TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Mobilní aplikace, poskytující informace o rozhlednách a okolním panoramatu

Diplomová práce

M14000183

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **David Václavek**
Vedoucí práce: Ing. Igor Kopetschke

Liberec 2017

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Mobile application providing information about lookout towers and its surrounding panorama

Master thesis

M14000183

Study programme: N2612 – Electronics and Informatics
Study branch: 1802T007 – Informational Technologies

Author: **David Václavek**
Supervisor: Ing. Igor Kopetschke

Liberec 2017

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2016/2017**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. David Václavek**
Osobní číslo:	**M14000183**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Mobilní aplikace poskytující informace o rozhlednách a okolním panoramatu**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Proveďte rešerši existujících alternativ včetně jejich nedostatků a použijte ji jako motivaci pro vývoj vlastního řešení
2. Uveďte krátký souhrn existujícího návrhu dle předešlého projektu
3. Navrhněte algoritmus pro opravu odchylky mezi magnetickým a severním pólem Země v závislosti na čase, poloze a nadmořské výšce
4. Navržený algoritmus implementujte do vzorové aplikace, která jej využije pro korektní zobrazení viditelného 360 panoramatu
5. V závěru rozeberte výsledky testovacího provozu a případné využití v praxi

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **40 - 60 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] MEDNIEKS, Zigurd R. Programming Android. 2nd ed. Beijing: O'Reilly, c2012, xvii, 542 s. ISBN 9781449316648.
[2] Android dokumentace [online]. [cit. 2014-10-10]. Dostupné z: http://developer.android.com/index.html
[3] GERBER a CRAIG. Learn android studio: build android apps quickly and effectively. Berkeley: Apress, 2014. ISBN 978-143-0266-013.
[4] DARWIN, Ian F. Android cookbook. 1st ed. Beijing: O'Reilly, c2012, xix, 661 s. ISBN 978-1-4493-8841-6.

Vedoucí diplomové práce: **Ing. Igor Kopetschke**
Ústav nových technologií a aplikované informatiky

Datum zadání diplomové práce: **20. října 2016**

Termín odevzdání diplomové práce: **15. května 2017**

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

L.S.

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 20. října 2016

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 3.1.2017

Podpis: 

Abstrakt

Tato diplomová práce se zabývá vytvářením mobilní aplikace, která uživateli poskytne informace o rozhlednách a především o okolním výhledu, na základě dat z databáze. Databáze je reprezentována pomocí souborů formátu JSON na serverové straně aplikace. Na klientské straně aplikace je navrhnut a implementován model, pomocí kterého lze eliminovat nepřesnosti v zobrazení panoramatu, způsobené rozdílnými polohami magnetického a geografického severního pólu Země. Dále je implementováno grafické rozhraní pro přívětivou komunikaci s uživatelem aplikace. Následným testováním aplikace a modelu v ní zahrnutého je vyhodnocena přesnost modelu i nutná míra přítomnosti kvalitních čidel v testovaném mobilním zařízení.

Klíčová slova: mobilní aplikace, World Magnetic Model, magnetické pole, 360° panorama

Abstract

This Master's Thesis deals with a mobile application creation. The application provides information to a user, about a lookout towers and surrounding panorama, especially. A information source is a database of JSON files. The source files are saved at a server side of the application. At a client side of the application, there is a model proposal and implementation, which allows to eliminate inaccurancies at a panorama rendering, due to different locations of the magnetic north and the geographic North Pole. There's also implemented a GUI for a friendly communication with the application user. Finally, by testing of the application and the model included in, an accuracy of the model is evaluated. There's evaluated necessary of presence of a sensor's quality in tested mobile device, as well.

Keywords: mobile application, World Magnetic Model, magnetic field, 360° panorama

Poděkování

V první řadě bych velmi rád poděkoval oboum mým vedoucím, které bylo z nutné z důvodu organizačních změn na škole v průběhu práce změnit. Děkuji tedy panu Ing. Radku Srbovi za ujmutí se ve vedení diplomové práce, přívětivou podporu a komunikaci. Velmi velký dík patří panu Ing. Igoru Kopetschkemu, a to za převzetí vedení práce před jejím dokončením, cenné rady při konzultacích a při vypracování diplomové práce. Dík dále patří mé rodině, která za mnou stála ve chvílích psychicky náročných, měla pevné nervy, která mě dokázala podržet a nabudit do další práce. Nakonec bych chtěl poděkovat všem mým přátelům a blízkým, kteří jakoukoliv měrou přispěli k úspěšnému dokončení této zavěrečné práce.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

API sbírka naprogramovaných prostředků, které může programátor využívat (Application Programming Interface)

APK formát souboru pro distribuci a instalaci mobilní aplikace (Android application package)

CSV jednoduchý souborový formát, určený pro výměnu tabulkových dat (Comma-separated values)

DP Diplomová práce

EMM rozšířený magnetický model (Enhanced Magnetic Model)

ER vztah mezi databázovými tabulkami (Entity Relationship)

FTPS rozšíření, zajišťující zabezpečený přenos citlivých informací přes počítačovou síť (File Transfer Protocol Secure)

GB označení jednotky množství informace, používané v informatice (Gigabyte)

GD volně dostupná knihova pro práci s obrázky (Gif Draw)

GIF grafický formát pro rastrovou grafiku, podpoorující jednoduché animace (Graphics Interchange Format)

GPS prostředek, poskytující poziční a navigační služby (General Positioning System)

HTTP internetový protokol, určený pro výměnu hypertextových dokumentů ve formátu HTML (Hypertext Transfer Protocol)

IDE software pro vývoj aplikací (Integrated Development Environment)

iOS mobilní operační systém, vytvořený společností Apple Inc.

JSON jazyk pro popis dat (JavaScript Object Notation)

JSP technologie pro vývoj HTML stránek v jazyce Java (JavaServer Pages)

LaTeX typografický systém, soubor maker pro sázecí program TeX

NFC technologie komunikace mezi elektronickými zařízeními (Near Field Communication)

NoSQL databázový koncept, využívající jiné prostředky, než tabulková schémata tradiční relační databáze (Not Only SQL)

ODF otevřený souborový formát pro kancelářské aplikace (Open Document Format)

PHP skriptovací jazyk, určený pro programování dynamických internetových stránek a webových aplikací (Hypertext Preprocessor)

POJO klasické Java objekty, používané pro mapování dat (Plain Old Java Object)

PRO Semestrální projekt

QR QR kód, prostředek pro automatizovaný sběr dat (Quick Response)

RAM typ paměti s přímým přístupem (Random Access Memory)

REST způsob, jak pracovat s informace ze serveru pomocí HTTP volání (Representational state transfer)

SFTP protokol pro bezpečný přenos souborů pomocí počítačové sítě (SSH File Transfer Protocol)

SQL dotazovací jazyk, který je používán pro práci s daty v relačních databázích (Structured Query Language)

SV sekulární variace, jedna vlastností z magnetického pole Země (Secular Variation)

UI rozhraní pro interakci s uživatelem (User Interface)

UIL knihovna pro práci s obrázky pro systém Android (Universal Image Loader)

uPnP sada síťových protokolů pro jednoduché připojení periferních součástí počítače (Universal Plug and Play)

VCS systém pro správu verzí programu (Version control system)

Wi-Fi označení pro několik standardů IEEE 802.11, popisujících bezdrátovou komunikaci v počítačových sítích

WMM Magnetický model Země (World Magnetic Model)

XML obecný značkovací jazyk, používaný mimo jiné pro serializaci dat (Extensible Markup Language)

XSL rodina jazyků umožňujících popsat, jak se mají XML soubory formátovat a převádět (eXtensible Stylesheet Language)

YAML formát pro serializaci strukturovaných dat (YAML Ain't Markup Language)

Úvod

Primárním úkolem této práce je navrhnout algoritmus, který opravuje odchylku mezi magnetickým a geografickým severním pólem Země. Tento algoritmus je použit jako hlavní složka pro vykreslení 360° viditelného panoramatu. Algoritmus bude následně implementován do vzorové aplikace. Ta má mimo korektního zobrazení panoramatu další důležitou funkci, a to zpřístupnit uživateli informace o rozhlednách.

Algoritmus, který bude implementací matematického modelu WMM, bude závislý na čase, poloze a nadmořské výšce uživatele. Pro korektní zobrazení panoramatu bude také třeba implementovat vyhovující kompas za pomoci čidel, dostupných v mobilním zařízení. Aplikace jako celek bude následně otestována v emulátoru i zkušebním provozu. Výsledky budou porovnány a úspěšnost navrženého řešení vyhodnocena.

Celkový rozsah aplikace bude rozdělen do tří částí, přičemž každá bude obstarávat svůj separovaný okruh funkcí. První částí aplikace bude část serverová, která bude sloužit jako databázová vrstva aplikace. Bude také obsluhovat aplikační rozhraní mezi klientem a vracet mu odpovídající výsledky. Další vrstvou je část klienta, která bude sloužit pro získání dat ze serveru, práci s nimi, výpočty na pozadí aplikace a jejich korektní aktualizaci. Poslední kus vývoje aplikace bude soustředěn na uživatelské rozhraní aplikace a přívětivé zobrazení dat.

Práce se také zabývá teoretickou rovinou, kde bude popsáno magnetické pole Země a děje, v něm se vyskytující. Dále budou popsány technologie a knihovny funkcí, které mohou napomoci k úspěšné realizaci modelu a následně tak i celé aplikace. Z nich budou vybrány ty, ze kterých bude následně algoritmus složen a realizován.

Motivace

Člověk se stále více upíná k používání mobilních technologií, jejichž rozmach se zdá být k nezastavení. V současné době je pro civilizovanou osobu skoro nemyslitelné, aby žila bez denního kontaktu s chytrým mobilním zařízením. I proto stále více zapomínáme na pohyb a naše zdraví. Jedním z motivů aplikace je tedy přimět lidi k pohybu.

Pokud se zamyslíme nad současnou situací rozhleden v naší zemi, majoritně převládá fakt, že nádherný výhled do krajin České republiky není doprovázen jejím popisem. Desky s panoramaty jsou buď zničené nebo poškrábané vandaly, ve velkém množství případů však chybí na rozhlednách úplně. Tato skutečnost byla pro mě, jako nadšeného turistu, prvním popudem k vytvoření aplikace tohoto rázu. Myšlenka, že by mohli lidé, za pomocí mobilní aplikace mít vždy, kromě informací o navštěvované rozhledně, po ruce také její panorama, je velmi lákavá.

Na aplikaci začal jsem pracovat v rámci předmětu PRO. Byl navržen prvotní model, který obsahoval spoustu nedořešených problémů. Jmenovitě například uživatelsky nepřívětivé grafické prostředí, nemožnost filtrace a vyhledávání v datech a složitou strukturu kódu. Hlavními nedostatky však byly nepřesnosti při zobrazení panoramatu a nutnost navštívit rozhlednu, před vlastním zobrazením panoramatu.

I přes tyto skutečnosti byl projekt obhájen na výbornou, pochválen a v rámci vyhlašování nejlepších prací také finančně oceněn. Zároveň mi bylo po obhajobě členy poroty navrhnuto, zda bych nechtěl na projektu pokračovat v diplomové práci.

To mě opravdu motivovalo, a tak jsem se rozhodl v rámci DP projekt přepracovat a odstranit nedostatky předchozího návrhu.

Jako posledním významný bod mé osobní motivace bych uvedl skutečnost, že žádný z oficiálních webových portálů, poskytujících aplikace pro mobilní platformy, prozatím aplikaci s podobnou funkcionalitou, která by se lišila od aplikací vyvíjených pro turistické účely, nenabízí. Tohoto faktu si začínají poslední dobou všímat i informační střediska v okrese Děčín.

Přehled existujících alternativ

Existuje velké množství serverů, které uživatelům mobilních zařízení zprostředkují přístup ke stažení mobilních aplikací. V přehledu existujících alternativ této práce je mířeno pouze na zdroje oficiální. Těmi jsou Google Play [16], který je shromaždištěm aplikací pro operační systém Google Android. Dále pak App Store [15], sloužící jako sklad aplikací pro platformu iOS. Posledním obchodem je potom Windows Phone Store [4], a to pro systém Windows Phone. Na tyto servery programátoři umisťují své aplikace, ať už volně dostupné nebo za poplatek.

Další oficiální portály, jako například Blackberry World [17], žádné aplikace z oblasti alternativ nenabízí, a proto analyzován nebude.

Pokud by měla být vystižena hlavní funkce aplikace (zobrazení viditelného okolního panoramatu rozhledny), tak ani na jednom z uvedených serverů není podobná aplikace dostupná. V následujícím přehledu je uveden popis aplikací, které jsou ostatními programovými vlastnostmi funkčně podobné a dají se tedy považovat za alternativu aplikace, vyvíjené v této práci.

Mapy.cz

Mapy.cz je aplikací od české firmy Seznam. cz, a.s. Aplikace Mapy.cz nabízí plánování trasy autem, na kole a pěšky. V aplikaci je implementována také turistická mapa, díky které se uživatel dozví o stezce a také hradu, vyhlídce či rozhledně. Dle [22] je zřejmé, že turistické prvky jsou v mapových podkladech zaneseny a po jejich zvolení se zobrazí podrobnosti o daném bodě. Po stáhnutí mapových podkladů do uložiště mobilního zařízení je možné je procházet i bez připojení k internetu. Kromě těchto prvků je zde také funkce takzvaného batůžku. Ta spočívá v možnosti uložit si oblíbená místa do zvláštního seznamu. Vše samozřejmě funguje za základě lokalizace uživatele aplikace pomocí polohových služeb. Mapy.cz jsou dostupné pro všechny tři analyzované obchody s mobilními aplikacemi. Aplikace je pravidelně aktualizována o nové funkce a podklady a je dostupná bez poplatku.

Pokud by tedy měly být srovnány společné prvky této aplikace a aplikace vyvíjené, jsou jimi zobrazení vyhlídek a rozhleden. Po zvolení prvku jsou zobrazeny detailní informace. V obou aplikacích jsou tato data dostupná bez připojení k internetu (za předpokladu, že byla data při prvním spuštění stažena do zařízení). Společnou vlastností je také fungování aplikace na základě zjištění polohy uživatele.

Aplikace žádné nevýhody nemá, splňuje vše, co by aplikace tohoto typu umět měla. A vzhledem k tomu, že není primárně zaměřena pouze na vyhlídky a rozhledny, není nepřítomnost zobrazení viditelného panoramatu místa na obtíž.

Seenery

Seenery je mobilní aplikací, která se obsahem podobá vyvíjené aplikaci nejvíce. Její hlavní vlastností je totiž zobrazení přehledu rozhleden v České republice. V přehledovém okně aplikace lze rozhledny vyhledávat a třídit dle vzdálenosti od místa polohy uživatele. Po klepnutí na rozhlednu v seznamu lze zobrazit její detail, který

zobrazují obdobné informace, jako Mapy.cz. Aplikace umožňuje uložit oblíbené rozhledny do odděleného seznamu, uživatel k nim tak získá rychlý přístup [5]. Aplikace je k dispozici zdarma, ale pouze pro mobilní platformu od společnosti Apple Inc. (iOS) a je aktualizována v řádu měsíců.

Společné prvky jsou zřejmé – zobrazení evidovaných rozhleden, vyhledávání v nich, zobrazení jejích detailu a získání polohy uživatele. Pokud se zmiňujeme o aplikaci, jejíž zaměření se týká pouze rozhleden, tak její hlavní nevýhodou je nemožnost zobrazení okolního výhledu místa.

Tipy na výlet – Vyletnik.cz

Tato aplikace umožňuje zobrazit tipy na výlety v rámci České republiky. Přímo v telefonu tak lze zobrazit turistická místa v okolí (včetně rozhleden), kde se uživatel nachází, vyhledávat podle názvu a v mapě. Navigace mezi místy je obdobná, jako u první zmiňované aplikace. Výhodou je zobrazení komentářů k místu. Uživatel tak snadno může zhodnotit atraktivitu výletu na základě předešlých uživatelských zkušeností. Každý registrovaný uživatel může do aplikace vložit místo dle vlastního uvážení [20]. Po schválení administrátorem se začne ono místo zobrazovat i dalším uživatelům. Tipy na výlet – Vyletnik.cz je k dispozici zdarma, a to pro systémy Google Android a iOS.

Nevýhodou aplikace je, že nebyla již dlouho aktualizována. Poslední aktualizace proběhla dne 3. dubna 2014 pro platformu od Google Inc., respektive 13. září 2012 pro iOS. Tato data jsou platná ke dni 22. listopadu 2016. Tím pádem nejsou zakomponovány nové vývojové prvky a potenciál na novějších zařízeních není plně využit. Další nevýhodou a rozdílem oproti vyvíjené aplikaci je stejně, jako u ostatních aplikací, nepřítomnost panoramatu při výhledu z vyhlídky či rozhledny.

TripAdvisor

Nepochybně jednou ze světoznámých mobilních aplikací z oblasti turistiky je TripAdvisor. Aplikace umožňuje cestovateli prohlížet fotografie, mapy, recenze, hotely, restaurace a další informace o destinacích po celém světě. TripAdvisor nabízí uživateli aktivity a zajímavosti z místa, kde se právě nachází. Jednotlivé položky lze samozřejmě třídit a filtrovat. Aplikaci lze bez poplatků stáhnout z obchodu pro platformy Google Android a iOS.

Vysočina GeoHra

Tato aplikace je hrou, která má kromě poznání krajů Vysočiny také přimět uživatele k pohybu. Vysočina GeoHra obsahuje dva herní okruhy: „Židovské cesty na Vysočině“ a „Rozhledny a vyhlídkové věže na Vysočině“ [23]. V daných místech každého okruhu hra uživateli zadává kvízové otázky. Jejich plněním lze získat body. Výsledky hry je možné sdílet přes sociální sítě a na dálku tak soutěžit se svými přáteli.

Aplikaci lze ohodnotit jako velmi zajímavou a je důkazem, turistický rozvoj Kraje Vysočina vzkvétá. Aplikace je dostupná zdarma pro operační systémy Google Android a iOS, je poměrně nová, a tak je v současné době pravidelně aktualizována.

Vysočina Tourism

Vysočina Tourism umožňuje poznat památky a mnohé další turistické zajímavosti Kraje Vysočina. Z aplikace lze také prohlížet aktuality a akce v kraji. Aplikace umožnuje filtraci a vyhledávání v datech. Výhodou aplikace je spolupráce s čtečkou QR kódů a NFC tagů, a tak je možné rychle dohledat informace o místě přímo na cestě. U nejvýznamnějších a nejzajímavějších turistických cílů, kterých je přes dvě stě, je umístěna tabulka s kódem nebo čipem. Ten uživatele přesměruje přímo na detail objektu. Aplikace, která je stejně, jako ta přechozí poměrně nová, je pravidelně aktualizována a je ke stažení zdarma v internetových obchodech Google Play, App Store, Windows Phone Store.

Aplikace získala čestné uznání na festivalu Tour Region Film v kategorii Mobilní aplikace a webové stránky [2]. Ocenění pouze dokazuje, že kraj jde s dobou a pro podporu turismu úspěšně využívá moderní nástroje.

1 Teoretický rozbor

Tato část práce se zabývá teoretickou částí řešené problematiky. Obsahuje shrnutí modelu, vyvinutého v rámci předmětu PRO. Dále jsou rozebrány technologie a čidla, obsažená v mobilních zařízeních, která mohou napomoci k nalezení ideálního systému. Tento systém lze definovat jako systém, který má minimální odchylky od reality. Jsou zde také rozebrány důvody výběru jednotlivých technologií a senzorů. Poté je prodiskutován výběr platformy serverové části aplikace, kde je určena například vhodná podoba databázové vrstvy nebo typ použitého serveru (z hlediska programovacího jazyka). Na závěr rozboru je zmíněn způsob správy programovaného kódu a výběr použitých vývojových prostředí.

1.1 Shrnutí prvotního modelu

V této sekci bude popsán původní návrh modelu, včetně jeho nedostatků. Bude popsán také způsob, jakým byl vyjádřen bod panoramatu. Závěr sekce bude poté patřit poznatkům a zkušenostem, získaných z návrhu, které byly impulzem pro jeho změnu a zpřesnění, což bude následně popsáno v praktické části práce.

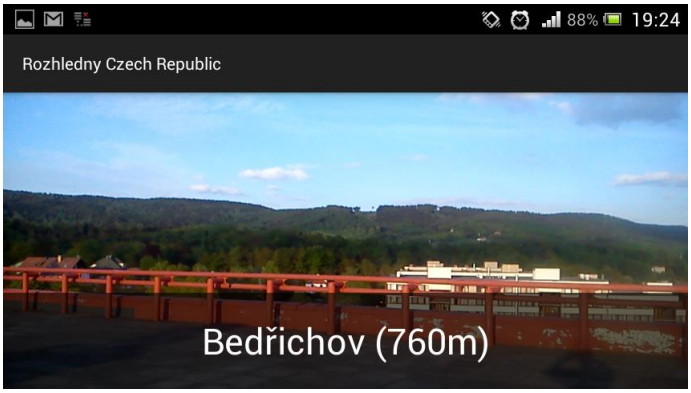
1.1.1 Odlišení od existujících alternativ, přednosti a popis

Model se od existujících alternativ odlišuje funkčně především tím, že kromě zobrazení zpracovaných dat z databáze, zobrazuje i panorama místa, kde uživatel stojí. To je jeho hlavní předností. Zpracovanými daty z databáze jsou myšleny informace o rozhlednách, které jsou uživateli zobrazeny tak, jak je uvedeno na obrázku 1.1. Tento pohled je platný pouze v případě, že je zařízení otočeno na šířku (nalevo).

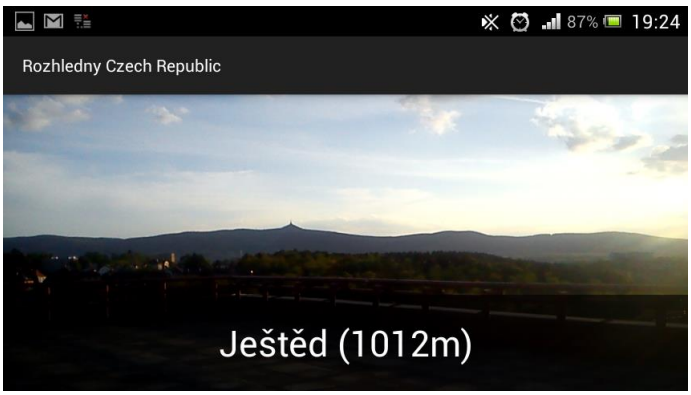
Pokud je uživatel pomocí polohových služeb lokalizován, že stojí nějaké z rozhleden, zanesených v databázi, je při poloze zařízení otočeného na výšku zobrazen detail rozhledny, na které právě stojí. V detailu je kromě názvu a nadmořské výšky také fotografie, reprezentující dané místo. V textovém popisu jsou dále informace o historii rozhledny a jak se na ní dostat.

V případě úspěšné lokalizace uživatele a otočení zařízení na šířku (napravo) je zobrazena aktivita s okolním panoramatem, viditelným z rozhledny. Příklad takového zobrazení je na obrázkách 1.2 a 1.3. Informace o panoramatu jsou načteny z databáze a pomocí sestrojeného kompasu jsou porovnávány azimuty míst s azimutem, který ukazuje kompas. Je-li absolutní hodnota rozdílu azimutů v toleranci, je zobrazen popisek a nadmořská výška místa.

Obrázek 1.1: Přehled rozhleden v původním návrhu



Obrázek 1.2: Zobrazení panoramatu v původním návrhu (1)



Obrázek 1.3: Zobrazení panoramatu v původním návrhu (2)

Databáze aplikace je realizována pomocí CSV souborů, které jsou uloženy v zařízení. Soubory obsahují data, která jsou parsována do podoby, popsané výše. Struktura souborů je popsána v práci [26]. V době prezentace práce bylo v databázi k dispozici 6 rozhleden.

1.1.2 Nedostatky

Na první pohled je zřejmým nedostatkem vizuální podoba uživatelského prostředí. Tomu dominují převážně černá barva na bílém pozadí, což není pro oko příjemné. Informace jsou zobrazeny v jednom grafickém prvku naráz, přičemž navigace probíhá pomocí rolování obrazovky. To způsobuje nepřehlednost.

Další problém úzce souvisí s problémem prvním. Je jím ovladatelnost aplikace. Přepínání mezi jednotlivými pohledy je realizováno pomocí naklápění telefonu na levý, respektive pravý bok. Tento způsob je funkční, ačkoliv je dost nepraktický. Uživatel nemůže takovéto chování aplikace implicitně předpovídat a stává se tedy nelogickým krokem při programování.

Problém, který uživatel pocítí při práci s aplikací, je nemožnost filtrace a vyhledávání v datech. Jak již bylo řečeno výše, v době vydání aplikace bylo v databázi 6 rozhleden. Uživatel tedy nemusel v datech složitě filtrovat a problém nepocítil. V případě větší databáze by však byl tento nedostatek na obtíž.

Dalším minusem aplikace, který je nepříjemný hlavně programátorovi, je složitá struktura kódu. K jednotlivým komponentám je přistupováno postupy, používanými v minulosti. Tím se kód stává méně přehledným, narůstá jeho objem a tím celý běh programu zpomaluje. Případné aktualizace a vylepšení kódu se tak stávají složitějšími na implementaci.

Je logické, že informační data lze do databáze uložit bez větších problémů. I přesto je však před jejich zanesením nutné rozhlednu navštívit. To se stává jedním ze dvou hlavních problémů aplikace. Je to proto, že data o okolním panoramatu jsou sbírány přímo z místa rozhledny. Postup, jak je s daty manipulováno, je popsáno v další podkapitole.

Posledním minusem aplikace jsou nepřesnosti při zobrazení panoramatu. Samotný návrh algoritmu není dostatečně kvalitní na to, aby byly nepřesnosti odstraněny v přijatelné míře. Aplikace také pracuje s kompasem, který nevyužívá všechny možnosti čidel, které by mohl. Tím pádem vznikají při prohlížení panoramatu odchylky, které jsou pro body ve větší vzdálenosti okem viditelné. Zároveň zde vzniká problém při požadavku zobrazení bodů, které leží ve velmi podobném úhlu pohledu, pouze každý v jiné vzdálenosti.

1.1.3 Způsob vyjádření bodu panoramatu

Bod panoramatu je základním prvkem v panoramatu. Je-li dána množina těchto bodů pro danou rozhlednu, lze z ní poté sestavit panorama této rozhledny. Množiny bodů jsou pro každou rozhlednu uloženy v databázi zvlášť. Bod panoramatu v původním návrhu aplikace vznikne tak, že pozorovatel, který stojí na rozhledně, zapíše

azimut a název místa, které chce do databáze zanést. Tímto postupným přidáváním bodů vznikne panorama vyhlídkového místa. Pokud by programátor aplikace chtěl získat panoramata velkého množství rozhleden, musí vynaložit nemalé úsilí. To je tedy dalším důvodem, proč je třeba systém přepracovat tak, aby bylo možné databázi míst snadno rozšiřovat.

1.1.4 Poznatky

Poznatky z aplikace byly čerpány především z jejího testování. Osoby, které aplikaci testovaly, objevily většinu nedostatků, zmíněných v této kapitole výše.

Prvním bodem je přepracování uživatelského rozhraní, které by mělo být intuitivní, jednoduché pro navigaci a graficky přívětivé.

Dalším bodem je implementace fitrování a třídění v datech, která by dle nového požadavku měla být snadno aktualizovatelná, a tak je zde možnost jejich prudkého nárůstu.

V neposlední řadě bude třeba zjednodušit způsob kódování aplikace, aby se případné aktualizace a zavedení nových funkčních modulů obešlo bez komplikací. Ty by mohly vyvrcholit v kompletní přepis celé struktury aplikace. Ta se tímto krokem také výkonnostně posune vpřed.

Dále je třeba hledat nový způsob, jak vyjádřit panorama rozhleden. Především potom, jak zpřesnit zobrazení a vyřešit problém překrývajících se míst ve výhledu. Zároveň s tím je dobré vyvinout způsob, jak by se uživatel o bodu panoramatu mohl dozvědět více, než je jeho název a nadmořská výška.

Na závěr bude dobré vyřešit způsob, jakým budou data o bodech panoramatu sbírána a strukturu jejich uložení v databázi.

1.2 Hledání ideálních prostředků

Tato podkapitola se zabývá hledáním ideálních prostředků pro úspěšnou realizaci aplikace. Tyto prostředky by tedy měly napomoci k tomu, aby byla aplikace naprogramována jednoduše a způsobem, který by umožňoval její bezproblémovou údržbu i modifikaci.

Prostředky by zároveň měly zajistit co nejmenší odchylku od reality. To znamená, že při realizaci zprostředkování panoramatu uživateli by měly být zobrazené informace na adekvátních místech. To se týká pedevším nalezení a použití správných typů čidel mobilního zařízení.

V této části práce jsou tedy zmíněny a diskutovány existující technologie, spolu se zdůvodněním jejich následného výběru.

1.2.1 Přehled existujících technologií

WMM

Země je jako obří magnet. Na každém místě na jejím povrchu nebo nad ním má její magnetické pole více či méně známý směr, který může být použit jako polohový

systém pro orientaci lodí, letectva, satelitů, v anténách nebo také vrtných zařízeních. Na některých místech této planety se horizontální směr magnetického pole shoduje se směrem geografického (pravého) severu. Tento případ však není obecně platný.

Hodnota úhlu, o kterou se směr magnetického severu od hodnoty geografického severu liší, se nazývá magnetická deklinace, magnetická odchylka nebo také jednoduše deklinace. Je to tedy hodnota, potřebná k převodu mezi magnetickým a geografickým severem.

Hlavním účelem, proč byl World Magnetic Model vytvořen, je stanovení magnetické deklinace pro jakoukoliv konkrétní polohu na zeměkouli. Kromě výpočtu deklinace umí WMM také informovat o kompletní geometrii magnetického pole od 1 km pod zemským povrchem, až do 850 km nad ním. Magnetické pole zasahuje jak hluboko do povrchu Země, tak i daleko do vesmíru, ale WMM není pro tyto extrémní případy platný.

Zemský magnetismus má několik zdrojů. Všechny tyto zdroje mají bezpochybně na daný instrument vliv, ale do WMM jsou zahrnuty vlivy pouze některé. Z daleka největší podíl na magnetickém poli má vnější zemské jádro. To je složeno především z polotekutého železa. Magnetické minerály v krustě a horním plášti jsou dalším znatelným zdrojem. Elektrické proudy, vyvolané prouděním mořské vody mají na pole také znatelný, ačkoliv slabší vliv. Všechny uvedené jevy jsou v modelu zahrnuty.

Z modelu byl záměrně vyloučen vliv takzvaných rušivých polí, které se vyskytují ve vyšších vrstvách atmosféry a v bezprostřední blízkosti prostoru Země. Tato pole jsou časově proměnlivá. Jejich další vlastností je indukování elektrických proudů pod povrchem Země a v oceánech. Tyto proudy vytvářejí sekundární vnitřní magnetická pole, která jsou obecně považována za pole rušivá, a proto nejsou ve WMM zahrnuta.

Jak je uvedeno v [3], matematická metoda, použitá pro vývoj WMM, spočívá v rozvoji magnetického napětí do sférických harmonických funkcí a stupňů. Tento rozvoj je dvanáctého řádu. Magnetické pole zemského jádra se v čase procentuálně mění. Tato vlastnost je nazvána sekulární variace. V modelu je reprezentována lineárním SV modelem. Model zajišťuje časovou nezávislost koeficientů magnetického napětí. Avšak, z důvodu nepředvídatelných, nelineárních změn magnetického pole jádra, musí být hodnoty koeficientů WMM každých 5 let aktualizovány. V této práci je použit model, platný pro období v letech 2015 až 2020.

Vektor geomagnetického pole B_m je popsán sedmi prvky tak, jak je vyobrazeno na 1.4. Jsou jimi severní intenzita X, východní intenzita Y, vertikální intenzita Z (kladná směrem dolů). Další prvky jsou poté odvozeny z X, Y, Z: horizontální intenzita H, celková intenzita F, inklinační úhel I (kladný směrem dolů) a deklinační úhel D (také nazýván magnetická variace). Úhel D je měřen po směru hodinových ručiček, od geografického severu, k horizontální složce vektoru pole.

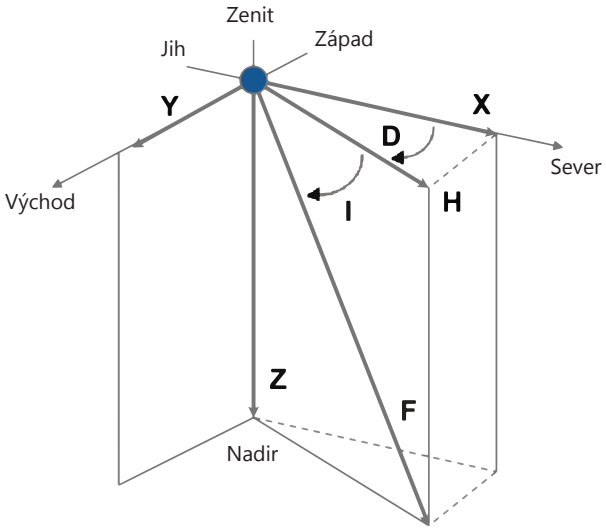
V polárních oblastech nebo poblíž osy rotace Země se úhel D silně mění, a to spolu se změnou zeměpisné délky pozorovatele. Zároveň je zde slabá měřitelnost směru vektoru B_m. Z tohoto důvodu se vědci, pracující na WMM rozhodli definovat

pomocný úhel GV, definující směr B_m ve vodorovné rovině. Jeho definice je:

$$GV = \begin{cases} D - \lambda & \phi < 55°, \\ D + \lambda & \phi > 55°, \\ nedefinováno & jinak, \end{cases} \tag{1.1}$$

kde
λ je zeměpisná délka [°],
ϕ je zeměpisná šířka [°].

Koeficienty modelu, také nazývané Gaussovy koeficienty, jsou obsaženy v [3]. Tyto koeficienty mohou být použity pro výpočet složek magnetického pole Země a jejich ročních přírůstků v jakémkoliv místě poblíž Zemského povrchu, mezi daty 2015.0 až 2020.0. Koeficienty hlavního pole jsou vyjádřeny v jednotkách nT. Koeficienty SV jsou v jednotkách nT/rok. Index n je stupeň a index m je řád. Jelikož nejsou koeficienty $h_n^m(t_0)$ and $\dot{h}_n^m(t_0)$ definovány pro hodnotu $m = 0$, jsou odpovídající pole ponechána prázdná.



Obrázek 1.4: Sedm prvků geomagnetického pole Země, přiřazených libovolnému bodu v prostoru

Nepočítáme-li chyby v měření, způsobené lidskou chybou, existují dva zdroje odlišností mezi pozorováním magnetického pole a WMM. První je dán nepřesnostmi v koeficientech modelu. Druhý zdroj je způsoben faktem, že WMM nezahrnuje všechna měření pole.

První zdroj je tedy suma chyb, způsobených nepřesnostmi v koeficientech, které popisují hlavní pole v roce 2015.0 a koeficientů sekulární variace, jež popisují lineární část variace pole, v období od 2015.0 do 2020.0. Nepřesnosti byly od počátku 2. tisíciletí ve velké míře redukovány, za pomoci sady velmi přesných satelitů, provádějících průzkum magnetického pole. Avšak, v případě sekulární variace se vyskytuje

ještě dodatečná chyba způsobená tím, že skutečná sekulární variace není čistě lineární. Tento fakt je dán změnami v proudění tekutin ve vnějším zemském jádru, a tak jsou změny v magnetickém poli Země lehce nelineární. Nelineární část sekulární variace je v současné době nepředvídatelná, ale naštěstí je odchylka od lineární části malá. Z toho vyplývá, že zkoumáním pole po řadu let je možné přesně popsat stávající pole a míru jeho změny, kterou lze poté lineárně extrapolovat v rámci několika budoucích let. Za předpokladu, že jsou k dispozici vhodná satelitní magnetická pozorování, je predikce WMM vysoce přesná, a to od data vydání modelu. Poté se přesnost během 5 leté epochy životnosti modelu postupně zhoršuje, až do okamžiku, kdy jsou koeficienty modelu nahrazeny novými.

