vzdialenosť bude menšia ako 45 metrov, sa bude predpokladať, že sa užívateľ nachádza na zastávke.

3.2 Problém nástupu do vozidla

V tomto prípade je potrebný zásah užívateľa, ktorý potvrdí nastúpenie do vozidla. Užívateľovi ako pomôcka poslúži číslo vozidla a čas odjazdu. Keďže linky zdieľajú rovnaký identifikátor, je potrebné kontrolovať aj čas. V prípade, že užívateľ nenastúpi do vozidla a aktuálny čas bude väčší ako čas odjazdu, bude nutné aby aplikácia vyhľadala novú trasu. Implementácia bez zásahu užívateľa nie je možná, keďže firma Kordis JMK aktualizuje polohu vozidiel približne každých pätnásť až tridsať sekúnd. Tento fakt by spôsoboval problém najmä pri zastávkach na znamenie. Problémom je tiež pomerne častá strata signálu, čo by zastavilo chod aplikácie, až kým by sa signál znova neobjavil.

3.3 Problém nenastúpenia do vozidla

Ako už bolo spomenuté pri probléme nástupu do vozidla, aplikácia sa nezaobíde bez zásahu užívateľa, ktorý musí potvrdiť nastúpenie do vozidla. V prípade, že bude aktuálny čas väčší ako čas odjazdu a zároveň užívateľ nepotvrdil nástup do vozidla, tak to aplikácia bude považovať za nenastúpenie do vozidla. Následne je nutné, aby aplikácia na základe aktuálnych súradníc zastávky vyhľadala novú trasu do destinácie.

3.4 Problém polohy užívateľa v rámci vozidla

V tomto prípade je riešenie jednoduché, kým užívateľ nepotvrdí vystúpenie z vozidla, bude aplikácia predpokladať, že sa užívateľ nachádza vo vozidle. Ak užívateľ potvrdí vystúpenie z vozidla, aplikácia bude predpokladať, že sa užívateľ vo vozidle už nenachádza.

3.5 Problém vyhľadania rýchlejšej trasy

Akonáhle bol prekročený rozsah 60 metrov od zastávky, tak aplikácia automaticky vykoná vyhľadanie rýchlejšej trasy z práve príchodzej zastávky. V prípade, že sa nájde rýchlejšia trasa, tak aplikácia poskytne užívateľovi možnosť nahradenia trasy touto novo vyhľadanou trasou. V inom prípade sa pokračuje použitím aktuálnej trasy.

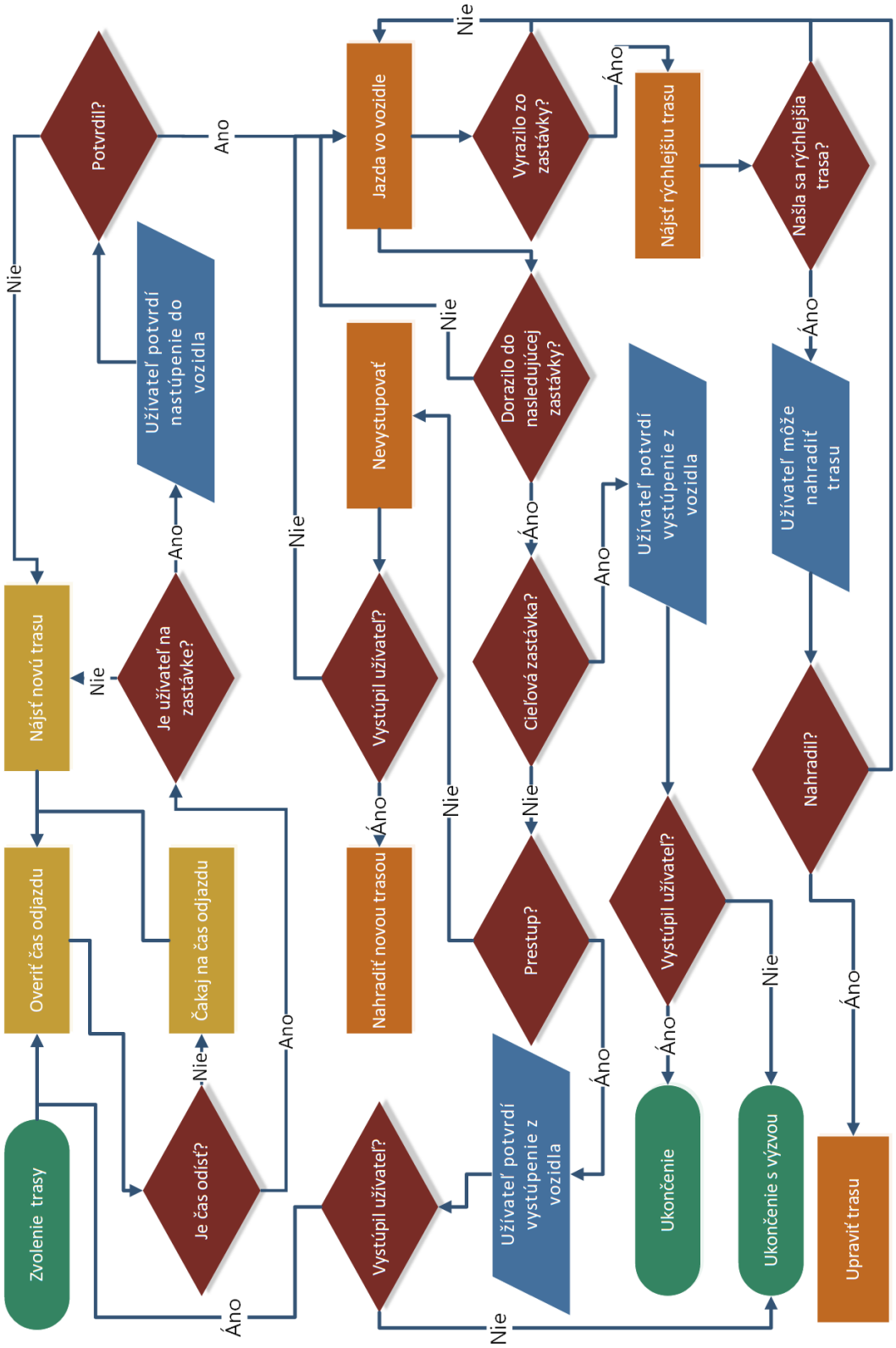
3.6 Problém vystúpenia z vozidla

Ako už bolo spomenuté pri probléme polohy užívateľa v rámci vozidla, aplikácia bude predpokladať, že sa užívateľ vo vozidle už nenachádza, ak užívateľ potvrdí vystúpenie z vozidla. Aplikácia však potrebuje detegovať nielen vystúpenie na správnej zastávke, ale aj vystúpenie užívateľa na nesprávnej zastávke a adekvátne reagovať. V prípade, že užívateľ vystúpil na nesprávnom mieste a vozidlo ešte nedorazilo do destinácie, je potrebné, aby aplikácia vyhľadala novú trasu do destinácie na základe polohy aktuálnej zastávky. V inom prípade aplikácia užívateľa vyzve, aby vystúpil na nasledujúcej zastávke a vyhľadal z nej novú trasu.

3.7 Problém prestupu

V prípade, že trasa obsahuje prestup, ktorý ma práve nastať, tak je potrebné, aby mobilná aplikácia opäť riešila už známe problémy:

1. Problém vystúpenia z vozidla.
2. Problém polohy užívateľa a zastávky.
3. Problém nástupu do vozidla.



Obr. 3.1: Diagram problematiky cestovania v MHD.

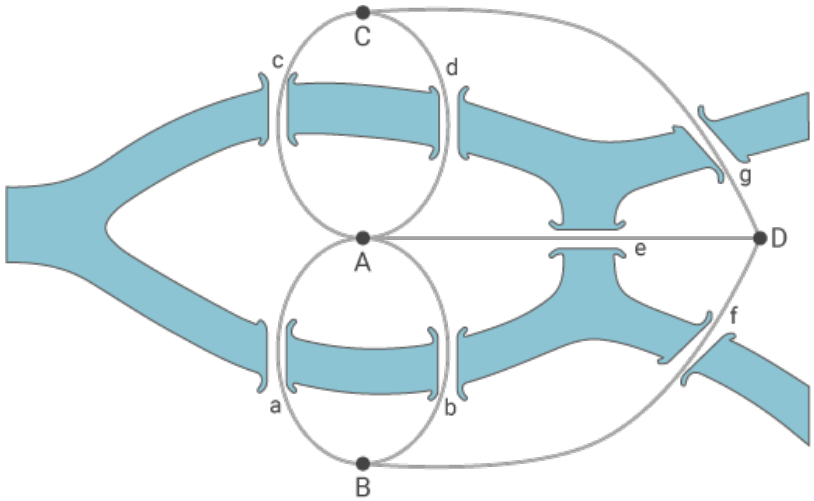
Kapitola 4

Teória grafov

Táto kapitola obsahuje popis teoretickej časti grafov, vrátane vysvetlenia základných pojmov a ďalších súčastí, ktoré sú nutné k pochopeniu nasledujúcich kapitol.

Mnohé situácie v reálnom svete môžu byť jednoducho opísané pomocou diagramu pozostávajúceho z množiny bodov spolu s čiarami spájajúcimi určité páry týchto bodov. Jednotlivé body potom môžu napríklad reprezentovať mestá v nejakom štáte a čiary cesty, ktoré tieto mestá spájajú, alebo ľudí, pričom čiary by predstavovali nejakú formu vzťahu, ako napríklad priateľstvo [7]. Matematická abstrakcia situácií takéhoto typu sa nazýva graf, pričom v konečnom dôsledku nás len zajíma, či sú dva body spojené čiarou. Spôsob akým sú dva body spojené je pre nás nepodstatný.

Vznik modernej teórie grafov je viazaný na rok 1936, kedy maďarský matematik Dénes König publikoval prvú monografiu z teórie grafov. Prvky myslenia používaného súčasnou teóriou grafov sa ale sporadicky objavovali už dávno predtým. Korene histórie teórie grafov je možné vyhľadať už v roku 1735, keď švajčiarsky matematik Leonhard Euler vyriešil problém siedmich mostov mesta Kaliningrad[1] ilustrovaného na obrázku 4.1.



Obr. 4.1: Problém siedmich mostov mesta Kaliningrad s Eulerovou grafickou reprezentáciou.

[1] Königsberg bridge problem

Podstata problému spočívala v nájdení cesty cez každý zo siedmich mostov, ktoré sa rozprestierajú cez vidlicovú rieku, ktorá preteká okolo ostrova, ale bez toho, aby dvakrát prekročila hocijaký mostík. Euler tvrdil, že takáto cesta neexistuje. Jeho dôkaz zahŕňal len odkazy na fyzické usporiadanie mostov, avšak v podstate dokázal prvý teorém v teórii grafov [9].

Predmet teórie grafov sa postupne rozrástol do významnej oblasti matematického výskumu s aplikáciami v oblasti chémie, operačného výskumu, spoločenských vied a informatiky.

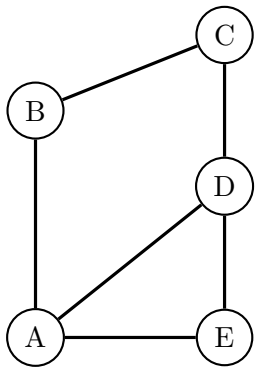
4.1 Typy grafov

Koncepčne si graf predstavíme ako súbor uzlov a hrán, ktoré tieto uzly spájajú. Formálne však hovoríme, že obyčajný graf je dvojica $G = (U, H)$, kde U je konečná množina uzlov (vrcholov) a $H \subseteq \big\{\{u, v\} : u, v \in U \wedge u \neq v\big\}$ je (konečná) množina hrán. V teórií grafov existuje viacero delení grafov. V rámci tejto práce nás hlavne zaujíma delenie grafov podľa hrany.

4.1.1 Neorientovaný graf

Neorientovaný graf G je zložený z množiny V vrcholov (uzlov) a množiny H hrán tak, že každá hrana $h \in H$ je priradená neusporiadanej dvojici (dvojprvkovej množine) vrcholov $u, v \in V$. Pokiaľ existuje jediná hrana $h \in H$, ktorá je priradená dvojici vrcholov $u, v \in V$, tak potom $h \equiv u, v$ [21].

Všeobecne môže byť jednej dvojici vrcholov priradených viacero hrán, ktoré sa potom nazývajú ako násobné. Tieto hrany takisto nemajú smer (sú orientované do oboch smerov). Vďaka tomu je možné každý neorientovaný graf konvertovať na ekvivalentný orientovaný graf, nahradením každej neorientovanej hrany dvomi orientovanými hranami v opačnom smere. Štruktúry, ktoré takýto typ grafu využívajú sú napríklad stromy.

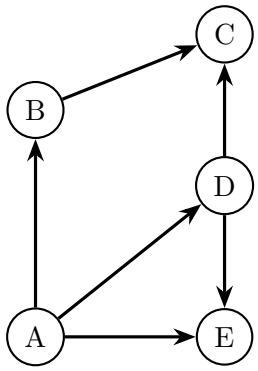


Obr. 4.2: Príklad jednoduchého neorientovaného grafu.

4.1.2 Orientovaný graf

Formálne sa orientovaný graf G skladá z množiny V vrcholov (uzlov) a množiny H hrán tak, že každej hrane $h \in H$ je priradená usporiadaná dvojica $(u, v) \in V \times V$ vrcholov $u, v \in V$. Pokiaľ existuje jediná hrana $h \in H$, ktorá je priradená dvojici (u, v) vrcholov $u, v \in V$, tak potom $h \equiv (u, v)$ [21].

Orientované grafy riešia prípady, ktoré neorientovné grafy riešiť nedokážu. Jednoduchým príkladom by bola situácia v doprave, kde je potrebné brať do úvahy aj jednosmerné ulice. V praxi sa využívajú hlavne acyklické orientované grafy [11] (napr. v genealógii[2]).

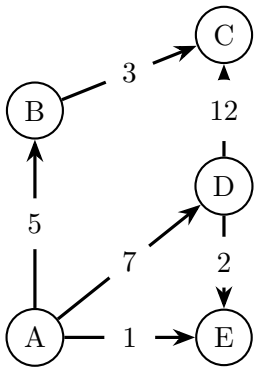


Obr. 4.3: Príklad jednoduchého orientovaného grafu.

4.1.3 Hranovo ohodnotený graf

Tento typ grafu môže byť orientovaný aj neorientovaný. Formálne hovoríme, že nech existuje G graf s množinou vrcholov (uzlov) V a množinou hrán H. Nech $g : H \to R$ je zobrazenie, potom f sa nazýva hranovým ohodnotením grafu G. Dvojica (G, g) sa nazýva hranovo ohodnotený graf [21].

V jednoduchosti to znamená, že každá hrana má určitú cenu. V prípade tejto práce bude touto cenou čas, ktorý ubehne kým sa užívateľ dostane z jednej zastávky na druhú.



Obr. 4.4: Príklad jednoduchého hranovo ohodnoteného grafu.

4.2 Problém najkratšej trasy

Keďže táto práca sa zaoberá cestovaním vo verejnej doprave, tak je potrebné riešiť problém najkratšej trasy, ktorý je jedným zo základných problémov v teórii grafov. Podstatou tohto problému je vytvorenie cesty v rámci daného grafu z vopred známeho počiatočného bodu

[2] Veda skúmajúca vývoj rodov a vzťah medzi rodovo príbuznými jedincami.

do ľubovoľného koncového bodu, ktorý sa nachádza v danom grafe, pričom zvolená cesta, ktorá reprezentuje určitú veličinu (v tomto prípade čas), musí byť čo najmenšia.

Formálne je možné tento problém zapísať nasledujúcim spôsobom. Majme ohodnotený orientovaný graf $G = (V, E)$ a váhovú funkciu $w : E \to \mathbb{R}$. Cena cesty $p = \langle v_0, v_1, \ldots, v_k \rangle$ je suma

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Cena najkratšej cesty z u do v je

$$\delta(u, v) = \begin{cases} min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{pokiaľ existuje cesta z } u \text{ do } v \\ \infty & \text{inak} \end{cases}$$

Najkratšia cesta u do v je potom ľubovoľná cesta p z u do v s $w(p) = \delta(u, v)$ [19]. V prípade, že je každá z hrán grafu je ohodnotená rovnakou cenou, tak je takáto nájdená cesta zároveň aj cestou s najmenším počtom hrán.
Problém najkratšej trasy má nasledujúce varianty:

- Hľadanie najkratšej cesty z daného počiatočného uzla do všetkých ostatných uzlov v grafe (single-source shortest-paths problem).
- Hľadanie najkratšej cesty z daného počiatočného uzla do daného koncového uzla v grafe (point-to-point shortest-paths problem).
- Hľadanie najkratšej cesty medzi ľubovolnými dvoma uzlami v grafe (all-pairs shortest-paths problem).

4.3 Grafové algoritmy

Existuje viacero algoritmov, ktoré riešia problematiku hľadania najkratšej trasy. Táto práca sa zaoberá hlavne hľadaním najkratšej cesty zo všetkých uzlov v grafe do daného konečného uzla (single-destination shortest-paths problem), avšak tento problém je možné redukovať na problém hľadania najkratšej cesty z daného počiatočného uzla do všetkých ostatných uzlov v grafe obrátením hrán v orientovanom grafe.

