VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

SYSTÉM PRO ZPRACOVÁNÍ GPS DAT Z VÍCE MOBILNÍCH ZAŘÍZENÍ
SYSTEM FOR PROCESSING OF GPS DATA FROM MULTIPLE MOBILE DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Josef Jurča

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Ing. Ivo Lattenberg, Ph.D.

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Josef Jurča ***ID:*** 119867
Ročník: 2 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Systém pro zpracování GPS dat z více mobilních zařízení

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte systém, který bude přenášet data o poloze z mobilních telefonů na server , kde budou tato data ukládána do databáze a dále zpracovávána. Vytvořte aplikaci pro mobilní zařízení s OS Android, která bude odesílat údaje o své poloze na server a bude také zobrazovat polohu stanic ve skupině. Bude možno také zpřesnit polohu stanice pomocí DGPS za předpokladu, že jedna stanice ve skupině bude pevně na svém místě. Vytvořte dále webovou aplikaci, která bude zpracovávat a prezentovat data s informacemi o polohách ze serveru, historii polohy a statistických údajích. Proveďte sadu měření a vyhodnoťte, jak se projeví zpřesňování polohy pomocí DGPS.

DOPORUČENÁ LITERATURA:

[1] MURPHY, M. L. Android 2 Průvodce programováním mobilních aplikací, Nakladatelství Computer Press, a.s. 2011, 376 stran, ISBN 978-80-251-3194-7

[2] WATSON, B., C# 4.0 - řešení praktických programátorských úloh, Zoner Press, 2010, 656 s., ISBN 978-8--7413-094-6

[3] VIRIUS, M., C# 2010 Hotová řešení, Computer Press, 2012, 424 s., ISBN 978-80-251-3730-7

Termín zadání: 1.2.2016 ***Termín odevzdání:*** 25.5.2016

Vedoucí práce: doc. Ing. Ivo Lattenberg, Ph.D.
Konzultant diplomové práce:

doc. Ing. Jiří Mišurec, CSc., *předseda oborové rady*

ANOTACE

Cílem diplomové práce bylo vytvořit informační systém skládající se z mobilní aplikace, serveru a webové aplikace. Mobilní aplikace byla navržena pro operační systém Android a slouží k odesílání GPS souřadnic mobilních zařízení na server. Tato data jsou následně na serveru zpracována a uložena do databáze. Webová aplikace zobrazuje aktuální polohu mobilních stanic ve skupině na mapě, dále zobrazuje historii polohy a statistické údaje. Pozici stanice je možné zpřesnit pomocí DGPS za předpokladu, že jedna ze stanic ve skupině je pevně na svém místě. Systém byl navržen a následně implementován s využitím moderních webových technologií a frameworků PhoneGap, Ionic, Node.js, Express, MongoDB, Mongoose a AngularJS. Dále byla provedena měření, která mají prezentovat vliv DGPS na přesnost určení polohy.

Klíčová slova: GPS, DGPS, lokalizace, Android, JavaScript, PhoneGap, Ionic, AngularJS, MongoDB, Node.js, REST

ABSTRACT

The aim of this thesis was to create an information system which consists of a mobile application, server and web application. Mobile application is designed for operating system Android and it is used for sending GPS coordinates of mobile devices to the server. These data are later processed by server and stored in database. Web application displays the current location of mobile devices in user group on a map, it also displays location history and statistics. Accuracy of the station location can be enhanced by using DGPS assuming that one of stations in a group does not change its location. The system was designed and then developed using modern web technologies and frameworks including PhoneGap, Ionic, Node.js, Express, MongoDB, Mongoose and AngularJS. Further measurements, that were performed, are supposed to present the effect of DGPS to location accuracy determination.

Keywords: GPS, DGPS, localization, Android, JavaScript, PhoneGap, Ionic, AngularJS, MongoDB, Node.js, REST

JURČA, J. *Systém pro zpracování GPS dat z více mobilních zařízení*. Brno: FEKT VUT v Brně, 2016. 57 stran. Vedoucí diplomové práce doc. Ing. Ivo Lattenberg, Ph.D.

Prohlášení

Prohlašuji, že svou diplomovou práci na téma „Systém pro zpracování GPS dat z více mobilních zařízení“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne …… .....

podpis autora

Poděkování

Děkuji vedoucímu práce panu doc. Ing. Ivovi Lattenbergovi, Ph.D. za velmi užitečnou metodickou pomoc a cenné rady při zpracování diplomové práce.

V Brně dne ……. .....

podpis autora

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Výzkum popsaný v této diplomové práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

Obsah

Seznam obrázků

1. Úvod

V dnešní době je téměř každé mobilní zařízení, ať už je to mobilní telefon nebo tablet, vybaveno GPS přijímačem. Stejně tak je většina těchto zařízení připojena k internetu. Tohoto faktu lze využít a v aktuálním čase sledovat polohy mobilních zařízení a tím pádem i jejich uživatelů, tyto polohy zpracovávat a prezentovat. Informační systém vhodně pracující s těmito možnostmi může mít široké spektrum užití. Mohl by být použit například ke sledování firemních vozidel, vozidel nákladní dopravy a nebo může sloužit například k tomu, aby rodiče vždy věděli o poloze svých dětí. Dále by podobný systém mohl sloužit jako sociální síť, kde by všichni uživatelé mohli vzájemně sdílet svoji polohu a mohli by tak mít přehled, kde se nachází a jednodušeji se pak třeba vzájemně scházet.

Cílem této práce bude navrhnout a následně implementovat informační systém, který se bude skládat z mobilní aplikace odesílající svoji polohu na vzdálený server a zobrazovat polohy ostatních zařízení, ze serveru, který bude tato data zpracovávat a z webového portálu, který bude data vhodně prezentovat uživateli. Uživatelé se budou moci do systému registrovat, přihlásit a každý z nich bude mít možnost zvolit, kteří jiní uživatelé mohou sledovat jeho pozici. V systému bude implementována technologie diferenciální GPS sloužící ke zpřesnění lokalizace.

Tato technická zpráva bude rozdělena do jedenácti kapitol. Kapitola 2 se bude zabývat systémem GPS, principem jeho fungování a jeho strukturou, přesností. Dále budou v této kapitole rozebrány alternativní systémy určování polohy. Následující kapitola 3 stručně popíše operační systém Android, řekne něco o historii tohoto operačního systému a popíše jeho charakteristiky. V kapitole 4 bude popsáno prostředí PhoneGap, včetně principu jeho fungování a Ionic SDK. Informační systém, který je cílem tohoto semestrálního projektu bude implementován s využitím moderních technologií pro vývoj webových aplikací, ty budou popsány v kapitole 5. Kapitola 6 čtenáři přiblíží a srovná již existující řešení zabývající se podobnou tématikou jako tato práce. Komplexní návrh informačního systému je rozebrán v kapitole 7. Tato kapitola se zabývá specifikací požadavků na výsledný systém, návrhem webového aplikačního rozhraní, databázového modelu, uživatelského rozhraní a systému DGPS. Po kapitole zabývající se návrhem aplikace logicky následuje kapitola 8 věnující se konkrétní implementaci řešení. Tato kapitola obsahuje také ukázky důležitých částí zdrojového kódu. Následující kapitolou je kapitola 9, která shrnuje hardwarové a softwarové požadavky jednotlivých částí systému. Kapitola 10 se zabývá vyhodnocením vlivu implementace diferenciální GPS na přesnost lokalizace. V kapitole 11, kapitole poslední, se nachází závěrečné zhodnocení práce včetně návrhů na budoucí rozšíření.

2. Systém GPS

GPS je globální polohovací systém. Je to mechanismus sloužící k nepřetržitému, přesnému určování polohy a poskytování údaje o aktuálním čase kdekoliv na světě, za jakéhokoliv počasí. Jedná se o jednosměrný pasivní systém poskytujících data neomezenému počtu uživatelů. S využitím tohoto systému nám GPS příjímače (například dnes často používané satelitní navigace) mohou zprostředkovávat informace o aktuálních zeměpisných souřadnicích, nadmořské výšce, rychlosti, směru a přesném čase [4, 18].

2.1. Historie GPS

Systém GPS byl vyvinut ministerstvem obrany Spojených států amerických pro armádní účely a jeho původním oficiálním názvem je „NAVSTAR GPS“ (Navigation Satellite Timing and Ranging System). Jeho vývoj byl zahájen v roce 1973 a vznikl sloučením dvou systémů, jedním z nich byl *System 621B* určený pro přesné určování polohy a druhým byl *Timation*, který sloužil pro přesné určování času. V letech 1974-1979 byly prováděny testy na pozemních stanicích a byl zkonstruován experimentální přijímač. Od roku 1978 začalo vypouštění družic na orbitu země a v roce 1994 byla na oběžnou dráhu umístěna kompetní sestava 24 družic. 17. července 1995 byla vyhlášena plná operační schopnost, takzvaná „FOC - full operational capability“ globálního polohového systému GPS, což znamená, že je zajištěna přítomnost alespoň 24 neexperimentálních satelitů GPS na oběžné dráze. Ve skutečnosti byl od tohoto data počet satelitů na orbitě země vždy vyšší než 24 [21].

Od roku 1990 byla do kódu radiového signálu GPS zanášena uměle vytvořená chyba, toto opatření pod názvem „Selektivní dostupnost“ mělo zabránit možnému zneužití například k navádění raket. Tato umělá chyba způsobovala nepřesnost v určování polohy až 45 metrů horizontálně. V roce 2000 byl vyvinut systém pro lokální rušení GPS a tak byla selektivní dostupnost zrušena [21].

2.2. Princip fungování systému GPS

Systém GPS lze rozdělit do tří segmentů zobrazených na obrázku obr. 2.1:

- kosmický
- řídící
- uživatelský

2.2.1. Kosmický segment

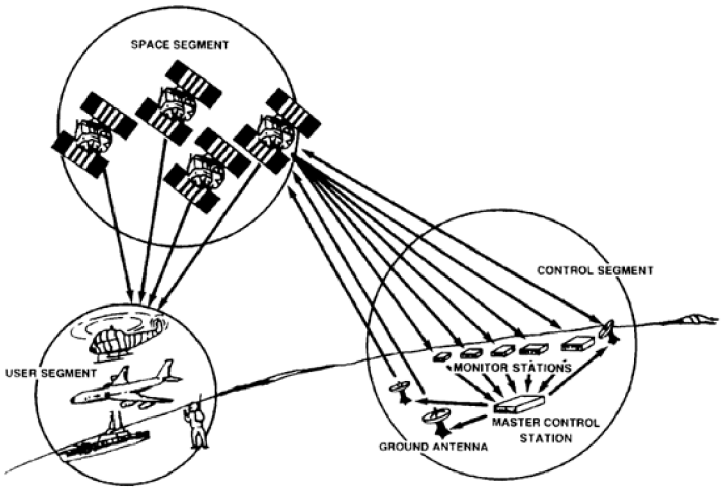
Hlavní součástí globálního polohovací systému GPS je soustava 24 satelitů na šesti téměř kruhových drahách (jedná se o elipsy s excentricitou maximálně 0,01) ve vzdálenosti kolem 26 560 km nad zemským povrchem. Na každé dráze se v jednom okamžiku vyskytuje 4 nebo 5 družic. Každá družice oběhne planetu zemi za přibližně 11 hodin a 58 minut. Tato konstelace zajišťuje, že jsou na každé zeměpisné poloze viditelné alespoň 4 satelity. Každý GPS satelit vysílá signál složený z několika komponent a to z dvou nosných frekvencí,

z dvou digitálních kódů a z navigační zprávy [4]. Více o účelu vysílaného signálu bude uvedeno v kapitole 2.3.

Aktuální počet operujících satelitů na oběžné dráze je 29. Pět satelitů je redundantních navíc oproti stavu plné operační schopnosti.

Klíčové části satelitů GPS jsou:

- 3 až 4 přesné atomové hodiny
- 12 antén pro vysílání rádiových signálů
- antény pro komunikaci s pozemními kontrolními stanicemi
- antény pro vzájemnou komunikaci družic
- optické, rentgenové a pulzní-elektromagnetické detektory, senzory pro detekci startů balistických raket a jaderných výbuchů
- solární panely a baterie jako zdroj energie [21]

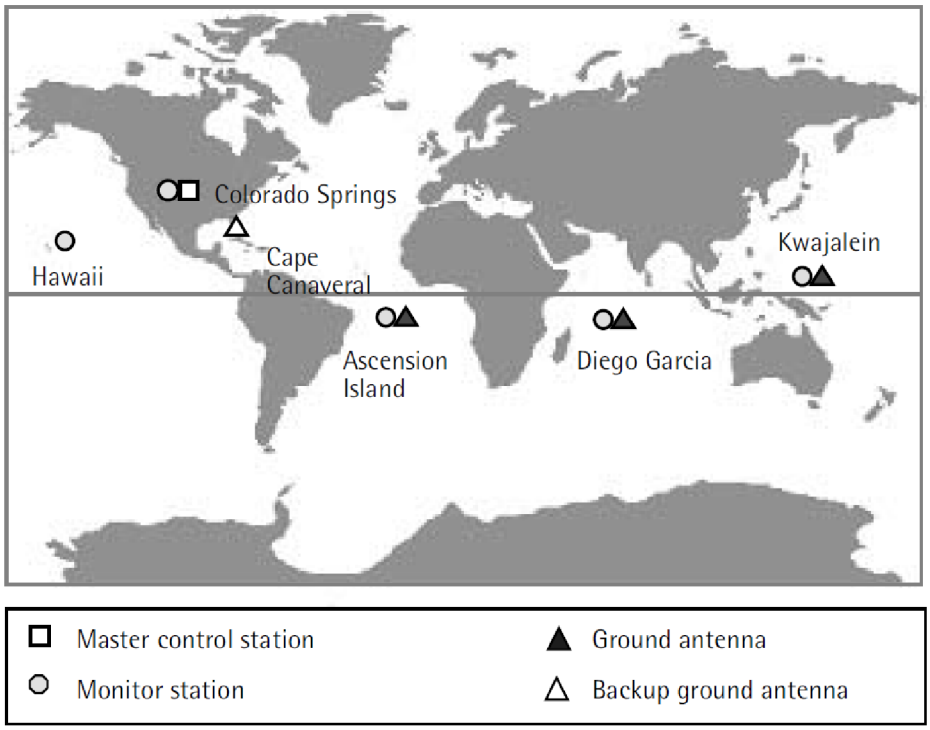


Obrázek 2.1: Segmenty systému GPS

2.2.2. Řídící segment

Řídící segment sestává z řídícího střediska „MCS - master control station“ a celosvětové sítě monitorovacích a kontrolních stanic. Pozice jednotlivých stanic jsou zobrazeny na obr. 2.2. V řídícím středisku je přítomen personál neustále obsluhující chod stanice. Monitorovacích stanic je 5 a jejich souřadnice jsou přesně známy. Každá z těchto stanic je vybavena kvalitními GPS příjimači a přesnými hodinami a jejich účelem je mimo jiné neustálé sledování pozice GPS družic v dohledu. Tři z těchto stanic (kontrolní stanice) jsou vybaveny také anténami pro upload informací do GPS satelitů. Všechny monitorovací stanice fungují bez lidské obsluhy, jsou řízeny z řídícího střediska vzdáleně [4].

Informace o GPS družicích jsou zasílány do řídícího střediska, kde jsou vyhodnocovány. Výstupem zpracování dat v MCS jsou mimo jiné předpovězená data o poloze satelitů, parametry družicových hodin, data o atmosféře a další. Tato data pak řídící středisko zasílá některé z kontrolních stanic a ta je pak uploaduje do GPS satelitů [4].



Obrázek 2.2: Pozice stanic řídícího segmentu GPS

2.2.3. Uživatelský segment

Uživatelský segment sestává z neomezeného počtu uživatelů, kteří pomocí GPS přijímače přijímají signály z jednotlivých družic, které jsou v danou chvíli nad obzorem. Na základě přijatých dat z jednotlivých družic obsahujících časové značky a jejich polohu příjmač spočítá své vlastní souřadnice a nadmořskou výšku [4].

Uživatele polohovacího systému GPS lze rozdělit na:

- autorizované uživatele
- neautorizované uživatele

Autorizovaní uživatelé

Autorizovanými uživateli systému GPS je armáda USA a vybrané spojenecké armády. Tito uživatelé mohou používat takzvaný „PPS - Precise Positioning System“, mají k dispozici dekódovací klíče a systém GPS jim poskytuje zaručenou vyšší přesnost. Více o PPS a rozílu v přesnosti PPS a SPS bude rozebráno v kapitole 2.4 [21].

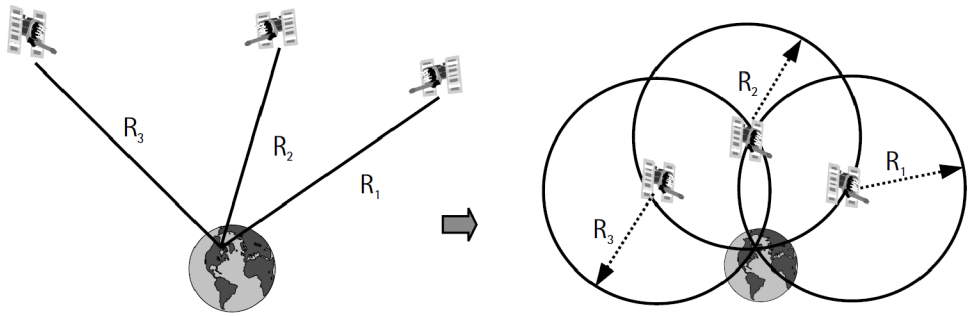
Neautorizovaní uživatelé

Neautorizovaní uživatelé jsou všichni ostatní uživatelé včetně široké veřejnosti. Tito uživatelé používají „SPS - Standard Positioning System“. Kvůli prevenci možného zneužití GPS pro navádění střel musí mít všechna zařízení exportována z USA omezení na určování pozice do nadmořské výšky 18km a rychlosti 515m/s [21].

2.3. Princip lokalizace pomocí systému GPS

Základní myšlenkou lokalizace pomocí GPS je, že pokud známe vzdálenost mezi místem na zemi (GPS přijímačem) a třemi GPS satelity, stejně tak jako pozice těchto satelitů, můžeme polohu tohoto místa matematicky spočítat pomocí principu trilaterace. Každý satelit nepřetržitě vysílá rádiový signál složený ze dvou nosných frekvencí, dvou digitálních kódů a z navigační zprávy. GPS přijímač pak tyto signály přijímá pomocí antény a dále jej zpracovává pomocí vestavěného software. Výsledkem tohoto zpracování je pak vzdálenost GPS satelitů a aktuální polohy družic. Pro výpočet vzdálenosti od satelitu se využívá časového rozdílu mezi okamžikem vyslání a přijmutí dat. Aktuální poloha družic, stejně jako informace o atomovém čase družice a přibližné poloze ostatních satelitů, je přenesena v navigační zprávě. K výpočtům vzdálenosti je nutné, aby byl čas na přijímači velmi přesný, každá i sebemenší odchylka by prakticky způsobovala významné chyby v lokalizaci, a proto je při inicializaci přijímače čas synchronizován s časem atomových hodin satelitu [4].