Druhý zdroj odchylek od skutečného popisu pole je dán částmi geomagnetického pole, které nemohou být popsány modelem WMM. Buď je jejich prostorové měřítko příliš malé, nebo mají příliš krátké časové měřítko. Největší podíl na tom mají nestálá rušivá pole, vyskytující se v ionosféře a magnetosféře. Další vliv na odlišnosti mají také anomálie, produkované horní vrstvou zemské kůry. Odchylky mezi skutečnou hodnotou magnetické deklinace a hodnotou, spočtenou za pomoci modelu, mohou přesáhnout až 10 stupňů. Odchylky, dosahující takových hodnot jsou velice neobvyklé, ale existují. Deklinační anomálie v řádu 3 až 4 stupňů nejsou neobvyklé, ale jsou obvykle velmi malého prostorového rozsahu.

Pro vytvoření přesného modelu je nutné, aby měření vektorových komponent podléhalo dobré míře globálního pokrytí a minimálnímu šumu. Tři satelity, které byly v rámci European Swarm Mission vypuštěny na oběžnou dráhu v listopadu 2013, jsou v současné době nejvhodnějšími prostředky pro zkoumání geomagnetického pole. Satelity krouží kolem Země ve výšce stovek kilometrů nad povrchem. Kromě těchto dat jsou k dispozici i pravidelná měření z povrchu, i když ne v takové hustotě. Všechna naměřená data jsou následně rozdělena do tří kategorií, statisticky zpracována a jejich vliv zanesen do modelu tak, jak je podrobně popsáno v [3].

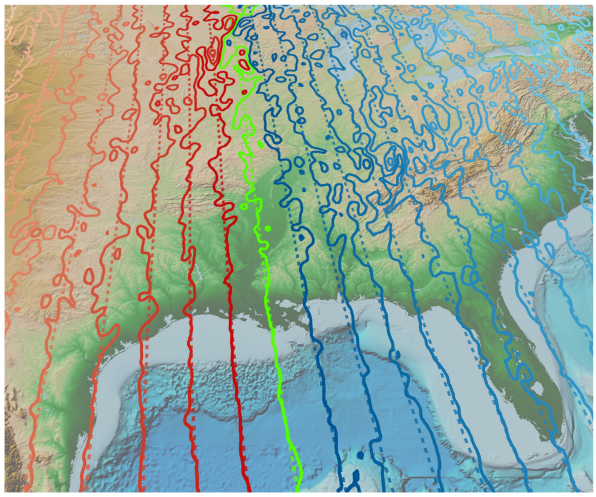
EMM

V návaznosti na WMM byl vytvořen také Enhanced Magnetic Model, který se vyznačuje zvýšenou přesností výpočtu. Tak, jako World Magnetic Model, i tento model pracuje na stejném principu rozvoje. Hlavním rozdílem v průběhu výpočtu je však fakt, že EMM dělá rozvoj řádu 720, narozdíl od WMM, který počítá pouze s rozvojem řádu 12. Tím se vypočtená data zpřesňují, jak lze vidět na ilustraci 1.5. Z obrázku lze vidět, že datové údaje o magnetické deklinaci z EMM (plná křivka), jsou odlišné od dat z WMM (přerušovaně). Barva dat poté vyznačuje pozitivní (červeně), negativní (modře) a nulovou deklinaci (zeleně).

Jako datový zdroj modelu slouží stejná databáze naměřených dat. Navíc jsou tato data obohacena o námořní měření.

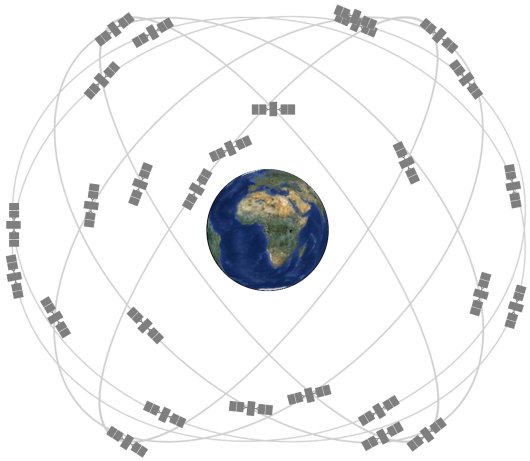
GPS

Globální navigační systém, neboli GPS, je služba Spojených států amerických, která poskytuje uživateli poziční, navigační a časovací prostředky. Systém se skládá ze tří segmentů: Kosmický segment, Řídící segment a Uživatelský segment.



Obrázek 1.5: Porovnání magnetické deklinace mezi WMM a EMM

Kosmický segment obsahuje soustavu satelitů, které uživateli posílají radiové signály. Spojené státy garantují [18], že po 95% času bude dostupných alespoň 24 operačních satelitů. V zájmu dodržení této přísahy létá nad Zemí po dobu několika let již 31 satelitů. Nadbytečné satelity slouží jako záloha při servisu či odstávce základních satelitů. Satelity létají na oběžné dráze ve výšce přibližně 20200 km. Každý satelit obkrouží Zemi dvakrát za den. Dráha satelitů je rozložena do šesti prostorově totožných křivek tak, jak je zobrazeno na obrázku 1.6. Každá křivka obsahuje alespoň 4 satelity. Tento systém zajišťuje, že uživatel může být viděn vždy alespoň čtyřmi satelity, ať už se nachází kdekoliv na planetě.



Obrázek 1.6: Rozšiřitelná poloha 24 satelitů dle standardu SPS Performance

Řídící segment je složen z několika středisek, sídlících po celé Zemi. Ty denně sledují dráhu satelitů, monitorují jejich komunikaci, provádí analýzu a posílají data s příkazy celé soustavě. Příkladem může být příkaz pro korekci času. Jednotky Air

Force mají za úkol nahrazovat vysloužilé satelity novými, pokud je třeba. Nové satelity samozřejmě nabízí vylepšenou přesnost a spolehlivost.

Uživatelský segment byl postupně vyvinut pro potřeby stovek aplikací moderního života. V současné době lze technologii GPS nalézt kromě zřejmých případů například v hodinkách, stavebních strojích, dopravních kontejnerech či bankomatech. Technologie také urychlila vývoj širokého spektra ekonomických aplikací, oblasti farmářství, konstrukce, těžebního průmyslu a logistiky. Hlavní komunikační sítě, bankovní systémy, burzy a elektrické sítě také významně spoléhají na GPS, a to kvůli časové synchronizaci.

Dobrým příkladem komunikace může být právě případ rozvodu elektrické energie. Základní požadavek na časovou a frekvenční přesnost pro efektivní přenos a distribuci elektrické energie je velmi vysoký. Opakované výpadky energie byly jen demonstrací a vodítkem k faktu, že je potřeba elektrickou síť časově synchronizovat. Analýza těchto výpadku vedla mnohé firmy k umístění synchronizačních zařízení, fungujících na principu GPS, do továren a trafo stanic. Tím je možná precizní analýza elektrických anomálií, které lze zpětně vypátrat a v dalších případech jim předejít.

GPS satelity vysílají radiové signály s jejich polohou, statusem a přesným časem t_1 z atomových hodin, instalovaných na desce satelitu. Radiové signály cestují vzduchem rychlostí světla c, tedy více než 299,792 $\frac{km}{s}$. Zařízení s GPS přijme radiové signály a vezme v potaz přesný čas přijetí informace t_2. Informace o časech jsou použity pro kalkulaci vzdálenosti od každého satelitu, od kterého byly přijaty. Jakmile zařízení ví vzdálenost alespoň od čtyř satelitů, může použít geometrické výpočty pro detekci polohy na Zemi, a to ve třech dimenzích. Pro výpočet vzdálenosti zařízení od satelitu je použit následující vztah:

$$d = c * (t_2 - t_1), \tag{1.2}$$

kde
d je vzdálenost zařízení od satelitu [km],
c je rychlost světla [$\frac{km}{s}$],
t_2 je čas přijetí informací zařízením,
t_1 je čas odeslání informací satelitem.

Vztahy pro výpočty s GPS souřadnicemi

Existuje soubor matematických vztahů [27], které uživateli umožňují počítat se souřadnicemi, obsahující údaj o zeměpisné šířce a zeměpisné délce. Tyto vztahy jsou aplikovány pro výpočty na kulové ploše, což je pro většinu případů dostatečně přesné. Země je ve však skutečnost lehce elipsoidická, a tak se zde vyskytují chyby, typicky maximálně do 0,3%. Efekt elipsoidy je tedy ignorován.

První z kategorie těchto vztahů je pro výpočet ortodromy mezi dvěma body. Ortodromická vzdálenost je nejkratší spojnice dvou bodů na kulové ploše. Jsou samozřejmě ignorovány veškeré nerovnosti, jako například hory či naopak nížiny.

Jednou z možností, jak tuto vzdálenost vypočítat, je užití *Haversine formula*. Vztahy vypadají následovně:

$$\begin{aligned} a &= \sin^2(\Delta\phi/2) + \cos\phi_1 * \cos\phi_2 * \sin^2(\Delta\lambda/2), \\ c &= 2 * \text{atan2} *(\sqrt{a}, \sqrt{1-a}), \\ d &= R * c, \end{aligned} \tag{1.3}$$

kde
ϕ je zeměpisná šířka [rad],
λ je zeměpisná délka [rad],
R je poloměr Země [km].

Tento vzorec je velmi vhodný pro numerické výpočty i na velké vzdálenosti, narozdíl od dalších vztahů této kategorie, zmíněných níže. Pro doplnění je dobré zmínit, že c je úhlová vzdálenost v radiánech a a vyjadřuje čtverec o polovině délky tětivy mezi počítanými body.

Další možností je použití *Sférické Kosinovy věty*. Na malé vzdálenosti ho lze použít jako velmi přesné měřítko a je také používán v programovacích jazycích. Vyjádření vypadá takto:

$$d = \arccos(\sin(\phi_1) * \sin(\phi_2) + \cos(\phi_1) * \cos(\phi_2) * \cos(\Delta\lambda)) * R, \tag{1.4}$$

kde
ϕ je zeměpisná šířka [rad],
λ je zeměpisná délka [rad],
R je poloměr Země [km].

Třetí z možností, jak vzdálenost mezi dvěma GPS souřadnicemi vypočítat, je *Ekvidistantní válcová aproximace*. Tuto možnost lze použít, pokud je třeba rychlého výpočtu, avšak za cenu nižší přesnosti. Výpočet je popsán vztahem 1.5 a aplikuje Pythagorovu větu na *Ekvidistantní válcovou projekci*. Podél poledníků se žádné chybové odchylky nevyskytují, v ostatních případech závisí na vzdálenosti, směru a zeměpisné šířce.

$$\begin{aligned} x &= \Delta\lambda * \cos((\phi_1 + \phi_2)/2), \\ y &= \Delta\phi, \\ d &= R * \sqrt{x^2 + y^2}. \end{aligned} \tag{1.5}$$

Použitými proměnnými jsou:
ϕ je zeměpisná šířka [rad],
λ je zeměpisná délka [rad],
R je poloměr Země [km].

Další kategorií výpočtů je výpočet směru mezi dvěma souřadnicemi. Obecně platí fakt, že směr se s měnící pozicí na ortodomu bude měnit. A tedy také v zásadě platí, že na cestě mezi místy a a b bude směr z místa b odlišný od směru z místa a. Vztah

1.6 popisuje výpočet počátečního směru. Pro výpočet cílového směru je třeba otočit GPS souřadnice a obrátit směr použitím $\Phi = (\Phi + 180)\%360$.

$$\Phi = \text{atan2}(\sin\Delta\lambda * \cos\phi_2, \cos\phi_1 * \sin\phi_2 - \sin\phi_1 * \cos\phi_2 * \cos\Delta\lambda), \tag{1.6}$$

kde
Φ je směr druhého bodu [rad],
ϕ je zeměpisná šířka [rad],
λ je zeměpisná délka [rad].

Za další typ výpočtu lze označit výpočet středového bodu mezi dvěma souřadnicemi, kde je vztah vyjádřen následovně:

$$\begin{aligned} B_x &= \cos\phi_2 * \cos\Delta\lambda, \\ B_y &= \cos\phi_2 * \sin\Delta\lambda, \\ \phi_m &= \text{atan2}(\sin\phi_1 + \sin\phi_2, \sqrt{(\cos\phi_1 + B_x)^2 + B_y^2}), \\ \lambda_m &= \lambda_1 + \text{atan2}(B_y, \cos\phi_1 + B_x). \end{aligned} \tag{1.7}$$

Význam identifikátorů proměnných se nemění, a je tedy shodný s předchozími případy. Jak bylo zmíněno výše, počáteční směr mezi dvěma body se liší od cílového. Je tedy třeba si uvědomit, že středový bod ve většině případů není na středu cesty mezi souřadnicemi – tedy středový bod mezi 35°N, 45°E a 35°N, 135°E je někde kolem 45°N, 90°E.

Další typem je výpočet cílové destinace, je-li zadán počáteční bod, vzdálenost a směr. Vztah je popsán postupem 1.8.

$$\begin{aligned} \delta &= \tfrac{d}{R}, \\ \phi_2 &= \arcsin(\sin\phi_1 * \cos\delta + \cos\phi_1 * \sin\delta * \cos\Delta), \\ \lambda_2 &= \lambda_1 + \text{atan2}(\sin\Delta * \sin\delta * \cos\phi_1, \cos\phi_1 * \sin\phi_2). \end{aligned} \tag{1.8}$$

Ve vztahu jsou použity:
ϕ je zeměpisná šířka [rad],
λ je zeměpisná délka [rad],
Δ je směr (kladný po směru hodinových ručiček) [rad],
δ je úhlová vzdálenost [-],
d je zadaná vzdálenost,
R je poloměr Země.

Posledním, pro tuto práci potencionálně využitelným výpočtem, je výpočet průsečíku mezi dvěma body se zadanými směry. Postup, který je oproti ostatním dosud

zmíněným komplexnější, je vyjádřen jako:

$$
\begin{aligned}
&\delta_{12} = 2 * \arcsin(\sqrt{\sin^2(\tfrac{\Delta\phi}{2}) + \cos\phi_1 * \cos\phi_2 * \sin^2(\tfrac{\Delta\lambda}{2})});\\
&\Phi_a = \arccos(\sin\phi_2 - \sin\phi_1 * \cos\delta_{12} / \sin\delta_{12} * \cos\phi_1);\\
&\Phi_b = \arccos(\sin\phi_1 - \sin\phi_2 * \cos\delta_{12} / \sin\delta_{12} * \cos\phi_2);\\
&\\
&když\ \sin(\lambda_2 - \lambda_1) > 0\\
&\ \ \Phi_{12} = \Phi_a;\\
&\ \ \Phi_{21} = 2 * \pi - \Phi_b;\\
&jinak\\
&\ \ \Phi_{12} = 2 * \pi - \Phi_a;\\
&\ \ \Phi_{21} = \Phi_b;\\
&konec\ když\\
&\\
&\alpha_1 = (\Phi_{13} - \Phi_{12} + \pi)\%2 * \pi - \pi;\\
&\alpha_2 = (\Phi_{21} - \Phi_{23} + \pi)\%2 * \pi - \pi;\\
&\\
&\alpha_3 = \arccos(-\cos\alpha_1 * \cos\alpha_2 + \sin\alpha_1 * \sin\alpha_2 * \cos\delta_{12});\\
&\delta_{13} = \operatorname{atan2}(\sin\delta_{12} * \sin\alpha_1 * \sin\alpha_2, \cos\alpha_2 + \cos\alpha_1 * \cos\alpha_3);\\
&\phi_3 = \arcsin(\sin\phi_1 * \cos\delta_{13} + \cos\phi_1 * \sin\delta_{13} * \cos\delta_{13});\\
&\Delta\lambda_{13} = \operatorname{atan2}(\sin\Phi_{13} * \sin\delta_{13} * \cos\phi_1, \cos\delta_{13} - \sin\phi_1 * \sin\phi_3);\\
&\lambda_3 = (\lambda_1 + \Delta\lambda_{13} + \pi)\%2 * \pi - \pi;
\end{aligned}
\tag{1.9}
$$

Ve výpočtu je použito:
$\phi_1,\ \lambda_1,\ \Phi_1$ jako souřadnice prvního bodu a počáteční směr,
$\phi_2,\ \lambda_2,\ \Phi_2$ jako souřadnice druhého bodu a počáteční směr,
$\phi_3,\ \lambda_3$ jako souřadnice průsečíku.

Je třeba dodat, že výpočet má nekonečno řešení, pokud $\alpha_1 = 0$ a zároveň $\alpha_2 = 0$. Pokud $\sin\alpha_1 * \sin\alpha_2 < 0$, má výpočet dvě řešení.

Google Maps APIs

Google Maps APIs je rozhraní dostupné pro Android, iOS, webové prohlížeče a skrze webové služby HTTP. Služby aplikačního rozhraní pro mobilní platformy nabízejí možnosti prohlížení map a informací o milionech indexovaných lokací. Dále umožňují navigaci mezi lokacemi, použitím HTTP dotazu a také převod adresy místa do podoby geografických souřadnic.

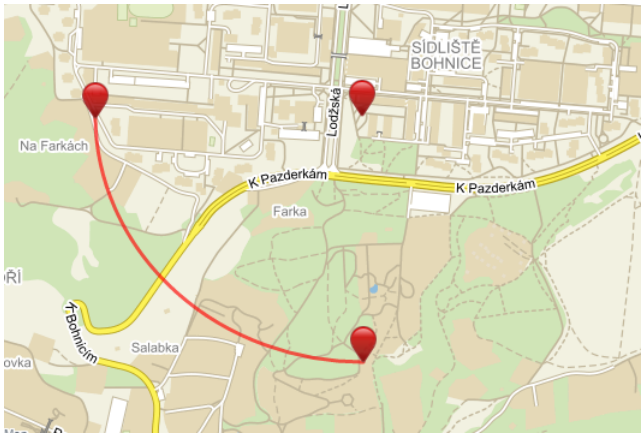
API, dostupné skrze služby HTTP umí kromě výše zmíněných možností také zjistit čas, který bude uživatel potřebovat pro cestu mezi dvěma destinacemi. V některých místech světa lze také sledovat dopravní informace a tím předejít komplikacím na cestě. Rozhraní také umí zjistit data o časovém pásmu pro jakékoliv místo na světě. Mimo toho je také schopno uživateli sdělit lokaci a nadmořskou výšku na základě dat z mobilních stanic a Wi-Fi uzlů.

Při implementaci práce by tedy bylo možné některé z výše zmíněných funkčních prvků použít jako prostředek pro zjišťování polohy a operování s geosouřadnicemi.

API Mapy.cz

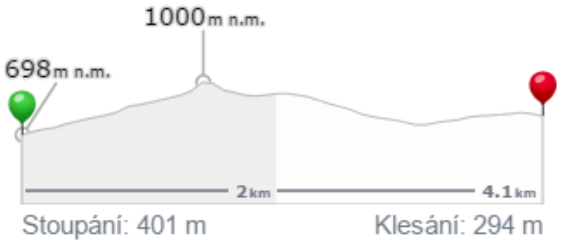
Aplikační rozhraní služby Mapy.cz je ve velké míře českým ekvivalentem Google Maps APIs, i když je dostupné pouze prostřednictvím webového prohlížeče. Kromě funkcí samozřejmých, jako je zobrazení a prohlížení map, lze vkládat samostatné ovládací prvky, provádět počty a převody mezi souřadnicemi. Dále lze také pracovat se značkami, body zájmu a přepínat mezi vstvami. Lze také využívat geokódování, zjistit nadmořskou výšku bodu nebo také například zobrazit pohled z ulice.

Jednou z možností, kterou lze v implementaci efektivně využít, je práce s geometrií tak, jak je ukázáno na obrázku 1.7. Lze tak docílit ohraničení míst, která leží pouze v zadaném radiusu. Tím je možné zamezit případnému zanesení bodů, které kvůli jejich vzdálenosti od výhledového místa nelze vidět.



Obrázek 1.7: Možný způsob mapování podkladů rozhleden

Další možností je zobrazení profilu krajiny mezi dvěma body. Lze tak zmapovat tvar určitého místa, například pohoří. Vypočtenou křivku je poté možné zanést do zobrazení panoramatu. Tím uživatel získá přesnější odhad tvaru místa a ho spojí s popiskem, který na místo ukazuje. Příklad je uveden na obrázku 1.8.



Obrázek 1.8: Příklad profilu krajiny

1.2.2 Výběr technologií

Na základě nastudování výše uvedených bodů této podkapitoly byla učiněna rozhodnutí o tom, které z technologií budou použity pro vývoj aplikace. Jak lze vidět, alternativ je v některých případech více, záleží tedy pouze na volbě programátora, jakou zvolí.

Pro návrh a následnou implementaci algoritmu pro opravu odchylky mezi magnetickým a geografickým severním pólem Země byla zvolena alternativa WMM. Z popisu a obrázku 1.5 je sice vidět, že je EMM přesnější, ale vzhledem k porovnání přesnosti a výpočetní náročnosti bylo rozhodnuto pro implementaci algoritmu World Magnetic Model.

Pro lokalizaci uživatele bude použita technologie GPS. Současná mobilní zařízení ji mají implementovanou na velmi dobré úrovni.

V aplikaci bude třeba také provádět výpočty s GPS souřadnicemi. Na základě zhodnocení možných alternativ bylo dospěno k závěru, že bude použita vlastní naprogramovaná knihovna funkcí pro počítání s geosouřadnicemi. Jedním z důvodů tohoto rozhodnutí je fakt, že by bylo dobré proniknout do jádra věci, a zjistit tak, jak výpočty fungují. Dále vyzkoušet různé alternativy, jak danou věc spočítat a následně otestovat přesnosti pro odlišné případy.

Jelikož je jedná aplikaci primárně určenou pro rozhledny a vyhlídky v České republice, bylo pro sběr dat a okolním panoramatu vybráno API Mapy.cz. S pomocí webového prohlížeče budou body okolního výhledu evidovaných rozhleden zaneseny do databáze.

1.2.3 Přehled existujících sensorů

Akcelerometr

Akcelerometr je elektromechanické zařízení, používané pro měření síly zrychlení. Tyto síly mohou být jak statické, jako například působení gravitační síly, tak i dynamické, jak je tomu v mnoha případech při užívání mobilních zařízení. Příkladem může být snímání pohybu nebo vibrací. Akcelerace je veličina změny rychlosti v čase.

Akcelerometr je používán v mnoha odvětvích. Například chrání pevné disky notebooků před poškozením. V případě náhlého pádu zařízení při jeho používání je pevný disk okamžitě vypnut, což zabrání nárazu čtecích hlav.

Dalším příkladem akcelerometru může být orientace zařízení. Vezmeme-li v potaz hodnotu akcelerace, tak lze odvodit, jak se zařízení pohybuje. Tím je myšleno, zda se zařízení hýbe nahoru, horizontálně nebo například padá dolů. Aplikací v praxi může být například detekce třesení telefonu, která vyvolá akci zpět.

Akcelerometry se v základu dělí na dva typy. Prvním z nich je piezoelektrický akcelerometr, což je nejvíce používaná forma. Sensor používá mikroskopické krystalické struktury, které jsou vlivem akceleračních sil napínány. Tento jev na krystalech vytváří napětí, které akcelerometr převede na rychlost a orientaci.

Kapacitní akcelerometry vnímají změny kapacity mezi mikrostrukturami, které se při pohybu také pohybují. Následná změna kapacity je převedena na napětí a poté na akceleraci.

Zařízení jsou typicky vyráběny pro detekci akcelerace ve více osách. V mobilních zařízeních se jedná o detekci tří os, ktežto automobily používají typicky dvouosou detekci.

Gyroskop

Gyroskop je zařízení, které používá pro detekci orientace zemskou gravitaci. Skládá se z volně rotujícího disku, nazvaného rotor. Ten je připevněn k otáčejícím se ložiskům, která mají nepatrné tření. Gyroskop principiálně funguje tak, že roztočením setrvačníku je kladen odpor na změnu osy jeho rotace. Otáčející se setrvačník má moment hybnosti, takže jeho osa bez působení vnějších sil udržuje stále stejný směr. Díky tomuto jevu, zvanému gyroskopická akcelerace, lze sledovat změnu polohy setrvačníku vzhledem ke gyroskopu. Tím je možné spočítat změny polohy celého zařízení ve 3 osách.

V telefonech se na smínání polohy rotoru nepoužívá sledování polohy samotného gyroskopu. Místo toho se sleduje změna kapacity kondenzátorů, které plnohodnotně zastupují optické snímače z klasických gyroskopů. Tyto kondenzátory, kterých je několik set, se nacházejí na obalu gyroskopu.

Hlavní rozdíl mezi akcelerometrem a gyroskopem je v tom, že akcelerometr umí zaznamenat orientaci stacionárního objektu vzhledem k povrchu Země. Pokud je proveden pohyb v partikulárním směru, není akcelerometr schopen tento typ pohybu rozlišit. Oproti tomu gyroskop to umí. Při rotaci kolem partikulární osy, například v letadle, je zobrazena její aktuální míra, dokud není objekt zcela stabilizován.

Sensor gravitace

Uvažujme existenci tří sensorů. První z nich snímá akceleraci, druhý lineární akceleraci a třetím sensorem je snímač gravitace. Pro tyto tři sensory platí následující vztah:

$$akcelerace = lineární\ akcelerace + gravitace. \tag{1.10}$$

Ze vztahu plyne, že data snímače gravitace jsou součástí dat z akcelerometru. Pro jejich extrakci je použit Kálmanův filtr. Je to algoritmus, který z dat, zatížených nepřesnostmi a šumem, odhaduje neznámé hodnoty proměnných. Při extrakci dat bývá také, jako pomocný sensor, využit gyroskop a magnetometr.

Magnetometr

Magnetometry v mobilních zařízeních fungují tak, že měří hodnoty lokálního magnetického pole ve třech osách. Toto pole je tvořeno součtem geomagnetického pole a magnetického pole, tvořeného okolím, kde se zařízení nachází. Pro měření jsou použity principy magnetoresistence. To je schopnost materiálu změnit svůj elektrický odpor, vlivem vnějšího magnetického pole.

Obecně se jedná o zmagnetizovaný plíšek, u něhož víme, jaký má za daného proudu odpor (při nulovém vlivu externího magnetického pole). Změny odporu, ke

kterým dochází při změnách magnetického pole, pak lze spolehlivě změřit. Magnetometry mohou být rozděleny na skalární a vektorové. Skalární magnetometry měří pouze intenzitu magnetického pole. Vektorové měří kromě intenzity také jeho směr.

Díky miniaturizaci a cenové dostupnosti mohly být tyto součástky začleněny i do integrovaných obvodů mobilních telefonů. Zde jsou použity především jako elektronické kompasy. Jejich konstrukce je založena na dvou nebo třech polovodičových snímačích magnetického pole, poskytujících data mikroprocesoru. Ten pak vypočítá správnou orientaci za pomoci trigonometrie.

Kompas

Kompas je zařízení k určování světových stran. Typický kompas obsahuje volně pohyblivou magnetickou střelku, která se vlivem zemského magnetického pole natáčí ve směru magnetického severu a jihu. Elektronický kompas lze klasifikovat spíše jako aplikací výstupů z čidel, než jako čidlem samotným. V mobilních zařízeních je nejdůležitější řazení typu kompasu na základě využitých čidel.

Prvním způsobem, jak získat orientaci zařízení, je využitím dat ze základních, nízkoúrovňových sensorů. Dobrým příkladem je užití akcelerometru v kombinaci s magnetometrem. Problémem však je, že naměřená data obsahují značné nepřesnosti. To je dáno tím, že jsou použita přímo od zdroje. Tím se azimut stává nestabilní a nepoužitelný.

Možností, jak využitelnost zlepšit, je použít filtr typu dolní propust, který je aplikován na výstup z akcelerometru. To však nese své výhody i nevýhody. Díky tomu, že jsou data filtrována, bude ve výstupu existovat určité zpoždění. Je také třeba použít dobrou mezní hodnotu pro filtraci α. Výhodou naopak může být možnost uchování fronty rotačních matic, z čehož lze stanovit průměr.

Mimoto je také možné rozšířit filtr (a frontu hodnot) tak, aby byly adaptabilní. To znamená, že pokud uživatel stojí na místě, použije algoritmus pro redukci šumu vysokou hodnotu parametru α. Pokud je rozpoznán větší pohyb v nějakém směru, použije filtr hodnotu α nižší, aby data mohla být získána rychleji.

To, co se snaží dolní propust udělat, je izolovat gravitační složku z dat akcelerometru. Je-li ale v zařízení dostupný sensor gravitace, lze jej použít místo čidla akcelerace tak, jak je popsáno v [12].

Další možností, jak kompas vylepšit, je použít spolu s akcelerometrem a magnetometrem ještě gyroskop. Gyroskop má nízkou dobu odezvy a může dramaticky zlepšit měření. Tato technika, která využívá pro výpočet data z několika čidel, se nazývá *Sensor fusion*. Data, získaná pomocí této techniky se vyznačují větší přesností, úplností a spolehlivostí.

1.2.4 Výběr sensorů

Po prostudování možných alternativ pro sestrojení vyhovujícího kompasu pro orientaci v prostoru byla vybrána možnost fúze sensorů. Důvody jsou zřejmé. Pro korektní zobrazení panoramatu, což je jeden ze stěžejních bodů práce, je třeba dostatečně kvalitně setrojený kompas, jehož předností bude především přesnost.

Po uvážení faktu, že se uživatel aplikace nebude pohybovat rychle, ale bude se pomalu otáčet a prohlížet okolní krajinu, bylo rozhodnuto o použití kombinace akcelerometr, gyroskop a magnetometr. Data z akcelerometru budou navíc filtrována dolní propustí. Tato varianta by měla zajistit přesnost kompasu a zároveň udržet dostatečně rychlou odezvu na to, aby uživatel nemusel čekat na odfiltrování při samotném prohlížení panoramatu.

Dalším důvodem, proč byla tato kombinace zvolena, je skutečnost, že sensor gravitace není dostupný ve všech zařízeních, která by vyvíjená aplikace měla podporovat.

1.3 Hledání platformy klienta

1.3.1 Proč mobilní platforma?

Jak již bylo zmíněno v motivačních řádcích, mobilní technologie jsou v současné době neodmyslitelnou součástí životů. Naše chytrá zařízení u sebe nosíme prakticky neustále. Jsou využívána pro velmi široké spektrum aplikací, ať už informačních, přes aplikace monitorujících naše zdraví, až po aplikace, které nás mají zabavit.

Zařízení, postavená na mobilních platformách, jsou snadno programovatelná a obsahují velké množství čidel, které mohou sloužit jako prostředek pro realizaci nejrůznějších aplikací. Tato fakta jsou tedy jasnými důvody, proč byla pro vytvoření klienta aplikace vybrána mobilní platforma.

1.3.2 Přehled nejpoužívanějších mobilních platforem

Android

Jedná se o operační systém, založený na jádře Linuxu. Při vývoji systému byla brána v úvahu omezení, kterým podléhají mobilní zařízení (nižší výkonnost, dostupnost paměti a výdrž baterie). Jádro systému je navrženo pro běh na různém hardwaru, a proto může být využit bez ohledu na čipovou sadu, velikost nebo rozlišení obrazovky.

Samotná platforma poskytuje nejen operační systém s uživatelským rozhraním pro koncové uživatele, ale také kompletní nasazení operačního systému pro výrobce zařízení. Platforma zastupuje největší podíl na trhu s chytrými mobilními zařízeními. Operační systém je stále aktualizován. Zatím poslední verze, vydaná v srpnu 2016, má řadové označení 7.0 Nougat. Tato verze mimo jiné přinesla možnost práce s více okny najednou, řešení přidělování výkonu aplikacím dle potřeby a nové API pro virtuální realitu.

iOS

Tento operační systém byl vyvinut společností Apple Inc., speciálně pro hardwarová zařízení, dodávaná touto firmou. V současné době iOS pohání řadu mobilních

zařízení společnosti. Jedná se o druhý nejprodávanější operační systém, hned za Androidem. Uživatelské rozhraní je založeno na jednoduchosti manipulace.

Majoritní verze systému bývají vydávány s roční periodou vždy, když společnost představuje novou generaci svých zařízení. Poslední majoritní verze ze září 2016 je označena iOS 10. Narozdíl od Androidu, který je uživatelsky otevřenější, v iOS nelze provádět tolik uživatelských změn. Příkladem může být použití widgetů. I přesto je ale operační systém iOS považován za designový vrchol, veškeré uživatelské prvky jsou propracovány do detailu. Minusem je, že vývoj aplikací pro iOS lze provádět převážně pouze v nástrojích, dostupných na počítačích značky Apple.

Windows Phone

Windows Phone je operační systém od firmy Microsoft. Jeho první verze byla vydána v roce 2010. V současné době jsou verze systém označeny jako Windows Phone 7 a Windows Phone 8. Ty však nejsou, narozdíl od různých verzí výše uvedených systémů, vzájemně kompatibilní. Dokonce jsou poháněny kompletně jinými jádry. Microsoft, podobně jako Apple, vsází na komunikaci mezi svými zařízeními, a tak umí Windows Phone dobře komunikovat například s herní konzolí Xbox.

Tento operační systém je třetí nejrozšířenější, hned za svými konkurenty. Z pohledu programátora lze aplikace vyvíjet snadno, a to pomocí sady nástrojů, která je součástí vývojového prostředí Visual Studio. Je však dobré podotknout, že možnosti mobilních zařízení, pracujících na této platformě, bývají často omezenější, než jak je tomu u předchozích dvou operačních systémů.

1.3.3 Výběr platformy klienta

Jako cílová platforma pro vyvíjenou aplikaci byl vybrán operační systém Android. Prvním důvodem výběru je jeho rozšířenost. Tento systém je nejrozšířenějším na trhu s mobilními zařízeními. Z toho plyne, že aplikací bude oslovena největší komunita.

Dalším důvodem výběru je programovatelnost aplikací. Ty jsou nejčastěji vyvíjeny v jazyce Java a spolu s SDK Tools poskytují všechny potřebné prostředky pro programování.

Posledním důvodem je dokumentace. Nástroje pro vývoj jsou kvalitně zdokumentovány, včetně ukázek a rad na oficiálních stránkách frameworku. Kromě zmíněné alternativy vývoje existuje ještě možnost programovat aplikace v jazyce C# ve vývojovém prostředí Visual Studio, prostřednictvím frameworku Xamarin. Tento způsob však není tolik používaný. Navíc, Android Studio je vyladěno přesně pro účely vývoje aplikací pro operační systém Android.

1.4 Hledání serverových technologií

Následující část práce je zaměřena na serverovou část aplikace, respektive na přehled technologií pro popis dat a následnou volbu vhodné datové struktury. Dále je diskutován výběr serveru z pohledu programovacího jazyka. Jsou také zohledněny

prostředky, potřebné ke komunikaci s klientskou částí aplikace. Veškeré závěry jsou zdůvodněny.

1.4.1 Přehled existujících technologií pro popis dat

JSON

JSON je odlehčený formát pro výměnu dat. Je snadno zapisovatelný i čitelný člověkem. Zároveň ho lze jednoduše analyzovat a generovat strojově. Je založen na podmnožině jazyka JavaScript. JSON je textový formát, který je zcela nezávislý na jazyce. Formát však využívá konvence, zavedené v jazycích rodiny C, a díky tomu je tedy ideálním jazykem pro výměnu dat.