4.3.1 Dijkstrov algoritmus

Dijkstrov algoritmus bol publikovaný v roku 1959 holandským počítačovým vedecom Edsgerom Wybe Dijkstrom. Algoritmus vytvorí strom najkratších ciest od počiatočného vrcholu do všetkých ostatných bodov grafu. Príklad implementácie Dijkstrovho algoritmu [26] je zobrazený pomocou zápisu 1 v jazyku Python.

Dijkstrov algoritmus patrí spolu s Bellman-Fordovým algoritmom medzi typické prehľadávacie algoritmy riešiace už zmienený problém. Dijkstrov algoritmus je však možné implementovať tak, že jeho časová zložitosť je nižšia ako u algoritmu Bellman-Forda, avšak je ho možné použiť len pre ohodnotené orientované grafy bez záporných ohodnotení. Keďže cenou hrany bude čas, tak nás tento nedostatok až tak trápiť nemusí.

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = {}
cost_so_far = {}
came_from[start] = None
cost_so_far[start] = 0

while not frontier.empty():
        current = frontier.get()

        # optimalizacia
        if current == goal:
                break

        for next in graph.neighbors(current):
                new_cost = cost_so_far[current] + graph.cost(current, next)

                if next not in cost_so_far or new_cost < cost_so_far[next]:
                        cost_so_far[next] = new_cost
                        priority = new_cost
                        frontier.put(next, priority)
                        came_from[next] = current
```

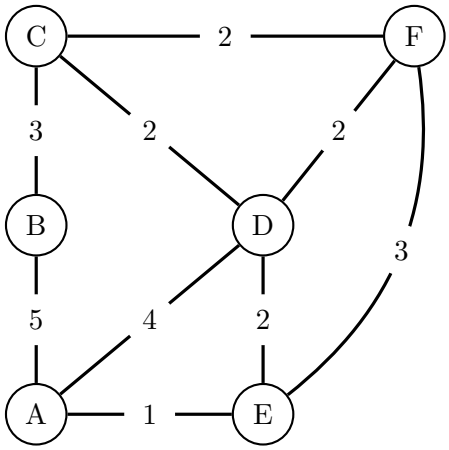
Zápis 1: Dijkstrov algoritmus [26].

Chovanie algoritmu je nasledovné [30]:

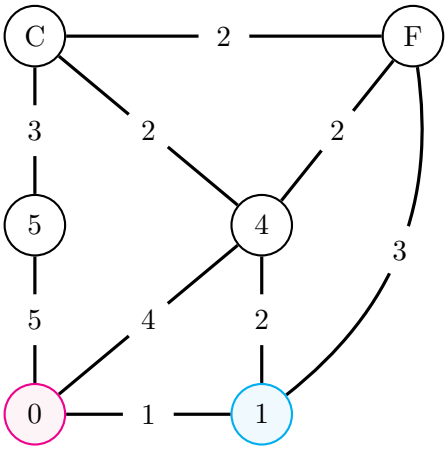
1. Na začiatku zvoľ začiatočné a koncové uzly, a pridaj štartovací uzol do množiny vyriešených uzlov (t.j. množina uzlov so známou najkratšou cestou od štartovacieho uzla) s hodnotou 0.
2. Prejdi prehliadaním do šírky z počiatočného uzla k najbližším susedom a zaznamenaj dĺžku cesty ku každému susednému uzlu.
3. Zober najkratšiu cestu k jednému z týchto susedov (v prípade rovnakej ceny je výber náhodný) a označ tento uzol za vyriešený, pretože teraz poznáme najkratšiu cestu od štartovacieho uzla k tomuto uzlu.
4. Z množiny vyriešených uzlov navštív najbližších susedov a zaznamenaj dĺžky ciest zo začiatočného uzla k týmto novým uzlom. Nenavštevuj žiadne uzly, ktoré už boli vyriešené, pretože k nim už najkratšie cesty poznáme.
5. Opakuj kroky 3 a 4, až kým označený cieľový uzol nie je vyriešený.

Princíp fungovania Dijkstrovho algoritmu ukazuje séria obrázkov 4.5. Počiatočným uzlom je bod A a koncovým bod C. Pre lepšie pochopenie by jednotlivé uzly mohli predstavovať zastávky, kde hrany by boli ulice, ktoré ich spájajú. Cenou hrany by bol čas potrebný k prejdeniu z jednej zastávky na druhú.

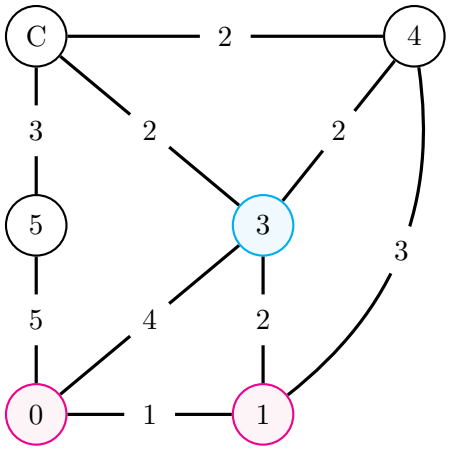
Keďže Dijkstrov algoritmus počíta iba dĺžky relatívne malej podmnožiny možných ciest v grafe, tak je je pomerne účinný. Vyriešením vrchola je známa najkratšia cesta od počiatočného vrchola, čo uľahčuje ďalšie prehľadávanie. Najrýchlejšia známa implementácia



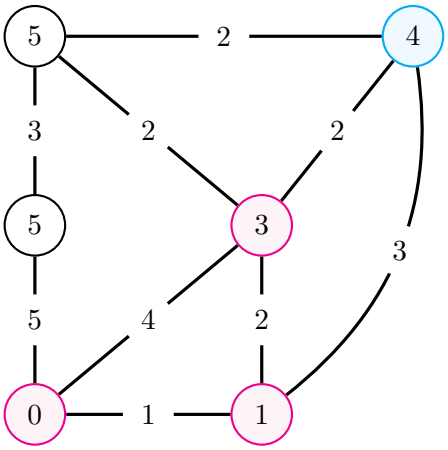
(a) Počiatočný stav.



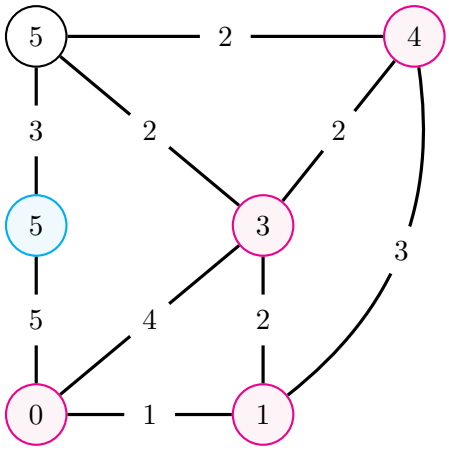
(b) Prvá iterácia.



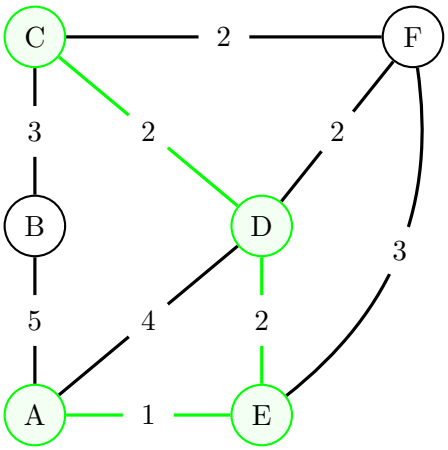
(c) Druhá iterácia.



(d) Tretia iterácia.



(e) Štvrtá iterácia.



(f) Výsledný podgraf.

Obr. 4.5: Priebeh fungovania Dijkstrovho algoritmu.

Dijkstrovho algoritmu v najhoršom prípade má výkon $O(|R| + |N|\, log\, |N|)$, kde R je počet vzťahov v grafe a N je počet vrcholov.

4.3.2 Algoritmus A*

Algoritmus prvýkrát popísali v roku 1968 Peter Hart, Nils Nilsson a Bertram Raphael. Rovnako ako Dijkstra, tak aj A* pracuje tým spôsobom, že vytvorí stromovú cestu s najnižším celkovým ohodnotením z počiatočného uzla do cieľového uzla. Príklad implementácie algoritmu A* [26] je zobrazený pomocou zápisu 2 v jazyku Python.

Základného chovanie algoritmu A* je nasledovné [28]:

1. Označ štartovací uzol ako otvorený a vypočítaj evaluačnú funkciu pre tento uzol.
2. Vyber uzol n označeného ako otvorený, ktorého hodnota evaluačnej funkcie je najmenšia. Vyber ľubovoľne, ale vždy v prospech niektorého koncového uzla.
3. Ak je uzol n koncový, tak ho označ za uzavretý a ukonči algoritmus.
4. V inom prípade, označ uzol n ako uzavretý. Vypočítaj evaluačnú funkciu pre každého následníka n a označ každého následníka, ktorý nebol označený ako uzavretý, za otvorený. Zmeň označenie hocijakého uzavretého uzla n_i na otvorený, ak je tento uzol následníkom uzla n a zároveň je hodnota evaluačnej funkcie tohto uzla teraz menšia, ako keď bol označený za uzavretý. Vráť sa na krok 2.

Pre pre mnohé vyhľadávania je algoritmus A* lepším, pretože pre každý uzol je použitá funkcia $f(n)$, ktorá dáva odhad celkovej ceny pri použití tohto uzla. Preto sa A* nazýva ako heuristická funkcia, ktorá sa líši od algoritmu v tom, že heuristika je skôr odhad a nemusí byť nevyhnutne správna. Evaluačná funkcia $f(n)$, ktorú A* využíva má nasledujúci tvar [28]:

$$f(n) = g(n) + h(n),$$

kde

- $f(n)$ = celková odhadovaná cena cesty cez uzol n.
- $g(n)$ = aktuálna cena k dosiahnutiu uzla n.
- $h(n)$ = odhadovaná cena z n do cieľa. Toto je heuristická časť funkcie, teda sa jedná o odhad.

Použitie dobrej heuristiky je dôležité pri určovaní výkonnosti algoritmu A*. Výpočet heuristickej funkcie $h(n)$ sa môže napríklad uskutočniť nasledujúcimi spôsobmi [1]:

- $h(n) = |x_{start} - x_{destination}| + |y_{start} - y_{destination}|$ - Táto funkcia sa nazýva ako Manhattanská vzdialenosť a je štandardnou heuristikou pre sieť. Nazýva sa tak preto, pretože sa vypočítava počítaním celkového počtu štvorcov presunutých horizontálne a vertikálne tak, aby bolo dosiahnuté cieľové pole od aktuálneho štvorca. Ignorujeme diagonálny pohyb a prekážky, ktoré by mohli byť v ceste.

- $h(n) = \sqrt{(x_{start} - x_{destination})^2 + (y_{start} - y_{destination})^2}$ - Táto funkcia sa nazýva ako Euklidovská vzdialenosť. Uvedená funkcia je presnejšia, ale je tiež pomalšia, pretože k nájdeniu cesty musí preskúmať väčšiu oblasť.

- $h(n) = 0$ - V tomto prípade sa algoritmus A* stáva Dijkstrovým algoritmom, ktorý zaručuje nájdenie najkratšej cesty.

Heuristická funkcia musí byť prípustná, čo znamená, že nemôže nadhodnotiť cenu k dosiahnutiu cieľa. Spôsoby uvedené vyššie sú prípustné.

Hlavnou nevýhodou algoritmu A* je jeho náročnosť na pamäť. Algoritmus A* je v praxi ťažko priestorovo obmedzený a na aktuálnych počítačoch nie je praktickejší ako algoritmus vyhľadávania nazývaný best-first.

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = {}
cost_so_far = {}
came_from[start] = None
cost_so_far[start] = 0

while not frontier.empty():
   current = frontier.get()

   if current == goal:
      break

   for next in graph.neighbors(current):
      new_cost = cost_so_far[current] + graph.cost(current, next)
      if next not in cost_so_far or new_cost < cost_so_far[next]:
         cost_so_far[next] = new_cost
         priority = new_cost + heuristic(goal, next)
         frontier.put(next, priority)
         came_from[next] = current
```

Zápis 2: Algoritmus A* [26].

Kapitola 5

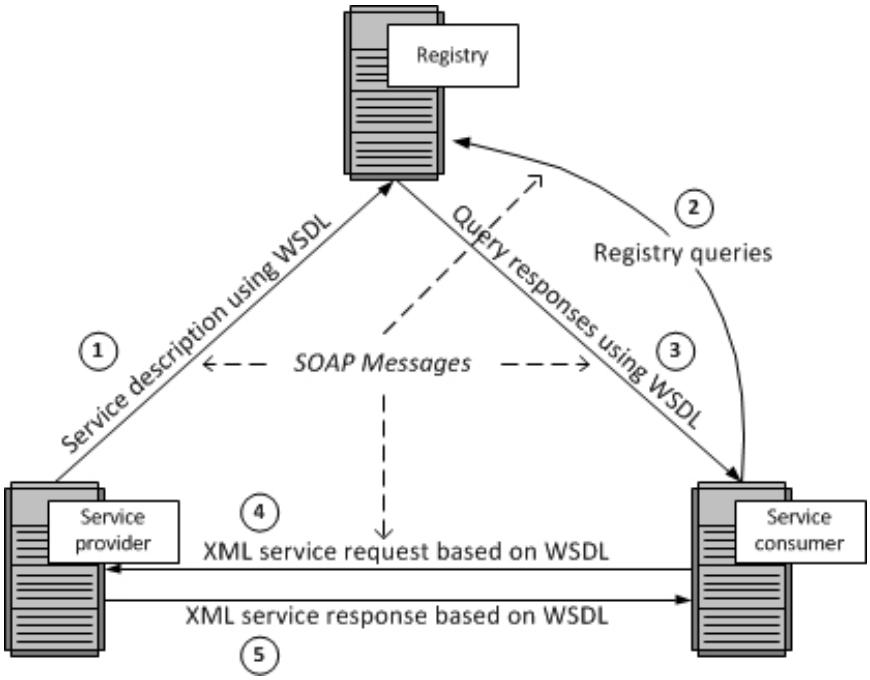
Webové služby

Táto kapitola sa zaoberá ako informáciami o webových službách, ktoré sú súčasťou riešenia tejto diplomovej práce, tak aj informáciami o externom poskytovateľovi dát.

Pojem webové služby je daný historicky, ale dnes už nejde len o služby poskytované prostredníctvom webu. Okrem iného hrozí zámena s rovnakým označením činnosti firiem vytvárajúcich WWW stránky alebo poskytujúcich priestor k umiestneniu WWW stránok. Webové služby sa preto označujú ako terminus technicus.

Webové služby sú reinkarnáciou technológií pre vzdialené volanie funkcií v distribuovaných systémoch, ako RPC, CORBA alebo RMI1 [22]. Technológia webových služieb je jednoduchá, otvorená a nezávislá na platforme.

Hlavnou výhodou webových služieb je, že sa súbor HTML stránok zmení na súbor statických alebo dynamických XML stránok, ktoré sú jednoducho čitateľné programami.



Obr. 5.1: Jednoduchý príklad fungovania webových služieb [6].

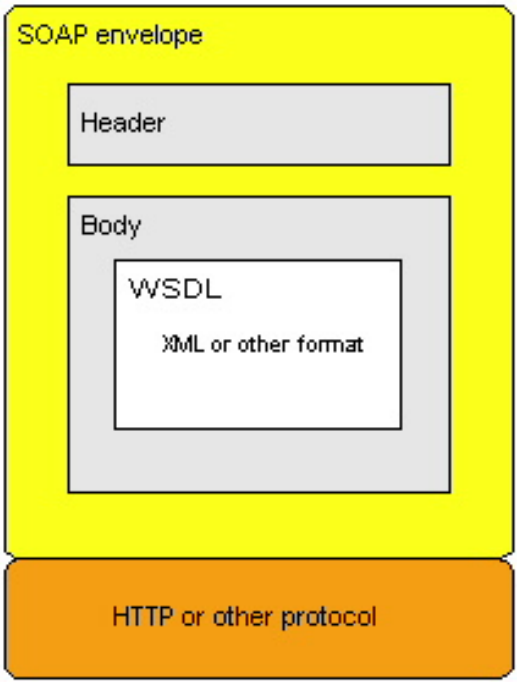
Ako je možné vidieť aj na obrázku 5.1, tak technológia webových služieb je zložená z troch častí:

- Protokol pre vzdialené volanie procedúr - SOAP.
- Jazyk pre popis poskytovaných sužieb - WSDL.
- Mechanizmus pre nájdenie služieb - UDDI.

5.1 SOAP

SOAP[1] je ľahký protokol, ktorého štandard je spravovaný konzorciom W3 [34]. Je určený na výmenu štruktúrovaných informácií v decentralizovanom a distribuovanom prostredí. Využíva technológie XML na definovanie rozširujúceho sa frameworku pre zasielanie správ, ktorý poskytuje štruktúru správ, ktorá môže byť vymenená prostredníctvom rôznych základných protokolov. Tento framework bol navrhnutý tak, aby bol nezávislý od akéhokoľvek konkrétneho programovacieho modelu a inej implementačne špecifickej sémantiky.

