VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE - BEZBARIÉROVÉ TRASY V RÁMCI CENTRA MĚSTA

MOBILE APPLICATION - BARRIER-LESS TRAJECTORIES IN CITY CENTRE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. ADAM PODOLA**
AUTHOR

VEDOUCÍ PRÁCE **Prof. Dr. Ing. PAVEL ZEMČÍK**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/19723/2016/xpodol05

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Podola Adam, Bc.**
Obor: Informační systémy
Téma: **Mobilní aplikace - bezbariérové trasy v rámci centra města**
Mobile Application - Barrier-Less Trajectories in City Centre
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s principy tvorby mobilních aplikací pro platformu Android. Seznamte se též s aplikacemi pro navigaci v městech.
2. Navrhněte aplikaci zobrazující bezbariérové objekty v rámci města. Aplikace by měla využívat stávající služby identifikující i přístupnosti vybraných objektů v centru. Aplikace by měla mít možnost plánování bezbariérových tras ajejich optimalizaci na základně aktuálního stavu dat, například o stavu stavebních úprav, překážek nebo využitelnosti komunikací.
3. Navrženou mobilní aplikaci implementujte a demonstrujte na vhodném příkladu.
4. Mobilní aplikaci zpřístupněte pomocí online distribuční služby Google Play.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti dalšího vývoje.

Literatura:

- Dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 2 zadání

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zemčík Pavel, prof. Dr. Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Diplomové práce popisuje základy frameworku Android z hlediska budoucího vývoje mobilních aplikací. Čtenář bude seznámen se stávajícími aplikacemi pro podporu tělesně postižených. Dále bude popsán návrh a implementace nové mobilní aplikace, zobrazující informace o vybraných budovách a poskytující možnost zadávání překážek nacházející se na veřejném prostranství v blízkosti centra města Brna. V neposlední řadě bude vysvětlen princip zvoleného postupu nalezení potenciálně bezbariérových tras, během kterého je brán zřetel na aktuální stav databáze překážek.

Abstract

This diploma thesis describes the fundamentals of the Android framework for the future development of mobile applications. The reader will be introduced to existing applications for the support of physically disabled people. It will also describe the design and implementation of a new mobile application, displaying information on selected buildings and giving the possibility of marking obstacles in the public space near the city centre of Brno. Finally, the principle of the chosen procedure of finding potentially barrier-free routes, during which the state of the obstacle database is taken into account, will be explained.

Klíčová slova

Android SDK, REST API, optimalizace trasy, Google Places, Google Directions.

Keywords

Android SDK, REST API, route optimization, Google Places, Google Directions.

Citace

PODOLA, Adam. *Mobilní aplikace - bezbariérové trasy v rámci centra města*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Zemčík Pavel.

Mobilní aplikace - bezbariérové trasy v rámci centra města

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Prof. Dr. Ing. Pavla Zemčíka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Adam Podola
22. května 2017

Poděkování

Poděkování patří profesorovi Pavlu Zemčíkovi za vedení diplomové práce, poskytnuté rady, důležité připomínky a věcné komentáře. Dále technické konzultantce magistře Janě Morávkové za čas strávený plánováním a realizací několika schůzek a poskytnutí závěrečné zpětné vazby. V neposlední řadě inženýrovi Jiřímu Jandovi za krátké konzultace ohledně serverové části výsledné aplikace.

Obsah

Kapitola 1

Úvod

Chůze, pro mnohé samozřejmost. Určitá skupina lidí je o tuto přirozenost ochuzena, nebo o ni v průběhu svého života z různých důvodů částečně či zcela přijde. Proto bychom dotyčným lidem měli pomáhat a usnadnit tak jejich mobilitu. Tato myšlenka se stala motivací pro tvorbu diplomové práce, jejíž výsledkem by měl být právě jeden z prostředků v omezené míře usnadňující mobilitu tělesně postiženým občanům města Brna anebo jeho návštěvníkům.

Hlavním cílem diplomové práce je vytvoření mobilní aplikace pro platformu Android, umožňující zobrazení vybraných budov města Brna společně s informacemi o jejich bezbariérové přístupnosti, primárně využitelné osobami s různými úrovněmi tělesného postižení. Poskytované informace pak mohou potřebným pomoci během rozhodování otázky navštívení konkrétního místa. Mobilní aplikace dále poskytuje možnost zadávání překážek nacházejících se na veřejném prostranství, především v blízkosti městských komunikací. Překážkou může být myšlena bariéra částečně nebo zcela bránící v průjezdu invalidního vozíku nebo chůzi tělesně postiženého jedince. Na základě množiny poskytnutých překážek pak mobilní aplikace bude moci zobrazit bariérové a potenciálně bezbariérové trasy. Uživatel tímto způsobem získá přehled o problematických úsecích, které se na jeho cestách mohou vyskytnout. Výsledná mobilní aplikace je určena výhradně pro použití na území statutárního města Brna, které je zároveň zadavatelem této diplomové práce. Veškerá data, kromě informací týkajících se budov, zadávají samotní uživatelé aplikace, kteří tak vytvářejí úzkou komunitu. Pro tyto účely jsou dostupné možnosti zadávání komentářů a hodnocení. Celkově tedy aplikace může fungovat bez nutnosti zásahů třetích osob.

Textová část diplomové práce je rozdělena do několika kapitol. Kapitola 2 poskytuje čtenáři základní informace o platformě Android se zaměřením na budoucí vývoj mobilní aplikace. Kapitola 3 se zabývá několika vybranými projekty, které stojí za stávajícími aplikacemi pro podporu osob s tělesným postižením. V následující 4. kapitole je uvedena neformální specifikace a analýza požadavků kladených na výslednou aplikaci. Kapitola 5 popisuje strukturu implementace serverové části řešení. V nejobsáhlejší kapitole 6 jsou uvedeny principy doplněné o detaily implementace klientské mobilní aplikace.

Diplomová práce volně navazuje na Semestrální projekt, který je základem kapitoly 2, tvořící teoretickou část současné práce. Dále byly využity snímky první verze aplikace uvedené v příloze A. Konečně, obsah podkapitoly 3.1 taktéž částečně vychází z textové zprávy Semestrálního projektu.

Kapitola 2

Platforma Android

Aktuální kapitola obsahuje popis vybraných základních prvků téměř každé mobilní aplikace určené pro platformu Android. Jako základní prvek můžeme označit třídy poskytované balíčkem SDK, zejména *Activity*, *Fragment* a *AsyncTask*. Dále budou popsány dostupné prostředky pro vývoj mobilních aplikací, a to zejména zmíněný balíček SDK, vývojové prostředí Android Studio doplněné o krátký popis adresářové struktury projektu (resp. modulu) se zaměřením na pohled *Android Project View*. Větší prostor bude věnován podkapitole popisující soubor *AndroidManifest*, který je „srdcem“ každé aplikace. Poté bude popsáno několik možností uchování a předávání aplikačních dat. Kapitola bude zakončena krátkým popisem věnující se systémovým oprávněním.

Tabulka 2.1 obsahuje verze platformy Android, které mají alespoň 0,1procentní podíl na současném trhu. Můžeme si povšimnout, že platforma Android je dosti fragmentována. Nejen díky těmto informacím se můžou návrháři lépe rozhodnout, které verze platformy bude vyvíjená aplikace podporovat s ohledem na vynaložené úsilí pro zpětnou kompatibilitu a pokrytí co nejvyššího počtu potenciálních uživatelů. Data z tabulky získává společnost *Google* analýzou přístupů do své mobilní aplikace *Google Play Store*, která může být spuštěna na zařízeních s platformou verze 2.2 a vyšší. Data jsou většinou aktualizována v sedmidenních intervalech [14].

Výše zmíněná tabulka obsahuje mj. sloupec „Podíl A“, který udává procentuální rozložení ke dni 1. 1. 2017, zatímco „Podíl B“ ke dni 1. 5. 2017. Z hodnot je patrný očekávaný vývoj, kdy starší verze platformy jsou postupně nahrazovány novějšími. Za relativně krátký časový úsek čtyř měsíců dokázaly přístroje od verze platformy 6.0 získat oproti starším verzím právě 9,4 procent. Je tedy zřejmé, že trh se rychle mění a návrháři musí neustále sledovat aktuální trendy a na tyto patřičně reagovat.

2.1 Přizpůsobení rozměrům obrazovky

Platforma Android je určena pro nejrůznějších zařízeních, která nabízejí odlišné velikosti a rozlišení displeje. Z toho důvodu je dostupné konzistentní vývojové prostředí napříč zařízeními, které zpracovává a napomáhá přizpůsobit uživatelské rozhraní dané aplikace aktuálním parametrům displeje. Současně poskytuje API (*Application Programming Interface*) dovolující modifikovat uživatelské rozhraní až při samotném běhu aplikace. Toho může být využito například pro rozhodnutí, zda se má zobrazit UI (*User Interface*) aktuálně přizpůsobené pro mobilní telefon či zařízení s větší zobrazovací plochou (např. u tabletu).