JSON je založen na dvou strukturách. První z nich je kolekce párů název/hodnota. Tu je možné v různých jazycích prezentovat například jako objekt nebo asociativní pole. Druhou strukturou je seřazený seznam hodnot, který je v jazycích realizován většinou jako pole nebo seznam.

Jedná se o univerzální datové struktury, které jsou v nějaké formě podporovány prakticky všemi moderními programovacími jazyky. Je tedy logické, aby na nich byl založen i na jazyce nezávislý výměnný formát. Mezi výhody jazyka JSON patří čitelnost. Je to především proto, že nevyužívá ukončovacích tagů, jako je tomu u XML. Tím pádem je i zápis stejných dat ve formátu JSON kratší. Další výhodou při vývoji webových aplikací je, že pro parsování lze použít nativní funkci jazyka JavaScript, narozdíl od XML, kde je nutné použít parser. Pro popis souboru tohoto typu se používá JSON Schema.

XML

XML je obecný značkovací jazyk, který byl původně navrhnut za účelem publikování dokumentů. V současnosti je používán jako formát pro serializaci a výměnu dat, stejně tak, jako JSON. Práce s XML je podporována řadou programovacích jazyků a nástrojů.

Jazyk je vyznačuje tím, že všechny elementy jsou párové, a tedy musí být uzavřeny. Formát je základem mnoha standardů, například protokolů uPnP, používaných pro komunikaci s moderními domácími elektronickými zařízeními. Dále bývá používán v textových formátech (například ODF) nebo grafických formátech (například SVG).

Pro popis konkrétního souboru typu XML se používá XML Schema. Je také podporována takzvaná XSL transformace, která dle požadavků uživatele převede soubor do jiné podoby. Mezi souborem typu JSON a XML existuje vzájemný jednoznačný převod.

YAML

YAML je další z formátů pro serializaci strukturovaných dat. Mezi jeho rysy, podobně jako u předchozích formátů, patří dobrá čitelnost – jak strojem, tak i člověkem, a také neomezený počet úrovní vnoření. Narozdíl od nich je však struktura

a hierarchie dat řešena indetací, neboli předsazením. Indetace o jednu úroveň je realizována pomocí dvou nebo čtyř mezer. Tabulátory nejsou povoleny.

Kromě přenosu dat bývá také používán jako formát pro konfigurační soubory. V YAMLu jsou povoleny uživatelské datové typy, ale nativně kóduje pouze skaláry, jako jsou řetězce, celá a plovoucí čísla, dále pak seznamy a asociativní pole. Tyto datové struktury vycházejí z jazyka Perl.

SQL

SQL je obecným nástrojem pro manipulaci, správu a organizování dat, uložených v databázi počítače. Je určen především uživatelům, i když je v mnoha směrech využíván i tvůrci aplikací. Je adaptovatelný pro jakékoliv prostředí.

SQL není pouze dotazovacím jazykem, jeho prostřednictvím lze také data definovat, následné entity naplnit a vytvořit vztahy a organizaci mezi položkami dat. Jazyk dále umožňuje řídit přístup k datům, a tak udělovat či odebírat oprávnění k přístupu na různých úrovních. Tím jsou data chráněna před náhodnou či úmyslnou nesprávnou manipulací.

Jazyk se používá ve vhodném prostředí k okamžitému řešení úloh, nejčastěji prostřednictvím dotazů. Další možností je vkládání příkazů přímo do hostitelského jazyka. Z důvodu, že SQL neobsahuje ve většině implementací řídící programové konstrukce a další prvky, které by měl obecný programovací jazyk mít, jedná se o neplnohodnotný jazyk, a je tedy klasifikován jako standardizovaný nástroj pro práci s relačními databázemi.

Jazyk umožňuje získat odpovědi i na velmi komplikované dotazy téměř okamžitě. Jde o nástroj neprocedurální, k datům přistupuje jako k množině. Relační databázi uživatel vmíná jako data v podobě soustavy provázaných tabulek. Každá z nich představuje množinu údajů, která je uspořádána v řádcích a sloupcích. Řádek představuje záznam a každý sloupec položku záznamu. Na hodnotu je odkazováno jako na prvek v matici.

Vybraná množina je ve většině případů množina dat z jedné nebo více tabulek. Množina slouží převážně jako vstupní údaje pro další zpracování (například vykreslení grafu). Jazyk je používán také pro vytváření náhledů. To umožňuje pro různé typy uživatelů vytvořit pohled pouze na taková data, která má vidět. Data v náhledech jsou dynamická, a tak jsou vázána na jakoukoliv změnou v tabulkách, že kterých je pohled vytvořen. Tato dynamičnost platí i v opačném vztahu.

NoSQL

Databáze typu NoSQL nejsou primárně postaveny na tabulkách, a tak pro práci s daty nemusí využívat SQL. Jejich přínosem je optimalizace pro vyhledávání, avšak na úkor funkcionality. Ta je často omezena pouze na prosté ukládání dat. Nedostatky jsou však kompenzovány jednoduchostí designu a škálovatelností.

Databáze tohoto typu jsou často vysoce optimalizovaná uložiště typu klíč-hodnota. Díky možnosti realizovat ukládání dat odlišnými strukturami, jako je například struktura stromová či grafová, je i algoritmická složitost pro operace různá. Obecně platí, že se aplikování daného typu databáze liší dle řešeného problému.

NoSQL se hodí pro ukládání velkého objemu dat, kde není třeba uchování vzájemných vztahů. I proto v současnosti tento segment databází roste a je využíván v oblasti Big Data a real-time webu.

1.4.2 Výběr technologií pro popis dat

Po prostudování základních vlastností uvedených způsobů popisu dat a práci s nimi, zhodnocení jejich kladů a negativ, byla pro uložení a manipulaci s daty vybrána notace JSON.

Důvodem výběru byl také fakt, že značně vyhovuje požadavkům zpracovávané databáze rozhleden. S daty se dá pomocí jazyka JSON snadno manipulovat a jsou lehce čitelná. Data se navíc pomocí externích knihoven dají mapovat na objekty, a tak k nim lze bez problémů přistupovat a pracovat s nimi ve vyšších programovacích jazycích. Na straně serveru lze pomocí vhodných prostředků data modifikovat, spojovat související vazby na základě klíčů, což je v návrhu databázové vrstvy vyvíjené aplikace žádoucí.

Podobné vlastnosti má také jazyk XML, ale na základě osobní empatie a předešlých zkušeností, kdy práce s tímto formátem nebyla tak intuitivní, byl vybrán právě JSON. Je zřejmé, že stejného výsledku by se dalo docílit také použitím relační databáze, ale pro potřeby rozsahu databáze práce je vybrané řešení vhodnou volbou.

1.4.3 Výběr platformy serveru

Pro realizaci serverové části aplikace byly uvažovány dvě alternativy. První z nich byl webový server PHP, druhou webová aplikace, psaná v Javě.

Aplikaci v PHP tvoří sada skriptů. Její nasazení na server je realizováno uložením skriptů do odpovídajícího adresáře. Aplikace ožívá ve chvíli, kdy server od klienta obdrží HTTP požadavek. PHP, volané webovým serverem, interpretuje příslušný skript. Po dokončení požadavku HTTP a následného odeslání odpovědi zpět klientu je běh celého skriptu ukončen. Tím je dokončeno také vykonávání celé aplikace, která kromě místa na disku nezabírá žádné další zdroje. Z toho plyne, že aplikace je načítána s každým novým HTTP požadavkem znovu.

Webová aplikace v Javě je sestavena ze sady JSP, kompilovaných tříd jazyka Java a eventuálně dalších knihoven. Aplikace je na server nasazena obvykle pomocí .war souboru, který se chová podobně, jako klasický archiv. Při nasazení dojde k rozbalení tohoto archivu a následné kompilaci JSP. Z nich jsou nejprve vygenerovány třídy jazyka Java, které jsou poté zkompilovány na bajtkód. Aplikace je inicializována a běží po celou dobu, dokud běží server nebo z něj aplikace není odstraněna či deaktivována.

V současných verzích jsou již oba jazyky jsou objektově orientované. Další společnou vlastností je otevřenost a volná dostupnost i pro komerční účely. Pravděpodobně největším rozdílem je, že Java je staticky typovaný jazyk, kdežto PHP dynamicky. To v praxi znamená, že v případě staticky typovaných jazyků je datový typ proměnné

uveden už při její deklaraci. U dynamicky typovaných jazyků probíhá kontrola datového typu až při samotném běhu aplikace. Díky tomu lze psát pružnější, avšak často méně přehledný kód. Jedna proměnná může tedy nabývat hodnot různých datových typů. Minusem je, že potenciální chyby jsou u dynamicky typovaných jazyků odhaleny až v provozu, kdežto v případě statického typování jsou zřejmé již v době kompilace.

Po srovnání základních vlastností obou alternativ, stanovení požadavků serverové části aplikace a zhodnocení složitosti její realizace pomocí obou alternativ, byl pro vývoj vybrán webový server PHP.

1.5 Správa kódu

Pro správu kódu byl v práci použit verzovací systém Git. Verzování z je pohledu informatika uchovávání historie veškerých změn, provedených v datech a zdrojových kódech vyvíjené aplikace. Systém správy změny eviduje, včetně informací, jako kdo, kdy a jak byly které řádky zdrojového kódu programu pozměněny. Díky tomu je možné zobrazit přesný stav evidovaných souborů kdykoliv z minulosti, odhalit příčinu případného nežádoucího chování a vrátit se ke starší verzi, pokud je třeba.

Významným prvkem verzování je možnost spolupráce více programátorů na jednom projektu. Verzovací aplikace totiž hlídají a umožňují řešit kolize v kódu. V současné době si lze jen těžko představit projekt většího rozsahu, který není verzovaný. Systémy správy verzí mohou být centralizované nebo distribuované. V případě centralizovaných systémů jsou data uložena na jediném serveru a většina akcí vyžaduje komunikaci s ním. V současné době se však často používají systémy distribuované. To znamená, že každý vývojář může mít kopii celé historie uloženou lokálně na svém stroji. To mu umožňuje rychlejší práci. Verzovací systémy ve většině případů neuchovávají úplný stav každé revize, ale pouze rozdíly mezi nimi. Tím se šetří prostor, avšak za cenu pomalejšího přístupu. Mezi nejznámější distribuované systémy správy verzí patří právě Git.

Jednou z webových služeb, která Git podporuje a která byla vybrána jako hostitelská služba vyvíjené aplikace, je BitBucket. Služba nabízí jak zdarma dostupné, tak i komerční řešení. Volně dostupné řešení uživateli umožňuje založit neomezeně soukromých repozitářů, až pro osm osob. BitBucket vznikl v roce 2010 a je napsán v Pythonu za použití Django web frameworku.

1.6 Výběr vývojových prostředí

Pro vývoj serverové části aplikace bylo vybráno vývojové prostředí PhpStorm od společnosti JetBrains. Jedná se o řešení komerční, avšak pro studenty existuje licence, nabízená zdarma.

Prostředí zahrnuje editor pro PHP, HTML a JavaScript s implementovanou analýzou kódu za chodu, analýzou chyb a automatizovaným refaktoringem kódu. Uživatel může IDE rozšířit o další pluginy, a kontrolovat tak potřebu dodatečné funkcionality. Prostředí nabízí editor kódu se zvýrazňováním syntaxe a kompletací kódu. Dále

je implementována podpora PHPDoc, umožňující snadnou dokumentaci psaného kódu. V prostředí je také obsažena podpora SQL a databází s aktualizací databázového schématu za chodu, nasazení aplikace na vzdálený server pomocí SFTP a FTPS s automatickou synchronizací. Výhodou je také podpora ladících nástrojů pro PHP, jako je Xdebug či Zend Debugger. Samozřejmostí je integrovaná komunikace se systémy pro správu verzí.

Pro vývoj klientské části aplikace bylo vybráno Android Studio. Toto IDE je v současné době nejpokročilejším prostředkem pro vývoj aplikací pro mobilní platformu Android. Aplikace byla také vytvořena firmou JetBrains, avšak je dostupná zdarma jako samostatná aplikace, Komerční řešení lze využít v případě integrace do IDE IntelliJ IDEA. Toto vývojové prostředí obsahuje nástroj Gradle, který slouží pro automatizaci sestavování programu. Dále obsahuje rychlý emulátor s možností simulovat funkci nejrůznějších čidel v telefonu. Pomocí Android Studia lze vyvíjet aplikace na všechna zařízení, fungující pod operačním systémem Android (například chytré hodinky, TV nebo Google Glass). Další užitečnou vlastností je možnost aktualizace aplikace bez nutnosti sestavení nového APK. Aplikace má stejně tak, jako PhpStorm, integrovanou komunikaci s VCS.

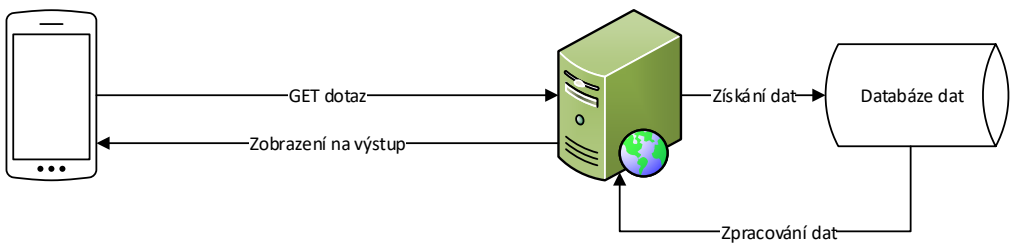
2 Serverová část

Tato část diplomové práce se týká implementace skriptů, které mají komunikovat s uživatelem na základě vznesených dotazů na data. Bude popsán celkový pohled na komunikaci, struktura jednotlivých datových souborů, včetně jejich definice, neboli schématu. Dále bude vysvětleno, jak funguje naprogramované aplikační rozhraní na serverové straně, postup při získání relevantních dat dle typu dotazu a serializace dat do formátu JSON. V závěrečném bodě kapitoly bude vysvětlen způsob zobrazení dat na výstupu.

2.1 Účel serverové části

Serverová část, která je napsána v jazyce PHP, slouží klientovi – uživateli mobilního zařízení – jako prostředek pro získání dat. Daty jsou myšleny veškeré informace o rozhlednách a v případě, že je uživatel lokalizován na některé z nich, tak také data, obsahující informace o okolním panoramatu.

Komunikace mezi klientem a serverem probíhá prostřednictvím HTTP dotazu. Ten je na server vznesen metodou GET, je tedy přenášen pomocí URI. Server tento dotaz převezme, na základě jeho parametrů rozhodne o typu zpracování a provede definovanou akci. Získaná data jsou v odpovídajícím formátu zobrazena na výstup. To umožňuje knihovnám v mobilním zařízení další manipulaci s nimi. Postup získání dat je graficky znázorněn na obrázku 2.1.



Obrázek 2.1: Komunikace klient-server

Klient je aktivním prvkem této komunikace. To znamená, že on je tím, kdo posílá serveru žádosti a následně čeká na odpověď. Oproti tomu server je pasivní člen, který

naslouchá na síti a čeká na žádosti klientů, které po přijetí korektně formulovaného požadavku obslouží.

2.2 Struktura datových souborů

Jako databázový zdroj informací slouží soubory typu JSON. Na serveru jsou nahrány tři druhy databázových souborů. Každý soubor reflektuje jednu databázovou entitu na serveru. První soubor obsahuje informace o rozhlednách (*lots.json*).

V druhém souboru (*view_points.json*) jsou uložena data o každém evidovaném bodu, který lze z databáze rozhleden spatřit. Je zřejmé, že z jedné rozhledny je možné vidět více bodů a zároveň jeden bod může být spatřen z více rozhleden. Pro udržení konzistentní databáze, a tedy i korektní simulaci této reality (vztahu $m : n$), je nutné vytvořit soubor třetí (*view.json*).

V něm jsou v datové struktuře zaznamenány body, vztahující se ke každé evidované rozhledně. Vztahy mezi entitami jsou znázorněny na ER diagramu 2.2.

Každý z těchto souborů je popsán vlastním JSON schématem, který může sloužit jako validační předpis při modifikaci. Schémata souborů jsou také uložena na serveru. V následujících bodech je každé ze nich rozebráno v podobě tabulky s vysvětlením.

Obrázek 2.2: Vztahy mezi entitami

2.2.1 Popis souboru s přehledem rozhleden

Jak již bylo zmíněno, v souboru jsou zaznamenány informace o evidovaných rozhlednách. Každý z uložených atributů má svou informační hodnotu. Jejich význam je uveden v tabulce 2.1.

Z tabulky lze vidět, že každá rozhledna je identifikována pomocí čísla, které je primárním klíčem. Toto unikátní číslo je dáno pořadím rozhledny v souboru. Vlastnosti, jejichž datovým typem je celé číslo a rozsahy jsou shora i zdola omezené, slouží jako číselníky. Transformace na hodnoty uživateli srozumitelné je provedeno na straně klienta. Výhoda číselníků pomáhá zjednodušit strukturu kódu a může být uplatněna také při vytváření jazykových mutací aplikací.

Tabulka obsahuje dva atributy, které jsou datového typu pole. Atribut popisu rozhledny obsahuje další dva podatributy. První z nich zmiňuje historii a fakta o

ATRIBUT	TYP	VÝZNAM	ROZSAH
id (PK)	celé číslo	unikátní identifikátor rozhledny	$\langle 1, \infty)$
altitude	celé číslo	nadmořská výška	$\langle 0, \infty)$
city	řetězec	město, kde rozhledna leží	-
country	řetězec	kód země, kde rozhledna leží	-
county	řetězec	samosprávný celek	-
created	řetězec	časové razítko vytvoření	-
description	pole	popis rozhledny	-
gallery	pole	galerie rozhledny	-
gradient	celé číslo	míra stoupání k rozhledně	$\langle 1, 3\rangle$
latitude	číslo	souřadnice zeměpisné šířky	$(-\infty, \infty)$
longitude	číslo	souřadnice zeměpisné délky	$(-\infty, \infty)$
main_image	řetězec	umístění hlavního obrázku	-
material	celé číslo	typ rozhledny	$\langle 1, 4\rangle$
name	řetězec	název rozhledny	-
view	celé číslo	čistota výhledu z rozhledny	$\langle 1, 5\rangle$
websites	řetězec	webové stránky rozhledny	-

Tabulka 2.1: Legenda přehledu rozhleden

rozhledně. Druhý popisuje cestu, jak se k ní lze dostat. Výhodou této konstrukce je větší přehlednost v datech. Lze ji také dobře využít při přidávání nových podatributů této kategorie.

Pole, které obsahuje informace o galerii týkající se rozhledny, je vnitřně rozděleno na objekty. Každý z nich má tři vlastnosti – *large*, *small*, *title*. První dvě zmíněné obsahují adresu cesty k obrazovému zdroji, každý pouze v jiné velikosti. Poslední atribut je textový popisek, který prvek galerie vystihuje.

Na závěr je třeba dodat, že všechny uvedené atributy jsou povinné, takže při vynechání některého z nich soubor neprojde validační procedurou.

2.2.2 Popis souboru s přehledem bodů panoramatu

Tento soubor obsahuje informace o bodech výhledu. Význam jednotlivých atributů je popsán tabulkou 2.2.

Klíčem pro přístup k datovému záznamu je opět unikátní identifikátor, jehož hodnota je dána pořadím bodu v souboru. Je jasné, že bod výhledu může být zároveň i rozhlednou, a proto je dostupný atribut, který eviduje tento příznak. S její pomocí lze poté na klientské straně aplikace implementovat například přepínání do detailu rozhledny přímo z panoramatu.

Parametr určující typ místa je číselník, nabývající hodnot celých čísel v rozsahu 1 až 7. Typem místa je myšlena jeho geografická povaha (město, vesnice, řeka, hora, atd.). Parametr napomáhá uživateli k snazšímu přiřazení popisku k místu. Konkrétní přiřazení hodnot bude vysvětleno při popisu klienta.

ATRIBUT	TYP	VÝZNAM	ROZSAH
id (PK)	celé číslo	unikátní identifikátor bodu	$\langle 1, \infty)$
altitude	celé číslo	nadmořská výška bodu	$\langle 0, \infty)$
country	řetězec	kód země, kde místo leží	-
created	řetězec	časové razítko vytvoření	-
is_lot	boolean	zda je místo i rozhlednou	$\{true, false\}$
latitude	číslo	souřadnice zeměpisné šířky	$(-\infty, \infty)$
longitude	číslo	souřadnice zeměpisné délky	$(-\infty, \infty)$
name	řetězec	název místa	-
type	celé číslo	typ místa	$\langle 1, 7\rangle$

Tabulka 2.2: Legenda přehledu bodů panoramatu

ATRIBUT	TYP	VÝZNAM	ROZSAH
id (FK)	celé číslo	unikátní identifikátor rozhledny	$\langle 1, \infty)$
view_points (FK)	pole	pole identifikátorů míst	-

Tabulka 2.3: Legenda přehledu panoramatu

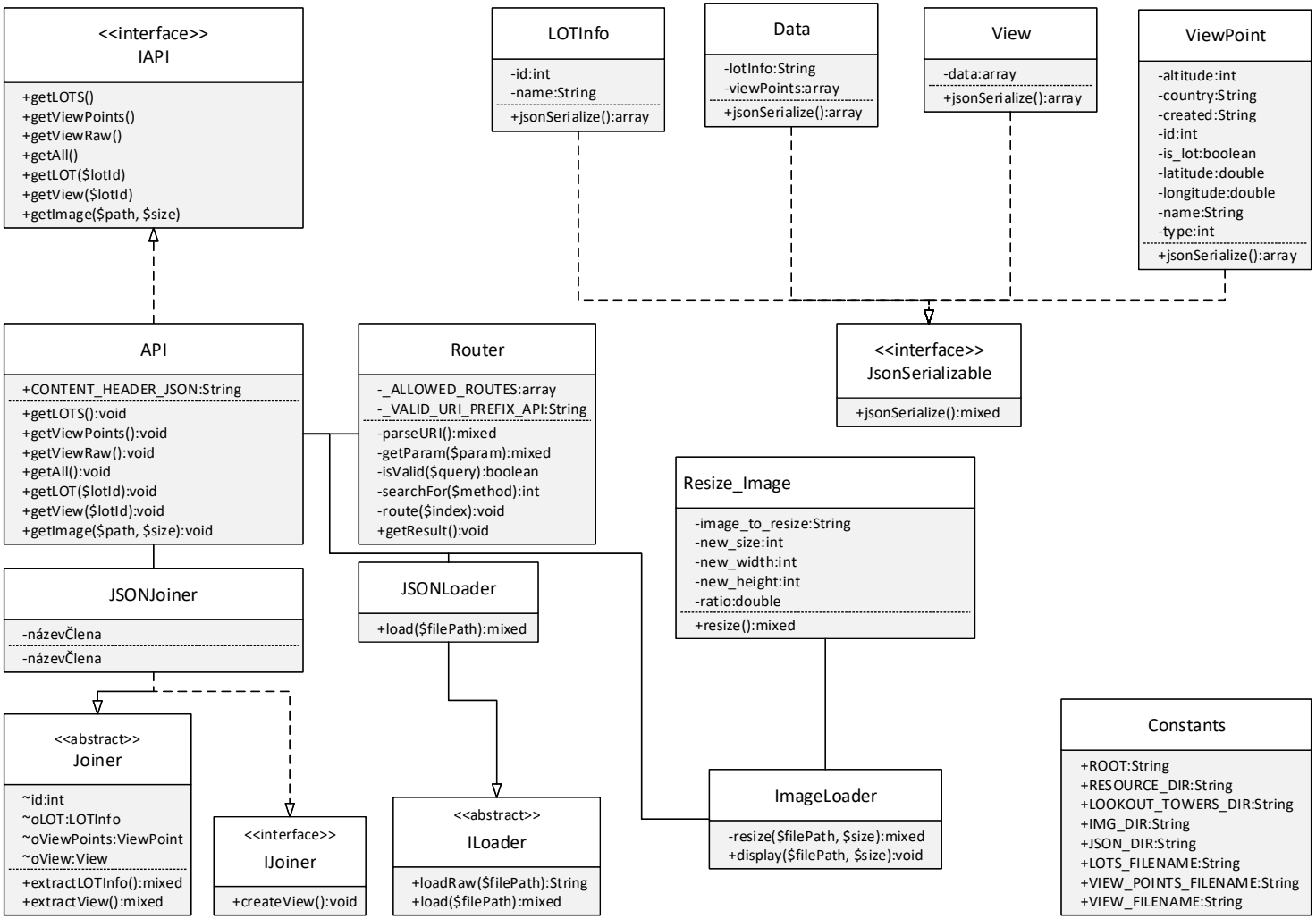
2.2.3 Popis souboru s panoramatem jednotlivých rozhleden

Struktura souboru, popisující panorama každé rozhledny, je oproti předchozím velmi jednoduchá. Tento popis je znázorněn tabulkou 2.3.

Ve struktuře figurují klíče z výše uvedených souborů jako cizí klíče. Identifikátor rozhledny je spárován s polem bodů výhledu, které jsou z místa viditelné. Pole, ve kterém je tedy uložen seznam klíčů, nemá nijak omezenou velikost. Díky tomuto systému lze vytvořit databázi rozhledu bez nutnosti uchovávat redundantní data. Způsob také zajišťuje, že při aktualizaci databáze není třeba provádět změny na více místech. Tím jsou také sníženy její režijní požadavky.

2.3 Popis aplikačního rozhraní serverové strany

Rozhraní *IAPI* obsahuje definice funkcí pro získání dat z databázových či obrazových souborů. Těla těchto funkcí jsou následně implementována ve třídě *API*. Díky tomuto návrhu programátor ví, jak aplikace na webovém serveru komunikuje navenek, při zachování skryté vnitřní implementace. Diagram tříd serverové části, spolu se seznamem názvů funkcí rozhraní *IAPI* je znázorněn na 2.3.



Obrázek 2.3: Diagram tříd serverové části

První funkce s názvem *getLOTS()* vrací data o všech evidovaných rozhlednách ve formátu JSON. Druhá funkce *getViewPoints()* má za úkol vracet veškerá data, uložená v souboru s body výhledu. Účelem funkce *getViewRaw()* je opět vrácení dat, tentokrát obsahujících informace o panoramatech rozhleden. Je důležité podotknout, že tyto tři zmíněné funkce vracejí data surová. To znamená, že pouze načtou soubor, který následně zobrazí na výstup.

Další pomyslnou kategorií funkcí jsou funkce, které se soubory již pracují – spojují je nebo z nich extrahují určitě informace. Jednou z těchto funkcí je funkce *getAll()*. Ta má za úkol spojit na základě cizích klíčů veškeré dostupné informace ze všech souborů do jednoho JSONu, který je následně zobrazen na výstupu. Funkce *getLOT($lotId)* je první, která přebírá parametr. Tímto parametrem je číslo, identifikující rozhlednu. Na jeho základě je rozhodnuto o rozhledně, jejíž informace budou zobrazeny na výstupu. Předposlední funkcí API je *getView($lotId)*. Ta zajišťuje vrácení korektních informací o okolním panoramatu rozhledny, na základě jejího identifikačního čísla.

Účelem poslední dostupné funkce *getImage($path, $size)* je zobrazit na výstup obrazová data. Jak lze z definice vidět, funkce přebírá dva parametry. Prvním je řetězec, specifikující cestu k obrázku, druhým je číselný údaj, určující hodnotu většího z rozměrů.

2.4 Serializace dat do formátu JSON

Kromě obrazových dat jsou všechna ostatní serializována do formátu JSON. Serializace je uskutečněna pomocí funkce *json_encode*. Funkce je nativní funkcí v PHP [13] a má následující definici:

```
1 string json_encode(mixed $value [, int $options=0 [, int $depth
    =512]])
```

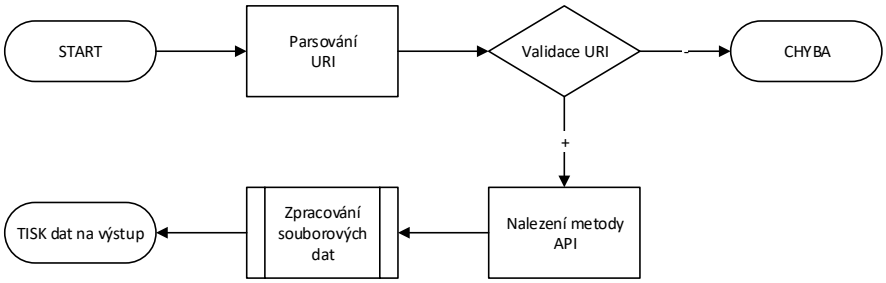
Funkce vrací řetězec ve JSON v případě, že serializace proběhla bez chyby. Řetězec je vytvořen z proměnné *$value*. Tato proměnná, která je tedy kódována na JSON, může být jakéhokoliv datového typu, kromě speciálního, který drží odkaz na externí zdroj, zvaný *resource*. Všechna data v řetězci musí být kódována ve formátu UTF-8. V případě, že při kódování nastala chyba, vrací funkce nepravdu (*FALSE*).

Parametrem *$options* lze předat konstanty, zajišťující čisté kódování výstupního proudu. Dobrým příkladem je konstanta *JSON_UNESCAPED_SLASHES*. Ta zajistí, že všechna dopředná lomítka ve výstupním JSONu nebudou escapována. Parametr je nepovinný.

Posledním parametrem, opět nepovinným, je *$depth*. Standardní hodnota této proměnné je 512. Číslo určuje maximální možnou hloubku vnoření dat v JSONu. Hodnota parametru musí být kladná hodnota.

2.5 Postup při získání relevantních dat

Při získávání dat je aplikován postup, vyjádřený vývojovým diagramem 2.4. V následujících bodech budou jednotlivé kroky konkretizovány a dovysvětleny.



Obrázek 2.4: Postup při zpracování HTTP dotazu

2.5.1 Parsování URI a rozpoznání metody zpracování příkazu

Volání všech HTTP dotazů na tuto webovou aplikaci je odchyceno kořenovým souborem *index.html*. To je zajištěno pomocí vhodné podoby konfiguračního souboru *.htaccess*. Obsah tohoto souboru je následující:

```
1 RewriteEngine On
2 RewriteCond %{REQUEST_FILENAME} !-f
3 RewriteCond %{REQUEST_FILENAME} !-d
4 RewriteRule \^([\^/]*)(.*)$index.php?first=$1\&second=$2\&third=$3 [
    QSA]
```

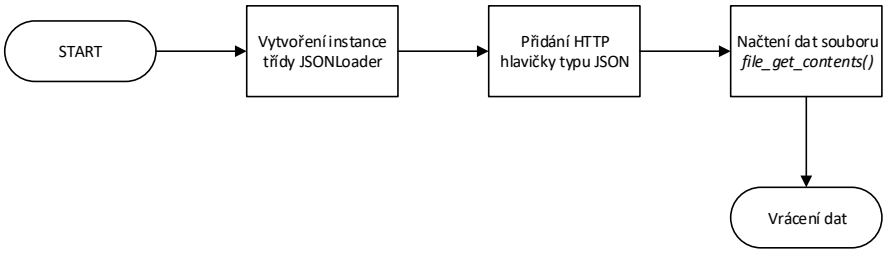
První řádek kódu zapíná přepisovací mód. Definováním dalších pravidel poté lze kontrolovat, jaký skript bude požadavek obsluhovat. Další dva řádky určují, že se nemají odchytávat dotazy na soubor či konkrétní složku. Poslední příkaz definuje, za jakých podmínek bude dotaz předán skriptu pro zpracování. Příznak QSA zařídí, že se do nové, přesměrované adresy, přenese také část adresy s GET parametry.

Po odchycení požadavku je vytvořena instance třídy *Router*. Následně je zavolána funkce této třídy *getResult()*. Ta v prvním kroku parsuje URI za pomoci PHP funkce *explode()*. Funkce vrací řetězec, který identifikuje metodu rozhraní API, kterou uživatel volá.

Dále je volána funkce, která rozpozná metodu pro zpracování požadavku. Ta na základě porovnání řetězců v poli validních metod (*_ALLOWED_ROUTES*) určí, zda volá klient korektní funkci API. V případě kladného výsledku je vrácen číselný index z pole, identifikující danou funkci. Tím lze přistoupit k samotnému vykonání implementované funkce z aplikačního rozhraní. V opačném případě je vrácen výraz *false*.

2.5.2 Získání souboru se surovými daty

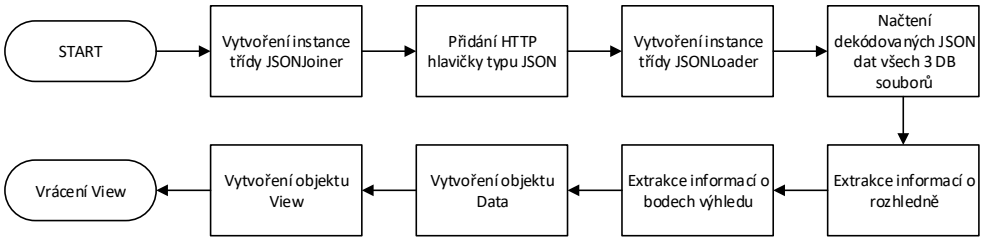
Prvním typem dat, který je zobrazován na výstupu, jsou data nezpracovaná, pouze načtená ze souboru. Ta jsou načítána pomocí funkce *loadRaw()*, která je implementována v abstraktní třídě *ILoader*. Načtení je realizováno PHP funkcí *file_get_contents($filePath)*, která jako parametr přebírá cestu k souboru. Před výpisem dat je prostřednictvím příkazu *header(self::CONTENT_HEADER_JSON)* přidána hlavička typu JSON a vše je následně vykresleno na výstup příkazem *echo*. Tento postup je také vyobrazen na diagramu 2.5.



Obrázek 2.5: Získání surových dat

2.5.3 Získání dat pomocí extrakce a spojení souborů

Dalším využitým způsobem, jak data získat, je jejich extrakce ze souborů a následné spojení. Takto jsou získána například data okolního výhledu, na nichž bude tento princip popsán. Celý proces je znázorněn na diagramu 2.6.



Obrázek 2.6: Získání dat výhledu z rozhledny

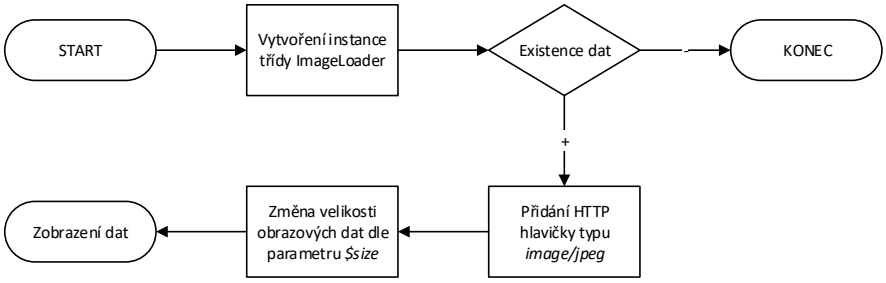
Z pohledu uživatele je nejdříve za pomoci superglobální proměnné *$_GET* získán z URI parametr, identifikující rozhlednu. Poté jsou načteny všechny tři databázové soubory, jejichž řetězec s umístěním je uložen ve třídě *Constants*. Načtení je realizováno mimo jiné i pomocí PHP funkce *json_decode()*, která daný soubor převede na objekt, a tak lze k datům snadno přistoupit pomocí objektové notace.

Dalším krokem je extrakce dat o rozhledně. Toho je docíleno za pomoci procházení příslušného souboru do doby, než je nalezen shodný identifikátor rozhledny. Následně je vytvořen objekt *LOTInfo*, do něhož jsou data extrahována. Extrakce dat výhledu probíhá obdobným způsobem, pouze s vytvořením jiného, odpovídajícího objektu. Objekty jsou následně uloženy do pole, kde jsou do sebe vhodně vnořeny.

