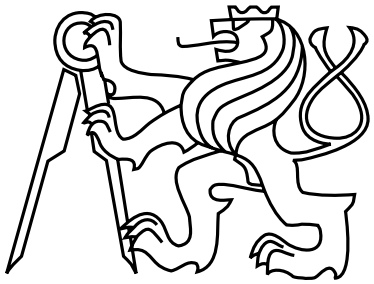
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Diplomová práce

WheelGo - Plánovač tras pro vozíčkáře

Bc. Martin Nuc

Vedoucí práce: Ing. Zdeněk Míkovec, Ph.D.

Studijní program: Otevřená informatika, Navazující magisterský

Obor: Softwarové inženýrství

29. prosince 2013

Poděkování

Rád bych poděkoval svému vedoucímu panu Ing. Zdeňku Míkovcovi a Ing. Janu Balatovi za konzultace a připomínky, které mi v průběhu realizace této práce poskytli. Dále bych rád poděkoval Jiřímu Boháčovi za inspiraci a motivaci. Velké dík také patří mým rodičům a přítelkyni, kteří mě při tvorbě této práce podporovali a poskytli mi potřebné zázemí.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 29. 12. 2013 .....

Abstract

This work deals with a problem of navigation for people in wheelchair using available map data and crowdsourcing. It describes typical difficulties of wheelchair users during the transportation and compares current projects for wheelchair users. It analyses parameterization of the navigation and suggests an algorithm for finding the optimal path. Author provides an implementation of such navigation system using cloud technologies and mobile devices. Prototype for mobile phone is tested with end users.

Abstrakt

Práce rozebírá problematiku navigace vozíčkářů s využitím veřejně dostupných mapových podkladů a crowdsourcingu. Popisuje typické překážky, na které vozíčkáři při cestování naráží, a srovnává současné projekty zbývající se problematikou vozíčkářů. Analyzuje možné parametry navigace a navrhuje algoritmus pro hledání optimální trasy. Je popsán návrh a implementace navigačního systému s využitím cloudových technologií a mobilních zařízení. Prototyp mobilní aplikace je pak otestován s koncovými uživateli.

Obsah

1 Úvod **1**
- 1.1 Motivace . . . 1
- 1.2 Cíl projektu WheelGo . . . 1
- 1.3 Negativní vymezení . . . 2

2 Analýza **3**
- 2.1 Potřeby a schopnosti vozíčkářů . . . 3
  - 2.1.1 Omezení vozíčkářů . . . 3
  - 2.1.2 Rozdíly mezi vozíčkáři . . . 4
- 2.2 Plánování trasy . . . 5
  - 2.2.1 Existující služby pro hledání tras . . . 5
  - 2.2.2 OpenStreetMap . . . 6
    - 2.2.2.1 O projektu, historie . . . 6
    - 2.2.2.2 API . . . 8
    - 2.2.2.3 Struktura odpovědi . . . 9
    - 2.2.2.4 Využití OpenStreetMap . . . 10
- 2.3 Typické překážky . . . 10
- 2.4 Stav aplikace z bakalářské práce . . . 13
  - 2.4.1 Základní popis . . . 13
  - 2.4.2 Druhy hlášení . . . 13
- 2.5 Ostatní existující projekty . . . 14
  - 2.5.1 WheelMap.org . . . 14
  - 2.5.2 Rollstuhlrouting.de . . . 15
  - 2.5.3 VozejkMap . . . 15

3 Návrh řešení **17**
- 3.1 Architektura . . . 17
  - 3.1.1 Mobilní telefon . . . 18
  - 3.1.2 WheelGo Server . . . 18
  - 3.1.3 WheelGo Navigator . . . 18
- 3.2 Komunikace jednotlivých částí . . . 18
  - 3.2.1 SOAP . . . 18
    - 3.2.1.1 ksoap2-android . . . 19
    - 3.2.1.2 Wsdl2Code . . . 19
    - 3.2.1.3 Android Web Service Client . . . 19

3.2.2 REST . . . 20
3.2.2.1 Podpora v systému Android . . . 20
3.3 WheelGo Navigator . . . 21
3.3.1 API . . . 21
3.3.2 Google AppEngine Tasks API . . . 21
3.3.3 Načtení dat z OSM . . . 22
3.4 Hledání cest . . . 24
3.4.1 Mapování souřadnic na uzel mapy . . . 24
3.4.2 Ohodnocení hran grafu . . . 27
3.5 Návrh algoritmu . . . 29
3.5.1 Popis Dijkstrova algoritmu . . . 29
3.5.2 Parametrizace algoritmu . . . 30
3.5.3 Nevýhody řešení . . . 30
3.6 WheelGo Server . . . 31
3.6.1 API . . . 31
3.6.2 OpenShift . . . 32
3.6.3 Napojeni na další systémy . . . 33
3.7 Android aplikace . . . 33
3.7.1 Uživatelské rozhraní . . . 33
3.7.2 Google Android Maps API v2 . . . 36

4 Implementace **39**
4.1 Transparentní REST service proxy . . . 39
4.2 Zpracování na pozadí pomocí Google AppEngine Tasks API . . . 42
4.3 Parametrizace Dijkstrova algoritmu . . . 43

5 Testování **45**
5.1 Testování Dijkstrova algoritmu . . . 45
5.2 Testování aplikace s vozíčkáři . . . 45
5.2.0.1 Otázky před testem . . . 46
5.2.0.2 Představení aplikace . . . 46
5.2.0.3 Test v terénu . . . 47
5.2.0.4 Otázky po testu . . . 47
5.2.1 Pilotní test . . . 47
5.2.2 Výsledky testování . . . 49
5.2.2.1 Vozíčkář 1 . . . 49
5.2.2.2 Vozíčkář 2 . . . 51
5.2.2.3 Vozíčkář 3 . . . 52
5.2.3 Závěr testování . . . 53

6 Závěr **55**

A Seznam použitých zkratek **61**

B Instalační příručka **63**
B.1 Instalace aplikace pro mobilní telefon 63
B.2 Instalace WheelGo Server 63
B.3 Instalace WheelGo Navigator 63

C Obsah přiloženého DVD **65**

Seznam obrázků

2.1 Mechanický vozík . . . 4
2.2 Editační rozhraní iD . . . 6
2.3 Struktura OpenStreetMap . . . 7
2.4 Bounding box pro severní polokouli . . . 8
2.5 Lešení na Smíchovském nádraží . . . 11
2.6 Špatně parkující automobily . . . 12
2.7 Prototyp WheelGo pro Android . . . 14
2.8 Centrum Prahy na WheelMap.org . . . 15
2.9 Nalezená trasa v rollstuhlrouting . . . 16

3.1 Diagram komponent . . . 17
3.2 Hloupý bounding box . . . 23
3.3 Bounding box bez cesty, na které se uživatel nachází . . . 23
3.4 Extrémní tvar bounding box . . . 24
3.5 Případ, kdy rozšířený bounding box nestačí . . . 25
3.6 Konečná verze bounding boxu . . . 25
3.7 Problém mapování souřadnic na nejbližší uzel mapy . . . 26
3.8 Mapování na nejbližší uzel k patě kolmice na cestu . . . 26
3.9 Procentuální vliv parametru sklon na ohodnocení hrany grafu . . . 28
3.10 Hlášeny problém zasahující spadající do více cest . . . 29
3.11 Diagram tříd - parametry navigace . . . 30
3.12 Sekvenční diagram - navigace z pohledu komponent . . . 31
3.13 Logo služby OpenShift . . . 32
3.14 Actionbar v Android 4.0+ . . . 34
3.15 Přesunutí funkcionality tlačítka pro menu v Androidu 4.0+ . . . 35
3.16 Ovládání pomocí navigation drawer . . . 35
3.17 Návrh ikon pro kategorie hlášení . . . 36
3.18 Vykreslení polygonu v komponentě Google mapy . . . 37

5.1 Zobrazení trasy v mapě . . . 46
5.2 Trasa testování . . . 47
5.3 Prvotní verze uživatelského rozhraní . . . 48
5.4 Uživatelské rozhraní po úpravě . . . 49

Seznam tabulek

2.1 Relevantní tagy v OpenStreetMap 10
2.2 Význam tagu wheelchair 15
2.3 Přehled VozejkMap API 16

3.1 HTTP metody a jejich ekvivalent CRUD operace v REST 20
3.2 API cloudové komponenty pro hledání nejkratší trasy 21
3.3 Váha parametru maximální sklon 28
3.4 Návrh API pro WheelGo Server 32

Kapitola 1

Úvod

1.1 Motivace

Téměř deset let dělám osobního asistenta člověku na vozíku. V době, kdy jsem chodil na gymnázium, se na školu obrátil vozíčkář s prosbou, zda by se nenašli dobrovolníci, kteří by mu pomohli se dostat z domu. Bydlel v bytě bez výtahu a potřeboval, aby ho někdo svezl pomocí schodolezu dolů ze schodů a zpět do bytu. Jak jsem později zjistil, byla to pouze jedna z mnoha komplikací, se kterou se takový člověk setkává.

Od té doby se pravidelně vídáme a pomáhám mu nejen se dostat domů, ale také ho doprovázím, když někam jede. Většinou jezdíme po Praze a přitom narážíme na mnohé překážky, které cestu velmi znesnadňují. Ať už se jedná o rozbité chodníky, špatně parkující auta, štěrk po zimě, ale i věci, kterých si zdravý člověk na první pohled vůbec nevšimne. Například jen pár metrů od místa, kde bydlí, je chodník, který je jako každý jiný sklopen k silnici, aby otékala dešťová voda směrem od domů. Sklon je tak příkrý, že vozík táhne silně do strany a i pro mně, jako asistenta je náročné jej udržet v přímém směru.

Přemýšlel jsem, jak nepříjemné musí být pro vozíčkáře jet někam, kde to nezná, a jak by se mu dalo v tomto směru pomoci. Inspiraci jsem našel v navigačním systému Waze, který je zaměřen na úplně jinou oblast - motoristickou navigaci.

Waze funguje na principu hlášení řidičů o vyskytujících se nehodách, uzavírkách, policejních hlídkách, o průjezdnosti jednotlivých silnic a jiných překážkách. Využívá crowdsourcingu ke sběru dat, která by se jinak těžko získávala a udržovala.

Napadlo mě tedy aplikovat podobnou metodu na problematiku vozíčkářů. Překážky, na které vozíčkáři na ulici naráží, jsou často krátkodobého charakteru (např. hromada sněhu po úklidu chodníků nebo lešení blokující cestu). Není v možnostech žádné instituce tyto překážky evidovat a jeví se tedy jako vhodné nechat samotné vozíčkáře, aby si navzájem překážky hlásili.

1.2 Cíl projektu WheelGo

WheelGo by měl vozíčkářům poskytnout možnost se navzájem informovat o nepředvídatelných překážkách pomocí moderních technologií.

Tento základní koncept jsem zpracoval v rámci své bakalářské práce [34], kde vznikl prototyp aplikace pro mobilní telefon. Tato aplikace umožňovala uživatelům se vzájemně upozorňovat na překážky, případně mezi sebou sdílet rady, pokud byla cesta někudy jinudy sjízdnější. Vyžadovala však od vozíčkáře znalost trasy, po které chtěl jet. Například pro turistu na vozíku, který se pohybuje v neznámém prostředí, by se hodila aplikace, která mu ukáže kudy přesně se lze na určité místo dostat.

V rámci mé diplomové práce jsem se tedy pokusil přidat do WheelGo navigaci na základě dostupných mapových podkladů v kombinaci s hlášeními uživatelů.

1.3 Negativní vymezení

Ve své práci vycházím z veřejně dostupných mapových podkladů OpenStreetMap, které mapují město geograficky a nezahrnují například spoje MHD, které by vozíčkář při pohybu po městě určitě využil, nebo interiéry budov. Navigace podle WheelGo tedy probíhá pouze na úrovni detailu mapy v OpenStreetMap.

Kapitola 2

Analýza

2.1 Potřeby a schopnosti vozíčkářů

2.1.1 Omezení vozíčkářů

Na první pohled jasné omezení spočívá v samotném vozíku. Ten prakticky určuje, kudy vozíčkář projede a kudy ne. Roli hraje šířka vozíku, tloušťka kol a velikost předních koleček. Základní dělení vozíků je na elektrický a mechanický.

Elektrický vozík mívá širší pneumatiky, takže se s ním dá přejet poklop od kanalizace, aniž by kolo zapadlo. Uživatel ho ovládá zpravidla joystickem s pomocí jedné ruky. Na rozdíl od mechanického vozíku bývá elektrický vozík těžší. Na místech, kde by vozíčkáři na mechanickém vozíku mohl někdo pomoci přes překážku, to s elektrickým nemusí být možné. Pro někoho také hraje významnou roli fakt, že elektrický vozík nelze připevnit na schodolez.

Mechanický vozík je cenově přijatelnější díky absenci motoru. Je také lehčí a skladnější (zpravidla se dá složit – hraje roli při použití automobilu nebo taxi). Uživatele při jízdě plně zaměstnává, protože musí zároveň používat obě ruce. Jízda na něm je fyzicky náročnější a pneumatiky bývají užší.

Na obrázku 2.1 je mechanický vozík. Přední kola jsou menší, aby byl vozík lépe ovladatelný. V některých situacích může dojít k jejich zasekávání např. při jízdě po nerovném povrchu (velmi nepříjemná je například dlažba typu „kočičí hlavy“). V takovém případě je pro vozíčkáře bezpečnější jet pozadu a využít větších zadních kol k překonání nerovnosti.

Za použití stejné techniky lze sjíždět z chodníku, kde není nájezd. Při sjíždění popředu by se vozíčkář mohl převážit a z vozíku vypadnout, proto je nezbytné jet pozadu. Tento manévr je možný pouze s asistentem.

Opačná technika se používá například při nastupování do autobusu nebo obecně při výstupu na nějakou vyšší plochu – schod, obrubník. Asistent vozíčkáře nakloní na zadní kolo a předním vjede na vyvýšenou plochu. Pak vozíčkáře vysune nahoru za předním kolem.

Je patrné, že jízda bez asistenta se výrazně liší od jízdy s ním. Přítomnost asistenta ovlivňuje i maximální sklon, který může vozíčkář překonat. Navigace vozíčkáře je tedy velice individuální a specifická.

Obrázek 2.1: Mechanický vozík

2.1.2 Rozdíly mezi vozíčkáři

První základní rozdíl jsem již zmínil – druh vozíku. Ten má vliv na způsob a flexibilitu pohybu. Elektrický vozík má navíc možnost připevnění mobilního telefonu nebo tabletu [43]. Vozíčkář by mohl mít případnou navigaci stále pod kontrolou. Naopak používání mobilního telefonu na mechanickém vozíku je složitější, protože vozíčkář musí zastavit, aby ho mohl ovládat.

Další věc, která s druhem vozíku úzce souvisí je možnost používat automobil. Takový vozíčkář se potřebuje zpravidla dostat jen od zaparkovaného vozu na cílové místo. Automobily pro vozíčkáře mají speciální úpravu, kdy vozíčkář do vozu zajede s celým vozíkem [42].

Postižení vozíčkářů často nespočívá pouze v ochrnutí spodních končetin, ale bývá rozsáhlejší. Můj kamarád má například omezenou schopnost používat pravou ruku. Když jsem ho pozval na testování aplikace na ČVUT v Praze, překvapilo mě, že nemohl projet vestibulem, který je bezbariérový. V tomto vestibulu jsou schody, které je možné objet po nájezdové rampě. Problém byl ale v tom, že zábradlí bylo pouze na jedné straně a zrovna na té, kterou nemohl použít.

Významným faktorem je také přítomnost asistenta. Toho zpravidla využívají všichni vozíčkáři, ale je finančně náročná a ne vždy si ji mohou dovolit.

Z těchto skutečností je patrné, že parametrizace navigace je velmi individuální. Samotný vozíčkář ví nejlíp, která překážka mu vadí a která nikoliv.

2.2 Plánování trasy

V průběhu své bakalářské práce jsem zkoumal, jakým způsobem si vozíčkáři plánují trasu a jaké trasy pravidelně jezdí. Vyplynulo, že obvykle jezdí stále opakující se trasy. A to především z důvodu, že je už dobře znají. Vydat se někam, kde to vozíčkář nezná, je risk a nemusí se na dané místo vůbec dostat. Pokud přesto jedou na místo, kde to neznají, tak si zjistí základní informace po telefonu a vydají se tam i přes riziko, že cestou mohou narazit na překážky.

Pokud se chystají na cestu, kterou už znají, nechávají i tak si výraznou časovou rezervu.

Testoval jsem aplikaci se třemi vozíčkáři a všichni se shodli, že by si trasu určitě naplánovali předem. Jeden z vozíčkářů byl navrženou aplikací nadšen a používal by ji i v terénu a aktivně by hlásil překážky. Zbylí dva vozíčkáři by ji používali pouze doma. Stojí za zmínku, že první vozíčkář byl ve věku 20-30, zatímco zbylí dva ve věku 50-60.

Při cestování po městě často využívají služeb MHD. Situace bezbariérové dopravy v Praze není ideální, ale postupně zlepšuje. V současné době je již 66,5 % autobusů nízkopodlažních [7] a řada stanic je bezbariérových.

Po Praze jezdí dvě speciální autobusové linky určené přímo pro vozíčkáře. Linka H1 ze stanice Chodov, přes Florenc až do stanice Obchodní centrum Černý most a linka H2 ze stanice Florenc do stanice Sídliště stodůlky. Provoz je pouze ve všední dny. Tyto autobusy mají redukovaný počet míst k sezení a rozšířený prostor pro přepravu lidí na invalidním vozíku.

Společná a centrální zastávka obou linek je Florenc, kde se nachází také bezbariérový výtah do metra. Ze zkušenosti vím, že jsem s vozíčkářem dojel k tomuto výtahu a byl mimo provoz. Museli zvolit alternativní cestu tramvají, která byla delší a komplikovanější. Kuriózní je, že to samé se stalo i při akci Crossroads 2008 [36], která je pořádána pod záštitou Červeného kříže a je určena vozíčkářům k vyměně zkušeností.

V současné době Dopravní podnik hl. m. Prahy zobrazuje aktuální stav výtahů a plošin na svých webových stránkách [8]. Ačkoliv se dá tato informace na stránkách nalézt, tak z hlavní stránky není jednoduše dostupná.

Ve výjimečných případech vozíčkáři využijí služeb taxi, který je pro ně nejspolehlivější dopravou. Tato služba je pro ně však finančně náročná.