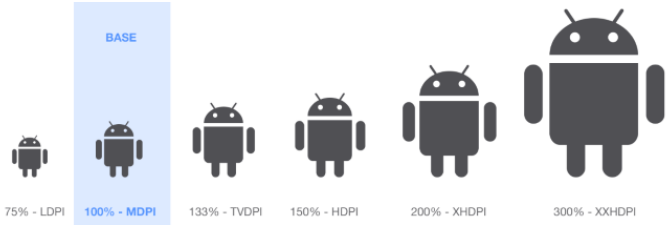
Verze	Označení	API	Podíl A (%)	Podíl B (%)
2.2	Froyo	8	0,1	-
2.3.3 - 2.3.7	Gingerbread	10	1,2	0,9
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1,2	0,9
4.1.x	Jelly Bean	16	4,5	3,5
4.2.x	Jelly Bean	17	6,4	5,1
4.3	Jelly Bean	18	1,9	1,5
4.4	KitKat	19	24,0	20,0
5.0	Lollipop	21	10,8	9,0
5.1	Lollipop	22	23,2	23,0
6.0	Marshmallow	23	26,3	31,2
7.0	Nougat	24	0,4	4,5
7.1	Nougat	24	-	0,4

Tabulka 2.1: Procentuální rozložení verzí platformy Android

V anglické literatuře popisující problematiku týkající se uživatelského rozhraní platformy Android se vyskytují následující základní pojmy[28][33]:

- *Screen size* — skutečná fyzická velikost displeje, definovaná jako jeho úhlopříčka. Pro zjednodušení se uvažují velikosti *small* (do 3"), *normal* (3-5"), *large* (4-7"), *extra-large* (nad 7").
- *Screen density* (hustota obrazovky) — obvyklé označení pomocí jednotek *dpi* - *dots per inch* - množství (téže hustota) zobrazovacích bodů (pixelů) v rámci fyzické oblasti displeje na jednotku palec. Pro zjednodušení Android seskupuje všechny hustoty obrazovky do šesti skupin: *low* (ldpi), *medium* (mdpi), *hight* (hdpi), *extra-hight* (xhdpi), *extra-extra-hight* (xxhdpi) a konečně *extra-extra-extra-hight* (xxxhdpi). Vizualizace zmíněného lze pozorovat na obrázku 2.1. Zde je uvedena i další hustota *tvdpi* (213 dpi), která se využívá při vývoji aplikací určených pro běh na televizních obrazovkách.
- *Orientation* — orientace zařízení - vodorovná (*landscape*) nebo svislá (*portrait*). Nutno podotknout, že různá zařízení pracují v různých výchozích orientacích, dále se může orientace za běhu aplikace měnit v případě, kdy tato funkce není explicitně zakázána uživatelem v nastavení zařízení nebo v programové části aplikace, přičemž první jmenovaná má vždy přednost před druhou.
- *Resolution* — klasické rozlišení displeje. Aplikace by neměla pracovat přímo s rozlišením, ale měla by uvažovat pouze velikost displeje a hustotu bodů na palec.
- *Density-independent pixel* — označení pomocí virtuálních jednotek *dp*. Právě použití této jednotky je doporučováno při definici uživatelského rozhraní, jako je například velikost a pozice UI prvků. Jeden *dp* pixel je ekvivalentní jednomu fyzickému pixelu na základním (*baseline*) displeji. Za základní hustotu displeje je uvažováno 160 *dpi*, což odpovídá skupině *mdpi* zvýrazněné na obrázku 2.1. Při běhu aplikace systém automaticky přepočítává velikosti zadané v *dp* jednotkách na hodnotu v pixelech, a to s použitím hodnoty *dpi* aktuálního displeje podle vzorce $px = dp * (dpi/160)$.

[1]Obrázek převzat a upraven z http://habrahabr.ru/post/237931/ [Online; navštíveno 01.01.2017]



Obrázek 2.1: Hustoty obrazovky, hdpi je považováno za základní hustotu.[1]

Výše zmíněný výčet zahrnoval mimo jiné jednotky *px*, *dpi*, *dp* použitelné pro specifikaci délkových hodnot. Následuje výčet dalších jednotek, určené především pro práci s prvky zobrazující textové údaje[33]:

- *sp* — scaled pixels - jedná se o jednotky v mnohém podobné jednotkám *dp*, avšak tyto primárně slouží pro definici velikostí textu. Pokud je velikost textu definována právě pomocí *sp* jednotek, bude při vykreslení textu brána v potaz zvolená hodnota velikosti textu v nastavení zařízení. Například uživatelé se zhoršeným zrakem mohou mít nastaveny vyšší hodnoty velikosti textu, pak se jim tedy text bude zobrazovat požadovaně větší.
- *pts* — points - jedná se o další možnost zvolení převážně velikosti písma. Jeden *pts* má fixní velikost a je přibližně roven 1/12 palce. Text bude mít stejný rozměr na všech displejích. Tato jednotka je široce známa z většiny textových editorů.
- *in* — inches; *mm* — milimeters - klasické jednotky.

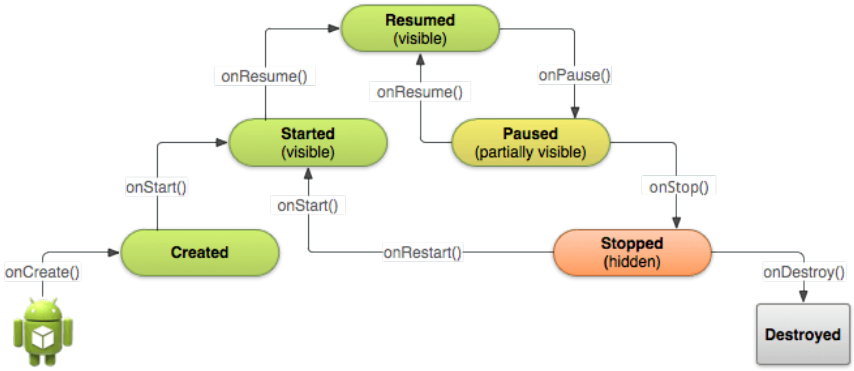
2.2 Základní stavební prvky aplikace

Následující podkapitoly popisuje základní stavební kameny téměř každé aplikace pro platformu Android. Klade si za cíl vysvětlit metody životního cyklu tříd reprezentující aplikaci, aktivitu, fragment a službu. Dále popisuje způsoby paralelního programování, zvláště pak s využitím třídy *AsyncTask*.

2.2.1 Aktivita

Každá aplikace pro platformu Android musí obsahovat alespoň jednu třídu dědící od třídy *android.app.Activity* (dále označována jako aktivita). Všechny aktivity dané aplikace procházejí v průběhu své existence vlastním životním cyklem začínaje vytvořením aktivity, během a konče odstraněním ze zásobníku aktivit (potažmo její zničení). Důvodem může být například optimalizace využití paměti či spotřeby elektrické energie, dále změna orientace zařízení z vodorovné na svislou nebo opačně[31].

Aplikace se obvykle skládá z více aktivit, přičemž jedna aktivita je v souboru manifest (kapitola 2.3.4) patřičně označena jako hlavní. To znamená, že po startu aplikace se zobrazí uživateli obsah poskytovaný právě takto označenou aktivitou. Pokud některá aktivita spustí jinou, bude předchozí umístěna na vrchol zásobníku LIFO (*last in, first out*) a systém ji pozastaví. Libovolná pozastavená aktivita se může stát znovu aktivní. Po uživatelském stlačení tlačítka zpět (nebo podobného mechanismu) bude z vrcholu zásobníku vyzvednuta předcházející aktivita, následně nastává obnovení jejího stavu (nebo opětovné vytvoření) a zobrazení uživateli. Aktivita má tedy v systému vlastní životní cyklus[36].



Obrázek 2.2: Životní cyklus aktivity[2]

Během průchodu životním cyklem jsou operačním systémem v jednotlivých fázích volány metody (jak je patrné z obrázku 2.2), které dovolují řídit chování aktivity. Metody jsou následující[31][33]:

- *onCreate* — metoda je volána bezprostředně po vytvoření aktivity. V tomto okamžiku je operačním systémem vše připraveno a jedná se o vhodné místo pro volání metody *setContentView*, práci s grafikou, přehrání zvuků a využívání senzorů zařízení. Metoda získá v parametru objekt třídy *Bundle* (kapitola 2.4.2), který v případě znovuvytvoření aktivity obsahuje programátorem v minulosti uložený stav, a to pomocí metody *onSavedInstanceState*. Aktivita tedy přechází do vnitřního stavu *Created*.

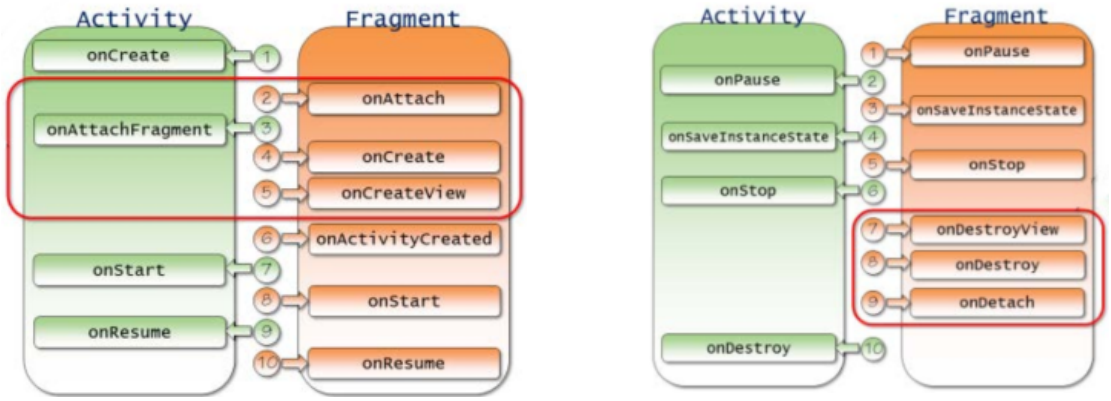
- *onStart* — volána před okamžikem, kdy se aktivita stane viditelná pro uživatele. Po jejím provedení bude kontrola předána metodě *onResume*. Jestliže se z nějakého důvodu nemůže aktivita stát viditelnou a přejít do popředí, bude místo metody *onResume* rovnou volána metoda *onStop*. Aktivita přechází do vnitřního stavu *Started*.