Na závěr je volána PHP metoda *json_encode()*, které je předáno pole vytvořených objektů. Díky tomu, že tyto objekty implementují rozhraní *JSONSerializable*, a tím i metodu *jsonSerialize()*, je zajištěn korektní převod objektu zpět do formátu JSON. Samotný výpis dat probíhá stejným způsobem, jako u předchozího případu.

2.5.4 Získání obrázku

Posledním typem dat, který lze pomocí API získat, je obrázek. Celý proces je znázorněn vývojovým diagramem 2.7.



Obrázek 2.7: Postup při zobrazení obrazových dat

Nejdříve je zkontrolována existence souboru. Pokud je na serveru dostupný, je nastavena HTTP hlavička pro obrazová data. Poté je volána funkce na změnu velikosti obrázku, na základě hodnoty parametru v URI. Nebyla-li velikost uvedena, je použita výcozí hodnota 500 px.

Nezáleží, zda je poloha obrázku na šířku či na výšku. Funkce zajistí, že velikost bude dána hodnotou větší z obou souřadnic, to vše při zachování poměru stran. Pro změnu velikosti je volána funkce, která je součástí standardní GD knihovny. Ta obsahuje funkce pro generování obrazových dat a práci s nimi. Knihovna samotná je napsaná v jazyce C, ale lze jí bez problémů používat i v jiných jazycích.

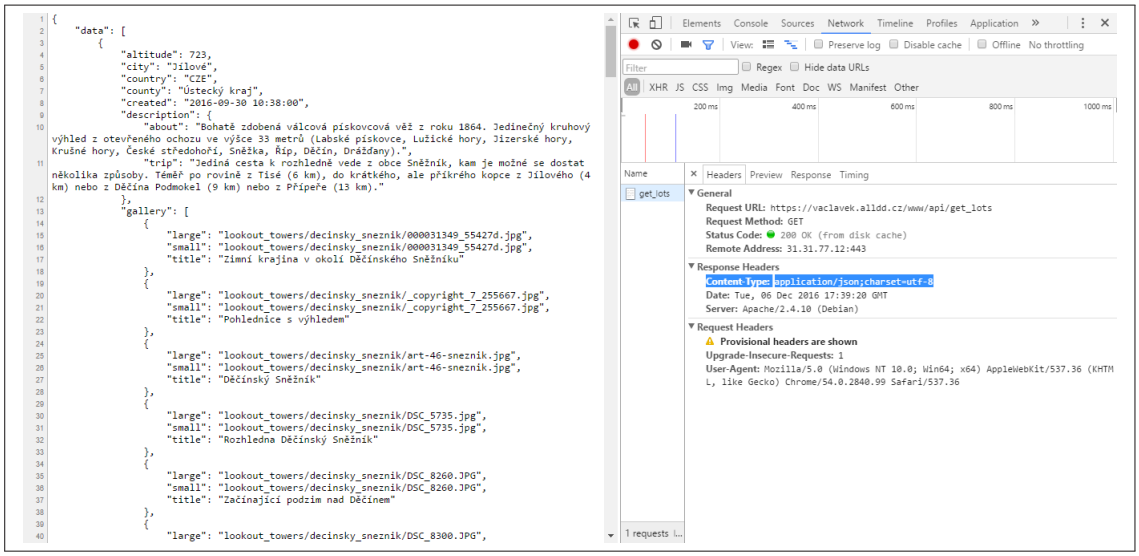
Následně je volána funkce *imagejpeg()*, která je také součástí zmíněné knihovny a jejím účelem je data vykreslit. Funkci jsou v parametru předána obrazová data se změněnou velikostí.

2.6 Datový výstup

Pro vizuální interpretaci výsledků získání dat jsou zde uvedeny dva příklady. Prvním příkladem je získání dat o rozhlednách. Toho je docíleno HTTP dotazem

```
1 https://vaclavek.alldd.cz/www/api/get_lots
```

Vzhled okna prohlížeče lze vidět na obrázku 2.8. Jak lze z hlavičky dokumentu vidět, data jsou opravdu ve formátu JSON.

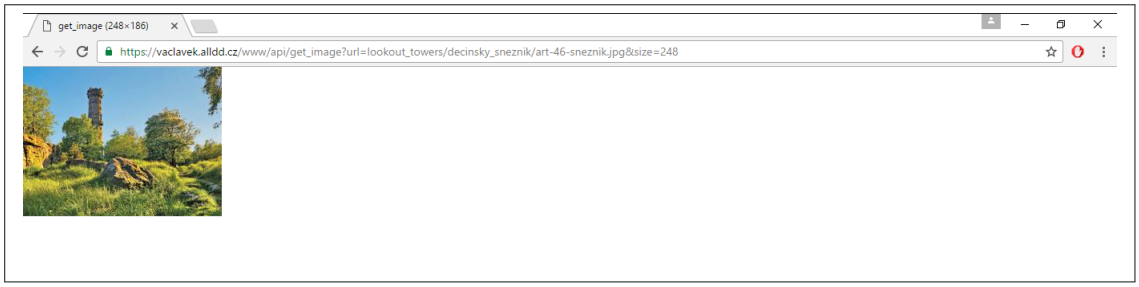


Obrázek 2.8: Výstupní proud dat rozhleden

Druhým příkladem je volání obrazových dat, a to dotazem

```
1 https://vaclavek.alldd.cz/www/api/get_image?url=lookout_towers/
    decinsky_sneznik/art-46-sneznik.jpg&size=248
```

Jak lze vidět, parametr *url* obsahuje cestu k obrázku a *size* obsahuje jeho velikost, kterou chce uživatel vrátit. Výstup je zobrazen na obrázku 2.9.



Obrázek 2.9: Výstupní proud dat obrázku

3 Klientská část (backend)

Další stěžejní kapitolou práce je návrh klienta – v tomto případě mobilní aplikace. Tato část se týká implementace backendové části aplikace. Jsou zde tedy popsány postupy zpracování dat, jejichž zobrazení bude vysvětleno v následující kapitole.

Kapitola je zaměřena na implementaci algoritmu pro výpočet odchylky mezi magnetickým a geografickým severním pólem Země. Kromě tohoto algoritmu je zde také realizován postup pro stanovení magnetického severu Země. Jsou také popsány pomocné knihovny, které těmto algoritmům sekundují.

V dalších krocích je popsána komunikace mezi mobilní aplikací a serverovou částí. Textová i obrazová data, která jsou stažena, je třeba v zařízení uchovat, což je jedním z dalších bodů kapitoly. Dalším z témat je analogicky také přístup k uloženým datům, aby mohla být ve frontendové části zobrazena.

3.1 Implementace algoritmu pro výpočet magnetické deklinace

3.1.1 Matematický pohled na algoritmus

Jak už bylo několikrát zmíněno, tento algoritmus je jedním ze stěžejních bodů práce. Ačkoliv je zřejmé, že záleží i na spoustě dalších faktorů, jako je například přesnost kompasu, tak právě algoritmus pro výpočet magnetické deklinace by měl zajistit rapidní zpřesnění orientace, a tím uživateli mobilní aplikace zpřístupnit korektní zobrazení panoramatu.

Tato podkapitola popisuje matematickou reprezentaci WMM a seznam výpočtů, potřebných k vyjádření složek magnetického pole pro zadanou polohu a čas, to vše za pomoci koeficientů WMM. Všechny proměnné, uvedené v této podkapitole, se řídí následujícími pravidly: úhly jsou uvedeny v radiánech, vzdálenosti v metrech, magnetická intenzita v jednotkách nano-Tesla (nT) a čas je vyjádřen v letech.

Hlavní magnetické pole B_m je pole napětí, a proto může být zapsáno v geocentrických sférických souřadnicích (zeměpisná délka λ, zeměpisná šířka ϕ', poloměr r) jako záporný prostorový gradient skalárního napětí

$$B_m\left(\lambda, \phi', r, t\right) = -\nabla V\left(\lambda, \phi', r, t\right) \tag{3.1}$$

Toto napětí může být rozvinuto pomocí sférických harmonických:

$$V\left(\lambda,\phi',r,t\right)=a\sum_{n=1}^{N}\left(\frac{a}{r}\right)^{n+1}\sum_{m=0}^{n}\left(g_n^m\left(t\right)\cos\left(m\lambda\right)+h_n^m\left(t\right)\sin\left(m\lambda\right)\right)\breve{P}_n^m\left(\sin\phi'\right) \tag{3.2}$$

kde N=12 je řád rozvoje WMM, a (6371200 m) je hodnota geomagnetického poloměru, (λ, ϕ', r) jsou zeměpisná délka, zeměpisná šířka, poloměr ve sférickém geocentrickém souřadném systému a $g_n^m(t)$ a $h_n^m(t)$ jsou časově závislé Gaussovy koeficienty stupně n a řádu m, popisující zemské magnetické pole. Pro každé reálné číslo μ, jsou $\breve{P}(\mu)$ Schmidtovy semi-normalizované asociované Legendreovy funkce definovány jako:

$$\breve{P}_n^m(\mu)=\begin{cases}\sqrt{2\frac{(n-m)!}{(n+m)!}}P_{n,m}(\mu) & m>0\\ P_{n,m}(\mu) & m=0\end{cases} \tag{3.3}$$

Zde uvedená definice $P_{n,m}(\mu)$ je běžně využívána v oboru geodézie a geomagnetismu. WMM2015 zahrnuje dvě sady Gaussových koeficientů. První z nich poskytuje model pole pro rok 2015.0 v jednotkách nT, druhá poskytuje prediktivní model sekulární variace pro rozmezí mezi roky 2015.0 až 2020.0 včetně, v jednotkách $\frac{nT}{rok}$. Model sekulární variace byl odvozen od geomagnetických dat, platných k roku 2015.0. Přesněji řečeno, model reprezentuje průměr z okamžitých změn hlavního pole v intervalech 0.1 roku, počínaje rokem 2013.6. Postup pro výpočet změn pole je aplikován dle nejlepších znalostí, které byly do vydání WMM shromážděny. I přesto je však předpokládáno, že hodnoty magnetického pole budou po dobu životnosti modelu podléhat nepřesnostem.

Níže je uveden chronologický postup pro výpočet prvků magnetického pole pro zadanou polohu a čas (λ, ϕ, h_{MSL}, t), kde λ a ϕ jsou geodetická délka a šířka, h_{MSL} je nadmořská výška a t je čas v desetinných jednotkách let.

V prvním kroku uživatel poskytne čas, polohu a nadmořskou výšku, pro kterou mají být magnetické složky spočítány. Geodetické souřadnice (λ, ϕ, h) jsou převedeny do geocentrických souřadnic (λ, ϕ', r). Lze si všimnout, že λ je v obou souřadných systémech stejná a (ϕ', r) jsou spočítány dle následujících vztahů:

$$\begin{aligned}p&=\left(R_c+h\right)\cos\phi\\ z&=\left(R_c\left(1-e^2\right)+h\right)\sin\phi\\ r&=\sqrt{p^2+z^2}\\ \phi'&=\arcsin\tfrac{z}{r}\end{aligned} \tag{3.4}$$

V případě $p=\sqrt{x^2+y^2}$ jsou x, y a z souřadnice geocentrického kartézského souřadného systému, v němž kladné hodnoty os x a z míří ve směru nultého poledníku, respektive osy rotace Země. Dále je pro elipsoidu WGS 84 definována délka hlavní poloosy A, převrácená hodnota zploštění $\frac{a}{f}$, kvadrát excentricity e^2 a poloměr

zakřivení hlavní vertikály R_c, pro danou zeměpisnou šířku ϕ jako:

$$\begin{aligned} A &= 6378137\ m \\ \tfrac{1}{f} &= 298,257223563 \\ e^2 &= f(2-f) \\ R_c &= \tfrac{A}{\sqrt{1-e^2\sin^2\phi}} \end{aligned} \tag{3.5}$$

V druhém kroku jsou stanoveny Gaussovy koeficienty $g_n^m(t)$ a $h_n^m(t)$ pro specifikovaný čas t, z koeficientů modelu $g_n^m(t_0)$, $h_n^m(t_0)$, $\dot{g}_n^m(t_0)$ a $\dot{h}_n^m(t_0)$ jako

$$\begin{aligned} g_n^m(t) &= g_n^m(t_0) + (t-t_0)\dot{g}_n^m(t_0) \\ h_n^m(t) &= h_n^m(t_0) + (t-t_0)\dot{h}_n^m(t_0) \end{aligned} \tag{3.6}$$

kde čas je zadán v desetinách let a $t_0 = 2015.0$ (referenční čas epochy modelu). Hodnoty $g_n^m(t)$ a $h_n^m(t)$ jsou nazvány koeficienty hlavního pole a hodnoty $\dot{g}_n^m(t_0)$ a $\dot{h}_n^m(t_0)$ jsou koeficienty sekulární variace.

Jako třetí krok jsou vypočítány vektorové komponenty X', Y' a Z' v geocentrických souřadnicích jako:

$$\begin{aligned} &X'(\lambda,\phi',r) = -\tfrac{1}{r}\tfrac{\partial V}{\partial \phi'} \\ &= -\textstyle\sum_{n=1}^{12}\left(\tfrac{a}{r}\right)^{n+2}\sum_{m=0}^{n}\left(g_n^m(t)\cos(m\lambda) + h_n^m(t)\sin(m\lambda)\right)\tfrac{d\breve{P}_n^m(\sin\phi')}{d\phi'} \end{aligned} \tag{3.7}$$

$$\begin{aligned} &Y'(\lambda,\phi',r) = -\tfrac{1}{r\cos\phi'}\tfrac{\partial V}{\partial \lambda} \\ &= \tfrac{1}{\cos\phi'}\textstyle\sum_{m=0}^{12}\left(\tfrac{a}{r}\right)^{n+2}\sum_{m=0}^{n} m(g_n^m(t)\sin(n\lambda) - h_n^m(t)\cos(m\lambda))\breve{P}_n^m(\sin\phi') \end{aligned} \tag{3.8}$$

$$\begin{aligned} &Z'(\lambda,\phi',r) = \tfrac{\partial V}{\partial r} \\ &= (n+1)\textstyle\sum_{n=1}^{12}(n+1)\left(\tfrac{a}{r}\right)^{n+2}\sum_{m=0}^{n}(g_n^m(t)\cos(m\lambda) + h_n^m(t)\sin(m\lambda))\breve{P}_n^m(\sin\phi') \end{aligned} \tag{3.9}$$

V této chvíli lze vypočítat sekulární variaci komponent pole jako:

$$\begin{aligned} &\dot{X}'(\lambda,\phi',r) = -\tfrac{1}{r}\tfrac{\partial \dot{V}}{\partial \phi'} \\ &= -\textstyle\sum_{n=1}^{12}\left(\tfrac{a}{r}\right)^{n+2}\sum_{m=0}^{n}\left(\dot{g}_n^m(t)\cos(m\lambda) + \dot{h}_n^m(t)\sin(m\lambda)\right)\tfrac{d\breve{P}_n^m(\sin\phi')}{d\phi'} \end{aligned} \tag{3.10}$$

$$\begin{aligned} &\dot{Y}'(\lambda,\phi',r) = -\tfrac{1}{r\cos\phi'}\tfrac{\partial \dot{V}}{\partial \lambda} \\ &= \tfrac{1}{\cos\phi'}\textstyle\sum_{m=0}^{12}\left(\tfrac{a}{r}\right)^{n+2}\sum_{m=0}^{n} m(\dot{g}_n^m(t)\sin(n\lambda) - \dot{h}_n^m(t)\cos(m\lambda))\breve{P}_n^m(\sin\phi') \end{aligned} \tag{3.11}$$

$$\begin{aligned} &\dot{Z}'(\lambda,\phi',r) = \tfrac{\partial \dot{V}}{\partial r} \\ &= (n+1)\textstyle\sum_{n=1}^{12}(n+1)\left(\tfrac{a}{r}\right)^{n+2}\sum_{m=0}^{n}(\dot{g}_n^m(t)\cos(m\lambda) + \dot{h}_n^m(t)\sin(m\lambda))\breve{P}_n^m(\sin\phi') \end{aligned} \tag{3.12}$$

$$\frac{d\breve{P}_n^m(\sin\phi')}{d\phi'} = (n+1)(\tan\phi')\breve{P}_n^m(\sin\phi') - \sqrt{(n+1)-m^2}(\sec\phi')\breve{P}_{n+1}^m(\sin\phi') \quad (3.13)$$

Ve čtvrtém kroku jsou vektory geocentrického magnetického pole X', Y' a Z' natočeny do elipsoidního souřadného systému, použitím:

$$\begin{aligned} X &= X'\cos(\phi'-\phi) - Z'\sin(\phi'-\phi) \\ Y &= Y' \\ Z &= X'\sin(\phi'-\phi) + Z'\cos(\phi'-\phi) \end{aligned} \quad (3.14)$$

Obdobně jsou vyjádřeny a otočeny také časové deriváty vektorových komponent $\dot{X}'$, $\dot{Y}'$ a $\dot{Z}'$, za použití:

$$\begin{aligned} \dot{X} &= \dot{X}'\cos(\phi'-\phi) - \dot{Z}'\sin(\phi'-\phi) \\ \dot{Y} &= \dot{Y}' \\ \dot{Z} &= \dot{X}'\sin(\phi'-\phi) + \dot{Z}'\cos(\phi'-\phi) \end{aligned} \quad (3.15)$$

V posledním kroku jsou z ortogonálních komponent vypočítány magnetické složky H, F, I a D:

$$H = \sqrt{X^2+Y^2},\ F = \sqrt{H^2+Z^2},\ I = \text{atan2}(Z,H),\ D = \text{atan2}(Y,X) \quad (3.16)$$

Sekulární variace těchto složek je následně vypočítána jako

$$\begin{aligned} \dot{H} &= \tfrac{X\cdot\dot{X}+Y\cdot\dot{Y}}{H} \\ \dot{F} &= \tfrac{X\cdot\dot{X}+Y\cdot\dot{Y}+Z\cdot\dot{Z}}{F} \\ \dot{I} &= \tfrac{H\cdot\dot{Z}+Z\cdot\dot{H}}{F^2} \\ \dot{D} &= \tfrac{X\cdot\dot{Y}+Y\cdot\dot{X}}{H^2} \\ G\dot{V} &= \dot{D}\dot{Z} = \dot{X}'\sin(\phi'-\phi) + \dot{Z}'\cos(\phi'-\phi) \end{aligned} \quad (3.17)$$

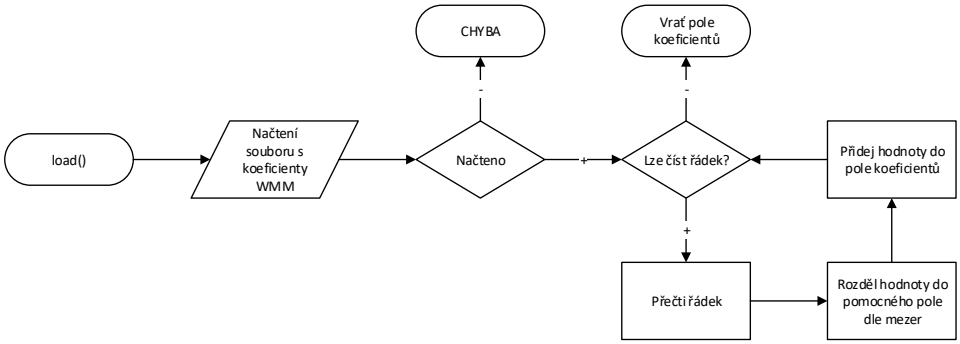
kde $\dot{I}$, $\dot{D}$ a $G\dot{V}$ jsou dány v $\frac{rad}{rok}$ [3].

3.1.2 Načítání koeficientů WMM

Pro načítání koeficientů Magnetického modelu Země slouží třída *WMMCoefficients*. Třída obsahuje hlavní funkci *load()*, která při běhu aplikace zajistí korektní načtení koeficientů. Postup při načítání koeficientů je vysvětlen vývojovým diagramem 3.1.

Soubor s koeficienty WMM je uložen v datech aplikace ve složce *res*. Tato složka je při programování pro Android standardním místem, kam se ukládají externí zdroje, což umožňuje spravovat je nezávisle na kódu. Tyto zdroje jsou ještě interně rozděleny dle kategorií do dalších složek. Soubor s koeficienty spadá do kategorie *raw*, a proto je uložen ve složce se stejnojmenným názvem.

Struktura dat jednoho řádku souboru je viditelná z tabulky 3.1, kde n je stupeň, m řád, $g_n^m(t_0)$ a $h_n^m(t_0)$ jsou koeficienty hlavního pole a $\dot{g}_n^m(t_0)$ a $\dot{h}_n^m(t_0)$ jsou koeficienty sekulární variace.



Obrázek 3.1: Načítání koeficientů WMM

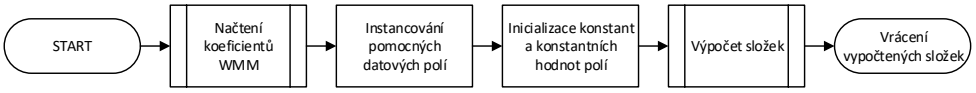
n	m	$g_n^m(t_0)$	$h_n^m(t_0)$	$\dot{g}_n^m(t_0)$	$\dot{h}_n^m(t_0)$

Tabulka 3.1: Struktura řádku souboru koeficientů WMM

3.1.3 Výpočet magnetické deklinace

Tato třída je programovou implementací postupu, popsaného v části 3.1.1. Obsahuje funkce, které jsou potřebné pro výpočet vektorových složek hlavního magnetického pole Země, a tedy i magnetické deklinace.

V následujících řádcích jsou popsány jednotlivé kroky výpočtu složek programově, s odkazy na související matematickou reprezentaci. Objektivní shrnutí celého postupu je vyjádřeno diagramem 3.2.



Obrázek 3.2: Postup výpočtu složek magnetického pole Země

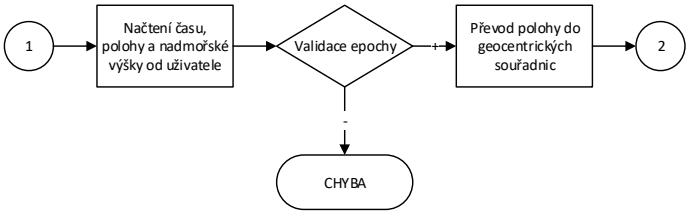
Prvním krokem je zjištění vstupních dat od uživatele, jejich validace a v případě kladného výsledku následuje převedení polohy do geocentrických souřadnic. Postup je pro přehlednost zobrazen také na diagramu 3.3. Data od uživatele jsou předána v parametrech funkce.

```
1 public Double[] getDeclination(double altitudeKm, double
      latitudeDegrees, double longitudeDegrees, double year) {...}
```

Validace epochy obsluhuje jednoduchý kontrolní kód, který v případě neplatného zadání vrací *null*.

```
1 double dt = year - this.EPOCH;
```

```
2  if(dt < 0d || dt > 5d) {
3    return null;
4  }
```

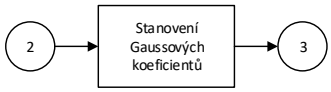


Obrázek 3.3: První krok výpočtu složek magnetického pole Země

Převod polohy do geocentrických souřadnic je realizován následujícími řádky kódu. Ty, za pomoci konstant a proměnných, které byly v kódu vyjádřeny dříve, reprezentují vztah 3.4.

```
1  q = Math.sqrt(this.A2 + this.C2 * srlat2);
2  q1 = alt * q;
3  q2 = Math.pow(((q1+this.A2)/(q1+this.B2)), 2);
4  ct = srlat/Math.sqrt(q2 * crlat2 + srlat2);
5  st = Math.sqrt(1d - Math.pow(ct, 2));
6  r2 = Math.pow(alt, 2)+2d*q1+(this.A4-this.C4*srlat2)/Math.pow(q, 2);
7  r = Math.sqrt(r2);
8  d = Math.sqrt(this.A2*crlat2 + this.B2*srlat2);
9  ca = (alt+d)/r;
10 sa = this.C2*crlat*srlat/(r*d);
```

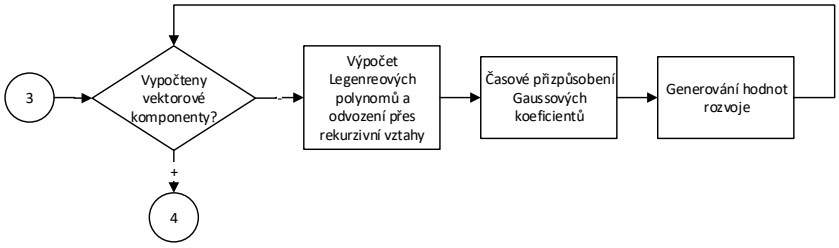
Úkolem druhého kroku je stanovení Gaussových koeficientů pro zadaný čas t z koeficientů modelu (diagram 3.4).



Obrázek 3.4: Druhý krok výpočtu složek magnetického pole Země

Stanovení koeficientů, které je implementací vztahu 3.6, je řízeno následujícím kódem.

```
1  for(m=2; m<=this._maxord; m++) {
2    this._sp[m] = this._sp[1]*this._cp[m-1]+this._cp[1]*this._sp[m-1];
3    this._cp[m] = this._cp[1]*this._cp[m-1]-this._sp[1]*this._sp[m-1];
4  }
```



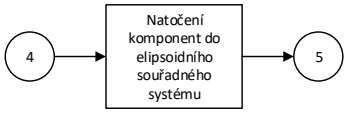
Obrázek 3.5: Třetí krok výpočtu složek magnetického pole Země

V třetím kroku jsou počítány vektorové komponenty za pomoci rozvoje dvanáctého řádu, což je blokově znázorněno diagramem 3.5.
Výpočet probíhá ve dvou smyčkách, jejichž definice, které jsou programovou implementací sum, uvedených ve vzorcích 3.7, 3.8 a 3.9. Smyčky vypadají následovně:

```
1  for(n = 1; n <= this._maxord; n++) {
2     ...
3     for(m = 0, D3=1, D4=(n+m+D3)/D3; D4>0; D4--, m+=D3) {
4        ...
5     }
6  }
```

Ve *for* smyčkách jsou naprogramovány všechny tři body, uvedené ve vývojovém diagramu výše.

Ve čtvrtém kroku je napsán kód, který odpovídá vztahu 3.14 (vývojový diagram 3.6).

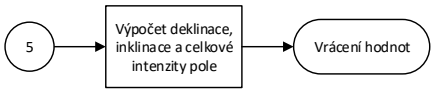


Obrázek 3.6: Čtvrtý krok výpočtu složek magnetického pole Země

Kód není přesnou replikou vztahů, a to proto, že některé proměnné byly vyjádřeny již v průběhu předchozích kroků.

```
1  bx = -bt*ca-br*sa;
2  by = bp;
3  bz = bt*sa-br*ca;
```

Pátým, zároveň i posledním krokem, je samotný výpočet složek magnetického pole a následné vrácení výsledků. Poslední krok je znázorněn diagramem 3.7.
Kód, který výpočet zajišťuje, je programovou podobou matematických vztahů, uvedených v 3.16.



Obrázek 3.7: Pátý krok výpočtu složek magnetického pole Země

```
1  bh = Math.sqrt(Math.pow(bx, 2) + Math.pow(by, 2));
2  ti = Math.sqrt(Math.pow(bh, 2) + Math.pow(bz, 2));
3  dec = Math.atan2(by, bx)/this.DTR;
4  dip = Math.atan2(bz, bh)/this.DTR;
```

Význam jednotlivých proměnných je zřejmý – *bh* je horizontální intenzita pole H, *ti* je celková intenzita pole F, *dec* je magnetická deklikace D a *dip* je vektor magnetické inklinace I. Funkce vrací tyto hodnoty v poli.

3.2 Implementace algoritmu pro stanovení magnetického severního pólu Země

3.2.1 Celkový pohled na algoritmus

Účelem tohoto algoritmu je sestrojení kompasu. Jak bylo zmíněno v teoretické části práce, různými konstrukcemi lze docílit rozdílných přesností při určování magnetického severu. Bylo dospěno k závěru, že vhodnou konstrukci lze získat za pomoci fúze sensorů typu akcelerometr, gyroskop a magnetometr, spolu s vyhovující filtrací naměřených hodnot.

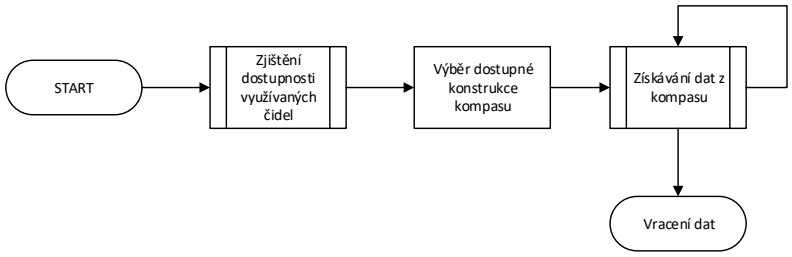
Ještě předtím, než jsou samotné výstupy z čidel zpracovány, je třeba zjistit, zda mobilní zařízení potřebná čidla obsahuje a umí je obsloužit. Na základě přítomnosti čidel lze stanovit nejpřesnější metodu určení azimutu, která je pro dané zařízení dostupná. Následně je možné přistoupit k odpovídající konstrukci kompasu, jejíž výsledkem je periodická aktualizace výstupních dat.

Obecný postup pro získání azimutu je zobrazen na vývojovém diagramu 3.8.

3.2.2 Zjišťování dostupnosti využívaných čidel

Zjištění dostupnosti čidel obsluhuje třída *HardwareChecker*. Třída implementuje rozhraní *SensorChecker*. Toto rozhraní obsahuje definice tří metod, kde každá z nich má za úkol zjistit dostupnost jednoho z používaných čidel (akcelerometr, gyroskop, magnetometr).

Příklad, jak zjistit dostupnost magnetometru v zařízení je uveden v následující ukázce kódu. Pro ostatní čidla lze aplikovat obdobný kód. Pro akcelerometr je definována programová konstanta *Sensor.TYPE_ACCELEROMETER*, pro gyroskop *Sensor.TYPE_GYROSCOPE*.



Obrázek 3.8: Obecný postup při získávání dat z kompasu

```
1 public static final int MAGNETOMETER = Sensor.TYPE_MAGNETIC_FIELD;
2 private boolean hasMagnetometer = false;
3 ...
4 if(sensorManager.getSensorList(MAGNETOMETER).size() > 0) {
5   hasMagnetometer = true;
6 }
```

Po zjištění dostupnosti čidel lze přistoupit k odposlouchávání výstupních dat kompasu, jehož implementace je popsána v dalším bodu. Ještě předtím je však nutné zvolit správný mód kompasu.

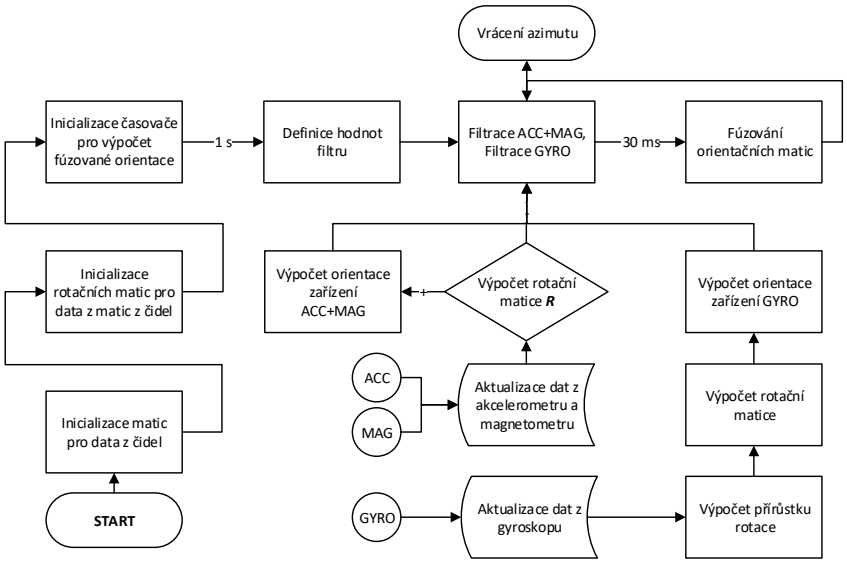
3.2.3 Vývoj kompasu pomocí fúze výstupu sensorů

O správný chod kompasu se stará třída *CompassFusion*. Postup, jakým je azimut získáván, je znázorněn na diagramu 3.9. Dle dostupných čidel je rozhodnuto o typu použitého kompasu. Typy jsou definovány výčtem *Mode*.

```
1 public enum Mode {
2   ACC_MAG,
3   GYRO,
4   FUSION
5 }
```

V průběhu bude popsán pouze typ *FUSION*. Ostatní případy se od něj liší tím, že nevyužívají některé možnosti, jak zlepšit přesnost. Princip postupu je stále shodný. Při práci se sensory při programování aplikací pro Android jsou získaná data uložena do matic. Jelikož budou údaje o orientaci pravidelně aktualizovány, je třeba mít tyto matice k dispozici. Zároveň je dobré mít stálý přístup k rotačním a orientačním maticím, které jsou z hodnot, naměřených čidly, počítány. S těmito maticemi je také periodicky operováno. Jejich inicializaci řeší první a druhý krok algoritmu.

Dalším krokem je inicializace časovače, který má za úkol periodicky fúzovat vypočtené orientace z čidel a tím aktualizovat samotný azimut. Časovač běží asynchronně. Každých 30 ms je volána instance privátní třídy, která dědí od třídy *TimerTask*. Tato třída umožňuje uživateli jednorázově či opakovaně asynchronně volat úlohu. Ještě předtím, než je poprvé úloha volána, je vyčkáno po dobu 1 s, a to proto, aby mohly být sensory inicializovány a začaly poskytovat datový výstup.



Obrázek 3.9: Postup při získání azimutu

```
1  private static final int DELAY = 1000;
2  public static final int TIME_CONSTANT = 30;
3  ...
4  this.mFuseTimer.scheduleAtFixedRate(new
       calculateFusedOrientationTask(), DELAY, TIME_CONSTANT);
```

Při instancování třídy jsou definovány filtrační hodnoty, které jsou v dalších fázích výpočtu aplikovány na orientační matice zařízení. Při výpočtu azimutu jsou asynchronně aktualizována data všech tří čidel. Aktualizace probíhá z vnější třídy (*PanoramaFragmentController*). Pomocí kódu níže jsou aktualizována data z akcelerometru.

```
1  public void setAccel(float[] sensorValues) {
2    System.arraycopy(sensorValues, 0, mAccel, 0, 3);
3  }
```

Hodnoty akcelerometru *sensorValues* jsou kopírovány od začátku (nulté pozice) do pole *mAccel*, opět od indexu nula. Posledním parametrem je kopírovaná délka – hodnoty z akcelerometru existují pro každou z os x, y, z.

Po aktualizaci dat z akcelerometru a magnetometru je proveden výpočet rotační matice R. K tomu je použita funkce *SensorManager.getRotationMatrix(...)*, která přejímá data, naměřená akcelerometrem a magnetometrem. Tato funkce transformuje vektor ze souřadného systému zařízení (lokálního) do globálního.

Vektor je uložen do rotační matice a je definován ortonormální bází [10], kde: X je dáno vektorovým součinem $Y \cdot Z$ (dotýká se země v místě polohy zařízení a přibližně míří východním směrem); Y se dotýká země v místě polohy zařízení a míří směrem k magnetickému severnímu pólu; Z je kolmý k zemi a míří tedy směrem vzhůru. Funkce

pro výpočet rotační matice vrací v případě úspěšného výpočtu *true*. Hodnota *false* je vrácena, je-li magnituda gravitace je menší, než $\frac{1}{10}$ nominální hodnoty (např. volný pád).