Teoreticky jsou pro určení polohy dostačující tři satelity, v tomto případě by byla poloha vypočítána jako průsečík tří kružnic. Každá z těchto kružnic by měla poloměr délky vzdálenosti mezi přijímačem a jednou z družic, jak je vidět na obr. 2.3. Prakticky se však



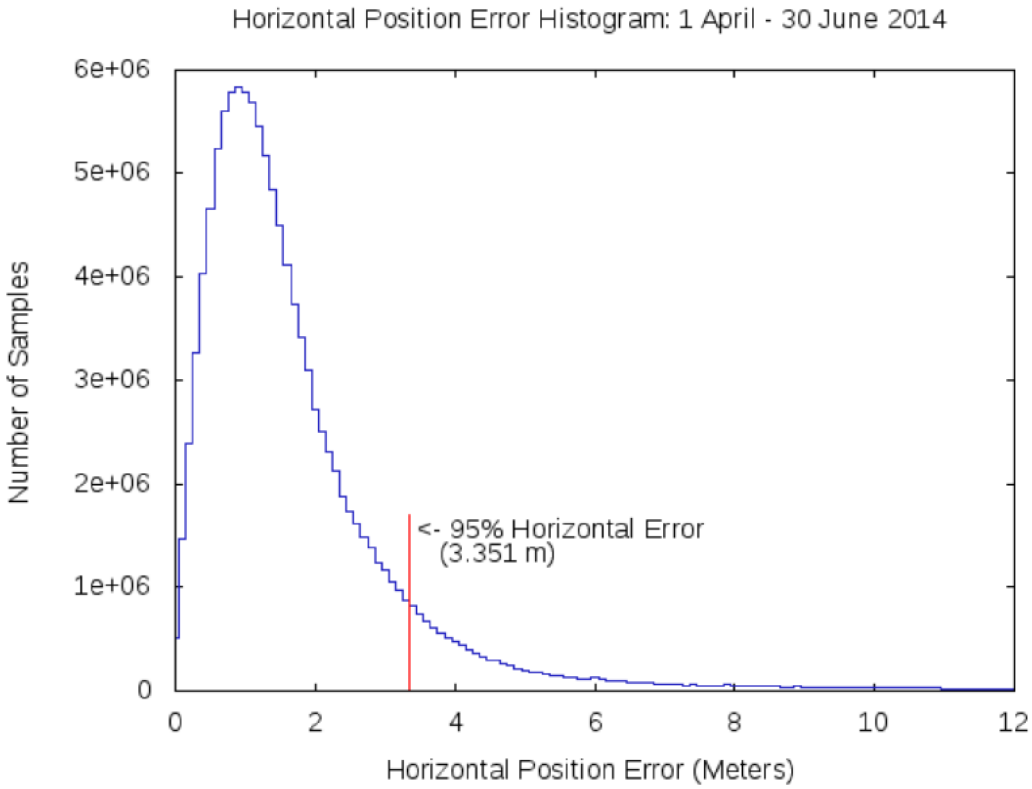
Obrázek 2.3: Základní princip určení polohy

pozice počítá z dat alespoň čtyř GPS satelitů, čtvrtý satelit je použit k výpočtu offsetu mezi časem přijímače a časem družic a tím i k významnému zpřesnění. Pro představu, při lokalizaci za užití pouze tří satelitů by chyba v čase velikosti $10^{-6}s$ prakticky způsobila chybu v určení pozice velikosti circa 300 metrů. Dnešní GPS přijímače dokážou přijímat a zpracovávat signál i z většího počtu satelitů než jsou čtyři, což má za následek další zvýšení přesnosti [4].

2.4. Přesnost systému GPS

Vláda spojených států je zavázána civilní veřejnosti zajišťovat jistou přesnost specifikovanou standardem „Standard Positioning System Performance Standard“. Jednou z těchto specifikací je, že GPS signál v kosmu bude poskytovat v nejhorším případě nepřesnost 7,8 metru s 95% jistotou. V tomto případě se nejedná o uživatelskou nepřesnost v určování pozice, ale o nepřesnost ve vzdálenosti satelitu a přijímače. Reálná data naměřena institucí *FAA* zobrazena na obrázku 2.4 názorně ukazují, že vysoce kvalitní GPS SPS

přijímač poskytuje horizontální přesnost s chybou menší než 3,5 metru. Dokonce je tato chyba poměrně často menší než 1 metr [20].



Obrázek 2.4: Experimentálně měřená horizontální uživatelská přesnost GPS

Jak bylo již zmíněno v odstavci 2.1, systém GPS byl původně vyvinut pro potřeby armády. I přes to, že byl tento systém uvolněn i pro širké civilní užití, ponechala si armáda USA výhodu spočívající v poskytování dvou úrovní služeb a to *Precise Positioning Service (PPS)* a *Standard Positioning Service (SPS)*. PPS umožňuje lokalizaci s vyšší přesností než SPS protože SPS vysílá na jedné frekvenci, kdežto PPS na dvou současně. Vysílání na dvou frekvencích umožňuje autorizovaným uživatelům korekce chyb, způbených průchodem signálu ionosférou. Probíhající program modernizace GPS má v úmyslu přidat více frekvencí, na kterých budou družice vysílat a umožní tak ionosférické korekce i civilním uživatelům. Rozdíly v přesnosti SPS a PPS tak budou prakticky eliminovány [20].

2.4.1. Možnosti zvýšení přesnosti lokalizace pomocí GPS

Mimo možnosti používání autorizovaného přístupu k GPS existují další cesty, jak zvýšit přesnost určování polohy pomocí tohoto systému a těmi jsou takzvané *rozšiřující systémy*. Rozšiřující systém GPS je jakýkoliv systém poskytující zlepšení v přesnosti navigace nebo přesnosti času, který není přímo součástí GPS. Tyto systémy lze rozdělit do dvou základních skupin a to:

GBAS - ground-based augmentation system

Tyto systémy rozšiřující GPS se skládají ze sítě pozemních stanic, které kontinuálně

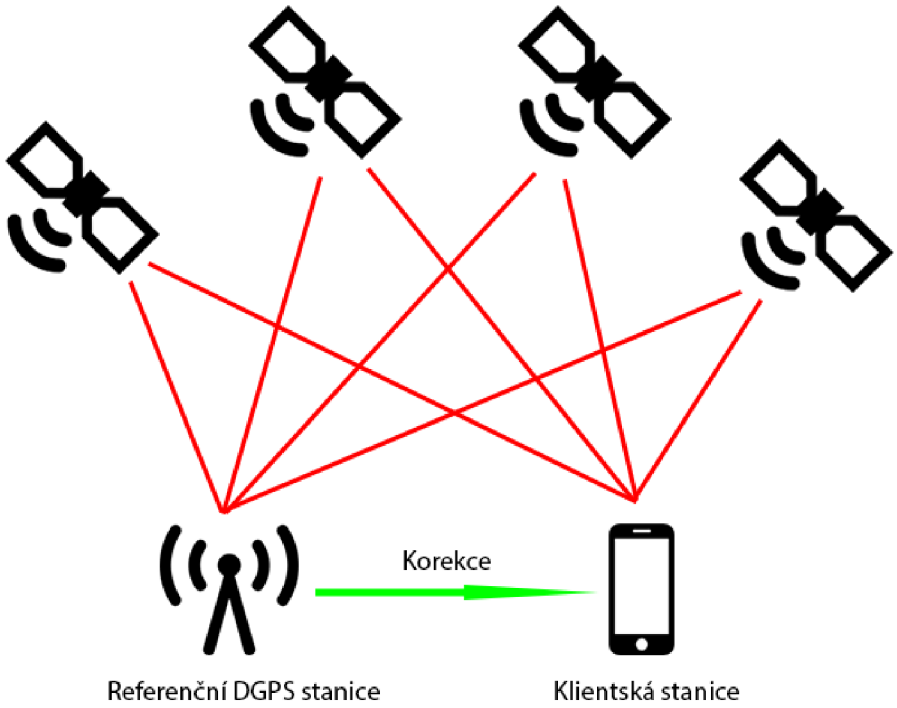
vyhodnocují stav kosmického segmentu, vypočítávají korekce a odesílají je uživatelům těchto systémů pomocí vysokofrekvenčního radiového spojení z pozemního vysílače. Příkladem takového systému je „Nationwide Differential GPS System (NDGPS)“, který operuje v USA a je schopen zvýšit přesnost GPS až na maximální odchylku 10-15cm.

SBAS - satellite-based augmentation systems

Základem systémů SBAS je podobně jako u systémů GBAS pozemní síť stanic vyhodnocujících stav kosmického segmentu a ionosféry. Na rozdíl od předchozího případu jsou data odesílána uživatelům skrze další družice. Nevýhodou tohoto řešení je, že vysílající družice mají poměrně slabý výkon, a proto je předurčeno pro leteckou případně námořní dopravu. Příkladem takového systému je „Wide Area Augmentation System (WAAS)“, který je určen k navigaci v letecké dopravě v Severní Americe [20].

DGPS

Dalším možným způsobem, jak zvýšit přesnost určení pozice je využitím diferenciálních korekcí DGPS. Princip je zobrazen na obrázku 2.5.



Obrázek 2.5: Schéma systému DGPS

Princip je založen na tom, že do bodu o známých souřadnicích umístíme referenční stanici, která bude porovnávat naměřenou polohu se svojí referenční polohou. Následně bude rozdíl mezi těmito dvěma údaji odesílat klientským stanicím. Klientské stanice tento rozdíl použijí jako korekci určení svojí vlastní pozice. Výsledná přesnost závisí na tom,

jak velká vzdálenost je mezi referenční stanicí a stanicemi odebírajícími korekce a také na tom, jak dlouhá doba uběhla od výpočtu korekce [15].

2.5. Alternativní systémy určování polohy

Souběžně se systémem GPS existuje několik dalších alternativních systémů pro určování polohy. Za zmínku určitě stojí ruský systém GLONASS a evropský systém Galileo. Tyto systémy jsou globální, jejich služby jsou dostupné z kteréhokoliv místa na zemi. Je také možné přijímat na jednom přijímači signály z více lokalizačních systémů a tak zvýšit výslednou přesnost a spolehlivost výsledného určení pozice.

2.5.1. GLONASS

„GLObal NAvigation Satellite System“ nebo také „GLObalnaya NAvigatsionnaya Sputnikovaya Sistema“ je globální navigační systém vyvinutý ministrem obrany Ruské federace [9]. Aktuálně se skládá z 28 satelitů obíhajících planetu Zemi po třech drahách ve výšce asi 19 130m s periodou oběhu 11 hodin 15 minut 44 sekund. Z těchto družic je 24 aktivních a slouží k určování polohy a přesného času, dva jsou záložní, jeden je ve výzkumu a jeden v testování [17].

2.5.2. Galileo

Globální systém Galileo, je Evropský autonomní navigační systém, který je aktuálně ve vývoji. V případě dokončení realizace by tento systém poskytoval Evropě nezávislost na americkém lokalizačním systému GPS nebo ruském GLONASS. Tento systém bude díky vysílání na duální frekvenci schopen poskytovat přesnost do jednoho metru [9]. Aktuálně je na oběžné dráze dostupných 9 satelitů systému Galileo (stav v dubnu 2016). Plné operační kapacity by měl Galileo dosáhnout kolem roku 2020 [5].

3. Google Android

Operační systém pro mobilní zařízení Android byl vyvinut začátkem 21. století a jeho tvůrcem je Andy Rubin. V roce 2005 tento operační systém koupil Google a Andy Rubin se stal ředitelem divize mobilních platforem v Google. Díky tomuto kroku se Google stal konkurenceschopným mezi ostatními společnostmi v té době vyvíjejícími mobilní platformy. Mezi hlavní konkurenční mobilní platformy patří Apple iOS a Microsoft Windows Mobile. Jak se Android vyvíjel, tak vznikaly jednotlivé verze tohoto systému, kdy každá přinesla nová vylepšení a nové možnosti. Jednotlivé verze jsou pojmenované podle dezertů, jak je vidět v tabulce na obr. 3.1. Předchozí verzí byla 5.0 Lollipop a aktuální se jmenuje Marshmallow [7].

VERSION	CODENAME	API LEVEL	MARKET SHARE
1.5	Cupcake	3	0.2 %
1.6	Donut	4	0.4 %
2.1	Eclair	7	3.7 %
2.2	Froyo	8	14 %
2.3.2	Gingerbread	9	0.3 %
2.3.7	Gingerbread	10	57.2 %
3.1	Honeycomb	12	0.5 %
3.2	Honeycomb	13	1.6 %
4.0.2	Ice Cream Sandwich	14	0.1 %
4.0.4	Ice Cream Sandwich	15	20.8 %
4.1	Jelly Bean	16	1.2 %

Obrázek 3.1: Verze operačního systému Android

Android je založen na operačním systému Linux a programovacím jazyku Java, jednom z nejpoužívanějších programovacích jazyků na světě. Jedná se o open source projekt dostupný všem. Platforma Android představuje nejenom operační systém, ale i kompletní řešení pro nasazení tohoto systému do provozu. Poskytuje ovladače zařízení a další, proto lze tento systém použít bez ohledu na použitý hardware nebo rozlišení obrazovky zařízení. V neposlední řadě poskytuje Software Development Kit, čímž umožňuje velmi efektivní vývoj aplikací [13].

Není neobvyklé, že přístroje běžící na platformě Android jsou vybaveny procesory s taktovací frekvencí přesahující 2GHz a operační pamětí přes 2GB. Tyto přístroje se tak pro některé uživatele stávají plnohodnotnou náhradou klasických stolních počítačů nebo notebooků [13].

V dnešní době mnoho velkých společností vyrábí přístroje běžící na platformě Android. Jako příklad uvedu korporace HTC, Samsung, LG. Výrobci mobilních zařízení stojící za vývojem operačního systému Android jsou seskupeni do skupiny *OHA - Open Handset Alliance*. OHA umožňuje vývojářům vyvíjet aplikace v jednom prostředí. Tyto aplikace pak mohou vývojáři jednoduše poskytnout velkému množství uživatelů pro jejich zařízení běžících na operačním systému Android nezávisle na tom, který výrobce tato zařízení vyrobil [13].

Dnes operační systém Android využívá mimo smartphony také řada dalších nejrůznějších zařízení jako jsou tablety, smart TV a další. Každé z těchto zařízení, pokud je připojeno

k internetu, má přístup k řadě aplikací pomocí online obchodu s aplikacemi nazývaného Google Play [13].

Android SDK nabízí velké množství Java balíčků, poskytujících vývojářům nástroje pro práci s mnoha nejrůznějšími prostředky zařízení, jako jsou akcelerometr, fotoaparát, GPS a další [13].

4. PhoneGap a Ionic

PhoneGap je open source framework určený k vytváření nativně instalovaných mobilních aplikací pro všechny hlavní mobilní platformy za užití standardních webových technologií jako jsou HTML, CSS a JavaScript.

Vývoj aplikací pro různé mobilní operační systémy, ať už se jedná o Google Android, Apple iOS, Windows Mobile a další, vyžaduje znalost několika programovacích jazyků a frameworků a tím zvyšuje náročnost a s tím spojené náklady na jejich vývoj. PhoneGap je framework, který řeší tento problém tak, že propojuje webové aplikace vytvořené standardními webovými technologiemi jako jsou HTML, JavaScript a CSS s mobilními aplikacemi. Za pomocí PhoneGapu lze z webové aplikace vytvořit jednotlivé mobilní aplikace pro všechny hlavní mobilní operační systémy. PhoneGap dnes používá více než 400 000 vývojářů a byly v něm vytvořeny tisíce aplikací přítomných v online obchodech s aplikacemi pro mobilní platformy [1].

PhoneGap je vyvíjen pod záštitou Apache Software Foundation (ASF) pod jménem Apache Cordova. Tento framework bude vždy k dispozici zdarma a zůstane Open Source pod licencí Apache Licence, Version 2.0 [1].

Uživatelské rozhraní PhoneGap aplikace je vytvořeno pomocí HTML, CSS a JavaScriptu. Vrstva uživatelského rozhraní je ve výsledné aplikaci tvořena webovým prohlížečem, který zabírá 100% šířky a 100% výšky velikosti obrazovky zařízení. Jedná se o takzvaný „chrome-less“ prohlížeč, který v sobě neobsahuje žádná ovládací tlačítka ani dekoraci okna, jako běžný webový prohlížeč. Přestože jsou PhoneGap aplikace vytvořeny pomocí standardních webových technologií, výsledkem je binární aplikace, kterou lze na zařízení instalovat a distribuovat ji pomocí standardních ekosystémů konkrétní platformy. PhoneGap poskytuje rozhraní pro programování aplikací, zkráceně *API (application programming interface)*, implementované takzvanými pluginy, které zpřístupňují aplikaci napsané v JavaScriptu funkcionalitu a prostředky nativního operačního systému. Vývojář pak prakticky píše aplikaci v JavaScriptu a PhoneGap se stará o komunikaci s operačním systémem [19].

Ionic je HTML5 SDK, které výrazně usnadňuje vytváření nativně působících mobilních aplikací pomocí technologií HTML, CSS a Javascript. Zaměřuje se hlavně na vzhled a to, jak interakce s uživatelským rozhraním na uživatele působí. Je to vhodný doplněk PhoneGapu [12].

5. Moderní technologie na webu

5.1. Node.JS a Express

Většinu webových aplikací lze rozdělit na klientskou část a serverovou část. Implementace serverové části vždy bývala poměrně těžkopádná. Oproti klientské části je serverová část často implementována odlišnými programovacími jazyky a to nutí programátory při vývoji používat a znát těchto jazyků více.

Nápad programovat serverovou část aplikace v JavaScriptu, jazyku typickém pro vývoj klientské části aplikací, byl ještě donedávna téměř nemyslitelný. Tento jazyk nebyl moc výkonný, postrádal dostatečně sofistikovanou správu paměti, nebyl dostatečně integrován do operačního systému. Aby mohl být JavaScript považován za jazyk vhodný pro serverová řešení, musely být všechny tyto aspekty zlepšeny. Problémy s výkonem a správou paměti byly vyřešeny s příchodem **V8 JavaScript**.

V roce 2009 Ryan Dahl na konferenci v Berlíně uvedl technologii zvanou Node.JS. Využil příležitosti jak dostat JavaScript na server a zabudoval JavaScript V8 do vrstvy v operačním systému [16].

Výhody tohoto přístupu jsou zřejmé, vývojáři od tohoto okamžiku mohou používat stejný programovací jazyk jak na klientské tak i serverové části webové aplikace. Dynamická povaha JavaScriptu zjednodušuje vývoj serverového kódu a umožňuje programátorům odpoutat se od tradičního programovacího modelu. Node.JS se stal okamžitě úspešným a vytvořila se kolem něj velká komunita vývojářů, podporují jej velké společnosti [16].

Vytváření webového serveru pomocí Node.JS je velmi efektivní a rychlé. Příkladem může být následující *helloworld* ukázkový kód pro server.

```
var http = require(‘http’);
var server = http.createServer(function (req, res) {
    res.writeHead(200);
    res.end(’Hello world’);
});
server.listen(80);
```

Výše uvedený ukázkový kód představuje vysoce výkonný web server naprosto srovnatelný nebo dokonce i výkonnější než tradiční řešení vytvořená pomocí software jako je *Apache* nebo *Nginx*. Vysoká rychlost a výkon Node.JS technologie je způsobena faktem, že běží na V8 JavaScriptovém interpretu a virtuálním stroji vyvinutém společností Google, který byl vytvořen za účelem dosažení vysoké rychlosti [16].

Node.JS obsahuje mnoho užitečné funkcionality už ve svém vlastním jádru. Navíc přichází s balíčkovým manažerem *NPM - Node Package Manager*, pomocí něhož lze do aplikace přidávat jednotlivé moduly vytvořené komunitou. To vede k vývoji spoléhajícím se pouze na základní Node.JS moduly a na vybrané moduly instalované pomocí NPM [16].