- *onResume* — volána po provedení metody *onStart* nebo pokud aktivita byla v minulosti pozastavena a nyní se stává znovu viditelnou. Pak by měla následovat obnova uložených uživatelských dat ze chvíle, kdy bylo provádění aktivity přerušeno například z důvodu příchozího hovoru či startu jiné aplikace. Aktivita tedy přechází do stavu *Resumed* a je viditelná uživateli, který s ní může interagovat pomocí klávesnice či dotykového displeje. Na tyto vstupy nyní může aktivita patřičně reagovat.

- *onPause* — volána ve chvíli, kdy aktivita přechází do stavu *Paused* z důvodu zobrazení dialogu nebo startu jiné aktivity. Aktivita po provedení již nebude plně viditelná. Jedná se o vhodné místo pro uložení rozpracovaných dat (v budoucnu mohou být obnovena v metodě *onResume*), o které by uživatel nerad přišel a očekává jejich dostupnost po opětovném přechodu aktivity do popředí. Dalším úkolem bývá zastavení úloh konzumujících velké množství procesorového času, které má za následek vysokou spotřebu energie (například GPS navigace).

- *onStop* — volána před přechodem do stavu *Stopped* znamenající úplné skrytí (překrytí) aktivity. V této metodě by měly být případně uvolněny zdroje, které se alokovaly v metodě *onCreate*.

[2]Obrázek převzat z http://www.codevoila.com/post/58/android-tutorial-activity-lifecycle [Online; navštíveno 01.01.2017]



(a) Sestavení a zobrazení

(b) Skrytí a smazání

Obrázek 2.3: Životní cyklus fragmentu[3]

- *onRestart* — volána, pokud byla aktivita ve stavu *Stopped*, přičemž je požadavek na opětovné zobrazení této aktivity. Provedením metody *onRestart* následuje vždy volání *onStart*[5].
- *onDestroy* — pro programátora poslední možnost provedení jakéhokoli akce v rámci aktuální aktivity, jako například uložení rozpracovaných dat perzistentní formou. Jestliže v průběhu životního cyklu byly alokovány zdroje, které doposud nebyly uvolněny, musí být jejich uvolnění provedeno nejpozději v této metodě.

2.2.2 Fragment

Fragment reprezentuje chování nebo část uživatelského rozhraní aktivity popsané v sekci 2.2.1. Výhodnou vlastností je možnost kombinovat více fragmentů v rámci jedné aktivity pro budovaní uživatelského rozhraní s více panely a v neposlední řadě jejich znovupoužitelnost mezi více aktivitami. O fragmentech bychom měli přemýšlet jako o modulárních sekcích, které mají vlastní životní cyklus a možnost přijímat a reagovat na vstupní události přijímané od uživatele. Fragmenty mohou být přidávány nebo odebírány za běhu aktivity[15].

Co se týče životního cyklu, fragment poskytuje částečně stejné metody zpětného volání, jako je tomu u aktivity. Navíc však poskytuje další, jenž souvisejí s životním cyklem aktivity, které je fragment přiřazen. S postupem času, jak se vyvíjená aplikace stává složitější a sofistikovanější, je třeba detailního porozumění životního cyklu fragmentu, aktivity a vztahu mezi nimi[37].

Sestavení a zobrazení

Obrázek 2.3a ukazuje životní cyklus fragmentu při jeho sestavení a zobrazení v závislosti na aktivitě, které je tento aktuálně přiřazen.

Nejprve je volána metoda *onCreate* aktivity. Ve většině případů je v této metodě voláno *setContentView* pro načtení UI obsahu, ve kterém předpokládejme obsažený fragment. Před vytvořením fragmentu je tento nejprve připojen pomocí metody *onAttach* aktivitě. Připojení znamená získání odkazu na rodičovskou aktivitu, která zrcadlově získá odkaz na fragment

[3]Obrázky převzaty a upraveny z [37]

pomocí metody *onAttachFragment*. Tímto mechanismem se tedy vymění reference obou stran. Fragment v mnoha případech může požadovat přístup k aktivitě pro získání dat v jeho procesu vytváření, protože aktivita často obsahuje informace, které má fragment zobrazit, nebo které jsou důležité z hlediska vytváření fragmentu. Zmíněný mechanismus získání požadovaných zdrojů podporuje. Po propojení je ve fragmentu provedena metoda *onCreate*, posléze pak *onCreateView* vytvářející uživatelský obsah. Pokud aktivita obsahuje více fragmentů, OS volá zmíněné metody (na obrázku v červeném obdélníku) postupně pro každý fragment. Postupné volání dovoluje každému fragmentu dokončit celý proces propojení a vytvoření sebe sama, přičemž následující metody jsou prováděny jednotlivě pro každý fragment v pořadí. Poté aktivita dokončí provádění své *onCreate* metody a o této akci upozorní všechny fragmenty pomocí *onActivityCreated*. Následně jsou volány zbývající metody *onStart* a *onResume*[37].

Skrytí a smazání

Předchozí sekce 2.2.2 popisovala životní fáze fragmentu při sestavení a zobrazení. Zbývá popsat opačnou – skrytí a smazání, zobrazenou na obrázku 2.3b, nastávající např. v případě minimalizace aplikace.

Z počátku se fragment chová stejně jako jeho rodičovská aktivita. Postupně se tedy volají metody *onPause*, *onSavedInstanceState* a *onStop*, a to vždy dříve, než se stejnojmenná metoda provede v rodičovské aktivitě. Následuje volání třech metod *onDestroyView*, *onDestroy* a *onDetach*. V první jmenované je odstraněn uživatelský obsah vytvořený v minulosti metodou *onCreateView*. Druhá jmenovaná odstraňuje samotný fragment. Poslední metoda způsobí odstranění odkazu na rodičovskou aktivitu. Obsahuje-li aktivita více fragmentů, operační systém volá sekvenci metod vyznačených červeným obdélníkem postupně pro každý fragment. Jakmile je sekvence metod zavolána pro všechny připojené fragmenty, následuje volání *onDestroy* metody aktivity[37].

Srovnání aktivity a fragmentu

Ve větší části je životní cyklus aktivity a fragmentu velmi podobný. Výjimkou je fragment existující v rámci aktivity bez nutnosti zobrazení jakéhokoli obsahu. Popsané chování může být dosaženo například v případě, kdy fragment čeká na příjem dat z internetu a nemá žádná data taková, která by mohl zobrazit uživateli. Za předpokladu, kdy není zobrazen žádný obsah, jsou volány pouze metody *onAttach* a *onCreate*, přičemž volání *onCreateView* je odloženo. Opačně, pokud bude požadavkem skrýt obsah fragmentu, bude volána pouze metoda *onDestroyView* (bez metod *onDestroy* a *onDetach*)[37].

2.2.3 Application

Další základní třídou, o jejíž existenci je vhodné při programování aplikací pro OS Android vědět, je *android.app.Application*. Programátor může vytvořit vlastní třídu dědící od zmíněné a její implementaci registrovat v souboru *AndroidManifest* popsaného v sekci 2.3.4 jako atribut *name* elementu *application*. Tímto postupem lze získat následující výhody[34]:

- Možný zásah do globální správy stavu aplikace,
- jednoduchý přesun objektů mezi aplikačními komponentami,
- spravovat a udržovat zdroje využívané více aplikačními komponentami.

Instance registrované třídy bude vytvořena v okamžiku startu aplikace, a to před vytvořením jakékoli jiné aplikační instance (myšleno instance ostatních tříd aplikace). Proměnné a globální zdroje definované ve třídě *Application* jsou pak přístupné kdekoli v aplikaci, díky čemuž získáváme popsané výhody. Dobrým zvykem je implementovat a využívat třídu pomocí návrhového vzoru Jedináček, jak je popisováno v literatuře[34].

Během životního cyklu celé aplikace prochází instance objektu třídy *Application* vlastním životním cyklem, podobně jako je tomu u aktivity a fragmentu popsané v podkapitolách 2.2.1 a 2.2.2. Třída *Application* poskytuje informace o událostech, na které může být žádoucí patřičně reagovat. Událostmi mohou být například vytvoření a zastavení aplikace, zjištění nízké hodnoty dostupné operační paměti, změna orientace zařízení. Programátor může doplnit reakce pomocí přepsání následujících metod[34]:

- *onCreate* — voláno při vytvoření aplikace. Jestliže vlastní třídu dědící od *Application* implementujeme pomocí návrhového vzoru Jedináček, pak právě v této metodě je třeba inicializovat její instanci. Dalšími kroky mohou být inicializace sdílených proměnných a zdrojů.

- *onTerminate* — může být volána při ukončování aplikace, ovšem nemáme žádnou záruku, že volání bude provedeno. Příkladem může být nedostatek volných zdrojů pro ostatní běžící aplikace. Jádro systému pak naši aplikaci ukončí bez volání této metody.

- *onLowMemory* — žádost při přetížení operačního systému z hlediska nedostatku operační paměti značící, aby její přidělenou část aktuálně běžící proces uvolnil. Pokud aplikace v okamžiku volání využívá nepotřebné zdroje (ve smyslu paměti), měla by se jich vhodně zbavit. Pokud aplikace nebude reagovat podle popisu, zvyšuje riziko jejího následného nežádoucího ukončení operačním systémem[10].

- *onConfigurationChange* — aktivita a fragment jsou při změně orientace zařízení opětovně vytvořeni. Opačně se chová objekt třídy *Applicatoin*, který při změně zůstává nedotčen, pouze je o skutečnosti informován voláním této metody.