V kladném případě je na základě rotační matice vypočítána orientace zařízení. Tento postup je implementován ve funkci *calculateAccMagOrientation()*.

```
1 public void calculateAccMagOrientation() {
2   if (SensorManager.getRotationMatrix(mRotationMatrix, null, mAccel,
        mMagnet)) {
3     SensorManager.getOrientation(mRotationMatrix, mAccMagOrientation
        );
4   }
5 }
```

Ve chvíli, kdy jsou získána data z gyroskopu, je dle [11] vypočten rotační vektor, popisující změnu rotace ve všech směrech, závislou na časovém úseku. Poté je vypočítána rotační matice, na kterou je rotační vektor převeden. Z rotační matice je za pomoci funkce *getOrientation(...)* vypočtena orientace zařízení.

Nyní, jsou-li k dispozici všechny údaje, je přistoupeno k fúzování orientačních matic. To je v případě azimutu reprezentováno následujícím kódem. Kromě azimutu obsahují orientační matice ještě informace o rotaci v dalších dvou osách zařízení (indexy 1, resp. 2). Filtrační parametry, uvedené v kódu, byly získány na základě výsledků testů.

```
1 float FILTER_COEFFICIENT = 0.98f;
2 float ONE_MINUS_COEFF = 1.0f - FILTER_COEFFICIENT;
3 ...
4 mFusedOrientation[0] = FILTER_COEFFICIENT * mGyroOrientation[0] +
    ONE_MINUS_COEFF * mAccMagOrientation[0];
```

Fúzováním je tedy rozuměno sečtení orientačních matic z více sensorů, spolu s jejich fitrací, při vhodně zvolených parametrech. Na závěr je vypočtená hodnota azimutu vrácena a lze s ní operovat dále, v tomto případě při práci s panoramatem.

3.3 Pomocné knihovny

3.3.1 Účel těchto knihoven

Úkolem pomocných knihoven je zprostředkovat programátorovi znovupoužitelnost určitých funkcí. Funkce jsou zařazeny do tříd dle jejich účelu. Při vývoji aplikace byly funkce v knihovnách postupně doplňovány, než bylo vytvořeno potřebné portfolio.

Obsah knihoven je statický, je tedy dostupný bez potřeby vytvářet ze třídy objekt. To je výhodné i z důvodu úspory paměti. Jelikož funkce mají za úkol pouze převzít data, provést výpočet a vrátit výsledek, není tento návrh v rozporu s postuláty objektově orientovaného programování.

3.3.2 Třída GeoCoord

Jestliže bylo řečeno, že obsah knihoven je statický, tak to v tomto případě není naprostá pravda. Třída poskytuje staticky dostupné funkce. Kromě toho slouží její instance jako objekt, předávaný ve funkcích knihovny pro počítání s GPS souřadnicemi. Tomu se bude věnováno v části 3.3.4.

Funkce, obsažené v této knihovně, mají za úkol převádět číselnou reprezentaci GPS souřadnic do textové a naopak. Knihovna také převádí azimut na textovou reprezentaci světových stran. Seznam staticky dostupných funkcí spolu s účelem je uveden v tabulce 3.2.

Funkce, které vracejí číselné reprezentace údajů, pracují na principu parsování informací, které jsou následně dle převodních vztahů vyjádřeny jako číslo. Reverzní postup je aplikován ve funkcích, vracejících reprezentaci textovou. Pro představu, jak knihovna pracuje, je uvedeno několik příkladů výstupu funkcí:

getDecimalAzimuth("348°46’26"", 6) → 348,773889,
getDecimalLatOrLon("52° 53’20"W") → -52.8888889,
getDecimalAzimuth("65°24’48"", 6), 3) → "ENE",
getDMSLat(-1.8216667, "dm") → "001.0°49.0’S",
getDMSLon(52.8888889, "dms") → "052.0°53.0’20.0"E",
getDMSAzimuth(2160.8888889, "d") → "001.0°".

NÁZEV	NÁVRATOVÝ TYP	PARAMETRY	VRACÍ
getCompassPoint	String	azimut, přesnost	světové strany dle azimutu, přesnost je rozlišení
getDecimalLatOrLon	double	řetězec GPS souřadnice	číselnou reprezentaci GPS souřadnice
getDecimalAzimuth	double	azimut, desetinných míst	číselnou reprezentaci azimutu na počet des. míst
getDMSLat	String	stupně, formát	řetězec zeměpisné šířky dle formátu
getDMSLon	String	stupně, formát	řetězec zeměpisné délky dle formátu
getDMSAzimuth	String	stupně, formát	řetězec azimutu dle formátu

Tabulka 3.2: Seznam funkcí třídy GeoCoord

3.3.3 Knihovna MyMath

Tato knihovna je aplikací pravidla shlukování jednotlivých typů funkcí do jedné třídy. Knihovna obsahuje pouze dvě funkce, obě řídí zaokrouhlování velkých desetinných čísel. Je používána ve třídě *GeoCoord* při zaokrouhlování při převodech.

Rozdíl mezi datovým typem *double* a použitým (*BigDecimal*) je takový, že druhý zmiňovaný číslo nezaokrouhlí, ale počítá přesně. To je pro správný výstup převodu žádoucí. *BigDecimal* je v Javě používán při počítání s financemi nebo v případech, kde je vyžadována naprostá přesnost. Tento fakt se projeví především při násobení či dělení čísel. Příklad zaokrouhlení směrem od půli nahoru je uveden v následujícím kódu.

```
1 public static double round(double value, int places) {
2   if (places < 0) throw new IllegalArgumentException();
3   BigDecimal bd = new BigDecimal(value);
4   bd = bd.setScale(places, RoundingMode.HALF_UP);
5   return bd.doubleValue();
6 }
```

Kromě tohoto módu zaokrouhlení jsou dostupné ještě další (například *HALF_DOWN*, *DOWN* nebo *CEILING*). Všechny možnosti, včetně ukázek, jsou uvedeny v [1].

3.3.4 Knihovna LocationOps

Tato knihovna je programovou implementací matematických vztahů z části 1.2.1. Knihovna, která je reprezentována třídou *LocationOps*, je v práci často využívána. Příkladem je řazení rozhleden dle jejich vzdálenosti od polohy uživatele nebo výpočet směru mezi uživatelem a bodem panoramatu. Tato a další použití jsou postupně vysvětlena v dalších částech práce. Zde je vysvětlena pouze implementace funkcí. Všechny funkce přebírají GPS souřadnice jako datový typ *GeoCoord*, jak je uvedeno v části 3.3.2.

Prvním typem funkcí jsou funkce pro výpočet vzdálenosti mezi dvěma GPS souřadnicemi. V knihovně jsou implementovány tři funkce tohoto typu. První z nich pro svůj výpočet využívá vztah 1.3.

```
1 double a =
2         Math.pow(Math.sin(dFi/2), 2) +
3         Math.cos(fi1) * Math.cos(fi2) *
4         Math.pow(Math.sin(dLambda/2), 2);
5 double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
6 double d = R * c;
```

Druhá pro výpočet vzdálenosti využívá vztah 1.4.

```
1 double d =
2         Math.acos(Math.sin(fi1) * Math.sin(fi2) +
3         Math.cos(fi1) * Math.cos(fi2) *
4         Math.cos(dLambda)) * R;
```

Třetí funkce je implementací matematického vztahu 1.5.

```
1 double x = (lambda2 - lambda1) * Math.cos((fi1 + fi2)/2);
2 double y = fi2 - fi1;
3 double d = Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2)) * R;
```

Z kódu je vidět, že vztahy jsou kódově totožné, jako jejich matematická předloha. Na základě testování bylo zjištěno, že nejpřesnějším způsobem, jak vzdálenost počítat, je použití vztahu 1.3. Proto je v práci využívána pouze funkce, která tomuto vztahu přísluší.

Další funkce slouží pro výpočet směru, což je zajištěno pomocí vztahu 1.6. Tomu přísluší následující kód.

```
1 double y = Math.sin(lambda2 - lambda1) * Math.cos(fi2);
2 double x =
3         Math.cos(fi1) * Math.sin(fi2) -
4         Math.sin(fi1) * Math.cos(fi2) *
5         Math.cos(lambda2 - lambda1);
6 double b = Math.toDegrees(Math.atan2(y, x));
```

Před samotným výpočtem je předaným parametrem zjištěno, zda chce uživatel počítat počáteční nebo cílový směr. V návaznosti na to jsou souřadnice dosazeny do proměnných.

Následující výňatek z kódu slouží pro výpočet středového bodu mezi dvěma souřadnicemi, dle vztahu 1.7.

```
1 double Bx = Math.cos(fi2) * Math.cos(lambda2 - lambda1);
2 double By = Math.cos(fi2) * Math.sin(lambda2 - lambda1);
3 double fi3 =
4     Math.atan2(Math.sin(fi1) + Math.sin(fi2),
5      Math.sqrt(Math.pow(Math.cos(fi1) + Bx, 2) + Math.pow(By, 2)));
6 double lambda3 = lambda1 + Math.atan2(By, Math.cos(fi1) + Bx);
7
8 return new GeoCoord(Math.toDegrees(fi3), Math.toDegrees(lambda3));
```

Jak lze vidět, funkce vrací instanci třídy *GeoCoord*, jíž jsou v konstruktoru předány údaje o zeměpisné šířce a délce. Tímto způsobem může být s informacemi manipulováno dále. K souřadnicím se lze dostat pomocí getterů.

Předposlední funkce je určena k výpočtu cílového bodu, je-li uživatelem zadán počáteční bod, směr a vzdálenost, kterou chce „ujít“. Toho je dosaženo pomocí vztahu 1.8.

```
1 double fi2 =
2     Math.asin(Math.sin(fi1) * Math.cos(d / R) +
3     Math.cos(fi1) * Math.sin(d / R) * Math.cos(bearing));
4 double lambda2 =
5     lambda1 +
6     Math.atan2(Math.sin(bearing) * Math.sin(d/R) * Math.cos(fi1),
7     Math.cos(d/R) - Math.sin(fi1) * Math.sin(fi2));
8
9 return new GeoCoord(Math.toDegrees(fi2), Math.toDegrees(lambda2));
```

Poslední naprogramovaná funkce počítá průsečík mezi dvěma souřadnicemi za předpokladu, že je zadán směr z každé z nich. Výpočet je kódovou podobou matematického vztahu 1.9.

```
1 double delta12 = 2 * Math.asin(Math.sqrt(Math.pow(Math.sin(dFi / 2),
      2) + Math.cos(fi1) * Math.cos(fi2) * Math.pow(Math.sin(dLambda /
      2), 2)));
2 double theta1 =
```

```
3     Math.acos((Math.sin(fi2) - Math.sin(fi1) * Math.cos(delta12)) / (
          Math.sin(delta12) * Math.cos(fi1)));
4   double theta2 =
5     Math.acos((Math.sin(fi1) - Math.sin(fi2) * Math.cos(delta12)) / (
          Math.sin(delta12) * Math.cos(fi2)));
6   ...
7   if(Math.sin(lambda2 - lambda1) > 0) {
8     theta12 = theta1;
9     theta21 = 2 * Math.PI - theta2;
10  } else {
11    theta12 = 2 * Math.PI - theta1;
12    theta21 = theta2;
13  }
14
15  double alpha1 =
16    (theta13 - theta12 + Math.PI) % (2 * Math.PI) - Math.PI;
17  double alpha2 =
18    (theta21 - theta23 + Math.PI) % (2 * Math.PI) - Math.PI;
19
20  double alpha3 =
21    Math.acos(-Math.cos(alpha1) * Math.cos(alpha2) +
22    Math.sin(alpha1) * Math.sin(alpha2) * Math.cos(delta12));
23  double delta13 =
24    Math.atan2(Math.sin(delta12) * Math.sin(alpha1) * Math.sin(alpha2)
          ,
25      Math.cos(alpha2) + Math.cos(alpha1) * Math.cos(alpha3));
26  double fi3 =
27    Math.asin(Math.sin(fi1) * Math.cos(delta13) +
28    Math.cos(fi1) * Math.sin(delta13) * Math.cos(theta13));
29  double dLambda13 =
30    Math.atan2(Math.sin(theta13) * Math.sin(delta13) * Math.cos(fi1),
31      Math.cos(delta13) - Math.sin(fi1) * Math.sin(fi3));
32  double lambda3 = lambda1 + dLambda13;
33  lambda3 = (lambda3 + 3 * Math.PI) % (2 * Math.PI) - Math.PI;
34
35  return new GeoCoord(Math.toDegrees(fi3), Math.toDegrees(lambda3));
```

Na první pohled je zřejmé, že tato funkce je výpočetně nejnáročnější funkcí knihovny. Při dalším možném vývoji aplikace však lze nalézt její platné využití, například při zlepšování přesnosti zobrazení panoramatu.

3.4 Komunikace se serverem

3.4.1 Knihovna Retrofit

Klient vznese prostřednictvím HTTP požadavku dotaz na server, který jej zpracuje postupy, vysvětlenými v kapitole 2. Vrácenou odpověď klient přijme jako soubor. Korektní přijetí souboru zajišťuje knihovna Retrofit.

Retrofit je typově bezpečný REST klient pro Android a Javu. REST je architektura rozhraní navržená pro prostředí, kde část programu běží na jiném stroji a pro komunikaci využívá počítačovou síť. Má za úkol jednoduše tvořit, číst, editovat

a mazat informace ze serveru, za pomoci jednoduchých HTTP volání. Stav aplikace je vyjádřen klíčovou abstrakcí. Každá má unikátní identifikátor URI. Komunikace je bezstavová, což znamená, že každý požadavek musí obsahovat všechny informace, nutné pro jeho vykonání.

Prvním krokem, který je nutné před samotným programováním za pomoci knihovny udělat, je importovat ji do projektu. To je provedeno pomocí příkazu

```
1 dependencies {
2   compile 'com.squareup.retrofit2:retrofit:2.1.0'
3   compile 'com.squareup.retrofit2:converter-gson:2.1.0'
4 }
```

Ten je přidán do definovaného místa v souboru *build.gradle*, který je součástí projektu, vytvořeného v Android Studiu. Po následné kompilaci tohoto souboru jsou prostředky knihovny dostupné a lze je využívat. Třetí řádek kódu zajišťuje správnou interpretaci souborů typu JSON.

3.4.2 Definice aplikačního rozhraní

Po úspěšném zavedení balíku Retrofit je třeba definovat aplikační rozhraní pro komunikaci se serverem. Vyvíjená aplikace poskytuje komunikační funkce v rozhraní *IRozhlednyCZRetrofit*.

```
1  @GET("get_lots")
2  Call<OLots> getLots();
3
4  @GET("get_view_points")
5  Call<OViewPoints> getViewPoints();
6
7  @GET("get_view_raw")
8  Call<OViewRaw> getViewPointsRaw();
9
10 @GET("get_view")
11 Call<OView> getView(@Query("id") int id);
12
13 @Streaming
14 @GET("get_image")
15 Call<ResponseBody> getImage(@Query("url") int url, @Query("size")
       int size);
```

Na serverové straně figurují ještě dvě funkce, ty ale nejsou v klientské části prozatím využívány, a proto nejsou v API definovány. Anotace *@GET* určuje, že požadavek bude obsloužen pomocí typu GET. Řetězec, obsažený v anotaci, se odkazuje na konkrétní funkci na serveru.

Přijatá data jsou zpracována pomocí rozhraní *Call*, které pracuje s genericitou. Jako generický typ je předán POJO, na který se odpověď automaticky namapuje. O proces mapování se stará Retrofit.

U posledních dvou funkcí si lze všimnout podoby předaných parametrů za pomoci anotace *@Query*. Řetězec, uvedený v anotaci, je určen názvem parametru v požadavku GET. V případě funkce pro získání obrázku je navíc aplikována anotace

@Streaming. Ta značí, že bude vrácen proud binárních dat – ta jsou obsažena v datovém typu *ResponseBody*.

3.4.3 Definice mapovacího POJO modelu

Jak bylo uvedeno výše, data ve formátu JSON jsou mapována na objekty. Díky tomu lze využívat přístupu k vráceným datům právě tak, jako kdyby byla objekt. Pro každý JSON, který je na serveru uložen nebo je funkcí rozhraní generován, je vytvořena mapovací třída.

Tato třída identicky kopíruje strukturu, vytvořenou ve vráceném JSONu. Vhodným příkladem, jak mapování vysvětlit, je třída *OViewPoints*. Na tu je mapován výsledek dotazu na soubor, obsahující všechny body panoramatu. Třída shromažďuje pole bodů panoramatu, které je uloženo v datové struktuře *List*.

```
1 private List<OViewPoint> data;
```

Bod panoramatu je reprezentován třídou *OViewPoint*. Jejími atributy jsou názvy jednotlivých klíčů v souboru JSON tak, jak je uvedeno v tabulce 2.2.

```
1  public class OViewPoint {
2    @Expose
3    private int id;
4    @Expose
5    private int altitude;
6    ...
7    @Expose
8    private int type;
9    ...
10 }
```

Anotací *@Expose* lze ovlivnit, zda bude daný atribut z JSONu serializován či nikoliv. V případě uvedení anotace serializován bude. Podobnou vlastnost lze provádět i pro deserializaci, a to použitím

```
1 @Expose(serialize = false, deserialize = false)
```

Použitím *true* nebo *false* lze zahrnutí proměnné kontrolovat v obou směrech.

3.4.4 Příprava REST klienta

Předtím, než budou moci být funkce z API v aplikaci použity, je třeba inicializovat REST klienta. To obstarává statická funkce třídy *RestClient*. Tato funkce vrací aplikační rozhraní *IRozhlednyCZRetrofit*. Tělo funkce je zobrazeno v následující ukázce kódu.

```
1 HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();
2 interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
3 OkHttpClient client = new OkHttpClient.Builder().addInterceptor(
      interceptor).build();
4
5 Retrofit retrofit = new Retrofit.Builder()
6   .baseUrl(ServerSettings.SERVER_PREFIX + ServerSettings.API_PREFIX)
7   .addConverterFactory(GsonConverterFactory.create())
```

```
8     .client(client)
9     .build();
10
11  return retrofit.create(IRozhlednyCZRetrofit.class);
```

První tři řádky kódu inicializují logování komunikace. Jsou tedy dobrým ladícím prostředkem při případném řešení potíží v komunikaci. Pátý až devátý řádek vytváří instanci balíku Retrofit.

Na šestém řádku je předána adresa serveru, spolu s cestou ke skriptu, obsluhujícímu aplikační rozhraní na serverové straně. Sedmý řádek ukázky zobrazuje předání instance konvertoru souborů. V případě, že uživatel požaduje využití jiného formátu než JSON nebo chce užít vlastní typ souborů, je třeba předat parametr, obsahující vhodný konvertor. Na osmém řádku je definice klienta pro komunikaci se serverem. V tomto případě je použit klient typu *OkHttpClient*, instancovaný výše. Poslední řádek vrací instanci, jíž je předáno rozhraní.

3.5 Uložení databázových dat v zařízení

3.5.1 Principy uložení databázových dat v zařízení

Všechna data, která nejsou obrazová nebo součástí dat panoramatu, jsou uložena v interní databázi zařízení. Databázové entity jsou uvedeny na obrázku 3.10.

Entity *County*, *Gradient*, *View* a *Material* jsou plněny přímo v zařízení, bez nutnosti stahování dat ze serveru. Obsahují správnou jazykovou mutaci dat, které jsou párovány pomocí klíčů typu číselník tabulky *Lot*. V ní jsou uloženy informace o rozhlednách. Tabulka *ViewPointType* je plněna také v zařízení a obsahuje geografické typy bodů panoramatu. Tyto entity tedy slouží jako číselníky.

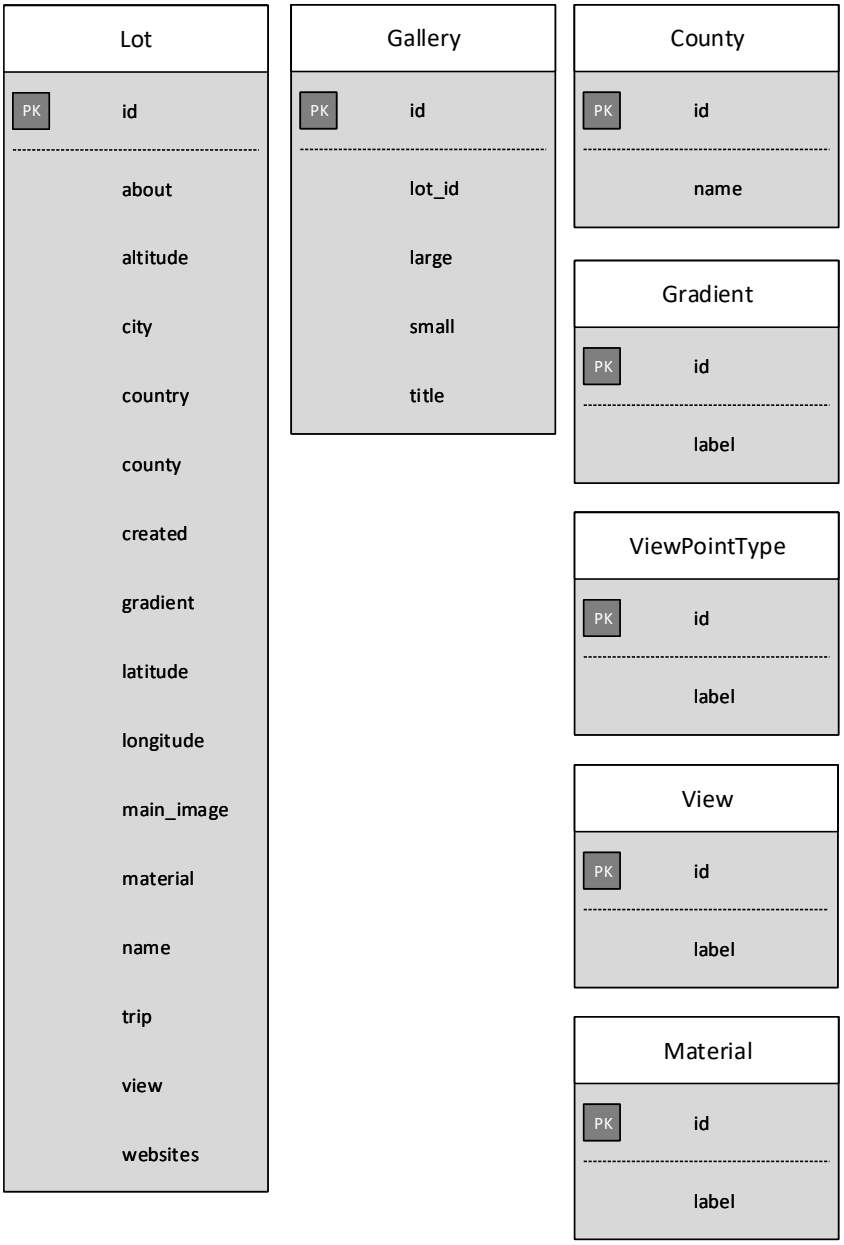
Tabulka *County* obsahuje seznam názvů krajů v České republice, *Gradient* má v sobě uloženy informace o prudkosti stoupání k rozhledně. Entita *View* uchovává data o tom, jakými směry lze z rozhledny vidět. Tabulka *Material* je poslední, která je párována. Jsou v ní informace o materiálu, z kterého byla rozhledna postavena. Entita *Gallery* uchovává informace o obrazových datech rozhledny.

3.5.2 Knihovna Requery

Pro správu databáze v zařízení je použita knihovna Requery. Ta mapuje objekty na databázové entity a generuje SQL. Spolu s podporou Javy 8 se tak stává vhodným nástrojem, jak pracovat s interními daty. Za pomoci Requery lze snadno mapovat či vytvářet databáze, provádět SQL dotazy a modifikovat data z jakékoliv platformy, která Javu používá [21].

Knihovna mapuje atributy a vytváří databázový model při kompilaci, což v případě platformy Android znamená, že výkon při čtení objektů pomocí dotazu je stejný, jako při využití standardních prostředků Android API (*Cursor* a *ContentValues*).

Dále je možné definovat vztahy mezi entitami za použití anotací. Entity, které mezi sebou mají vztah $m : n$, mohou být generovány automaticky. Knihovna podpo-



Obrázek 3.10: Databázové entity v zařízení

ruje velké množství databází, například PostgresSQL, MySQL, SQLite nebo Oracle DB. V případě Androidu je obrovskou výhodou podpora databindingu, což v praxi přináší markantní úsporu kódu a prakticky bezúdržbovou aktualizaci dat při jejich zobrazení. Výsledky dotazů lze uchovávat v mnoha různých datových strukturách, záleží tedy pouze na uživateli knihovny, které z nich jsou pro něj vhodné.

Před samotným použitím knihovny je třeba provést její import.

```
1 dependencies {
```

```
2     apt 'io.requery:requery-processor:1.0.0'
3     compile 'io.requery:requery-android:1.0.0'
4   }
```

3.5.3 Definice databázových entit

Tabulky mohou být definovány buď jako abstraktní třída, nebo rozhraním. V práci byl zvolen postup definice za pomoci rozhraní. Vhodným příkladem demostrace definice je entita *Lot*. Atributy entit jsou definovány dle vzoru databáze z obrázku 3.10. Ostatní entity jsou pouze analogicky upraveny.

```
1  @Entity
2  public interface Lot extends Observable, Parcelable, Persistable {
3    @Bindable @Key int getId();
4    @Bindable int getAltitude();
5    @Bindable String getCity();
6    ...
7    @Bindable int getGradient();
8    @Bindable double getLatitude();
9    @Bindable double getLongitude();
10   ...
11   @Bindable String getName();
12   @Bindable int getView();
13   @Bindable String getWebsites();
14 }
```

Nezbytným krokem pro rozpoznání databázové entity je uvedení anotace *@Entity* před definicí rozhraní. Každé rozhraní je zároveň rozšířením tří dalších. V případě, že dojde k asynchronní změně dat v entitě, jsou pozorovatelé díky *Observable* notifikováni a mohou provést další akce.

Díky *Parcelable* lze předávat objekty mezi aktivitami a fragmenty. Toho je využíváno v případě potřeby předání dat do jiné části aplikace, kde s nimi lze po přijetí pracovat. Rozhraní *Persistable* je součástí knihovny Requery a zajišťuje správnou funkcionalitu ve vygenerovaných entitách. Primární klíč entity je definován pomocí anotace *@Key*. Anotace *@Bindable* umožňuje používat atribut tabulky pomocí databindingu. Atributy jsou generovány pomocí getterů, kde název atributu začíná velkým písmenem.

3.5.4 Vytváření databáze

Vytvoření databáze je realizováno třídou *LOTSCZApplication*, která rozšiřuje třídu *Application*. Ta slouží jako základní třída pro správu obecného stavu aplikace a je instancována při jejím spuštění.

LOTSCZApplication má jediný privátní atribut, který udržuje databázi.

```
1  private EntityDataStore<Persistable> dataStore;
```

EntityDataStore je rozhraní, které jako generický typ převezme *Persistable* – to je implementováno v jednotlivých entitách. Toto rozhraní provádí konkrétní implementaci pro použití s databázemi SQL.

Při spuštění aplikace je provedena kontrola, zda již byla instance databáze vytvořena. Pokud ne, je vytvořen nový databázový model. Dále jsou naplněny entity s daty, získanými interně. Postup, jakým je databáze vytvářena, je uveden v následující ukázce. Pro ostatní interně vytvořené entity je aplikován obdobný postup.

```
1  if(this.dataStore == null) {
2    EntityModel model = Models.DEFAULT;
3    DatabaseSource source = new DatabaseSource(this, model, 4);
4    Configuration configuration = source.getConfiguration();
5    this.dataStore =  new EntityDataStore<Persistable>(configuration);
6
7    Set<County> countyEntitySet = new TreeSet<>(...);
8    CountyEntity e = new CountyEntity();
9    e.setId(1);
10   e.setName(getResources().getString(R.string.hlavni_mesto_praha));
11   countyEntitySet.add(e);
12   ...
13   this.dataStore.insert(countyEntitySet);
14 }
```

Definice *EntityModel* je vytvářena automaticky při kompilaci. Při instancování databázového zdroje na třetím řádku je důležité si všimnout třetího parametru. Je jím číslo, označující verzi databáze. Je nutné, aby bylo iterováno kdykoliv, byla-li provedena změna v databázovém schématu. Kompilátor tím pozná, že změna nastala a provede nové sestavení databáze.

Na sedmém řádku je vytvořena struktura, uchovávající jednotlivé řádky tabulky. Příklad vytvoření jednoho z nich je uveden na řádkách osm až jedenáct. Po dokončení vytváření potřebných dat je celá struktura uložena do tabulky. To je úkolem řádku číslo čtrnáct. Způsob přístupu k datům bude vysvětlen při implementaci grafického rozhraní, až bude potřeba s nimi pracovat.

3.6 Uložení obrazových dat v zařízení

3.6.1 Knihovna Universal Image Loader

Pro ukládání a přístup k obrazovým datům, která nejsou součástí grafiky aplikace, je použita knihovna Universal Image Loader. Knihovna poskytuje uživateli přizpůsobitelný a výkonný nástroj pro načítání, cachování a zobrazení obrázků. Umožňuje velké množství konfiguračních alternativ a kontrolu nad celým procesem.

Knihovna umí načítat obrázky ve více vláknech, a to jak synchronně, tak i asynchronně. V možnostech konfigurace lze nastavit parametry, jako je simultánní počet vláken, dekodér obrázků, velikost cache a další.

UIL je opět nutné nejdříve do projektu importovat. To je provedeno následujícím příkazem.

```
1 dependency {
2   compile 'com.nostra13.universalimageloader:universal-image-loader
        :1.9.5'
3 }
```

3.6.2 Konfigurace knihovny

Před použitím v aplikaci je nezbytným krokem knihovnu nakonfigurovat. Níže je uvedena část kódu, která tento bod řeší.

```
1  private static DisplayImageOptions mDisplayImageOptions = new
       DisplayImageOptions.Builder()
2    .bitmapConfig(Bitmap.Config.RGB_565)
3    .cacheInMemory(true)
4    .cacheOnDisk(true)
5    .build();
6  ...
7  ImageLoaderConfiguration config = new ImageLoaderConfiguration.
       Builder(c)
8    .defaultDisplayImageOptions(mDisplayImageOptions)
9    .writeDebugLogs()
10   .threadPoolSize(...)
11   .build();
12 ImageLoader.getInstance().init(config);
```

Instance knihovny je Singleton, a tak jí lze získat jediným přístupovým bodem v rámci celé aplikace. Statická proměnná *mDisplayImageOptions* určuje dekodér a další parametry, jako možnost cachovat obrázky do interní paměti nebo na paměťovou kartu zařízení.

V konfiguraci je dále povoleno logovat veškeré akce UIL. Je také nastaven počet zároveň pracujících vláken. Celá konfigurace je předána v parametru inicializační metody. Proměnná *c* je datového typu *Context* a je v ní uchován kontext aplikace.

Ještě před konfigurací knihovny je v manifestu projektu nutné povolit ukládání do externí paměti (za předpokladu, že bude použito).

```
1 <uses-permission android:name="android.permission.
      WRITE_EXTERNAL_STORAGE" />
```

3.6.3 Způsob uložení obrazových dat

Po získání instance knihovny je další možnou akcí obrázek vykreslit. Díky tomu, že bylo v konfiguračních parametrech povolení cachování, jsou obrázky ukládány automaticky, bez nutnosti dalšího zásau programátora. Není tedy nutné si někde uchovávat cestu s souboru v zařízení, vše je v režii knihovny, která se stará o bezproblémový chod všech stažených zdrojů.

Způsob, jak obrázek vykreslit do prvku grafického rozhraní *ImageView*, je v následující ukázce.

```
1  ...
2  ImageLoader.getInstance().displayImage(url, imageView);
```

Volání obsahuje jednoduchý příkaz, kde je předána URL adresa k obrázku. Druhým parametrem je instance prvku grafického rozhraní, do kterého chce uživatel obrázek vykreslit.

4 Klientská část (frontend)

Poslední částí práce, která předchází testování navrženého systému, je frontendová část mobilní aplikace. Jejím účelem je komunikace s uživatelem prostřednictvím navrženého uživatelského prostředí.

Nejdříve bude popsána navigace v grafickém rozhraní a způsob zobrazení dat. Poté bude vysvětleno, jak lze při návrhu implementovat použití vlastních fontů. Následně se bude věnováno návrhům a funkcím jednotlivých obrazovek aplikace. V tom jsou zahrnuty také techniky pro kalibraci čidel a zobrazení panoramatu.

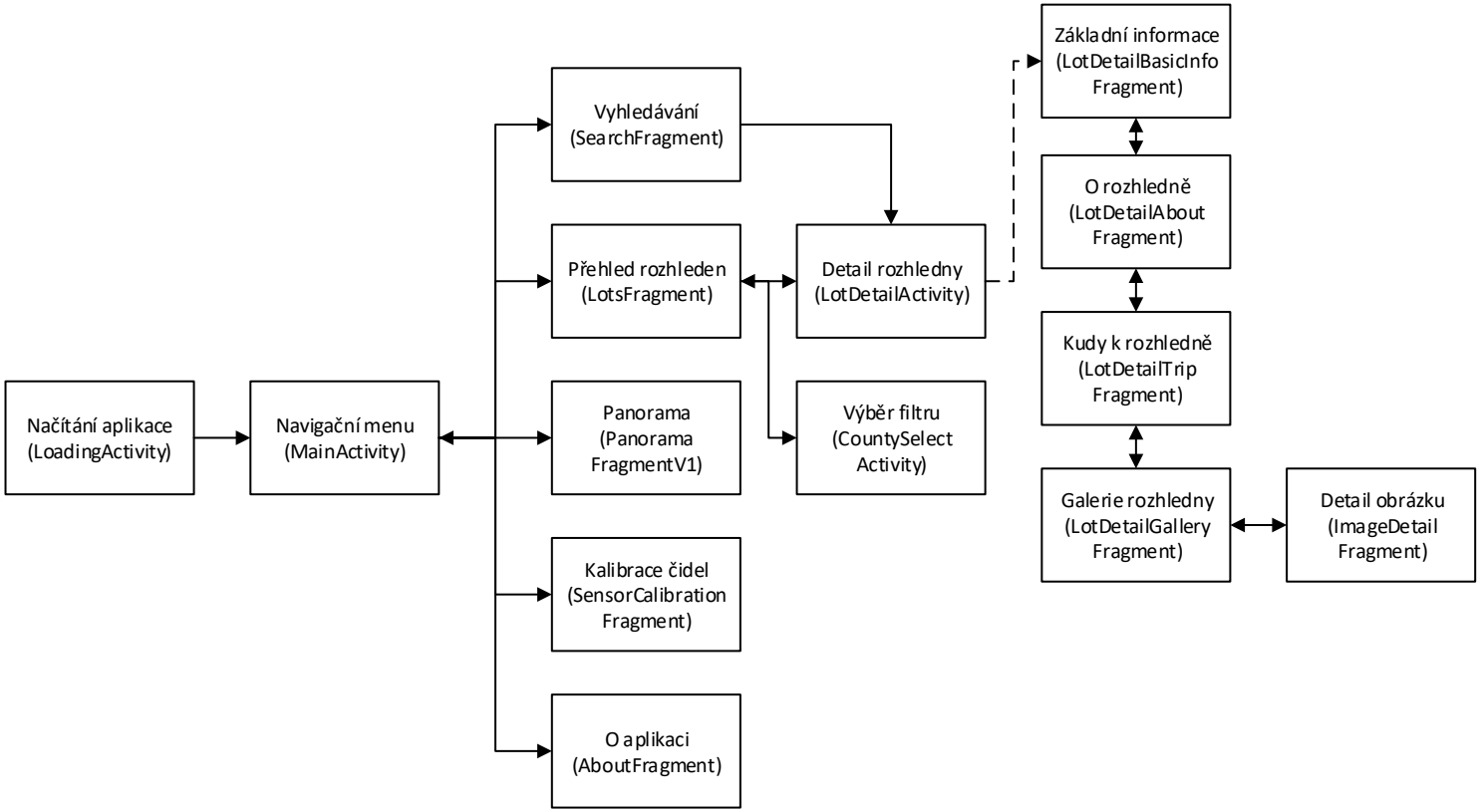
4.1 Celkový pohled na grafické rozhraní aplikace

4.1.1 Navigace mezi okny

Aplikace sestává z několika funkčních oken, v nichž uživatel může procházet údaje o rozhlednách, filtrovat data, třídit je a vyhledávat v nich. Aplikace také zahrnuje možnost kalibrovat čidla – pro případnou eliminaci nepřesností při prohlížení panoramatu. Schéma přepínání mezi jednotlivými okny je znázorněno na obrázku 4.1.