Pro zjednodušení použití Node.JS pro webové aplikace vznikl framework Expess.js. Tento framework pomáhá organizovat webovou aplikaci, poskytuje mnoho mechanismů jako je volba správné cesty na základě regulárního výrazu, parsování HTTP dotazů a odpovědí a dalších.

5.2. MongoDB a Mongoose

MongoDB je výkonný, flexibilní, škálovatelný databázový systém pro obecné užití. Je to *dokumentově-orientovaný* databázový systém. Na rozdíl od například známé a na webu již dlouhá léta používané mySQL se nejedná o relační databázi. Hlavním důvodem k odklonu od relačního modelu databáze je mimo jiné hlavně zjednodušení škálovatelnosti. Dokumentově orientované databáze nahrazují koncept „řádku“ tabulky, který v relačních databázích představuje záznam, konceptem „dokumentu“, který je mnohem flexibilnější. Dokument umožňuje reprezentovat komplexní hierarchické vztahy jediným záznamem, což více odpovídá způsobu, jak se v moderních objektově orientovaných programovacích jazycích nahlíží na data. Dále zde nejsou žádná předdefinovaná schémata, klíče a hodnoty dokumentu nejsou omezovány na konkrétní datové typy nebo velikost, což zjednodušuje přidávání a odstraňování dat. Ve výsledku mají tyto vlastnosti pozitivní dopad na rychlost vývoje aplikací [10].

Používání MongoDB ve webových aplikacích je samo o sobě podstatný krok vpřed, nicméně i tak je přímá interakce s *mongodb* modulem pro NodeJS trochu těžkopádná a nepříliš uživatelsky přívětivá. Přesně z toho důvodu vznikl framework Mongoose. Mongoose zjednodušuje a zpřehledňuje organizaci databáze pomocí takzvaných schémat (nebo také definic modelu). Mimo schémata také poskytuje možnosti validace a virtuálních vlastností (properties). Mongoose je ale pořád MongoDB, jedná se v podstatě o jeho nástavbu, takže lze využívat i všech vlastností MongoDB [14].

5.3. REST

REST *(Representational State Transfer)* je architektura aplikačního rozhraní navržená pro vytváření a organizaci distribuovaných systémů. Navrhl ji Roy Fielding, jeden z hlavních autorů HTTP protokolu. Aplikace využívající REST architekturu se nazývají RESTful aplikace. Tyto RESTful aplikace využívají klasických HTTP dotazů pro operace čtení, vytváření, mazání a změnu dat na serveru známé pod souhrnnou zkratkou CRUD. K provádění těchto operací slouží standardní HTTP metody *GET, POST, PUT a DELETE*. REST rozhraní je odlehčená alternativa k mechanismům jako jsou RPC (Remote Procedure Calls) a k Web Services jako je například SOAP (Simple Object Access Protocol). Základním stavebním kamenem REST architektury jsou takzvané *resources*. Resources mají následující vlastnosti:

Reprezentations Tato vlastnost definuje způsob reprezentace dat. Můžou to být binární data, JSON, XML a další.

Identifier Jedná se o adresu URL, která vrací jeden konkrétní resource v jakémkoliv čase.

Metadata Jedná se o metadata jako jsou například *content-type, last-modified time* atd.

Control data Informace *is-modifiable-since* a *cache-control* [3].

5.3.1. Základní vlastnosti REST architektury

client/server

Server obsahuje sadu služeb a naslouchá dotazům na tyto služby. Dotazy jsou tvořeny na straně klienta a slouží k dotazování se konkrétních služeb serveru. Hlavní vlastností tohoto přístupu je oddělení kódu pro klientskou část a pro serverovou část, což umožňuje nezávislý vývoj obou komponent [3].

Komunikace je bezstavová

Komunikace mezi serverem a klientem musí být bezstavová - stateless, což prakticky znamená, že každý dotaz klienta na server musí obsahovat veškeré informace, které server potřebuje k jeho vyřízení, aniž by bylo třeba se spoléhat na jakákoliv uložená data. Tento přístup představuje následující zlepšení pro ostatní části systému:

viditelnost Monitorování výsledného systému se podstatně zjednoduší, jelikož všechny informace, které k němu potřebujeme, máme přímo v dotazu.

škálovatelnost Jelikož není mezi jednotlivými dotazy potřeba ukládat žádná data, server může rychleji uvolňovat prostředky.

spolehlivost Bezstavový systém lze v případě pádu obnovit jednodušeji, jelikož stačí obnovit aplikaci samotnou.

jednodušší implementace Jelikož není třeba spravovat uložená data na serveru, případně i mezi několika servery, je výsledná serverový kód jednodušší.

Na druhé straně má tento přístup i nevýhodu, kterou je větší množství opakujících se dat odesílaných na server. Je třeba zvážit dle každé konkrétní aplikace, zda je vhodné REST architekturu použít [3].

cacheable

Každá odpověď serveru na dotaz klienta může být explicitně nebo implicitně označena jako cacheovatelná. Cachování (ukládání) odpovědí serveru přidává celkové architektuře několik výhod. Na serverové straně se lze kompletně vyhnout některým interakcím (například dotazům na databázi) v případě opakujících se dotazů. Na klientské straně lze dosáhnout podstatného zrychlení [3].

uniformní rozhraní

Jedna z hlavních charakteristik REST je, že definuje jednotné rozhraní, což zjednodušuje klientskou interakci se serverem. Používání standardního rozhraní umožňuje komunikaci různých druhů klientů se serverem nezávisle na jejich implementaci [3].

vrstvený systém

Architektura REST je navržena pro použití ve vrstvených systémech. Oddělením jednotlivých komponent serveru do vrstev a umožněním každé vrstvě pouze využívat vrstvy pod sebou a poskytovat data vrstvě nad sebou lze zjednodušit celkovou komplexnost systému. Tato vlastnost je velkou výhodou hlavně pro stále rostoucí velké systémy [3].

Code-On-Demand

Klient může stáhnout a spustit kód poskytnutý serverem [3].

5.4. AngularJS

AngularJS je open source JavaScriptový framework, který je založen na MVC (Model View Controller) architektuře a je spravovaný společností Google. MVC architektura je popsána v 5.4.1. Vznikl v roce 2009 a vytvořili ho vývojáři Miško Hevery and Adam Abrons. Původně se mělo jednat o komerční projekt pro urychlení vývoje webových aplikací, následně však byl uvolněn jako open source knihovna. Cílem bylo vytvořit framework, se kterým půjde aplikace nejenom rychle a jednoduše vytvořit, ale také dále rozšiřovat, spravovat a testovat [8].

5.4.1. Základní koncepty angularu

Existuje několik základních myšlenek frameworku AngularJS, kterých je ve výsledné aplikaci využíváno, a které dohromady činí tento framework výjimečným oproti ostatním alternativám.

Client-Side Templates

V běžných vícestránkových dynamických webových aplikacích jsou HTML stránky konstruovány a plněny daty na serveru a až poté, co jsou kompletně vytvořeny, jsou posílány klientské straně. Většina jednostránkových aplikací (známých také jako AJAXové aplikace) to dělají analogicky až na některé části. Angular má k tomuto opačný přístup. HTML šablony jsou zde odesílány klientovi ihned a jejich obsah je do nich doplněn až na klientské straně. Role serveru je v tomto případě taková, že pouze poskytuje statické soubory a požadovaná data.

Tuto funkcionalitu se pokusím prezentovat následujícím příkladem. Příklad se skládá z HTML šablony a kontroleru (JavaScriptového kódu).

HTML šablona:

```
<html ng-app>
    <script src="angular.js"></script>
    <script src="controllers.js"></script>
<body>
    <div ng-controller='HelloController'>
    <p>{{greeting.text}}, World</p>
    </div>
</body>
</html>
```

JavaScript kód v souboru *controllers.js*

```
function HelloController($scope) {
    $scope.greeting = { text: 'Hello' };
}
```

Řetězec ve vlastnosti *greeting.text* objektu *$scope* je vložen do HTML šablony. Není třeba žádných HTML tříd nebo id k identifikaci toho, kam řetězec vložit, jako tomu je v jiných frameworcích [8].

Model View Controller (MVC)

MVC struktura aplikace byla uvedena již v 80tých letech jako součást jazyka SmallTalk a brzy se stala velmi populární. Ať už používáme C++, Javu nebo Objective-C, jsou zde prvky MVC k dispozici, nicméně donedávna tato architektura neměla nic společného s vývojem webu. Základní myšlenkou této architektury je přesné oddělení kódu na části, které se starají o data (*model*), aplikační logiku (*controller*) a prezentování dat uživateli (*view*). *View* zobrazuje data z *model* uživateli, ten provádí nějakou akci (kliknutí, psaní na klávesnici) a *controller* na základě této akce mění data na *model*. V případě Angularu *view* představuje DOM (Document Object Model), *controllers* jsou JavaScriptové pseudotřídy a data jsou uložena ve vlastnostech objektů [8].

Tento přístup zapříčiňuje lepší čitelnost výsledného kódu a zajišťuje mnohem jednodušší možnosti rozšiřování, udržování a testování aplikace.

Data Binding

Předtím, než se rozšířily AJAXové jednostránkové aplikace, poskytovaly nám platformy jako *PHP*, *Ruby on Rails* nebo *JSP* možnost přidat data do řetězců HTML před odesláním klientské aplikaci ke zobrazení uživateli. Frameworky pro vývoj klientské části aplikace jako je jQuery poskytují podobnou funkcionalitu, ale navíc umožňují aktualizovat pouze část HTML dokumentu bez nutnosti pokaždé znovu načíst celou stránku [8].

Tyto technologie fungují velmi dobře, ale situace, kdy potřebujeme zobrazovat uživateli data vždy aktuální, nebo měnit data na základě uživatelského vstupu lze řešit pouze netriviální přidanou logikou, která musí zajistit, že data v JavaScriptových objektech i v HTML šabloně jsou korektní v každém okamžiku. Angular tuto problematiku řeší mechanismem nazývaným data-binding, který dokáže zajistit, že určitá data v JavaScriptu aplikace a v HTML šabloně jsou vždy aktuální, zajišťuje neustálou synchronizaci [8].

Praktický příklad by vypadal následovně. Opět se bude skládat z HTML šablony a z kontroleru. Aby bylo možné uživatelským vstupem měnit hodnotu *greeting.text*, byl do šablony přidán formulářový prvek *input*, který je na tuto proměnnou navázán.

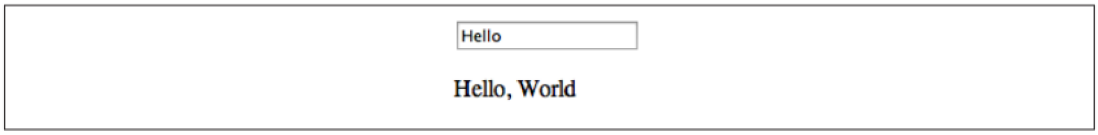
HTML šablona:

```
<html ng-app>
<head>
    <script src="angular.js"></script>
    <script src="controllers.js"></script>
</head>
<body>
    <div ng-controller='HelloController'>
        <input ng-model='greeting.text'>
        <p>{{greeting.text}}, World</p>
    </div>
</body>
</html>
```

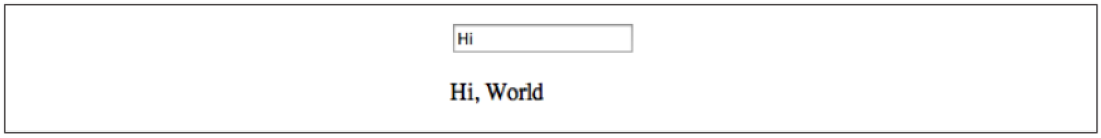
JavaScript kód v souboru *controllers.js*

```
function HelloController($scope) {
    $scope.greeting = { text: 'Hello' };
}
```

Výsledná HTML stránka otevřená ve webovém prohlížeči vypadá jako na obrázku 5.1. Pokud budeme měnit text v textovém inputu, tato změna se bude dynamicky téměř ihned propagovat. Takže pokud změníme text z „Hello“ na „Hi“, bude stránka v prohlížeči vypadat jako na obrázku 5.2.



Obrázek 5.1: Původní stav HTML stránky



Obrázek 5.2: Stav HTML stránky po změně propagované pomocí data bindingu

Dependency Injection

Angular *Dependency Injection* je systém správy závislostí, postavený na principu minimální znalosti. V podstatě jde o to, že si konkrétní část aplikace řekne o závislost kterou potřebuje a dále se nestará o to, jak a kde vznikla. Angular poskytuje specializované objekty, které poskytují určitou funkcionalitu a lze si je vložit „injectnout“ do vlastního kódu. Tyto objekty lze i vytvářet. Mezi tyto objekty patří například objekt *$location*, který má ve své režii adresní řádek prohlížeče, nebo již v předchozích příkladech použitý objekt *$scope* [8].

Directives

Direktivy (anglicky directives) je mechanismus, jak lze v Angularu rozšířit HTML syntax o další elementy nebo parametry elementů a těm pak přidat vlastní zapouzdřenou funkcionalitu. Mnoho direktiv je již v Angularu samotném a lze v něm vytvářet také své vlastní. Jedná se o velmi efektivní nástroj k vytváření znovupoužitelných komponent [8].

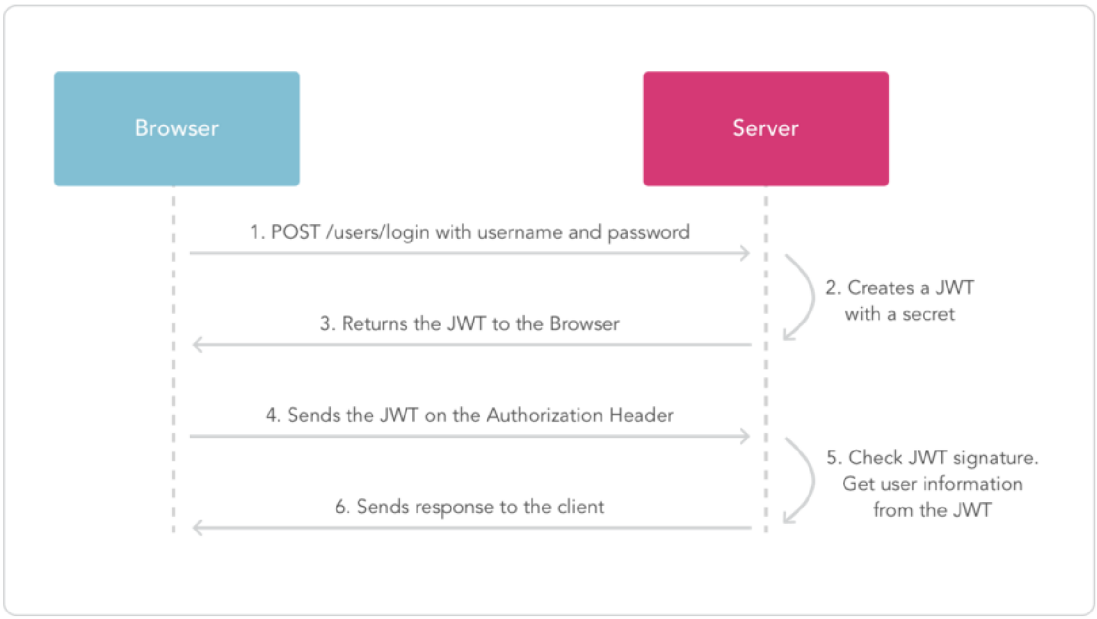
5.5. JSON Web Token

JSON Web Token (JWT) je otevřený standard (RFC 7519) definující způsob, jak posílat zabezpečené informace ve formě JSON objektu mezi dvěma stranami. Tyto informace jsou zabezpečeny buď s využitím tajného klíče a algoritmu HMAC a nebo pomocí páru privátní/veřejný klíč a algoritmu RSA. Vzhledem ke kompaktní velikosti mohou být JWT tokeny odeslány přímo jako součást URL, jako POST parametr a nebo uvnitř HTTP hlaviček. Zašifrovaný obsah JWT tokenu obsahuje veškeré potřebné informace o uživateli

(včetně času expirace tokenu), tudíž klient nemusí zasílat další dotazy pro tyto informace na server a server pro tyto informace nemusí přistupovat do databáze, proto tato technologie podstatně snižuje režie. JSON Web Token je vhodný pro odesílání nejrůznějších dat u kterých si potřebujeme být jisti, že je adresát opravdu ten, za koho se vydává a že obsah dat nebyl změněn. Hlavním využitím této technologie je autentizace uživatele [11].

5.5.1. Autentizace s využitím JWT

Princip autentizace pomocí JSON Web Token je názorně zobrazen na obrázku 5.3. V případě, že se uživatel úspěšně přihlásí, je na serveru vygenerován nový JWT token, který si klientská aplikace lokálně uloží. Následně pokaždé, když chce klient odeslat požadavek na zabezpečenou službu serveru, odešle spolu s dotazem i tento token, nejčastěji jako součást HTTP hlavičky. Server po přijetí dotazu token ověří a v případě úspěšného ověření odešle požadovaná data. Standard JWT velmi dobře vyhovuje konceptu bezstavovosti architektury REST (která je popsána v kapitole 5.3), jelikož server nikdy neukládá stav uživatele v jeho lokální paměti, všechny informace o uživateli získá z tokenu [11].



Obrázek 5.3: Princip autentizace pomocí JWT

6. Analýza již existujících řešení

6.1. Google Latitude

Google Latitude byla jednou z funkcí Map Google pro mobilní telefony. Tato služba umožňovala zobrazení aktuální polohy „přátel“, propojených uživatelů této služby. Tito „přátelé“ byli zobrazováni na mapě pomocí ikony obsahující jejich fotografii jak je zobrazeno na obrázku 6.1 a po zvolení ikony konkrétního přítele bylo možné tuto osobu kontaktovat zprávou SMS, chatu, nebo jim rovnou zavolat. V aplikaci šlo nastavit možnosti ochrany osobních údajů a zobrazování polohy, takže si každý uživatel mohl zvolit, která data o sobě komu sdílí. Tato služba byla v roce 2013 ukončena a v podstatě nahrazena možností sdílení polohy na sociální síti Google+ [23].

Google Latitude byla služba dostupná pro všechny hlavní mobilní platformy.

Obrázek 6.1: Google Latitude

6.2. Foursquare

Foursquare je celosvětová hra, kterou vymyslel Dennis Crowley v roce 2008. V této aplikaci mohou mít zajímavá místa jako jsou kavárny, parky, obchody svoji adresu. Uživatel této aplikace po příchodu na toto místo provede „check-in“ a za ně dostává virtuální odznaky. Pokud jich má na určitém místě nejvíce ze všech uživatelů foursquare, stává se „starostou“ místa. Místu, které ještě žádnou adresu ve Foursquare nemá, lze adresu přiřadit pomocí webové aplikace [24].

Tato aplikace se těší velké oblibě, měsíčně užívá Foursquare až 50 milionů uživatelů na webu, nebo v mobilní aplikaci [6].

6.3. LogBookie

V české republice působí společnost LogBookie, která nabízí širokou škálu placených služeb sledování polohy. Tato firma nabízí služby jako jsou sledování dětí, seniorů, nákladních vozidel, osobních automobilů, zemědělských strojů, lodí a dalších. Výstupem jejich sledování je i historie pohybu včetně mnoha statistik.