2.2.4 AsyncTask

Po spuštění aplikace vytvoří systém hlavní vlákno mající na starosti vykreslení a delegaci obsluhy událostí vznikajících v komponentách uživatelského rozhraní. Komponentami jsou míněny prvky z balíčků *android.widget* a *android.wiew*, jako například tlačítka, obrázky a textová pole. Na základě této skutečnosti se hlavní vlákno často označuje pojmem UI vlákno. Časově náročná obsluha událostí prováděná uvnitř hlavního vlákna má za následek nevhodné chování aplikace. Jestliže obsluha bude trvat déle než několik sekund, systém zobrazí uživateli ANR (*Application Not Responsing*) dialog. Uživatel může pomocí dialogu aplikaci zavřít a pokud se podobné chování opakuje, existuje vysoká pravděpodobnost odinstalace aplikace uživatelem. V průběhu provádění zmíněné obsluhy tedy dochází k blokaci UI vlákna mající za následek pozastavení vykreslování veškerých UI komponent, což má negativní dopad na uživatelův prožitek. Takovému chování je třeba se vyhnout, podobně jako v jiných systémech, použitím samostatných pracovních vláken určených pro časově náročné operace. Pracovní vlákna však musí splňovat zásadní omezení. Nesmí přímo přistupovat a manipulovat s objekty odkazujícími na prvky uživatelského rozhraní[20].

Pro řešení popsaného problému poskytuje Android několik možností, jak nepřímo přistupovat k UI prvkům z pracovních vláken:

- *Activity.runOnUiThread* — spouští akci definovanou parametrem metody typu *Runnable* v rámci hlavního vlákna. Pokud je metoda volána uvnitř UI vlákna, bude akce provedena okamžitě. V opačném případě se akce umístí do fronty požadavků spojené s UI vláknem a její provedení tedy bude odloženo[5].
- *View.post* — akce bude přidána do fronty požadavků a později spuštěna v rámci UI vlákna. Metoda vrací informaci o úspěšném přidání akce do fronty[30].
- *View.postDelayed* — metoda s podobným chováním jako předchozí, avšak navíc, jako další parametr, přijímá časový údaj o požadovaném odloženém spuštění[30].
- Použití třídy *AsyncTask*[20].

Třída *AsyncTask* poskytuje možnost provádění operací na pozadí, snadné publikování výsledků v UI vlákně, a to bez nutnosti explicitní manipulace se samotnými vlákny. Výsledný kód je pak lépe čitelný a udržovatelný[20]. *AsyncTask* je ideální volba pro krátké operace v řádu sekund. Pro delší činnosti je doporučeno použití různých API poskytovaných v balíčcích *java.util.concurrent*. Asynchronní operace ve třídě *AsyncTask* je definována třemi generickými typy a čtyřmi kroky. Výčet generických typů následuje[13]:

- *Params* — typ parametrů posílaných úloze prováděné na pozadí před jejím provedení.
- *Progress* — typ parametrů posílaných při publikování změn do UI vlákna během provádění operace na pozadí.
- *Result* — typ parametrů posílaných při dokončení operace na pozadí.

Po spuštění úlohy tato prochází čtyřmi kroky[13]:

1. *onPreExecute()* — volána uvnitř hlavního vlákna před začátkem provádění operace na pozadí. Jedná se o vhodné místo pro nastavení proměnných operace či zobrazení oznámení uživateli.
2. *doInBackground(Params...)* — po volání *onPreExecute* začne provádění této metody definující dlouhotrvající operaci prováděnou v rámci samostatného vlákna. Výsledek operace je předán do 4. kroku. Pomocí metody *publishProgress(Progress...)* můžeme informovat o průběhu operace.
3. *onProgressUpdate(Progress...)* — volána v rámci UI vlákna po invokaci metody *publishProgress(Progress...)*. Obyčejně slouží pro aktualizaci oznámení uživateli o posunu v prováděné operaci.
4. *onPostExecute(Result)* — jako parametr přijímá výsledek ze 2. kroku. Slouží pro modifikaci uživatelského rozhraní na základě přijatého výsledku.

Pokud uvážíme situaci, kdy je úloha definována pomocí *AsyncTask* spuštěna v aktivitě, která bude před dokončením výpočtu restartována například z důvodu rotace zařízení. Pokud metoda *onPostExecute* modifikuje prvky uživatelského rozhraní, pak se tyto změny nemohou projevit, nebo v horším případe aplikace spadne (objekty mohou odkazovat na neexistující prvky). Proto je vhodné používat *AsyncTask* na krátké operace v řádu sekund, kterým nevadí restart aktivity nebo ve vhodných chvílích kontrolovat, zda k restartu došlo, či nikoli[32].

2.2.5 Service

Služby (anglický výraz *services*) jsou v Android aplikacích využívány pro vykonávání dlouhotrvajících operací na pozadí, u nichž není kladen požadavek modifikace uživatelského rozhraní. Jakákoli komponenta aplikace může nastartovat službu, která začne provádět operaci. Provádění pokračuje i v případě přepnutí se uživatele do jiné aplikace. Služby jsou použitelné například pro přehrávání muziky, internetovou komunikaci či poskytování vstupně-výstupních operací. Důležitým faktem u služeb je jejich spuštění v hlavním vlákně aplikace. Služba sama nevytváří další vlákna, povinnost je ponechána na programátorovi. Proto, aby platilo výše uvedené, musí být pro službu poskytující provedení dlouhotrvající operace vytvořeno nové vlákno, ve kterém bude tato provedena[25].

Rozhodnutí použití služby nebo klasického vlákna pro dlouho-trvající operaci je prakticky jednoznačné. Pokud není požadavkem, aby uživatel mohl interagovat s aplikací při provádění operace na pozadí, je vhodné použít službu. V opačném případě je doporučováno použití klasického vlákna[25].

Všimněme si rozdílu mezi službou a *AsyncTask* popsané v kapitole 2.2.4. *AsyncTask* je určena pro krátkodobé operace, jejichž výsledkem je (většinou) modifikace uživatelského rozhraní s možností zasílání zpráv do UI o průběhu provádění operace. Jedná se tedy o opak služeb, které můžeme rozdělit následovně:

- *Scheduled* — plánovaná služba je taková, která je spuštěná v definovaný časový okamžik pomocí plánovacího API *JobScheduler*[4] dostupného od Androidu 5.0 (více tabulka 2.1). Společnost Google doporučuje používat právě tento typ služby v aplikacích s minimálním podporovaným API verze 21 (více o možnostech podporovaných API v kapitole AndroidManifest 2.3.4)[25].

- *Started* — službu typu *Started* může vytvořit libovolná komponenta, například aktivita, a to pomocí volání metody *startService*. Tento typ se používá pro služby běžící neomezeně dlouhou dobu (vytvářející komponenta nemusí v době běhu existovat), u kterých není požadováno navrácení výsledku po konci jejich činnosti. Službu může ukončit jakákoli komponenta pomocí metody *stopService* nebo služba samotná voláním *shopSelf*[36].

- *Bound* — volně přeloženo jako „navázaná služba“. Tato je vytvořena voláním metody *bindService*. Navázat se k tomuto typu služby může více komponent mající možnost provádět komunikaci typu klient-server pomocí IPC (*Inter Process Communication*). Pro odpojení se od služby slouží metoda *unbindService*. Jestliže neexistuje žádné aktivní spojení, bude tato automaticky zrušena[25][36].

Životní cyklus služby

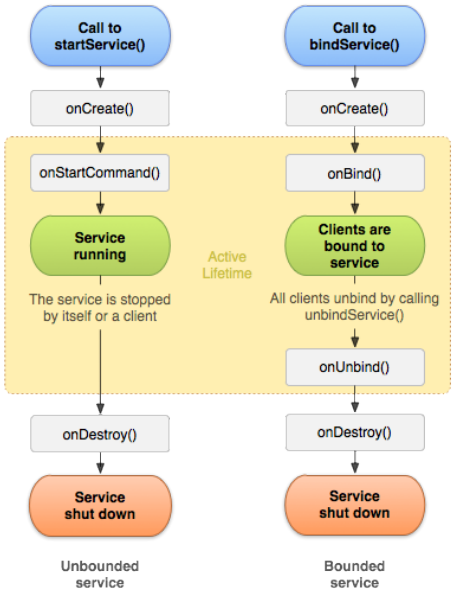
Obrázek 2.4 zobrazuje životní cyklus služby. Levá část znázorňuje službu typu *Started* (spuštěna metodou *startService*), pravá pak *Bound* (vytvoření či navázání).

Služby v průběhu svého života reagují na následující metody[36]:

- *onCreate* — volána během vytváření. V případě typu *Bound* je volána jen při prvním navázání.

[4] https://developer.android.com/reference/android/app/job/JobScheduler.html [Online; navštíveno 07.01.2017]

[5] Obrázek převzat z [25]



Obrázek 2.4: Životní cyklus služby[5]

- *onBind* — volána pro navrácení objektu třídy *IBinder* sloužícího ke komunikaci mezi službou a vázanou komponentou.
- *onStartCommand* — voláno systémem během každého požadavku na start služby[8].
- *onUnbind* — volána po odpojení veškerých připojených komponent pomocí *bindService*.
- *onDestroy* — voláno před zrušením služby, vhodné místo pro uvolnění alokovaných zdrojů.

2.3 Prostředky pro vývoj

V červnu roku 2015 oficiálně oznámil produktový manažer projektu Android *Jamal Eason* konec podpory ADT (*Android Developer Tools*) pluginu pro vývojové prostředí Eclipse[7]. Poslední verze ADT 23.0.7 byla vydána v srpnu roku 2015[6]. Na základě oznámení se oficiálním vývojovým prostředím pro platformu Android stalo *Android Studio*, které mimo jiné nahradilo nástroj pro sestavení (myšlen anglický výraz *build*) *Ant*[6] za *Gradle*. Nutno podotknout, že dva zmíněné softwarové balíčky pro sestavení fungují samostatně, programátor tedy teoreticky nepotřebuje žádné vývojové prostředí, vystačí si pouze s Android SDK.