Z obrázku lze zaregistrovat, že mezi některými obrazovkami jde procházet cyklicky. Navigační menu je pro uživatele vstupním bodem v aplikaci. Menu je zobrazeno po načtení aplikace a lze se díky němu dostat do jednotlivých částí. Při navigaci v detailu rozhledny je možné využít přepínání mezi čtyřmi fragmenty (přerušovaná šipka znázorňuje přímou implementaci fragmentů v rámci aktivity).

Při vyhledávání v datech se lze přesměrovat do jejich detailu. V přehledu evidovaných rozhleden je možné vyvolat podnabídku, která uživateli zpřístupní filtraci rozhleden dle geografického rozložení.



Obrázek 4.1: Přehled navigace mezi aktivitami

4.1.2 Android Data Binding Library

Každá aplikace pracuje s nějakou formou dat. Ta mohou být získána z internetu, interní databáze cílového zařízení nebo mohou být vytvořena uživatelem v průběhu využívání aplikace. Pro všechny uvedené případy platí, že vývojář musí zajistit, aby data mohla být zobrazena v prvcích uživatelského rozhraní, které je bude prezentovat.

Typickým způsobem, jak při vývoji aplikací pro Android data prezentovat, je získat layout dané aktivity (případně fragmentu), následně zaměřit uživatelský prvek pomocí metody *findViewById* a nakonec přiřadit elementu data.

Na konferenci Google I/O 2015 byla představena nová knihovna, nazvaná Data Binding Library. Tato knihovna pomáhá vývojářům realizovat výše uvedené kroky pouze využitím layoutů a specifikované definice tříd a proměnných.

Výhodou tohoto přístupu je úspora času, potřebného pro psaní kódu. Ušetřených řádků je o desítky, v případě složitějšího grafického rozvržení aktivity i o stovky méně. Z toho také plyne lepší čitelnost kódu, která přispívá ke snazšímu nalezení zanesených chyb.

Pro použití této knihovny je nutné do souboru *build.gradle* v aplikačním modulu vložit následující [7].

```
1 android {
2     ...
3     dataBinding {
4       enabled = true
5     }
6 }
```

Je také třeba se ujistit, že v závislostech v konfiguračním souboru projektu je verze Gradle 1.5 nebo vyšší. Po následné synchronizaci je knihovna Data Binding vývojáři k dispozici.

4.2 Knihovna Calligraphy

4.2.1 Účel knihovny

Knihovna Calligraphy slouží programátorům aplikací pro Android k jednoduché implementaci vlastních fontů. Ve výchozí konfiguraci podporuje Android pouze písmo Roboto. Chce-li tedy programátor použít font vlastní, je nutné použít jednu z existujících knihoven.

Kromě Calligraphy existuje ještě například možnost využití knihovny Fontain. Ta sice poskytuje programátorovi větší prostor v nastavení, s tím ale souvisí její složitost. Pro účely aplikace postačí první zmíněná knihovna.

Bezespornou výhodou Calligraphy oproti jiným knihovnám je skutečnost, že fonty jsou aplikovány přímo na prvky nativního uživatelského rozhraní. Ostatní knihovny však mají ve většině případů vytvořeny grafické prvky vlastní.

Před použitím knihovny je nutné provést její import do projektu.

```
1 dependencies {
```

```
2     compile 'uk.co.chrisjenx:calligraphy:2.2.0'
3 }
```

4.2.2 Nastavení knihovny

Vlastní fonty je třeba nahrát do složky *assets*. Tu lze vytvořit buď v adresářové struktuře projektu, nebo jednoduše prostřednictvím vývojového prostředí Android Studia. Složka musí být umístěna v adresáři *app/src/main/*. V ní je následně vytvořena podsložka *fonts*, kam jsou písma nahrána.

Ve vyvíjeném projektu jsou použity dva fonty. Prvním je *Open Sans* [1], který podporuje českou diakritiku a je hlavním fontem aplikace. Druhé použité písmo *Parker* [2] podporu českých znaků nemá a je použito pro pouze pro logo aplikace.

Při inicializaci aplikace je nutné knihovnu nakonfigurovat. To obstarává vytvořená třída *Application*, která dědí od *LOTSCZApplication*. Tato třída je vstupním bodem celé aplikace. Knihovna je konfigurována v implementované metodě *onCreate()*.

```
1 CalligraphyConfig.initDefault(new CalligraphyConfig.Builder()
2     .setDefaultFontPath("fonts/OpenSans-Light.ttf")
3     .setFontAttrId(R.attr.fontPath)
4     .build()
5 );
```

Calligraphy počítá s umístěním fontů ve složce *assets*, a tak je cesta k písmu uváděna od ní. Tímto kódem je knihovna nakonfigurována a až na jednu akci je veškeré renderování fontů v její režii.

4.3 Načítání aplikace

4.3.1 Grafické rozvržení obrazovky

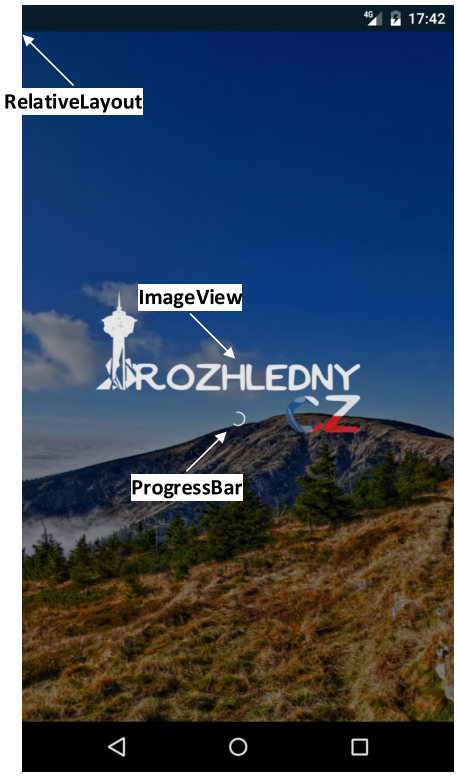
Obrazovka načítání aplikace obsahuje jednoduchý layout. Celé rozvržení je obaleno prvkem *RelativeLayout*. To umožňuje popsat pozicování prvku za pomoci vztahu k jeho rodiči. Druhou možností je popis umístění ve vztahu k jinému prvku stejné úrovně.

Jako pozadí layoutu je nastaven obrázek, který je uložen ve zdrojích, takzvaných *drawables*. Dále je vloženo logo aplikace, které je na obrazovce vycentrováno. Ve vztahu k němu je pozicován *ProgressBar*, který má za úkol indikovat načítání aplikace. Rozvržení je graficky demonstrováno obrázkem 4.2.

4.3.2 Rozšíření funkčních možností třídy

Třída dědí od třídy *Activity*. Ta je obecným prostředkem, jak s uživatelem komunikovat. Stará o vytváření okna s grafickým rozhraním. Ačkoliv jsou aktivity

[1] https://fonts.google.com/specimen/Open+Sans
[2] https://www.behance.net/gallery/18632037/Parker-Font-(Free)



Obrázek 4.2: Grafické rozvržení LoadingActivity

nejčastěji uživateli prezentovány jako celoobrazovková okna, lze je také používat jinými způsoby, například jako plovoucí okno nebo uvnitř jiné aktivity.

Aktivita se stará o celý životní cyklus obrazovky, který je popsán v [6]. Pro tyto účely je implementována celá řada metod.

4.3.3 Průběh načítání aplikace

Prvním krokem, který je proveden ještě před samotnou kompilací, je provázání databindingu z aktivity do jejího layoutu. V XML souboru s grafickým rozvržením je třeba přidat následující řádky.

```
1  ...
2  <data>
3    <import type="android.view.View"/>
4    <variable name="visible" type="boolean"/>
5  </data>
6  ...
7    <ProgressBar
8      ...
9      android:visibility="@{visible ? View.VISIBLE : View.GONE}"
10   />
11   ...
```

Kódem uvnitř tagu *data* se určuje, jaké proměnné a datové typy budou pro databinding aktivity používány. V tomto případě je importován datový typ *View*, s jehož pomocí bude kontrolována viditelnost prvku, indikujícího načítání aplikace. Poté je vytvořena proměnná *visible*, která je datového typu *boolean*. Tato proměnná je provázána s aktivitou, kde k ní lze přistupovat.

Způsobem uvedeným na devátém řádku lze kontrolovat viditelnost *ProgressBaru*. Každá změna hodnoty proměnné *visible* je automaticky promítnuta do layoutu.

Poté je třeba projekt zkompilovat, čímž je vygenerována třída, obsluhující binding. Jméno této třídy je závislé na názvu souboru s layoutem [8]. Párování je znázorněno v kódu.

```
1  private ActivityLoadingBinding mBinding;
2  ...
3  mBinding = DataBindingUtil.setContentView(this, R.layout.
       activity_loading);
4  mBinding.setVisible(true);
5  ...
```

Čtvrtý řádek kódu ukazuje, jak lze k proměnné definované v layoutu přistupovat. Po jeho provedení se *ProgressBar* stává viditelným.

V dalším kroku je provedena asynchronní akce instancováním třídy *AsyncTask*, což je při programování aplikací pro Android základním prostředkem, jak provádět operace na pozadí vlákna UI. Akce za předpokladu, že je databáze prázdná, obslouží stažení dat ze serveru a jejich následné uložení. Postup je realizován následující ukázkou.

```
1  if(((LOTSCZApplication) getApplication()).isDataStoreEmpty()) {
2    Call<OLots> oCall = RestClient.getRestClient().getLots();
3
4    oCall.enqueue(new Callback<OLots>() {
5      @Override
6      public void onResponse(Call<OLots> c, Response<OLots> r) {
7        ...
8        Intent i = new Intent(getApplicationContext(), MainActivity.
             class);
9        mBinding.setVisible(false);
10       startActivity(i);
11       finish();
12     }
13   ...
14   });
15 }
16 ...
```

Druhý řádek kódu volá metodu z API, popsaného v části 3.4.2. Na čtvrtém řádku je volána metoda *enqueue*, která vytváří nové vlákno. V případě úspěšného volání je v metodě *onResponse* provedeno uložení informací o rozhlednách obdobně, jak bylo vysvětleno v 3.5.4.

Následně je ukazatel načítání skryt a uživatel je přesměrován do hlavní aktivity celé aplikace. Jedenáctý řádek kódu odstraní aktivitu načítání z kontextu. To v praxi znamená, že při užití tlačítka zpět nebude uživatel do této aktivity vrácen.

Nastane-li případ, že data ze serveru stažena nebyla, je zavolána metoda *onFailure*, kde může programátor provést další kroky. Před samotným používáním síťové komunikace je nutné do souboru *AndroidManifest.xml* přidat oprávnění pro její použití.

```
1 <uses-permission android:name="android.permission.INTERNET" />
```

4.4 BaseFontActivity

Tato třída slouží pro aplikaci uživatelského fontu v aktivitě. Všechny ostatní aktivity, které budou použity, od této aktivity dědí. Nastavení fontu probíhá v metodě *attachBaseContext*, která slouží pro modifikaci chování třídy bez toho, aniž by musel být pozměněn originální kontext.

```
1 @Override
2 protected void attachBaseContext(Context newBase) {
3   super.attachBaseContext(CalligraphyContextWrapper.wrap(newBase));
4 }
```

Třída tedy slouží jako pomocný prvek, do kterého je obalen kontext ostatních aktivit.

4.5 Navigační menu aplikace

4.5.1 Grafické rozvržení obrazovky

Grafické rozhraní navigačního menu je obaleno elementem *FrameLayout*. Toto rozložení je vhodné používat v případě, že je potřeba rezervovat místo na obrazovce pro design jednoho prvku. Obecně platí, že by tento layout měl být používán pro obsluhu jediného potomka typu *view*. Organizace bez vzájemného překrývání je při jejich větším počtu obtížná.

Uvnitř je v prvku *ImageView* vnořeno pozadí menu. Druhým vnořeným elementem je *DrawerLayout*. Ten slouží jako obecný kontejner pro návrh posuvných interaktivních oken, která se primárně chovají jako menu. Tato okna mohou být vysunuta z jedné nebo obou stran obrazovky.

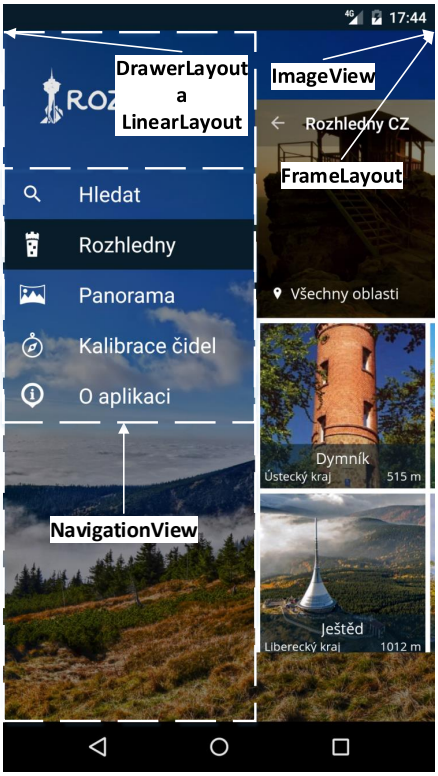
Do tohoto prvku je kromě definice chování menu při zobrazení (třída *FadingViewBehavior*) vložen *LinearLayout*. Ten slouží pro shlukování grafických prvků v jednom směru – buď horizontálně nebo vertikálně – přičemž prvky jsou uskupeny jeden po druhém.

V lineárním layoutu je vložen *NavigationView*, což je standardní postup při tvorbě menu. To může být pro přehlednost popsáno v externím souboru, což dokazuje tento případ. Soubor je umístěn ve složce *res/menu*, což je doporučené umístění pro layouty navigace.

```
1 ...
2 <group android:checkableBehavior="single">
3   <item
```

```
4       android:id="@+id/nav_search"
5       android:title="@string/main_drawer_search"
6       android:icon="@drawable/ic_nav_search"
7     />
8     ...
9   </group>
10  ...
```

Tímto způsobem jsou postupně definovány všechny položky menu. Celé rozvržení aktivity je pro lepší pochopení také na obrázku 4.3.



Obrázek 4.3: Grafické rozvržení MainActivity

Jak je z obrázku vidět, aktivní fragment je při vyjetí menu škálován. To zajišťuje metoda *onDrawerSlide*, která je aplikována na *DrawerLayout* takto:

```
1  ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(...) {
2    @Override
3    public void onDrawerSlide(View drawerView, float slideOffset) {
4      super.onDrawerSlide(drawerView, slideOffset);
5      ...
6      mBinding.mainLayout.getRoot().setScaleX(1 - slideOffset / 5);
7      mBinding.mainLayout.getRoot().setScaleY(1 - slideOffset / 5);
8      ...
9    }
10 };
```

```
11  mBinding.drawerLayout.addDrawerListener(toggle);
```

Nejdříve je nutné vytvořit objekt typu *ActionBarToggle*, který je metodou *onDrawerSlide* připraven odposlechnout akci vysunutí menu. V ní lze modifikovat samotné rozměry layoutu dle určených parametrů (například poměr míry vysunutí a konstanty). Připravený objekt je následně v metodě *addDrawerListener* předán layoutu menu, čímž odposlech začne. Propojení s databindingem je realizováno obdobným způsobem, jako je popsáno v části 4.3.3.

Pro použití *NavigationView* je nejprve nutné provést import podpůrné knihovny. Ačkoliv je vytvořena Googlem, není standardní součástí frameworku.

```
1  dependencies {
2    compile "com.android.support:support-v4:23.3.0"
3  }
```

4.5.2 Rozšíření funkčních možností třídy

Aktivita implementuje dvě rozhraní. Prvním z nich je *View.OnClickListener*, pomocí kterého je realizována reakce na stisk prvku UI. Tím je ve třídě generována přepsaná metoda *onClick*. Ta je volána v případě stisku prvku uživatelského rozhraní. V parametru je předán datový typ *View*, který prvek reprezentuje.

Druhým rozhraním je *NavigationView.OnNavigationItemSelectedListener*. Díky tomu je možné odchytit akce na navigačních položkách. To v tomto případě znamená, že funkce *onNavigationItemSelected*, která je rozhraním implementována, je volána při stisku položky menu. Tím se lze přesměrovat na příslušnou obrazovku. Funkce v parametru předává stisknutou položku navigačního menu.

4.5.3 Zobrazení jednotlivých fragmentů

Jak bylo zmíněno, navigace mezi hlavními fragmenty aplikace je realizována v metodě *onNavigationItemSelected*. Samotné přepínání obluhuje následující funkce.

```
1  private void showFragment(Class<? extends Fragment> fragmentClass) {
2    FragmentManager fm = getSupportFragmentManager();
3    fm.beginTransaction().replace(R.id.content, Fragment.instantiate(
         this, fragmentClass.getName())).commit();
4  }
```

Funkci je předána třída, která rozšiřuje třídu *Fragment*. V jejím těle je potřeba získat *FragmentManager*, který slouží pro komunikaci s objekty typu *Fragment* uvnitř aktivity. Poté je možné zavolat transakci, která nahradí obsah okna novou instancí předaného fragmentu.

Před přepnutím obsahu je však nezbytné zjistit, jaká položka menu byla stlačena.

```
1  int id = item.getItemId();
2
3  switch(id) {
4    case R.id.nav_search:
5      showFragment(SearchFragment.class);
6      break;
```

```
7       ...
8   }
9   mBinding.drawerLayout.closeDrawer(GravityCompat.START);
```

Poté lze vhodnou podmínkovou konstrukcí přepínání docílit. Pro vizuální efekt je na závěr nutné menu zavřít, což zajišťuje poslední řádek ukázky.

4.6 Vyhledávání v uložených datech

4.6.1 Grafické rozvržení obrazovky

Obrazovka fragmentu obsahuje kořenový prvek *FrameLayout*. V něm jsou vnořeny dva elementy. Prvním z nich je *RecyclerView*. Ten slouží jako meziúroveň abstrakce mezi daty a manažerem, který se stará o vykreslení grafického rozvržení. Díky němu je možné detekovat hromadné změny v datech v průběhu vykreslování layoutu a zobrazit je například jako seznam. To má kladný vliv na výkon celé aplikace. Párování dat do layoutu je prováděno ve stejnou dobu jako jeho vykreslení, a tak jsou nepotřebné kroky vynechány.

Druhým prvkem je *TextView*, což je obyčejné textové pole. To má v tomto případě za úkol zobrazit uživateli informace, jak fragment využívat.

Viditelnost obou prvků je vzájemně inverzní – je-li první vidět, druhý je skryt a naopak. Vše záleží na tom, zda byla při vyhledávání nějaká data nalezena. Rozvržení je také patrné z obrázku 4.4. Pro účely vysvětlení jsou oba prvky zobrazeny najednou.

Posledním prvkem, který je však vytvářen ve třídě aktivity, je *SearchView*. To poskytuje textové pole pro datový vstup při hledání v DB.

Pro použití elementu *RecyclerView* je nutný jeho import ve formě externí závislosti.

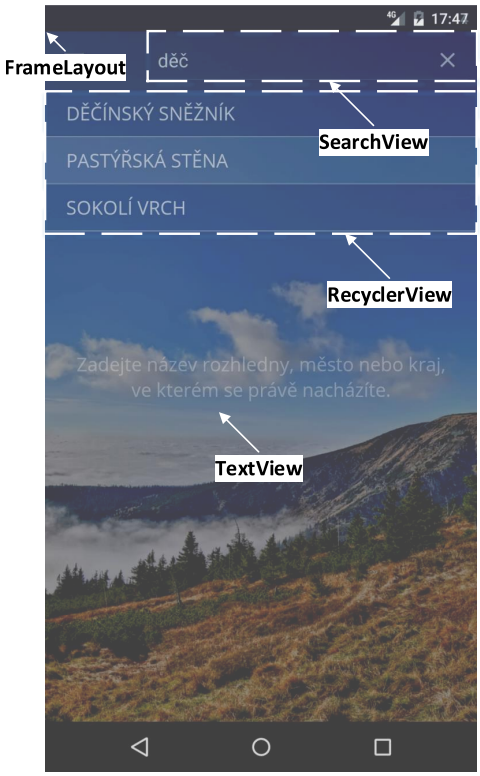
```
1  dependencies {
2    compile "com.android.support:recyclerview-v7:23.3.0"
3  }
```

4.6.2 Rozšíření funkčních možností třídy

SearchFragment dědí od třídy *Fragment*. Ta slouží jako prostředek pro definici chování části uživatelského rozhraní, která je vložena v aktivitě. Je možné kombinovat více fragmentů v jedné aktivitě, a tak lze vytvořit například rozhraní s více záložkami. Fragmenty lze také využít ve více aktivitách. Dá se na ně pohlížet jako na modulární část aktivity, která má svůj vlastní životní cyklus a zpracovává vlastní vstupní požadavky. Je možné je přidat a odebrat přímo za běhu aktivity.

Druhým rozšířením třídy je rozhraní *RecyclerViewBindings.ClickHandler<E>*. *RecyclerViewBindings* je naprogramovaná třída, obsahující rozhraní a další statické funkce, usnadňující práci s prvkem *RecyclerView* pomocí pokročilých možností databindingu.

Jelikož třída podporuje genericitu, lze na ni aplikovat různé modely dat, čehož bude využíváno. Třída obsahuje tři statické funkce.



Obrázek 4.4: Grafické rozvržení SearchFragment

```
1  @SuppressWarnings("unchecked") @BindingAdapter("items")
2  public static <T> void setItems(RecyclerView recyclerView,
       Collection<T> items) {...}
3  @SuppressWarnings("unchecked") @BindingAdapter("clickHandler")
4  public static <T> void setHandler(RecyclerView recyclerView,
       ClickHandler<T> handler) {...}
5  @SuppressWarnings("unchecked") @BindingAdapter("itemViewBinder")
6  public static <T> void setItemViewBinder(RecyclerView recyclerView,
       ItemBinder<T> itemViewMapper) {...}
```

Anotace *@SuppressWarnings("unchecked")* oznamuje kompilátoru, že prováděná akce je legální a v době vykonávání platná. Někdy totiž nastane případ, že kompilátor Javy nenechá programátora udělat to, co chce. V takové chvíli je nutné použít tuto anotaci.

Anotace *@BindingAdapter(...)* umožnuje využít funkci, na kterou je tato anotace aplikována, jako atribut v XML layoutu u elementu, který je v parametrech uveden jako první. Pro lepší pochopení je uvedena část schématu, které vysvětlovanou problematiku zahrnuje.

```
1  ...
2  <android.support.v7.widget.RecyclerView
3      ...
```

```
4     app:items="@{fragment.data}"
5     app:clickHandler="@{fragment}"
6     app:itemViewBinder="@{fragment.itemViewBinder}"
7     ...
8   />
9   ...
```

Zde je již souvislost patrná. První funkce slouží pro naplnění dat do adaptéru. Druhá slouží k navázání funkčnosti při stisku prvku ze seznamu a třetí pro provázání každé položky v seznamu s jejím grafickým rozvržením. V případě tohoto fragmentu je genericita zastoupena datovým typem *Lot* (popsán v části 3.5.3).

4.6.3 Vyhledání a zobrazení dat

Akce, související s hledáním a zobrazováním dat, se budou týkat datového typu *Lot* – data jsou tedy vybírána z databáze rozhleden, na základě stanovených vlastností. Po obvyklém způsobu párování dat fragmentu s layoutem je samotné vyhledávání realizováno následovně.

```
1  searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener
       () {
2     ...
3     @Override
4     public boolean onQueryTextChange(String newText) {
5       mData.clear();
6
7       if(!newText.equals("")) {
8         mData.addAll(getDatabase().select(Lot.class)
9           .distinct()
10          .where(LotEntity.NAME.like("%"+newText+"%"))
11          .or(LotEntity.CITY.like("%"+newText+"%"))
12          .or(LotEntity.COUNTY.like("%"+newText+"%")).get().toList());
13      }
14
15      if(mData.size() > 0) {
16        mInfoVisible.set(false);
17      }else{
18        mInfoVisible.set(true);
19      }
20      ...
21    }
22 });
```

Proměnné, reprezentující vyhledávací pole, je přiřazen *OnQueryTextListener*. Tím je započato sledování změn v textu. Metoda *onQueryTextChange* je volána při každé změně vyhledávacího řetězce. V prvním kroku jsou všechna data v seznamu (proměnná *mData*) odstraněna. Je-li textový řetězec neprázdný, je vykonán SQL dotaz.

Ten z entity uložených rozhleden vrací hodnoty pouze jednou – dle názvu, města nebo kraje. Jsou vybrány takové hodnoty, kde je vyhledávací řetězec jejich součástí. Výsledek je vrácen do seznamu. Proměnná *mData* je tímto seznamem naplněna, což způsobí detekci změny v datech. Neobsahuje-li proměnná žádná data, je zobrazen informační text a skryt prázdný seznam, jinak nastane situace opačná.

Díky databindingu je zobrazení dat provedeno automaticky. Detekovaná změna páruje nová data za pomoci metody *getData()*, který proměnnou *mData* vrací. Metoda je s layoutem spárována atributem *app:items*. Tím je aktualizován adaptér s daty, což způsobí jejich zobrazení.

Grafické rozvržení řádku dat je definováno souborem *item_search.xml*, který se nachází ve složce se všemi layouty. Párování vzhledu je řízeno v XML, a to atributem *tools:listitem*.

4.6.4 Odkaz na detail výsledku hledání

Po výběru vyhledané položky je uživatel přesměrován do detailu rozhledny (sekce 4.9). Tuto akci zajišťuje metoda *onClick*, která je součástí implementovaného rozhraní.

```
1 @Override
2 public void onClick(Lot viewModel) {
3   Intent i = new Intent(getContext(), LotDetailActivity.class);
4   i.putExtra(LotDetailActivity.M_LOT_OBJECT, viewModel);
5   startActivity(i);
6 }
```

Metoda přijímá objekt *Lot*, v němž jsou uloženy všechny informace o vybrané rozhledně. Objekt je při instancování aktivity předán parametrem. V aktivitě je ošetřeno jeho korektní přijetí.

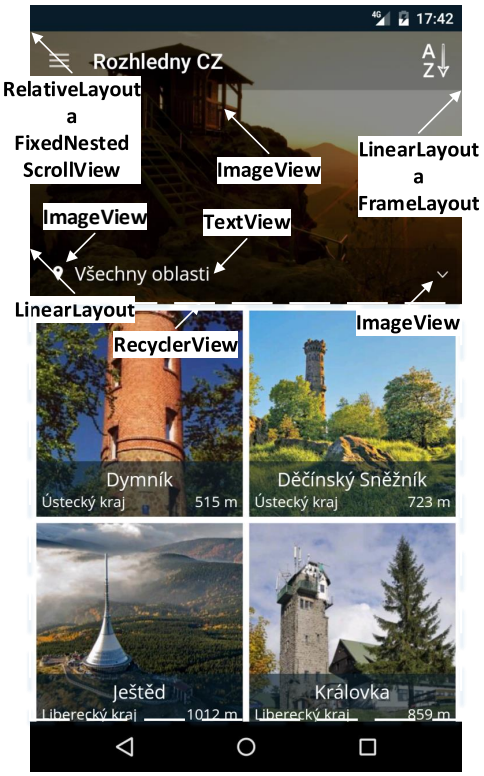
4.7 Prohlížení evidovaných rozhleden

4.7.1 Grafické rozvržení obrazovky

Grafika fragmentu *LotsFragment* je umístěna v kořenovém prvku *RelativeLayout*. Ten se využívá pro zobrazení grafických elementů, jejichž pozici lze specifikovat relativně k dalším prvkům stejné úrovně. Druhou možností je určení umístění ve vztahu ke kořenovému prvku layoutu. V případě, že programátor použije více vnořených prvků *LinearLayout*, je pro navýšení výkonu vhodné je nahradit jediným elementem *RelativeLayout*.

V něm je vnořen *FixedNestedScrollView*. Jedná se o třídu, která dědí od *NestedScrollView* a ošetřuje chybu, která nastávala při prohlížení dat [14]. Třída rodiče slouží jako obal obsahu, který je fyzicky větší než displej. Celý posuvný obsah je v prvku *LinearLayout*. Ten obsahuje další elementy, které jsou pro přehlednost uvedeny pouze na obrázku 4.5.

V prostřední části obrazovky se nachází dialog pro výběr filtrační oblasti. Filtrovat rozhledny je možné dle krajů České republiky nebo vzdálenosti od polohy uživatele.



Obrázek 4.5: Grafické rozvržení LotsFragment

4.7.2 Rozšíření funkčních možností třídy

Třída implementuje tři rozhraní. Prvním z nich je *View.OnClickListener*. Možnosti rozhraní jsou využity při stisku prvku pro filtraci dat.

Druhým rozhraním je *RecyclerViewBindings.ClickHandler<E>*. Generickým typem je stejně jako v případě fragmentu pro vyhledávání v datech datový typ *Lot*. Důvod je stejný – ve fragmentu jsou zobrazena data o rozhlednách, a proto ho je třeba použít, aby bylo možné s načtenými informacemi pracovat.

Posledním použitým rozhraním je *LocationListener*. To je použito pro příjem polohy od *LocationManager*u v případě její změny. Získaná poloha je využita při třídění rozhleden dle vzdálenosti.

4.7.3 Načtení a zobrazení dat

Po instancování fragmentu postupem popsaným v 4.5.3 jsou také načtena data s rozhlednami. Propojení se šablonou je realizováno opět pomocí databindingu tak, jako tomu bylo v předchozích případech. Načítání dat obsluhuje metoda *getData()*. V ukázce kódu jsou zobrazeny stěžejní kroky.

```
1 switch (mCountyId) {
```

```
2       ...
3       ALL_REGIONS_SELECTED:
4           if(mSortASC.get()) {
5               mData.addAll(getDatabase().select(Lot.class).orderBy(LotEntity
                    .NAME.asc()).get().toList());
6           } else {
7               mData.addAll(getDatabase().select(Lot.class).orderBy(LotEntity
                    .NAME.desc()).get().toList());
8           }
9           ...
10      ...
11  }
```

Lze si všimnout, že metoda pro získání dat je připravena obsloužit více variant, rozlišovaných proměnnou *mCounty*. Ta indikuje výběr všech oblastí, kraje nebo třídění dle polohy uživatele. Tyto varianty jsou vysvětleny dále.

Dalším prvkem fragmentu je proměnná *mSortASC*, která rozhoduje, zda budou data tříděna abecedně sestupně či vzestupně. Na základě její hodnoty je do proměnné *mData* plněn seznam rozhleden. Ten je získán SQL dotazem.

Z obrázku 4.5 lze vidět, že každá položka seznamu s rozhlednami (*RecyclerView*) obsahuje kromě ilustrace rozhledny i další informace. Implementace je možná díky vlastnímu stylu každé položky. Styl lze vytvořit způsobem, zmíněným v části 4.6.2.

Nyní nastala vhodná chvíle pro vysvětlení jeho propojení. V prvním kroku je v *RecyclerView* definován atribut.

```
1  <android.support.v7.widget.RecyclerView
2      ...
3      tools:listitem="@layout/item_lot"/>
```

Dále je třeba, aby byl soubor s odkazovaným vytvořen. Tím je zprostředkován cílový styl, znázorněný obrázkem 4.6.

Obrázek 4.6: Grafické rozvržení položky v přehledu rozhleden

Po nastylování jsou využity možnosti databindingu pro spárování dat s prvky UI. Je vytvořen tag *<data>*, v kterém je umístěna proměnná *lot*. Ta bude použita jako zdroj informací. Zmíněný postup funguje díky tomu, že jsou data plněna ze struktury *List<Lot>*.

```
1  ...
2  <data>
3  <variable
4    name="lot"
5    type="cz.tul.dp.vaclavek.david.rozhlednycz.data_objects.requery.
         Lot"/>
6  />
7  ...
```

Pro napárování dat je poté třeba pouze určit zdroj (vlastnost proměnné *lot*) u atributu UI elementů.

```
1  ...
2  <ImageView
3    ...
4    app:imageUrl="@{lot.main_image}"/>
5  ...
6  <TextView
7    ...
8    android:text='@{lot.county, default="#lot.county"}'/>
9  ...
```

Lze zaregistrovat, že pro vykreslení spárovaného obrázku byla vytvořena funkce pomocí anotace *@BindingAdapter* obdobným způsobem, jaký byl uveden v 4.6.2. Tato funkce je implementována ve třídě *DataBinder*. Vykreslení obrázku probíhá postupem, uvedeným v pododdílu 3.6.3.

4.7.4 Odkaz na detail rozhledny

Jak bylo uvedeno výše, pro přepnutí je využita funkce rozhraní. Samotný kód je téměř identický s kódem, který byl pro přepnutí do detailu použit ve fragmentu 4.6.

```
1  if(ImageLoader.getInstance() != null && ImageLoader.getInstance().
       isInited()) {
2    ImageLoader.getInstance().stop();
3  }
4  ...
5  startActivity(i);
```

Jedinou akcí, kterou je třeba provést navíc, je pozastavení načítání obrázků knihovny UIL. To je provedeno na druhém řádku ukázky za předpokladu, že je knihovna inicializována. Pokud by tento krok proveden nebyl, aplikace skončí chybou.

4.8 Operace se zobrazenými daty rozhleden

4.8.1 Abecední třídění dat

Prvním způsobem, jak lze se zobrazenými rozhlednami pracovat, je jejich abecední třídění. Ve výchozím stavu jsou rozhledny seřazeny sestupně. K přepínání mezi nimi slouží tlačítko, umístěné v pravém horním rohu obrazovky. Jelikož je tlačítko v záhlaví fragmentu, je definováno v XML souboru ve složce *menu*. Pro práci s ním je implementována přepsaná metoda.

```
1  private ObservableField<Boolean> mSortASC = new ObservableField<>(
       true);
2  ...
3  @Override
4  public boolean onOptionsItemSelected(MenuItem item) {
5    if(mSortASC.get()) {
6      mSortASC.set(false);
7      item.setIcon(getResources().getDrawable(R.drawable.ico_desc));
8      ...
9    } else {
10     mSortASC.set(true);
11     item.setIcon(getResources().getDrawable(R.drawable.ico_asc));
12     ...
13   }
14   mData.clear();
15   getData();
16   ...
17 }
```

Metoda obsahuje parametr typu *MenuItem*, v kterém je předána stisknutá položka. Jelikož se střídají pouze dva stavy, je jisté, že vždy bude přepnuto právě na ten druhý. Po přepnutí je ještě změněna ikona tlačítka, aby bylo zřejmé, jaký typ řazení je aktivní.

Nakonec je nutné seznam s daty vyprázdnit a naplnit ho znovu. To se děje v těle funkce *getData()*, což je popsáno v části 4.7.3.