7. Návrh aplikace

7.1. Analýza požadavků uživatele

Požadavky uživatele na výsledné aplikace vychází ze zadání diplomové práce a lze je rozdělit na ty, které se týkají webové aplikace a na požadavky týkající se mobilní aplikace. Účelem mobilní aplikace je hlavně odesílání dat o poloze uživatele na server, zobrazování poloh stanic ve skupině na mapě a v neposlední řadě také možnost stát se stacionární stanicí DGPS sloužící k výpočtu korekcí GPS lokalizace za předpokladu, že bude stanice nehybně na svém místě. Webová aplikace bude sloužit pro prezentaci dat ze serveru, což bude aktuální poloha uživatele a polohy ostatních uživatelů ve skupině, historie polohy a statistické údaje. Obě z aplikací umožní uživateli registraci a přihlášení. Uživatelé budou mezi sebou moci vytvářet „spojení“, přijetím nabídky spojení uživatel umožní jinému uživateli vidět pozici na mapě.

7.1.1. Požadavky na mobilní aplikaci

1. Správa uživatelů
   (a) Registrace - uživatel zvolí své uživatelské jméno a heslo, vytvoří tak nového uživatele.
   (b) Přihlášení - do aplikace se lze přihlásit pomocí uživatelského jména a hesla.
   (c) Odhlášení - z aplikace se lze odhlásit.
2. Správa spojení
   (a) Uživatel může vyhledat jiného uživatele.
   (b) Uživatel může jinému uživateli odeslat žádost o spojení.
   (c) Uživatel může zrušit žádost, kterou dříve odeslal.
   (d) Uživatel může přijmout žádost, kterou mu odeslal jiný uživatel.
   (e) Uživatel může odstranit již existující spojení.
3. Zobrazení pozic uživatelů na mapě
   (a) Aplikace zobrazí pozice ostatních uživatelů ve spojení na mapě.
   (b) Aplikace zobrazí na mapě uživatele ve spojení, kteří jsou v režimu stacionární stanice DGPS.
   (c) Uživatel může vybrat ze stacionárních stanic DGPS stanici, pomocí které dojde ke zpřesnění vlastní pozice prostřednictvím korekcí DGPS.
   (d) Uživatel může vypnout zpřesnění pomocí DGPS.
4. Odesílání GPS pozice na server
   (a) Uživatel může zapnout odesílání svých GPS pozic na server.
   (b) Uživatel má možnost vypnout odesílání svých GPS pozic na server.

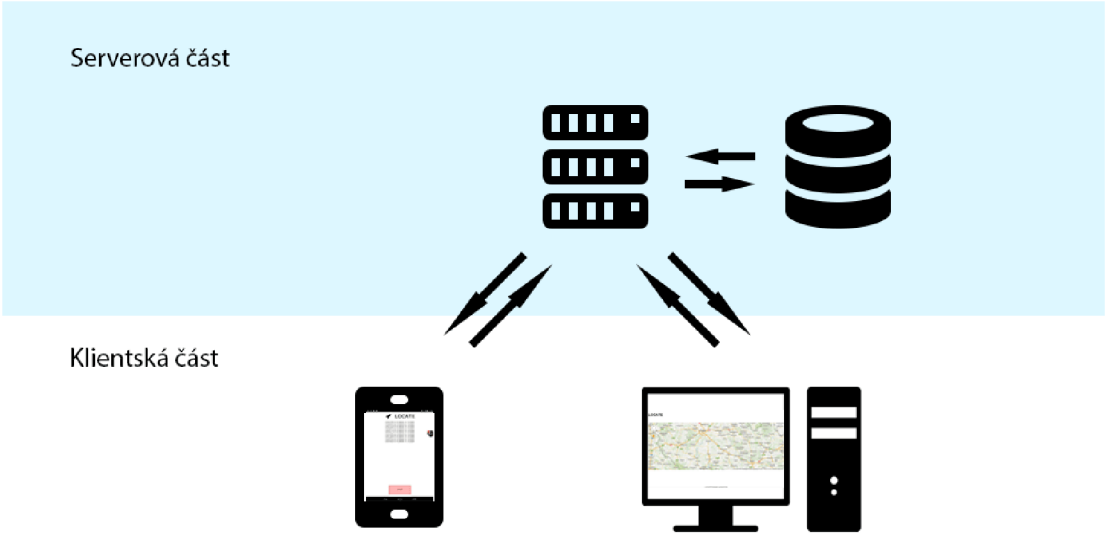
5. Režim DGPS
   (a) Uživatel může zapnout režim stacionární stanice DGPS.
   (b) Uživatel může vypnout režim stacionární stanice DGPS.
   (c) Aplikace zajistí vypnutí režimu stacionární stanice DGPS v případě změny polohy zařízení.

7.1.2. Požadavky na webovou aplikaci

1. Správa uživatelů
   (a) Registrace - uživatel zvolí své uživatelské jméno a heslo, vytvoří tak nového uživatele.
   (b) Přihlášení - do aplikace se lze přihlásit pomocí uživatelského jména a hesla.
   (c) Odhlášení - z aplikace se lze odhlásit.
2. Správa spojení
   (a) Uživatel může vyhledat jiného uživatele.
   (b) Uživatel může jinému uživateli odeslat žádost o spojení.
   (c) Uživatel může zrušit žádost, kterou dříve odeslal.
   (d) Uživatel může přijmout žádost, kterou mu odeslal jiný uživatel.
   (e) Uživatel může odstranit již existující spojení.
3. Zobrazení pozic uživatelů na mapě
   (a) Aplikace zobrazí pozice ostatních uživatelů ve spojení na mapě.
   (b) Aplikace zobrazí na mapě uživatele ve spojení, kteří jsou v režimu stacionární stanice DGPS.
   (c) Aplikace zobrazí seznam uživatelů a stacionárních DGPS stanic zobrazených na mapě.
4. Statistiky
   (a) Aplikace zobrazí graf přesností lokalizace v závislosti na čase.
   (b) Uživatel může vybrat den v minulosti pro zobrazení statistiky.
5. Historie poloh
   (a) Aplikace zobrazí historii poloh na mapě.
   (b) Uživatel může zvolit den a čas v minulosti pro zobrazení historie.
   (c) Uživatel může zvolit zobrazení poloh v historii mezi zobrazením s aplikovanými korekcemi DGPS a bez nich.

7.2. Návrh informačního systému

Navrhovaný informační systém se bude skládat ze čtyř hlavních částí, jak je zobrazeno na obrázku 7.1. Těmito částmi jsou mobilní aplikace běžící na operačním systému Android, webová aplikace, server a databáze. Každá z těchto částí bude mít jasně vymezenou funkcionalitu. Webová aplikace stejně jako mobilní aplikace budou se serverem komunikovat pomocí jednotného REST rozhraní prostřednictvím protokolu HTTP.



Obrázek 7.1: Návrh informačního systému

7.2.1. Serverová část

Serverová část informačního systému bude mít dvě hlavní role. Tyto role jsou úložiště dat a komunikační prostředek mezi jednotlivými částmi informačního systému a serverem. Prakticky mobilní zařízení pošle na server data prostřednictvím webového aplikačního rozhraní, server tato data zpracuje a uloží do databáze. Veškeré zpracovávání dat bude probíhat přímo na serveru, budou zde vytvořeny vhodné datové struktury pro uložení a prezentaci dat. Klientské aplikace si následně tato data vyžádají pomocí stejného aplikačního rozhraní a server jim je v případě úspěšné autentizace poskytne.

Server musí zajistit datovou konzistenci. Každému záznamu bude třeba přiřadit jednoznačný identifikátor. Dále bude také třeba zajistit správné mazání záznamů z databáze. Server musí také ošetřit korektnost všech klientských požadavků a správně reagovat na nekorektní požadavky. V neposlední řadě bude server poskytovat mechanismus správy uživatelů a možnost autentizace uživatele.

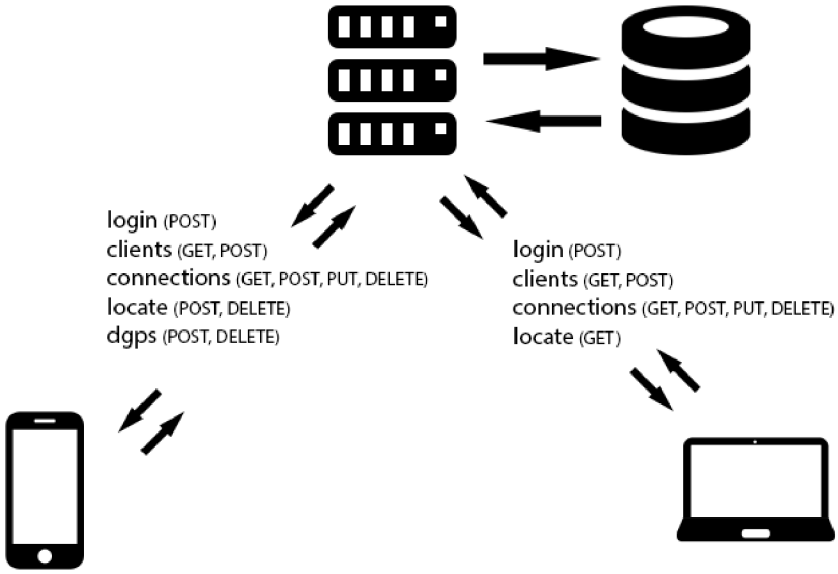
7.2.2. Klientská část

Klientská část sestává z mobilní a webové aplikace. Klienti se budou serveru dotazovat na data a tato data budou následně zobrazovat. Mobilní aplikace pak bude pomocí metod

rozhraní k tomu určených odesílat na server své pozice. Obě klientské aplikace umožní uživatelům registraci a přihlášení. Po úspěšném přihlášení obdrží JWT token, který uloží do lokálního úložiště *LocalStorage*, kde budou tato data přístupná i po případném restartu aplikace.

7.3. Návrh webového aplikačního rozhraní

Pro komunikaci mezi jednotlivými částmi informačního systému je třeba navrhnout a naimplementovat veřejné webové aplikační rozhraní, dále jen API. Toto API se bude držet REST architektury, která je popsána v kapitole 5.3. Rozhraní bude bezstavové. API nebude implementovat všechny metody HTTP protokolu, ale pouze ty, které budou aplikacemi opravdu používány (více v kapitole 7.3.1). Metody mohou očekávat GET a POST parametry, tyto parametry mohou být povinné a nebo volitelné. Server bude na každý dotaz korektně reagovat podle toho, zda byl schopen požadavek vyřídit nebo došlo k chybě. V případě chyby bude klientovi odesílat také její popis, aby na ni mohl klient správně reagovat. Způsob užití API jednotlivými komponentami systému je zobrazen na obrázku 7.2.



Obrázek 7.2: Návrh webového aplikačního rozhraní

Vzhledem k tomu, že data jsou přístupná pouze autorizovaným uživatelům, je potřeba implementovat uživatelskou autentizaci. K ověření identity uživatele poslouží nejprve uživatelské jméno a heslo, po úspěšném přihlášení pak JWT token s omezenou platností (byl popsán v kapitole 5.5), který bude od klientů na server přenášen v HTTP hlavičce.

7.3.1. Metody aplikačního rozhraní

V této kapitole budou popsány jednotlivé služby webového API včetně jejich URL, POST a GET parametrů a HTTP metod s popisem jejich užití.

Metoda API *login* ověřuje totožnost uživatele podle uživatelského jména a hashe jeho hesla, tudíž jako jediná nevyžaduje JWT token k autentizaci uživatele. Token uživatel obdrží až v odpovědi serveru na tuto metodu. Na dotaz klienta server odpoví HTTP stavovým kódem 200 v případě úspěchu a odešle klientovi JWT token, v opačném případě odpoví kódem 400.

login

URL:
`<hostname>/interface/login`

Metody:
`POST` - slouží k přihlášení uživatele
`POST parametry`
`username` - uživatelské jméno
`password` - hash hesla

V případě všech následujících služeb musí být uživatel přihlášen a musí disponovat JWT tokenem. Tento token odešle v požadavku na server jako součást HTTP hlavičky. V opačném případě mu server odpoví stavovým kódem `401 Unauthorized` a klient neobdrží požadovaná data. Pokud na serveru proběhne všechno v pořádku, odpoví server klientovi odpovědí s HTTP stavovým kódem 200 a doručí požadovaná data. V opačném případě odešle odpověď se stavovým kódem 400.

clients

URL:
`<hostname>/interface/clients`

Metody:
`GET` - slouží k získání seznamu uživatelů
`GET parametry`
`connections` - přidá informace o stavu spojení
`coords` - přidá aktuální souřadnice každého spojeného klienta
`POST` - slouží k registraci uživatele
`POST parametry`
`username` - uživatelské jméno
`password` - hash hesla

connections

URL:

`<hostname>/interface/connections`

Metody:

`GET` - slouží k získání všech spojení, žádostí o spojení a odeslaných žádostí o spojení

`POST` - klient odešle žádost o spojení jinému uživateli

`POST parametry`

`_id` - unikátní identifikátor uživatele

`username` - uživatelské jméno

`PUT` - klient přijme žádost o spojení od jiného uživatele

`POST parametry`

`connectionId` - unikátní identifikátor spojení

`DELETE` - klient odstraní žádost jiného uživatele

`GET parametry`

`connectionId` - unikátní identifikátor spojení

locate

URL:

`<hostname>/interface/locate`

Metody:

`GET` - slouží k získání souřadnic aktuálního uživatele (historie pozic)

`GET parametry`

`from` - časové razítko okamžiku, od kterého uživatel žádá historii pozic

`to` - časové razítko okamžiku, do kterého uživatel požaduje historii pozic

`POST` - slouží k odeslání údajů o pozici klienta

`POST parametry`

`timestamp` - časové razítko zjištění pozice

`latitude` - zeměpisná šířka

`longitude` - zeměpisná délka

`accuracy` - přesnost lokalizace

`GET parametry`

`dgps` - identifikátor DGPS stanice pro odběr korekcí - nepovinný parametr

`DELETE` - server nastaví klienta offline

dgps

URL:

`<hostname>/interface/dgps`

Metody:

`POST` - slouží k odeslání pozice stanice DGPS

`POST parametry`

`timestamp` - časové razítko zjištění pozice

`latitude` - zeměpisná šířka

`longitude` - zeměpisná délka

`DELETE` - vyrozumí server o vypnutí režimu stacionární stanice DGPS

7.4. Návrh zpřesnění lokalizace systémem DGPS

Princip diferenciální GPS je popsán v kapitole 2.4.1. Pro aplikaci DGPS v implementovaném systému je potřeba vyřešit několik problémů. Prvním problémem je, jak získat referenční souřadnice vzhledem ke kterým budou počítány korekce. Tento problém je ve výsledné aplikaci řešen tak, že stanice umístěná na neměnné pozici bude referenční pozici počítat jako dlouhodobý průměr naměřených zeměpisných souřadnic a po určité dostatečně dlouhé době bude tato průměrná hodnota akceptována za referenční.

Dále je třeba zajistit, že se referenční DGPS stanice nepohybuje, nemění svoji pozici. Tento problém je řešen akcelerometrem, který je součástí většiny dnešních mobilních zařízení a je schopen detekovat zrychlení ve všech směrech, tedy zahájení pohybu.

Aplikace korekcí na souřadnice ostatních stanic probíhá na serveru. DGPS stanice prostřednictvím API odesílá aktuální korekce zároveň se svojí polohou na server a ten je následně poskytuje ostatním klientům, kteří se o ně přihlásí.

7.5. Návrh databáze

V implementovaném systému je použit databázový systém MongoDB s frameworkem Mongoose (popsána v kapitole 5.2). Tato databáze je na rozdíl od častěji se vyskytujících relačních SQL databází databáze *dokumentově-orientovaná*, data jsou v databázi uloženy ve formě dokumentů. Pro každý dokument lze definovat takzvaný *model*, což je prakticky vzor sloužící k vytváření dokumentů následně vkládaných do databáze. Určitou podobnost modelů lze nalézt u tříd v objektově orientovaných jazycích. Každý dokument je pak analogií instance třídy, tedy objektu. Model definuje strukturu dokumentu. Na modelu lze také definovat metody. Každý dokument obsahuje unikátní identifikátor vygenerovaný databázovým systémem.

Dokumenty nejsou v databázi ukládány v tabulkách jako u relačních databází, ale v kolekcích. Podobně jako v relačních databázích lze k výsledku dotazu nad určitou kolekcí připojit data z jiných kolekcí. V relačních databázích ke spojení tabulek slouží klíčové slovo *JOIN* a jeho specifičtější alternativy. V dokumentově orientovaných databázích k tomu slouží mechanismus „Query population“. Pomocí mechanismu populace lze dokument rozšířit o dokumenty z jiných kolekcí. Lze také specifikovat, které dokumenty požadujeme a jejich počet, řazení.

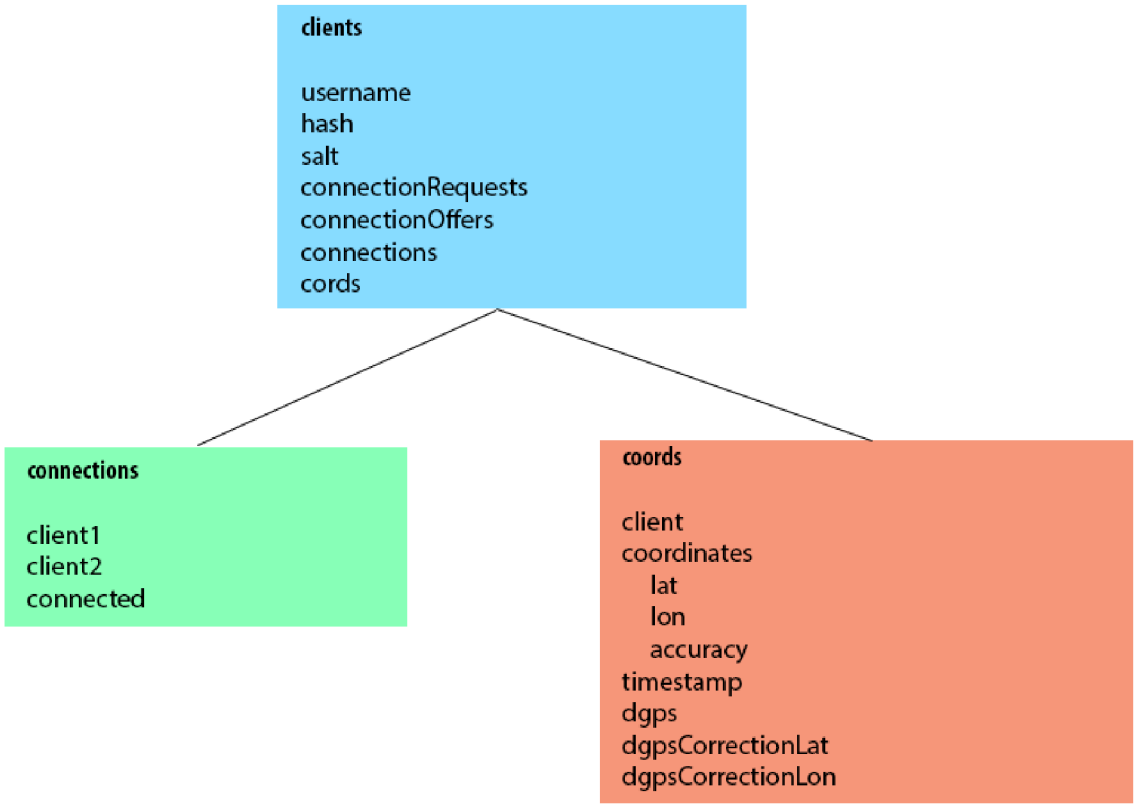
Pro účely ukládání dat v databázi byly navrženy modely zobrazené na obrázku 7.3.

clients Model *clients* definuje datovou strukturu k uložení uživatelů, obsahuje jeho uživatelské jméno, hash hesla a salt. Salt neboli sůl je určena ke zvýšení bezpečnosti, ztěžuje případnému útočníkovi získání hesla hrubou silou. Dalšími parametry modelu clients jsou *connectionRequests*, *connectionOffers* a *connections*. Tyto parametry vyjadřují odeslané žádosti o spojení ostatním uživatelům, žádosti od ostatních uživatelů a již akceptovaná spojení, prakticky obsahují pouze pole identifikátorů jednotlivých spojení a mechanismem populace je při získávání dat z databáze lze rozšířit o konkrétní spojení. Posledním parametrem je *coords*, ten obsahuje pole

identifikátorů jednotlivých souřadnic. Identifikátory souřadnic lze také mechanismem populace rozšířit o konkrétní souřadnice.

connections Model *connections* představuje datovou strukturu pro jednotlivá spojení. Obsahuje identifikátory zúčastněných uživatelů a příznak značící stav spojení (zda byla žádost akceptována). Mechanismem populace lze identifikátory rozšířit přímo o dokumenty uživatelů a získat tak jedním dotazem na databázi i informace o nich.

coords Model *coords* je datová struktura určená k uložení souřadnic. Obsahuje identifikátor klienta, který souřadnice odeslal, souřadnice a čas zaznamenání souřadnic. Dále obsahuje také příznak, zda jsou k dispozici korekce DGPS a vlastní korekce.