2.3.1 SDK

Jak Eclipse společně s ATD pluginem v minulosti, tak Android Studio v současnosti pro vývoj aplikací vyžaduje balíček SDK – *Software Development Kit*.

Android SDK obsahuje vše potřebné k vývoji, testování a ladění (angl. výraz *debug*) aplikace[34]:

[6] Bližší informace o nástroji – http://ant.apache.org/

- Android API — jádro SDK tvoří API – jedná se o knihovny využívané samotnou společností Google k vývoji jejich nativních aplikací.[7] K funkcím samotného systému se přistupuje právě přes toto API. Pro aktuálně využívané verze API lze nahlédnout do tabulky 2.1.

- AVD (*Android Virtual Device*) Manager a Emulátor — emulátor, běžící na AVD, simuluje reálné hardwarové zařízení s možností nastavení mnoha parametrů (např. rozlišení displeje, velikost operační paměti, počet CPU jader). Můžeme tedy otestovat (myšleno odsimulovat) aplikaci před finálním vydáním na mnoha typech zařízení. Testy nám mohou pomoci zvýšit stabilitu a zlepšit vizuální efekt na zařízeních s různými verzemi operačního systému.

- Dokumentace a vzorové kódy — SDK zahrnuje výběr vzorových aplikací demonstrující vybranou funkcionalitu jednotlivých API.

Pro proveditelný překlad je mimo samotného SDK potřeba dodatečná komponenta *SDK Build Tools*[8]. Veškeré uvedené lze pohodlně instalovat prostřednictvím nástroje *Android SDK Manager*[9].

2.3.2 NDK

NDK je zkratkou pro *Native Development Kit*. Jedná se o množinu nástrojů dovolující pracovat s kódem napsaným v jazyce C a C++. NDK dále poskytuje knihovny pro správu nativních aktivit a přístupu k fyzickým součástem zařízení, jako jsou například senzory a dotyková obrazovka. Klasická aktivita byla popsána v kapitole 2.2.1. Aktivitu můžeme označit za nativní, pokud je zcela napsána ve zmíněných jazycích C nebo C++.

NDK není příliš vhodné pro aplikace, ve kterých se obejdeme pouze s možnostmi programovacího jazyka Java. Avšak využití NDK bychom měli zvážit minimálně v následujících případech[16]:

- Zvýšení výkonu zařízení pro dosažení nízké latence nebo pro běh výpočetně náročných operací (např. herní nebo fyzikální simulace).

- Znovupoužití celých knihoven napsaných v jazycích C nebo C++.

Samotné NDK nebylo v řešení diplomové práce využito, je uvedeno pouze pro ilustraci. Bližší informace a návody lze nalézt v literatuře[16].

2.3.3 Projekt

Následuje popis projektu ve vývojovém prostředí Android Studio, jenž sestává z minimálně jednoho modulu, standardně pojmenovaného jako „app“. Modul je kolekce zdrojových kódů a nastavení pro překlad. Výhodnou vlastností je možnost rozdělit projekt do několika samostatných funkčních jednotek, které mohou mít závislost na dalších jednotkách. Každý modul může být nezávisle sestavován, testován a laděn. Použití modulů bývá užitečné pro vytváření knihoven v rámci jednoho projektu nebo pokud chceme vytvořit odlišné množiny zdrojových

[7] Příklady nativních aplikací společnosti Google – Gmail, Google Maps.

[8] Bližší informace o nástroji – https://developer.android.com/studio/releases/build-tools.html [Online; navštíveno 03.01.2017]

[9] Bližší informace o nástroji – https://developer.android.com/studio/intro/update.html [Online; navštíveno 03.01.2017]

kódů pro různé typy zařízení, jako jsou mobilní telefony, tablety, televizory a chytré hodinky s cílem zachování veškerých zdrojových souborů a dalších zdrojů v rámci jednoho projektu, a to například z důvodu sdílení zdrojových kódů mezi moduly. Další výhodou může být snazší plánování a vzájemná spolupráce v rámci vývojového týmu. Následuje popis dvou základních typů modulů poskytovaných Android Studiem[21]:

- Android app — hlavní modul vytvářené aplikace - jeho obsah bude diskutován později.
- Knihovna (*library*) — poskytuje kontejner pro opětovně použitelný kód. Tento modul může být využit jako závislost jiných modulů v rámci stejného či odlišného projektu. Vývojové prostředí poskytuje dva typy knihovních modulů. Prvním je *Android Library*, ve kterém mohou být obsaženy veškeré soubory podporované v Android projektu (např. zdrojové kódy, obrázky, audio soubory a manifestový soubor). Výsledkem překladu pak bude *Android Archive (AAR)*. Druhým typem je *Java Library* dovolující zapouzdřit pouze Java zdrojové soubory. Výsledkem překladu bude známější *Jar* archiv.

Adresářová struktura

Android Studio poskytuje několik pohledů určených pro procházení adresářové struktury projektu. Základním pohledem je tzv. *Android View*. Tento nereflektuje aktuální hierarchii souborů uložených na disku. Organizuje projektové soubory do modulů, může skrývat nedůležité a dosud nepoužívané soubory. Dále se jím nebudeme zabývat.

Pro prozkoumání aktuální adresářové projektu struktury slouží pohled *Android Project View*, navíc zobrazující všechny soubory a složky jinak skryté v *Android View*. Mezi nejdůležitější patří[21]:

- *build* — složka obsahuje výstup překladu,
- *libs* — soukromé knihovny, většinou ve formátu *Jar* archivu,
- *src* — obsahuje veškeré zdrojové soubory a zdroje (anglický výraz *resources* - dále bude použit český překlad, pokud nebude uvedeno jinak) pro aktuální modul,
- *src/androidTest* — zdrojové kódy instrumentálních testů běžících na zařízení s OS Android. Základní informace o testování Android aplikací jsou popsány ve článku *Test your app*[10],
- *src/test* — složka pro zdrojové kódy testovacích tříd pro lokální testování (typicky Unit testy),
- *src/main/AndroidManifest.xml* — podrobněji vysvětlen v samostatné sekci 2.3.4,
- *src/main/java* — obsahuje balíčky se zdrojovými kódy samotné aplikace napsané v jazyce Java,
- *src/main/jni* — do této složky se umísťují zdrojové soubory využívající JNI (*Java Native Interface*) a funkce NDK, které bylo krátce popsáno v sekci 2.3.2,
- *src/main/gen* — obsahuje soubory v jazyce Java generované Android Studiem,

[10] https://developer.android.com/studio/test/index.html [Online; navštíveno 03.01.2017]

- *src/main/res* — obsahuje aplikační zdroje jako jsou *drawable* soubory, soubory pro definici vzhledu (tzv. *layout file*) a další,
- *src/main/assets* — zde mohou být umístěny soubory, jenž budou během překládání projektu umístěny do výsledného *.apk*[11] souboru. Jedná se tedy o vhodné umístění audio a video nahrávek, textur a herních dat,
- *src/build.gradle* — konfigurační soubor překlad modulu. Definuje informace o použité verzi *SDK Build Tools*, specifikaci klíče pro podepsání aplikace a mnoho dalších.

2.3.4 AndroidManifest

Srdcem každé aplikace, respektive každého modulu v rámci projektu Android Studia, je bezesporu soubor ve formátu XML *AndroidManifest*, dále označovaný zkráceně slovem „manifest“. Slouží pro deklaraci aktivit společně s jejími vlastnostmi, služeb (popsaných v kapitole 2.2.5), požadovaná přístupová práva nutných pro běh aplikace a mnoho další[1].

Manifest je také název prvního a zároveň kořenového elementu celého XML souboru. Musí vlastnit atribut *package*, jehož hodnota udává název kořenového balíčku všech zdrojových kódů napsaných v jazyce Java. Tato hodnota zároveň slouží jako jedinečný identifikátor vyvíjené aplikace v rámci Google Play nebo jiné distribuční služby, pomocí které bude aplikace nabízena uživatelům. Element *manifest* dále musí obsahovat atributy *versionCode* a *versionName*. První jmenovaná musí mít hodnotu typu *integer* a v případě publikování novější verze totožné aplikace, v rámci kterékoli distribuční službu k tomu určené, musí mít zmíněný atribut vyšší hodnotu než má aktuálně publikovaná verze. Druhý jmenovaný nabývá uživatelsky přívětivé hodnoty atributu *versionCode*[1].

Dalším atributem kořenového elementu *manifest* může být *installLoccation* specifikující instalační lokaci aplikace na daném zařízení. Povolenou hodnotou je *preferExternal*, resp. *auto*. První jmenovaná vynucuje instalaci aplikace na externí paměťové úložiště (jako je např. SD karta), ovšem systém nezaručuje splnění tohoto požadavku. Nemusí mu být vyhověno, pokud externí úložiště není aktuálně dostupné nebo má nedostatečnou zbývající kapacitu. Uživateli je rovněž povoleno libovolně přesouvat aplikaci mezi interním a externím úložištěm. Druhou jmenovanou hodnotou indikujeme volnost operačního systému o rozhodnutí umístnění aplikace. Použití atributu *installLoccation* bylo povoleno od verze API 8. Pokud atribut není uveden, aplikace bude vždy instalována pouze do interního úložiště zařízení a zároveň nebude povoleno její přesouvání. Instalace na externí úložiště by neměla mít dopad na výkonost aplikace. Zároveň privátní uživatelská data a soubory s databázemi jsou uložena na úložišti interním[9].