2.2.1 Existující služby pro hledání tras

Nejprve jsem porovnal mapové služby dvou největších společností Microsoft a Google.

Microsoft nabízí Routes API [33], kde lze hledat trasu pro automobily a pro pěší. Umožňuje velice základní parametrizaci (vyhnout se dálnicím, více bodů, kudy má trasa vést, druh přepravy a jiné.). Výpočet trasy pak probíhá na serverech Microsoftu.

Google poskytuje Google Directions API [20], které funguje velice podobně jako Routes API. Na rozdíl od svého konkurenta nabízí ještě hledání cyklotras. Do vyhledávání trasy není možné nijak zasahovat z důvodu pevně daných parametrů.

Další možností je použití jedné z navigací založených na OpenStreetMap. Žádná z těchto navigací však neřeší problém nepředvídatelných překážek. Výhodou OpenStreetMap je možnost využití mapových podkladů k vytvoření specifické navigace pro vozíčkáře, proto jsem se rozhodl projekt využít v systému WheelGo.

2.2.2 OpenStreetMap

2.2.2.1 O projektu, historie

Projekt OpenStreetMap vznikl roku 2004 a jeho cílem je vytvořit volně dostupnou mapu světa nezatíženou žádnými licencemi. Idea projektu je podobná jako u Wikipedie. Mapu rozšiřuje a spravuje komunita uživatelů (k lednu 2013 bylo registrováno milión uživatelů [23]).

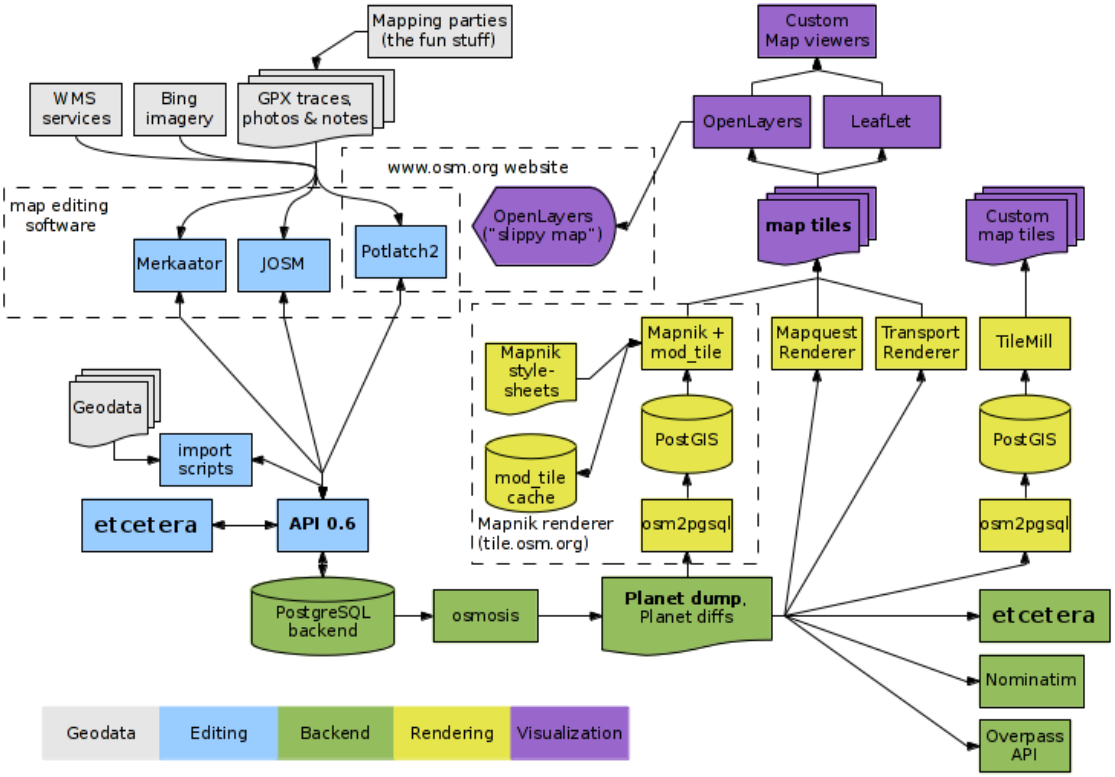
V současné době se pro editaci používá webové rozhraní iD 2.2 [24], které v květnu 2013 nahradilo starší rozhraní Potlatch [30]. K dispozici je také Java aplikace JOSM, která je určena zkušenějším uživatelům a poskytuje více možností než webová rozhraní.

Základní stavební kameny mapy jsou uzel, cesta, uzavřená cesta a vztah [26]. Uzly značí bod na určitém místě (např. telefonní budku, bankomat atp.). Cesta je seznam uzlů, které tvoří celek, a používá se k označení ulic, ale také i komunikací pro pěší. Uzavřená cesta je skupina cest, které tvoří cyklus. Používá se pro označení většího území – např. parky, vodní plochy atd. Vztahy se používají v případě, kdy něco netvoří fyzický celek, ale spadá do nějaké skupiny (např. národní park skládající se z více oddělených celků).

Každý objekt v mapě může mít ještě různé atributy. Ty slouží k bližšímu popisu elementu, ke kterému jsou navázány. Atribut se skládá z klíče a hodnoty. Například `highway=residential` znamená, že se jedná o silnici, která slouží k zpřístupnění domů.

Obrázek 2.2: Editační rozhraní iD

Mapu můžeme prohlížet pomocí komponenty Slippy Map. Jednotlivé zobrazované dlaždice pak renderuje systém Mapnik na základě dat v PostgreSQL databázi s rozšířením PostGIS. Databáze pro renderování dlaždic se liší od centrální databáze OSM a pravidelně se z ní aktualizuje. Schéma fungování OSM je na obrázku 2.3.



Obrázek 2.3: Struktura OpenStreetMap

2.2.2.2 API

OpenStreetMap poskytuje přístup ke svým datům pomocí veřejně dostupného REST API. Poslední verze nese číslo 0.6 a je dostupná od dubna 2009. Pro úpravu mapy je nutná registrace a přihlášení pomocí HTTP Basic Authorization nebo OAuth. Pro čtení mapy přihlášení potřeba není.

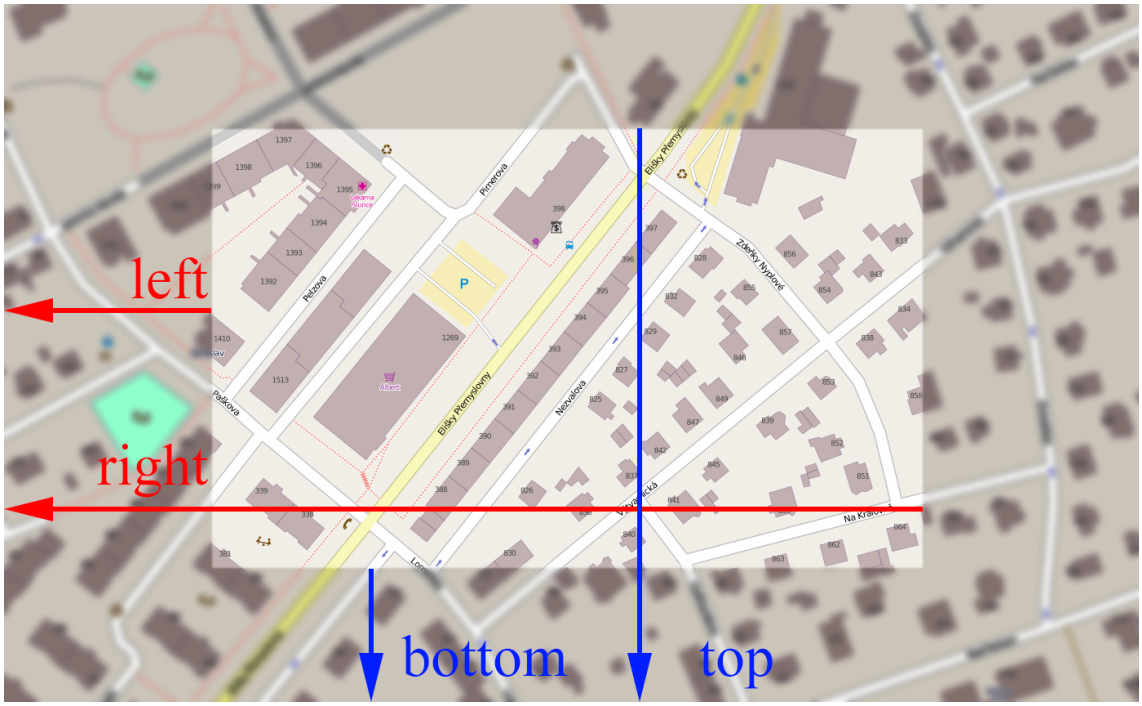
Minulý rok (2012) bylo zveřejněno Overpass API [25], které se specializuje na čtení dat. Neumožňuje mapu upravovat, ale navíc poskytuje dotazovací jazyk a je rychlejší. Komunikuje přes dotaz GET pomocí jazyka Overpass QL.

Pro případnou možnost doplňování OSM o povrchu vozovky apod. jsem se rozhodl použít API 0.6. Pro WheelGo je důležité data získat v nějaké omezené oblasti. K tomu slouží GET požadavek na adresu:

```
/api/0.6/map?bbox=left,bottom,right,top
```

Parametry - určují ohraničení tzv. *bounding box*

- left - zeměpisná délka hrany vlevo (nejzápadnější bod)
- bottom - zeměpisná výška hrany dole (nejjižnější bod)
- left - zeměpisná délka hrany vpravo (nejvýchodnější bod)
- top - zeměpisná výška hrany nahoře (nejsevernější bod)