Soap má dva hlavne ciele - jednoduchosť a rozšíriteľnosť. Tieto ciele sa pokúša splniť vynechaním funkcií, ktoré sa často vyskytujú v distribuovaných systémoch. Takéto funkcie zahŕňajú okrem iného spoľahlivosť, bezpečnosť, koreláciu, smerovanie a vzory pre výmenu správ (MEP)[2]. SOAP je alternatívou k technológiam ako REST[3] alebo JSON[4] [4]. Všeobecne používa HTTP, ale môžu byť použité aj iné spôsoby pripojenia, ako napríklad SMTP.



Obr. 5.2: Štruktúra SOAP [4].

[1] Simple Object Access Protocol
[2] Message Exchange Patterns
[3] Representational State Transfer
[4] JavaScript Object Notation

5.1.1 Štruktúra správ SOAP

Správa SOAP je zadaná ako infoset XML, ktorého položky poznámok, elementov, atribútov, menného priestoru a znakových informácií môžu byť serializované ako XML 1.0. Správa SOAP obsahuje nasledujúce časti:

- **Obálka** - predstavuje koreňový prvok SOAP správy. Obálka je zložená z dvoch hlavných častí, a to z hlavičky a tela.
- **Hlavička** - poskytuje informácie o autentifikácii, kódovaní údajov alebo o tom, ako by ju mal spracovať príjemca. Hlavička je voliteľnou časťou obálky.
- **Telo** - obsahuje správu, ktorá predstavuje požiadavok alebo odpoveď od konkrétnej webovej služby. Tieto správy je možné definovať pomocou špecifikácie WSDL. Telo je povinnou časťou obálky.

```
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xs="http://www.w3.org/1999/XMLSchema"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
 <env:Header/>
 <env:Body>
  <m:jePrvocislo xmlns:m="urn:mojeURI">
    <cislo xsi:type="xs:long">1987</cislo>
  </m:jePrvocislo>
 </env:Body>
</env:Envelope>
```

Zápis 3: Príklad jednoduchej SOAP správy [22].

```
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
 <env:Body>
  <ns1:jePrvocisloResponse
    xmlns:ns1="urn:mojeURI"
    env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <return xsi:type="xsd:boolean">true</return>
  </ns1:jePrvocisloResponse>
 </env:Body>
</env:Envelope>
```

Zápis 4: Príklad odpovedi na SOAP správu [22].

5.2 WSDL

Ak by sme nevedeli, aké funkcie sa dajú zavolať (vrátane ich parametrov a návratových hodnôt), tak by nám bola možnosť na diaľku zavolať funkcie pomocou SOAP úplne k ničomu. Tento problém však rieši jazyk WSDL[5], ktorý je založený na XML a vo veľkej miere využíva štandardy XML Namespaces a XML Schema.

WSDL opisuje webové služby začínajúce správami, ktoré sa vymieňajú medzi žiadateľom a agentmi providera. Samotné správy sú opísané abstraktne a potom viazané na konkrétny sieťový protokol a formát správ. Definície webových služieb je možné mapovať na ľubovoľný implementačný jazyk, platformu, objektový model alebo systém zasielania správ. Pokiaľ sa odosielateľ aj príjemca dohodnú na popise služby (napríklad súbor WSDL), tak implementácie za webovými službami môžu byť hocijaké.

5.2.1 Štruktúra WSDL

Štruktúru WSDL tvoria nasledujúce tri časti:

- Definície.
- Operácie.
- Väzby služby.

Definície sú vo všeobecnosti vyjadrené v XML a zahŕňajú ako definície typov dát, tak aj definície správ, ktoré definície typu dát používajú. Tieto definície sú zvyčajne založené na niektorých dohodnutých XML slovných zásobách. XML Namespaces sa používajú na zabezpečenie jedinečnosti názvov XML elementov v definíciách, operáciách a väzbách služby. XML však nie je jediný jazyk pre popis definícií. Namiesto XML sa dá použiť aj jazyk IDL[6].

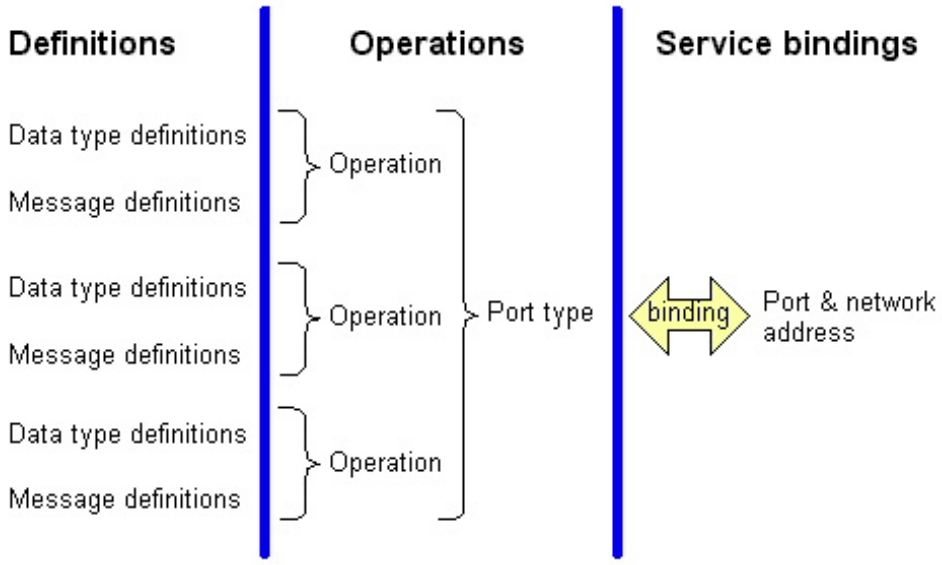
Operácie opisujú akcie pre správy podporované webovou službou. K dispozícii sú nasledujúce štyri typy operácií:

- **Jednosmerne** - vyžadujú sa správy odoslané bez odpovede.
- **Žiadosť / odpoveď** - odosielateľ pošle správu a príjemca odošle odpoveď.
- **Požiadať odpoveď** - žiadosť o odpoveď.
- **Oznámenie** - správy sú odosielané viacerým príjemcom.

Operácie sú následne zoskupené do typov portov, ktoré definujú súbor operácií podporovaných webovou službou.

Väzby služby pripájajú typy portov k portu. Port je definovaný priradením sieťovej adresy k typu portu. Zbierka portov definuje službu. Táto väzba sa bežne vytvára použitím SOAP, ale môžu sa použiť aj iné formy, ktoré môžu zahŕňať .NET, JMS[7] alebo IIOP[8].

[5] Web Services Description Language
[6] Interface Definition Language
[7] Java Message Service
[8] Internet Inter-ORB Protocol



Obr. 5.3: Vzťah základných častí WSDL [5].

5.3 UDDI

UDDI[9] poskytuje infraštruktúru pre softvérové prostredie založené na webových službách pre verejne dostupné služby, ako aj pre služby, ktoré sú v rámci organizácie odhalené iba interne. Hlavnou myšlienkou je odhaliť organizácie a ich služby, pričom by to malo formu podobnú telefónnemu zoznamu. UUID poskytuje:

- Definíciu súboru služieb podporujúcich popis a objavovanie podnikov, organizácií a iných poskytovateľov webových služieb.
- Webové služby, ktoré jednotlivé organizácie a podniky ponúkajú.
- Technické rozhrania, ktoré sa môžu použiť na prístup k týmto službám.

Samotný systém UDDI Business Registry sa skladá z troch adresárov:

- Biele UDDI stránky, ktoré obsahujú základné informácie, ako je názov spoločnosti, adresa a telefónne čísla, ako aj ďalšie štandardné obchodné identifikátory (daňové čísla).
- Žlté UDDI stránky, ktoré ponúkajú podrobné obchodné údaje organizované príslušnými obchodnými klasifikáciami. Verzia žltých UDDI stránok klasifikuje firmy podľa novších kódov NAICS[10].
- Zelené UDDI stránky, ktoré zahŕňajú informácie o kľúčových obchodných procesoch spoločnosti, ako napríklad operačnú platformu, podporované programy alebo nákupné metódy.

[9] Universal Description, Discovery, and Integration
[10] North American Industry Classification System

5.4 Kordis JMK

Firma Kordis JMK začala svoje pôsobenie v meste Brno v roku 2002 [20]. Bola založená za účelom vykonávania dvoch hlavných činností:

- Koordinácia základnej dopravnej obslužnosti na území Juhomoravského kraja.
- Príprava, realizácia a prevádzkovanie integrovaného dopravného systému postupne na celom území Juhomoravského kraja.

Okrem týchto činností sa táto spoločnosť zaoberá aj sledovaním a vyhodnocovaním prepravných potrieb, návrhom cestovných poriadkov, optimalizáciou vedenia liniek alebo vedením trvalej informačnej a propagačnej kampane. Tiež spolupracuje na modernizácií a vývoji vozového parku, či vybavení zastávok a prestupových terminálov.

Aby bolo možné splniť ciel tejto diplomovej práce, je potrebné mať dáta obsahujúce dopravné informácie a vedieť s nimi ďalej pracovať. Získanie takýchto dát je však nad časové a finančné možnosti študenta. Tieto dáta teda poskytuje už vyššie spomenutá firma Kordis JMK, a.s, ktorá spolupracuje s fakultou informačných technológií VUT. V poskytnutých dátach sú obsiahnuté ako statické informácie (názov, poloha a popis zastávok, ...), tak aj dynamické informácie (aktuálne meškanie, poloha vozidla, ...). Dáta sú dostupné z dvoch zdrojov.

Prvým zdrojom je webová služba typu SOAP s WSDL popisom, ktorá poskytuje rôzne aktuálne informácie vrátane GPS súradníc vozidiel, štatistických údajov alebo aktuálne meškanie spoja.

Druhým zdrojom je FTP server, ktorý je aktualizovaný približne každých päť až sedem dní. Server obsahuje hlavne informácie o jednotlivých zastávkach, ako sú napríklad názov, zóna alebo GPS súradnice zastávky. Tiež obsahuje informácie o cestovných poriadkoch alebo popis služieb.

5.4.1 Popis rozhrania poskytujúceho live dáta

Vďaka už zmienenej spolupráci s firmou Kordis JMK, nie je potrebné implementovať webovú službu pre prístup k live dátam. Z WSDL súboru je možné túto službu automaticky vygenerovať použitím napríklad vývojového prostredia IntelliJ Idea[11] od firmy JetBrains. Z vygenerovanej služby je potom možné jednoducho pristupovať k jednotlivým triedam a ich metódam, ktoré poskytujú potrebné live dáta. Následne už len stačí zanalyzovať vygenerované triedy, vybrať triedy a metódy nevyhnutné k riešeniu tejto diplomovej práce, a vykonať nad nimi rôzne výpočty, či transformácie.

INearDeparturesService

Trieda poskytujúca aktuálne informácie o odchodoch vozidiel zo zastávky. Táto trieda obsahuje nasledujúcich päť metód:

- **GetNearDepartures** - metóda vracia všetky odchody na základe čísla zastávky a čísla stĺpika.
- **GetNearDeparturesWithCount** - metóda vracia všetky odchody na základe čísla zastávky a čísla stĺpika. Navyše je však potrebné špecifikovať počet odjazdov, ktoré má metóda vrátiť.

[11] https://www.jetbrains.com/idea

- **GetNearDeparturesByTime** - metóda vracia všetky odchody na základe čísla zastávky, čísla stĺpika, hodiny a minúty.
- **GetNearDeparturesByTimeWithCount** - metóda vracia všetky odchody na základe čísla zastávky, čísla stĺpika, hodiny a minúty. Navyše je však potrebné špecifikovať počet odjazdov, ktoré má metóda vrátiť.
- **GetNearDeparturesAcrossPoints** - metóda vracia všetky odchody na základe čísla zastávky.

Jednotlivé metódy vracajú potrebné informácie o odchodoch (meno cieľovej stanice, číslo linky, čas odchodu, ...) zabalené v objekte. V rámci tejto diplomovej práce je nevyhnutná metóda **GetNearDeparturesAcrossPoints**.

ITrafficState

Trieda poskytujúca všetky vozidlá, ktoré sa aktuálne nachádzajú v teréne. Táto trieda obsahuje nasledujúce tri metódy:

- **GetTrafficState** - metóda vracia zoznam všetkých dostupných vozidiel MHD.
- **GetTrafficStateVLDBusService** - metóda vracia zoznam všetkých vozidiel patriacich ostatným dopravcom v Juhomoravskom kraji (diaľkové spoje).
- **GetTrafficStateDPMBWoService** - metóda vracia zoznam všetkých služobných jázd.

Jednotlivé metódy vracajú potrebné informácie o vozidlách (číslo trasy, číslo vozidla, poloha, ...) zabalené v objekte. V rámci tejto diplomovej práce je nevyhnutná metóda **GetTrafficState**.

5.4.2 Popis súborov obsahujúcich statické dáta

Ako už bolo spomenuté vyššie, v rámci spolupráce bol tiež poskytnutý prístup k FTP serveru obsahujúceho súbory nevyhnutné k riešeniu tejto diplomovej práce.

Súbor obsahujúci zastávky

Tento súbor obsahuje informácie o všetkých zastávkach. Štruktúra súboru je nasledujúca:

```
01001 101 'Achtelky' 'Achtelky' 'Achtelky' 'Achtelky' 'Achtelky'
  S01 +099252582 +295112496 M 000 '>Kam.vrch'
  S02 +099260496 +295111644 S 000 '>Mysl,Pisárky'
  S03 +099240792 +295114782 M 000 '>Koh.od K.v.'
  S04 +099252582 +295112496 S 000 '>Koh.od Mys.'
...
```

Zápis 5: Príklad štruktúry súboru so zastávkami.

Zastávka predstavuje jedno miesto a obsahuje viacero stĺpikov. V súbore je zastávka zložená z čísla zastávky, zóny a názvu (má viacej variant). Stĺpik sa skladá z čísla stĺpika, koordinátov a informačného popisu. Koordináty sú však vo vlastnom formáte, a je teda potrebné ich previesť pomocou nasledujúcej formuly:

```
P = X alebo Y / 100000, kde X, Y sú koordináty
° = P / 60 (výsledok sa zaokrúhľuje nadol)
´ = ((P / 60) - °) * 60 (výsledok sa zaokrúhľuje nadol)
´´ = ((((P / 60) - °) * 60) - ´) * 60 (výsledok sa zaokrúhľuje nadol)
´´´ = ((((((P / 60) - °) * 60)-´) * 60) - ´´) * 60 (výsledok sa zaokrúhľuje nadol)
```

Zápis 6: Formula na prevod koordinátov.

Súbor obsahujúci služby

Tento súbor obsahuje informácie o službách (spojoch). V jednom dni môže byť dostupná len jedna služba rovnakého čísla. Štruktúra súboru je nasledujúca:

```
#017000101 01 01 1 D 48
L001 C00039 000
 Z01 09410 002 05:16 05:16 N MN NN PN
 Z02 01343 002 05:17 05:17 N MN NN PN
 Z03 01769 002 05:17 05:17 N MN NN PN
 Z04 01768 002 05:18 05:18 N MN NN PN
...
```

Zápis 7: Príklad štruktúry súboru so službami.

V súbore je služba zložená z čísla služby, dopravcu, prevádzkarne, druhu dopravy a typu služby (denná / nočná). Pod službou sa nachádzajú linky, ktoré obsahujú zastávky. Linka sa skladá z čísla linky, cieľa a čísla spoja. Zastávky sa skladajú z poradového čísla, čísla zastávky, čísla stĺpika, času príjazdu, času odjazdu, typu odjazdu (verejný / neverejný), míľnika a náväznosti.

Súbor predstavujúci deň

Tento súbor obsahuje informácie o službách (spojoch), ktoré jazdia v daný deň. Štruktúra súboru je nasledujúca:

```
#2018-02-23
 002010401
 002010402
 002010403
...
```

Zápis 8: Príklad štruktúry súboru so dostupnými službami v daný deň.

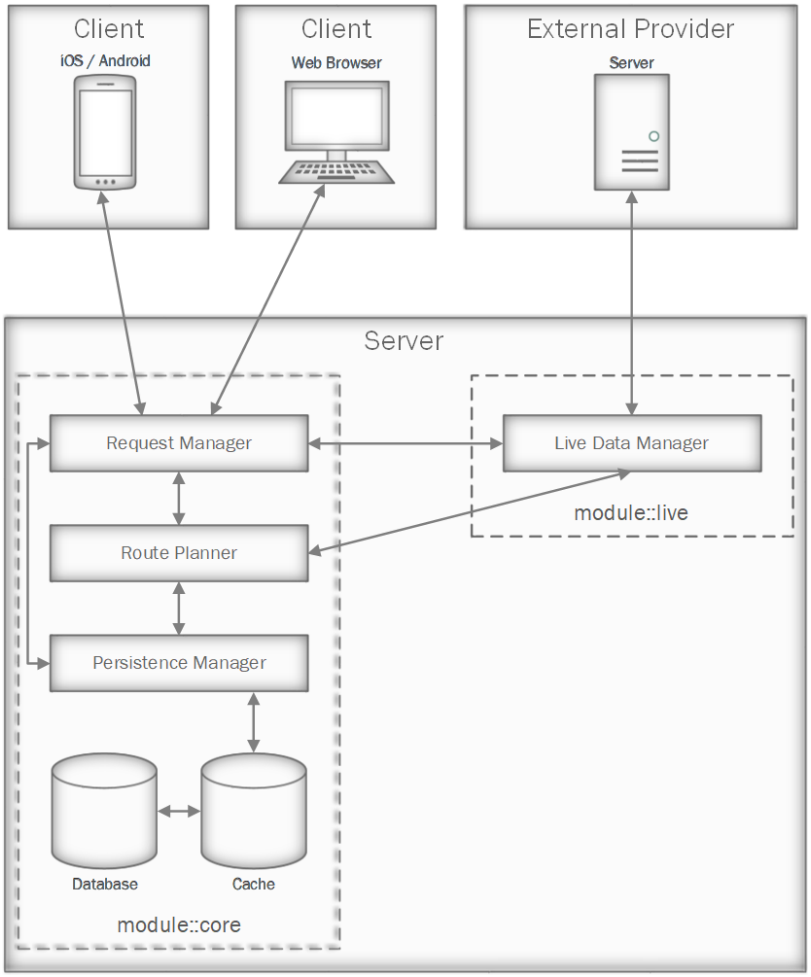
V súbore je deň definovaný len dátumom. Pod týmto dátumom sa nachádzajú dostupné služby v tento deň. Každých päť až sedem dní je dostupných sedem takýchto súborov.