Kořenový *manifest* musí obsahovat potomka *uses-sdk* s atributem *minSdkVersion*, pomocí něhož volíme minimální verzi API, kterou musí zařízení určené pro běh aplikace vlastnit. Pokud se uživatel snaží instalovat aplikaci na zařízení, jehož systém obsahuje API menší než deklarované, pak mu instalace není povolena. Atributem *targetSdkVersion* definujeme verzi API, které je použité pro vývoj a testovaní aplikace. Pokud bude aplikace spuštěna na zařízení s vyšší hodnotou API než je deklarovaná v atributu, pak systém zapíná funkce zpětné kompatibility. Popisované atributy obsahují v názvu „Sdk“, ovšem ve skutečnosti se jedná o verzi „Api“[12]. Tato skutečnost není brána v potaz například v literatuře [1], což by mohlo býti pro většinu čtenářů matoucí. Pokud budou v souboru *build.gradle*, zmíněném v sekci 2.3.3, totožné atributy uvedeny, budou hodnoty těchto přepsány. Doporučením je tedy v případě použití Gradle vynechání atributů v souboru manifest[29].

[11] Android Application Package

Dalším podstatným elementem je *application*. Tento obsahuje základní povinné atributy pro nastavení názvu, ikony a výchozího stylu aplikace. Nepovinným elementem *hardwareAccelerated* můžeme specifikovat povolení či zakázání hardwarové akcelerace. Pokud cílíme na zařízení s API verze 14 a výše (viz atribut *targetSdkVersion*), je akcelerace zapnuta automaticky bez nutnosti explicitního uvedení. V oficiální dokumentaci jsou uvedeny občasné problémy při povolené hardwarové akceleraci (např. vykreslování neviditelných elementů či jejich částí nebo invokace výjimek). Tato tedy může být vypnuta na úrovni *application* elementu a povolena pouze pro určité aktivity, nebo je možno zvolit opačný přístup. Hardwarová akcelerace zvyšuje nároky aplikace na paměť RAM, proto její využití je v některých případech (větší a složitější projekty) nutné kombinovat s atributem *largeHeap*. Operační systém pak může aplikačním procesům přidělit vyšší kapacitu zásobníku, pokud je v dané chvíli dostupná[11][17].

Element *application* obsahuje pod-elementy *activity* pro povinné deklarování všech aktivit (2.2.1) v rámci aplikace. Pokud některá není uvedena, bude po jejím spuštění vyvolána výjimka *ActivityNotFoundException*[1].

V předchozích odstavcích byly popsány nutné základy souboru manifest. Pro bližší informace je dostupná oficiální dokumentace[11].

2.4 Uchování a předávání aplikačních dat

Platforma Android nabízí nepřeberné možnosti uchování a manipulaci s daty využitelných při vývoji mobilních aplikací. Mezi základní patří manipulace se soubory na interním či externím úložišti. Následující podkapitoly podrobněji popisují jednotlivé poskytované možnosti. Jejich použití by vždy mělo být zváženo, a to z hlediska rychlosti, efektivity a robustnosti[34].

2.4.1 SharedPreferences

Ve své podstatně se jedná o množinu dvojic klíč-hodnota. Tato je uložena v samostatném souboru s možností manipulací pomocí třídy *android.content.SharedPreferences* poskytující možnost získání objektu, prostřednictvím něhož je umožněno provádět čtení a zápis. Množina dvojic může být privátní nebo veřejná (přístupné ostatním aplikacím). Ve skutečnosti není počet množin (potažmo souborů) nikterak omezen, pokud není uvažována kapacita dostupné paměti. Hodnota libovolného uloženého záznamu může nabývat datových typů boolean, string, float, long a integer[34].

Pro vytvoření nového, nebo pro přístup k minulosti již vytvořenému souboru množin klíč-hodnota slouží následující metody[24]:

- *getSharedPreferences* — vhodné v případě potřeby více množin klíč-hodnota. Každá je jednoznačně identifikována svým názvem, který je specifikovaný pomocí prvního parametru metody, přičemž dobrou praktikou bývá zvolení jednoznačného názvů napříč všemi aplikacemi. Například každý název může být složen z prefixu a sufixu, kde prefix tvoří název kořenového balíčku aplikace (více v podkapitole 2.3.4 atribut *package*) a sufix pak libovolné, taktéž unikátní, označení. Metoda pomocí druhého parametru přijímá některý z povolených pracovních módů vysvětlených níže.

- *getPreferences* — vrací instanci třídy *SharedPreferences* pro aktuální aktivitu (podkapitola 2.2.1), ve které je metoda volána. Není tedy nutno explicitně specifikovat název

množiny. Ve skutečnosti se jedná o zkrácenou formu volání metody *getSharedPreferences*, kdy se jako první parametr specifikující název množiny použije název třídy aktivity[5].

Obě popisované metody přijímají jako parametr pracovní mód množiny klíč-hodnota, kde pracovním módem rozumíme přístupnost ostatních aplikací k souboru s uloženou množinou. Nejčastěji používaný parametr *PRIVATE* oznamuje privilegovaný přístup k datům. Může být však požadavkem sdílení množiny, pak se použije mód *WORLD_READABLE*, resp. *WORLD_WRITABLE* povolující čtení, resp. zápis ostatních aplikací v systému za předpokladu znalosti identifikátoru (názvu) množiny. Z uvedeného vyplývá použití metody *getPreferences* pouze s privilegovaným přístupem[24].[12]

Modifikace dat se provádí pomocí metod statického rozhraní *SharedPreferences.Editor* získaného voláním *edit* na instanci třídy *SharedPreferences*. Metody modifikace dat jsou volány dávkově. Potvrzení dávky se provede voláním *commit* nebo *apply*. První zmíněná vrací informaci o úspěchu operace potvrzení, která se provádí synchronně, druhá zmíněná asynchronně[26].

2.4.2 Bundle

Instance třídy *Bundle* se často využívá k předávání informací při startu nové aktivity, která nemůže být vytvořena přímo voláním konstruktoru, ale pomocí třídy *Intent*. Objekt typu *Bundle* slouží jako nosič dat ve formátu množiny klíč-hodnota. Voláním metody *putExtra* na instanci třídy *Intent* se provede vložení dat, která pak mohou být využita během procesu vytváření nové aktivity a jejího obsahu[5]. Hodnotou ve dvojici klíč-hodnota může být primitivní datový typ (podobně jako u *SharedPreferences* v podkapitole 2.4.1), ale také libovolný objekt třídy implementující rozhraní *java.io.Serializable*[13] a *android.os.Parcelable*[4].

Neméně častým využitím třídy *Bundle* bývá uložení dat samotnou aktivitou v metodě *onSavedInstanceState*, která jako parametr poskytuje připravenou instanci. Po obnovení aktivity je tato předána do metody *onCreate* (více popis obrázku 2.2 životního cyklu aktivity)[5][18].

2.4.3 Parcelable

Aby mohl být libovolný objekt předáván mezi aktivitami pomocí třídy *Bundle*, musí implementovat rozhraní *android.os.Parcelable*. Zásluhou metody *writeToParcel* je složitější objekt dekomponován na množinu hodnot primitivních datových typů, které jsou postupně ukládány do kontejnerového objektu *android.os.Parcel*. Třída implementující *Parcelable* rozhraní musí povinně vlastnit statický nenulový atribut *CREATOR*, pomocí kterého je naopak složitější objekt komponován z kontejnerového objektu *Parcel*[14][34][19].

2.4.4 SQLite

Jedná se o kompaktní knihovnu poskytující relační databázový systém, který na rozdíl od mnoha jiných databázových systémů není typu klient-server. Předpokládá se využití dat na stejném softwaru a hardwaru, jenž databázi uchovává a obsluhuje. Kompletní SQL databáze s mnoha tabulkami, indexy, triggery a pohledy je uložena pouze v jediném souboru

[12] Uvedené názvy módů neobsahují prefix „*MODE*“

[13] Dokumentace: https://developer.android.com/reference/android/os/Parcelable.html

[14] Fakticky i samotný objekt třídy *Bundle* může být uložen do kontejnerového *Parcel*.

na disku, jehož formát je multiplatformní, tudíž kompletní databáze je lehce přenositelná a zálohovatelná. Databázové transakce za každých okolností splňují vlastnosti ACID[2].

Platforma Android poskytuje API potřebné pro manipulaci s SQLite v balíčku *android.database.sqlite*. Soubor s databází ukládá do soukromého umístění na disku určeného pro právě běžící aplikaci a lokalizovaného v interním úložišti zařízení. Databázová data jsou pak zabezpečena proti neoprávněnému přístupu ostatních aplikací. Není tedy nutné požadovat oprávnění pro zápis na externí úložiště, které se řadí mezi nebezpečná (více v kapitole 2.5). S využitím třídy *SQLiteOpenHelper* lze získat referenci k databázi, a to pomocí metod *getWritableDatabase*, resp. *getReadableDatabase* umožňující čtení a zápis, resp. pouze čtení z databáze. Volání první zmíněné může selhat, např. z důvodu zaplněného paměťového prostoru. Aby byla uložená data v databázi přístupná i v uvedeném situaci, může být využito druhé zmíněné metody. Po uvolnění potřebného místa může být volána *getWritableDatabase* vracející objekt podporující i zápis. Provedení zmíněných bývá časově náročnou operací, proto je vhodné zvážit umístit volání do samostatného vlákna, a to například s využitím třídy *AsyncTask* popsané v kapitole 2.2.4[23].