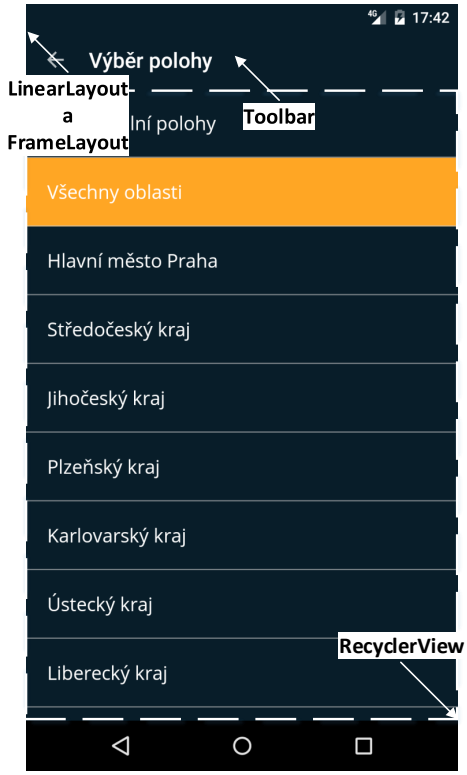
Je důležité zmínit, že proměnná *mSortASC* je datového typu *ObservableField<Boolean>*. Ten pracuje obdobně jako jeho primitivní verze, pouze s rozdílem, že je využíván při databindingu, a tak lze okamžitě aplikovat změnu jeho hodnoty i do UI.

4.8.2 Práce s dialogem výběru oblasti dat

Po užití tlačítka pro filtraci je uživatel přesměrován do nové aktivity. Jelikož se jedná o obousměrnou komunikaci, je aktivita *CountySelectActivity* volána příkazem *startActivityForResult()*.

```
1 Intent i = new Intent(getContext(), CountySelectActivity.class);
2 i.putExtra(CountySelectActivity.M_COUNTY_ID, mCountyId);
3 startActivityForResult(i, REQUEST_CODE);
```

Do aktivity je odeslán identifikátor vybrané oblasti filtrace, která díky tomu může být graficky zvýrazněna. Po instancování je vykreslena obrazovka. Seznam oblastí je opět tvořen prvkem *RecyclerView*. Layout je patrný z obrázku 4.7.



Obrázek 4.7: Grafické rozvržení CountySelectActivity

Data jsou získána z entity *County* a v aktivitě zobrazena pomocí databindingu. Pro data je ve třídě aktivity vytvořen statický model *ItemViewModel*, který uchovává informace o kraji a příznak, zda je pole aktivní, a je tedy podbarveno.

Po výběru je název a identifikátor vybrané oblasti odeslán zpět do fragmentu s přehledem rozhleden. Nakonec je volán příkaz *finish()* – ze stejného důvodu, jaký je uveden v podsekci 4.3.3.

```
1  @Override
2  public void onClick(ItemViewModel viewModel) {
3      Intent i = new Intent();
4      i.putExtra(M_COUNTY_ID, viewModel.getCounty().getId());
5      i.putExtra(M_COUNTY_NAME, viewModel.getCounty().getName());
6      setResult(Activity.RESULT_OK, i);
7      finish();
8  }
```

Příjem dat z aktivity probíhá v metodě *onActivityResult* způsobem, uvedeným v následující ukázce kódu.

```
1  @Override
2  public void onActivityResult(int requestCode, int resultCode, Intent
       data) {
3    ...
4    mCountyId = data.getExtras().getInt(CountySelectActivity.
       M_COUNTY_ID);
5    ...
6    getData();
7  }
```

Data jsou získána na základě stejného klíče, jako byla odeslána. Identifikátor oblasti je uložen v proměnné *mCountyId*. Nakonec je volána metoda pro aktualizaci dat.

4.8.3 Filtrace dat dle krajů

Po získání indexu oblasti filtrace a jeho uložení do proměnné je volána funkce pro aktualizaci dat. Nabývá-li proměnná jedné z hodnot, připadajících indexu některého z krajů, vstoupí aplikace při výběru dat do větve *default*, kde je proveden podobný dotaz.

```
1  mData.addAll(getDatabase()
2    .select(Lot.class)
3    .where(LotEntity.COUNTY.eq(mCountyName.get()))
4    .orderby(LotEntity.NAME.asc()).get().toList());
```

Na základě názvu kraje a typu abecedního řazení jsou z databáze vybrána data, která jsou vložena do vyprázdněného seznamu. Ten je v uživatelském rozhraní automaticky vykreslen.

4.8.4 Získání polohy uživatele

Pro periodickou aktualizaci polohy uživatele zařízení je třeba provést několik kroků. Prvním z nich je již zmíněná implementace rozhraní *LocationListener*. Druhým krokem je vytvoření instance třídy *LocationManager*, které jsou následně předány parametry pro odposlech polohy.

```
1  private LocationManager mLocationManager;
2  ...
3  mLocationManager = (LocationManager) getActivity().getSystemService(
       Context.LOCATION_SERVICE);
4  ...
5  mLocationManager.requestLocationUpdates(
6    LocationSettings.LOCATION_PROVIDER,
7    LocationSettings.LOCATION_MIN_TIME_CHECK_PERIOD,
8    LocationSettings.LOCATION_MIN_DISTANCE_CHECK_PERIOD,
9    this);
10 ...
```

Prvním předaným parametrem je typ poskytovatele služeb, kde je možné zvolit výběr mezi GPS a internetovou sítí. Dalším je časová perioda pro získání polohy a pak vzdálenost, po které bude poloha znovu získána. Posledním parametrem je instance třídy s rozhraním *LocationListener* – což je v tomto případě tento fragment.

Je dobré si hodnoty prvních tří parametrů promyslet, protože mají vliv na vybíjení baterie zařízení. Čím menší frekvence aktualizací, tím menší přesnost – je třeba zvážit požadavky a zvolit kompromis.

Pro zpřístupnění služby získávání polohy je nutné aplikaci povolit využívání polohových služeb, dostupných v zařízení. Toho je docíleno přidáním následujících oprávnění do souboru *AndroidManitest.xml*.

```
1 <uses-permission android:name="android.permission.
      ACCESS_NETWORK_STATE" />
2 <uses-permission android:name="android.permission.
      ACCESS_COARSE_LOCATION" />
3 <uses-permission android:name="android.permission.
      ACCESS_FINE_LOCATION" />
```

Poloha uživatele je získána v metodě *onLocationChanged*. Je provedeno ověření, že nová získaná poloha je přesnější, než ta předchozí. K tomu byla využita statická funkce [9] ve třídě *LocationChecker*. Bylo-li vyhodnoceno, že poloha přesnější je, tak je údaj o ní aktualizován, v opačném případě je stávající poloha ponechána.

4.8.5 Filtrace dat dle polohy uživatele

Byla-li v dialogu výběru oblasti filtrace zvolena možnost řazení rozhleden dle polohy, je ve funkci *getData()* vybrána větev realizující tuto akci. Poté je spuštěno získávání polohy za pomoci kódu, uvedeného v předchozím bodě. Tím je zajištěno periodické volání metody *onLocationChanged*.

Po získání polohy jsou z databáze načtena data o všech rozhlednách, která jsou uložena do pomocné proměnné. Tato data jsou spolu se současnou polohou uživatele předána řadícímu algoritmu heapsort [25], implementovanému ve třídě *HeapSortLots*. Pro náhodně řazená data dosahuje algoritmus rychlých výsledků seřazení, a proto byl vybrán. Průběh algoritmu si lze popsat souborem kroků, zobrazených ve vývojovém diagramu 4.8.

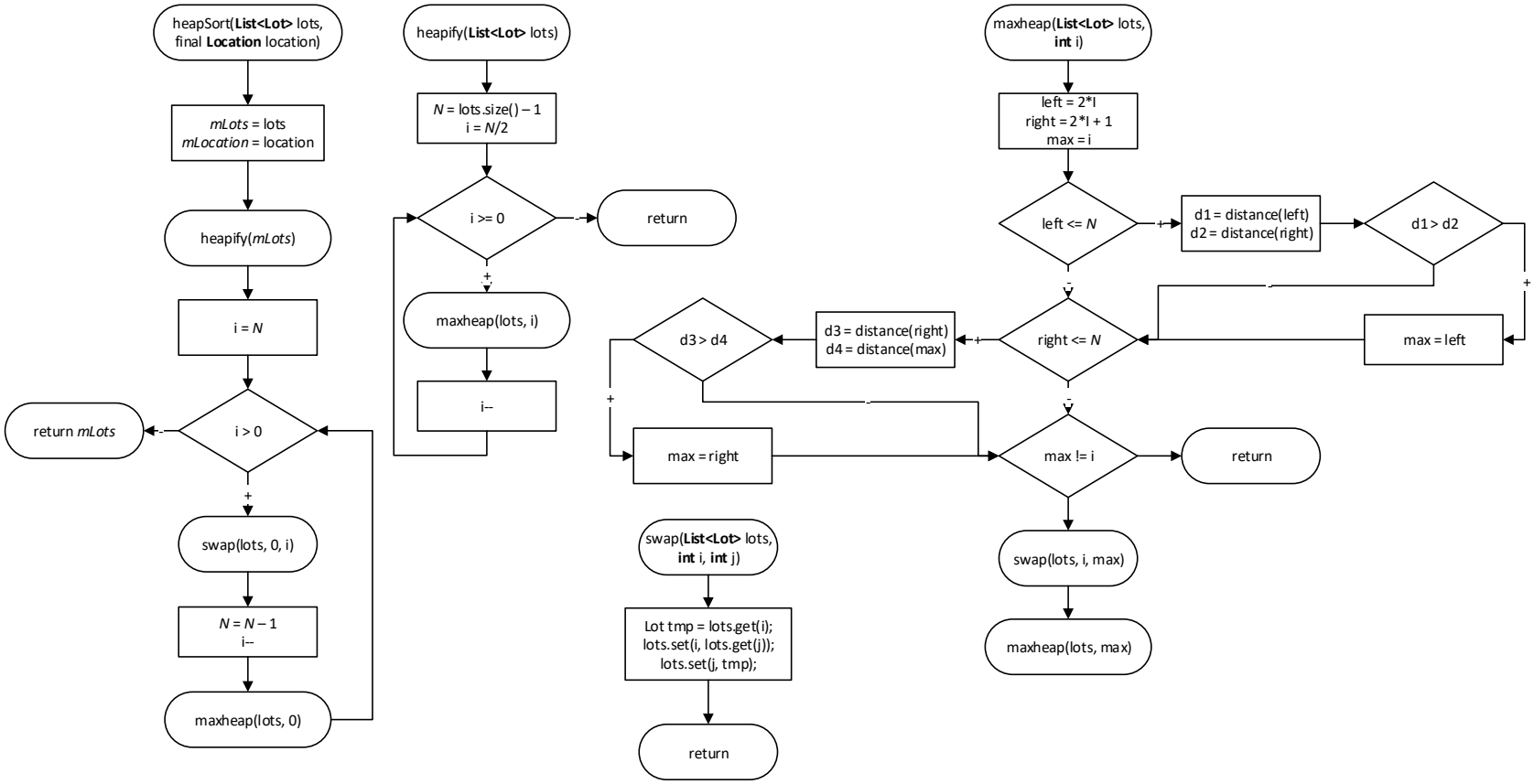
Z něj je vidět, že je v průběhu řazení volána funkce *distance()*. Jejím účelem je výpočet vzdálenosti mezi aktuální polohou a polohou rozhledny pomocí vybraného způsobu, uvedeného v části 1.2.1 (kódově realizován v 3.3.4). Volání funkce pro proměnnou *d1* má následující podobu.

```
1 double d1 = LocationOps.calcHaversineDistance(
2   new GeoCoord(mLocation.getLatitude(), mLocation.getLongitude()),
3   new GeoCoord(lots.get(left).getLatitude(), lots.get(left).
      getLongitude()));
```

Statické funkci jsou předány dva objekty typu *GeoCoord*. Výsledná vzdálenost je do proměnné vrácena a algoritmus řazení haldou pokračuje dále.

Po dokončení je vzestupně setříděný seznam rozhleden předán zpět do fragmentu, kde jsou data aktualizována. Je-li z nějakého důvodu potřeba zastavit aktualizace polohy, je nutné zavolat funkci *removeUpdates*, které je v parametru předán *LocationListener*.

```
1 mLocationManager.removeUpdates(this);
```



Obrázek 4.8: Průběh řazení rozhleden ale vzdálenosti

4.9 Prohlížení detailu rozhledny

4.9.1 Grafické rozvržení obrazovky

Uživatelské rozhraní aktivity je obaleno elementem *CoordinatorLayout*. Tato třída je obsažena v již importované knihovně. Layout se chová podobným způsobem, jako *FrameLayout*. Narozdíl od něj je však možné definovat sofistikovanější úroveň kontroly nad událostmi mezi vnořenými prvky. Ty mohou poté provádět interakce sami mezi sebou.

V něm je vnořen *AppBarLayout*. Ten se chová jako vertikální *LinearLayout*, který navíc implementuje dodatečnou funkcionalitu pro design, použitý v této aktivitě. Dalším vnořeným elementem je *CollapsingToolbarLayout*, který je navrhnut pro používání jako přímý potomek.

Dalším prvkem je klasický *LinearLayout*, v němž jsou umístěna dvě textová pole pro zobrazení úvodních informací o rozhledně. V designu také figuruje třída *CenteredTabLayout*, která slouží jako navigační prostředek v rámci aktivity. Jedná je o třídu, dědící od prvku *TabLayout*. Opravuje problémy, které v navigaci nastávaly.

Posledním prvkem je *ViewPager*. Ten umožní uživateli použít gesto pohybu prstem po obrazovce vlevo nebo vpravo pro navigaci mezi stránkami s informacemi. Pro jeho použití je třeba implementovat nějakou formu třídy *PagerAdaper*, která bude navigaci obsluhovat. Rozložení je graficky znázorněno na obrázku 4.9. Třída nevyužívá žádná rozšíření.

4.9.2 Přepínání mezi záložkami informací

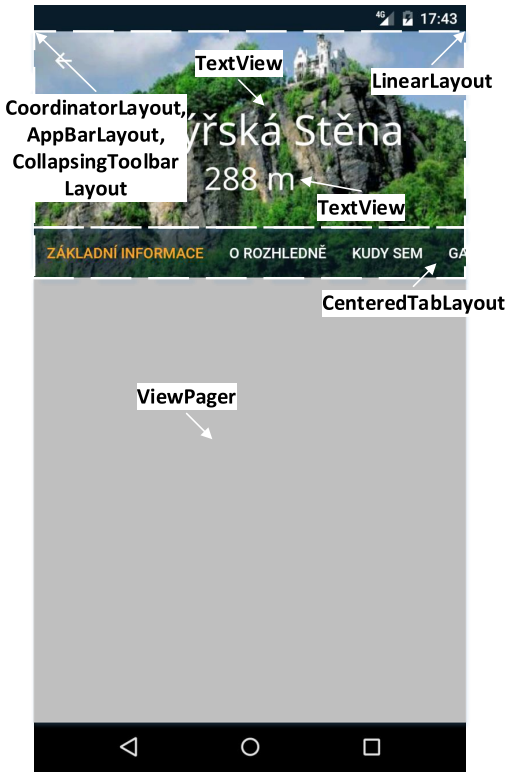
Přepínání mezi panely je realizováno adaptérem. V prvním kroku je nezbytné vytvořit mapu, která bude uchovávat pozice použitých fragmentů.

```
1 private Map<Integer, String> mTabs = new Hashtable<>();
2 ...
3 mTabs.put(LOT_DETAIL_BASIC_INFO_FRAGMENT_INDEX, ...);
4 mTabs.put(LOT_DETAIL_ABOUT_FRAGMENT_INDEX, ...);
5 mTabs.put(LOT_DETAIL_TRIP_FRAGMENT_INDEX, ...);
6 mTabs.put(LOT_DETAIL_GALLERY_FRAGMENT_INDEX, ...);
```

Do mapy je jako klíč předána konstanta, identifikující fragment. Hodnotou je řetězec, definovaný v souboru *string.xml*. V souboru jsou uloženy zdroje, použité pro jazykovou lokalizaci aplikace. Po naplnění mapy je třeba samotný adaptér vytvořit.

```
1 LotDetailPagerAdapter adapter =
2   new LotDetailPagerAdapter(mTabs, mLot, getApplicationContext(),
        getSupportFragmentManager());
3 mBinding.pager.setAdapter(adapter);
4 mBinding.tabs.setupWithViewPager(mBinding.pager);
```

Je vytvořena nová instance třídy *LotDetailPagerAdapter*. V konstruktoru je předána mapa; rozhledna, přijatá při instancování aktivity; aplikační kontext; manažer, zajišťující korektní komunikaci. Adaptér je pomocí databindingu propojen s prvkem *ViewPager*. Ten zajišťuje odchycení akce při provádění gesta na obrazovce. Pager je



Obrázek 4.9: Grafické rozvržení LotDetailActivity

třeba ještě předat v metodě elementu *CenteredTabLayout*, který se stará o vykreslení přepínatelného navigačního menu.

Gesto prstu vyvolá metodu *getItem*, která slouží pro přepínání mezi fragmenty. Metoda je implementována v adaptéru.

```
1  switch (position) {
2    case LOT_DETAIL_BASIC_INFO_FRAGMENT_INDEX:
3      fragment = LotDetailBasicInfoFragment.newInstance(mLot);
4      break;
5    ...
6    case LOT_DETAIL_GALLERY_FRAGMENT_INDEX:
7      fragment = LotDetailGalleryFragment.newInstance(mLot);
8      break;
9  }
10 return fragment;
```

Na základě pozice, předané ve vstupním parametru metody, je rozhodnuto o instancování korektního fragmentu. Tomu je předán objekt s rozhlednou, aby s daty bylo možné dále pracovat.

Jelikož je rozhledna předávána všem instancovaným fragmentům, je vytvořen abstraktní předek *BaseLotDetailFragment*, kde je rozhledna uložena v proměnné. Fragmenty tedy od této třídy dědí.

4.9.3 Záložka „Základní informace“

Fragment, obsahující základní informace o rozhledně, má za účel zobrazit její polohu včetně GPS souřadnic; prudkost stoupání k rozhledně; čistotu výhledu; materiál, ze kterého je vyrobena a webové stránky s dalšími informacemi.

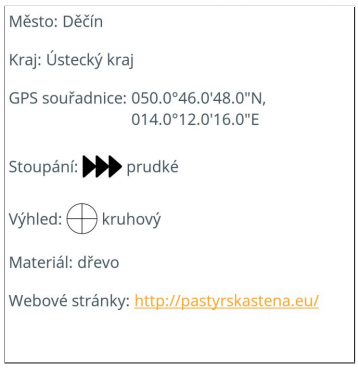
Grafické rozložení je realizováno skupinou prvků *LinearLayout*, umístěných vedle sebe. Každý z nich tvoří jeden informační řádek. Skupina je obalena kořenovým prvkem *FixedNestedScrollView*. Data jsou načítána pomocí databindingu. Dosud nepoužitým příkladem je načtení ikony, symbolizující míru stoupání k rozhledně.

```
1  ...
2  <import type="...DataBinder"/>
3  <variable name="fragment" type="...LotDetailBasicInfoFragment"/>
4  ...
5  <ImageView
6     ...
7     android:text="@{DataBinder.bindGradient(fragment.lot.gradient)}"/>
```

V XML layoutu je importována třída *DataBinder* a také používaný fragment. V příslušném atributu je volána metoda *bindGradient*. Té je předán index, určující stoupání. Na jeho základě je prvku typu *ImageView* nastaven obrázek ze zdrojů, uložených v datech aplikace.

```
1  switch (id) {
2     case 1:
3        imageView.setImageDrawable(context.getResources().getDrawable(R.
             drawable.icon_arrow));
4        break;
5     ...
6     case 3:
7        imageView.setImageDrawable(context.getResources().getDrawable(R.
             drawable.icon_arrow_tripple));
8        break;
9  }
```

Konkrétní příklad vzhledu fragmentu je znázorněn na obrázku 4.10.



Obrázek 4.10: Grafické rozvržení LotDetailBasicInfoFragment

4.9.4 Záložky „O rozhledně“ a „Kudy sem“

Tyto dva fragmenty mají identické grafické rozvržení. Narozdíl od soustavy layoutů, použitých v předchozí záložce, jsou využity možnosti prvku *WebView*. Jedná se o UI element pro zobrazení webových stránek. Bývá základním nástrojem při programování vlastních webových prohlížečů nebo v případě potřeby zobrazení internetového obsahu v rámci aktivity nebo fragmentu. Element používá renderovací engine WebKit, v němž jsou také zahrnuty metody pro navigaci historií vpřed a vzad, zoom a textové vyhledávání.

Data pro tyto záložky jsou uložena formou HTML kódu. Ten obsahuje také formátování a obrázky. Celý obsah je ve *WebView* vykreslen. To v tomto případě neprobíhá pomocí databindingu, protože prvek přístup nepodporuje.

Rendering je realizován standardně pomocí metody, což je ve fragmentu *LotDetailAboutFragment* provedeno následujícím kódem.

```
1 mBinding.content.loadData(getmLot().getAbout(), "text/html; charset=
      utf-8", "UTF-8");
```

Grafický prvek je získán. Poté je volána jeho metoda *loadData*, která umožní vykreslení HTML kódu. Ten je vrácen metodou *getAbout*. Dalšími parametry jsou definovány hlavička a kódování znaků.

4.9.5 Záložka „Galerie“

Grafickým návrhem je záložka, kterou reprezentuje fragment *LotDetailGalleryFragment* prakticky identická, jako část fragmentu s přehledem rozhleden, která zobrazuje jejich seznam. Rozdílem při vykreslení je použití datového modelu *Gallery*. Tím je také dokázána síla generického typování v Javě. Není třeba zbytečně připravovat téměř identické funkce pro podobné účely, pouze s jiným modelem dat.

Postup při použití je praktický stejný. Z toho plyne fakt, že galerie rozhledny je vykreslena do prvku *RecyclerView*, definovaného v layoutu. Fragment implementuje rozhraní *RecyclerViewBindings.ClickHandler<Gallery>*. Díky tomu může být uživatel přesměrován na zobrazení položky galerie přes celou obrazovku.

Po instancování fragmentu je třeba z databáze získat data o položkách, přiřazených k rozhledně. K tomu slouží následující kód.

```
1 mData = getDatabase()
2     .select(Gallery.class)
3     .where(GalleryEntity.LOT_ID.equal(super.getmLot().getId()))
4     .get()
5     .toList();
```

Příkazem jsou v datech vybrány všechny údaje z entity *Gallery*, pro které platí, že *ID* je rovno identifikátoru rozhledny.

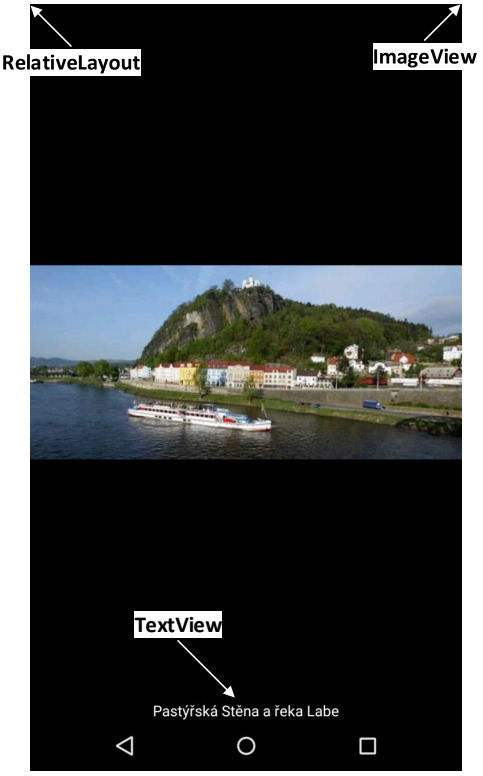
4.9.6 Zobrazení položky galerie na celou obrazovku

Po výběru jednoho z obrázků je volána funkce *onClick*. V parametru funkce je předána proměnná *viewModel*, jejíž datový typ je *Gallery*. Pro zobrazení přes

celou obrazovku je instancován fragment *ImageDetailFragment*, který v konstruktoru proměnnou *viewModel* přebírá.

UI fragmentu je složeno z *ImageView* a *TextView*. Prvky jsou uzavřeny v kořenovém elementu *RelativeLayout*. Obrázek je propojen pomocí databindingu. V textovém poli v dolní části obrazovky je zobrazen popisek. Grafické rozložení je čitelné také z obrázku 4.11. Při vytváření instance fragmentu je vybrán styl, zajišťující zobrazení fullscreen. Způsob nastavení je v ukázce kódu.

```
1 setStyle(DialogFragment.STYLE_NORMAL, android.R.style.
      Theme_Black_NoTitleBar_Fullscreen);
```



Obrázek 4.11: Grafické rozvržení ImageDetailFragment

4.10 Kalibrace čidel zařízení

4.10.1 Důvod kalibrace čidel

Při používání čidel zařízení je někdy jejich přesnost snížena. To je dáno působením vnějších magnetických sil, které výstupní data ovlivňují. Pro představu situace v kontextu s vyvíjenou aplikací je uveden příklad: uživatel vyleze na rozhlednu, která je vyrobena z vodivého materiálu. Ten má nepochybně své vlastní magnetické pole

– je tedy možné, že výstupní data z čidel zařízení mohou být polem ovlivněny. Tyto chyby jsou zavlečeny do algoritmů a panorama může být vůči realitě posunuto.

Proto je v třeba čidla kalibrovat. Z toho důvodu je vytvořen fragment, který uživateli ukazuje způsob provedení kalibrace – co nejpřesněji a nejrychleji.

Kalibrace se standardní procedura, kterou je třeba na čidlech (v tomto případě především magnetometru) provádět. Příkladem také mohou být v dnešních dnech stále populárnější drony, které také obsahují velké množství čidel. Na jejich správnou funkčnost uživatel dronu samozřejmě spoléhá. V záporném případě se může dron stát nekontrolovatelným.

4.10.2 Grafické rozvržení obrazovky

V kořenovém prvku *LinearLayout* jsou vloženy dva elementy. Prvním je *PlayGifView*. Jedná je o třídu, která dědí od *View* a obsluhuje cyklické přehrávání obrázku ve formátu GIF. Framework tuto funkci nativně nepodporuje. Druhým prvkem je textové pole s instrukcemi pro kalibraci. Vzhled obrazovky je možné vidět na obrázku 4.12. Samotné přehrávání je realizováno následujícím příkazem.

```
1  mMovie = Movie.decodeStream(getResources().openRawResource(id));
```

V proměnné *id* je obsažen odkaz na GIF, který je uložen v datech aplikace ve složce *raw*. Rozměry obrázku jsou třídou vypočítány automaticky.

4.10.3 Průběh kalibrace

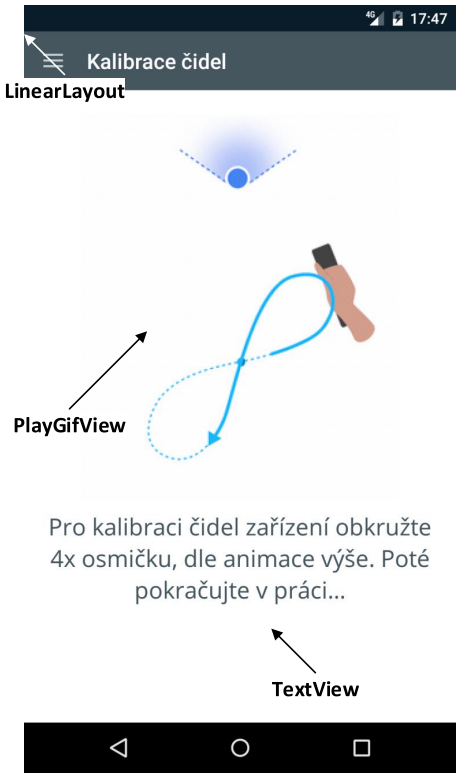
Existuje více způsobů, jak čidla zkalibrovat. Někdy bývá upřednostněn způsob několikanásobného otočení zařízení okolo každé ze tří os. V posledních letech se však prosazuje takzvaný „Figure 8 Pattern“. Jedná se o pohyb ruky, který vytváří číslici osm. Po několikanásobném otočení by měla být čidla opět zkalibrována a uživatel by měl být schopen bez problémů pokračovat v práci.

Zamyslí-li se člověk nad způsobem pohybu, tak se také v podstatě jedná o kroužení kolem tří os. V zápěstí dochází k ohybu, a tak je zařízení vykloněno i ve směru třetí osy.

4.11 Zobrazení informací “O aplikaci”

Fragment slouží jako informační prostředek pro sdělení základních údajů o aplikaci – její verze; za jakým účelem vznikla; kdo je autor aplikace. Tomu také odpovídá UI. Kořenovým prvkem je *ScrollView*. To umožní uživateli posun na obrazovce v případě, že budou informace přesahovat její délku. V tomto elementu je vložen *LinearLayout*, kde je *ImageView* s logem aplikace. Pod ním se vyskytuje několik textových polí, obsahujících informace načtené ze zdroje řetězců.

Principy propojení již byly vysvětleny v průběhu předchozího popisu, nemá tedy příliš smysl je znovu opakovat.



Obrázek 4.12: Grafické rozvržení SensorCalibrationFragment

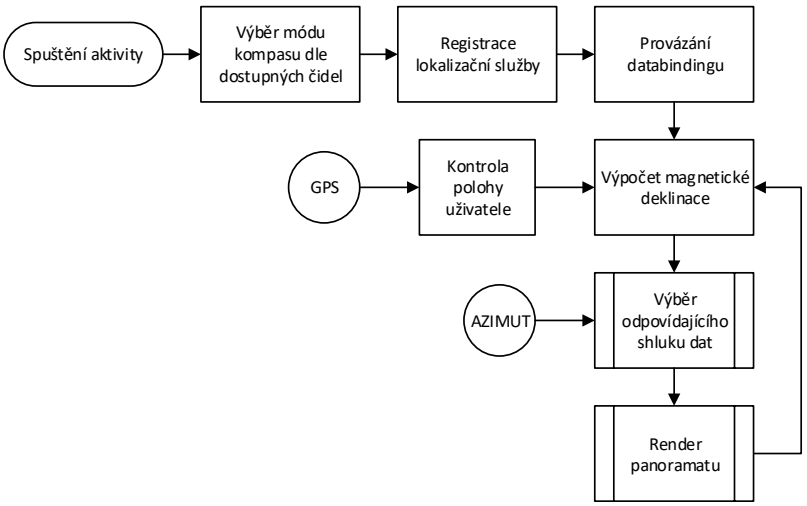
4.12 Zobrazení viditelného 360° panoramatu

4.12.1 Celkový pohled na průběh zobrazení panoramatu

Zobrazení panoramatu rozhledny je jednou ze stěžejních částí práce. Jedná se o sled několika důležitých kroků, jejichž výsledkem je poskytnutí korektních informací uživateli. Tyto kroky jsou zobrazeny v diagramu 4.13.

Při vytváření fragmentu je nejprve třeba zaregistrovat odposlech využívaných čidel. Po vytvoření obrazovky je layout provázán s instancí fragmentu. Poté již nastávají cyklicky se opakující akce. Ty jsou zastaveny v případě, že uživatel ukončil prohlížení panoramatu nebo bylo kontrolou polohy zjištěno, že se nenachází v místě rozhledny.

Nenastane-li jedna z předešlých situací, je za pomoci získaných GPS souřadnic s polohou uživatele (viz 4.8.4) vypočtena magnetická deklinace. Na základě výstupu z kompasu (viz 3.2.3) jsou získána relevantní data panoramatu. To spočívá v prvotní korekci azimutu o hodnotu magnetické deklinace (dále jen orientace), následnému získání dat o panoramatu a výběru odpovídající části z nich. Data jsou na obrazovku vykreslena pomocí periodického přepočtu, která je při pohybech uživatele automaticky aktualizována.



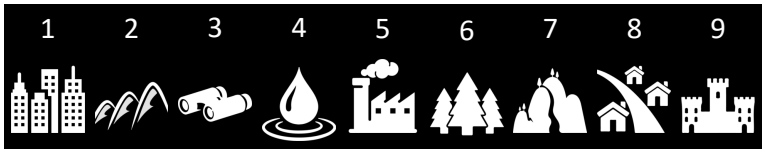
Obrázek 4.13: Celkový pohled na zobrazení panoramatu

4.12.2 Grafické rozvržení obrazovky

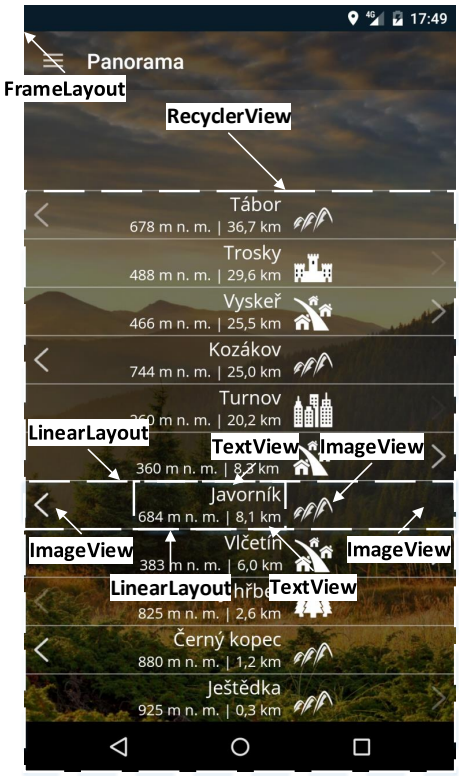
Při návrhu prostředí pro zobrazení panoramatu byl kladen důraz na jeho jednoduchost. V situaci, kdy má být vykresleno velké množství informací najednou, musí být uživatel informován jasně a přehledně. UI elementy jsou obaleny prvkem *FrameLayout*. V něm se kromě *RecyclerView* (seznam míst) vyskytuje také *TextView*, které je zobrazeno pouze v případě, že se uživatel aplikace v místě rozhledny nenachází (tuto informaci mu sděluje).

Každý prvek seznamu má definován vzhled – opět v externím souboru. Ten je složen z lineárního layoutu, kde jsou vnořeny dva elementy typu *ImageView*. Ty reprezentují šipky, jejichž hodnota průhlednosti je závislá na směru a velikosti odchylky aktuálního orientace od orientace skutečné.

Uprostřed je vložen další *LinearView*, který obsahuje základní fakta o místě, jako je název, nadmořská výška a aktuální vzdálenost uživatele od něj. Posledním prvkem je *ImageView*. Jeho účelem je informovat uživatele o typu bodu výhledu. V aplikaci je registrováno devět typů, které jsou ve uvedeny v obrázku 4.14. Celé grafické rozvržení je zobrazeno na obrázku 4.15.



Obrázek 4.14: Přehled typů bodů panoramatu



Obrázek 4.15: Grafické rozvržení PanoramaFragmentV1

Význam jednotlivých ikonek je následující:

1. město,
2. hora,
3. rozhledna,
4. vodní plocha,
5. průmyslový objekt,
6. les,
7. přírodní památka,
8. vesnice,
9. hrad.

4.12.3 Rozšíření funkčních možností třídy

Třída implementuje dvě rozhraní. Prvním z nich je *SensorEventListener*, které je využito pro implementaci kompasu. Druhým rozhraním je *LocationListener*. To je použito pro získání polohy uživatele. Poloha je aktualizována pouze za podmínek, které popisuje funkce [9].

Ukázky stěžejních částí kódu již byly uvedeny v kapitolách 4.8.4, respektive 3.2.3. Opět platí skutečnost, že odposlech sensorů je ukončen v případě pozastavení či ukončení životního cyklu fragmentu.