Kapitola 6

Návrh

Táto kapitola sa zaoberá návrhom vhodnej architektúry. Súčasťou je tiež návrh serverovej a mobilnej časti riešenia.

6.1 Architektúra



Obr. 6.1: Architektúra riešenia.

Typ požiadávky	Popis
Plánovanie trasy	Nastáva komunikácia s časťou Route planner, pomocou ktorej sa vykonajú všetky potrebné kroky k naplánovaniu trasy. Ak si trasa vyžaduje dáta zo systému Kordis, tak táto časť môže komunikovať aj s časťou Live data manager, ktorá tieto dáta poskytne.
Aktuálne údaje	Ak má prijatá požiadavka obsahovať ako odpoveď aktuálne údaje napr. o odjazdoch alebo polohe pozidiel, tak Request manager priamo kontaktuje Live data manager, ktorý tieto dáta poskytne. V prípade, že sa jedná o dáta, ktoré môžu byť poskytnuté viacerým užívateľom ako napr. zoznam všetkých vozidiel na mape, tak Request manager kontaktuje Persistance manager, ktorý mu dáta poskytne z Cache.
Statické dáta	Ak má prijatá požiadavka vrátiť ako odpoveď statické dáta, tak Request manager naviaže komunikáciu s s Persistence manager, ktorý tieto dáta poskytne.

Tabuľka 6.1: Prehľad typov požiadavok.

Z obrázku 6.1 uvedeného vyššie vyplýva, že serverová časť aplikácie je zložená z dvoch modulov: core a live. Modul core obsahuje časti request manager, route planner a persistence manager. Tento modul príjme všetky požiadavky od klientov, ktoré následne spracuje na základe typu prijatej požiadavky. Prehľad jednotlivých požiadavok je bližšie popísaný v tabuľke 6.1.

Úlohou core modulu je okrem prijímania požiadavok od klientov, aj spracovávať statické dáta pridávané administrátorom, akými sú napríklad cestovné poriadky, či aktualizované informácie o zastávkach.

Modul live slúži k získavaniu a spracovaniu reálnych dát z externých zdrojov. Ako už bolo spomenuté v kapitole 5, tak v rámci tejto diplomovej práce je použitý len jeden externý zdroj, a to webová služba spoločnosti Kordis JMK. Vďaka tejto službe sú nám poskytnuté rôzne informácie, ktoré sú nevyhnutné pre správne naplánovanie trasy v rámci mestskej hromadnej dopravy v Brne.

6.2 Server

Aby bolo možné používať mobilnú aplikáciu, treba najprv disponovať dátami, ktoré budú obsahovať požadované informácie o doprave. Tieto dáta bude poskytovať už zmienená firma Kordis JMK. Na navrhnutom servery bude potom vďaka klientovi, ktorý bude vygenerovaný z WSDL súboru, prebiehať vyhľadanie trasy, vyhľadanie najbližších odjazdov zo zastávky, vyhľadanie vozidiel, či vyhľadanie rýchlejšej trasy do destinácie.

6.2.1 API

Veľkou výhodou je, ak je možné klientskú a serverovú aplikáciu samostatne vyvíjať bez akejkoľvek závislosti na seba. Klient by mal vedieť iba URI[1] zdrojov. Vďaka tomuto prístupu je možné vyvíjať klienta na rôznych platformách a nemeniť pri tom implementáciu servera. Preto bude server implementovaný v jazyku Java s využitím frameworku Spring Boot a architektúry REST.

V REST architektúre je rozhranie definované pomocou zdrojov, ktoré sú identifikované prostredníctvom URI adries. Ak by sme chceli potom manipulovať so zdrojmi, tak je to možne prostredníctvom protokolu HTTP a metód GET, POST, PUT a DELETE. Keďže server bude poskytovať len dáta, bude použitá len metóda GET.

Spring Boot framework bude použitý najmä z dôvodu urýchlenia vývoja eliminovaním konfigurácie pomocou XML súborov a využitím vloženia závislostí. Ako databáza bude použitá NoSQL databáza MongoDB, hlavne z dôvodu rýchlosti a podpory ovládačov pre reaktívne vyhľadávanie v rámci frameworku Spring Boot. Tieto technológie budú bližšie predstavené v kapitole 7.
Pre splnenie funkčnosti musí server podporovať minimálne nasledujúce operácie:

- **Poskytnúť všetky pozície zastávok** - bude potrebné filtrovať súbor so zastávkami a získať tak požadované informácie.
- **Poskytnúť informácie o vozidlách** - potrebné informácie budú dostupné prostredníctvom triedy `TrafficState` a jej metód.
- **Poskytnúť informácie o odjazdoch** - použitá bude trieda `NearDeparturesService`, ktorá tieto informácie uchováva.
- **Poskytnúť informácie o spojoch** - bude potrebné mať informácie o zastávkach a polohe vozidiel, a použiť súbor obsahujúci informácie o službách.

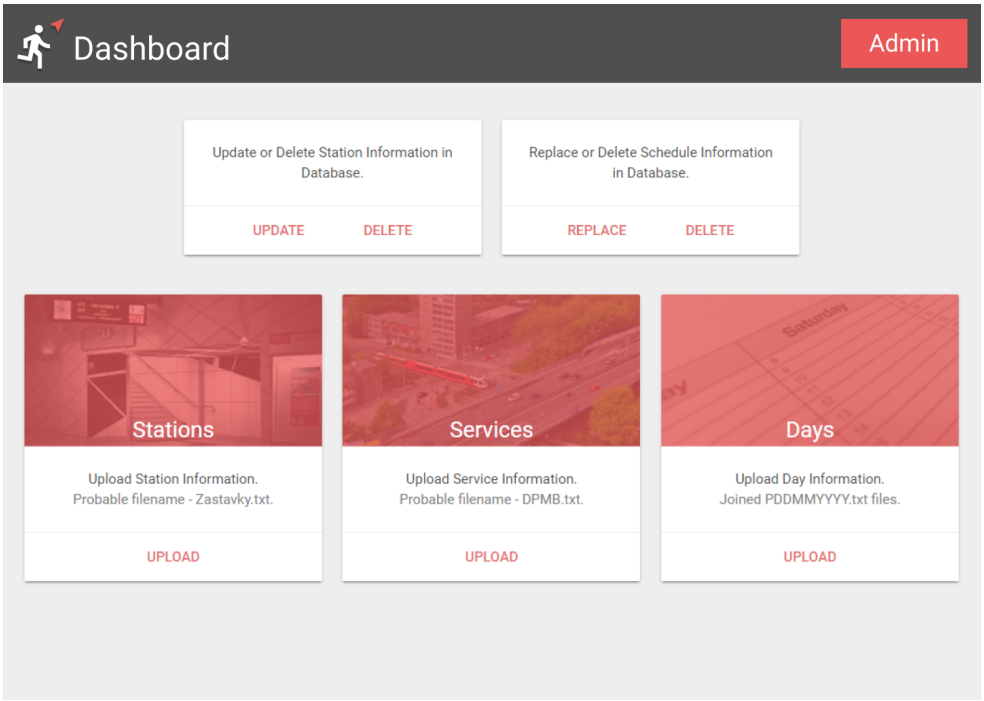
6.2.2 Administrátorská sekcia

Ako už bolo spomenuté pri architektúre, server musí vedieť spracovávať statické dáta pridávané administrátorom. K tomu bude slúžiť grafické užívateľské rozhranie dostupné cez jeden endpoint, ktorý poskytne API. Práca s príkazovým riadkom by bola v podstate rýchlejšia, avšak administrátor by si potreboval pamätať všetky príkazy. Navrhnuté grafické užívateľské rozhranie by malo spĺňať nasledujúce základné princípy [29]:

- **Jasnosť** - užívateľ by mal mať prehľad o tom, čo sa stalo, kde sa nachádza, čo môže spraviť a čo sa stane, keď to spraví.
- **Flexibilita** - navrhnúť niečo tak, aby to vyzeralo dobre vo všetkých situáciách.
- **Oboznámenosť** - užívateľovi sa uľaví, keď v novej neznámej aplikácii spozná známe položky a vzory.
- **Efektívnosť** - uistiť sa, že užívateľ dokončí svoju úlohu čo najefektívnejším spôsobom a nikdy stratí výsledok svojej práce.
- **Konzistencia a štruktúra** - aplikácia by nemala mať nekonzistentnú farebnú schému alebo náhodné rozhádzané grafické prvky.

Na základe týchto princípov bolo navrhnuté grafické užívateľské rozhranie predstavujúce administrátorskú sekciu, ktoré je zobrazené na obrázku 6.2. Táto sekcia však nemôže byť voľne dostupná, a teda je potrebné sa predtým prihlásiť pomocou jednoduchého dialógu s menom užívateľa a heslom.

[1] Uniform Resource Identifier



Obr. 6.2: Administrátorská sekcia.

6.3 Mobilná aplikácia

Hlavným cieľom práce je vytvoriť mobilnú aplikáciu s operačným systémom Android, ktorá príjme informácie zo servera a umožní tak užívateľovi použiť telefón ako navigáciu v mestskej hromadnej doprave v rámci mesta Brno. Aplikácia by mala mať jednoduché a intuitívne ovládanie, čo je dosiahnuté použitím Material[2] dizajnu, ktorý vytvorila spoločnosť Google.

V rámci implementácie budú použité moderné technológie ako RxJava, Retrofit a Dagger, tak aj jazyk Kotlin, ktorý spoločnosť Google začína presadzovať ako oficiálny programovací jazyk pre vývoj na platforme Android. RxJava je knižnica pre asynchrónne programovanie s využitím pozorovateľných prúdov dát. Dagger zabezpečí vloženie závislostí a Retrofit sa postará o komunikáciu so serverom. Tieto technológie budú bližšie predstavené v kapitole 7.

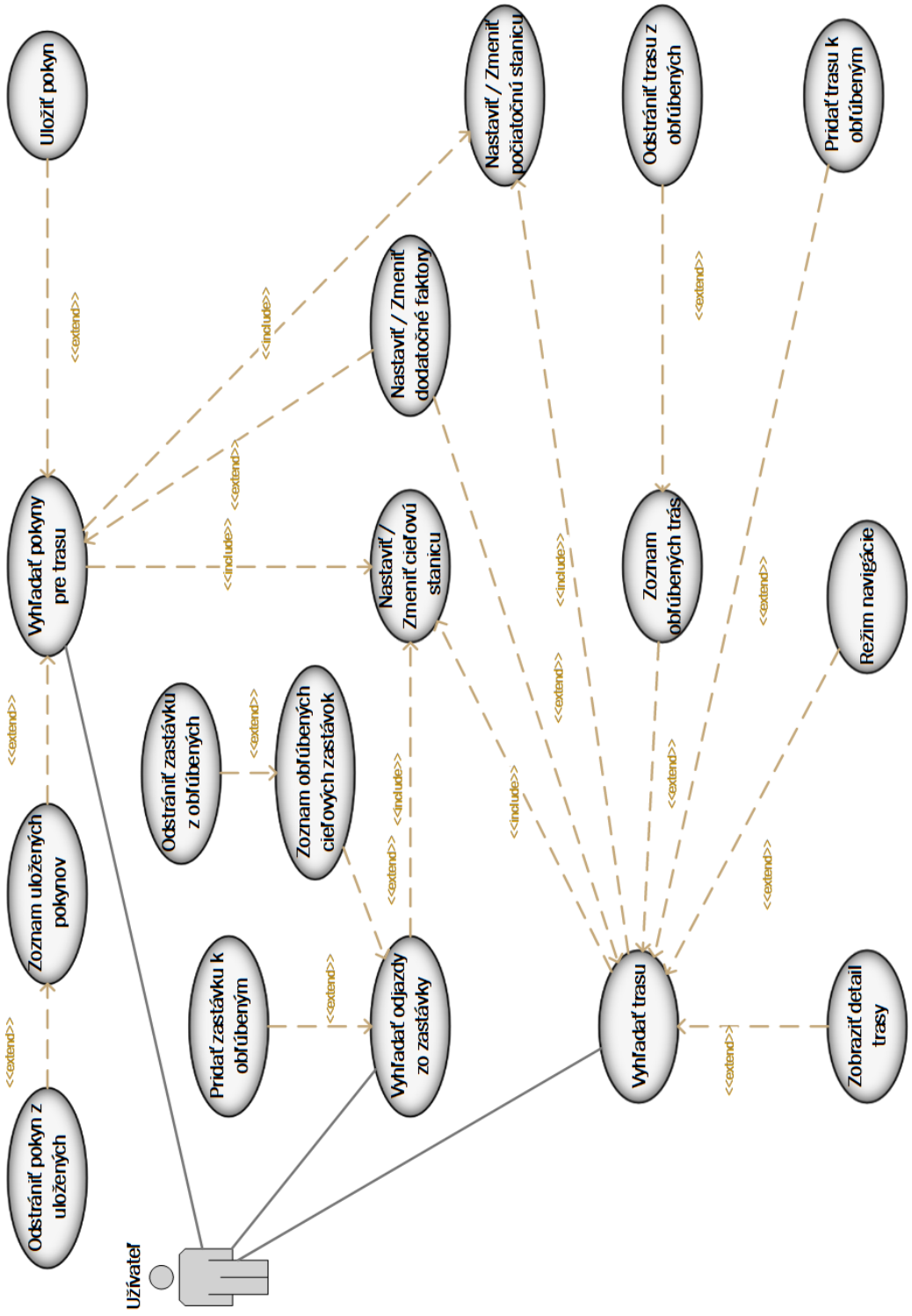
Aplikácia bude využívať návrhový vzor MVVM[3] s databindingom, ktorého výhodou je lepšie testovanie a jednoduchšia práca s rotáciou obrazovky. Tento prístup odporúča samotný Google.

6.3.1 Diagram prípadov použitia

Ako vypláva z kapitoly 3, mobilná aplikácia má za úlohu vyhľadať najrýchlejšiu trasu z jedného miesta do druhého, pričom sa po každom odchode zo zastávky toto hľadanie zopakuje. Okrem vyhľadania trás tiež ponúka možnosť zobrazenia všetkých odchodov z danej zastávky alebo zobrazenie pokynov pre zvolenú trasu. Pre lepšie pochopenie interakcie

[2] http://www.material.io
[3] Model View ViewModel

užívateľa s aplikáciou bol vytvorený diagram prípadov použitia v jazyku UML[4], ktorý je zobrazený na obrázku 6.3.



Obr. 6.3: Diagram prípadov použitia.

[4] Unified Modeling Language

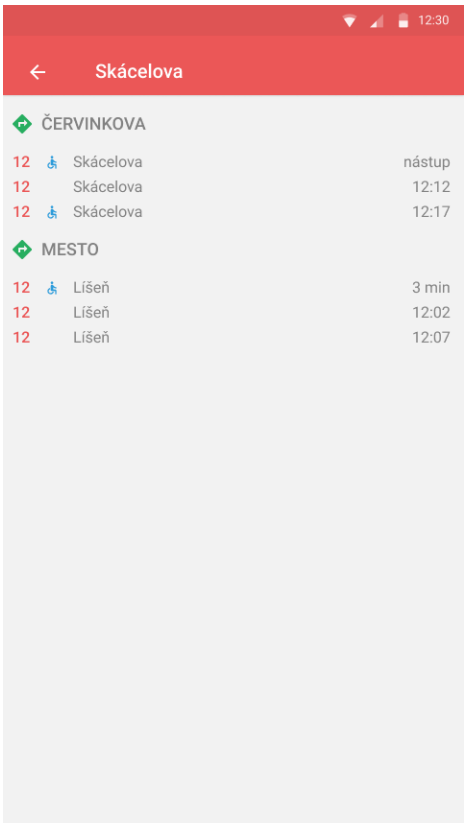
6.3.2 Užívateľské rozhranie

Návrh užívateľského rozhrania bol vypracovaný pomocou programu Figma[5], ktorý slúži ako pre dizajn a prototypovanie, tak aj pre získavanie spätnej väzby v rámci tímu.