Obrázek 2.4: Bounding box pro severní polokouli

2.2.2.3 Struktura odpovědi

Server vrátí seznam uzlů a cest v XML formátu:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <osm version="0.6" generator="CGImap 0.3.1 (28133 thorn-02.openstreetmap.org)"
3    copyright="OpenStreetMap and contributors"
4    attribution="http://www.openstreetmap.org/copyright"
5    license="http://opendatacommons.org/licenses/odbl/1-0/">
6    <bounds minlat="49.9741426" minlon="14.3755848" maxlat="49.9881589"
7        maxlon="14.3970636"/>
8      <node id="10703783" visible="true" version="4" changeset="8167689"
9        timestamp="2011-05-17T01:27:35Z" user="BiIbo" uid="3516" lat="49.9780433"
10       lon="14.3839927"/>
11     <node id="10703784" visible="true" version="5" changeset="8167689"
12       timestamp="2011-05-17T01:27:39Z" user="BiIbo" uid="3516"
13       lat="49.9806940"    lon="14.3882072"/>
14     <node id="10703781" visible="true" version="3" changeset="374893"
15       timestamp="2009-02-12T20:09:00Z" user="rpodgorny" uid="1901" lat="49.9746469"
16       lon="14.3793024"/>
17
18  ...
19
20     <way id="22737172" visible="true" version="8" changeset="12254595"
21       timestamp="2012-07-17T01:13:29Z" user="OSMF Redaction Account" uid="722137">
22       <nd ref="30006022"/>
23       <nd ref="1288420560"/>
24       <nd ref="243900013"/>
25       <tag k="highway" v="tertiary"/>
26       <tag k="name" v="Ke Krňovu"/>
27     </way>
28     <way id="22737176" visible="true" version="5" changeset="7342317"
29       timestamp="2011-02-20T13:25:15Z" user="lpechacek" uid="217051">
30       <nd ref="243900024"/>
31       <tag k="highway" v="trunk_link"/>
32       <tag k="oneway" v="yes"/>
33       <tag k="source" v="uhul:ortofoto"/>
34     </way>
35
36  ...
37
38     <relation id="13116" visible="true" version="195" changeset="19098640"
39       timestamp="2013-11-24T20:27:44Z" user="Ťule" uid="1400175">
40       <member type="way" ref="46423932" role=""/>
41       <member type="way" ref="22737183" role=""/>
42       <tag k="wikipedia" v="cs:Turistická značená trasa 0001"/>
43     </relation>
44 </osm>
```

V první části výstupu (řádky 8 až 16) jsou uzly, které se v bounding boxu nachází. Služba k nim přidá navíc i uzly, které sice nejsou přímo v oblasti bounding boxu, ale jsou součástí cesty, u které se některý z jejích uzlů v bounding boxu nachází.

V druhé části (řádky 20 až 34) následuje seznam cest. Každá cesta vede přes skupinu uzlů, na které je odkazováno pomocí atributu `ref` s hodnotou id uzlu. Pro WheelGo je důležitý atribut `highway`, který určuje, že se jedná o komunikaci. V OpenStreetMap existuje také atribut `surface`, který určuje povrch cesty, ale v České republice není často specifikován (zpravidla je u cyklotras). V tabulce 2.1 je seznam atributů schválených komunitou OSM, které by WheelGo mohlo využít.

Tag	Význam
highway	Druh komunikace (např. cyklostezka, pěší cesta, dálnice)
surface	Povrch (asfalt, dlažební kostky, štěrk)
width	Šířka v metrech
incline	Sklon
wheelchair	Vhodné pro invalidní vozík. Jde o velmi obecný atribut, možná hodnota je pouze ano/ne/omezeně

Tabulka 2.1: Relevantní tagy v OpenStreetMap

Výstup serveru končí relacemi, které pro WheelGo nejsou důležité a nebudu je proto popisovat.

2.2.2.4 Využití OpenStreetMap

V tabulce 2.1 je naznačeno, jaké existující tagy jsou v mapách k dispozici a zároveň jsou vhodné pro WheelGo. Pro přesnou navigaci vozíčkáře by bylo vhodné mít mapu s rozlišením jednotlivých chodníků, protože někdy je možné překážku objet po druhé straně vozovky. Také u nájezdů na chodník nestačí vědět, že se v ulici nachází, ale je třeba vědět, na jaké straně chodníku přesně je. V současné době se pro chodník v OSM používá tag `sidewalk` [27]. K němu je možné dospecifikovat další vlastnosti například pomocí `sidewalk:left:surface`, nicméně stále je chodník součástí jedné cesty.

Díky systému hlášení problémů se tento nedostatek v rámci WheelGo může vyřešit tím, že problematické místo nějaký vozíčkář do systému zadá a tím budu chybějící data do jisté míry kompenzovat.

2.3 Typické překážky

Hlavní výhoda WheelGo oproti existujícím projektům spočívá v možnosti zachycení problémů, které nejsou trvalého charakteru a vznikají ve městě nahodile. Na základě vlastní zkušenosti a výpovědí vozíčkářů z akce Crossroads [36] jsem se pokusil rozdělit překážky podle:

1. kategorie
2. délky trvání

Rozdělení podle kategorie Účastnící Crossroads si stěžovali na špatnou kvalitu chodníků, absenci nájezdových ramp, nedostupnost památek, nedostatek bezbariérových WC a na nefunkčnost výtahu v metru. Z vlastní zkušenosti mohu uvést příklady: prudká nájezdová rampa, lešení blokující chodník, rozkopaný chodník a obecně neprůjezdná překážka (jednou jsem narazil na trubku na silnici, kterou pěší překročí, ale vozíčkář nepřejede), hromada sněhu v zimě atd. Nejčastěji se ale setkáváme s nefunkční plošinou.

Obrázek 2.5: Lešení na Smíchovském nádraží

Některé z těchto překážek nemůžeme do systému zanést (např. absenci nájezdových ramp). V mé bakalářské práci jsem tedy hlášení rozdělil na problémy a tipy. Pokud vozíčkář ví o místě, kde naopak nájezdová rampa je, zanese tuto informaci do systému ne jako problém, jako "pozitivní informaci"– tip pro někoho jiného. V diplomové práci se ale soustředím na navigaci a zabývám se pouze problémy. Více v kapitole 2.4.

Na základě toho jsem rozdělil překážky do následujících skupin podle charakteru[1]:

- Nesjízdná místa – nesjízdné chodníky, schody, prudký sklon
- Technické závady – nefunkční plošiny, výtahy
- Stavební práce – rozkopaný chodník, zábor, lešení

[1] Jedním z případných zdrojů hlášení jsou maminky s kočárky. První kategorie je specifická pro vozíčkáře, ale ostatní tři mohou zajímat a hlásit i maminky.

- Překážky – obecná kategorie pro vše ostatní – např. zmíněná hromada sněhu, špatně parkující automobily atd.

Rozdělení podle délky trvání Druhý způsob, jak se dají překážky rozdělit, je podle předpokládané délky trvání. Například u zmiňované plošiny může být upozornění, kdy se očekává, že bude v provozu, nebo u špatně parkujících aut se dá očekávat, že tam druhý den už nebudou. Podobně hromada sněhu nevydrží přes léto.

V případě, že nemáme přesnou informaci, kdy překážka zmizí, musíme její délku trvání odhadnout. Čím kratšího charakteru překážka je, tím přesněji tuto délku umíme určit. Např. hromadu sněhu určíme v řádu měsíců, možná týdnů naopak špatně parkující automobily mohou odjet za 2 hodiny. U déle trvajících překážek stačí tedy jen zhruba odhadnout, kdy pominou. Podle délky trvání můžeme překážky rozdělit takto:

- Řád hodin – špatně parkující automobily,
- Řád dnů – nefunkční plošiny, vysypaná hromada uhlí
- Řád týdnů/měsíců – opravy chodníků, uzavírky, lešení
- Trvalé

Obrázek 2.6: Špatně parkující automobily

Pro účely WheelGo je otázka, zda hlášení trvající v řádu hodin nejsou spíše překážkou. V principu tato hlášení „nestihnou“ pomoci tolika uživatelům a jsou teoreticky častější (mohou se opakovat třeba každý den). O překážce, která se opakuje tak často, se dá pak říci, že je trvalá, ale s určitou pravděpodobností výskytu.

2.4 Stav aplikace z bakalářské práce

2.4.1 Základní popis

Ve své bakalářské práci jsem navrhoval systém hlášení, kterým by si mohli vozíčkáři navzájem radit a upozorňovat se na překážky. Vznikl prototyp aplikace pro mobilní telefon, který byl otestován v terénu. Důraz byl kladen na přizpůsobené uživatelské rozhraní s ohledem na motorické problémy vozíčkářů. Prototyp fungoval offline a ukládal data pouze do telefonu. Nebyl tedy připojen k žádnému serveru, kam by data odesílal.

Na hlavní obrazovce byla mapa, na které se zobrazovala jednotlivá hlášení (viz obrázek 2.7). Když uživatel klikl na ikonu hlášení, zobrazil se její detail. Pomocí tlačítka `Nahlásit problém` mohl zadat informaci o překážce. Z menu pak bylo možno vyvolat další typy hlášení. Zároveň bylo možno zadat textový popis a fotografii.

Při plánování trasy se uživatel podíval na mapu a zobrazil si hlášení, která se nacházela cestou, kudy chtěl jet. Na základě svých zkušeností se podle fotografie a popisu rozhodl, zda místem pojede či nikoliv.

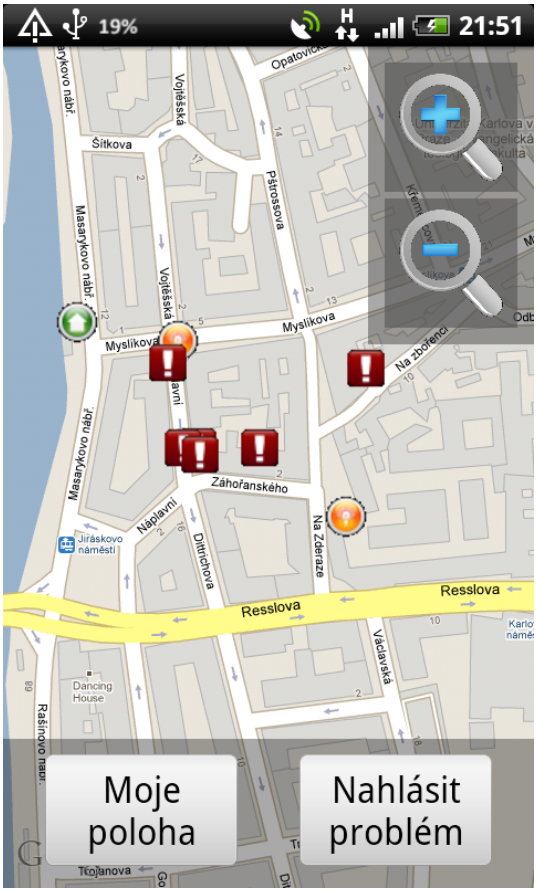
Jako velice užitečná se ukázala funkce pro zobrazení StreetView. Uživatel si mohl projít terén, aniž by na místo musel fyzicky dojet. Funkce byla dostupná v aplikaci při podržení prstu na místo v mapě.

2.4.2 Druhy hlášení

Hlášení se dělila na 3 druhy:

1. **Problém** – značí libovolnou překážku, která by vozíčkáři mohla znepříjemnit cestu.
2. **Tip** – informace, která by jinému vozíčkáři mohla nějakým způsobem pomoci, např. zkratka, nájezd na chodník atp.
3. **Místo** – místo, které by vozíčkáře mohlo zajímat, např. bezbariérové divadlo, bezbariérová restaurace

Ve své diplomové práci se soustředím na navigaci a omezím se pouze na problémy. U ostatních druhů hlášení počítám s propojením na jiné projekty. Například hlášení míst řeší projekt VozejkMap 2.5.2.



Obrázek 2.7: Prototyp WheelGo pro Android

2.5 Ostatní existující projekty

2.5.1 WheelMap.org

Tento projekt vznikl v Německu z iniciativy neziskové organizace Sozialhelden e.V. [39] v květnu 2009. Uživatelům poskytuje mapu míst s ohodnoceným stupněm bezbariérovosti. Od uvedení do provozu eviduje WheelMap téměř 400 000 klasifikovaných míst (prosinec 2013).

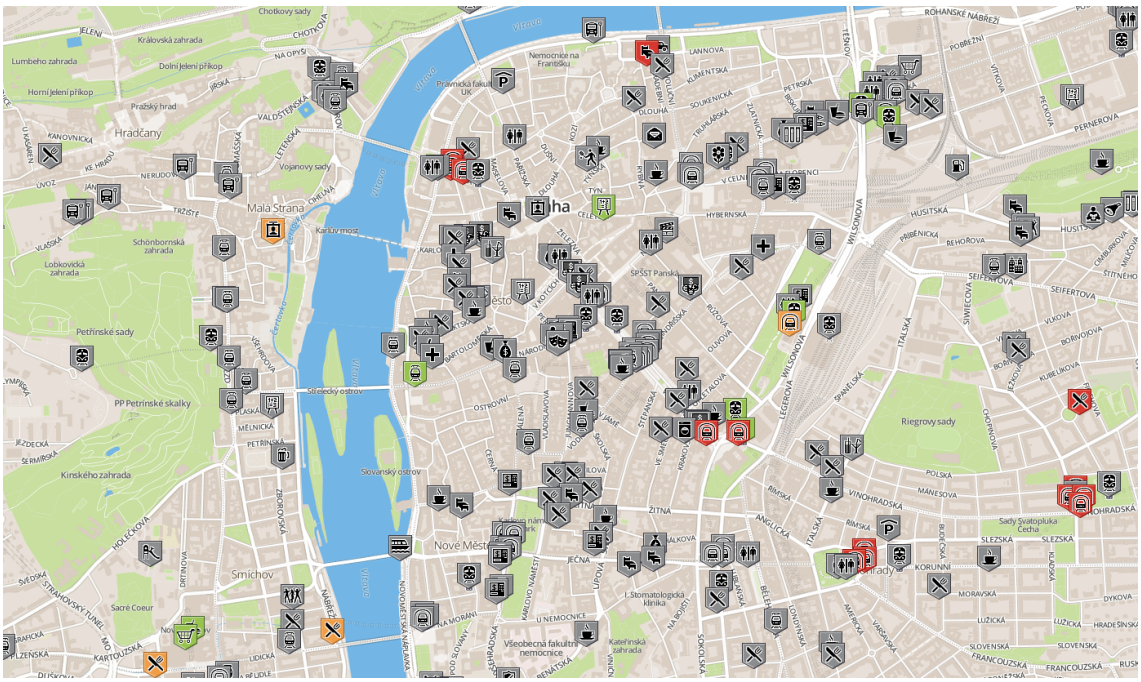
Projekt využívá OpenStreetMap. Díky tomuto projektu v OSM existuje tag `wheelchair`. Tento tag může nabývat tří hodnot viz tabulka 2.2. Popis není příliš přesný a WheelMap plánuje přidání tagu `wheelchair:description`, který by měl obsahovat slovní dodatek. Tento tag ale ještě nebyl schválen komunitou OpenStreetMap [29].

Pro WheelMap.org existuje aplikace pro mobilní telefon (Android a iPhone), takže je možné místa zobrazovat a klasifikovat v terénu.

Potěšující je, že ve WheelMap.org najdeme ohodnocené například všechny stanice metra v Praze, stanice vlaků, některé restaurace a památky po celé České republice, zpravidla ve větších městech. Pro ilustraci přikládám obrázek 2.8, který reprezentuje stav v Praze.

Hodnota	Význam
yes	plně bezbariérové
limited	částečně přístupné - např. když chybí bezbariérová toaleta
no	úplně nepřístupné

Tabulka 2.2: Význam tagu wheelchair



Obrázek 2.8: Centrum Prahy na WheelMap.org

2.5.2 Rollstuhlrouting.de

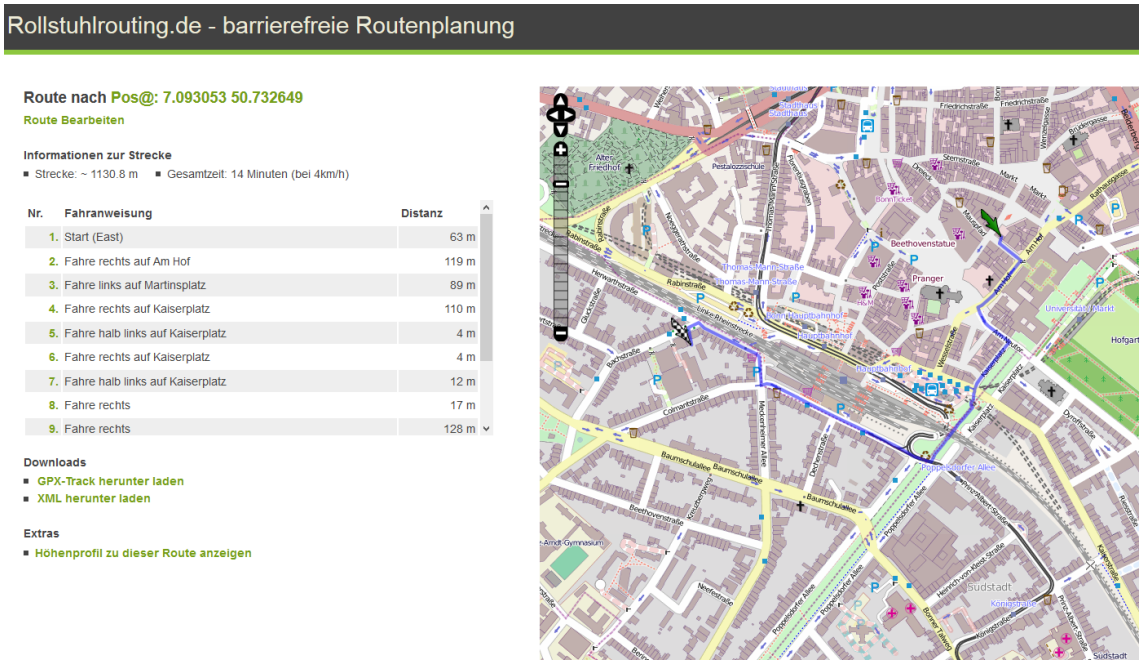
Rollstuhlrouting vznikl také v Německu rok po uvedeni WheelMap.org. Pilotní běh probíhal v Bonnu v rámci jedné diplomové práce na Ruprecht-Karlově univerzitě v Heidelbergu [13]. Narozdíl od WheelMap.org nabízí uživateli vyhledávání optimální trasy.

Navigace se počítá po cestách, které jsou explicitně označeny jako pro vozíčkáře přístupné [28] a mají specifikovaný sklon, povrch apod. Uživatel si pak může zvolit parametry navigace jako maximální sklon, maximální výška obrubníku nebo druh povrchu.

Projekt je přístupný pouze přes webové rozhraní (obrázek 2.9). Neexistuje aplikace pro mobilní telefon, která by uživatele naváděla přímo na místě.

2.5.3 VozejkMap

VozejkMap byl spuštěn v únoru 2013 a svým charakterem se podobá WheelMap.org – eviduje bezbariérová místa. V současné době (prosinec 2013) je zaznamenáno 4230 bezbariérových míst. Místa jsou vkládána a ověřována samotnými uživateli. Kontrolu provádí



Obrázek 2.9: Nalezená trasa v rollstuhlrouting

operátor, který je sám vozíčkář. Místa jsou rozdělena do kategorií a najdeme u nich textový popis, fotografii, hodnocení a komentáře.

Projekt vznikl z příspěvku Nadace Vodafone a provozuje ho CZEPRA (Česká asociace paraplegiků). Poskytuje veřejné API, kterým je možné k datům přistupovat. Hodí se tedy na evidenci bezbariérových míst z mé bakalářské práce.

K API je možné přistupovat pomocí HTTP požadavku GET na adresu:

`www.vozejkmap.cz/opendata/locations.json`

Server VozejkMapu vrátí veškerá evidovaná místa ve formátu JSON ve struktuře podle tabulky 2.3. Tento výstup je aktualizovaný jednou za 24 hodin [4].

Atribut	Typ	Popis
title	string	Název místa
location_type	int	ID typu místa
lat	float	Zeměpisné šířka místa
lng	float	Zeměpisná délka místa
attr1	int	ID typu bezbariérového přístupu
attr2	string (yes/no)	Bezbariérové WC
title	string (yes/no)	Parkoviště pro vozíčkáře
title	string	Autor

Tabulka 2.3: Přehled VozejkMap API

Kapitola 3

Návrh řešení

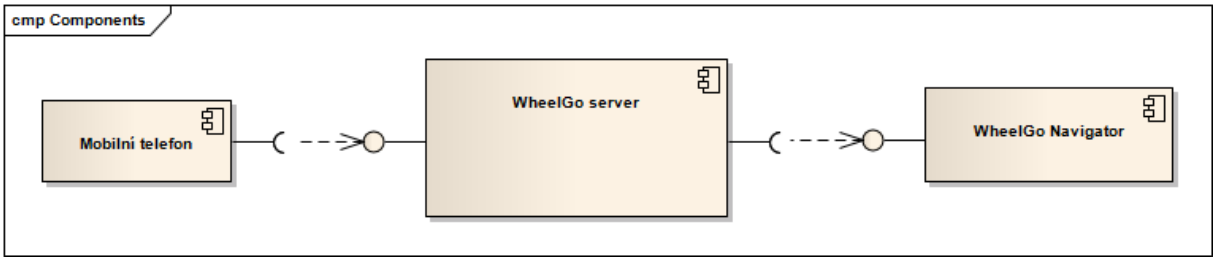
Cílem je poskytnout vozíčkářům systém, který jim bude hledat optimální cestu z jednoho místa na druhé. Cesta by se měla hledat pomocí mobilního zařízení, být zpracována na serveru a nakonec prezentována uživateli. Při zadávání navigace by měl mít možnost vybrat hlášení, kterým se chce vyhnout, a systém je poté v trase zohlední.

Pro zjednodušení jsem oproti své bakalářské práci vynechal tipy a místa. Budu se tedy zabývat pouze hlášenými překážkami.

3.1 Architektura

Aby byl systém modulární, rozdělil jsem ho do tří částí. Každou část a její úlohu rozeberu v následujících podkapitolách.

1. Mobilní telefon
2. WheelGo server - zde se spravují hlášení o překážkách
3. WheelGo navigator - umístění v cloudu, zde se počítá optimální trasa ve spolupráci s OpenSreetMap



Obrázek 3.1: Diagram komponent

3.1.1 Mobilní telefon

Jako cílovou platformu pro aplikaci na mobilním telefonu jsem zvolil systém Android. Hlavním důvodem je fakt, že je na trhu celá řada i cenově dostupnějších mobilní zařízení. Vozíčkáři, se kterými jsem se setkal, nejsou v situaci, kdy by si mohli koupit dražší telefon, a proto jsem zvolil cenově nejdostupnější variantu.

Mluvit o mobilním telefonu není úplně přesné. Může se využít také tablet. Pokud bude uživatel plánovat trasu doma, například z důvodu že se mu poměrně malý telefon špatně ovládá, může to provést právě na tabletu. Aplikace pro tablety a telefony jsou v Androidu společné a není třeba udržovat paralelně dva produkty.

Design mobilní aplikace jsem se rozhodl oproti zpracování v bakalářské práci pozměnit tak, že jsem se striktně držel Android Guidelines [15] s cílem získat srovnání oproti bakalářské práci. V té jsem naopak vymýšlel design, který by byl vhodný pro použití v terénu. Při testování se na tuto změnu. Podrobněji se o designu Android aplikace zmíním v kapitole 3.7.

3.1.2 WheelGo Server

Jedná se o centrální komponentu, která má za úkol uchovávat hlášení od uživatelů a delegovat požadavek na navigaci do cloudové části. Celý systém jsem se rozhodl postavit na Java platformě, abych se vyhnul problému nekompatibility datových typů (např. datum) při komunikaci jednotlivých komponent.

3.1.3 WheelGo Navigator

Hledání nejkratší cesty může být časově náročný úkol a není vhodné s ním zatěžovat hlavní server. Rozhodl jsem se tedy hledání trasy vyčlenit zvlášť jako samostatnou komponentu, kterou bude možné nasadit do cloudového prostředí a rozložit tak zátěž mezi více počítačů. Výhodou řešení je také možnost škálovat výkon do budoucna, pokud by počet požadavků vzrostl.

Aby nebylo třeba řešit architekturu, hledal jsem službu typu PaaS [2]. Rozhodoval jsem se mezi platformou Microsoft Azure a Google App Engine. Nakonec jsem zvolil Google App Engine [16], protože nabízené služby byly prakticky totožné. App Engine i Azure dokáže zpracovávat úlohy na pozadí. V době výběru architektury disponoval Azure navíc pouze databází SQL Azure, což je modifikovaný MS SQL Server. Tato databáze byla za měsíční poplatek a není pro plánování tras potřeba (dnes něco podobného poskytuje i App Engine). Stejně jako u volby architektury pro server jsem preferoval společný programovací jazyk Javu, takže nakonec jsem dal přednost App Engine.

3.2 Komunikace jednotlivých částí

3.2.1 SOAP

SOAP je standardní řešení pro webové služby. Je velice dobře popsané a z WSDL souboru je možné vygenerovat kód pro klienta i server. Webové služby SOAP používají formát XML, který je dobře čitelný pro člověka, ale přenáší oproti binárním protokolům více dat. To může

být na závadu zvlášť u mobilní aplikace, kde je v našich podmínkách omezené připojení k internetu a může mít i negativní dopad spotřebu elektrické energie.

Implementace webové služby pomocí SOAP je v Javě velice jednoduchá. Podle specifikace JAX-WS stačí třídu oanotovat jako `@WebService` a metodu jako `@WebMethod`. Virtuální stroj se pak sám postará o vygenerování deskriptoru WSDL, mapování objektů a předávání požadavků kódu. Aplikační server zpravidla zahrnuje nějakou implementaci JAX-WS. V opačném případě lze použít referenční implementaci Metro nebo Apache CXF.

Klientskou část je možné vygenerovat na základě souboru WSDL. Při porovnávání technologií se ukázalo, že k tomuto účelu lze použít aplikaci `wsdl2java` [40], která je součástí frameworku Apache CXF.

Vyzkoušel jsem také použitelnost SOAP pro Android. Cílem bylo na základě WSDL souboru vygenerovat rozhraní, které se bude používat pro komunikaci se serverem a odstíní kód aplikace od implementačních detailů jako parsování XML, navazování http spojení apod.

3.2.1.1 ksoap2-android

První výsledek při hledání Googlem odkazuje na knihovnu ksoap2-android. Její použití je relativně nízkoúrovňové. Programátor zde v podstatě skládá a dekomponuje XML dokument pomocí objektů a implementace vyžaduje značné množství kódu navíc.

3.2.1.2 Wsdl2Code

Částečným řešením je služba Wsdl2Code [3], která na základě souboru WSDL generuje zdrojový kód používající knihovnu ksoap2-android. V době testování jednotlivých řešení měla tato služba výpadky a někdy se stalo, že se bezdůvodně WSDL nepodařilo zparsovat. Tento přístup byl tedy též zavržen.

3.2.1.3 Android Web Service Client

Další alternativou je generátor `android-ws-client` [1], který využívá framework Apache CXF a generuje kód pomocí `wsdl2java`. Tento způsob není možné použít přímo, protože Android neobsahuje některé třídy z JAXB. `Android-ws-client` s sebou přináší několik knihoven, které chybějící třídy doplňují. Podařilo se vytvořit funkční kód pro jednoduchou Hello World aplikaci. V dalším kroku bylo testováno posílání objektů, které fungovalo bezchybně. Vygenerovaný kód obsahuje anotace, které pak `android-ws-client` používá k mapování objektů pomocí reflexe.

Tento směr se zdál úspěšný, ale při dalším testování začaly vznikat problémy. První nastal, když se předával děděný objekt. Celý postup jsem oddebugoval a zjistil jsem, že se setter zděděných atributů očekává pouze na potomkovi a nikoliv na předkovi. Bug jsem opravil a v dubnu 2013 odeslal na stránky knihovny [32]. Projekt se zdá být neaktivní, na bugfix nepřišla žádná odpověď a poslední release verze je z 20. června 2012.

Další problém nastal v okamžiku, kdy metoda vracela typ void a `android-ws-client` nastavoval chybně Response objekt v důsledku čehož klientská část padala. Na základě těchto zkušeností jsem se rozhodl vyzkoušet REST.

Metoda	Ekvivalent CRUD	Funkce
GET	READ	získá data ze serveru
PUT	UPDATE	aktualizuje prostředek nebo vytváří nový s určitým identifikátorem
POST	CREATE	vytváří reprezentaci prostředku
DELETE	DELETE	odstraní prostředek

Tabulka 3.1: HTTP metody a jejich ekvivalent CRUD operace v REST

3.2.2 REST

REST definoval Roy Fielding roku 2000 ve své disertační práci [10]. Jedná se o architektonický styl pro sdílení dat. Definuje tato architektonická omezeni:

- klient-server (client-server) - oddělení klientské a serverové části
- jednotné rozhraní (uniform interface) - každý prostředek má unikátní adresu, manipuluje se s reprezentací prostředku
- bezstavovost (stateless) - umožňuje škálování, zavádí nutnost v každém požadavku posílat potřebné informace k jeho zpracování
- kód na vyžádání (code on demand) - klientská aplikace může stahovat kód ze serveru a rozšířit tím svou funkcionalitu
- možnost využití vyrovnávací paměti (cacheable) - zkráceni latence
- vrstvený systém (layered system) - aplikace se skládá z více vrstev a ty nevidí dále než na vrstvy sousední

Pokud jsou tato omezení splněna, říkáme, že je aplikace RESTful. Použití není vázané na žádný protokol, nicméně v současné době se používá převážně přes HTTP. Data jsou adresována pomocí URL. K práci s nimi se využívá metod HTTP protokolu GET, PUT, DELETE, POST (a další). V tabulce 3.1 je přehled, jak implementovat základní CRUD operace pomocí REST. Pro úplnost – používají se také http metody HEAD, OPTIONS a PATCH, viz zdroj [12], kde jsou metody vysvětleny. V praxi tedy definujeme prostředky, metody a URL, na kterých budou dostupné.

3.2.2.1 Podpora v systému Android

Po špatné zkušenosti s webovými službami pomocí SOAP jsem se zaměřil na podporu REST v Android. Narozdíl od SOAP je REST Googlem podporován a dokonce je k dispozici prezentace Developing Android REST client applications [41] z Google I/O 2010. Google v této prezentaci (15:46) doporučuje použít formát JSON pro serializaci objektů, protože implementace v systému Android je v porovnání s parsováním XML velice dobře optimalizovaná.

Při posílání objektů se objevily problémy se serializací kolekcí, které je třeba deserializovat ručně. Google proto vydal knihovnu Gson, která tento problem řeší. Podporuje nejen složitější

objekty, ale také například dědičnost nebo Javovská generika. Použití je velice jednoduché. Pro serializaci:

```
private String serialize(ComplexClass input)
    Gson gson = new Gson();
    return gson.toJson(input)
}
```

A pro deserializaci:

```
private ComplexClass deserialize(String input)
    Gson gson = new Gson();
    return gson.fromJson(reader, ComplexClass.class);
}
```

Odeslání požadavku na server je pak jednoduchý HTTP paket. Proto jsem se rozhodl použít REST v kombinaci s knihovnou Gson.

3.3 WheelGo Navigator

Jako samostatnou komponentu jsem vyčlenil část, která načítá data z OpenStreetMap a pak v nich hledá nejkratší cestu. Myšlenka je taková, že při požadavku na navigaci WheelGo server předá požadavek do cloudu, kde se trasa začne počítat a WheelGo Server se pak průběžně dotazuje cloudu, zda je trasa už spočítána. Nakonec k ní připojí hlášení, která se v blízkosti trasy nachází, a výsledek předá uživateli.

3.3.1 API

Pro účely WheelGo stačí pouze jednoduchá REST služba, která přijme požadavek o navigaci a pak dokáže vrátit výsledek. V App Engine toto můžeme realizovat pomocí knihovny Jersey. Metodou POST se bude navigace zadávat ke zpracování. Po výpočtu trasy dojde k vyzvednutí výsledku serverem pomocí metody GET. Přehled API je uveden v tabulce 3.2.

URL	HTTP Metoda	Popis
/navigate	POST	Zadání požadavku navigace
/result	GET	Vyzvednutí nalezené trasy

Tabulka 3.2: API cloudové komponenty pro hledání nejkratší trasy

3.3.2 Google AppEngine Tasks API

Aby mohl být úloha zpracována na pozadí, poskytuje AppEngine dvě možnosti:

- Task Queue API

- Backend

Backend je placená služba a je určena aplikacím, které potřebují neustále běžet a něco zpracovávat. Pro WheelGo je tedy vhodnější Task Queue, což je v podstatě jednorázový úkol ke zpracování na pozadí.

Doba zpracování úlohy u Tasks Queue je omezena na 10 minut, což je v tomto případě stačí. Kdyby výpočet trval déle, uživatel o něj už nebude mít zájem.

V rámci Task Queue je na výběr použití Push Queue a Pull Queue. Výhodou je, že při použití Push Queue se AppEngine postará o alokaci zdrojů, loadbalancing a životní cyklus úkolu sám. Naopak při použití Pull Queue má sice programátor více kontroly, ale zároveň i více práce. Výhodnější je tedy použití Push Queue.

Úloha na pozadí je jako taková Servlet, který je aktivní na určité URL. Až na úlohu přijde řada, App Engine ho sám zavolá s předem určenými parametry. Pro výpočet trasy je třeba zadat geografické souřadnice bodu, ze kterého se cestu bude hledat, souřadnice bodu cílového a parametry navigace (souřadnice problémů, kterým se chce vozíčkář vyhnout, maximální skon apod.).

Aby byla úloha potom dohledatelná, je nutno vygenerovat hash kód (dále jen `task id`) na základě souřadnic odkud a kam se má navigovat. Pod tímto identifikátorem se uloží nalezená cesta do AppEngine datastore, odkud bude později serverem vyzvednuta.

3.3.3 Načtení dat z OSM

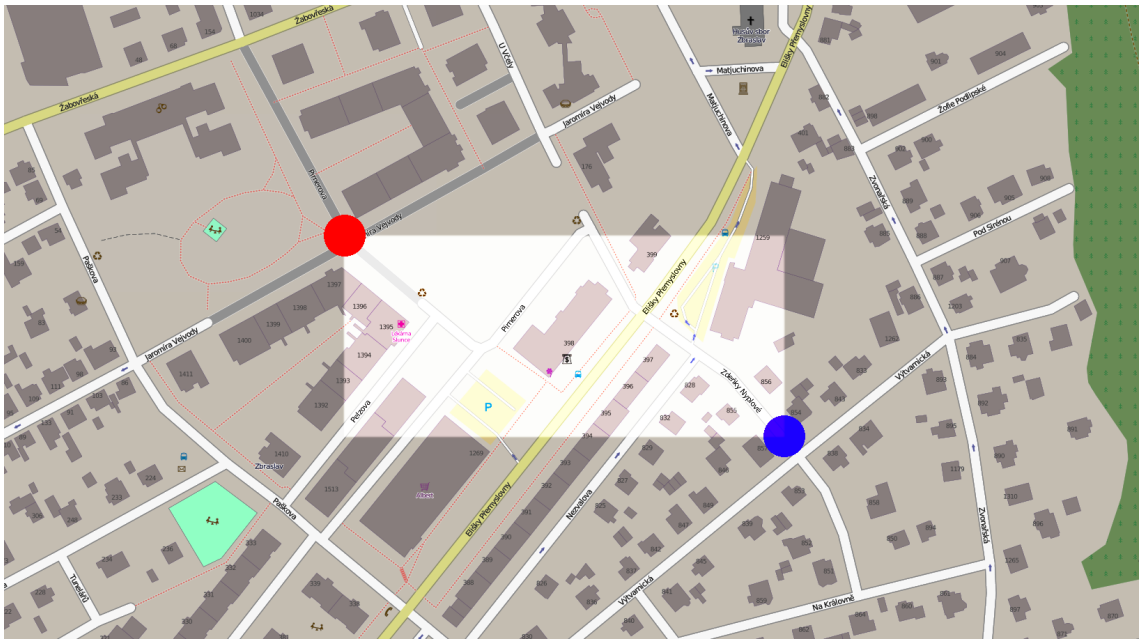
Po zadání požadavku na navigaci se musí načíst data z OpenStreetMap. K tomuto lze použít metodu API, která vrací entity v tzv. bounding boxu, tedy v ohraničené oblasti. Metoda se zavolá požadavkem GET na adresu:

```
/api/0.6/map?bbox=left,bottom,right,top
```

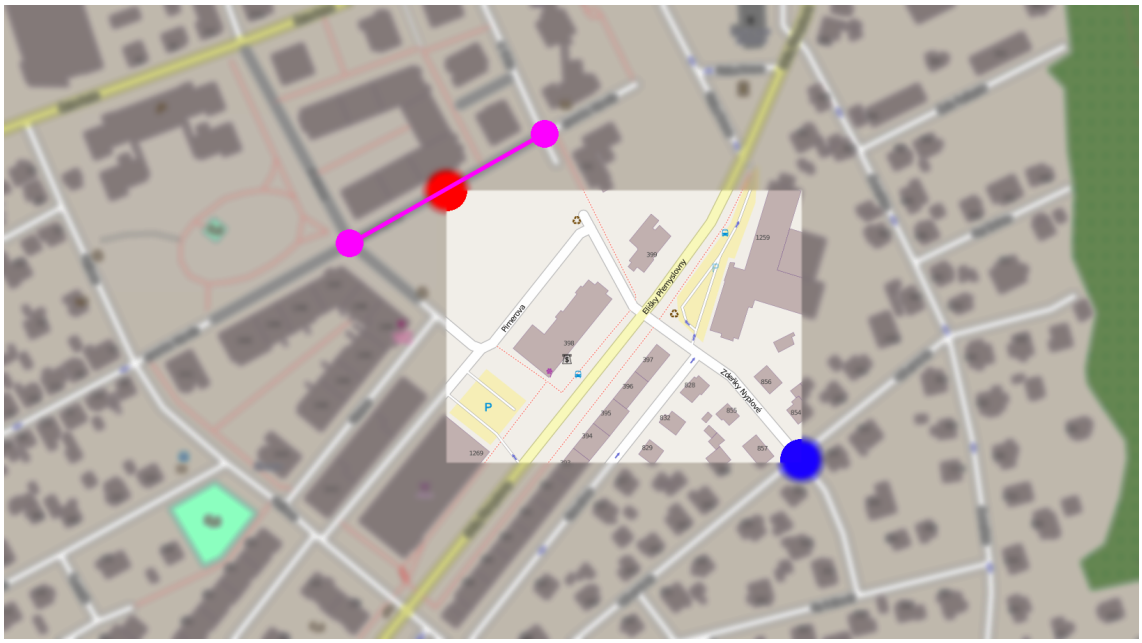
Je nezbytné určit ohraničení bounding boxu. Ten je možné definovat jako obdélník, kde výchozí a cílový bod navigace jsou jeho protilehlé vrcholy (obrázek 3.2). OpenStreetMap API vrací i entity, které přímo v bounding boxu nejsou, protože jsou součástí cesty, jejíž část se v něm nachází. Navrhované ohraničení přesto není vhodné a to ze dvou důvodů.

1. **Cesta procházející bounding boxem nemusí mít v bounding boxu žádný uzel a pak ji API nevrátí.** Na obrázku 3.2 se toto neprojeví, protože cesta, na které se uživatel nachází, je přímo v bounding boxu. Naopak na obrázeku 3.3 se růžově označená cesta sice nachází v bounding boxu, ale nemá tam žádný ze svých uzlů. Taková cesta pak ve výstupu OSM API chybí.

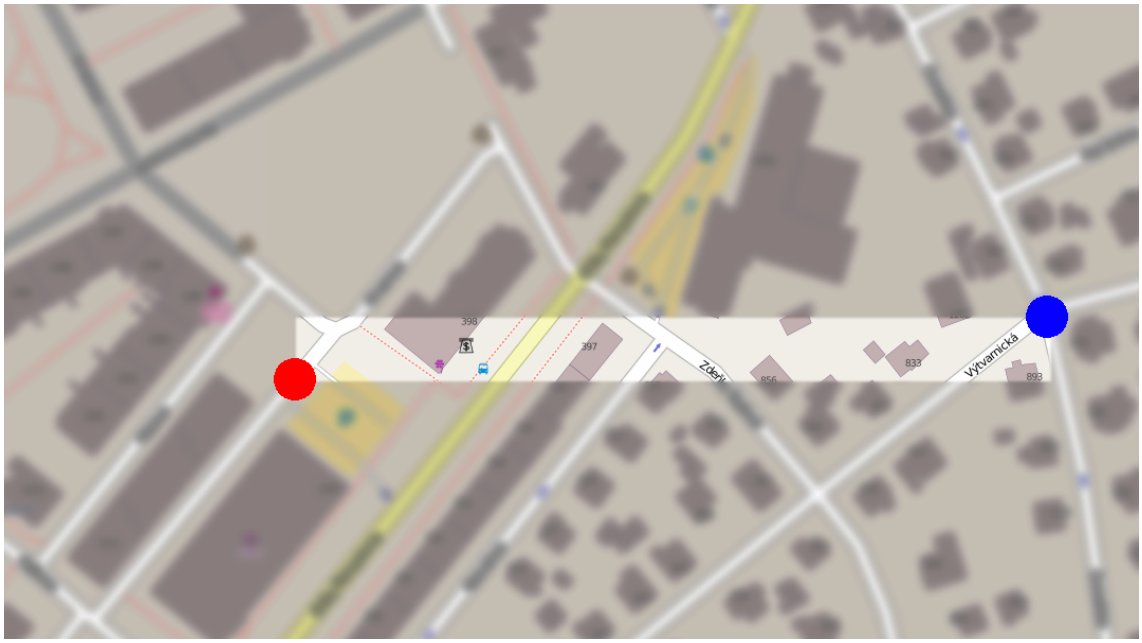
2. **Hledaná cesta může vést mimo bounding box.** K této situaci dochází, pokud optimální trasa vede mimo bounding box a pak se do něj vrací, protože přímější cesta neexistuje. Podobným případem, kdy toto může nastat je při extrémním tvaru bounding boxu, jako je na obrázku 3.4. OSM API v takovém případě vrátí jen několik ulic.



Obrázek 3.2: Hloupý bounding box



Obrázek 3.3: Bounding box bez cesty, na které se uživatel nachází



Obrázek 3.4: Extrémní tvar bounding box

Tyto dva problémy lze vyřešit tak, že bounding box zvětší a pokryje pokud možno i ulice okolo. Zvětšení bounding boxu má vliv na množství dat, které OSM server vrátí, a tím pádem i negativní vliv na dobu trvání výpočtu optimální trasy.

Částečným řešením je rozšíření souřadnic bounding boxu o konstantu 0.003. V praxi to znamená posunutí hranic boundingboxu o 0,8 km všemi směry. Optimální trasa je pak hledána i v okolních ulicích.

Toto opatření druhý problém neřeší úplně, protože trasa může vést i mimo rozšířený bounding box jako na obrázku 3.5. Takto extrémní situace nelze řešit, protože by musela být cesta hledána v kompletní mapě OSM. V případě, kdyby 0,8 km se v praxi ukázalo jako nedostatečné, je třeba konstantu zvýšit, případně zavést nějakou heuristiku, jak ji určovat.

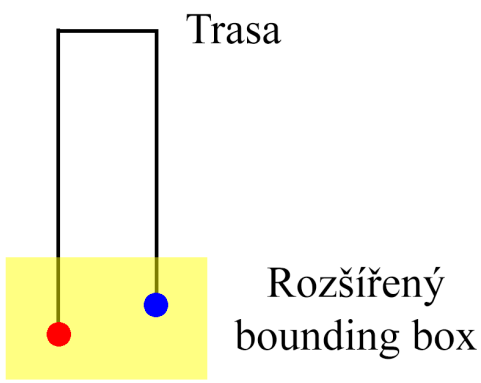
Použitý bounding box tedy vypadá podobně jako na obrázku 3.5.

3.4 Hledání cest

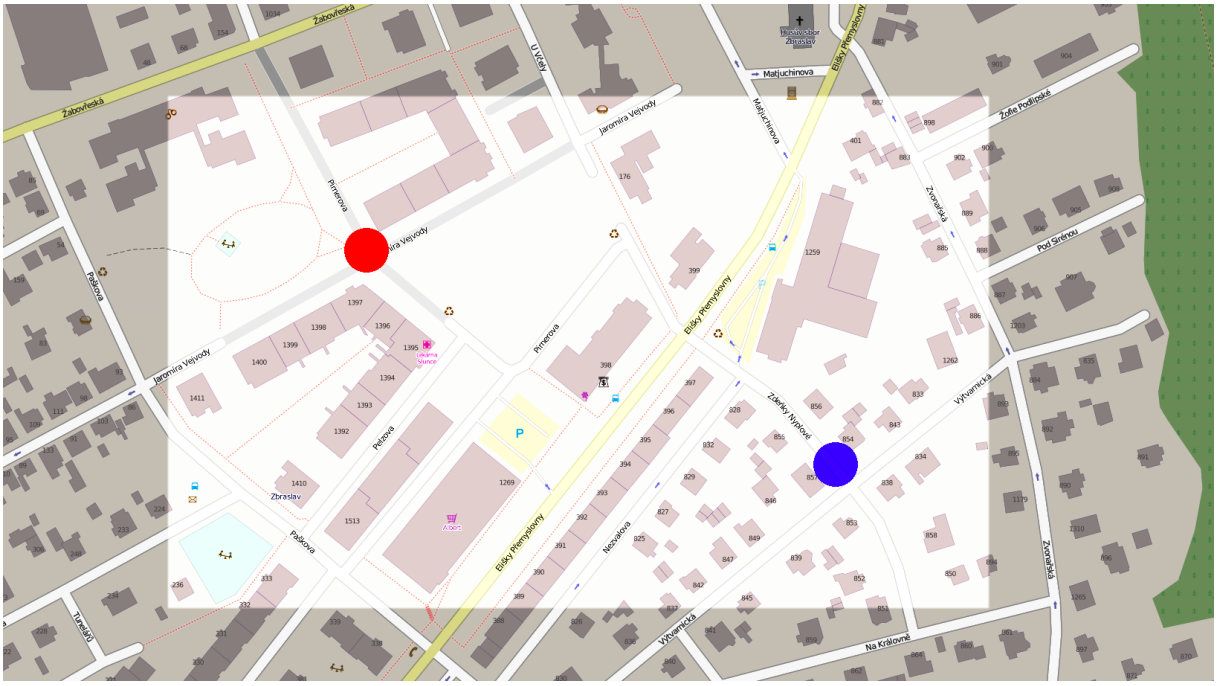
Z výstupu z OpenStreetMap (detail odpovědi serveru 2.2.2.2) lze sestavit na základě cest a uzlů neorientovaný graf. Před vlastním hledáním cesty je třeba namapovat uživatele na uzel v grafu, ze který je výchozím uzlem pro vyhledávání nejkratší cesty. Totéž je třeba provést i pro cílový uzel.

3.4.1 Mapování souřadnic na uzel mapy

První způsob, který se nabízí, je nalezení k daným vstupním souřadnicím geograficky nejbližší uzel v mapě. Tento způsob je výpočetně jednoduchý, ale nedokáže uživatele mapovat

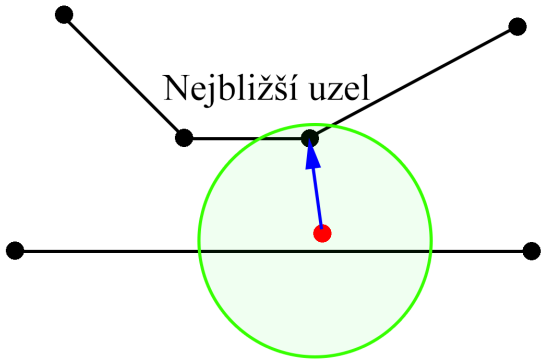


Obrázek 3.5: Případ, kdy rozšířený bounding box nestačí



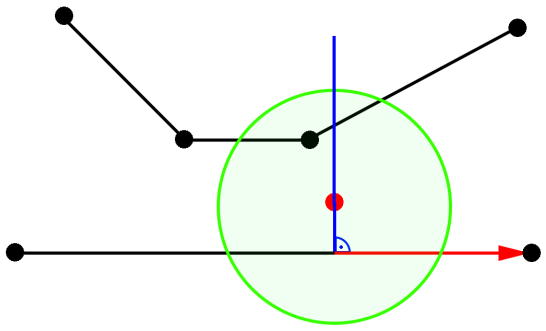
Obrázek 3.6: Konečná verze bounding boxu

na cestu. Zmíněná situace je znázorněna na obrázku 3.7, kde ačkoliv je uživatel evidentně na cestě, tak ho tato metoda mapuje na uzel, který je součástí cesty vedle, protože cesta, na které se ve skutečnosti nachází, má své uzly daleko.



Obrázek 3.7: Problém mapování souřadnic na nejbližší uzel mapy

Aby se tomuto předešlo, nabízí se řešení vyhledat k souřadnicím nejbližší hranu. Po nalezení této hrany, se spočítá vzdálenost k jejím krajním uzlům od paty kolmice procházející souřadnicemi uživatele. Souřadnice se pak namapuji na bližší z uzlů. Na obdobném obrázku 3.8 je vidět, že se souřadnice tímto způsobem mapují na správnou cestu.



Obrázek 3.8: Mapování na nejbližší uzel k patě kolmice na cestu

3.4.2 Ohodnocení hran grafu

Poslední, co zbývá, je zvolit způsob, jak ohodnotit hrany grafu, aby měl routovací algoritmus podle čeho trasu hledat. Pro zjednodušení bude navigace ovlivněna těmito parametry[1]:

- **Geografická vzdálenost** mezi dvěma body – vyjádřitelná pomocí Haversinovy rovnice [6]
- **Maximální sklon** – atribut cesty v OSM a v parametru navigace
- **Hlášení, kterým se chce uživatel vyhnout** – parametr navigace

Rovnici pro cenu hrany vyjádříme jako součin parametrů viz. 3.1.

$$cena = k * l * p \tag{3.1}$$

kde k je koeficient penalizace podle sklonu (tabulka 3.3)
l je geografická délka cesty
p je parametr závislý na hlášených problémech v okolí hrany

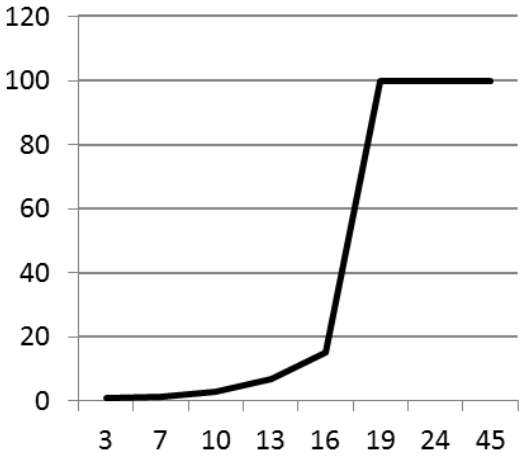
Maximálním sklon lze řešit dvěma způsoby. Za prvé jakákoliv cesta strmější než uživatelem specifikovaný maximální sklon bude ohodnocena maximální cenou. Tím se jí uživatel úplně vyhne. Pokud naopak sklon bude menší než omezení uživatele, lze jí použít a cena hrany bude určena ostatními parametry.

Za druhé je možné cestu penalizovat podle toho, jak se blíží k maximálnímu sklonu, který uživatel dokáže vyjet. Závislost výsledné ceny hrany na sklonu nemusí být lineární. Například pokud uživatel určil jako maximální sklon 19 °. Lze tento interval rozdělit do několika hladin (ilustrační obrázek 3.9).

- **0° - 7°** – pro uživatele velice snadné projet
- **7° - 10°** – stále ještě celkem snadné projet
- **10° - 13°** – sklon je větší, začínáme cestu více penalizovat
- **13° - 16°** – náročnost stoupá, penalizujeme ještě více
- **16° - 19°** – sklon se blíží hodnotě, kterou je vozíčkář schopen v nejhorším případě vyjet. Takovou cestu penalizujeme výrazně.
- **19° a více 100** % – penalizace, tudy trasu vést vůbec nemůžeme

Konkrétní podoba a počet hladin je otázkou dalšího výzkumu. Není známá publikace zabývající se touto problematikou. Proto je zvoleno experimentálně 5 hladin (tabulka 3.3).

[1] Problematika vozíčkářů je samozřejmě složitější a navigace by měla být parametrizována přesněji. Cílem této práce je vytvořit základní koncept navigace tak, aby bylo parametry možné do budoucna rozšiřovat podle toho, jak se objeví ve zdroji dat.



Obrázek 3.9: Procentuální vliv parametru sklon na ohodnocení hrany grafu

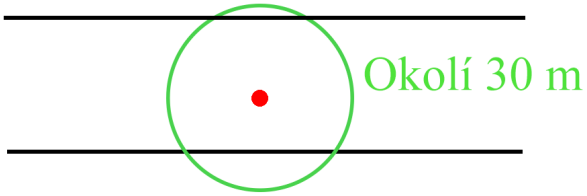
% z maximálního sklonu	koeficient penalizace k
0-20	1
20-50	1,1
50-80	1.4
80-100	1,8
100	∞

Tabulka 3.3: Váha parametru maximální sklon

Dále je třeba zohlednit **hlášení, kterým se chce uživatel vyhnout**. Parametrem je seznam souřadnic, kde se překážky nachází. Pokud nějaká hrana prochází blízko jedné z překážek, kterému se rozhodl uživatel vyhnout, tak je hrana ohodnocena nekonečnou cenou. V rovnici pro výpočet ceny je označen tento parametr jako p. Hodnota bude ∞, pokud existuje v okolí hrany překážka, které se chce uživatel vyhnout. V opačném případě bude hodnota 1, aby koeficient na výpočet neměl vliv.

Zbývá definovat okolí hrany. Hlášený problém nemusí být přímo na cestě. Vlivem nepřesnosti GPS nebo nepřesnosti ovládání displeje při ručním zadávání se dá předpokládat, že budou souřadnice hlášených problémů někde blízko cesty, nikoliv ale přesně. Lze využít metody pro mapování souřadnic na uzel hrany 3.4. Při hledání optimální trasy bude problémů, kterým se chce uživatel vyhnout, v řádu jednotek, maximálně desítek. Je tedy možné při zpracovávání hran a výpočtu jejich ceny vždy změřit vzdálenost ke všem problémům. Pokud bude vzdálenost menší než nějaká konstanta, znamená to, že problém je v okolí hrany a koeficient p bude ∞.

Tato konstantu byla experimentálně zvolena 0.0001, což odpovídá přibližně 30 metrům. Při větší hustotě komunikací může dojít k problému v případě, kdy dvě ulice povedou blízko sebe a navigátor označí obě jako nevhodné, protože není jasné, v které ulici se problém nachází (obrázek 3.10).



Obrázek 3.10: Hlášeny problém zasahující spadající do více cest

Situaci by zkomplikovalo, pokud by byly k navigaci používány mapové podklady s detailem na úrovni chodníků. Navigace by vozíčkáři radila úplně jinou trasu, ačkoliv by stačilo přejet na druhý chodník. V takovém případě by bylo třeba metodu lokalizace problému zpřesnit.

3.5 Návrh algoritmu

Úkolem je najít optimální trasu na neorientovaném ohodnoceném grafu ze zdrojového uzlu do cílového. Existuje celá řada algoritmů pro hledání nejkratší cesty. V [9] se navrhuje použití Dijkstrova algoritmu a heuristického algoritmu A*.

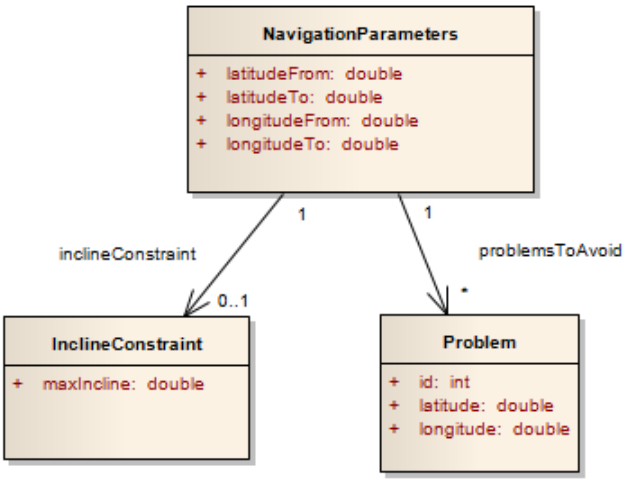
3.5.1 Popis Dijkstrova algoritmu

Tento algoritmus je možné použít pouze na grafy s nezápornými délkami hran. Algoritmus při každé iteraci zpracuje jeden vrchol a tím je zaručeno, že vždy doběhne do konce. Průběh algoritmu je popsán pseudokódem v [9].

Druhý zmiňovaný algoritmus A* se snaží obohatit Dijkstrův algoritmus o heuristiku určující, kterým směrem by měl hledat cestu. V případě navigace vozíčkářů je zřejmé, že je vhodné hledat směrem k cílovým souřadnicím. Pokud by šlo o klasickou navigaci, vzdálenost mezi uzly v Dijkstrově algoritmu by byla ovlivněna heuristickou funkcí, jejíž výstup odpovídá vzdálenosti vrcholu od cílového bodu vzdušnou čarou. Tato úprava způsobí, že algoritmus A* preferuje cesty vedoucí k cílovému uzlu. Jakmile je nalezena cesta k cílovému uzlu, tak algoritmus končí. Neprochází tedy celý stavový prostor, ale výstup je velmi závislý na heuristické funkci.

Nicméně je problematické definovat heuristickou funkci, protože do hry vstupuje více parametrů než jen geografická vzdálenost. Optimální cesta může vést oklikou a být pro vozíčkáře daleko vhodnější než cesta, která je sice přímější, ale obtížněji sjízdná.

Dijkstrův algoritmus není vhodný pro hledání cesty ve větších grafech [35], protože na rozdíl od A* prohledává graf všemi směry. V případě WheelGo to tolik nevadí, protože graf je omezený a počáteční bod je vždy téměř v rohu (viz určení bounding boxu na obrázku 3.6). Na základě těchto faktů je vhodnější použít Dijkstrův algoritmus.



Obrázek 3.11: Diagram tříd - parametry navigace

3.5.2 Parametrizace algoritmu

Parametrizace algoritmu je určena způsobem ocenění hran grafu. Ocenění hran je popsáno v kapitole 3.4.1. K této operaci potřebujeme následující vstupy:

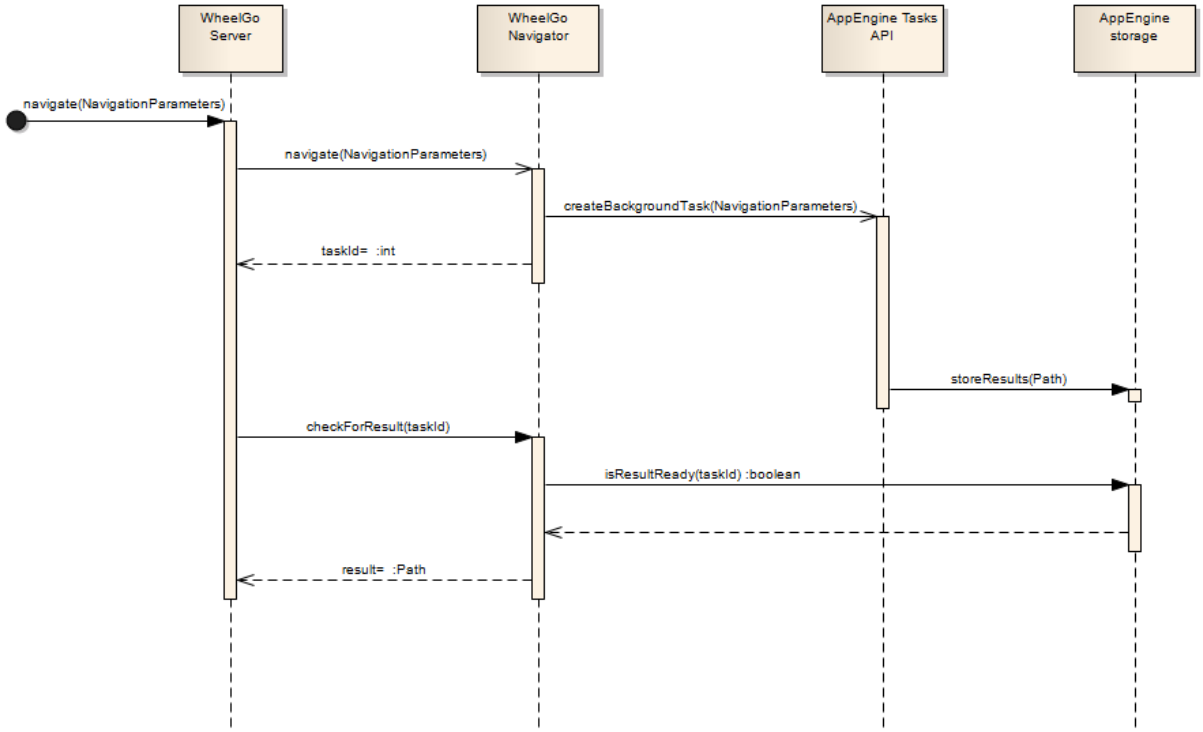
- odkud navigovat
- kam navigovat
- maximální sklon
- problémy, kterým se chce vozíčkář vyhnout

Pro komunikaci přes REST rozhraní byl navržen Data Transfer Object popsaný diagramem tříd na obrázku 3.11.

3.5.3 Nevýhody řešení

Oddělením výpočetní části od WheelGo serveru se nabízí otázka, jak zjistit, že server výpočet dokončil. Pokud nebude výpočet trvat příliš dlouho, je možné po celou dobu udržovat spojení, až do chvíle, než se výsledek spočítá. Tím by došlo ale pouze k přesunu čekání do WheelGo Navigatora, protože úloha hledání cest je prováděna na pozadí a při vytvoření se pouze zařadí do fronty úloh. O spuštění úlohy následně rozhoduje App Engine. Konec hledání trasy se pozná tak, že se v Datastore App Engine objeví entita s ID požadavku (viz 3.3.2).

Zvolíme-li variantu, kde se po zadání navigační úlohy předá řízení zpět WheelGo Serveru, znamená to, že se server musí průběžně dotazovat na výsledek Navigatora. Nabízí se řešení nechat Android aplikaci, aby si cestu vyzvedla přímo z App Enginu. Toto řešení není v tomto případě vhodné, protože cestu vrácenou Navigatorem je třeba v serveru obohatit o



Obrázek 3.12: Sekvenční diagram - navigace z pohledu komponent

problémy, které se vyskytují okolo nalezené cesty. WheelGo Server tedy musí cestu projít a ze své databáze do výstupu doplnit okolní problémy. Na obrázku 3.12 je vidět konečný návrh komunikace jednotlivých komponent při požadavku na navigaci.

3.6 WheelGo Server

3.6.1 API

Z předchozího textu lze definovat požadavky na centrální server:

- vytváření problémů
- mazání problémů
- zobrazení detailu problému
- získání problému kolem souřadnic
- navigace

Na základě těchto požadavků bylo navrženo aplikační rozhraní popsané v tabulce 3.4. Metoda pro navigaci bude přijímat parametry prostřednictvím třídy NavigationParameters (viz obrázek 3.11) v těle požadavku POST.

Funkce	URL	HTTP Metoda
hlášení problému	/problem	POST
čtení problémů	/problem?latitude=..&longitude=..&radius=..	GET
čtení detailů o problému	/problem/id	GET
mazání problémů	/problem/id	DELETE
navigace	/navigate	POST

Tabulka 3.4: Návrh API pro WheelGo Server

Při požadavku na navigaci server předá výpočet navigace WheelGo Navigatoru a v několikasekundových intervalech pak kontroluje, zda je trasa nalezena (viz obrázek 3.12). Potom server projde celou trasu a ke každému uzlu trasy vyhledá ve své databázi okolní hlášení. Do Android aplikace pak vrátí nalezenou cestu včetně těchto hlášení, která se pak uživateli zobrazí.

Server bude ukládat hlášení do databáze a bude u nich evidovat atributy:

- zeměpisná dálka
- zeměpisná šířka
- datum nahlášení
- expirace
- fotografie

3.6.2 OpenShift

U enterprise aplikací v Javě je mnohdy problematické najít levný hosting. Jedno z řešení je využití služby OpenShift. Jedná se řešení hostingu v cloudu typu PaaS od firmy RedHat. Umožňuje nasazení aplikací v různých programovacích jazycích (zmíním Java, PHP, Ruby, Python, JavaScript) v kombinaci s volitelnou databází (MySQL, PostreSQL, MongoDB).

Obrázek 3.13: Logo služby OpenShift

Společnost RedHat stojí také za aplikačním serverem JBoss, pro který je v OpenShift plná podpora. Další důvod pro technologie od RedHat je framework RESTEasy (odkaz [38]). Tento framework je certifikovanou implementací JAX-RS a umožňuje vývojáři snadno vytvořit REST API.

RedHat nabízí bezplatný tarif, v rámci kterých je možné využít 3 tzv. small gear [37]. Každá databáze, aplikační server nebo framework si vezme jeden gear. Klient využívající

bezplatný tarif si tak může složit například konfiguraci využívající PHP, MySQL databaázi s aplikací phpMyAdmin.

Sestava pro WheelGo by se tedy skládala z aplikačního serveru (JBoss Application Server 7) a databáze PostreSQL 9.2. Pro tuto konfiguraci stačí bezplatná tarifikace.

Škálování probíhá tak, že při větším počtu klientů OpenShift automaticky vytvoří novou instanci serveru, která si zabere jeden gear. To znamená, že škálovat WheelGo s bezplatným tarifem by bylo možné pouze v rámci jednoho gearu (1 gear zabral aplikační server a jeden gear zabrala databáze).

3.6.3 Napojeni na další systémy

WheelGo Server je možné místo k napojení na existující projekty, pokud by bylo třeba informace z externích zdrojů nějak obohatit na základě uložených hlášení. Zatím je zřejmá možnost spolupráce s projektem VozejkMap 2.5.2, který supluje hlášení bezbariérových míst z mé bakalářské práce.

Například by bylo možné nabízet bezbariérová místa vozíčkáři při hledání trasy. Nebo naopak poskytnout API pro VozejkMap, aby mohl zobrazovat hlášení, pokud si uživatel místo prohlíží na webu VozejkMapu.

3.7 Android aplikace

3.7.1 Uživatelské rozhraní

Aplikace v telefonu je primární způsob, jak bude uživatel k navigaci přistupovat. Na hlavní obrazovce by měla být stejně jako v původním prototypu mapa hlášení. Rozdíl je však v přítomnosti navigace.

Uživateli bude umožněno se navigovat pouze z aktuální pozice.. Dále bude uživatel muset zadat cílový bod, kam se chce navigovat. Pro zadání tohoto bodu se nabízí dvě možnosti:

1. zadání adresy, která se pak přeloží na souřadnice
2. výběr místa v mapě

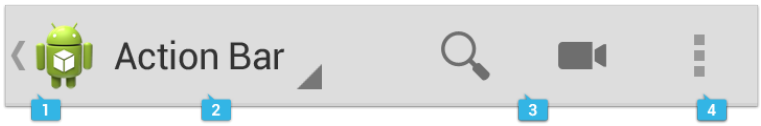
V Androidu existuje možnost převedení souřadnic na adresu. Není ale známo, jak tuto operaci udělat opačným směrem. Přímo Android tuto funkcionalitu neposkytuje a pravděpodobně by bylo třeba využít nějakou externí službu.

Jediný způsob, jak uživatel zadá cíl navigace je tedy výběrem bodu v mapě. Ten lze vybrat buď t'uknutím do mapy, nebo dlouhým podržením. První možnost by interferovala s ovládáním mapy jako takové (posun, přibližování). Druhý způsob byl v prototypu z bakalářské práce již použit pro StreetView, které je třeba zachovat. Podobně jako určení cíle navigace jde o vybrání konkrétního bodu v mapě. Jako řešení bylo zvoleno zobrazení dialogu, který se uživatele zeptá, co přesně chce dělat. Po dlouhém podržení prstem do mapy se zobrazí otázka, zda chce uživatel na místo navigovat. Uživatel má pak na výběr z následujících možností:

- Ano - odešle se požadavek na navigaci
- Zobrazit StreetView - aplikace zobrazí režim StreetView
- Ne - dialog se zavře, tato možnost je pro případ, kdyby uživatel podržel prst v mapě omylem nebo kdyby bylo špatně vyhodnoceno gesto pro posun mapy

Po potvrzení požadavku na navigaci se zavolá služba WheelGo Serveru pro navigaci. Předají se parametry místa odkud a kam se naviguje a parametry navigace (budou upřesněny následně). Server by měl vrátit trasu pro vozíčkáře a seznam hlášení, které se v okolí nalezené trasy vyskytují. Trasa se vykreslí do mapy pomocí Google Android Maps API v2 (pro podrobnosti viz 3.7.1).

Od tvorby prototypu v bakalářské práci došlo u telefonů se systémem Android k podstatné změně ovládání. Dříve povinné hardwarové tlačítko pro zobrazení menu bylo odstraněno a menu se v nových verzích Androidu vyvolává pomocí ikony v tzv. actionbaru [14]. Menu v actionbaru je reprezentováno třemi tečkami v pravém rohu aplikace – čtvrtá značka na obrázku 3.14.



Obrázek 3.14: Actionbar v Android 4.0+

Tato změna měla za následek, že se menu kvůli absenci actionbaru v prototypu zobrazovalo vpravo vedle softwarových ikon tak jako na obrázku 3.15. Na některých telefonech se dokonce zobrazilo po podržení tlačítka pro poslední spuštěné aplikace. Tento způsob byl neintuitivní a bylo třeba aplikaci znovu navrhnout s použitím actionbaru.

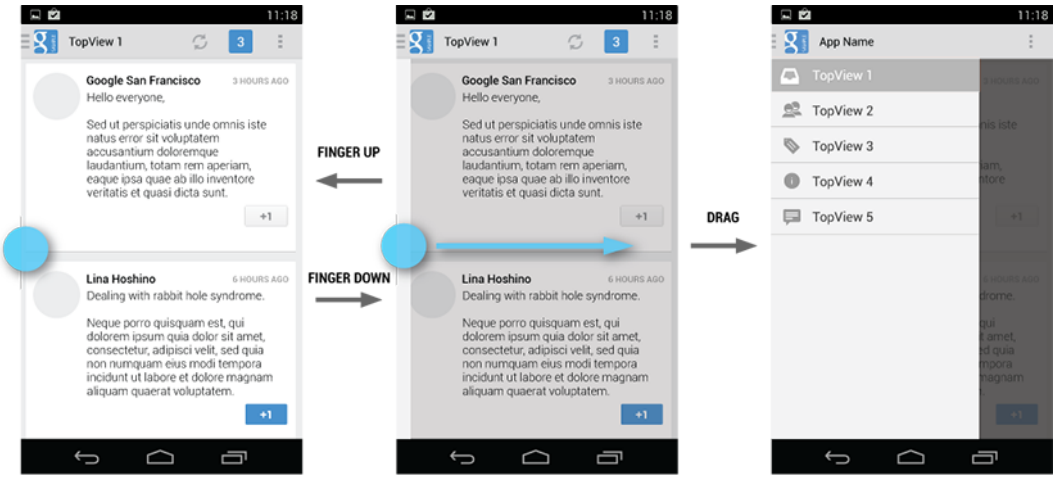
Pro navigaci v aplikaci bylo zvoleno místo standardního menu tzv. Navigation Drawer [22]. Jedná se o nabídku, které se vysouvá ze strany displeje, (3.15) a jde v současné době o preferovanou volbu navigace na platformě Android.

Vzhledem k tomu, že jsem přistoupil k novému návrhu uživatelského rozhraní, bylo použito nových ovládacích prvků, které doporučuje Google v Android guidelines [15]. V principu to znamená vůbec nepřizpůsobovat uživatelské rozhraní uživatelům. Zajímavé je srovnání s původním prototypem at' už vzhledem k ovládání mapy nebo k orientaci uživatele díky většímu prostoru pro mapu.

V mapě uživatel uvidí hlášené překážky a kliknutím na její ikonu by se měl zobrazit základní popis, aby měl zhruba představu, čeho se problém týká. Úplně se nabízí zobrazovat přímo z mapy fotografie, ale tato varianta byla nevyhovující, protože zmenšená fotografie by měla malou vypovídací hodnotu a navíc by se při každém kliknutí na hlášení musela stahovat ze serveru. To by mělo za následek dlouhou dobu reakce a negativně by to ovlivnilo celkový dojem z aplikace.

V první verzi návrhu aplikace uživatel musel zadat krátký textový popis problému, který měl být zobrazen v mapě. Nakonec bylo rozhodnuto, že se v mapě bude zobrazovat název kategorie. Důvod je ten, že textový popis se na telefonu špatně píše, ale kategorii uživatel

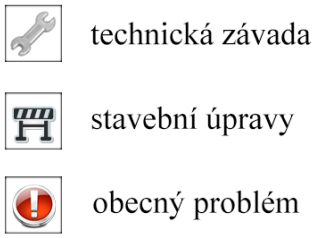
Obrázek 3.15: Přesunutí funkcionality tlačítka pro menu v Androidu 4.0+



Obrázek 3.16: Ovládání pomocí navigation drawer

vybere z předem připraveného seznamu. Mohlo by stát, že u řady hlášení by textový popis zcela chyběl.

Pro každou kategorii hlášení byla navržena samostatná ikona (obrázek 3.17), aby měl uživatel základní představu hned při pohledu na mapu. Po kliknutí na tuto ikonu se tedy zobrazí název kategorie. Pokud uživatel klikne na tento název, přenese se do detailu problému. Zde by se měla zobrazit fotografie, popis a datum, kdy byl problém nahlášen. V actionbaru by měl mít uživatel možnost zařadit problém mezi ty, kterým se chce při navigaci vyhnout, a možnost nahlásit problém jako neplatný.



Obrázek 3.17: Návrh ikon pro kategorie hlášení

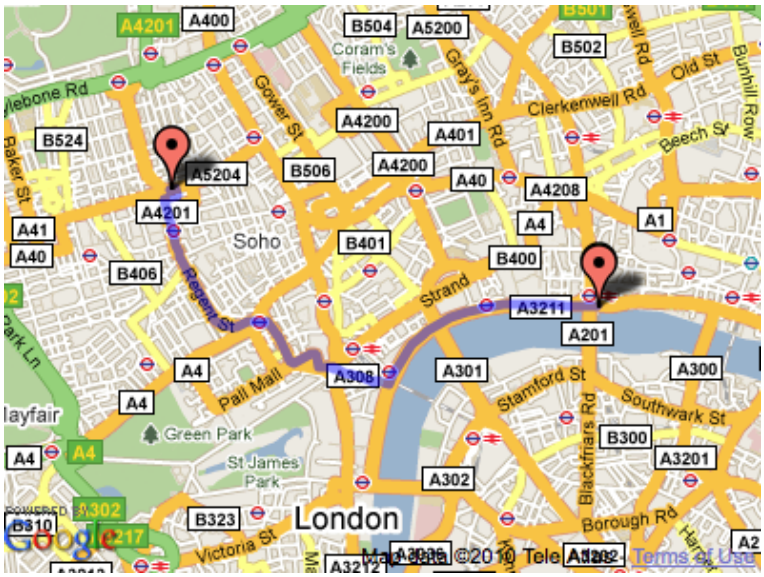
3.7.2 Google Android Maps API v2

Od doby vzniku původního prototypu WheelGo se v Androidu změnilo nejen menu, ale také Google vydal novou verzi Google Android Maps komponent [19]. Nová verze přináší pro programátora výrazné ulehčení.

V první verzi se používaly vrstvy, které se skládaly na mapu přes sebe. Takže WheelGo mělo jednu vrstvu pro problémy, další vrstvu pro místa a další vrstvu pro tipy. Vyvolat například dialog po kliknutí na značku v mapě byla programátorsky poměrně náročná úloha.

To se změnilo ve verzi 2. Přibyla podpora pro tyto dialogy, pro kreslení polygonů do mapy, podpora pro fragmenty, animace a celkově došlo k zjednodušení práce s mapou.

Pro WheelGo je důležitá možnost kreslit do mapy polygony, protože tím budeme uživateli prezentovat vyhledanou trasu. Komponentě mapy stačí předat seznam souřadnic v daném pořadí a barvu, jakou se má trasa vykreslit. Výstup pak vypadá jako na obrázku 3.18.



Obrázek 3.18: Vykreslení polygonu v komponentě Google mapy

Kapitola 4

Implementace

4.1 Transparentní REST service proxy

Aby se se službou snadno komunikovalo, byla vytvořena třída `WheelGoService`, která má metody odpovídající WheelGo Serveru.