4.12.4 Získání dat o panoramatu

Při získávání dat panoramatu je nejprve provedena kontrola polohy uživatele v bezprostředním okolí nějaké z evidovaných rozhleden. Funkce *isLotPosition*, která kontrolu obstarává, se nachází ve třídě *PanoramaFragmentController*. Účelem třídy je oddělit vykreslení dat od jejich přípravy a nezbytných doplňujících výpočtů. Třída proto navenek komunikuje pomocí implementovaného rozhraní *IPanoramaFragmentController*.

```
1  private static double LOT_TOLERANCE_RADIUS = 100; //[m]
2  ...
3  if(mCurrentLot != null) {
4    distance = LocationOps.calcHaversineDistance(...);
5    if(distance != null && distance <= LOT_TOLERANCE_RADIUS) {
6      mLocation = location;
7      mDeclination = mWmmOps.getDeclination(...)[0];
8      ...
9      return true;
10   }
11 }
12 ...
```

V první kroku je stavem proměnné *mCurrentLot* zjištěno, zda uživatel na nějaké rozhledně stojí. Pokud ano, je aktuální vzdálenost přepočítána. Vzdálenost je třeba porovnat s hodnotou tolerance, která je stanovena na radius 100 metrů od rozhledny. Je-li poloha uživatele v rámci meze, je aktualizována hodnota magnetické deklinace pro současnou polohu. Doplňující údaje v panoramatu jsou také aktualizovány. Následně je vrácen příznak, indikující že uživatel stojí na rozhledně.

```
1  ...
2  for(Lot lot : lots) {
3    distance = LocationOps.calcHaversineDistance(...);
4    if(distance != null && distance <= LOT_TOLERANCE_RADIUS) {
5      mCurrentLot = lot;
6      setPanorama();
7      ...
8      return true;
9    }
10 }
11 ...
12 return false;
```

Nebyl-li uživatel na žádné z rozhleden prozatím lokalizován, je projita jejich databáze. Pro každou položku je vypočítána vzdálenost od současné polohy. Za splnění podmínek je přiřazena nová rozhledna a je nastaveno panorama (funkce *setPanorama()*). Nejsou-li podmínky splněny, jsou hodnoty odpovídajících proměnných nastaveny do inicializačního stavu.

Pro získání dat panoramatu je použita funkce API *getView*, která je volána v *AsyncTask*. Ukázka volání funkce API je uvedena v pododdílu 4.3.3. Data jsou při jejich úspěšném získání mapována na objekty.

```
1  private List<OPanoramaPoint> mPanorama = new ArrayList<>();
2  ...
3  for(OViewPoint point : response.body().getData().getView_points()) {
4    mPanorama.add(new OPanoramaPoint(point));
5  }
6  ...
```

Podoba panoramatu je uchována v proměnné *mPanorama*. Seznam je plněn ve smyčce. Získaný objekt typu *List<OViewPoint>* je procházen a jeho položky mapovány na objekty typu *OPanoramaPoint*. Toto mapování je aplikováno z důvodu nutnosti výpočtu dalších informací, které je třeba nastavit až v dalších fázích.

4.12.5 Získání relevantního výřezu dat z panoramatu

Po vytvoření seznamu bodů panoramatu je třeba dopočítat zbylé hodnoty, které jsou závislé až na konkrétní poloze a uložit je do vlastností každého objektu ze seznamu. Jelikož je k datům přistupováno při každém zjištění hodnot některého ze zaregistrovaných čidel, je alternativa výpočtu těchto údajů při každém přístupu nesmyslná.

Pro doplnění hodnot slouží statická metoda třídy *OPanoramaController*. V prvním kroku je každému bodu nastaven azimut. To je provedeno pomocí funkce *calcBearing* (viz 3.3.4). Dalším krokem je nastavení skutečné orientace (azimut, upravený o hodnotu magnetické deklinace). Jedná se o jednoduché odečtení hodnoty deklinace od azimutu. Posledním krokem je vzestupné setřídění hodnot dle orientace. Třídění opět probíhá pomocí algoritmu heap sort (viz 4.8.5), který je pouze aplikován na jinou datovou strukturu.

Setříděné body jsou následně rozděleny do clusterů. Clustery slouží pro zvýšení přehlednosti při zobrazení panoramatu. Na základě pokusů bylo rozhodnuto, že nejoptimálnější bude rozdělit data do devíti clusterů. Každý cluster pokrývá pásmo $360° \div 9 = 40°$. Clustery jsou uchovány v seznamu seznamů *OPanoramaPoint*.

Po rozdělení jsou body každého clusteru setříděny sestupně dle vzdálenosti od aktuální polohy uživatele. To umožní zobrazit informace přehledněji. Třídění probíhá obdobným způsobem, jako v případě seznamu rozhleden (viz 4.8.5). Rozdílem je aplikace na odlišnou strukturu dat.

Relevantní cluster je získán dle hodnoty azimutu při volání implementované metody *onSensorChanged*. Pro to je třeba udělat několik kroků. Jako první je nutné zkontrolovat, zda hodnota azimutu patří stále do stejné oblasti dat.

```
1  ...
```

```
2  if(!mController.isSameCluster(mAzimuth.get())){
3    mAppropriatePanoramaCluster.clear();
4    mAppropriatePanoramaCluster.addAll(
5      mController.getAppropriateCluster(mAzimuth.get()));
6  }
7
8  setAlphas();
9  ...
```

Pokud ano, je volána funkce *setAlphas()*, jejíž obsah je popsán v dalším bodu práce. V opačném případě je třeba získat správná data, k čemuž slouží funkce v ukázce výše.

Nejdříve je původní cluster odstraněn ze seznamu, který je předáván k vykreslení. V druhém kroku je seznam naplněn novými hodnotami. Plnění spočívá v převodu azimutu na index clusteru, dle rozdělení, realizovaného výše. Po vrácení indexu je volána funkce *setAlphas()*. Převodní funkce má následující podobu:

```
1  private int bearingToCluster(Double bearing) {
2    final double clusterResolution = 360d/CLUSTERS_COUNT;
3    int clusterIndex = (int) (bearing/clusterResolution);
4    return clusterIndex;
5  }
```

4.12.6 Zobrazení a přepočet panoramatu

Zobrazení je zajištěno naplněním proměnné *mAppropriatePanoramaCluster*. Jak lze z UI návrhu fragmentu vidět, na obou stranách každého řádku je grafický symbol šipky. Dle velikosti odchylky mezi orientacemi je manipulováno s jejich průhledností, což zajišťuje funkce *setAlphas()*.

```
1  private void setAlphas() {
2    double transformedAzimuth = mController.getTransformedAzimuth();
3    for(OPanoramaPoint point : mAppropriatePanoramaCluster) {
4      point.setAlpha(transformedAzimuth);
5    }
6  }
```

Ve funkci je nejprve azimut transformován. Interval hodnot $\langle -180, 180\rangle$, které vrací kompas je třeba převést na interval $\langle 0, 360\rangle$, se kterým počítají body panoramatu. K tomu je použit jednoduchý převodní vztah 4.1.

$$a_t = (a + 360)\%360 \tag{4.1}$$

kde
a_t je transformovaný azimut [°],
a azimut [°].

Cluster s aktuálním seznamem bodů panoramatu je postupně procházen. Je volána metoda každého bodu *setAlpha*, které je v parametru předán transformovaný azimut. Ve funkci nastává jedna ze tří možností – buď je odchylka mezi orientací a azimutem záporná, nulová a nebo je kladná.

```
1  public void setAlpha(Double bearing) {
2    Double devAbs = Math.abs(mRealBearing - bearing);
3    Double signum = Math.signum(mRealBearing - bearing);
4    switch (signum.intValue()) {
5      case -1:
6        setAlphaL((float) (devAbs * mSample));
7        setAlphaR(0f);
8        break;
9      case 0:
10       setAlphaL(0f);
11       setAlphaR(0f);
12       break;
13     case 1:
14       setAlphaL(0f);
15       setAlphaR((float) (devAbs * mSample));
16       break;
17     default:
18       setAlphaL(1f);
19       setAlphaR(1f);
20       break;
21   }
22 }
```

Typ odchylky je stanoven pomocí matematické funkce signum. Kód poté vstoupí do podmínky, která je typem odchylky určena. Průhlednost je číslo v intervalu $\langle 0, 1 \rangle$. V proměnné *mSample* je vzhledem k počtu clusterů uložena poměrná část průhlednosti. Násobením proměnných je alfa hodnota nastavena. Druhá šipka je nastavena na průhlednou (není zobrazena). V případě nulové odchylky není zobrazena žádná z šipek. Nastane-li nějaká chyba, ze zavolána větev *default* – obě šipky jsou zcela neprůhledné. Pomocí databindingu jsou data propojena s atributem prvku *ImageView*, nazvaného *android:alpha*.

5 Dílčí vyhodnocení a postřehy

Každá stěžejní dílčí část aplikace byla po implementaci funkcí důkladně otestována. Tím byla se silnou pravděpodobností eliminována možnost výskytu chyb uvnitř složených celků, což významně pomohlo k jednoduššímu ladění grafického rozhraní aplikace.

Nejprve byly otestovány funkce pomocných knihoven. Následně byl proveden test výpočtu magnetické deklinace. Poté byly provedeny dva praktické testy. Prvním z nich je test kompasu. Druhým je testování korektního zobrazení panoramatu. Praktické testy byly provedeny jak na emulátoru, tak v praxi. Na závěr je diskutováno další praktické využití.

Testy knihoven jsou realizovány pomocí frameworku JUnit. Testy na emulátoru byly provedeny na virtuálním telefonu LG Nexus 5 se 2 GB přidělené paměti RAM. Reálné testy byly uskutečněny na přístroji LG Nexus 5.

5.1 Testování pomocných knihoven

Vzorová data, použitá jako korektní výsledky návratových hodnot testovaných funkcí byla počítána appletem [27]. U funkcí, které nejsou v appletu implementovány, byly výsledky testovaných hodnot počítány ručně.

5.1.1 Třída GeoCoord

Funkce byly v otestovány pro každou variantu, která může nastat. To znamená, že v případě geografických souřadnic byla příslušná funkce otestována pro záporné i kladné zeměpisné délky a šířky. Tabulka 5.1 zobrazuje výsledky úspěšnosti testování knihovny.

Pro funkce, jejichž návratovým typem je *double*, byla stanovena hranice tolerance 10^{-5}. Funkce, které vrací *String*, musí pro úspěšné projití testem přesně kopírovat předpis. Z tabulky lze vyvodit jasný závěr. JUnit testy dokázaly, že knihovna byla naprogramována správně a je tedy funkční.

5.1.2 Knihovna LocationOps

Výsledky testů knihovny *LocationOps* jsou znázorněny v tabulce 5.2. Každá funkce byla testována alespoň pro osm případů. Testy funkcí pro výpočet vzdálenosti bylo byly provedeny pro různé okruhy distancí.

FUNKCE	TEST. HODNOT	SPRÁVNĚ
getDecimalAzimuth	3	3
getDecimalLatOrLon	6	6
getCompassPoint	6	6
getDMSLat	6	6
getDMSLon	6	6
getDMSAzimuth	6	6

Tabulka 5.1: Testování knihovny GeoCoord

Výsledky testů do 100 km, které jsou pro aplikaci nejdůležitější, byly úspěšné s tolerancí do 10 m. To je vyhovující skutečnost. Funkce *calcHaversineDistance* a *calcSphericalLawOfCosinesDistance* jsou schopny vyhovět stejné toleranci – do 1 km v případech do 100000 km vzdálených míst a 10 km u vzdáleností delších. I přesto však bylo při bližším prozkoumání odchylek vypozorováno, že je *calcHaversineDistance* přesnější, a proto byla v aplikaci zvolena. Třetí funkce pro výpočet vzdálenosti je pro vzdálenosti delší než 50 km nepoužitelná.

Výsledky testů funkce pro výpočet směru vyhověly toleranci 0,1°. Funkce byla testována pro 20 hodnot, 10 pro každý typ směru. GPS souřadnice byly vzdálené maximálně 100 km.

Další testem byl výpočet středového bodu. Všech 10 testovaných hodnot vyhovělo toleranci $2 * 10^{-4}$, což v rámci GPS souřadnice odpovídá přibližně 18 m.

Funkce pro výpočet průsečíku vyhověla ve všech 8 testovaných případech opět se stejnou tolerancí $2 * 10^{-4}$. Test knihovny tedy proběhl úspěšně. Vzhledem ke zvoleným tolerancím lze tvrdit, že vybrané funkce jsou pro účely aplikace použitelné.

FUNKCE	**TYP**	**TOLERANCE**	**TEST. HODNOT**	**SPRÁVNĚ**
calcHaversineDistance	$<= 10^2\ km$	$10^{-2}\ km$	13	13
calcHaversineDistance	$<= 10^5\ km$	$10^0\ km$	10	10
calcHaversineDistance	$> 10^5\ km$	$10^1\ km$	2	2
calcSphericalLawOfCosinesDistance	$<= 10^2\ km$	$10^{-2}\ km$	13	13
calcSphericalLawOfCosinesDistance	$<= 10^5\ km$	$10^0\ km$	10	10
calcSphericalLawOfCosinesDistance	$> 10^5\ km$	$10^1\ km$	2	2
calcEquirectangularApproxDistance	$<= 20\ km$	$10^{-2}\ km$	10	10
calcEquirectangularApproxDistance	$> 50\ km$	$10^2\ km$	15	15
calcBearing	počáteční	$10^{-1\circ}$	10	10
calcBearing	cílová	$10^{-1\circ}$	10	10
calcMidpoint	-	$2 * 10^{-4}$	15	15
calcDestPointGivenDistAndBearing	-	$2 * 10^{-4}$	10	10
calcIntersection	-	$2 * 10^{-4}$	8	8

Tabulka 5.2: Testování knihovny LocationOps

TYP MÍSTA	PRŮMĚRNÁ. ODCHYLKA		
	D [°]	I [°]	TI [nT]
CZ	0,041	0,045	768
svět	0,030	0,020	527

Tabulka 5.3: Testování WMM

5.2 Testování WMM

V rámci knihovny WMM byl testován výpočet magnetické deklinace D, magnetické inklinace I a celkové intenzity magnetického pole TI. Pro každou z těchto veličin bylo otestováno 57 míst. Z nich leží 42 na území České republiky. Zbylými 15 místy jsou světové metropole. Ty byly přidány z důvodu, aby byla otestována funkčnost algoritmů po celém světě. Příkladem můžou být města San Diego, Johannesburg nebo Sydney.

Místy na území republiky jsou rozhledny nebo vyhlídkové body. Ty byly vybrány z knihy [24]. Ke všem místům jsou přidány potřebné informace (GPS souřadnice a nadmořská výška). Test proběhl dne 29.9.2016, výsledky jsou tedy platné k tomuto dni. Shrnutí je zobrazeno v tabulce 5.3.

Referenční hodnoty výsledků byly získány z [19]. Z výsledků v tabulce lze vidět, že odchylky od referenčních hodnot magnetické deklikace a inklinace jsou v podstatě zanedbatelné, a to pro oba typy testovaných míst. V případě celkové intenzity magnetického pole se odchylka pohybuje v řádu stovek nT, což je ve vztahu k intenzitě pole Země přibližně 1,5% chyba. Z toho důvodu lze všechny výsledky považovat za správné a tím algoritmus za korektně naprogramovaný.

5.3 Testování algoritmu pro stanovení magnetického severního pólu Země

5.3.1 Testy na emulátoru

Po spuštění aplikace byly nastaveny hodnoty magnetického pole ve všech třech osách. Tyto hodnoty byly získány za pomoci výpočtu, provedeného v [19]. Následně bylo se zařízením pomocí virtuálních ovládacích prvků otáčeno. Na základě dalšího výpočtu bylo zjištěno, v jakém směru musí být zařízení natočeno, aby ukazovalo přímo na magnetický sever Země.

Testy proběhly na třech místech (Ještěd, Pastýřská stěna, Tanečnice). Souřadnice míst byly do zařízení zaslány pomocí vstupního pole v emulátoru. Na obrazovce byl zobrazen azimut a bylo sledováno chování zařízení.

To se chovalo stabilně. Při testování bylo experimentováno s prahem filtru kompasu – tím byla nastavena jeho vyhovující hodnota. Při rychlých pohybech je třeba pro ustálení kompasu chvíli vyčkat. Tato skutečnost však není v reálných podmínkách předpokládána, jelikož si uživatel bude chtít panorama prohlížet pomalu a

důkladně. V případě nevhodného otočení zařízení (přílišné naklopení na jednu z os) je azimut samozřejmě zobrazován chybně.

Přirozené pohyby jsou algoritmem vyhodnoceny korektně. Při natočení zařízení severním směrem ukazoval azimut nulu, a tak byl učiněn závěr, že v teoretické rovině algoritmus pro stanovení magnetického severního pólu Země funguje.

5.3.2 Testy v reálných podmínkách

Testy v reálných podmínkách proběhly na stejných místech, jako testy na emulátoru. Referenční hodnotou byl klasický mechanický kompas. Při testech byly okem pozorovány stejné výsledky na obou zařízeních.

Jediným rozdílem bylo, že v případě mobilního zařízení trvalo déle, než se azimut kompasu ustálil na konečné hodnotě. Důvodem této skutečnosti je programová konstrukce, která zahrnuje filtraci hodnot. Stejně jako při testech na emulátoru je v případě nekorektní polohy zařízení azimut chybný. Tento fakt platí také pro mechanický kompas.

Na základě výše uvedených faktů byl učiněn závěr, že algoritmus pro stanovení magnetického severního pólu Země je v praxi použitelný.

5.4 Testování korektního zobrazení viditelného 360° panoramatu

Testy zobrazení panoramatu proběhly na třech vybraných rozhlednách – Ještěd (Liberec), Pastýřská stěna (Děčín) a Hnědý vrch (Pec pod Sněžkou). Sběr dat proběhl pomocí portálu Mapy.cz. V testech byla zkoumána především skutečnost jestli směr, kterým zařízení míří, koresponduje s popiskem zaměřeného místa. Dále bylo kontrolováno, zda svítivost šipek ukazujících směr odpovídá odchylce od dalších bodů.

5.4.1 Testy na emulátoru

V prvním kroku byla pro každé místo nastavena intenzita magnetického pole Země. Údaje o ní byly opět získány dle GPS souřadnic místa z [19]. Dále byla do virtuálního zařízení odeslána informace o poloze, čímž byl uživatel lokalizován a začalo se zobrazovat panorama.

Vždy bylo nutné počkat po dobu přibližně tří sekund v klidové poloze, než se výstup kompasu ustálil. Poté bylo možné se zařízením pohybovat a otáčet. Při testech změn polohy pomocí zaslání nových GPS souřadnic (v tolerovaném radiusu) bylo panorama dynamicky přepočítáváno, což je žádoucí. Především u blízkých bodů panoramatu bylo znát, že se po změně polohy uživatele nachází v jiném úhlu.

Pokud se uživatel ocitl mimo tolerovaný radius, bylo panorama nahrazeno informačním textem, že se nachází mimo prostory rozhledny.

Přepínání mezi jednotlivými clustery dat bylo plynulé, bez nepříjemných vizuálních efektů pro oko. Avšak, při rychlých pohybech bylo opět nutné ponechat zařízení

několik málo sekund v klidové poloze, aby byl kompas ustálen a panorama bylo zobrazeno korektně.

Pomocí pomocných výpočtů bylo zjištěno, na jakém azimutu by se kontrolované místo mělo nacházet a bylo otestováno, zda tomu tak opravdu je. Výsledky na emulátoru bez výjimky potvrdily, že algoritmy, zpracovávající místa panoramatu fungují korektně a daný bod se na odpovídajícím úhlu nacházel.

Po otestování celkem čtyřiceti bodů byl učiněn závěr, že zobrazení panoramatu na emulátoru funguje.

5.4.2 Testy v reálných podmínkách

Rozhledny byly postupně navštíveny v průběhu testování aplikace. Ještěd byl navštíven dne 20.11.2016, Pastýřská stěna dne 6.12.2016 a Hnědý vrch dne 22.12.2016. Časový údaj předaný algoritmu je tedy platný ke dni návštěvy. Test proběhl z pohledu uživatele aplikace. Lokalizace uživatele proběhla bez problémů, pro získání polohy bylo nutné počkat po dobu několika sekund.

Následně začlo být zobrazováno panorama. Data z kompasu bylo třeba nechat ustálit přibližně po tři sekundy. Při pomalém otáčení telefonu se panorama měnilo, interaktivně odpovídalo výstupům z čidel. Provedl-li uživatel rychlý nepřirozený pohyb, bylo třeba vyčkat na ustálení dat z kompasu. Bylo nutné, aby uživatel držel zařízení v přirozené poloze, tedy rovnoběžně se zemí nebo pouze mírně nakloněné (popsáno v 5.3).

Přesnost panoramatu odpovídala zobrazovaným popiskům. Při prohlížení panoramatu rozhledny Hnědý vrch byla po nějaké době přesnost zobrazení snížena. To je dáno chybnými výstupními daty z kompasu, který se stal nekalibrovaným. Situace byla pravděpodobně zapříčiněna kontrukcí rozhledny, která zahrnuje ocelové nosníky a zábradlí. Je tedy možné, že bylo v nejbližších prostorech rozhledny indukováno magnetické pole, které kompas rušilo. Po kalibraci kompasu doporučenou procedurou bylo zobrazení opět korektní.

Je zajímavé, že tento jev nastal také v případě rozhledny Ještěd. Situace byla registrována i přesto, že vyhlídkové plochy rozhledny jsou umístěny na více místech, která jsou od budovy vzdáleny několik desítek metrů. Na druhou stranu, Ještěd funguje jako vysílač. V jeho blízkosti je tedy připevněno velké množství antén, které jsou zdrojem elektromagnetického záření. Jev nastával především v případě pohledu severovýchodním směrem. U vyhlídkové plošiny se totiž nachází desítky vysílačů, které s největší pravděpodobností ovlivňují správný chod kompasu. Přístroj bylo nutné při prohlížení panoramatu několikrát kalibrovat.

Při prohlížení panoramatu z Pastýřské stěny žádný výrazný problém nenastal. Zařízení bylo nutné zkalibrovat pouze po spuštění aplikace. Poté se prohlížení obešlo bez dalších komplikací. Naopak se ukázalo, jak důležitým krokem bylo rozdělení bodů panoramatu do dostatečného počtu clusterů. Většina bodů leží severovýchodním až jihovýhodním směrem, a tím by se panorama stalo nepřehledné.

Problémy, které nastaly, jsou způsobeny vnějšími vlivy a nelze je softwarovým řešením odstranit. Z toho důvodu byl vyvozen závěr, že navržený algoritmus implementovaný do vzorové aplikace je naprogramován korektně.

5.4.3 Vzájemné porovnání výsledků a vyhodnocení testů

Díky tomu, že v emulátoru lze nastavit ideální podmínky pro testy čidel – přesné hodnoty intenzity magnetického pole, nadmořská výška, GPS souřadnice polohy – je tento způsob vhodný pro otestování algoritmů bez zkoumání vlivu vnějších sil. Tím je možné je odladit a připravit je na reálný provoz.

Rozdíl mezi teoretickou a praktickou rovinou testů je patrný. Je jím nutnost kalibrace čidel, způsobená lokálními magnetickými poli. Ty v případě emulace neexistují, a tak do algoritmů vstupují idealizovaná, nezkreslená data. Z toho důvodu jsou výsledky přesné a pro ověření funkčnosti algoritmů vyhovující.

Lokální pole lze v praxi odstranit kalibrací čidel zařízení. Poté je navržený systém opět použitelný v praxi. Kalibrace je neodmysliteným prvkem, míra její potřeby závisí především na kvalitě čidel a míře vlivu vnějších nežádoucích sil. Vyskytuje-li se uživatel v prostoru, kde nejsou vlivná magnetická pole, není častá kalibrace nutná.

Po zhodnocení obou typů provozu se testy ukázaly jako úspěšné. Navržený systém je použitelný jak v teoretické rovině, tak v praxi.

5.5 Případné využití v praxi

Navržený algoritmus není platný pouze pro rozhledny. Lze jej využít pro jakékoliv místo na Zemi, které je pouze nutné zavést do databáze. Navržené knihovny mohou být využity v algoritmech, využívajících navigaci pomocí GPS. Díky implementovanému modelu WMM lze stanovit magnetickou deklinaci, inklinaci a intenzitu magnetického pole kdekoliv na Zemi. Toho může být využito také pro výzkumné účely. Sestrojený elektronický kompas lze kromě funkce při přepočtu panoramatu použít jako navigační prostředek na výletech a túrách.

Aplikaci jako celek by bylo možné rozšířit mezi informačními středisky v České republice. Tím by se dal zjistit její potenciál a další využití. Jedním z příkladů může být zapůjčení interaktivního zařízení na pokladně rozhledny, které by uživateli panorama navštíveného místa zprostředkovalo. To by také omezilo shlukování turistů u informačních tabulí s panoramatem (pokud se v místě vůbec nachází).

Další využití je jako turistický průvodce rozhlednami. Díky seznamu uložených rozhleden, které jsou dostupné i bez připojení k internetu, lze snadno naplánovat výlet.

6 Závěr

Algoritmus pro opravu odchylky mezi magnetickým a geografickým severním pólem byl navržen. Pro jeho úspěšnou implementaci bylo třeba získat informace o magnetickém severu a magnetické deklinaci. Pomocí těchto údajů bylo možné provést opravu odchylky.

Magnetický sever byl získán prostřednictvím algoritmu, využívajícího výstup z čidel mobilního zařízení. Magnetická deklinace, závislá na poloze uživatele, čase a nadmořské výšce byla získána programovou implementací matematického modelu WMM.

Pro přívětivou komunikaci s uživatelem bylo vytvořeno grafické rozhraní, které kromě zobrazení panoramatu evidovaných rozhleden (implementace algoritmu pro opravu odchylky) umožňuje místa výhledu také prohlížet, třídit, filtrovat a vyhledávat v nich dle daných parametrů. Každá rozhledna obsahuje detailní popis, s jehož pomocí získá uživatel potřebné informace o místě.

Byla vytvořena databáze rozhleden a bodů výhledu, kterou reprezentuje několik souborů typu JSON. Ty byly soustředěny na serverové straně aplikace. Pro komunikaci mezi databázovou vrstvou aplikace a klientem bylo za pomoci funkčních knihoven vytvořeno aplikační rozhraní. To usnadňovalo samotnou komunikaci.

Pro ušetření datového provozu byly databázové soubory mapovány na interní databázové uložiště v mobilním zařízení. To pracovalo na principu relačních databází SQL. Pomocí frameworku s nimi bylo možné jednoduše pracovat.

Před vývojem aplikace byla provedena rešerše existujících alternativ, včetně jejich nedostatků. Ty byly použity jako motivace pro vývoj vlastního řešení. Proběhla také rešerše existujících technologií. Na základě jejich vlastností byl vybrán seznam vhodných technologií. Z nich byl sestaven teoretický model aplikace. Ten byl následně implementován do programové podoby, jejíž výsledkem je mobilní aplikace typu klient-server.

Byl uveden krátký souhrn prvotního návrhu, který byl demonstrován v rámci předmětu PRO. Především z nedostatků tohoto projektu bylo čerpáno při definici požadavků na aplikaci, vyvinutou v této diplomové práci.

V průběhu vývoje byly programové komponenty průběžně testovány, aby byla v co nejvyšší míře eliminována možnost chyb při volání funkcí naprogramovaných knihoven. Díky tomu bylo možné snadněji programovat samotné grafické prostředí aplikace.

Nejdříve byla otestována každá funkce třídy GeoCoord, se 100% funkčností. Následně proběhl důkladný test třídy LocationOps. U funkcí, které počítaly stejnou věc různými způsoby, byla stanovena tolerance. Byly vybrány nejvhodnější metody

pro použití v aplikaci. Proběhlo celkem 128 testů funkcí této třídy, z nichž pouze některé vyhověly tolerančním požadavkům aplikace.

Dále proběhl podrobný test implementace WMM. Testování bylo provedeno pro 57 míst, z toho 42 jich bylo na území České republiky. Zbylými místy byly světové metropole. Díky tomu byla otestována globální funkčnost implementovaného modelu. Průměrné odchylky výsledků od referenčních hodnot byly tak malé, že je lze považovat za zanedbatelné. Pravděpodobně byly způsobeny fyzickou reprezentací datových typů.

Testy funkčnosti kompasu (emulátor, praxe) proběhly úspěšně. S parametrem filtru výstupu kompasu bylo experimentováno – díky tomu byla nastavena jeho vhodná hodnota.

Test korektního zobrazení viditelného 360° panoramatu na emulátoru proběhl úspěšně. Díky simulaci ideálních podmínek emulátorem byly odladěny chyby, které by jinak mohly být přičítány vnějším vlivům. Výsledky testovacího provozu ukazují nutnost kalibrovat čidla zařízení v případě, že se kolem něj vyskytovala rušivá lokální magnetická pole. Po kalibraci byly zobrazené výsledky uspokojivé a test, který proběhl na třech rozhlednách, byl ukončen jako úspěšný.

Na testovaném zařízení se aplikace chovala stabilně. Na závěr bylo diskutováno další možné využití v praxi.

Do budoucna by bylo vhodné provést důkladný test napříč zařízeními, z kterého lze vyvodit závěry pro zlepšení stability aplikace na různých produktových řadách. Dále by bylo dobré přepracovat fragment pro kalibraci čidel, který by mohl s uživatelem komunikovat interaktivně a sám tak pomocí existujících technik zjistit, kdy jsou čidla zkalibrována. Tato implementace kalibrace do zobrazení panoramatu by mohla být dalším krokem ve zlepšení aplikace.

Vylepšit by se také dalo prohlížení rozhleden, kde by mohla být automaticky nabízena další zajímavá místa či restaurace v okolí.

Rapidně vylepšit by šlo také zobrazení panoramatu. Při využití kamery zařízení a pokročilého zpracování obrazu by mohly být popisky přímo nad body výhledu. K tomu mohou napomoci algoritmy, využívané u augmentované reality. Možnými kandidáty jsou knihovny Wikitude nebo OpenCV.

Rozšiřování databáze evidovaných rozhleden spolu s jejich viditelným panoramatem je také klíčem k udržení životnosti aplikace a jejímu rozšíření. Jelikož je navržený systém platný po celém světě, je možné provést jazykové mutace. Spolu s rozšířením databáze o zahraniční rozhledny lze aplikaci expandovat také do cizích zemí.

Literatura

[1] ORACLE AND/OR ITS AFFILIATES. *Enum RoundingMode* [online]. 2016. Dostupné z: <https://docs.oracle.com/javase/7/docs/api/java/math/RoundingMode.html>.

[2] BROTHÁNKOVÁ, M. *Ocenění aplikace Vysočina Tourism* [online]. 2016. Dostupné z: <https://www.kr-vysocina.cz/cestne-uznani-z-festivalu-tour-region-film-pro-mobilni-turistic\-kou-aplikaci-kraje-vysocina/d-4076311/p1=84932>.

[3] CHULLIAT A. aj. The US/UK World Magnetic Model for 2015-2020: Technical Report. Technical report, National Geophysical Data Center, NOAA, 2015.

[4] MICROSOFT CORPORATION *Obchod Windows Phone* [online]. 2016. Dostupné z: <https://www.microsoft.com/en-us/store/apps/windows-phone?icid=en_US_Store_UH_apps_WinPho>.

[5] HÁJEK, M. *Aplikace Seenery* [online]. 2016. Dostupné z: <http://seeneryapp.com/>.

[6] *Activity* [online]. 2016a. Dostupné z: <https://developer.android.com/reference/android/app/Activity.html>.

[7] GOOGLE INC. *Data Binding Library* [online]. 2016b. Dostupné z: <https://developer.android.com/topic/libraries/data-binding/index.html>.

[8] GOOGLE INC. *Data Binding Library: Custom Binding Class Names* [online]. 2016c. Dostupné z: <https://developer.android.com/topic/libraries/data-binding/index.html#custom_binding_class_names>.

[9] GOOGLE INC. *Location Strategies: Maintaining a current best estimate* [online]. 2016d. Dostupné z: <https://developer.android.com/guide/topics/location/strategies.html>.

[10] GOOGLE INC. *SensorManager: Rotation Matrix* [online]. 2016e. Dostupné z: <https://developer.android.com/reference/android/hardware/SensorManager.html#getRotationMatrix(float[],float[],float[],float[])>.

[11] GOOGLE INC. *SensorEvent: Gyroscope* [online]. 2016f. Dostupné z: <https://developer.android.com/reference/android/hardware/SensorEvent.html#values>.

[12] GOOGLE INC. *Sensor Fusion on Android Devices: A Revolution in Motion Processing* [online]. 2010. Dostupné z: <https://youtu.be/C7JQ7Rpwn2k?t=2040>.

[13] The PHP Group *JSON Functions* [online]. 2016. Dostupné z: <http://php.net/manual/en/function.json-encode.php>.

[14] HORACEK, L. *Recycler view inside NestedScrollView causes scroll to start in the middle* [online]. 2016. Dostupné z: <http://stackoverflow.com/questions/36314836/recycler-view-inside-nestedscrollview-causes-scroll-to-start-in\-the-middle>.

[15] APPLE INC. *Obchod App Store* [online]. 2016. Dostupné z: <https://itunes.apple.com/us/genre/ios/id36?mt=8>.

[16] GOOGLE INC. *Obchod Google Play* [online]. 2016. Dostupné z: <https://play.google.com/store>.

[17] RESEARCH IN MOTION LIMITED *Obchod Blackberry World* [online]. 2016. Dostupné z: <https://appworld.blackberry.com/webstore/?countrycode=CZ&lang=en>.

[18] NATIONAL COORDINATION OFFICE FOR SPACE-BASED POSITIONING, NAVIGATION AND TIMING *Space Segment* [online]. 2016. Dostupné z: <http://www.gps.gov/systems/gps/space/>.

[19] NATIONAL GEOPHYSICAL DATA CENTER *Magnetic Field Calculators* [online]. 2015. Dostupné z: <https://www.ngdc.noaa.gov/geomag-web/>.

[20] PASEO *Tipy na výlet – Vyletnik.cz* [online]. 2014. Dostupné z: <http://www.vyletnik.cz/android>.

[21] REQUERY.IO *Requery* [online]. 2016. Dostupné z: <https://github.com/requery/requery/blob/master/README.md>.

[22] SEZNAM. CZ *Aplikace Mapy.cz* [online]. 2016. Dostupné z: <https://napoveda.seznam.cz/cz/aplikace/aplikace-mapy/aplikace-mapy-vlastnosti-funkce/>.

[23] ALTAIR SOFTWARE *Vysočina GeoHra* [online]. 2016. Dostupné z: <https://play.google.com/store/apps/details?id=cz.altairsoftware.vysocina.geohra>.

[24] ŠTEKL, J. *RÁJEM ROZHLEDNOVÝM na kole, pěšky, lanovkou i tramvají.* Cykloknihy, 2009. ISBN 978-80-87193-04-4.

[25] TOPTAL *Heap Sort* [online]. 2016. Dostupné z: <https://www.toptal.com/developers/sorting-algorithms/heap-sort>.

[26] VÁCLAVEK, D. Mobilní aplikace poskytující informace o rozhlednách. Projekt, Technická univerzita v Liberci, 2015.

[27] VENESS, CH. *Calculate distance, bearing and more between Latitude/Longitude points* [online]. 2002. Dostupné z: <http://www.movable-type.co.uk/scripts/latlong.html>.

A Obsah přiloženého DVD

- Zdrojové kódy diplomové práce,
- dokument s testem WMM,
- manuál WMM,
- elektronická podoba tohoto dokumentu,
- vývojové diagramy aplikace Visio,
- soubory aplikace LaTeX, použité při vytváření tohoto dokumentu.