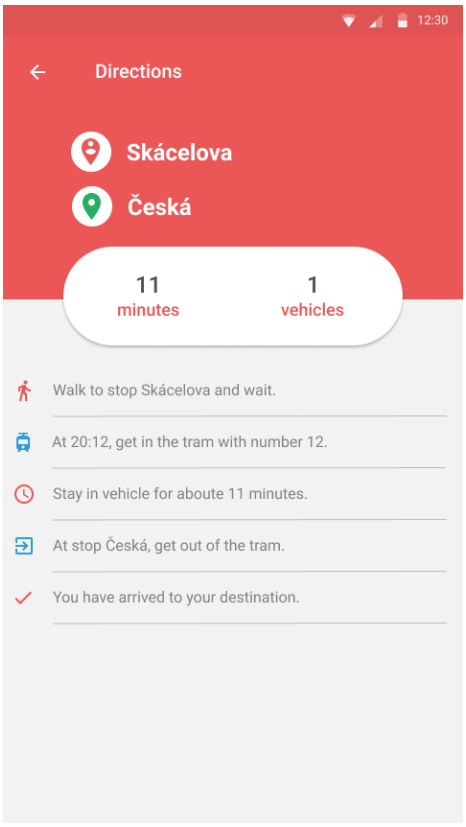
Po spustení aplikácie sa užívateľovi zobrazí hlavná obrazovka uvedená na obrázku 6.6, ktorá obsahuje tri možnosti, a to vyhľadať určitú trasu, zobraziť všetky spoje odchádzajúce z danej zastávky a zobraziť pokyny pre určitú trasu.

Sekcia pre vyhľadanie všetkých spojov zo zastávky obsahuje formulár pre vyhľadanie týchto spojov a zoznam obľúbených zastávok pre zjednodušenie práce užívateľovi. Po vyhľadaní spoja sa zobrazí obrazovka obsahujúca jednotlivé odjazdy. Táto obrazovka je uvedená na obrázku 6.4.

Sekcia pre zobrazenie pokynov, obsahuje formulár pre vyhľadanie trasy. Okrem základných možností zadania počiatočnej a cieľovej zastávky, má užívateľ k dispozícii aj pokročilé možnosti, ktorými sú definovanie dátumu a času. Keďže táto sekcia predstavuje časť aplikácie dostupnej bez nutnosti internetového pripojenia, tak sa pod formulárom nachádza zoznam uložených trás. Po kliknutí na uloženú trasu sa zobrazí obrazovka s pokynmi uvedená na obrázku 6.5.



Obr. 6.4: Obrazovka s odjazdami.

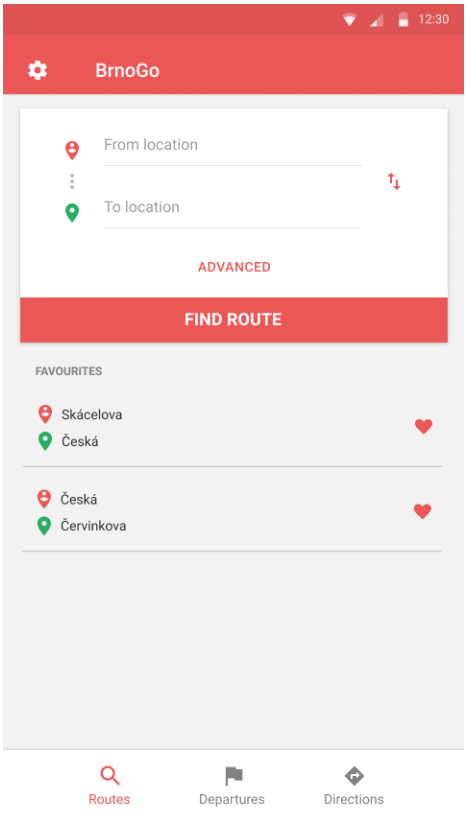


Obr. 6.5: Obrazovka s pokynmi.

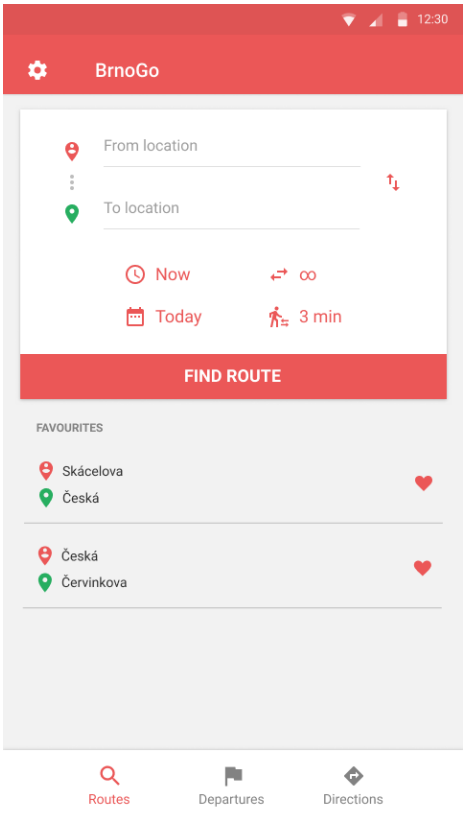
Hlavnou sekciou aplikácie je sekcia pre vyhľadanie určitej trasy uvedenej na obrázkoch 6.6 a 6.7. Táto možnosť obsahuje formulár pre vyhľadanie trasy a zoznam obľúbených trás. Rovnako ako v sekcii pre zobrazenie pokynov, aj v tejto sekcii môže užívateľ bližšie špecifikovať svoje požiadavky. Navyše má k dispozícii možnosti pre definovanie maximálneho

[5] http://www.figma.com

počtu prestupov a minimálneho času potrebného k týmto prestupom. Pokiaľ má užívateľ danú trasu uloženú ako obľúbenú, potom je možné túto trasu vybrať, čo má za následok automatické doplnenie počiatočného a cieľového miesta.



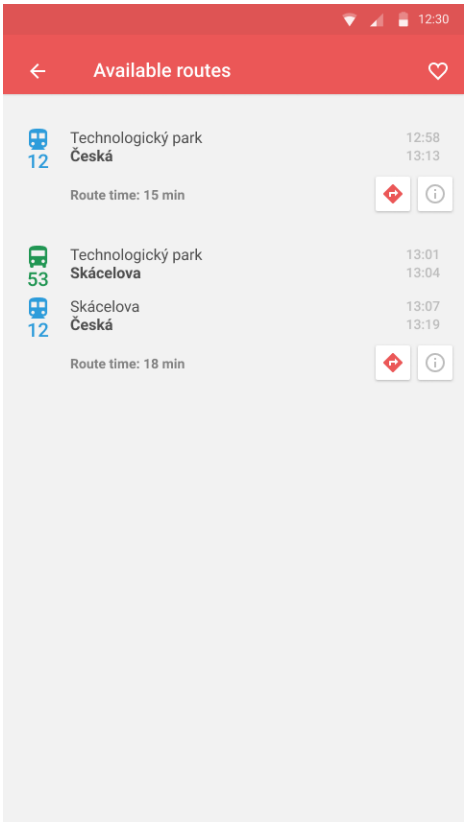
Obr. 6.6: Možnosť vyhľadania určitej trasy.



Obr. 6.7: Pokročilé nastavenia pre vyhľadanie určitej trasy.

Po kliknutí na tlačidlo pre vyhľadanie trasy sa zobrazí nová obrazovka so všetkými trasami, ktoré spĺňajú dané požiadavky. Toto okno je ilustrované na obrázku 6.8. Každá z trás obsahuje základné informácie ako číslo a typ vozidla, počet prestupov a čas odjazdu z počiatočnej zastávky spolu s časom príjazdu do destinácie. V tejto sekcií si užívateľ môže pridať trasu medzi obľúbené, a následne sa môže, prostredníctvom tlačítka pre spustenie navigácie, dostať do režimu navigácie, ktorý je zobrazený na obrázkoch 6.10 a 6.11.

Pokiaľ užívateľ klikne na tlačidlo s informáciami, tak sa mu zobrazí obrazovka s detailnejšími informáciami o danom spoji, ktoré navyše obsahujú všetky zastávky, ktorými daný spoj prechádza, všetky vozidlá, ktoré boli použite v rámci trasy, a čas príjazdu a odchodu z jednotlivých zastávok. Táto obrazovka je zobrazená na obrázku 6.9.



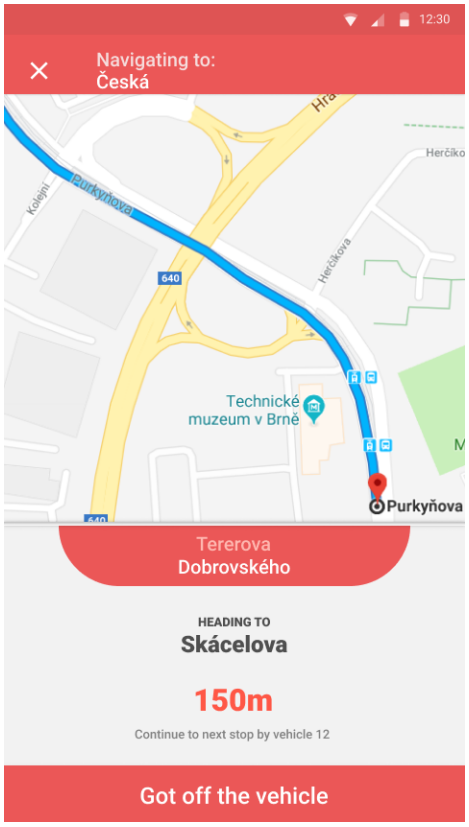
Obr. 6.8: Nájdené spoje.

Obr. 6.9: Detail spoja.

Poslednou a najdôležitejšou obrazovkou je režim navigácie. Táto obrazovka obsahuje mapu, na ktorej môže užívateľ sledovať svoju aktuálnu polohu. Túto mapu užívateľ využije najmä pri prestupe medzi zastávkami.

Pod mapou sa nachádzajú informácie obsahujúce zoznam nasledujúcich zastávok. Tento zoznam obsahuje maximálne dve zastávky.

Pod týmto zoznamom sa nachádza zóna obsahujúca aktuálnu zastávku a informácie s ďalšími pokynmi. Táto zóna zobrazuje rôzne informácie na základe aktuálneho stavu užívateľa. Pokiaľ užívateľ čaká na zastávke, tak sa mu zobrazí informácia ukázaná na obrázku 6.11, ktorá ho informuje o čísle vozidla a čase jeho odjazdu. Pokiaľ nastal čas nastúpenia tak sa zobrazí tlačidlo, ktorým užívateľ potvrdí aplikácii svoj nástup do vozidla. Naopak, pokiaľ sa užívateľ nachádza vo vozidle, tak sa mu zobrazí informácia, ktorá mu oznámi, koľko metrov ešte ostáva do ďalšej zastávky vrátane ďalších pokynov. Táto informácia je zobrazená na obrázku 6.10. Pod pokynmi sa navyše nachádza tlačidlo, ktorým užívateľ potvrdí aplikácii vystúpenie z vozidla.



Obr. 6.10: Užívateľ je vo vozidle.

Obr. 6.11: Užívateľ čaká na spoj.

Kapitola 7

Technológie

Táto kapitola obsahuje informácie o implementačných technológiách použitých v rámci tejto diplomovej práce. Jedná sa najmä o technológie, ktoré boli použité pre vytvorenie servera alebo mobilnej aplikácie, a nie sú až tak známe. Zároveň je tiež popísané ich využitie v rámci tejto diplomovej práce.

7.1 Univerzálne

V tejto sekcii je predstavený jazyk Kotlin, ktorý je spolu s jazykom Java použitý ako v serveri, tak aj v mobilnej aplikácii.

7.1.1 Kotlin

Kotlin je nový programovací jazyk vyvinutý spoločnosťou JetBrains zameraný na Java platformu. Je sústreďovaný na interoperabilitu s kódom Java, pričom je voľne dostupný vrátane kompilátora, knižníc a všetkých súvisiacich nástrojov. Tiež je ho možné používať takmer všade, kde sa používa technológia Java, či už pre vývoj na strane serverov, alebo na vytvorenie aplikácií pre systém Android. Kotlin funguje so všetkými existujúcimi Java knižnicami a frameworkami, pričom beží s rovnakou úrovňou výkonu ako Java.

Primárnym cieľom Kotlinu je poskytnúť stručnejšiu, produktívnejšiu a bezpečnejšiu alternatívu k Jave [16]. Kotlin môže pomôcť vývojárom dosiahnuť svoje ciele s menším počtom riadkov v kóde a menej nepríjemnosťami. Kotlin rieši viacero problémov, ktorými Java trpí [17]:

- Nulové referencie sú riadené typovým systémom.
- Žiadne surové typy.
- Polia v Kotline sú invariantné.
- Kotlin má správne typy funkcií, na rozdiel od SAM[1] konverzií v Jave.
- Kotlin nezahŕňa kontrolované výnimky.

Medzi hlavné výhody Kotlinu tiež patrí automatické generovanie často používaných metód. Túto výhodu je možné vidieť v zápise 9.

[1] Single Abstract Method

```java
public class Person {
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getNameLength() {
        return name.length();
    }

    // toString ...
    // hashCode ...
    // equals ...
    // copy ...
}
```

```kotlin
data class Person(var name: String) {

    fun getNameLength(): Int {
        return name.length
    }
}
```

Zápis 9: Porovnanie kódov Javy a Kotlinu.

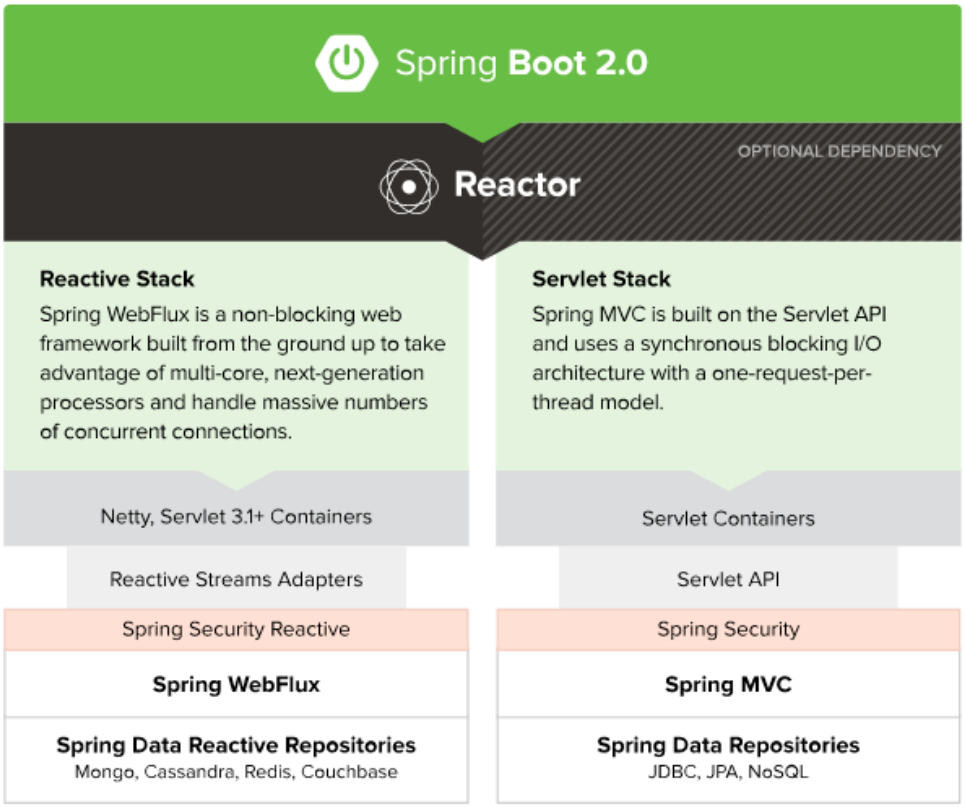
7.2 Server

V tejto sekcii sú popísané najdôležitejšie technológie, ktoré sú nevyhnutné pre správne fungovanie serveru.

7.2.1 Spring boot 2

Spring framework je tu už viac ako desať rokov, a za ten čas sa de facto stal štandardným frameworkom pre vývoj Java aplikácií. Avšak jeho konfigurácia nie je vždy jednoduchá. Túto nevýhodu rieši framework Spring Boot, ktorý vznikol pre čo najrýchlejšie nasadenie aplikácií do prevádzky. Vďaka nemu je možné vyvíjať Spring aplikácie s väčšou obratnosťou a sústrediť na riešenie funkčných potrieb aplikácie s minimálnym myslením na konfiguráciu samotného Springu [35].

V rámci tejto diplomovej práce sa používa Spring Boot 2, pre ktorý je minimálnou požiadavkou použitie Javy 8 a vyššej. Spring Boot 2 prináša súbor nových štartovacích prvkov pre rôzne reaktívne moduly. Medzi ne patrí WebFlux a reaktívne náprotivky pre databázy MongoDB, Cassandra alebo Redis. Hlavou výhodou použitia WebFlux frameworku je vysporiadanie sa s veľkými počtom súčasných pripojení [31]. Štruktúru Spring Boot 2 frameworku je možné vidieť na obrázku 7.1.



Obr. 7.1: Štruktúra Spring Boot 2 frameworku [31].

Táto obľúbená knižnica sa tiež snaží pomôcť v rôznych oblastiach:

- Správa závislosti.
- Autokonfigurácia.
- Predpripravená funkcionalita (Actuator, vylepšené prihlasovanie, monitorovanie, metriky, ...).
- Zlepšená skúsenosť s vývojom (viacero testovacích utilít, ...).