```
public class WheelGoService {

    private String url;
    private Gson gson = new Gson();
    private GetRequest get ;
    private PostRequest post ;

    public WheelGoService(String serverIp) {
        this.url = "http://" + serverIp + "/WheelGoServer/rest/api/";
        this.get = new GetRequest(url);
        this.post = new PostRequest(url);
    }

    public Route navigate(NavigationParameters parameters) throws ConnectException {
        post.setJsonBody(gson.toJson(parameters));
        InputStream source = post.sendRequest("navigate");
        Reader reader = new InputStreamReader(source);
        return gson.fromJson(reader, Route.class);
    }

    public List<ProblemDto> getProblemsAround(double latitude, double longitude,
     double radius) throws ConnectException {...}

    public boolean addProblem(ProblemDto problem) throws ConnectException {...}

    public boolean deleteProblem(long problemId) throws ConnectException {...}
}
```

Při inicializaci se vytvoří objekt GetRequest, PostRequest a DeleteRequest. Tyto objekty reprezentují jednotlivé metody protokolu HTTP. Jako příklad bude popisována metoda POST, protože je o něco složitější než zbylé dvě.

Před odesláním požadavku se nejprve serializují parametry pomocí knihovny Gson a předám je třídě PostRequest. Voláním `post.sendRequest("navigate")` se inicializuje samotný požadavek na URL `/navigate`.

Protože hledání trasy může trvat dlouho, tak v metodě `sendRequest(String path)` musí být přenastavit timeout, jinak by bylo spojení po standardních 30-ti sekundách ukončeno.

```
DefaultHttpClient client = new DefaultHttpClient();
final HttpParams httpParameters = client.getParams();

HttpConnectionParams.setConnectionTimeout(httpParameters, 180 * 1000);
HttpConnectionParams.setSoTimeout        (httpParameters, 180 * 1000);
```

Pro zahájení spojení k serveru je použita třídu HttpPost, která je přímo součástí Androidu. V proměnné json jsou serializované parametry, které se předají do těla požadavku. Pomocí hlavičky Content-type se nastaví formát json a požadavek se odešle.

```
HttpPost postReqest = new HttpPost(url + path);
if (json != null) {
    postReqest.setEntity(new ByteArrayEntity(json.getBytes("UTF-8")));
}
postReqest.setHeader("Content-Type", "application/json; charset=utf-8");
HttpResponse getResponse = client.execute(postReqest);
```

Po vyhledání cesty aplikace přečte odpověď a vrátí ji třídě WheelGoService:

```
if (statusCode != HttpStatus.SC_OK) {
    HttpEntity getResponseEntity = getResponse.getEntity();
    return getResponseEntity.getContent();
}
```

WheelGoService potom odpoved' deserializuje:

```
Reader reader = new InputStreamReader(source);
return gson.fromJson(reader, Route.class);
```

Tímto způsobem je odstíněno volání služby od nutnosti starat se o HTTP spojení. Přidání nové metody obnáší rozšířit WheelGoService jen o pár řádků kódu.

Volání není asynchronní. To znamená, že nesmí běžet v hlavním vlákně aplikace, a proto je třeba použít AsyncTask. Příklad načítání problémů:

```
private class LoadProblemsTask extends AsyncTask<String, Void, List<ProblemDto>> {
   @Override
   protected List<ProblemDto> doInBackground(String... params) {
      WheelGoService port = new WheelGoService(params[0]);
      double latitude = Double.parseDouble(params[1]);
      double longitude = Double.parseDouble(params[2]);
      double radius = 1;
      try {
          return port.getProblemsAround(latitude, longitude, radius);
      } catch (Exception e) {
           return null;
      }
   }

   @Override
   protected void onPostExecute(List<ProblemDto> result) {
      loading.set(false);
      if (result == null) {
         Toast.makeText(getActivity(), "Nepodarilo se nacist problemy.",
               Toast.LENGTH_SHORT).show();
         return;
      } else {
         for (ProblemDto p : result) {
            problems.put(p.id, p);
         }
      }
      redrawMap();
   }
}
```

AsyncTasku je v parametrech předána IP serveru, kam se má připojit, souřadnice a rozsah, jak daleko mají být problémy kolem souřadnic zobrazeny. Tento parametr by měl odpovídat přiblížení mapy. Nicméně pro testovací účely je zvolena hodnota 1, což odpovídá území cca 150x150 km.

Výsledek volání je uložen do třídy LruCache [21], což je třída v Androidu vytvořená přímo pro udržování omezeného počtu objektů. To znamená, že pokud uživatel zobrazí 10 hlášení na určitém místě a pak přejede mapou jinam, kde načte dalších 10 hlášení, v cache nám zůstanou nová i původní hlášení, a pokud se vrátí na původní místo, jsou hlášení stále ještě načtena a je možné je okamžitě zobrazit.