Aby platilo výše zmíněné, musí být vytvořena vlastní třída dědící od *SQLiteOpenHelper*, v níž bývají přepsány některé z následujících metod získávající jako jeden z parametrů objekt třídy *SQLiteDatabase* poskytující metody pro vyváření, mazání a spouštění SQL příkazů[27]:

- *onCreate* — volána při pokusu o otevření databáze, která doposud nebyla vytvořena. Jedná se o abstraktní metodu, musí být tedy přepsána vždy, přičemž po jejím provedení budou vytvořeny a inicializovány tabulky pomocí uvedených SQL příkazů prováděných nad *SQLiteDatabase* objektem.

- *onUpgrade* — volána v případě potřeby aktualizace databázového schématu v minulosti definovaného pomocí metody *onCreate*. Ve většině případů se bude jednat o zahození stávajících tabulek, vytvoření nových a opětovné naplnění. Požadavkem může být pouze přidání sloupce do již existující databáze, pak lze využít příkaz *ALTER TABLE*. V případě potřeby odebrání sloupce (či více sloupců) je nutné pomocí zmíněného příkazu stávající tabulku přejmenovat, vytvořit novou s požadovaným schématem a posléze ji naplnit daty z přejmenované tabulky, kterou lze následně smazat. Tento mechanismus je důsledkem omezenější funkcionality příkazu *ALTER TABLE* oproti standardu SQL-92[3]. Implementace metody *onUpgrade* je podobně jako u *onCreate* vynucena.

- *onOpen* — volána po případném provedení výše zmíněných metod a značí otevření databáze. V případě potřeby provedení modifikace uložených dat musí být zajištěna kontrola předchozího otevření databáze umožnující zápis. Pro tento účel slouží metoda *isReadOnly* objektu třídy *SQLiteDatabase*.

Konstruktor třídy *SQLiteOpenHelper* přijímá mj. název a verzi databáze. Názvem se rozumí textový identifikátor jednoznačný v rámci vyvíjené aplikace, nikoli celého systému. Verze nabývá celočíselné hodnoty. Jestliže se konstruktoru předá vyšší verze než při posledním volání, pak bude volána metoda *onUpgrade*, v případě nižší potom *onDowngrade*. Tato pokud nebude implementována, bude systémem vyvolána výjimka *SQLiteException*. Každá invokace metody *onUpgrade* a *onDowngrade* bude provedena v rámci jedné transakce.

2.5 Systémová oprávnění

V anglické literatuře zabývající se platformou Android značí pojem *permission* oprávnění, která potřebuje mít aplikace přidělena od operačního systému pro vykonání požadované funkcionality. Specifikace oprávnění se provádí v souboru manifest, popsaného v kapitole 2.3.4, pomocí elementu *uses-permission* umístěného před element *application*[4]. Pokud nebude dodrženo pořadí, veškerá specifikovaná oprávnění budou systémem ignorována.

Od verze API 23 a výše je nutné (pokud nabývá atribut *targetSdkVersion* hodnoty alespoň 23) požadovat oprávnění až za běhu aplikace, a to ideálně až v případě skutečné potřeby. Uživatel tedy před instalací aplikace nepotvrzuje přidělení žádných oprávnění, jako tomu bylo až do API verze maximálně 22. Pokud uživatel jednou přidělí aplikaci určitá oprávnění, má jej možnost odebrat prostřednictvím správce aplikací v nastavení telefonu nebo reinstalací celé aplikace. Programátor tedy vždy před provedením akce musí dostupnost oprávnění kontrolovat. V negativním případě dostupnosti bude vyvolána výjimka *SecurityExcepton*[4].

Systémová oprávnění se dělí do několika úrovní, přičemž zjednodušené a nejdůležitější dělení je právě do dvou následujících skupin[22]:

- Normální (*normal*) — skupina oprávnění pokrývající oblasti, do kterých aplikace potřebuje přistupovat mimo svůj vyhrazený prostor působnosti (anglický výraz *sandbox*). Jedná se o data a zdroje s nízkým rizikem zneužití, a to hlavně ve smyslu ochrany soukromí uživatele nebo dat ostatních aplikací nacházející se na tomtéž zařízení. Mezi normální oprávnění patří například přístup k internetovému připojení, použití vibračního modulu, dotaz na zjištění instalovaných aplikací a stav Wi-Fi. Kompletní seznam lze dohledat v citované literatuře. Tato oprávnění musí, jak již bylo řečeno, býti uvedena v souboru manifest. Ovšem oproti nebezpečným oprávněním budou systémem přidělena na vyžádání aplikace bez spoluúčasti uživatele.

- Nebezpečná (*dangerous*) — skupina s vysokým rizikem zneužití soukromých dat ostatních aplikací nebo uživatele samotného. Mezi nebezpečná oprávnění patří čtení či zápis kalendáře, externího úložiště, manipulaci s fotoaparátem (pokud jej zařízení vlastní), přístup k přibližné i přesné poloze a jakákoli manipulace s telefonními hovory. Kompletní výčet je rovněž přístupný v literatuře. Pro přidělení nebezpečných oprávnění je nutná spoluúčast uživatele, jak bylo popsáno výše v této podkapitole[35].

Kapitola 3

Stávající aplikace

Obsah kapitoly seznamuje čtenáře se současnými trendy webových a mobilních aplikací z oblasti podpory tělesně postižených. Detailněji popisuje webovou aplikaci města Brna poskytující informace o vybraných objektech. Současně jsou popsány webové služby, které tato využívá. Následně budou vybráni tři zástupci mobilních aplikací pro podporu tělesně postižených společně s popisem principů fungování a způsobu sběru dat. Nechybí taktéž zástupci z oblasti mobilních aplikací pro účely navigace.

3.1 Bezbariérové objekty města Brna

Magistrát města Brna poskytuje svým obyvatelům a široké veřejnosti webovou aplikaci zobrazující informace o vybraných budovách města Brna, včetně údaje o přístupnosti využitelné osobami s tělesným postižením. Její vzhled můžeme vidět na obrázku 3.1. Uživatelské rozhraní se skládá z části zobrazující mapový podklad a části s akčními tlačítky, jinými slovy menu. Magistrát města Brna na vývoji spolupracuje se společností T-Mapy[1]. Jako mapový podklad využívá data poskytovaná mezinárodní neziskovou organizací *OpenStreetMap*[2] ze stejnojmenného projektu, poskytující geografická data pod licencí *ODBL v1.0* povolující sdílení, modifikaci, celkového volného využití dat, a to bez jakékoli záruky. Webovou aplikaci lze využívat v libovolném webovém prohlížeči po otevření následujícího odkazu:

http://gismb.tmapserver.cz/mapa/bezbarierove-objekty

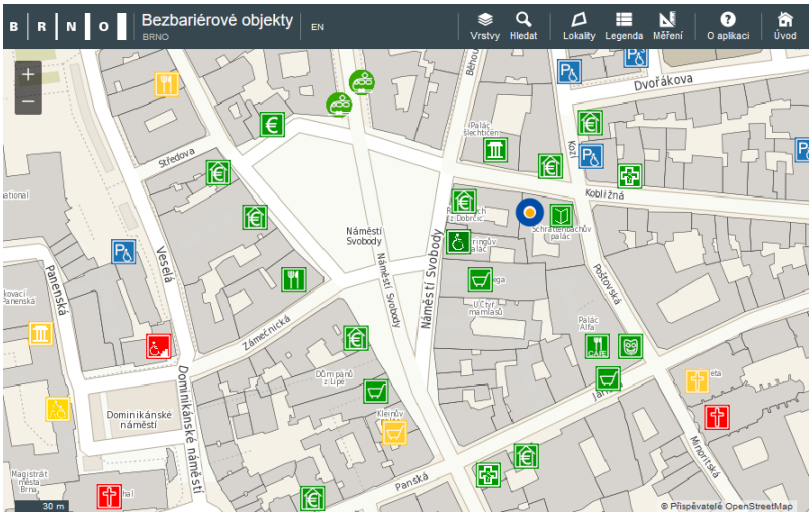
3.1.1 Funkce aplikace

Navštívením výše zmíněného odkazu je uživateli zobrazen mapový podklad s vykreslenými piktogramy reprezentující vybrané objekty a dále aplikační menu s následujícími položkami:

- Vrstvy — poskytuje volbu podkladových a překryvných vrstev hlavního okna aplikace. U první zmíněné má uživatel na výběr mezi podkladovou vrstvou tvořenou leteckými snímky nebo podkladem *OpenStreetMap* s možností specifikace průhlednosti. U druhé zmíněné, jak lze pozorovat na obrázku 3.2b, je umožněna pomocí zaškrtávacích políček volba zobrazení či skrytí vrstev kategorizovaných do třech kategorií. Vrstva Budovy obsahuje podkategorie veřejně přístupných staveb například z oblastí

[1] T-Mapy spol. s r.o., webová prezentace společnosti - www.tmapy.cz

[2] OpenStreetMap Foundation - http://wiki.osmfoundation.org/wiki/Main_Page [Online; navštíveno 04.01.2017]



Obrázek 3.1: Ukázka webové aplikace Bezbariérové objekty

zdravotnictví, kultury, státní správy a restaurací. Vrstva Doprava obsahuje některé objekty veřejné či automobilové dopravy. Poslední vrstvou je Podklad. Po zvolení podkategorie Kostely a kulturní objekty budou do mapy vykresleny pouze polygony značící území výskytu těchto objektů. Pomocí poslední podkategorie ve vrstvě Podklad (není viditelná na obrázku 3.2b) budou vyobrazeny vstupy do některých budov, sklony komunikací a umístění schodišť na chodnících. Tato podkategorie obsahuje jen velmi malé množství objektů fyzicky umístěných v blízkém okolí centra města Brna.