Obrázek 7.3: Návrh databáze

7.6. Návrh uživatelského rozhraní

Uživatelské rozhraní bylo navrženo s ohledem na maximální jednoduchost a přehlednost. Díky použití frameworku AngularJS (který je popsán v kapitole 5.4), bylo dosaženo dynamického chování aplikací, všechny změny v datové části jsou ihned propagovány do uživatelského rozhraní bez nutnosti opětovného načtení celé stránky.

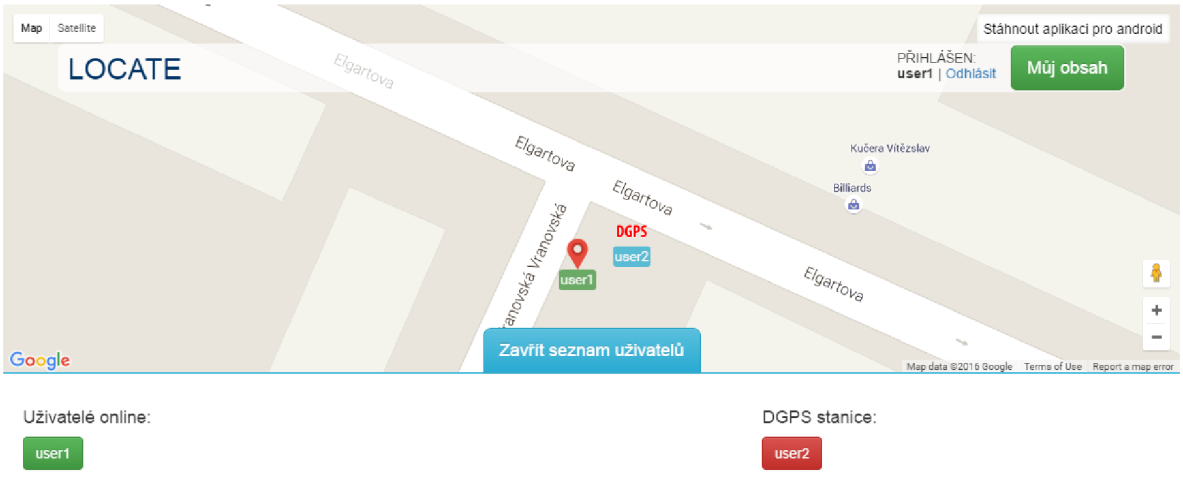
7.6.1. Uživatelské rozhraní webové aplikace

Rozhraní webové aplikace bylo rozděleno na dvě části a to na *Mapa* a *Můj obsah*. V obou částech má uživatel možnosti přihlášení, registrace a v případě že je přihlášen se může odhlásit. K přepínání mezi těmito částmi slouží tlačítko, toto tlačítko je znázorněno v pravém horním rohu obrázku 7.4. Pokud uživatel přihlášen není, postrádá možnost přechodu do uživatelské sekce smysl, a proto je v takovém případě tlačítko skryto.

Mapa

Obrázek 7.4 zobrazuje sekci *Mapa*. Hlavním účelem této sekce je zobrazování klientských stanic na mapě, z toho důvodu mapa zabírá většinu prostoru obrazovky. Stanice uživatelů, které odesílají své pozice na server, jsou na mapě zobrazeny prostřednictvím *markerů*, které představují jejich aktuální pozici. Součástí těchto markerů je i uživatelské jméno náležící ke stanici. Kliknutím na marker stanice se otevře detail obsahující podrobnosti o její poloze. Je-li zobrazená stanice zároveň v režimu stacionární stanice DGPS, je zvýrazněna odlišnou ikonou markeru. Uživatel může mapu ovládat pomocí nativních ovládacích prvků umožňujících přepnutí mezi režimy mapy, přiblížení a oddálení.

Obrazovka *Mapa* obsahuje také vysunovací nabídku sloužící ke zobrazení stanic zobrazených na mapě. Stanice v této nabídce jsou rozděleny na běžné uživatele a DGPS stacionární stanice. Po kliknutí na konkrétní stanici je mapa vycentrována na její pozici.



Obrázek 7.4: Uživatelské rozhraní webové aplikace

Můj obsah

Sekce *Můj obsah* je rozdělena na dvě podsekce. Jednou z nich je *Má spojení* sloužící ke správě spojení mezi přihlášeným uživatelem a ostatními uživateli. Druhou je podsekce *Informace o polohách*, která slouží ke zobrazení historie poloh a grafu přesností.

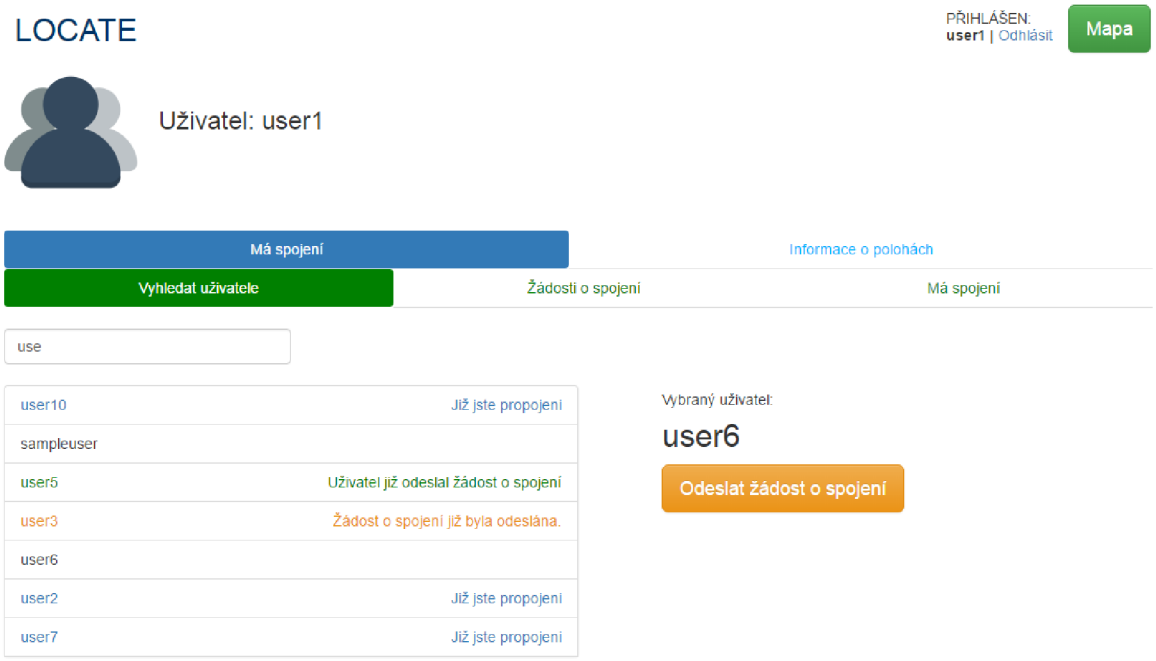
Správa spojení je zobrazena na obrázku 7.5. V této části je zobrazen seznam uživatelů uložených v databázi spolu se stavem spojení ve vztahu k aktuálnímu uživateli. Zadáním textového řetězce do formulářového pole nad seznamem uživatelů lze vyhledávat konkrétního uživatele nebo podmnožinu uživatelů, jejichž uživatelské jméno odpovídá vyhledávanému řetězci. Při zadávání řetězce je seznam ihned automaticky aktualizován. Po

kliknutí na konkrétního uživatele je v pravé straně obrazovky zobrazen detail a tlačítko s možností akce vycházející z aktuálního stavu spojení.

Jednotlivé stavy spojení a odpovídající akce mohou být:

1. Uživatelé nejsou spojeni - přihlášený uživatel může odeslat žádost o spojení.
2. Přihlášený uživatel již odeslal žádost o spojení - může tuto žádost zrušit.
3. Přihlášený uživatel již obdržel žádost o spojení - může tuto žádost akceptovat.
4. Uživatelé jsou již spojeni - přihlášený uživatel může spojení zrušit.

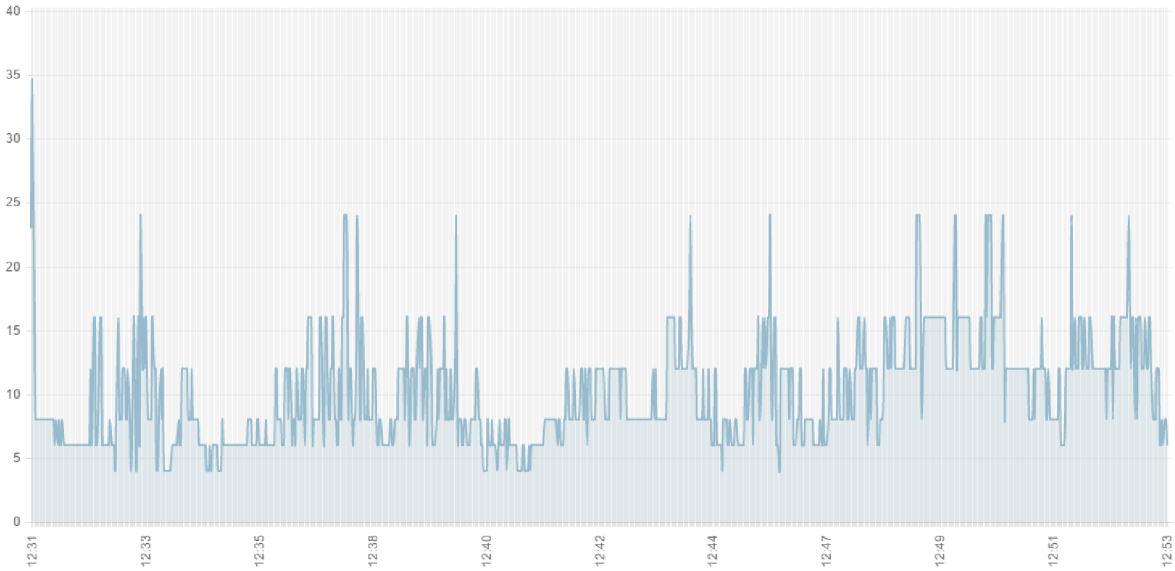
Správa spojení obsahuje podnabídku umožňující zobrazení všech uživatelů, dále uživatelů, kteří odeslali žádost o spojení (spojení čekajících na schválení) a aktivních spojení.



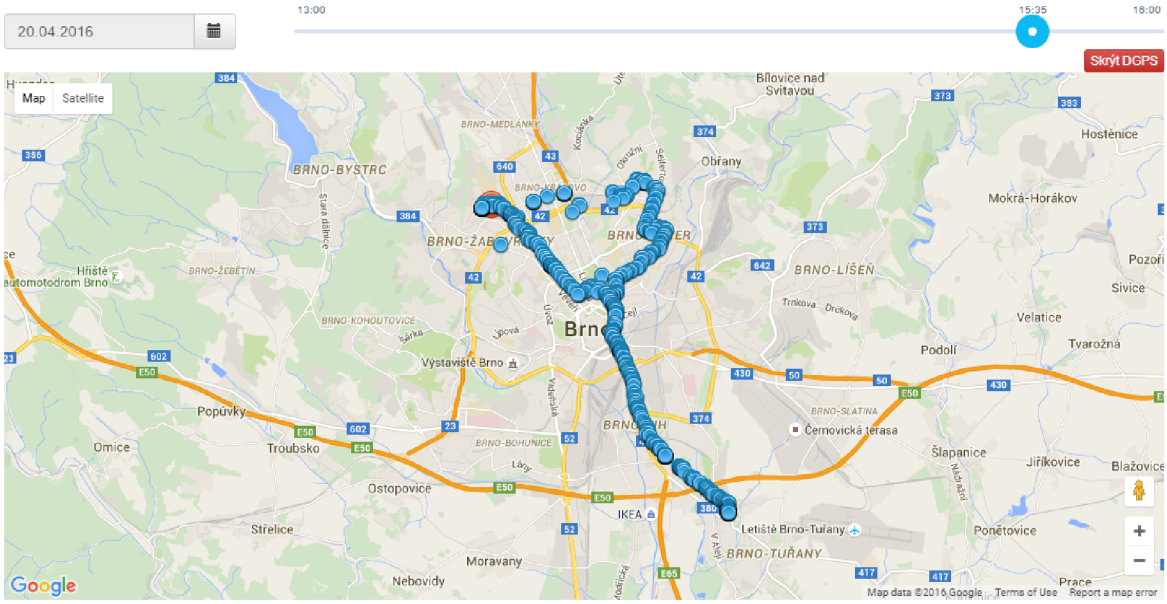
Obrázek 7.5: Správa spojení

Podsekce *Informace o polohách* obsahuje historii poloh a graf přesností. Na obrazovce grafu přesností může uživatel zvolit den, ze kterého mají být použita data. Tato data jsou pak vynesena do grafu, který zobrazuje jednotlivé přesnosti v čase. Tento graf je zobrazen na obrázku 7.6.

Historie poloh je zobrazena na obrázku 7.7. Podobně jako v případě grafu přesností je na stránce historie poloh možnost zvolit den pro zobrazení historie. Jednotlivé souřadnice poloh v historii jsou zobrazeny na mapě. Prostřednictvím tlačítek *Skrýt DGPS* a *Zobrazit DGPS* může uživatel zvolit, zda mají být zobrazeny souřadnice po aplikaci korekcí DGPS (v případě, že jsou tyto korekce k dispozici) a nebo zda mají být zobrazeny původní souřadnice. Vedle volby data se nachází časová osa. Posunem jezdce po časové ose je v mapě zobrazována nejbližší pozice odpovídající údaji na časové ose. Krajní hodnoty této osy jsou upraveny podle toho, ve kterém časovém intervalu byly souřadnice získány.



Obrázek 7.6: Graf přesností

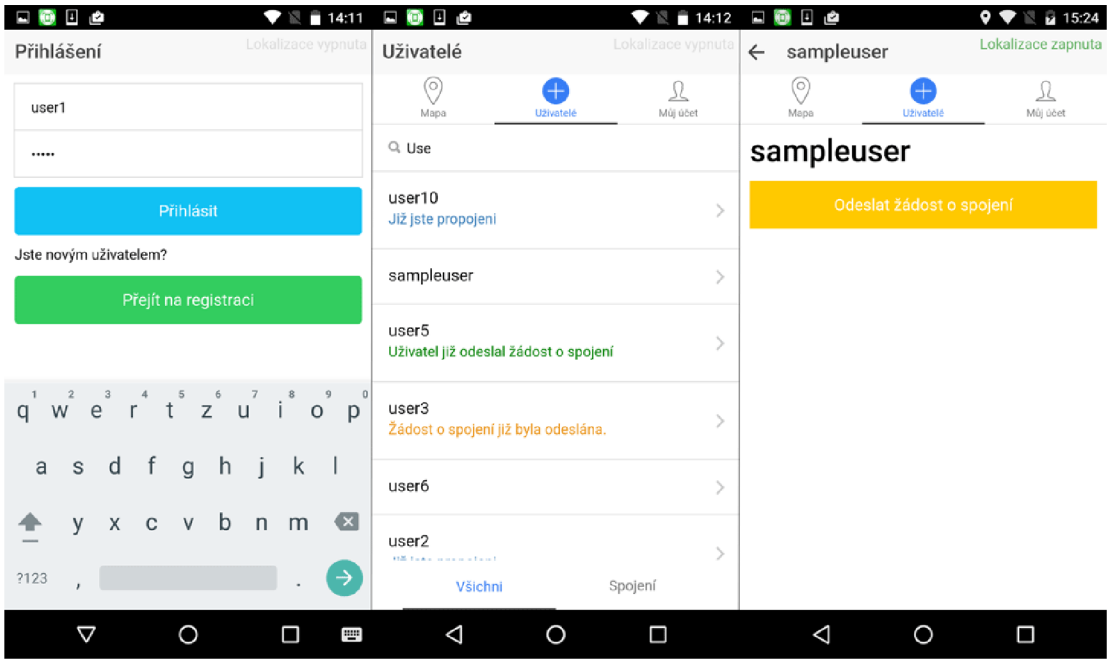


Obrázek 7.7: Historie poloh

7.6.2. Uživatelské rozhraní mobilní aplikace

Jednotlivé obrazovky aplikace jsou zobrazeny na obrázcích 7.8 a 7.9. Při prvním spuštění aplikace se uživatel musí přihlásit nebo registrovat, bez přihlášení nelze pokračovat. Po úspěšném přihlášení je zobrazena mapa a nabídka, jejíž jednotlivé položky jsou *Mapa*, *Uživatelé* a *Můj účet*. Tato nabídka je napříč celou aplikací neměnná.