```
LruCache<Long, ProblemDto> problems = new LruCache<Long, ProblemDto>(50);
```

4.2 Zpracování na pozadí pomocí Google AppEngine Tasks API

Hledání cesty se musí zpracovat na pozadí, protože může trvat delší dobu. K tomuto účelu bylo použito Tasks API. Nejprve je třeba získat odkaz na výchozí frontu úkolů a poté úkol vytvořit. Vytváří se tak v podstatě požadavek na volání servletu, takže je třeba specifikovat URL, HTTP metodu a parametry. Parametry navigace se předávají v serializované podobě. Tímto dojde k zařazení úkolu do fronty a AppEngine ho z ní později vyzvedne a zavolá.

```
Queue queue = QueueFactory.getDefaultQueue();
TaskOptions taskOptions = TaskOptions.Builder.withUrl("/findPath")
.param("latFrom", latFrom.toString())
.param("longFrom", longFrom.toString())
.param("latTo", latTo.toString())
.param("longTo", longTo.toString())
.param("gsonParameters", params).method(Method.POST);
queue.add(taskOptions);
```

V rámci servletu, který se zavolá, jsou nastaveny parametry a zavolá se metoda, která má za úkol hledání trasy.

```
@Produces({ MediaType.APPLICATION_JSON })
private void performPathFinding(HttpServletRequest req,
      HttpServletResponse resp) throws IOException {
   Gson gson = new Gson();
   Double latFromDouble = Double.parseDouble(req.getParameter("latFrom"));
   Double longFromDouble = Double.parseDouble(req.getParameter("longFrom"));
   Double latToDouble = Double.parseDouble(req.getParameter("latTo"));
   Double longToDouble = Double.parseDouble(req.getParameter("longTo"));

   NavigationParameters params = gson.fromJson(req.getParameter("gsonParameters"),
         NavigationParameters.class);
   List<NavigationNode> path = NavigationTask.navigate(latFromDouble,
         longFromDouble, latToDouble, longToDouble, params);
```

Když se trasa spočítá, vygeneruje se pro trasu unikátní taskid, pod kterým je uložen do AppEngine Datastoru, odkud si ho později WheelGo Server vyzvedne:

```
int id = NavigationTask.generateCode(latFromDouble, longFromDouble,
        latToDouble, longToDouble, params.locationsToAvoid);
Text gsonPath = new Text(gson.toJson(path));
result.setProperty("path", gsonPath);
result.setProperty("timestamp", new Date());
Util.persistEntity(result);
```

4.3 Parametrizace Dijkstrova algoritmu

Průběh hledání optimální trasy je ovlivněn metodou ocenění hran grafu, nad kterým se hledá nejkratší cestu. Ocenění hran má na starost metoda `calculateWeight`. Nejprve se vypočte geografickou vzdálenost uzlů pomocí Haversinovy rovnice [6].

```
public static float calculateWeight(Vertex vertex, Vertex oldVertex, Element edge,
       NavigationParameters params) {
   double earthRadius = 6369;
   double dLat = Math.toRadians(vertex.getLatitude()
               - oldVertex.getLatitude());
   double dLng = Math.toRadians(vertex.getLongitude()
               - oldVertex.getLongitude());
   double a = Math.sin(dLat / 2) * Math.sin(dLat / 2)
            + Math.cos(Math.toRadians(oldVertex.getLatitude()))
            * Math.cos(Math.toRadians(vertex.getLatitude()))
            * Math.sin(dLng / 2) * Math.sin(dLng / 2);
   double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
   double dist = earthRadius * c;
   ...
```

Poté se vyjádří vliv sklonu podle tabulky 3.3. A výsledná cenu vypočte jako:

```
float price = dist * incline;
```

Ve vzorci chybí zohlednění problémů, kterým se chce uživatel vyhnout. To je z důvodu, že hrany blízko těchto problému se filtrují už při parsování výstupu z OpenStreetMap, takže v grafu vůbec nefigurují.

Kapitola 5

Testování

5.1 Testování Dijkstrova algoritmu

Jako zdroj pro implementaci Dijkstrova algoritmu byla použita verze od Lard Vogela [31], jejíž součástí je unit test pro vyhledání nejkratší trasy v předpřipraveném grafu. Algoritmus byl použit nejprve na hledání geograficky nejkratší trasy, kdy byl aplikován přímo na graf vytvořený z mapových podkladů. Souřadnice cesty vypsal server na konzoli a tento výstup byl pak ověřen pomocí webové aplikace [5], která znázornila cestu na mapě (viz obrázek 5.1).

Při testování rychlosti hledání trasy se ukázalo, že načítání dat ze serveru trvá příliš dlouho. Na základě toho byla komponenta WheelGo Navigator upravena tak, aby používala OverPass API, které je rychlejší. Výstup OverPass API má stejný formát jako původně používané API, takže bylo třeba změnit pouze tvar a cíl požadavku. Výhodou také bylo, že API umožňuje filtrovat výstup rovnou na serveru, takže bylo možné ve výstupu vynechat relace, což mělo kladný vliv na celkovou dobu zpracování výstupu z OSM. Zrychlení bylo přibližně desetinásobné.

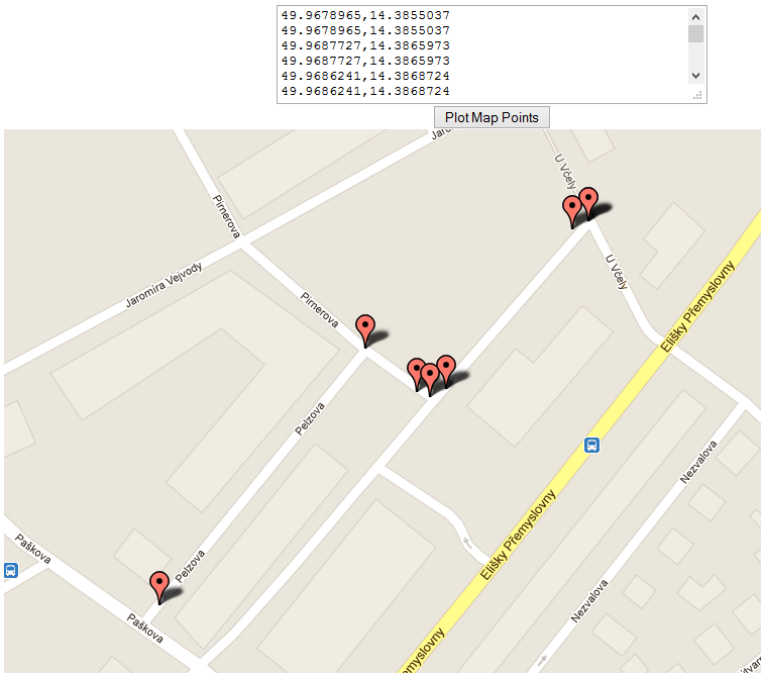
5.2 Testování aplikace s vozíčkáři

Hlavním úkolem testování s uživateli bylo ověřit použitelnost nového designu aplikace pro mobilní zařízení jak doma, tak v terénu. Zásadní změnou oproti původnímu designu WheelGo je nový pohyb při otevírání postranního menu. Pro testování byla použita následující zařízení:

- **Nexus 7** - tablet s úhlopříčkou displeje 7 palců
- **HTC Desire X** - mobilní telefon s úhlopříčkou displeje 4 palce

Tablet byl použit pro základní seznámení se s aplikací a testování doma. Telefon byl pak použit v terénu.

Testu předcházela příprava, která byla náročná hlavně kvůli absenci informací o sklonu v mapových podkladech OSM. Protože aplikace čte data z oficiálního serveru OSM, bylo třeba dodat informaci o sklonu ručně. Na základě ID cesty a odhadu sklonu byla tato data nasimulována v WheelGo Navigatoru při parsování výstupu OSM API.



Obrázek 5.1: Zobrazení trasy v mapě

5.2.0.1 Otázky před testem

Samotnému testu aplikace předcházela série otázek. První otázka se týkala věku postiženého, protože mladší generace má zpravidla kladnější vztah k technice. V souvislosti s tím jsem se zeptal i na zkušenost s PC a chytrými telefony.

Další otázky se týkaly zdravotního stavu. Konkrétně na rozsah postižení horních končetin, případně další související postižení. Podstatná byla informace, zda jezdí sám nebo s asistentem a zda by nechal asistenta používat mobilní zařízení s WheelGo.

Následovaly otázky, které zjistit, jakým způsobem uživatel normálně cestuje. Jaký používá vozík, jakou si dává časovou rezervu, jak trasu plánuje, zda jezdí na neznámá místa a jak se na takovou cestu připravuje.

Kromě těchto otázek jsem zaznamenal i pohlaví uživatele a jaké bylo počasí.

5.2.0.2 Představení aplikace

V další části testování jsem vozíčkáři aplikaci ukázal na tabletu a vysvětlil mu na příkladech způsob fungování systému a jaké problémy se snaží řešit. Byly mu vysvětleny následující úkony, které pak sám zopakoval.

- ovládání mapy (posun, vrácení se do výchozího stavu)
- nahlášení překážky
- vyhledání nejkratší trasy

- parametrizace trasy – specifikace překážek, kterým se chce vyhnout

5.2.0.3 Test v terénu

Na test v terénu používal vozíčkář mobilní telefon HTC Desire X s menším displejem v porovnání s tabletem, na kterém se s aplikací seznamoval.

Trasa, kterou vozíčkář měl absolvovat začínala v Petýrkově ulici a vedla k obchodnímu centru Chodov. Existují v zásadě dva způsoby, jak se do obchodního centra dostat. Obě dvě cesty jsou pro vozíčkáře sjízdné, ale na jedné (cesta dole na obrázku 5.2) je nepříjemný sráz přímo do vozovky. U přechodu není rovina, na které by mohl vozíčkář pohodlně zastavit a rozhlédnout se, zda se k přechodu neblíží žádný automobil. Tento sráz měl vozíčkář za úkol nahlásit jako překážku, vybrat, že se jí chce vyhnout, a trasu přeplánovat.

Obrázek 5.2: Trasa testování

5.2.0.4 Otázky po testu

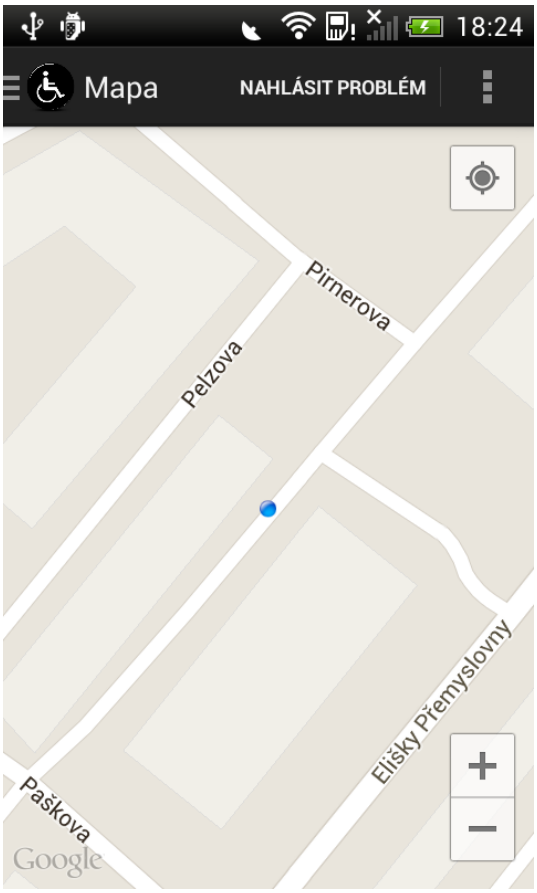
Nakonec byl vozíčkář dotázán, zda se mu aplikace ovládala pohodlně, co by změnil, zda mu něco vyloženě vadilo a co se mu naopak líbilo. Důležitá otázka byla také, zda by aplikaci používal a zda si umí představit, že by mu byla v životě užitečná.

5.2.1 Pilotní test

První zkušební test se prováděl s kvadruplegikem, kde se dají očekávat největší problémy při ovládání aplikace. Uživatel mohl používat omezeně jen jednu ruku. Aplikace se testovala

pouze v domácím prostředí, protože v terénu by ji v tomto případě měl ovládat asistent. Cílem bylo ověřit, zda si uživatel může trasu naplánovat sám bez pomoci.

V aplikaci WheelGo bylo použito rozhraní systému Android podle Android Guidelines. Hlavní rozdíl spočíval v použití action baru a originálních ovládacích prvků mapy Google Map. Na obrázku 5.3 je vidět úvodní obrazovka s mapou.



Obrázek 5.3: Prvotní verze uživatelského rozhraní

Už při testování na tabletu se ukázalo, že ovládací prvky mapy jsou velice malé, a hlavně, že tlačítka pro přiblížení jsou moc blízko sebe. Uživatel si stěžoval, že neví, které mačká.

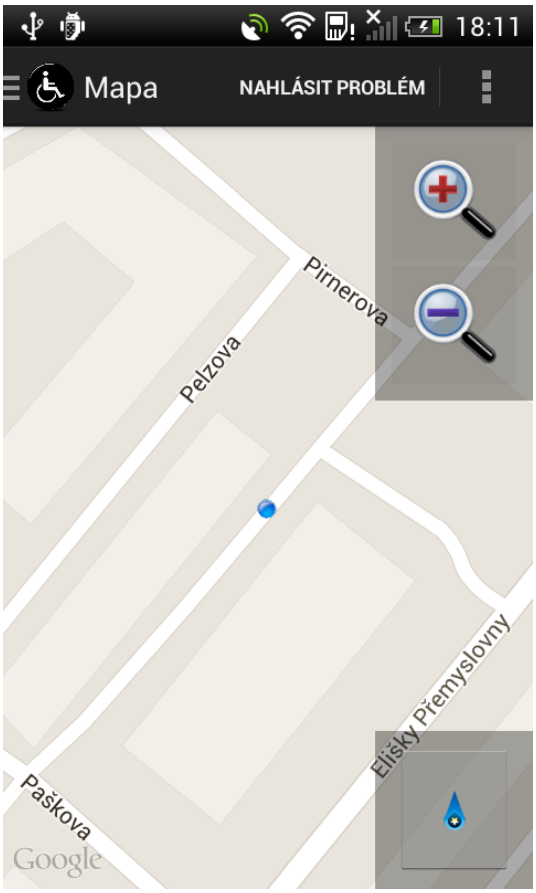
To bylo dáno nejen velikostí tlačítek, ale také tím, že uživatel neviděl přes své ruce na displej. Pomohlo zařízení naklonit. V tomto případě je tedy nutnost použít držák na mobilní zařízení.

Tlačítko v action baru pro hlášení problému uživatel stiskl bez problémů.

V rámci pilotního testu bylo také ověřeno, zda je pro uživatele snadné otevřít vysouvací menu. Tento pohyb nakonec uživateli nečinil žádné problémy, ale bylo třeba vysvětlení, že má táhnout prstem od kraje obrazovky až na druhý. Ve skutečnosti toto třeba není, ale uživateli se stávalo, že postranní panel omylem zatáhl zpět. Pohybem až na druhý konec obrazovky se tomu zamezilo.

Samotné plánování trasy se také podařilo, ale uživateli dělalo problémy podržení prstu v mapě. Bylo třeba klikat do mapy prstem kolmo k displeji, což bylo pro uživatele obtížné, protože se na tuto akci musel soustředit.

Na základě výsledků pilotního testu jsem se rozhodl použít uživatelské rozhraní vyvinuté v rámci mé bakalářské práce s mírnými změnami. Upravená úvodní obrazovka je vidět na obrázku 5.4. Oproti původní verzi WheelGo (viz obrázek 2.7) bylo odstraněno tlačítko pro hlášení problému, protože je nyní přístupné z action baru. A tlačítko pro přesun na aktuální polohu bylo přemístěno doprava dolu a byla mu nastavena průhlednost.



Obrázek 5.4: Uživatelské rozhraní po úpravě

5.2.2 Výsledky testování

5.2.2.1 Vozíčkář 1

Pohlaví: muž

Věk: 53 let

Počasí: oblačno, cca 10 °C

Druh postižení: kvadruplegik, úroveň postižení III

Zkušenost s mobilními telefony nebo PC: používá klasický mobilní telefon, PC nikoliv

Rozsah postižení horních končetin: jedna ruka částečně funkční

Další postižení: žádné

Využívá asistenta? Cca 20-30 %

Druh vozíku: mechanický

Kam a jak zpravidla cestuje Zpravidla jezdí pravidelné trasy. Nemá čas jezdit na nová místa, ačkoliv by rád. Časovou rezervu si dělá v řádu desítek minut.

Jak zjišťuje přístupnost míst, která nezná Řídí se tím, zda na místo vede nízkopodlažní MHD. Mimo město si telefonicky ověřuje hlavně bezbariérovost ubytování a na samotné cestě žádá ostatní o pomoc.

Představení aplikace Vozíčkář 1 již testoval verzi aplikace z mé bakalářské práce a byl s ní tedy částečně seznámen. Na začátek mu bylo znovu vysvětleno fungování a principy WheelGo. Z předchozího testování bylo známo, že uživateli vyhovuje satelitní zobrazení, které bylo používáno i při tomto testu. Na tabletu se mu aplikace ovládala bez problémů. Problematická byla samotná orientace v mapě. Uživatel potřeboval záchytné body, které zná. Nutností bylo používat dioptrické brýle, protože písmo v komponentě Google Maps je poměrně malé.

Po ovládání mapy měl uživatel za úkol nahlásit imaginární překážku. Problematické bylo psaní na klávesnici tabletu. Protože uživatel nepoužívá PC, činilo mu problémy orientovat se v rozložení klávesnice. Textový popis však není u hlášení povinný.

Dále měl uživatel za úkol vyhledat trasu k obchodnímu centru Chodov. Dlouhé podržení prstem do mapy mu nedělalo žádné problémy.

Po vyhledání trasy měl označit předtím hlášenou imaginární překážku, že se jí chce vyhnout. Po připomenutí, že tato volba se nachází v detailu hlášení se mu to podařilo.

Test v terénu Používání mobilního telefonu bylo pro uživatele nepohodlné kvůli jeho větším prstům. Navíc musel mít mobilní telefon na klíně, což znesnadňovalo jeho použití. Na použití jednou rukou, na které je zvyklý, byl telefon moc velký.

Překážku se podařilo nahlásit, ale bez fotografie, protože uživatel jednou rukou nemohl zároveň namířit mobilní telefon na problematické místo a stisknout tlačítko pro spoušť pro vyfotografování.

Označení překážky, že se jí chce vyhnout se podařilo opět po připomenutí, kde se tato volba nachází. Nakonec trasu úspěšně přeplánoval. Orientace v mapě činila uživateli velké problémy a bylo třeba mu dát dost času.

Otázky po testu Uživateli vadilo malé písmo v mapě a velikost telefonu. Aplikaci by používal pouze doma na tabletu. Líbil se mu nápad, ale měl obavy, že by tam lidé nevkládali žádná hlášení.

5.2.2.2 Vozíčkář 2

Pohlaví: žena

Věk: 50 let

Počasí: –

Druh postižení: kvadruplegik, úroveň postižení III

Zkušenost s mobilními telefony nebo PC: používá pouze pevnou linku – telefon s hlasitým reproduktorem

Rozsah postižení horních končetin: obě ruce ochrnuté, může k ovládání telefonu používat pouze jeden prst ruky

Další postižení: žádné

Využívá asistenta? vždy

Druh vozíku: mechanický, vždy potřebuje asistenta

Kam a jak zpravidla cestuje Jezdí pravidelné trasy. Ráda jeden i někam, kde to nezná, ale spoléhá se při tom na asistenta.

Jak zjišťuje přístupnost míst, která nezná Přístupnost zjišťuje telefonicky nebo často dostává tip od známých vozíčkářů, kteří na místě už byli.

Představení aplikace Aplikace byla uživatelce představena už při pilotním testu 5.2.0.4. Uživatelka ocenila větší tlačítka a mapa se jí ovládala daleko snáz. Problematická byla však orientace v mapě, protože mapu ve svém životě téměř nikdy nepoužívala. Trochu pomohlo satelitní zobrazení.

Už při pilotním testu jsme zjistili, že uživatelka potřebuje držák pro zařízení, jinak nevidí, kam přesně kliká. Také se ukázalo, že je pro ní problematické klikat na displej, protože musí mít prst kolmo k obrazovce a musí se na toto zvlášť soustředit. Uvedla, že při stresovém dni by jí toto dělalo potíže. Celkem dobře se jí do displeje klikalo, ale držení prstu jí činilo problémy. Některé pohyby nedělá vědomě a často místo kliknutí se displeje dotkla víckrát, což vyvolalo jinou akci.

Naopak se jí snadno vytahovala postranní lišta a uvedla, že po chvíli tréninku by jí tento způsob vyhovoval.

Všechny úkoly v testu zvládla bez problémů. Vzhledem k postižení jí nešlo vůbec psát na dotykové klávesnici. Navigace vyžadovala delší podržení prstu do mapy a bylo třeba několika pokusů, aby se nabídka navigace vyvolala.

Test v terénu Bohužel uživatelka neměla k dispozici asistenta, takže nebylo možné otestovat aplikaci v terénu. Dle jejích slov by zařízení většině asistentům půjčila a hlásila by překážky. Ona sama by ho používala pouze doma.

Otázky po testu Aplikace se jí líbila, ale vadilo jí ovládání dotykového displeje. Uvítala by, kdyby se z aplikace daly hlášení delegovat příslušným orgánům (například rozbitý chodník na Technickou správu komunikací).

5.2.2.3 Vozíčkář 3

Pohlaví: muž

Věk: 30 let

Počasí: oblačno, cca 10 °C

Druh postižení: ochrnutá pravá strana těla

Zkušenost s mobilními telefony nebo PC: Pracuje na PC. Používá chytrý mobilní telefon.

Rozsah postižení horních končetin: možnost používat pouze jednu ruku

Další postižení: porucha zraku, postižení paměti a kognitivních funkcí

Využívá asistenta? vždy

Druh vozíku: elektrický

Kam a jak zpravidla cestuje? Často necestuje a pokud ano, tak vždy pravidelné trasy. Trasu si plánuje na internetu, ale potřebuje s tím pomoc. Nedokáže celou trasu poskládat do jednoho celku. Dělá si velké časové rezervy.

Jak zjišťuje přístupnost míst, která nezná? přes internet

Představení aplikace Uživatel sám od sebe začal aplikaci prozkoumávat a ovládání mu nedělalo sebemenší problémy. Tablet musel mít položený na stole. Všechny úkoly zvládl bez problému, prozkoumal dokonce nastavení aplikace, kam se ostatní uživatelé nedostali (nebylo požadováno a ani předvedeno).

Test v terénu Na rozdíl od tabletu se mu telefon ovládal lépe. Držel ho a zároveň ovládal jednou rukou. Překážku nahlásil i s pořízením fotografie a textovým popisem. Ovládání v terénu mu nedělalo žádné problémy.

Otázky po testu Aplikace a celkový nápad se mu velice líbil, ale měl obavu, že by v mapě nebyly postiženy všechny možné komplikace, protože není dostatek uživatelů, kteří by hlášení zadávali. Ocenil by verzi pro webový prohlížeč, která by se synchronizovala do telefonu, protože se mu ovládá snáze myš u PC než tablet.

5.2.3 Závěr testování

Během testování se ukázalo, že nové rozhraní není pro uživatele překážkou. U mapy není vhodné použít ovládací prvky, které poskytuje Google, protože pro uživatele s motorickými problémy jsou příliš malé. Kladně byla přijata výsuvná lišta, jejíž používání žádnému z vozíčkářů nedělalo problémy.

Občas nebylo jasné, jak nastavit, aby se trasa plánovala mimo určitou překážku. Tato volba je přístupná v detailu hlášení. Standardně se uživatel do detailu musí nejdříve podívat, aby zhodnotil, zda je pro něj překážka závažná, a v tu chvíli se rozhoduje, zda se jí vyhne nebo ne. Tím, že vozíčkář v testu hlásil překážku sám, neměl pak motivaci se podívat do detailu hlášení a proto nebylo zřejmé, kde se tato funkce nachází.

Kapitola 6

Závěr

Úkolem této práce bylo analyzovat problematiku plánování tras pro vozíčkáře a vytvořit systém, který doplňuje existující projekt WheelGo o možnost navigace. V souvislosti s tím bylo třeba přepracovat původní design aplikace pro mobilní telefony. Vzhledem k tomu, že systém Android od vzniku WheelGo prošel výraznými změnami, bylo třeba nahradit staré ovládací prvky a aktualizovat uživatelské rozhraní pro nové verze systému.

Během této práce byla vytvořena také serverová část, která umožňuje ostré nasazení aplikace. K tomu bylo využito cloudových technologií, aby bylo možné navigační úlohu škálovat. Systém byl úspěšně nasazen do RedHat OpenShift a Google AppEngine a mobilní aplikace publikována na Google Play.

- **Google Play** – klientská Android aplikace
  <https://play.google.com/store/apps/details?id=cz.nuc.wheelgo.androidclient>
- **RedHat OpenShift** – REST služba zpracovávající požadavky z mobilního zařízení
  <http://wheelgo-nuc.rhcloud.com/WheelGoServer/rest/api/hello>
- **Google AppEngine** – servlet zpracovávající požadavky na navigaci
  <http://1.wheelgo-navigator.appspot.com/api/hello>

K spolehlivé navigaci bohužel chybí data v mapových podkladech. Hlavně popis sklonu vozovky a povrchu. V použitých OpenStreetMap není zatím vhodný způsob, jak takovéto informace do mapy zanést. Například atribut vyjadřující sklon vozovky je atributem pro celou cestu (tzn. prakticky pro celou ulici), ale není možnost zaznamenat sklon pro jednotlivé její úseky.

Jistou nadějí by mohl být Evropský parlament schválil 13. června 2013 aktualizaci evropské směrnice [11], která ukládá členským státům, aby subjekty jejich veřejného sektoru poskytovaly dokumenty ve strojově čitelném formátu veřejnosti.

Projekt WheelGo byl prezentován 5. září 2013 na jednání Rady hl. m. Prahy pro Prahu bezbariérovou a otevřenou. Z jednání bylo patrné, že Praha v tomto směru chce spolupracovat, a že by mohlo v budoucnu dojít k zpřístupnění detailních mapových podkladů, na základě kterých by se dalo vozíčkáře navigovat.

Další vývoj projektu závisí nejen na kvalitních mapových podkladech, ale také na aktivizaci případných uživatelů. Ze schůzky s autory projektu VozejkMap, která se uskutečnila 28. listopadu 2013 vyplynulo, že komunita vozíčkářů je v jejich projektu velice pasivní. Z toho důvodu by mohl být do systému zanesen například systém ocenění, kde by si uživatele navzájem děkovali a autor hlášení by tak přímo věděl, že někomu pomohl.

Při testování se ukázalo, že i samotní vozíčkáři nevěří, že by do systému uživatelé vkládali data. Problematické je použití samotného mobilního telefonu, které je v první řadě nepohodlné a hlavně kvůli němu vozíčkář musí zastavit, což pro něj znamená zdržení a komplikaci. Možné řešení je použití Google Glass [18], které by měl uživatel neustále na sobě a mohl je ovládat hlasem. Finanční náročnost takového zařízení je však nesrovnatelně vyšší.

Literatura

[1] asraf344. *Android Web Service Client* [online]. 2012. Dostupné z: <http://code.google.com/p/android-ws-client/>.

[2] BHASKAR PRASAD RIMAL, I. L. E. C. *A Taxonomy, Survey, and Issues of Cloud Computing Ecosystems*. 1. Oval Road, London, UK : Springer London, 4th edition, 2010.

[3] Bubble Software Solutions ltd . *WSDL to Java* [online]. 2013. Dostupné z: <http://www.wsdl2code.com/pages/home.aspx>.

[4] CZEPRA. *VozejkMap Open Data* [online]. 2013. Dostupné z: <http://www.vozejkmap.cz/opendata/>.

[5] Darrin Ward. *Plot Lat/Long Points on Map by Coordinates* [online]. 2013. Dostupné z: <http://www.darrinward.com/lat-long/>.

[6] Doctor Rick. *Deriving the Haversine formula* [online]. 1999. Dostupné z: <http://mathforum.org/library/drmath/view/51879.html>.

[7] Dopravní podnik hl. m. Prahy. *Bezbariérové cestování v autobusech Dopravního podniku hl. m. Prahy* [online]. 2013. Dostupné z: <http://www.dpp.cz/bezbarierove-cestovani/autobusy/>.

[8] Dopravní podnik hl. m. Prahy. *Stav bezbariérových zařízení* [online]. 2013. Dostupné z: <http://www.dpp.cz/stav-bezbarierovych-zarizeni/>.

[9] FEDOROVá, L. Bakalárska práca Vyhladávanie najkratších ciest nad dátami z OpenStreetMap, 2009.

[10] FIELDING, R. T. Architectural Styles and the Design of Network-based Software Architectures, 2000.

[11] Fond Otakara Motejla. *Datová revoluce klepe na dveře* [online]. 2013. Dostupné z: <http://www.otevrenadata.cz/datova-revoluce-klepe-na-dvere/>.

[12] Geert Jansen. *Rest methods* [online]. 2013. Dostupné z: <https://restful-api-design.readthedocs.org/en/latest/methods.html>.

[13] Geoinformatics Research Group. *GIScience / Geoinformatics Research Group* [online]. 2013. Dostupné z: <http://www.geog.uni-heidelberg.de/gis/index_en.html>.

[14] Google. *Action Bar* [online]. 2013. Dostupné z: <http://developer.android.com/design/patterns/actionbar.html>.

[15] Google. *Google Directions API* [online]. 2013. Dostupné z: <http://developer.android.com/design/index.html>.

[16] Google. *Google App Engine* [online]. 2013. Dostupné z: <https://developers.google.com/appengine/>.

[17] Google. *Using the Google Plugin for Eclipse* [online]. 2013. Dostupné z: <https://developers.google.com/appengine/docs/java/tools/eclipse>.

[18] Google. *Google Glass - what it does* [online]. 2013. Dostupné z: <http://www.google.com/glass/start/what-it-does/>.

[19] Google. *Google Maps Android API v2* [online]. 2013. Dostupné z: <https://developers.google.com/maps/documentation/android/>.

[20] Google. *Google Directions API* [online]. 2013. Dostupné z: <https://developers.google.com/maps/documentation/directions/>.

[21] Google. *LruCache* [online]. 2013. Dostupné z: <http://developer.android.com/reference/android/util/LruCache.html>.

[22] Google. *Navigation Drawer* [online]. 2013. Dostupné z: <http://developer.android.com/design/patterns/navigation-drawer.html>.

[23] Komunita OSM. *Historie OSM* [online]. 2013. Dostupné z: <http://wiki.openstreetmap.org/wiki/History_of_OpenStreetMap>.

[24] Komunita OSM. *Editor iD* [online]. 2013. Dostupné z: <http://wiki.openstreetmap.org/wiki/ID>.

[25] Komunita OSM. *Overpass API* [online]. 2013. Dostupné z: <https://wiki.openstreetmap.org/wiki/Overpass_API>.

[26] Komunita OSM. *Open Street Maps primitives* [online]. 2013. Dostupné z: <http://wiki.openstreetmap.org/wiki/Data_Primitives>.

[27] Komunita OSM. *Sidewalk tag* [online]. 2013. Dostupné z: <http://wiki.openstreetmap.org/wiki/Key:sidewalk>.

[28] Komunita OSM. *Wheelchair routing* [online]. 2013. Dostupné z: <http://wiki.openstreetmap.org/wiki/DE:Wheelchair_routing>.

[29] Komunita OSM. *WheelMap* [online]. 2013. Dostupné z: <http://wiki.openstreetmap.org/wiki/Wheelmap>.

[30] Komunita OSM. *Potlatch 2* [online]. 2013. Dostupné z: <http://wiki.openstreetmap.org/wiki/Potlatch>.

[31] Lars Vogel. *Dijkstra's shortest path algorithm in Java* [online]. 2009. Dostupné z: <http://www.vogella.com/articles/JavaAlgorithmsDijkstra/article.html>.

[32] Martin Nuc. *Patch for /trunk/AndroidWSProxyClient/src-ws/hu/javaforum/commons/ReflectionHelper.java* [online]. 2013. Dostupné z: <http://code.google.com/p/android-ws-client/issues/detail?id=27>.

[33] Microsoft. *Routes API* [online]. 2013. Dostupné z: <http://msdn.microsoft.com/en-us/library/ff701705.aspx>.

[34] NUC, M. *Navigace pro vozíčkáře*. 1. Karlovo nám. 13, 121 35 Praha 2 : ČVUT Fakulta elektrotechnická, 1th edition, 2001.

[35] RAFAEL RODRíGUEZ-PUENTE, M. S. L.-C. Algorithm for shortest path search in Geographic Information Systems by using reduced graphs. *SpringerPlus 2:291*. 2013, 2, 1.

[36] Redakce portálu hl. m. Prahy. *Jak se jezdí vozíčkářům v Praze* [online]. 2008. Dostupné z: <http://www.praha.eu/jnp/cz/home/zivot_v_praze/praha_bezbarierova/lide_s_telesnym_postizenim/jak_se_jezdi_vozickarum_v_praze$5465-export.html?aid=173975>.

[37] RedHat. *OpenShift Online Pricing* [online]. 2013. Dostupné z: <https://www.openshift.com/products/pricing>.

[38] RedHat. *RESTEasy framework* [online]. 2013. Dostupné z: <http://www.jboss.org/resteasy>.

[39] SOZIALHELDEN e.V. *Wheelmap.org* [online]. 2013. Dostupné z: <http://sozialhelden.de/category/projekte/wheelmap-org/>.

[40] The Apache Software Foundation. *WSDL to Java* [online]. 2013. Dostupné z: <http://cxf.apache.org/docs/wsdl-to-java.html>.

[41] Virgil Dobjanschi. *Developing Android REST client applications* [online]. 2010. Dostupné z: <http://www.google.com/events/io/2010/sessions/developing-RESTful-android-apps.html>.

[42] www.zawatzky.de. *Automobil pro vozíčkáře* [online]. 2011. Dostupné z: <http://www.zawatzky.de/fahrzeugumbauten/selbst-auto-fahren-im-rollstuhl/caddy-aktiv/index.html>.

[43] www.zawatzky.de. *Připevnění iPadu na vozík* [online]. 2013. Dostupné z: <http://www.rammount.com/NewProducts/wheelchairipadmounts/tabid/3968/Default.aspx>.

Příloha A

Seznam použitých zkratek

OSM OpenStreetMap

API Application Programming Interface

HTTP Hypertext Transfer Protocol

REST Representational State Transfer

XML Extensible Markup Language

SQL Structured Query Language

SOAP Simple Object Access Protocol

WSDL Web Service Definition Language

JAXB Java Architecture for XML Binding

CRUD create, read, update, delete

JSON JavaScript Object Notation

URL Uniform Resource Locator

GPS Global Positioning System

ID Identifikátor

Příloha B

Instalační příručka

B.1 Instalace aplikace pro mobilní telefon

Pro instalaci klientské aplikace je třeba mít přistup na Google Play. Tomu je třeba mít registrovaný účet u Google. K instalaci aplikace stačí z mobilního telefonu navštívit URL:

<https://play.google.com/store/apps/details?id=cz.nuc.wheelgo.androidclient>

Po instalaci aplikace do telefonu je přednastaven server `wheelgo-nuc.rhcloud.com`. V případě použití jiného serveru je třeba změnit URL pomocí menu aplikace – Nastavení – Server IP.

B.2 Instalace WheelGo Server

Pro zprovoznění serveru na jakémkoliv JBoss Serveru je třeba vytvořit nejprve PostgreSQL data source s identifikátorem `java:jboss/datasources/PostgreSQLDS`. Poté je možné deploynout `WheelGoServer.war` nahráním do adresare `/jboss/standalone/deployments`.

V případě nasazení do OpenShift probíhá deploy commitnutím souboru `WheelGoServer.war` do adresáře `deployments` v GIT repozitáři.

B.3 Instalace WheelGo Navigator

K nasazení komponenty WheelGo Navigator se používá plugin do Eclipse podle návodu v [17]. Plugin se postará o nahrání komponenty do AppEngine. Potřebný je pouze účet Google.

Příloha C

Obsah přiloženého DVD

- WheelGo-MartinNuc.pdf – text diplomové práce v formátu PDF
- WheelGoAndroid – projekt klientské aplikace pro Android využívající vývojové prostředí Android Studio
- WheelGoNavigator – projekt komponenty WheelGo Navigator pro Eclipse s pluginem pro AppEngine
- WheelGoServer – projekt komponenty WheelGo Server pro IntelliJ Ultimate