7.2.2 MongoDB

MongoDB je dokumentovo orientovaná open-source databáza patriaca do skupiny nerelačných (NoSQL) databáz. Obsahuje veľké množstvo užitočných funkcií pre podporu agregácie a geopriestorových indexov, či vyhľadávanie textu. Medzi hlavné prednosti MongoDB patrí [23]:

- **Vysoký výkon** - Indexy podporujú rýchlejšie dotazovanie a môžu tiež obsahovať kľúče z vložených dokumentov a polí. Podpora pre vstavané dátové modely znižuje vstupno-výstupnú aktivitu v databázovom systéme.
- **Vysoká dostupnosť** - Podpora skupiny serverov MongoDB nazývaných ako replica set, ktoré uchovávajú rovnaký súbor dát, čím poskytujú redundanciu a zvýšenie dostupnosti týchto údajov v prípade zlyhania.
- **Horizontálne škálovanie** - Podpora metódy nazývanej sharding[2], ktorá distribuuje dáta naprieč viacerými zariadeniami. Od verzie 3.4 je podporované aj vytváranie zón

[2] https://docs.mongodb.com/manual/sharding/#sharding-introduction

na základe shard kľúča, ktorý sa používa na distribúciu dokumentov medzi členmi zdieľaného klastra.
- **Podpora pre viaceré úložné enginy** - Podpora enginov WiredTiger, In-Memory a MMAPv1. Poskytnutie API k umožneniu vytvárania vlastných úložných enginov pre MongoDB.

Hlavným dôvodom pre odklon od relačného modelu je zjednodušenie škálovania [10]. Základnou myšlienkou je nahradiť koncept riadku flexibilnejším modelom, v tomto prípade dokumentom vo formáte BSON[3], ktorý je podobný formátu JSON[4]. Maximálna veľkosť tohto dokumentu je 16MB. Dokumentovo orientovaný prístup umožňuje reprezentovať komplexné hierarchické vzťahy jedným záznamom.

MongoDB je bez schém, čo znamená, že kľúče dokumentov nie sú vopred definované, ani fixne dané. Namiesto vynútenia rovnakého tvaru pre všetky dáta, môže byť problém nových a chýbajúcich kľúčov riešený na úrovni aplikácie.

```
{
  name: 'John Doe',
  age: '30',
  employers : [
     { name: 'Apple', year: '2004' },
     { name: 'Microsoft', year: '2010' }
  ]
}
```

Zápis 10: Príklad dokumentu uloženého v MongoDB.

V rámci tejto práce je táto databáza použitá pre ukladanie zastávok a dát, ktoré vznikli spojením a transformáciou súborov so službami, a dňami, kedy sú tieto služby dostupné. Tieto dáta obsahujú napríklad informácie o nasledujúcich zastávkach, čísla spojov alebo jednotlivé odjazdy.

7.3 Mobilná aplikácia

V tejto sekcii sú bližšie popísané najdôležitejšie technológie, ktoré sú použité pre správne fungovanie mobilnej aplikácie.

7.3.1 Dagger 2

Dagger je plne statický framework pre vkladanie závislostí. Je ho možné použiť ako v Jave, tak aj v Androide. Framework bol pôvodne vytvorený firmou Square a o jeho udržiavanie sa momentálne stará firma Google. Dagger sa zameriava na riešenie mnohých problémov týkajúcich sa vývoja a výkonu, ktoré trápili riešenia založené na reflexii [12].

Pre správne fungovanie je potrebne definovať rôzne moduly a komponenty [8]. Triedy s anotáciou `@Module` sú zodpovedné za poskytnutie objektov, ktoré môžu byť vložené. Tieto triedy obsahujú metódy s anotáciou `@Provides`, ktorá zabezpečí, že vrátený objekt bude k

[3] Binary JSON - http://bsonspec.org
[4] JavaScript Object Notation - https://www.json.org

dispozícií pre vloženie závislosti. Veľmi dôležitá je anotácia `@Inject`, pomocou ktorej definujeme závislosť. Rozhrania s anotáciou `@Component` definujú spojenie medzi jednotlivými modulmi a objektami, ktoré vyjadrujú závislosť. Dagger si triedu pre toto spojenie sám vygeneruje.

Príklad triedy využívajúcej Dagger je možné vidieť v zápise 11. Túto triedu je možné následne vložiť pomocou vkladania závislostí prostredníctvom jedného riadku, ukázaného pomocou zápisu 12.

```
@Singleton
public class DataManager {

    private Context context;
    private DatabaseHelper databaseHelper;

    @Inject
    public DataManager(@ApplicationContext Context context,
                       DatabaseHelper databaseHelper) {
        this.context = context;
        this.databaseHelper = databaseHelper;
    }
    ...
}
```

Zápis 11: Príklad triedy využívajúcej Dagger.

```
@Inject
DataManager dataManager;
```

Zápis 12: Príklad vloženia triedy prostredníctvom Daggeru.

7.3.2 Retrofit 2

Retrofit je typovo bezpečný HTTP klient pre Android a Javu. Bol vytvorený firmou Square a umožňuje relatívne jednoducho načítať, či nahrávať JSON[5] (alebo iné štruktúrované dáta) cez webovú službu založenú na REST[6] architektúre. Využíva knižnicu OkHttp pre HTTP žiadosti.

V Retrofite je možné nakonfigurovať, ktorý konvertor bude použitý na serializáciu dát. Pre konverziu z a do formátu JSON je možné použiť Gson, Jackson alebo Moshi. Pre konverziu Protocol Bufferu je možné použiť Protobuf alebo Wire. Ak by bolo potrebné konvertovať XML, tak je možné použiť Simple XML. Retrofit však umožňuje pridávanie aj vlastných konvertorov.

Pre správnu funkčnosť tejto knižnice sú potrebné definovať [33]:

[5] JavaScript Object Notation
[6] Representational state transfer

1. Modelové triedy, ktoré sa použijú na mapovanie JSON údajov.
2. Rozhrania, ktoré definujú možné operácie protokolu HTTP. Každá metóda rozhrania predstavuje jedno možné volanie rozhrania API. Možné definície metód sú ukázané na zápise 13.
3. Inštancia triedy Retrofit.Builder používajúca rozhranie a Builder API, ktoré umožňuje definovanie koncového URL bodu pre HTTP operácie.

```
1  @GET("users")
2  Call<List<User>> getUsers()
3
4  @GET("users/{name}/commits")
5  Call<List<Commit>> getCommitsByName(@Path("name") String name)
6
7  @GET("users")
8  Call<User> getUserById(@Query("id") Integer id)
9
10 @POST("users")
11 Call<User> postUser(@Body User user)
```

Zápis 13: Príklad definícií metód rozhrania [33].

V rámci tejto diplomovej práce je táto knižnica použitá pre akúkoľvek komunikáciu s vytvoreným serverom.

7.3.3 RxJava 2

RxJava, port ReactiveX pre Javu, bola vytvorená vo veľkej miere Benom Christensenom z Netflixu a Davidom Karnokom. Verzia 1.0 bola vydaná v novembri 2014 a verzia 2.0 o dva roky neskôr. RxJava sa snaží riešiť mnoho problémov, s ktorými sa programátor denne stretáva. Medzi tieto problémy patrí zaobchádzanie s udalosťami, zápasenie so zastaranými dátovými stavmi, či obnova po výnimke [25].

Reaktívne programovanie umožňuje rýchlo analyzovať a pracovať s live dátami, ako sú napríklad ceny akcií. Je tiež možné zrušiť a presmerovať prácu, škálovať so súbežnosťou, či vysporiadať sa s rýchlo emitujúcimi dátami. Jednotlivé prúdy dát a udalosti je možné zmiešať, zlúčiť, filtrovať, rozdeliť, či transformovať pomocou veľkého počtu metód. RxJava je asynchronická, ale obsahuje aj operátory pre konverziu asynchronických prúdov dát na synchronické. Tieto operátory sa však využívajú hlavne pre testovanie.

Základným typom je `Observable`, ktorý posúva položky (tzv. emisie) určitého typu cez sériu operátorov, až kým nedosiahne konečného pozorovateľa, ktorý tieto položky spotrebuje. Pozorovateľ však musí byť prihlásený na odber týchto položiek. Príklad takéhoto prístupu je možné vidieť na zápise 14. Na najvyššej úrovni obsahuje `Observable` tri typy udalostí:

- **onNext()** - posúva položku jednu po druhej smerom od zdroja k pozorovateľovi.
- **onComplete()** - oznámenie dokončenia udalosti, naznačuje, že nenastane už žiadne ďalšie volanie onNext().

- **onError()** - oznamuje chybu pozorovateľovi, kde pozorovateľ zvyčajne definuje, ako sa s chybou vysporiadať.

Nie vždy však potrebujeme plnú silu typu `Observable`. Preto má RxJava k dispozícií špecializované odnože tohto typu:

- **Single** - emituje len jednu pokožku. Udalosti onNext() a onComplete() sú nahradené udalosťou onSuccess(), ktorá vráti práve jednu emisiu.
- **Maybe** - podobné Single, ale vracia práve jednu alebo žiadnu emisiu.
- **Completable** - zaujíma ho iba vykonanie akcie. Obsahuje preto len udalosti onError() a onComplete().

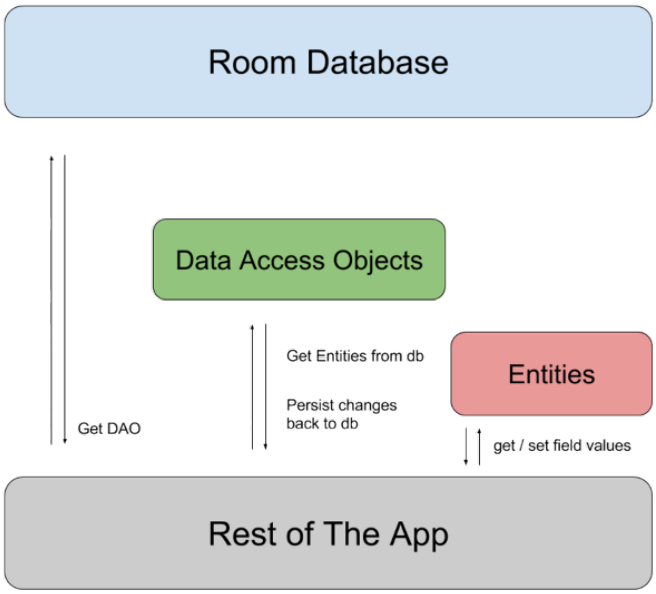
```
Observable.just("Alpha", "Beta", "Gamma", "Delta","Epsilon")
        .toMap(s -> s.charAt(0), String::length)
        .subscribe(s -> System.out.println("Received: " + s));
}
// Received: {A=5, B=4, D=5, E=7, G=5}
```

Zápis 14: Príklad reaktívneho prístupu [25].

Táto knižnica je použitá pre akúkoľvek asynchronnú komunikáciu v rámci vytvorenej aplikácie. Pomocou tejto knižnice bude aplikácia napríklad získavať aktuálnu polohu a oneskorenie vozidla za určitú periódu, získavať novú alebo rýchlejšiu trasu, ukladať obľúbené položky do databázy, či získavať dáta zo servera prostredníctvom knižnice Retrofit.

7.3.4 Room

Room je knižnica vytvorená firmou Google, ktorá poskytuje abstraktnú vrstvu nad SQLite. Knižnica umožňuje plynulý prístup k databáze a využiť plný výkon SQLite.



Obr. 7.2: Architektúra knižnice Room [3].

Z obrázku 7.2 vyplýva, že knižnica je zložená z troch hlavných komponent:

- **Entita** - predstavuje tabuľku v rámci databázy. Trieda je definovaná pomocou anotácie `@Entity`.
- **DAO**[7] - objekt definovaný anotáciou `@Dao`. Obsahuje metódy pre prístup do databázy. Tieto metódy sú definované pomocou anotácií medzi ktoré patria `@Query`, `@Insert`, či `@Delete`.
- **Databáza** - slúži ako hlavný prístupový bod pre základné pripojenie k uloženým relačným dátam. Používa anotáciu `@Database` na definovanie zoznamu entít a verzie databázy. Obsah tejto triedy definuje DAO metódy.

Pre platformu Android existujú známejšie a rýchlejšie databázy ako Realm[8] alebo ObjectBox[9] [36]. Avšak v rámci tejto diplomovej práce je databáza použitá len pre ukladanie pokynov alebo obľúbených zastávok. Z tohto dôvodu nie je potrebné najrýchlejšej databázy. Keďže Room len poskytuje abstraktnú vrstvu nad SQLite, zaberá len 50KB. Na účely tejto práce Room absolútne postačuje.

[7] Data Access Object
[8] https://www.realm.io
[9] http://objectbox.io

Kapitola 8

Implementácia

V tejto kapitole je popísaná ako implementácia serverovej časti, tak aj implementácia mobilnej aplikácie. Implementácia vychádza z návrhu aplikácie popísanej v kapitole 6 a využíva technológie popísané v kapitole 7.

8.1 Server

Zo zadania tejto práce vyplýva, že okrem vytvorenia mobilnej aplikácie je jedným z hlavných cieľov práce vytvoriť server, ktorý poskytne aplikácií potrebné dáta.

V rámci vyhľadávania najrýchlejšej trasy sa však serverová aplikácia nezaobíde bez komunikácie s webovou službou poskytovanou spoločnosťou Kordis JMK. Komunikáciu s touto službou zabezpečuje modul `live`. Avšak služba je dostupná len prostredníctvom jediného zariadenia v sieti VUT, ktoré je dostupné pomocou adresy `pcuifs2.fit.vutbr.cz`. Pre účely tejto diplomovej práce bol preto vytvorený účet na tomto zariadení dostupný cez `ssh` protokol, prostredníctvom ktorého bolo možné tunelovať potrebné požiadavky zo serveru až na webovú službu firmy Kordis JMK.

Ako bolo už spomenuté v kapitole 5.4, webová služba je typu SOAP s WSDL popisom, na základe ktorého je následne možné túto službu automaticky vygenerovať napríklad použitím vývojového prostredia IntelliJ Idea od firmy JetBrains. K vygenerovaniu je však potrebné nastaviť proxy v rámci vývojového prostredia, v inom prípade by službu nebolo možné vygenerovať.

Keďže jednou z podmienok pre použitie webovej služby je maximálny počet dvadsiatich požiadavok za päť sekúnd, obsahuje modul `live` aj metódy zabezpečujúce cachovanie dát. Pre toto cachovanie je použitá knižnica Caffeine[1], ktorá sa vyznačuje vysokým výkonom. Doba, po ktorú sú dáta platné sa líši v závislosti od typu dát.

8.1.1 Naplánovanie trasy

Ako vyplýva z kapitoly 6, komunikáciu medzi klientom a serverom zabezpečuje modul `core`. Pre naplánovanie trasy je potrebné, aby klientská aplikácia požiadala server o dáta. Server následne spracuje jednotlivé parametre požiadavku a spustí algoritmus A*, ktorý je umiestnený v balíku `graph`. Tento algoritmus je napísaný v jazyku Kotlin, a je použitý ako pre vyhľadanie trás, tak aj pre vyhľadanie pokynov.

[1] https://github.com/ben-manes/caffeine

V prípade, že sa vyhľadávajú trasy, potrebuje algoritmus A* navyše rátať aj s oneskorením vozidiel. Pre toto vyhľadanie sú teda potrebné dáta z modulu `live`. Ak by mal algoritmus vyhľadať len pokyny, tak nie sú live dáta potrebné.

Server získa trasy prostredníctvom triedy `AStar`, ktorá obsahuje funkcie pre naplánovanie a získanie trasy. Pre vyhľadanie trás je použitá metóda `findMultipleRoutes()`, ktorá má na svojom vstupe počiatočnú stanicu, cieľovú stanicu a konfiguračné detaily obsahujúce údaje ako počiatočný čas, minimálny čas na prestup, či limit vyhľadávania. Na základe limitu vyhľadávania je prostredníctvom tejto metódy použitá metóda `findSingleRoute()`. Táto metóda predstavuje hlavnú metódu pre naplánovanie a vytvorenie trasy. Na vstupe navyše očakáva počiatočný čas.

Na začiatku je vytvorený štartový a cieľový uzol a následne sú tieto uzly porovnané. Ak sa uzly rovnajú je ako trasa vrátený `null`. V opačnom prípade je štartový uzol pridaný do prioritnej fronty, pričom je inicializovaná cena uzla a je mu tiež priradená prvá zastávka v stanici. Priorita fronty je daná celkovým ohodnotením uzla. Po pridaní uzla do fronty začne prebiehať expanzia daného uzla. Táto expanzia prebieha až kým sa fronta nevyprázdni, alebo až kým nie je aktuálny uzol zhodný s koncovým.

Na začiatku expanzie je potrebné vytvoriť uzly pre všetky ostatné zastávky v rámci aktuálnej stanice. Následne sa pre každý uzol vyhľadá odpovedajúci rozvrh. Z rozvrhu sa vyberú nasledujúce stanice dostupné z aktuálnej stanice. Pre každú stanicu je potom vyhľadaný najlepšie odpovedajúci záznam obsahujúci odjazd. Pre vyhľadanie záznamu je použitá jedna z metód v závislosti na potrebe live dát:

- `getCorrectLiveScheduleRow()` - metóda je použitá pri vyhľadávaní trás. Táto metóda berie do úvahy oneskorenie vozidiel, pričom berie do úvahy aj záznamy, ktoré by za normálnych okolností nevyhoveli, ale pre oneskorenie vozidla vyhovujú.
- `getCorrectScheduleRow()` - metóda je použitá pri získavaní pokynov. Táto metóda vyhľadá najbližší vyhovujúci záznam, pričom s oneskorením vozidiel sa nepočíta.