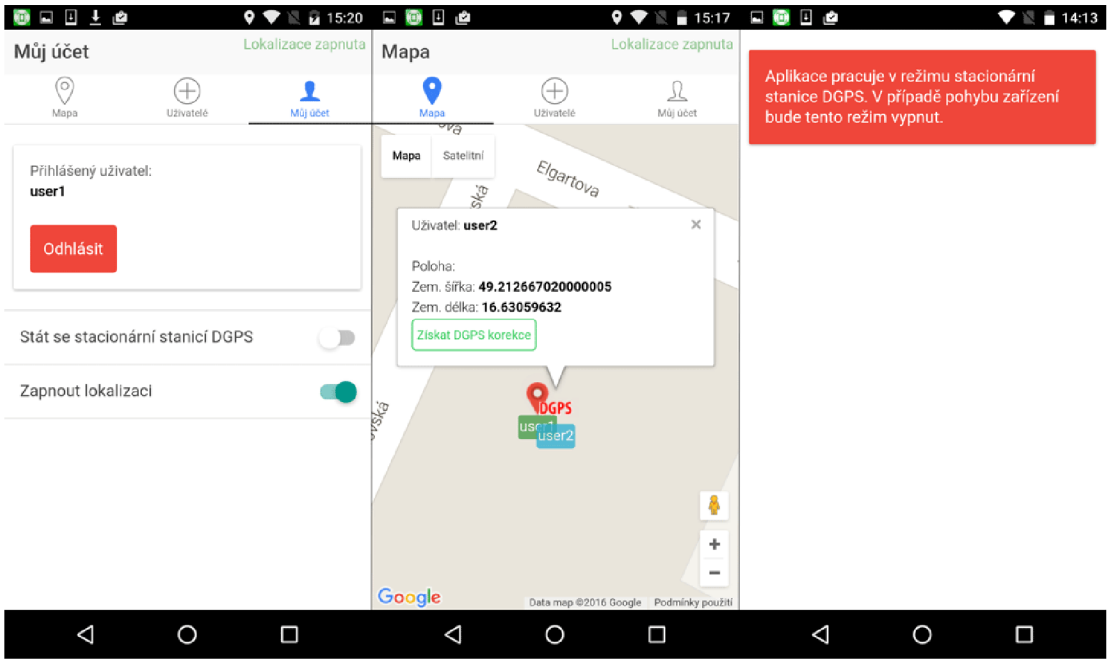
Zvolením položky *Uživatelé* z menu aplikace je otevřen seznam ostatních uživatelů, který je znázorněn na prostřední obrazovce na obrázku 7.8. Aktuálně přihlášený uživatel může mezi ostatními uživateli vyhledávat a spravovat spojení stejným způsobem, jaký byl popsán v návrhu webové aplikace v kapitole 7.6.1. Po kliknutí na konkrétního uživatele je otevřena stránka s detailem, zobrazena vpravo na obrázku 7.8.



Obrázek 7.8: Mobilní aplikace: Správa uživatelů

Další částí mobilní aplikace je *Mapa*, na které jsou pomocí markeru zobrazeny klientské stanice stejným způsobem, jako je popsáno v návrhu webové aplikace v kapitole 7.6.1. Po kliknutí na klientskou stanici je otevřen detail stanice, v případě, že je stanice zároveň v režimu stacionární stanice DGPS, je v detailu zobrazeno tlačítko „Získat DGPS korekce“ viz obrázek 7.9. Kliknutím na toto tlačítko aktuální uživatel začne od zvolené DGPS stanice odebírat korekce a upravovat dle nich informace o své poloze.

Sekce *Můj účet* je zobrazena na obrázku 7.9. V této sekci se nachází tlačítko sloužící k odhlášení aktuálně přihlášeného uživatele. Dále jsou zde dva přepínače. Jedním z nich je přepínač pro zapnutí samotné lokalizace. Tento přepínač v aplikaci spustí proces získávání své vlastní polohy a odesílání této polohy na server. Druhý přepínač slouží k zapnutí režimu stacionární stanice DGPS. Po přepnutí tohoto přepínače do aktivní polohy je uživateli zobrazen varovný dialog informující o tom, že v režimu stacionární stanice DGPS nemá se zařízením pohybovat. Po potvrzení tohoto dialogu je uživateli skryt veškerý obsah a zobrazí se pouze textové pole informující o aktuálním režimu, jak je zobrazeno vpravo na obrázku 7.9. Je také spuštěn proces výpočtu DGPS korekcí. Při detekci pohybu zařízení je tento režim automaticky vypnut.



Obrázek 7.9: Mobilní aplikace: Mapa a nastavení

8. Implementace

8.1. Implementace serverové části

Jak bylo navrženo v 7.2.1, server bude sloužit jako úložiště dat a jako komunikační prostředek mezi serverovou a klientskou částí.

Pro implementaci serveru bylo zvoleno prostředí *Node.JS*, které je popsáno v kapitole 5.1 a to kvůli tomu, že jeho programovacím jazykem je JavaScript. Celý projekt díky tomu bude z pohledu programovacího jazyka jednotný. Dále pak pro jeho rychlost a jednoduchou škálovatelnost. Node.JS není zbytečně těžkopádný jako některé konkurenční projekty. Mimo jiné další pozitivní vlastnosti v něm lze velmi elegantně tvořit aplikační rozhraní.

Pro realizaci databáze byl zvolen databázový systém *MongoDB*. Koncept dokumentově-orientované databáze je velmi vhodný pro implementaci tohoto projektu, protože díky němu není třeba měnit datovou strukturu pro uložení v databázi a pro použití na klientské straně. Pro zjednodušení a zefektivnění práce s MongoDB databází jsem využil framework *mongoose*.

Ověřování identity uživatele na straně serveru je řešeno s využitím autentizační mezivrstvy *Passport.js* využívající technologii *JSON Web Token* popsanou v kapitole 5.5. Logika výpočtu DGPS korekcí je taktéž řešena na serveru.

8.1.1. Implementace webového aplikačního rozhraní

Jednotlivé metody aplikačního rozhraní jsou přístupné klientským aplikacím prostřednictvím objektu *express.Router()*, který poskytuje framework *express*. Každá metoda API tak má definovánu URL a v případě, že je odeslán dotaz na server s touto konkrétní URL, server provede kód pro danou metodu.

Příklad definice metody GET rozhraní:

```
var router = express.Router();
var jwt = require('express-jwt');
var auth = jwt({secret: 'LOCATESECRET', userProperty: 'user'});

router.get('/interface/locate', auth, function(req, res) {
   //nastavení potřebných hlaviček
   res.header("Access-Control-Allow-Origin", "usedDomainName");
   res.header("Access-Control-Allow-Headers", "Content-Type");
   if(vseProbehloVporadku){
      res.writeHead(200, {"Content-Type": "application/json"});
      //odeslání dat
      response.end(JSON.stringify(data))
   }else if(nastalProblem){
      res.writeHead(400, {"Content-Type": "application/json"});
      //odeslání chybové hlášky
      response.end(JSON.stringify({message: "Nastala chyba"}))
}});
```

V případě, že operace na serveru proběhne úspěšně, server odešle uživateli HTTP odpověď se stavovým kódem 200 a data ve formátu JSON. V opačném případě odešle odpověď se stavovým kódem 400 a chybovou hlášku. Operací na serveru může být míněno například získání dat z databáze, nebo uložení dat do databáze. Důvodem k použití formátu JSON (JavaScript Object Notation) je to, že jak serverová, tak i klientská část systému je psána v JavaScriptu a lze tak data rovnou použít. Není třeba měnit jejich datovou strukturu.

8.1.2. Práce s databází

Pro ukládání dat do databáze bylo třeba nejdříve nadefinovat schémata, která následně slouží jako vzor pro každý záznam (jedná se o speciální JavaScriptové pseudotřídy). Každý záznam vkládaný do databáze pak musí odpovídajícímu schématu korespondovat, v opačném případě nebude vložen.

Příklad definice schématu definujícího datovou strukturu pro souřadnice:

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var coordsSchema = new Schema({
    client: { type: Schema.Types.ObjectId, ref: 'clients' },
    coordinates: {
        lat: {type: Schema.Types.Mixed, default: null},
        lon: {type: Schema.Types.Mixed, default: null},
        accuracy: {type: Schema.Types.Mixed, default: null}
    },
    timestamp: Number,
    dgps: {type: Boolean, default: false},
```

```
    dgpsCorrectionLat: {type: Schema.Types.Mixed, default: null},
    dgpsCorrectionLon: {type: Schema.Types.Mixed, default: null}
});
mongoose.model('coords', coordsSchema);
```

Databáze MongoDB umožňuje stejně jako jiné databázové systémy záznamy v databázi vyhledávat, do databáze vkládat a záznamy mazat.

Vyhledávání je v MongoDB velmi efektivní. Pro vyhledávání je v databázi k dispozici metoda `find()`, která jako svůj parametr přebírá objekt, který specifikuje vyhledávání. Tato metoda vrací pole záznamů splňujících daná kritéria. Kromě této metody existuje také metoda `findOne()`, která vrátí první záznam splňující vyhledávací kritéria. Mimo vyhledávací kritéria lze metodě `find()` předat i seznam parametrů, které z databáze ke každému záznamu požadujeme.

Do dotazu lze také přidat porovnání s určitou hodnotou. Výsledky lze řadit, omezit jejich množství a další.

Příklad vyhledání záznamu v databázi:

```
var coords = mongoose.model('clients');
var searchObject = {
    client: userId,
    timestamp:{
        $gte: timestampFrom,
        $lte: timestampTo
    }};
Coords.find(searchObject, 'coordinates timestamp',
    function (err, coords) {
        //callback pro zpracování výsledků dotazu na databázi
    }).limit(5).sort({timestamp:-1});
```

Přidávání záznamu probíhá tak, že je nejdříve vytvořena instance třídy schématu a na té je následně zavolána metoda `save()` jak je vidět na príkladu níže.

```
var Client = mongoose.model('clients');
var newClient = new Client();
newClient.username = username;
newClient.setPassword(password);
newClient.save();
```

Mazání záznamů probíhá prostřednictvím metody `remove()`, která podobně jako metoda `find()` má parametr, kterým je objekt definující vyhledávací kritéria. Dále existuje metoda `findOneAndRemove()`, která kromě smazání záznamu ještě vrátí data.

Příklad mazání dat v databázi:

```
mongoose.model('clients').remove({_id: userId});
```

8.1.3. Autentizace uživatelů

Autentizace uživatele je na straně serveru řešena mezivrstvou *Passport.js*. K autentizaci je nejdříve potřeba, aby byl uživatel zaregistrovaný. K registraci slouží */interface/clients* metoda aplikačního rozhraní volaná HTTP metodou POST. Klient při registraci odešle na server v POST parametrech uživatelské jméno a MD5 hash hesla. Schéma pro uložení uživatele do databáze vypadá následovně:

```
var clientsSchema = new Schema({
    username: String,
    hash: String,
    salt: String,
    connectionRequests: [{type: Schema.Types.ObjectId, ref: 'connections'}],
    connectionOffers: [{type: Schema.Types.ObjectId, ref: 'connections'}],
    connections : [{type: Schema.Types.ObjectId, ref: 'connections'}],
    coords: [{type: Schema.Types.ObjectId, ref: 'coords'}]
});
```

Před vložením záznamu do databáze je třeba vygenerovat nový hash hesla, který bude uložen v databázi spolu se *salt*, což je mechanismus sloužící ke zvýšení zabezpečení proti útokům hrubé síly. Ke generování těchto údajů slouží metoda *setPassword* definovaná přímo na schéma pro uložení uživatele. Takto definovaných metod je více a jsou následně dostupné kdykoliv na každé instanci schéma, tedy na každém objektu klienta. Další metody budou popsány v části věnující se přihlášení uživatele.

Metoda *setPassword* :

```
clientsSchema.methods.setPassword = function(password){
    this.salt = crypto.randomBytes(16).toString('hex');
    this.hash = crypto.pbkdf2Sync(password, this.salt, 1000, 64)
    .toString('hex');
};
```

Funkce setPassword generuje *hash* hesla pomocí modulu *crypto*, který je přímo součástí prostředí *Node.js*. Ke generování *hash* je použita synchronní derivační funkce *pbkdf2Sync*. Tato funkce používá hašovací funkci *SHA-256*. K vytvoření *salt* je použita funkce *randomBytes* stejného modulu, která generuje pseudonáhodný řetězec.

Po vygenerování *hash* a *salt* může server tato data uložit do databáze a uživatel se může přihlásit. K přihlášení uživatele slouží metoda API popsaná v 7.3.1. Při volání této metody je použita mezivrstva *passport*, konkrétně její funkce *authenticate*, která volá některou z definovaných lokálních strategií, v našem případě je to strategie *login*.

Lokální strategie *login* je prakticky funkce, která dotazem na databázi zkontroluje, zda je uživatel, který se chce přihlásit, registrován. Pokud je registrován, zavolá metodu *validPassword* na objektu *client*, která je definovaná podobně jako *setPassword*. Tato metoda porovná *hash* hesla použitého pro přihlášení s *hash* uloženým v databázi.

Metoda *validPassword*:

```
clientsSchema.methods.validPassword = function(password){
    var hash = crypto.pbkdf2Sync(password, this.salt, 1000, 64)
```

```
    .toString('hex');
    return this.hash === hash;
};
```

Poté v případě, že porovnání proběhne úspěšně, je vygenerován JWT token a ten je odeslán zpět uživateli. Ke generování JWT tokenu slouží další metoda definovaná na objektu *client* a tou je metoda *generateJWT*.

Metoda *generateJWT* :

```
    // nastaví dobu expirace tokenu na dva dny
    var today = new Date();
    var exp = new Date(today);
    exp.setDate(today.getDate() + 2);

    return jwt.sign({
        _id: this._id,
        username: this.username,
        exp: parseInt(exp.getTime() / 1000),
    }, 'LOCATESECRET');
```

Takto vygenerovaný JWT token následně odesílá v hlavičce každého HTTP požadavku na server webová i mobilní aplikace.

K zabezpečení kterékoliv metody API pak slouží funkce *jwt* modulu *express-jwt*, jejíž použití lze vidět na ukázce v kapitole 8.1.1. Tato funkce již sama zajistí odeslání odpovědi s HTTP stavovým kódem **401** informujícím uživatele o pokusu o neautorizovaný přístup ke službě v případě, že dotaz na server neobsahuje platný JWT token.

8.1.4. Diferenciální GPS

Server má na starost také funkcionalitu diferenciální GPS. Pro lokální reprezentaci DGPS stanic byla implementována JavaScriptová pseudotřída *DGPSStation*. V případě, že některá z klientských stanic zahájí režim stacionární stanice DGPS, je pro ni vytvořena instance této třídy.

Konstruktor třídy *DGPSStation*:

```
var DGPSStation = function (userId) {
    this._id = userId;
    this.averagePosition = {
        lat: null,
        lon: null
    };
    this.counter = 0;
    this.lastPosition = null;
    this.sumPosition = {
        lat: 0,
        lon: 0
    };
};
```

Každá instance třídy obsahuje metody *learnPosition* a *getCorrections*. Metoda *learnPosition* průběžně počítá průměrnou pozici stanice. Tato pozice je následně považována za referenční. Zároveň ukládá poslední známou pozici stacionární DGPS stanice. Poté lze prostřednictvím metody *getCorrections* získat aktuální korekce, které představují rozdíl mezi referenční a poslední pozicí DGPS stanice. Pokud je poslední pozice nepřesná o více než 40 metrů, je toto měření považováno za měření s příliš velkou chybou a není zahrnuto do průměru souřadnic. Není tedy použito k výpočtu korekcí. Z obrázku 7.6 je vidět, že přesnost je většinou lepší.

Metoda *learnPosition*:

```
DGPSStation.prototype.learnPosition = function (position) {
    if(!(position.latitude && position.longitude)) return;
    if(position.accuracy < 100){
        this.counter++;
        //součet všech pozic
        this.sumPosition.lat += parseFloat(position.latitude);
        this.sumPosition.lon += parseFloat(position.longitude);
        //výpočet průměrné pozice
        this.averagePosition.lat = this.sumPosition.lat / this.counter;
        this.averagePosition.lon = this.sumPosition.lon / this.counter;
        //poslední uložená pozice
        this.lastPosition = position;
    }
};
```

Metoda *getCorrections*:

```
DGPSStation.prototype.getCorrections = function () {
    return {
        correctionLat: (this.averagePosition.lat
        - parseFloat(this.lastPosition.latitude)),
        correctionLon: (this.averagePosition.lon
        - parseFloat(this.lastPosition.longitude))
    };
};
```

K odesílání souřadnic určených k výpočtu korekcí slouží stacionárním DGPS stanicím metoda aplikačního rozhraní *dgps*.

8.2. Implementace webové aplikace

Webová aplikace je naimplementována ve frameworku AngularJS. Tento JavaScriptový framework byl zvolen, protože je velmi vhodný pro dynamické aplikace s často se měnícím obsahem. Hlavními stavebními prvky Angular.JS jsou *Services*, *Controllers* a *Directives*.

8.2.1. Angular services

Angular Services neboli česky servisy jsou JavaScriptové objekty sloužící primárně jako zdroj dat a lze je kdekoliv napříč aplikací vyžádat mechanismem „dependency injection“. Tyto objekty jsou inicializovány až v momentě, kdy jsou vyžádány, a proto nezpomalují počáteční inicializaci aplikace. Této vlastnosti se říká *lazy-loading*. Jedná se o singletony, což v tomto případě znamená, že ať jsou pomocí *dependency injection* vyžádány kdekoliv, jedná se vždy o tutéž instanci [2]. Z tohoto důvodu jsou vhodné pro sdílení prostředků a společné funkcionality napříč aplikací.

V implementované webové aplikaci jsou celkem čtyři servisy. První tři z nich se týkají autentizace uživatele, což bude rozvedeno v kapitole 8.2.5. Čtvrtou je *webAPIService* která má na starosti veškerou komunikaci se vzdáleným serverem a jeho API.

8.2.2. Angular controllers

Controllers, česky kontrolery, jsou Javascritové konstruktory, které jsou primárně určeny pro rozšiřování funkcionality *$scope*. *$scope* je objekt, který je referencí na aplikační model, na aktuální data prezentovaná uživateli [2]. *$scope* slouží ke zpracovávání uživatelských akcí a k implementaci reakcí na tyto akce a mimo jiné poskytuje metody *$watch* pro rozpoznávání mutací modelu a *$apply* k propagaci změn modelu, které proběhnou mimo Angular, do Angularu.

V implementované aplikaci jsou dva kontrolery, jedním z nich je *mapCtrl*, který v definovaném intervalu volá *webAPIService* a tak získává data, která potom prezentuje uživateli. Dále je v tomto kontroleru implementována veškerá logika týkající se manipulace s mapou.

Druhým kontrolerem je *userCtrl*, který má na starosti část aplikace *Můj obsah* popsanou v kapitole 7.6.1. Je v něm implementována správa spojení a veškerá logika s tím spojená.

8.2.3. Angular directives

Direktivy jsou mechanismus k rozšíření běžné HTML syntaxe o vlastní prvky s vlastní zapouzdřenou funkcionalitou. Každá direktiva může mít vlastní HTML kód, kontroler a může využívat ostatních komponent aplikace pomocí *dependency injection*.

Ve webové aplikaci jsou implementovány direktivy pro hlavní menu, graf přesností, historii poloh a otevírací seznam uživatelů na stránce s mapou. Jako příklad uvedu direktivu pro horní menu. Direktiva pro zobrazení a funkci horního menu se jmenuje *topMenu* a její JavaScriptový kód vypadá následovně:

```
angular.module('locate').directive('topMenu',
    function(authenticationService, loginRegisterService, $location) {
    return {
        templateUrl: 'templates/directives/topMenuDirective.tpl.html',
        link: function(scope, element, attrs){

        //vlastní logika
        }
    }});
```

Soubor *topMenuDirective.tpl.html* obsahuje HTML kód, kterým bude rozšířen původní kód v místě vložení direktivy. Direktiva je následně použita jednoduše tímto způsobem:

```
<div top-menu></div>
```

8.2.4. Komunikace se serverem

Aplikace komunikuje s aplikačním rozhraním serveru prostřednictvím standardních HTTP požadavků. Kód pro volání požadavků na server je implementován ve *webAPIService* a využívá interní servisu angularu *$http*. Servisa *webAPIService* implementuje také reakce na chybové odpovědi serveru.

Příklad kódu pro odeslání požadavku na webový server, konkrétně pro registraci uživatele:

```
$http.post('/interface/clients', userCredentials)
            .success(function(resp){
                //úspěch
            }).error(function(err){
                //ošetření chyby
            });
```

8.2.5. Autentizace uživatele

Uživatel se může registrovat, přihlásit a odhlásit. Autentizace na straně klienta je řešena pomocí tří servis. Servisa *loginRegisterService* má na starosti správu dialogů pro přihlášení, registraci a odhlášení. Je závislá na *authenticationService*, která poskytuje metody k přihlášení, registraci. Ta je závislá na *authenticationTokenService* a *webAPIService*. Servisa *authenticationTokenService* má na starost práci s JWT tokenem.

Registrace probíhá tak, že je po kliknutí na tlačítko registrace otevřen registrační dialog s formulářem prostřednictvím metody v *loginRegisterService*. Po potvrzení jsou odpovídající metodou *authenticationService* odeslána uživatelská data na server. V případě úspěšné registrace je nový uživatel automaticky přihlášen.

V případě přihlášení je stejně jako u registrace otevřen dialog s formulářem. Po potvrzení dialogu je volána odpovídající metoda *authenticationService*. V případě úspěšného přihlášení obdrží aplikace od serveru JWT token, ten je servisou *authenticationService* předán *authenticationTokenService*. Ta token uloží. Pro uložení tokenu je ve webové aplikaci využita *localStorage*, což je interní úložiště prohlížeče, data zde uložená přetrvávají i po zavření prohlížeče. V případě opětovného otevření se aplikace podívá do tohoto úložiště a pokud je zde odpovídající neexpirovaný token uložen, použije ho ke komunikaci se serverem a uživatel se nemusí znovu přihlašovat. Servisa *authenticationTokenService* poskytuje napříč celou aplikací všechny informace o přihlášeném uživateli, které jsou dostupné v JSON Web Tokenu.

Odhlášení probíhá jednoduše tak, že je token smazán z lokálního úložiště.

8.2.6. Zobrazování v mapě

Ke zobrazení *markeru* na mapě je použito Google Maps API. Toto API poskytuje řadu metod pro práci s mapovými podklady od společnosti Google. Nový marker lze vytvořit tak, že vytvoříme novou instanci třídy *google.maps.Marker* a jejímu konstruktoru předáme konfigurační objekt. V aplikaci jsou dále použity metody pro změnu pozice markeru a jeho odstranění.

Níže je uveden příklad přidání nového markeru do mapy.