Po vyhľadaní záznamu sa pre zrýchlenie algoritmu porovná čas odchodu s počiatočným časom, a zahodia sa všetky záznamy ktoré sú novšie alebo staršie ako 12 hodín. Tiež sa vypočíta čas reprezentujúci čakaciu dobu. Po tomto výpočte sa cena uzla rovná súčtu aktuálnej ceny uzla a tohto času.

Ak je potrebný prestup, tak sa skontroluje, či čas potrebný pre tento prestup nie je menší ako čas pre prestup stanovený užívateľom. V kladnom prípade sa ešte skontroluje maximálny počet prestupov, a čas potrebný pre prestup sa pripočíta k cene uzla. Následne je zavolaná metóda `offerNode()`, ktorá skontroluje, či je cena uzla menšia ako cena uložená v `HashMap`[2]. Ak je cena menšia, tak je vytvorený nový uzol. Tomto uzlu je potom priradené celkové ohodnotenie pozostávajúcej z ceny, ku ktorej je pripočítaná cena do nasledujúcej stanice vypočítaná ako čas do nasledujúcej stanice, a časový odhad do cieľového uzla v sekundách. Tento odhad predstavuje heuristickú funkciu, ktorá je vypočítaná pomocou nasledujúceho vzorca:

$$h(x) = D \div S * 60^2 \tag{8.1}$$

, kde D je vzdialenosť v kilometroch od aktuálneho uzla do koncového vypočítaného pomocou formule Haversine [24], a S je priemerná rýchlosť vozidla v kilometroch za hodinu zobrazená v tabuľke 8.1. Následne je uzol pridaný do prioritnej fronty.

Opäť sa vyberie uzol z prioritnej fronty pričom sa porovná, či je súčasný uzol koncovým. Ak uzol nie je koncovým, tak začne opäť prebiehať expanzia daného uzla. V opačnom

[2] https://www.javatpoint.com/java-hashmap

prípade je zavolaná metóda `createRoute()` z triedy `RouteHelper`. Táto metóda vytvorí trasu z posledného uzla a zároveň správne nastaví jednotlivé časy príchodov a odchodov zo zastávok.

Električka	Autobus	Trolejbus
15.6	14.0	12.2

Tabuľka 8.1: Prehľad rýchlostí v kilometroch za hodinu [18].

8.2 Mobilná aplikácia

Najhlavnejším cieľom práce je vytvoriť mobilnú aplikáciu na platforme Android, ktorá bude užívateľa navigovať do destinácie v čo najkratší čas. Android je mobilný operačný systém vyvinutý spoločnosťou Google na základe upravenej verzie linuxového jadra a iného open-source softvéru. Je primárne určený pre mobilné zariadenia s dotykovou obrazovkou, ako sú napríklad smartphony a tablety.

Vytvorená aplikácia je dostupná pre všetky zariadenia využívajúce systém Android od verzie 5.0 (API 21) s kódovým označením Lollipop až po súčasne najvyššiu verziu 9.1 (API 27), ktorá nesie kódové označenie Oreo. Aplikácia tak podporuje viac ako 84% všetkých Android zariadení [2]. Aplikácia je štandardne v angličtine. V závislosti na použitom systémovom jazyku však aplikácia podporuje okrem angličtiny aj češtinu a slovenčinu.

Okrem technológií spomenutých v kapitole 7.3, bolo pre vytvorenie aplikácie nutné použiť vývojové prostredie Android Studio[3] a nástroj pre automatizáciu Gradle[4].

8.2.1 Režim navigácie

Ako bolo spomenuté v kapitole 6, tak po vyhľadaní trás a vybratí jednej z nich môže užívateľ spustiť režim navigácie. Tento režim je najdôležitejšou časťou aplikácie. Aby však tento režim fungoval bez problémov, je potrebné povolenie užívateľa pristupovať k aktuálnej polohe zariadenia. Bez udelenia tohto povolenia nebude možné určiť polohu a tým pádom sa navigácia stáva nefunkčnou.

Po spustení režimu s potrebnými povoleniami sa inicializuje mapa a zároveň sa aktivuje získavanie polohy zariadenia s periódou obnovy jednej sekundy, ktorú zabezpečuje interaktor vytvorený pomocou RxJavy. Následne sa na základe tejto polohy zobrazia pokyny pre užívateľa. Aplikácia zároveň spustí zisťovanie meškania súčasného vozidla. Keďže Kordis aktualizuje informácie o vozidlách každých pätnásť až tridsať sekúnd, tak bola perióda obnovy pre toto zisťovanie stanovená na každých desať sekúnd.

Pri každom získaní polohy je vypočítaná vzdialenosť do nasledujúcej zastávky pomocou formuly Haversine [24], ktorá berie do úvahy aj zakrivenie zeme. Na základe tejto vzdialenosti je následne určená aktuálna akcia, ktorá nadobúda jednu z nasledujúcich hodnôt:

- `TYPE_EXIT`
- `TYPE_WAIT_FOR_NEW_ROUTE`
- `TYPE_WAIT_FOR_VEHICLE_TO_LEAVE`
- `TYPE_BOARD`
- `TYPE_RIDE`

[3] https://developer.android.com/studio
[4] https://gradle.org

- `TYPE_WALK`
- `TYPE_WAIT`

Pre určenie hodnoty aplikácia zohľadňuje ako dosiahnutie zastávky užívateľom, tak aj, či sa užívateľ nachádza vo vozidle. Zastávka je považovaná za dosiahnutú v prípade, že je aktuálna vzdialenosť menšia ako 45 metrov. Následne môžu nastať tieto prípady:

1) **Užívateľ sa nachádza na zastávke a nie je vo vozidle** - aktuálna akcia je nastavená na `TYPE_WAIT`. Aplikácia zobrazí užívateľovi pokyny spolu s odpočítavajúcim sa časom pre odjazd vozidla zo zastávky. Zároveň sa tiež zobrazí tlačidlo, ktorým užívateľ potvrdí svoj nástup do vozidla. Aplikácia porovnáva súčet odpočítavajúceho sa času, oneskorenia vozidla a korekcie pre odjazd, s aktuálnym časom. Korekcia pre odjazd je stanovená na jednu minútu. Pokiaľ nastane čas nastúpenia do vozidla, ktorý odpovedá jednej minúte pred odjazdom zo zastávky, tak sa aktuálna akcia nastaví na `TYPE_BOARD`. Po nástupe sa okrem nastavenia aktuálnej akcie na `TYPE_WAIT_FOR_VEHICLE_TO_LEAVE`, zároveň nastaví aj premenná určujúca, či sa užívateľ nachádza vo vozidle na `true`. Navyše sa vygeneruje približná cesta do poslednej zastávky aktuálnym vozidlom prostredníctvom Directions API[5] a zobrazí sa tlačidlo potvrdzujúce aplikácii vystúpenie užívateľa z vozidla. V prípade, že užívateľ nenastúpil do vozidla a čas vyhradený pre nástup ubehol, tak sa aktuálna akcia nastaví na `TYPE_WAIT_FOR_NEW_ROUTE` a nájde sa nová trasa, ktorá automatický nahradí aktuálnu, pričom sa resetujú všetky uložené informácie k danej trase okrem stavu nastúpenia užívateľa do vozidla.

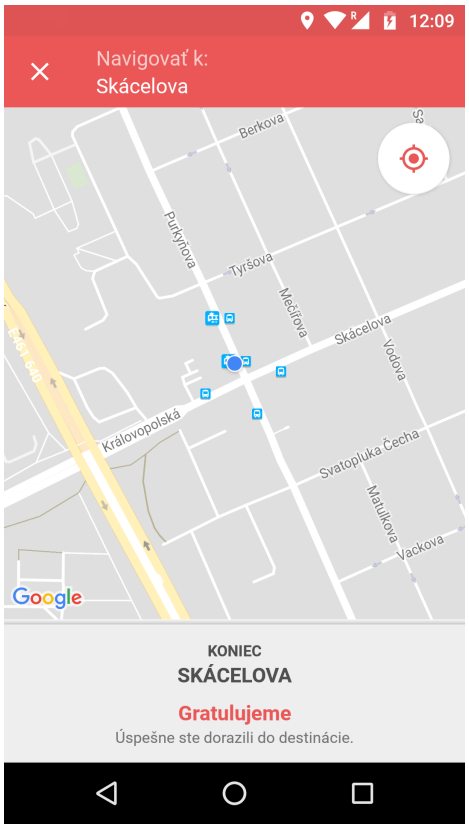
2) **Užívateľ sa nenachádza na zastávke a je vo vozidle** - tento prípad nastane ak je aktuálna vzdialenosť medzi súčasnou zastávkou a užívateľom väčšia ako 60 metrov. Po prekročení vzdialenosti nastaví aplikácia aktuálnu akciu na `TYPE_RIDE` a tiež nastaví súčasnú zastávku na ďalšiu v poradí. Užívateľovi sa zobrazia informácie ako pokyny, vzdialenosť do novej súčasnej zastávky a maximálne dve nasledujúce zastávky. Okrem toho aplikácia vyhľadá aj najrýchlejšiu trasu zo súčasnej zastávky a porovná konečný čas aktuálnej trasy s novo vyhľadanou. V prípade, že je konečný čas vyhľadanej trasy menší ako dve minúty, ponúkne aplikácia možnosť nahradiť trasu pomocou dialógu zobrazenom na obrázku 8.2. K tomuto vyhľadaniu je použitý interaktor využívajúci RxJavu. V prípade nahradenia trasy sa resetujú všetky uložené informácie k danej trase okrem stavu nastúpenia užívateľa do vozidla. V opačnom prípade sa pokračuje pomocou aktuálnej trasy.

3) **Užívateľ sa nachádza na zastávke a je vo vozidle** - v prípade, že táto zastávka nie je klasifikovaná ako prestupná alebo konečná, tak je aktuálna akcia nastavená na akciu `TYPE_WAIT_FOR_VEHICLE_TO_LEAVE`, čím dôjde k výzve užívateľa aby nevystupoval. Ak by vystúpil, tak by aplikácia vykonala rovnaké kroky ako v prvom prípade. V prípade, že je táto zastávka prestupnou alebo konečnou, tak je aktuálna akcia nastavená na `TYPE_EXIT`. Užívateľ svoje vystúpenie potvrdí stlačením tlačidla potvrdzujúceho vystúpenie, čím sa premenná určujúca, či sa užívateľ nachádza vo vozidle nastaví na `false`. Ak sa jedná o koncovú zastávku, tak aplikácia zobrazí užívateľovi gratulačný text zobrazený na obrázku 8.1. Ak sa jedná o nasledujúcu zastávku, tak sa zmení vozidlo na ďalšie v poradí, nastaví sa ďalšia zastávka a zobrazia sa rovnaké informácie ako vo štvrtom prípade. Ak užívateľ nevystúpi z vozidla, tak aplikácia po

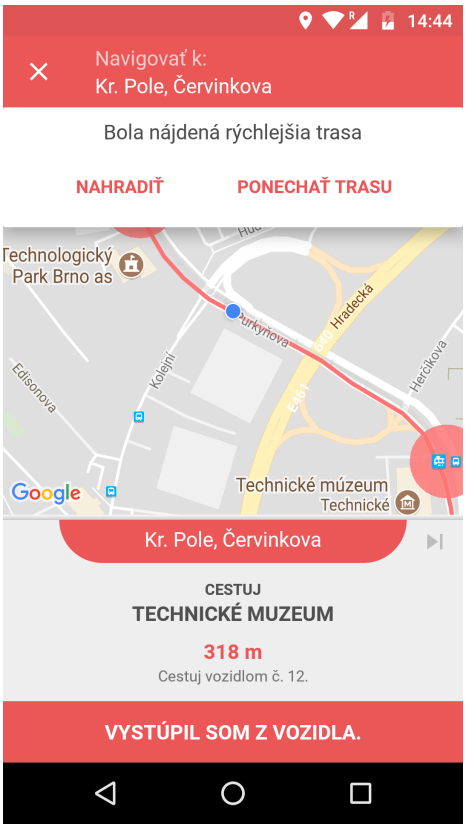
[5] https://developers.google.com/maps/documentation/directions/intro

presiahnutí hranice 60 metrov upozorní užívateľa o nerešpektovaní pokynov. Užívateľ je následne vyzvaný aby režim ukončil a vyhľadal novú trasu z ďalšej zastávky.

4) **Užívateľ sa nenachádza na zastávke a nie je vo vozidle** - aktuálna akcia je nastavená na `TYPE_WALK`. Aplikácia zobrazí užívateľovi pokyny spolu s informáciami ako názov zastávky a vzdialenosť k nej. Navyše sa mu na mape vygeneruje trasa k súčasnej zastávke opäť pomocou Directions API.



Obr. 8.1: Gratulačný text zobrazený aplikáciou.



Obr. 8.2: Dialóg pre nahradenie trasy.

8.2.2 Zobrazenie najbližších odchodov zo zastávky

Po vybratí zastávky aplikácia kontaktuje server prostredníctvom interaktora využívajúceho RxJavu. Tento interaktor následne dáta pretransformuje tak, že sa zo získaného objektu vyberú všetky vozidlá a vložia sa za novo vytvorený objekt reprezentujúci smer odjazdu. Takto transformované dáta sa poskytnú adaptéru, ktorý tieto dáta zobrazí.

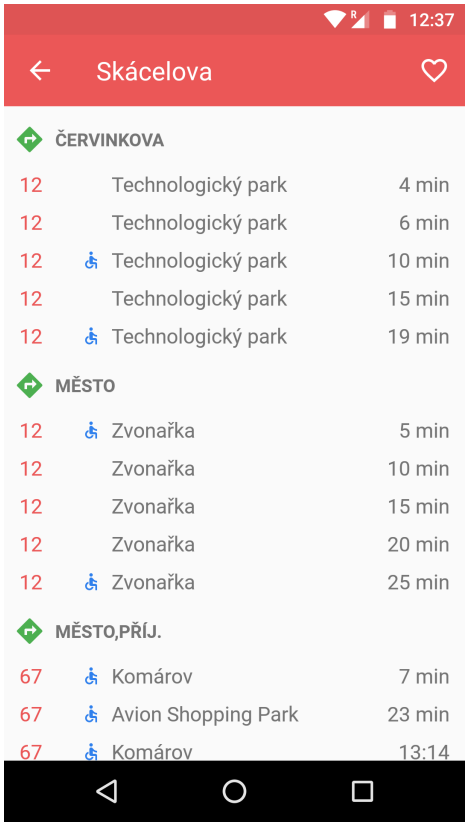
Okrem zobrazenia odchodov má užívateľ navyše možnosť uložiť si zastávku medzi obľúbené. Toto uloženie je vykonané pomocou ďalšieho interaktora, ktorý na základe toho, či je zastávka už uložená, vymaže alebo uloží zastávku z/do databázy. Finálnu podobu tejto obrazovky je možné vidieť na obrázku 8.3.

8.2.3 Režim pokynov

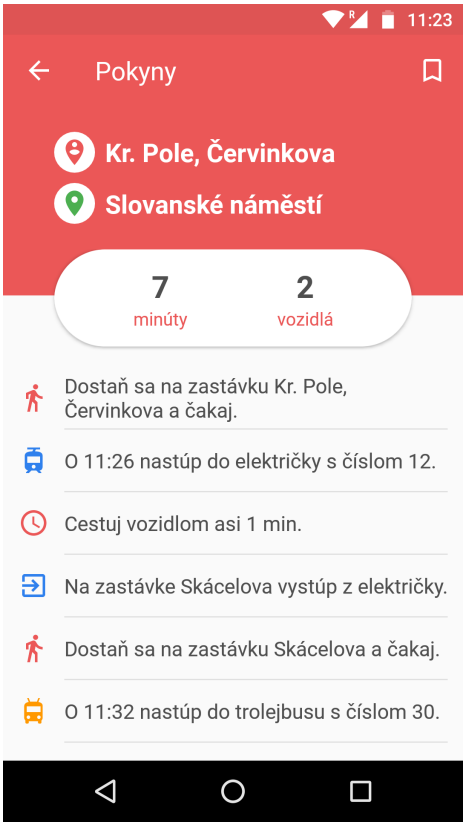
Ako bolo naznačené v kapitole 6, tak posledným dôležitým režimom je režim pokynov. Tento režim bol vytvorený nad rámec zadania a slúži najmä k navigácii bez pripojenia k internetu. V tomto režime sa však nezobrazuje mapa, ani sa aplikácia nesnaží užívateľa navigovať na základe súčasnej polohy. Namiesto toho aplikácia zobrazí pokyny, pomocou ktorých sa užívateľ dostane do destinácie.

Podobne ako pri zobrazení najbližších odchodov zo zastávky, tak aj v tomto prípade po vybratí zastávok aplikácia kontaktuje server prostredníctvom interaktora využívajúceho RxJavu. Tento interaktor následne dáta pretransformuje tak, že sa zo získaného objektu vyberú všetky vozidlá a pre každé z nich sa vytvorí sada pokynov. Tieto pokyny sú následne vložené za objekt reprezentujúci trasu. Takto transformované dáta sa poskytnú adaptéru, ktorý tieto dáta zobrazí.

Keďže má tento režim reprezentovať offline časť aplikácie, je možné uložiť si tieto pokyny. Toto uloženie je vykonané pomocou ďalšieho interaktora, ktorý na základe toho, či sú pokyny už uložené, vymaže alebo uloží pokyny z/do databázy. Finálnu podobu tejto obrazovky je možné vidieť na obrázku 8.4.