```
var marker = new google.maps.Marker({
  position: latlng,
  map: map,
  title: 'Marker title'
});
```

Objekt marker poskytuje metody, prostřednictvím kterých můžeme měnit parametry markeru, nebo ho z mapy odstranit. Pozici markeru lze změnit prostřednictvím metody *setPosition*, odstranit lze metodou *setMap* s parametrem *null*.

Příklady změny pozice a odstranění markeru:

```
marker.setPosition(position);
marker.setMap(null);
```

Veškerá manipulace s mapou probíhá v kontroleru *mapCtrl*. Markeru jde také přiřadit vlastní ikonu a je možné ho stylovat pomocí CSS. Kromě manipulace s markery lze manipulovat i s mapou. Mapě může být nastavena úroveň přiblížení metodou *setZoom*. Metoda *fitBounds* slouží k vystředění mapy podle pozic markerů na mapě.

8.3. Implementace mobilní aplikace

Pro implementaci mobilní aplikace byla zvolena technologie *PhoneGap*. Tato technologie je popsána v kapitole 4. Díky PhoneGapu lze nativní aplikaci pro mobilní zařízení implementovat pomocí standardních technologií používaných při vývoji webových aplikací jako jsou HTML, CSS a JavaScript a také ve frameworku Angular.JS. Navíc lze kód aplikace zkompilovat pro většinu mobilních platforem a není jej nutné vyvíjet pro každou platformu zvlášť.

Vzhled celé aplikace řeší framework Ionic. Ionic obsahuje například komponenty pro menu i s pamatováním historie, pro formuláře, obsahuje vlastní tlačítka, přepínače a mnoho dalšího. Tyto komponenty jsou připraveny k použití pro mobilní zařízení, jsou tedy nastylovány tak, aby byly správně zobrazeny jak na výšku, tak na šířku a na různě velkých displejích. Komponenty Ionicu jsou implementovány jako direktivy frameworku Angular.JS.

Přístup k nejrůznějším prostředkům operačního systému jako je GPS přijímač, akcelerometr, teploměr a další v prostředí PhoneGap je zajištěn prostřednictvím dodatečných pluginů, které jsou open source a jsou dostupné v online repozitářích. Plugin *geolocation* slouží ke zpřístupnění API GPS přijímače. K použití akcelerometru zařízení slouží plugin

device-motion. Plugin *insomnia* slouží k prevenci vypnutí displeje zařízení a přechodu aplikace do režimu běhu na pozadí. Posledním použitým pluginem je *whitelist*, který umožňuje komunikaci aplikace se vzdáleným serverem. Jednotlivé objekty pluginů následně rozšiřují objekt „navigator“ a jejich metody lze v JavaScriptu jednoduše volat.

8.3.1. Využití komponent z webové aplikace

Mobilní aplikace je implementována ve stejném frameworku jako webová aplikace a velká část funkcionality je shodná, bylo tedy možné mnoho funkcionality webové aplikace převzít a použít znovu v mobilní aplikaci.

Shodná je logika registrace, přihlášení a odhlášení. Jediným rozdílem je, že namísto dialogů pro zobrazení formuláře jsou použity specializované stránky. Implementace autentizace je popsána v kapitole 8.2.5.

Další částí téměř zcela převzatou z webové aplikace je správa spojení. Jak je vidět na obrázku 7.8, detail uživatele s tlačítkem akce je v mobilní aplikaci zobrazen na samostatné stránce na rozdíl od webové aplikace, kde je detail vedle seznamu uživatelů.

Mobilní aplikace stejně jako ta webová obsahuje *angular service* pro komunikaci se vzdáleným serverem. Opět ke komunikaci používá interní servisu angularu *$http*. V mobilní aplikaci je třeba navíc použít PhoneGap plugin *whitelist* a ten správně nakonfigurovat z důvodu bezpečnostních restrikcí PhoneGapu a Androidu pro komunikaci se vzdáleným serverem.

Stejným způsobem jako ve webové aplikaci je v mobilní aplikaci používána mapa a pro zobrazování markerů na mapě i pro jejich vzhled je použit shodný kód. Popis markeru byl rozšířen o tlačítko umožňující uživateli přihlášení se k odběru korekcí DGPS od konkrétní stanice.

8.3.2. Získávání a odesílání údajů o poloze zařízení

K získávání GPS souřadnic byla implementována servisa *locateService*. Prostřednictvím této servisy lze zapnout a vypnout proces získávání souřadnic a jejich následné odesílání na API serveru. K získávání GPS souřadnic slouží PhoneGap plugin *geolocation*. Jeho použití je znázorněno na následujícím příkladě:

```
navigator.geolocation.watchPosition(function (position) {
    //odeslání souřadnic prostřednictvím servisy webAPI na server
}, errorCallback, watchPositionOptions);
```

8.3.3. Režim stacionární stanice DGPS

Logika funkcionality stacionární stanice DGPS je implementována v servise *dgpsService*. V případě, že uživatel zapne režim stacionární stanice DGPS, je upozorněn, že v tomto režimu se zařízením nemá pohybovat. Po potvrzení tohoto upozornění je spuštěn proces detekce pohybu zařízení. Pohyb je detekován v aplikaci použitím PhoneGap pluginu *device-motion*, který dokáže detekovat zrychlení zařízení do všech směrů.

Detekce pohybu zařízení:

```
navigator.accelerometer.watchAcceleration(onSuccess, onError, options);
```

V případě detekce pohybu je zavolána funkce *onSuccess* a je jí předán objekt obsahující parametry x, y a z, které reprezentují aktuální zrychlení ve třech osách prostoru. Pokud zrychlení překročí určitou definovanou hodnotu, je režim stacionární stanice DGPS vypnut. Je třeba zohlednit také gravitační zrychlení země, které lze považovat jako konstantu, kterou je třeba zahrnout do porovnávací podmínky. Pomocí konfiguračního objektu *options* lze mimo jiné nastavit frekvence kontroly pohybu.

Pokud je režim stacionární stanice DGPS zapnut, jsou na server odesílány souřadnice stejným způsobem jako v části 8.3.2.

Stanice se může přihlásit k odběru korekcí DGPS od stacionární DGPS stanice ve spojení. V případě, že přihlášený uživatel zvolí DGPS stanici pro odběr korekcí, je v každém dotazu na server, ve kterém jsou odesílány jeho souřadnice přítomen GET parametr obsahující identifikátor stacionární DGPS stanice a na serveru jsou následně k souřadnicím přidány odpovídající korekce.

9. Hardwarové a softwarové požadavky

9.1. Server

Jako webový server je použit *Node.JS* a ten je třeba mít na serveru nainstalovaný. Toto prostředí lze instalovat jak na operační systém Windows, tak i na Unixové operační systémy. Dále je pro běh nutné nainstalovat databázový systém *mongodb*. Server je použit nejen pro veřejné aplikační rozhraní, ale i jako serverová část pro webovou aplikaci poskytující uživatelům potřebné soubory. Server pro svůj běh vyžaduje instalaci doplňkových balíčků využitých při implementaci. Seznam těchto balíčků je uveden níže.

```
bcrypt-nodejs
body-parser
cookie-parser
debug
express
express-jwt
jade
jsonwebtoken
mongodb-uri
mongoose
morgan
passport
passport-local
q
serve-favicon
```

Všechny tyto balíčky lze instalovat jednoduše pomocí NPM tímto způsobem:

```
npm install
```

9.2. Webová aplikace

Požadavkem pro běh webové aplikace je webový prohlížeč interpretující scripty jazyka JavaScript, což je naprostá většina v dnešní době se vyskytujících prohlížečů. Nicméně aplikace byla vyvíjena a testována v prohlížeči Google Chrome a správný běh aplikace a vzhled je zaručen pouze v něm.

Soubory webové aplikace poskytuje klientům server *Node.JS*. Tyto soubory jsou na serveru umístěny v adresáři *app*. Aplikace má několik závislostí na balíčcích třetích stran.

Balíčky třetích stran:

```
angular
angular-animate
```

```
angular-bootstrap
angular-charts.js
angularjs-slider
angular-md5
angular-route
Chart.js
```

Balíčky třetích stran lze stáhnout programem *bower*, což je nástroj na správu balíčků. Následně je třeba tyto balíčky umístit do podadresáře

```
app/bower-components
```

9.3. Mobilní aplikace

Mobilní aplikace byla testována a lze ji nainstalovat na zařízení s operačními systémy Android 4+, vzhledem k většinovému zastoupení verze Android vyšší než 4 na celosvětovém trhu nebyla při implementaci podpora starších verzí tohoto systému považována za důležitou. Ke správné funkcionalitě mobilní aplikace je nutné, aby zařízení mělo GPS přijímač a bylo připojeno k internetu.

Pro kompilaci zdrojových JavaScript, HTML a CSS souborů do výsledné aplikace je třeba mít nainstalováno prostředí *PhoneGap* a *Android SDK*. Dále je nutná instalace všech dodatečných balíčků a PhoneGap pluginů.

Balíčky vyžadované pro běh aplikace:

```
angular
angular-animate
angular-md5
angular-sanitize
angular-ui-router
ionic
markerwithlabel
```

PhoneGap pluginy vyžadované pro běh aplikace:

```
cordova-plugin-whitelist
cordova-plugin-geolocation
cordova-plugin-insomnia
cordova-plugin-device-motion
```

Balíčky lze stáhnout stejným způsobem jako v případě webové aplikace a je třeba je umístit do adresáře *lib* v adresáři, ve kterém se nachází všechny zdrojové soubory.

Jednotlivé pluginy lze instalovat tímto způsobem:

```
cordova plugin add cordova-plugin-whitelist
```

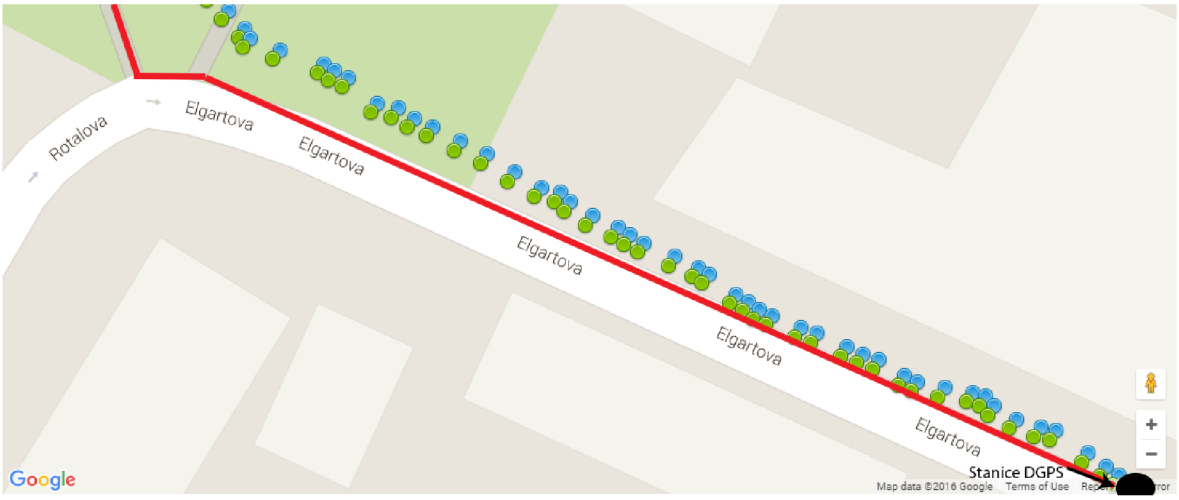
10. Vliv DGPS na přesnost lokalizace

Aplikace korekcí od stacionárních stanic DGPS na získané souřadnice by mělo mít pozitivní vliv na přesnost lokalizace stanic. Abychom ověřili tento fakt, bylo uskutečněno několik zkušebních měření.

10.1. Lokalizace na trase v krátké vzdálenosti od stacionární stanice DGPS

Měření proběhlo tak, že zařízení v režimu stacionární stanice DGPS bylo umístěno za čelní okno automobilu. Pohledem do mapy byla pozice stanice postupně upravována až se po určité době přestala měnit a odhadem odpovídala reálné pozici automobilu. Po dvou hodinách aproximace pozice stacionární stanice DGPS byla provedena krátká pěší procházka začínající na pozici stacionární stanice DGPS pokračující po chodníku podél cesty do vzdálenosti přibližně 400 metrů.

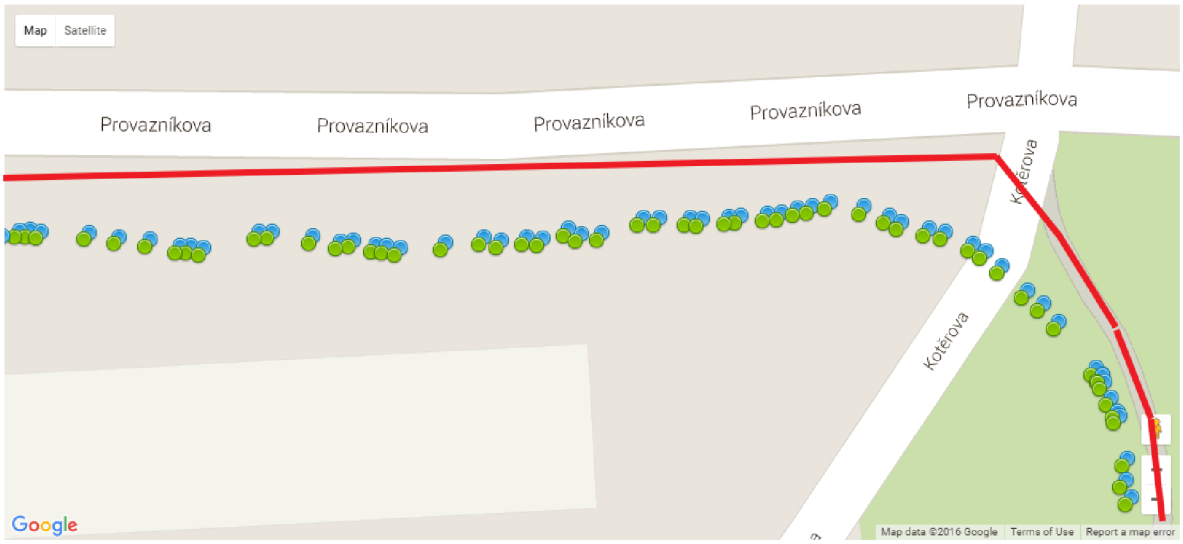
Na obrázku 10.2 je znázorněno porovnání historie poloh bez použití korekcí a s korekcemi. Modré body na obrázku znázorňují jednotlivé souřadnice bez aplikace korekcí DGPS a zelené znázorňují ty po aplikaci korekcí. Červený pruh představuje reálnou trasu procházky. Z měření je evidentní, že aplikace korekcí měla pozitivní dopad na přesnost lokalizace.



Obrázek 10.1: Porovnání naměřených souřadnic s korekcemi DGPS a bez korekcí v malé vzdálenosti od stacionární stanice DGPS.

10.2. Lokalizace na trase začínající ve větší vzdálenosti od stacionární stanice DGPS

Druhé měření je pokračováním měření prvního. Stacionární stanici DGPS stějně jako v předchozím měření představuje mobilní zařízení za oknem automobilu. Trasa znázorněná na obrázku začíná jeden kilometr od stacionární stanice a má délku přibližně 300 metrů. Z obrázku 10.2 je zřejmé, že aplikace korekcí získaných od stejné DGPS stanice má naopak negativní vliv na přesnost lokalizace. V obrázku stejně jako v předchozím případě modré body znázorňují souřadnice bez korekcí a zelené s korekcemi, červeně je vyznačena trasa.



Obrázek 10.2: Porovnání naměřených souřadnic s korekcemi DGPS a bez korekcí ve větší vzdálenosti od stacionární stanice DGPS.

Z předchozích dvou měření vyplývá, že vzdálenost od stacionární DGPS stanice má velký vliv na reálnou použitelnost, a že ve větších vzdálenostech může mít aplikace DGPS korekcí negativní dopad na přesnost určení pozice.

10.3. Porovnání rozptylu GPS souřadnic

Cílem tohoto měření bylo porovnat rozptyl naměřených souřadnic s aplikací korekcí DGPS a bez těchto korekcí. Měření proběhlo tak, že jedno mobilní zařízení bylo v režimu stacionární stanice DGPS a učilo se svoji průměrnou pozici po dobu dvou hodin. Následně bylo na stejnou pozici (zařízení byla vedle sebe) umístěno druhé zařízení, které se přihlásilo o odběr korekcí DGPS od první stanice a začalo určovat svoji pozici.

Obě zařízení byla po celou dobu na stejném místě a následně bylo měření ukončeno. Výpočty rozptylu proběhly z 400 po sobě jdoucích naměřených hodnot. Tyto hodnoty byly webovou aplikací vygenerovány z historie poloh do souboru CSV. Tato data byla následně zpracována v tabulkovém kalkulátoru Microsoft Office Excel.

Výsledky rozptylu s aplikací DGPS korekcí:

- Rozptyl zeměpisných šířek : $\mathbf{5,80529 \cdot 10^{-10}}$
- Rozptyl zeměpisných délek : $\mathbf{4,09481 \cdot 10^{-10}}$

Výsledky rozptylu bez aplikace DGPS korekcí:

- Rozptyl zeměpisných šířek: $\mathbf{6,06858 \cdot 10^{-10}}$
- Rozptyl zeměpisných délek: $\mathbf{4,21307 \cdot 10^{-10}}$

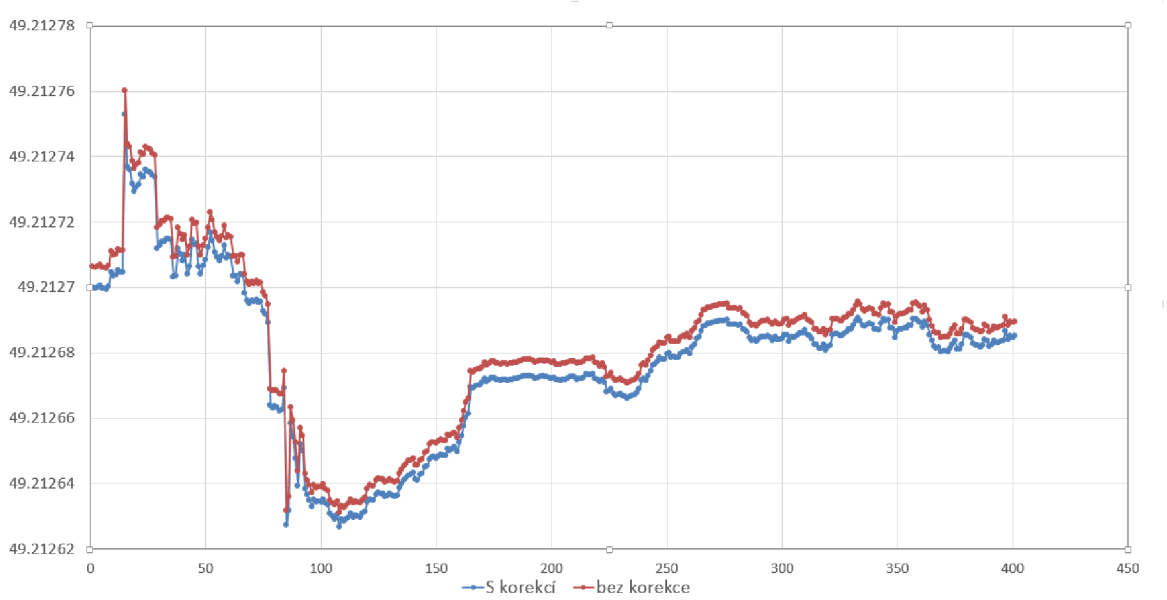
Vzhledem k tomu, že zařízení neměnilo svoji pozici, by měl být rozptyl teoreticky roven nule, což kvůli nepřesnostem a chybám v měření není možné. Z výsledných hodnot vyplývá, že za daných podmínek byl rozptyl s použitými korekcemi DGPS menší, tudíž měření bylo přesnější. Grafy na obrázcích 10.3 a 10.4 srovnávají jednotlivé po sobě následující zeměpisné šířky a délky. V grafech na obrázcích 10.5 a 10.6 jsou znázorněny jednotlivé absolutní odchylky naměřených hodnot s DGPS korekcemi od hodnot bez těchto korekcí. Tyto odchylky představují velikost korekce, která byla aplikována na souřadnice.

Dále byly vypočítány průměrné velikosti korekcí a hodnota v metrech představující vzdálenost mezi dvěma souřadnicemi před a po aplikaci korekcí DGPS.

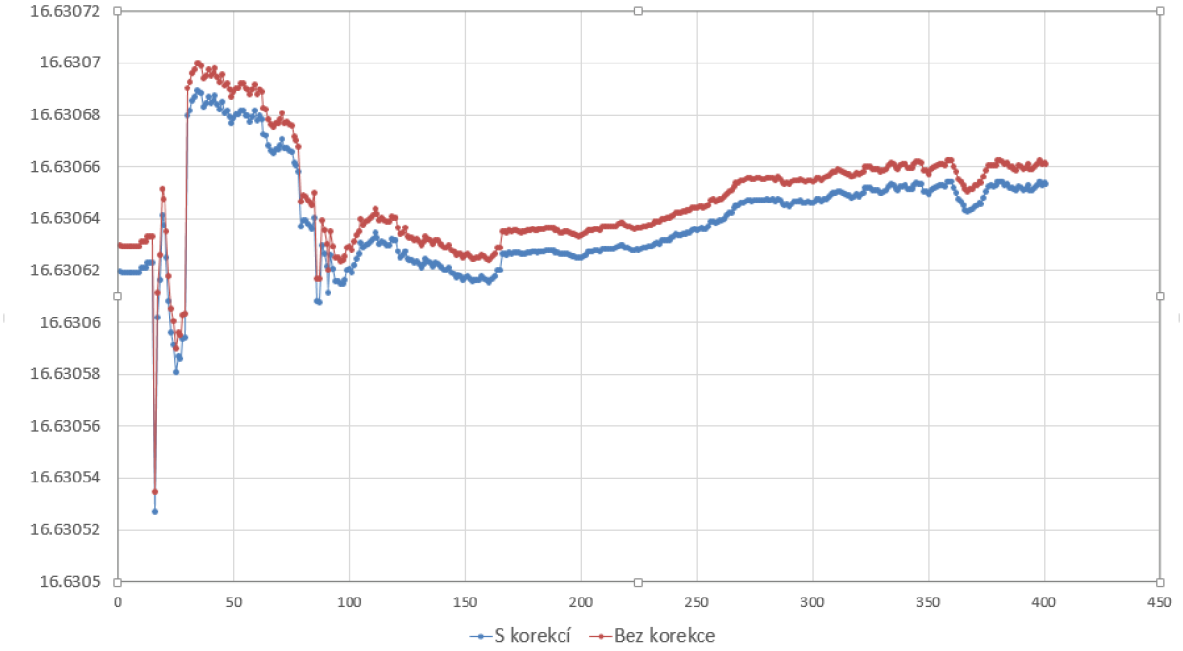
- Průměrná velikost korekce zeměpisné šířky: $\mathbf{5,00078 \cdot 10^{-6}}$
- Průměrná velikost korekce zeměpisné délky : $\mathbf{8,77391 \cdot 10^{-6}}$

Průměrná korekce zeměpisné šířky představuje vzdálenost $0,55m$, korekce zeměpisné délky představuje vzdálenost $0,64m$.

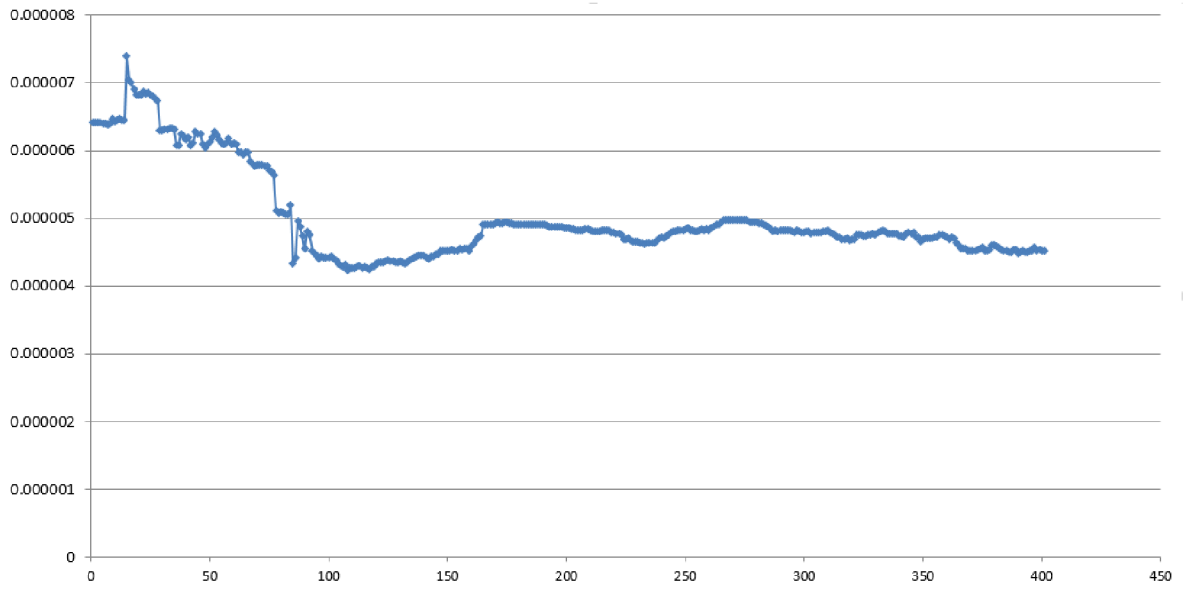
Na přiloženém CD je k dispozici soubor pro Microsoft Office Excel, kde jsou tato data zpracována. Přesné umístění tohoto souboru je uvedeno v seznamu příloh v kapitole 13.



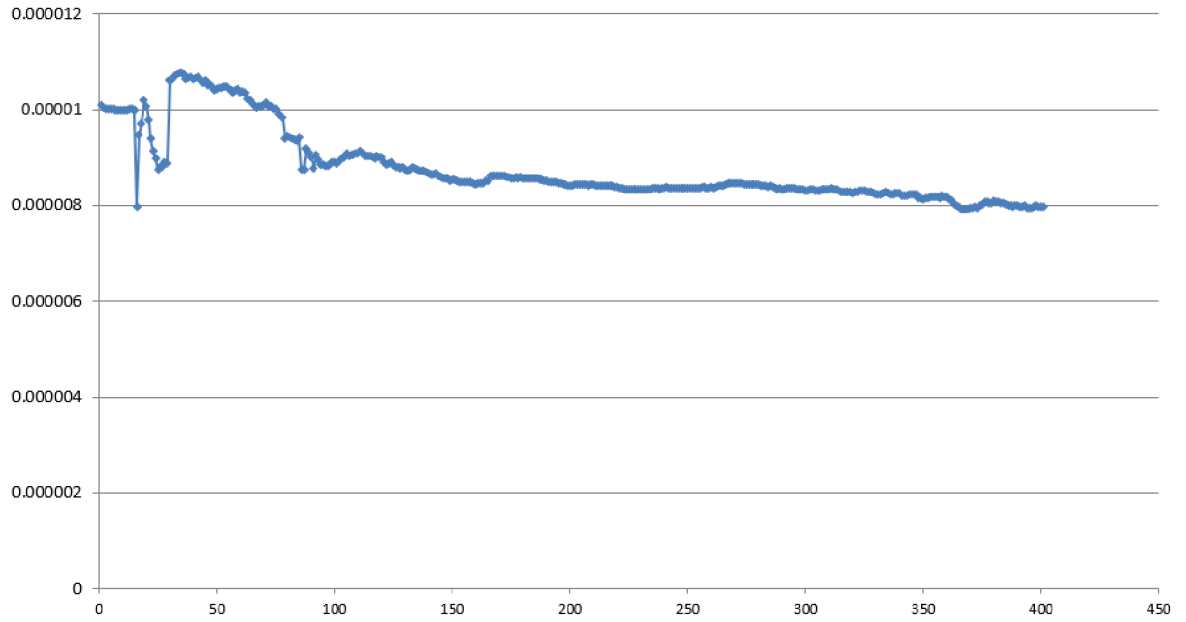
Obrázek 10.3: Porovnání naměřených zeměpisných šířek s korekcemi DGPS a bez korekcí.



Obrázek 10.4: Porovnání naměřených zeměpisných délek s korekcemi DGPS a bez korekcí.



Obrázek 10.5: Absolutní odchylky zeměpisných šířek s korekcemi DGPS od hodnot bez korekcí.



Obrázek 10.6: Absolutní odchylky zeměpisných délek s korekcemi DGPS od hodnot bez korekcí.

11. Závěr

11.1. Zhodnocení dosažených výsledků

Cílem této semestrální práce bylo navrhnout a implementovat informační systém skládající se z mobilní aplikace odesílající své údaje o poloze na vzdálený server, z webového serveru, který tyto údaje zpracuje a z webové aplikace prezentující tyto informace uživateli. Dále bylo cílem implementovat technologii diferenciální GPS pro zpřesnění lokalizace. K získání znalostí potřebných k úspěšné realizaci požadovaného informačního systému bylo třeba nastudovat problematiku kolem GPS, tato technologie je popsána v kapitole 2. Dále bylo nutné seznámit se s platformou Android, pro kterou byla vyvinuta mobilní aplikace. Tento operační systém je popsán v kapitole 3. Nezbytné bylo nastudování technologií později použitých při implementaci. Technologie PhoneGap, Ionic, Node.JS, Express, MongoDB, Mongoose a AngularJS jsou popsány v kapitolách 4 a 5.

Po nastudování teoretických znalostí byla provedena analýza již existujících řešení v kapitole 6, byla provedena analýza požadavků a informační systém byl navržen, návrh byl popsán v kapitole 7 a následně byl informační systém implementován, jak bylo popsáno v kapitole 8. Bylo provedeno měření vlivu diferenciální GPS na přesnost lokalizace, to bylo vyhodnoceno v kapitole 10.

Výsledkem implementace je funkční informační systém, který splňuje všechny body zadání v plném rozsahu. V době recenzního řízení bude webový portál veřejně dostupný na adrese **http://128.199.55.115:3000/**. Na úvodní stránce se v pravém horním rohu nachází odkaz na stažení mobilní aplikace pro OS Android. Pro přihlášení lze použít uživatelské jméno „user1“ s heslem „heslo“ a následně lze při zvolení data 13.5.2016 popřípadě 14.5.2016 na stránce s historií nebo s grafem přesností zobrazit naměřená vzorová data. Korektní chování a vzhled aplikace je otestováno v prohlížeči Google Chrome.

11.2. Návrhy na budoucí rozšíření

11.2.1. Běh mobilní aplikace v režimu na pozadí

Aktuálně mobilní aplikace nikdy nepřechází do režimu běhu na pozadí a display zařízení zůstává zapnut. V podobném režimu fungují například aplikace pro navigaci. Pokud by aplikace přešla do režimu běhu na pozadí, odesílání souřadnic na server by bylo automaticky ukončeno operačním systémem.

V případě, že by byla žádaná funkcionalita odesílání souřadnic i s vypnutým displejem, například kdyby chtěl uživatel mobilní aplikaci používat na zařízení v kapse, bylo by třeba rozšířit implementaci tak, aby aplikace byla schopna odesílat souřadnice i v režimu běhu na pozadí.

11.2.2. Možnost ručního zadání souřadnic stanice DGPS

Pokud by uživatel věděl přesnou pozici stacionární stanice DGPS, mohl by zadat její souřadnice ručně. V takovém případě by odpadla nutnost pozici určovat z dlouhodobého průměru. Problém je však v tom, aby takové řešení bylo spolehlivé a uživatelé nezadávali tyto souřadnice špatně. Řešit by se to dalo novým ověřeným typem uživatele.

Literatura

[1] Adobe Systems Inc.: PhoneGap [online]. 2014 [cit. 31. 11. 2014]. http://phonegap.com/about/

[2] AngularJS [online]. 2014 [cit. 31. 11. 2014]. https://docs.angularjs.org/guide

[3] Doglio, F.: Pro REST API Development with Node.js. Apress, 2015. ISBN 9781484209172.

[4] El-Rabbany, A.: Introduction to GPS: The Global Positioning System. Artech House, 2002. ISBN 9781580531832.

[5] European GNSS Service Centre [online]. 2014 [cit. 29. 4. 2016]. http://www.gsc-europa.eu/

[6] Foursquare [online]. 2016 [cit. 1. 5. 2016]. https://foursquare.com/about

[7] Google Android [online]. 2016 [cit. 29. 4. 2016]. http://www.android.com/history/

[8] Green, B. a Seshadri, S.: AngularJS. O'Reilly Media, 2013. ISBN 9781449355876.

[9] Hofmann-Wellenhof, B., Lichtenegger, H. a Wasle, E.: GNSS – Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and more. Springer, 2007. ISBN 9783211730171.

[10] Chodorow, K. a Dirolf, M.: MongoDB: The Definitive Guide. O'Reilly Media, 2010. ISBN 9781449396985.

[11] Introduction to JSON Web Tokens [online]. 2016 [cit. 2. 5. 2016]. https://jwt.io/introduction/

[12] Ionic Documentation Overview [online]. 2016 [cit. 29. 4. 2016]. http://ionicframework.com/docs/overview/

[13] Jackson, W.: Android Apps for Absolute Beginners. Apress, 2012. ISBN 9781430247883.

[14] Krol, J.: Web Development with MongoDB and Node.js. Packt Publishing, 2014. ISBN 9781783987313.

[15] MADRON, T.: Diferenciální GPS. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009, 75 s. Vedoucí diplomové práce doc. Ing. Luděk Žalud, Ph.D.

[16] Rauch, G.: Smashing Node.js: JavaScript Everywhere. Wiley, 2012. ISBN 9781119963103.

[17] Russian system of differentional correction and monitoring [online]. 2014 [cit. 29. 11. 2014]. http://www.sdcm.ru/

[18] Thurston, J.; Moore, J.P.; Poiker, T.K.: Integrated Geospatial Technologies: A Guide to GPS, GIS, and Data Logging. Wiley, 2003, p.134-137. ISBN 9780471244097.

[19] Trice A.: PhoneGap Explained Visually [online]. 2012 [cit. 31. 11. 2014]. http://phonegap.com/2012/05/02/phonegap-explained-visually/

[20] U.S. government: Official U.S. Government information about the Global Positioning System (GPS) and related topics [online]. 2014 [cit. 26. 11. 2014]. http://www.gps.gov/systems/gps

[21] Wikipedia: Global Positioning System [online]. 2014 [cit. 21. 11. 2014]. http://cs.wikipedia.org/wiki/Global_Positioning_System

[22] Wikipedia: Differential GPS [online]. 2014 [cit. 26. 11. 2014]. http://en.wikipedia.org/wiki/Differential_GPS

[23] Wikipedia: Google Latitude [online]. 2014 [cit. 4. 12. 2014]. http://en.wikipedia.org/wiki/Google_Latitude

[24] Wikipedia: Foursquare [online]. 2014 [cit. 4. 12. 2014]. http://en.wikipedia.org/wiki/Foursquare

12. Seznam použitých zkratek a symbolů

API	Application Programming Interface
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
CSV	Comma-separated values
DGPS	Differential Global Positioning System
DOM	Document Object model
GPS	Global Positioning System
FAA	Federal Aviation Administration
FOC	Full Operational Capability
JSON	JavaScript Object Notation
JWT	JSON Web Token
MCS	Master control station
MVC	Model View Controller
NPM	Node Package Manager
OHA	Open Handset Alliance
OS	Operační systém
PPS	Precise Positioning System
REST	Representational state transfer
SDK	Software development kit
SPS	Standard Positioning System
URL	Uniform Resource Locator

13. Seznam příloh

13.1. Obsah CD

K diplomové práci je přiloženo CD obsahující následující materiály:

- zdrojové kódy diplomové práce v LaTeX

  `/text/latex`

- text diplomové práce v PDF

  `/text/pdf`

- zdrojové kódy webové aplikace a serveru

  `/sources/locate-webportal`

- zdrojové kódy mobilní aplikace

  `/sources/locate-mobile`

- přeložená mobilní aplikace připravená k instalaci na OS Android

  `/sources/bin`

- soubor ve formátu pro Microsoft Excel s výpočty rozptylů a absolutních odchylek

  `/stats`