Obr. 8.3: Finálna podoba obrazovky s odjazdami.



Obr. 8.4: Finálna podoba obrazovky s pokynmi.

8.2.4 Možnosť prispôsobenia

Pre zlepšenie užívateľskej prívetivosti ponúka aplikácia obrazovku s nastaveniami. V rámci tejto obrazovky sú k dispozícii nasledujúce možnosti:

- **Štartovacia obrazovka** - užívateľ má v rámci hlavnej obrazovky možnosť vybrať si jednu z troch sekcií spomenutých v kapitole 6. Po spustení aplikácie sa zobrazí zvolená sekcia.
- **Limit pre vyhľadávanie** - užívateľ zvolí počet vyhľadaných trás v rozmedzí jedna až päť. Nastavenie slúži k zrýchleniu vyhľadania trasy alebo len k zmene počtu vyhľadaných trás.
- **Automatické nastúpenie do vozidla** - v rámci navigácie môže užívateľ danú možnosť aktivovať. Pri zapnutí možnosti bude aplikácia v čase odjazdu automaticky predpokladať, že sa užívateľ nachádza vo vozidle. V prípade, že nie je možné automatické nastúpenie vykonať, je užívateľ o tejto situácii informovaný prostredníctvom textu pod pokynmi.
- **Synchronizácia zastávok** - aplikácia ponúka možnosť synchronizácie zastávok pre prípad, že by užívateľ nadobudol pocit neaktuálnosti informácií o zastávkach.

Kapitola 9

Testovanie

Táto kapitola je zameraná na popis testovania oboch implementovaných častí. Serverová časť bola testovaná na počítači Asus N56JR, ktorý má nasledovné parametre:

- **CPU:** Intel Core i7-4700HQ 3.4 GHz.
- **RAM:** 8 GB.
- **OS:** Windows 10 Education.

Aby však bolo testovanie možné, bolo potrebné do tohto počítača pristupovať z internetu. Prístup do počítača bol zaistený pomocou nástroja Localtunnel[1], ktorý umožňuje zdieľať webovú službu bežiacu na lokálnom vývojovom stroji pomocou automaticky vygenerovanej adresy. Túto adresu je možné upraviť tak, aby bola vždy rovnaká.

V prípade serverovej časti boli vykonané najmä výkonnostné testy pomocou nástroja VisualVM[2]. Vďaka tomuto nástroju bolo odhalené, že aplikáciu najviac spomaľovalo získanie dát z databázy a získanie oneskorenia vozidla. Preto bolo potrebné implementovať cachovanie dát z databázy a premiestniť niektoré volania pre získanie oneskorenia vozidla. Zavedené optimalizácie výrazne zrýchlili vyhľadanie trasy.

Trasa	**Počet staníc**	**Počet vozidiel**	**Live dáta**	**Cache**	**Trvanie**
Technologický Park - Červinkova	3	1	Nie	Nie	0.719
			Nie	Áno	0.011
			Áno	Nie	0.794
			Áno	Áno	0.029
Semilasso - Technologický Park	7	2	Nie	Nie	3.673
			Nie	Áno	0.389
			Áno	Nie	4.641
			Áno	Áno	1.404
Semilasso - Česká	10	1	Nie	Nie	3.214
			Nie	Áno	0.207
			Áno	Nie	4.626
			Áno	Áno	1.649

Tabuľka 9.1: Prehľad rýchlosti vyhľadania jednej trasy.

[1] https://localtunnel.github.io/www
[2] http://visualvm.github.io

Konkrétne testovacie scenáre sú zobrazené v tabuľke 9.1, z ktorej vyplýva, že rýchlosť vyhľadania jednej trasy je ovplyvnená najmä celkovým počtom staníc, potrebou prestupu a potrebou live dát.

V prípade mobilnej aplikácie na platforme Android je dôležite testovať na viacerých zariadeniach najmä z dôvodu rôznych verzií operačného systému, či rôzneho rozlíšenia. Preto bola aplikácia testovaná vybranou skupinou ľudí s rôznymi zariadeniami, ktoré sú uvedené v tabuľke 9.2.

Zariadenie	**Verzia OS**	**Rozlíšenie**
Lenovo Moto G5	7.0	1920 × 1080
Samsung Galaxy A8	7.1	2220 × 1080
Samsung Galaxy A3	6.0	2220 × 1080
Sony Xperia Z5 Compact	7.1.1	1280 × 720

Tabuľka 9.2: Prehľad zariadení.

Testovanie bolo zamerané najmä na jednoduchosť užívateľského rozhrania, stabilitu aplikácie a jej celkovú funkčnosť v režime navigácie. Pre testovanie boli použité hlavne trasy uvedené v tabuľke 9.1. Počas testovania prešla aplikácia po každej zmene dvoma fázami:

1) **Simulačné prostredie** - použitie falošných súradníc pomocou aplikácie GPS emulator[3].
2) **Reálne prostredie** - začne až v prípade úspešného prejdenia aplikácie simulovaným prostredím.

V priebehu testovania boli odhalené štyri závažné problémy. Prvý problém bol odhalený už počas prvotnej fázy vývoja v simulovanom prostredí. Zvyšné tri problémy boli odhalené až v reálnom prostredí.

Prvým problémom bolo využitie polohy užívateľa a vozidla pre detekciu nástupu, či výskytu užívateľa vo vozidle. Avšak vzhľadom na problém s frekvenciou obnovy GPS súradníc vozidla a ich strácaním, bolo potrebné implementovať nové riešenie, ktoré si však vyžaduje interakciu s užívateľom.

Druhým problémom bola detekcia príchodu na zastávku. V prípade, že sa užívateľ nachádzal na úplnom konci vozidla, aplikácia to nebola schopná detegovať. Tento problém vznikal hlavne z dôvodu nepresnosti poskytnutej polohy zastávok. Bolo preto potrebné zvýšiť rozsah detekcie o desať metrov.

Tretí problém súvisel s oneskorením vozidla. V prípade, že užívateľ potreboval prestúpiť a vozidlo dohnalo svoje oneskorenie, tak sa užívateľovi nezobrazilo tlačidlo potvrdzujúce nastúpenie do vozidla. Tento problém vznikal z dôvodu obnovovacej frekvencie údajov vozidla, či ich straty. Riešenie spočívalo v podobe zobrazenia tlačidla už pri dosiahnutí zastávky.

O štvrtom probléme informovala už práca Sledování pohybu v MHD [27]. Týmto problémom je poloha zastávok. Súbor so zastávkami naďalej obsahuje zastávky, ktorých poloha neodpovedá ich skutočnej polohe. Navyše súbor so zastávkami nie je aktualizovaný ani v prípade výluky. Ako čiastočné riešenie bolo implementované tlačidlo pre preskočenie zastávky ktoré je možné vidieť na obrázku 8.2. Toto riešenie však nerieši problém v prípade, že je zastávka prvá alebo posledná v poradí. Úplné riešenie tohto problému bohužiaľ nie je v rukách študenta.

[3] https://play.google.com/store/apps/details?id=com.rosteam.gpsemulator

Kapitola 10

Možné rozšírenia

Vďaka vytvorenej mobilnej aplikácií a servera je možné pomocou určitých faktorov naplánovať, a následne vyhľadať najrýchlejšie trasy medzi dvoma zastávkami v rámci mestskej hromadnej dopravy v Brne. Aplikácia okrem toho poskytuje aj vyhľadanie odjazdov zo zastávky, či zobrazenie pokynov pre trasu, ktoré, ak sú uložené, je možné zobraziť aj bez pripojenia na internet.

V rámci budúceho rozvoja by však mohli byť implementované ďalšie rozšírenia, ktoré by zlepšili aplikáciu, čim by sa mohla zvýšiť ako užívateľská základňa, tak aj spokojnosť užívateľov. Medzi tieto rozšírenia napríklad patria:

- **Notifikácie** - v budúcnosti by mohla byť pridaná podpora pre notifikácie, ktoré by mohli užívateľovi pripomenúť blížiaci sa odjazd pre uloženú trasu, prípadne iné relevantné informácie.
- **Aplikácia na platforme iOS** - v rámci tejto diplomovej práce bol vytvorený len klient pre platformu Android, preto by v budúcnosti bolo vhodné rozšíriť aplikáciu na platformu iOS, ktorú využíva vo svojich produktoch firma Apple. Podpora oboch platforiem by zvýšila počet užívateľov majúcich záujem o aplikáciu.
- **Podpora kráčania medzi stanicami** - hlavnou nevýhodou aplikácie je zmena spoja len v rámci danej stanice. Avšak niekedy by presun na inú stanicu kráčaním mohol byť rýchlejší ako prostredníctvom mestskej hromadnej dopravy.
- **Počítanie s výlukami** - súčasná implementácia počíta s tým, že každá zastávka nasleduje za sebou. Pomocou implementovaného tlačidla je možné zastávku preskočiť na ďalšiu v poradí. Avšak preskočenie zastávky sa nezaobíde bez interakcie s užívateľom. V budúcnosti by preto bolo vhodné upraviť aplikáciu tak, aby bola automaticky sama schopná spraviť toto preskočenie.

Kapitola 11

Záver

Zadaním tejto prace bolo okrem oboznámenia sa dostupnými službami IDS JMK, preštudovania relevantných prác riešených v minulých rokoch a identifikovania nedostatkov, aj vytvorenie mobilnej aplikácie na platforme Android, ktorá bude schopná navigovať užívateľa do destinácie v čo najkratšom čase.

Súčasťou zadania bolo tiež vytvorenie serverovej aplikácie, ktorá by poskytovala potrebné dáta. Pre splnenie zadania tejto práce bolo nutné si naštudovať okrem princípov vytvárania aplikácií pre platformu Android, aj teóriu grafov spolu s jednotlivými grafovými algoritmami.

Vytvorená serverová aplikácia využíva k nájdeniu najrýchlejšej trasy grafový algoritmus A*. Navyše je schopná poskytovať dáta aj iným klientom. Mobilná aplikácia umožňuje okrem vyhľadávania trasy medzi jednotlivými stanicami, navyše aj zobrazovať najbližšie odjazdy z vybratej stanice, či zobraziť a uložiť pokyny k trase. Aplikácia tiež ponúka možnosť nastavenia počtu vyhľadaných trás, zvolenie štartovacej obrazovky, či zapnutie automatického nástupu do vozidla. Najdôležitejšou časťou aplikácie je režim navigácie, ktorý má za úlohu navigovať užívateľa do destinácie v čo najkratšom čase. Ako však vyplynulo z testovania, problém s polohou zastávok ojedinele komplikuje detekciu užívateľa na zastávke, čo negatívne ovplyvňuje správnu funkčnosť režimu.

Literatúra

[1] Abiy, T.; Pang, H.; Tiliksew, B.: *A* Search*. [Online; navštívené 13.01.2018]. URL https://www.brilliant.org/wiki/a-star-search/?subtopic=algorithms&chapter=graph-algorithms

[2] Android: *Distribution dashboard*. [Online; navštívené 08.05.2018]. URL https://developer.android.com/about/dashboards/

[3] Android: *Save data in a local database using Room*. [Online; navštívené 02.05.2018]. URL https://developer.android.com/training/data-storage/room/

[4] Barry, D. K.: *SOAP*. [Online; navštívené 14.11.2017]. URL https://www.service-architecture.com/articles/web-services/soap.html

[5] Barry, D. K.: *Web Services Description Language (WSDL)*. [Online; navštívené 14.11.2017]. URL https://www.service-architecture.com/articles/web-services/web_services_description_language_wsdl.html

[6] Barry, D. K.: *Web Services Explained*. [Online; navštívené 14.11.2017]. URL https://www.service-architecture.com/articles/web-services/web_services_explained.html

[7] Bondy, J. A.; Murty, U. S. R.: *Graph theory with Applications*. Elsevier Science Publishing Co., Inc., 1982, ISBN 0-444-19451-7.

[8] Borah, B. B.: *Dagger 2 Generated Code*. [Online; navštívené 27.04.2018]. URL https://medium.com/mindorks/dagger-2-generated-code-9def1bebc44b

[9] Carlson, S. C.: *Graph theory*. [Online; navštívené 24.12.2017]. URL https://www.britannica.com/topic/graph-theory

[10] Chodorow, K.; Dirolf, M.: *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 2010, ISBN 978-1-449-38156-1.

[11] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; aj.: *Introduction to Algorithms*. Massachusetts Institute of Technology, 2009, ISBN 978-0-262-03384-8.

[12] GitHub: *Dagger*. [Online; navštívené 27.04.2018]. URL https://google.github.io/dagger

[13] Google Play: *DPMBinfo*. [Online; navštívené 17.10.2017]. URL https://play.google.com/store/apps/details?id=cz.dpmb.dpmbinfo

[14] Google Play: *iRIS*. [Online; navštívené 17.10.2017].
URL https://play.google.com/store/apps/details?id=cz.zolex.iris

[15] Google Play: *Jízdní řády IDOS*. [Online; navštívené 17.10.2017].
URL https://play.google.com/store/apps/details?id=cz.mafra.jizdnirady

[16] Jemerov, D.; Isakova, S.: *Kotlin in Action*. Manning Publications, 2017, ISBN 978-1-61729-329-0.

[17] JetBrains: *Comparison to Java Programming Language*. [Online; navštívené 01.04.2018].
URL https://kotlinlang.org/docs/reference/comparison-to-java.html

[18] Jezdím pro Brno: *JAK FUNGUJE KALKULAČKA?* [Online; navštívené 10.05.2018].
URL https://www.jezdimprobrno.cz/jak-funguje-kalkulacka

[19] Křivka, Z.: *Introduction to Graph Algorithms for Shortest-Paths Problems*. [Online; navštívené 31.12.2017].
URL http://www.fit.vutbr.cz/research/pubs/index.php?id=11500

[20] Kordis JMK: *Kordis JMK, a.s.* [Online; navštívené 03.01.2018].
URL http://www.idsjmk.cz/onas.aspx

[21] Kovár, M.: *Diskrétní matematika*. [Online; navštívené 28.12.2017].
URL http://www.umat.feec.vutbr.cz/~kovar/webs/personal/IDA.pdf

[22] Kuba, M.: *Web Services*. [Online; navštívené 13.11.2017].
URL http://webserver.ics.muni.cz/bulletin/articles/269.html

[23] MongoDB: *Introduction to MongoDB*. [Online; navštívené 27.04.2018].
URL https://docs.mongodb.com/manual/introduction

[24] Movable Type Scripts: *Calculate distance, bearing and more between Latitude/Longitude points*. [Online; navštívené 10.05.2018].
URL https://www.movable-type.co.uk/scripts/latlong.html

[25] Nield, T.: *Learning RxJava*. Packt Publishing Ltd., 2017, ISBN 978-1-78712-042-6.

[26] Patel, A.: *Introduction to A**. [Online; navštívené 09.01.2018].
URL https://www.redblobgames.com/pathfinding/a-star/introduction.html

[27] Paulovčák, J.: *Sledování pohybu v MHD*. Diplomová práca, Vysoké učení technické v Brně, Fakulta informačních technologií, 2017.

[28] Peter Hart and Nils Nilsson and Bertram Raphael: *A Formal Basis for a Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics, 1968, [p.100–107].
URL http://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/astar.pdf

[29] Portman, J.: *The Core Principles of UI Design*. [Online; navštívené 12.01.2018].
URL https://www.invisionapp.com/blog/core-principles-of-ui-design

[30] Robinson, I.; Webber, J.; Eifrem, E.: *Graph Databases*. O'Reilly Media, Inc., 2013, ISBN 978-1-449-35626-2.

[31] Saxena, R.: *SpringBoot 2 performance - servlet stack vs WebFlux reactive stack*. [Online; navštívené 01.04.2018].
URL https://medium.com/@the.raj.saxena/springboot-2-performance-servlet-stack-vs-webflux-reactive-stack-528ad5e9dadc

[32] Tůma, J.: *Mobilní asistent pro cestování MHD*. Diplomová práca, Vysoké učení technické v Brně, Fakulta informačních technologií, 2017.

[33] Vogel, L.; Scholz, S.; Weiser, D.: *Using Retrofit 2.x as REST client - Tutorial*. [Online; navštívené 24.03.2018].
URL http://www.vogella.com/tutorials/Retrofit/article.html

[34] W3: *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. [Online; navštívené 12.11.2017].
URL https://www.w3.org/TR/soap12-part1

[35] Walls, C.: *Spring Boot in Action*. Manning Publications, 2015, ISBN 978-1-61729-254-5.

[36] Yankov, R.: *Realm, ObjectBox or Room. Which one is for you?* [Online; navštívené 02.05.2018].
URL https://notes.devlabs.bg/realm-objectbox-or-room-which-one-is-for-you-3a552234fd6e

Príloha A

Obsah CD

CD priložené k tejto diplomovej práci obsahuje nasledujúce adresáre a súbory:

Adresár brnogo-server:
Obsahuje všetky zdrojové súbory serverovej časti.

Adresár brnogo-android:
Obsahuje všetky zdrojové súbory Android aplikácie.

Adresár tex:
Obsahuje všetky zdrojové súbory textu diplomovej práce.

Adresár video:
Obsahuje videá prezentujúce fungovanie aplikácie.

Súbor masters-thesis.pdf:
Diplomová práca vo formáte PDF.