- Hledat — nabízí funkci *fulltextového* vyhledávání budov obsažených ve vrstvách popsaných v předchozím bodě. Pokud je daná budova úspěšně nalezena, bude její umístění označeno pomocí pinu na aktuální podkladové vrstvě. Druhou možností je vyhledávání dle zadané adresy, a to jak celkové, tak její části. Eventuálně lze vyhledat pouze ulici na základě jejího názvu.

- Lokality — zobrazení panelu se seznamem městských částí města Brna společně se vstupním elementem sloužícím pro snadné filtrování záznamů ve zmíněném seznamu. Při výběru libovolné položky nastává přesun grafického zobrazení mapy nad zvolenou městskou část s pevně nastaveným měřítkem.

- Legenda — informační klíč k pochopení významu piktogramů znázorňujících vybrané budovy města Brna. Jedná se o obdobu legendy v mapových dílech.

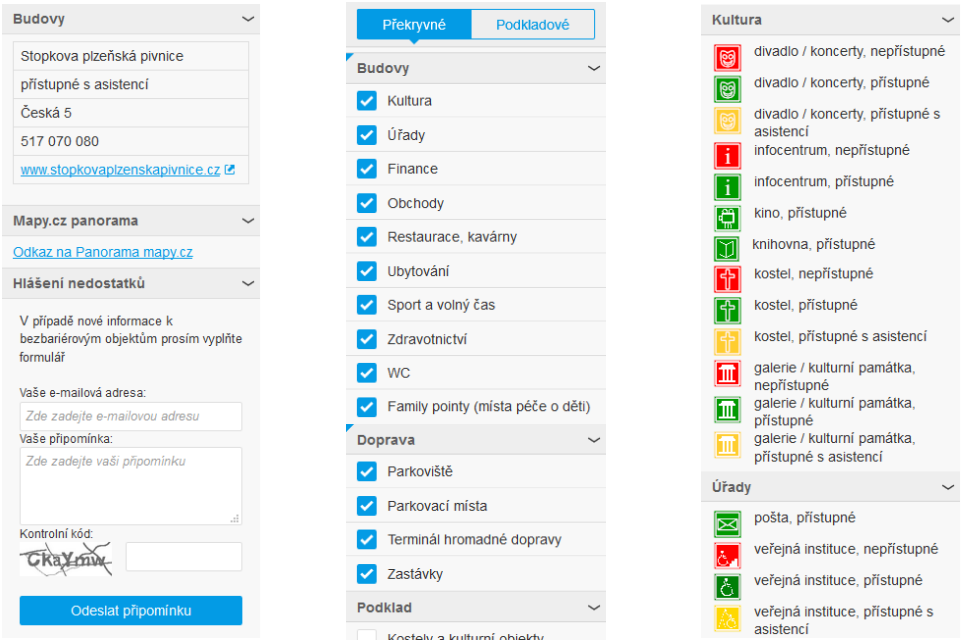
- Měření — zpřístupní funkci měření vzdálenosti, obvodu či plochy. Vstupní data ve formě bodů uživatel zadává interakcí s mapovým dílem.

- O aplikaci — zobrazí panel informace a nápověda webové aplikace.

- Úvod — přesměruje uživatele na webovou prezentaci s rozcestníkem mapových nástrojů a aplikací Magistrátu města Brna[3].

Uživatel může dále pomocí menu volit jazyk webové aplikace a informací o budovách. Dostupnými jazyky jsou český a anglický. Po výběru piktogramu na mapovém podkladu

[3]Rozcestník nástrojů – http://gismb.tmapserver.cz [Online; navštíveno 04.01.2017]



(a) Detail objektu (b) Výběr vrstev (c) Legenda

Obrázek 3.2: Možnosti webové aplikace GIS Brno

reprezentující budovu se zobrazí panel s dostupnými informacemi o adrese a přístupnosti, jak je vidět na obrázku 3.2a. Uživatel má rovněž možnost pomocí jednoduchého formuláře zaslat připomínku k poskytovaným informacím.

Samotná práce s webovou aplikací je přímočará. Interakce probíhá s využitím myši, pomineme-li použití klávesnice pro vyplnění vstupních polí během vyhledávání a případného zasílání připomínky. Velkým nedostatkem se může jevit skrytí všech piktogramů na mapovém podkladu již při relativně nízké úrovni oddálení. Tato skutečnost nutí uživatele pracovat s aplikací při vyšších hodnotách přiblížení, což může mít za následek špatnou orientaci v mapovém díle, zvláště pak pro nemístní část uživatelů.

3.1.2 Podpůrné služby

Popsaná webová aplikace ke své činnosti využívá *ArcGIS REST API*, pomocí kterého získává data o vybraných objektech města Brna. Zmíněné API může být bez poplatků použito k libovolným účelům. Adresář služeb je dostupný prostřednictvím následujícího odkazu:
http://gis.brno.cz/arcgis/rest/services/PUBLIC/bezbarierove_objekty/MapServer

Dotaz na server poskytující API je posílán prostřednictvím HTTP protokolu metodou GET v následujícím tvaru URL adresy `{api_url}/{id}/query?{query_url}`, kde:

- `{api_url}` je nahrazeno URL adresou API serveru,
- `{id}` je nahrazeno číselným identifikátorem požadované vrstvy získané z tabulky 3.1,
- `{query_url}` je nahrazeno omezením pro dotaz.

Například, výsledná URL pro získání všech objektů vrstvy Kultura ve formátu JSON se všemi atributy popisující daný objekt bude ve tvaru:

Kořenová vrstva	Název vrstvy	ID	Typ geometrie
Objekty	Kultura	1	Bod
	Úřady	2	Bod
	Finance	3	Bod
	Obchody	4	Bod
	Restaurace, kavárny	5	Bod
	Ubytování	6	Bod
	Sport a volný čas	7	Bod
	Zdravotnictví	8	Bod
	WC	9	Bod
	Family pointy	10	Bod
	Kulturní objekty	11	Polygon
	Kostely	12	Polygon
Podklad	Vstup	14	Bod
	Schody	15	Polygon
	Sklony	16	Polygon
Doprava	Zastávky	18	Bod
	Terminál hromadné dopravy	22	Bod
	Parkovací místa	19	Bod
	Parkoviště	20	Bod
Budovy_all		21	Bod

Tabulka 3.1: Dostupné vrstvy ArcGis REST API

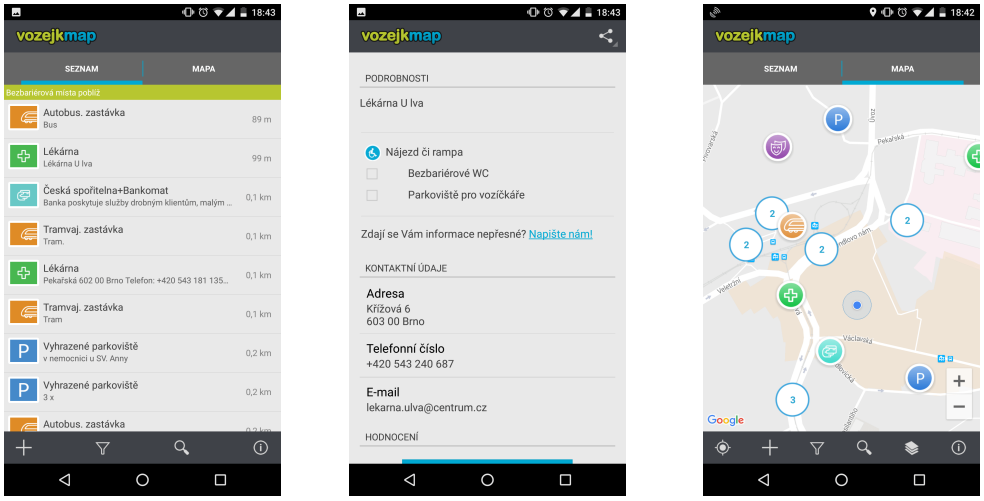
http://gis.brno.cz/arcgis/rest/services/PUBLIC/bezbarierove_objekty/MapServer/1/query?where=1%3D1&outFields=*&f=pjson.

Již byla zmíněna tabulka 3.1. Pro každou nekořenovou vrstvu jednoznačně identifikovatelnou hodnotou číselného identifikátoru ID je dovoleno provádět dotazy. Výjimkou je kořenová vrstva Budovy_all, jak již z názvu plyne, navracející všechny budovy dostupné v systému. U každé vrstvy je uveden typ geometrie objektů v odpovědi na dotaz nabývající hodnoty bod nebo polygon.

Formát odpovědi

Poskytované API umožňuje zasílat textovou odpověď v pěti různých formátech: HTML, JSON (JavaScript Object Notation), AMF (*Action Message Format*), KMZ (komprimované *Keyhole Markup Language* soubory), GeoJSON (založen na JSON). V odpovědi se vyskytují symboly s českou diakritikou, proto je vhodně použito kódování UTF-8. V dalším textu bude rozebrán JSON formát odpovědi. Pro příklad nyní uvažujme dotaz pro získání všech objektů vrstvy Kultura, pro který již byla URL dříve uvedena. Textovou odpovědí pak bude právě jeden objekt ve formátu JSON s následujícími indexy:

- *displayFieldName* — udává název indexu libovolného objektu z kolekce objektů *features*, který mám být v aplikaci použit jako název daného objektu (v našem příkladu objekt reprezentuje budovu z vrstvy Kultura).
- *fieldAliases* — aliasy indexů v kolekci objektů *fields*. Každý index má totožný alias, v budoucnu tedy není třeba aliasy uvažovat.



(a) Seznam míst (b) Detail místa (c) Mapa míst

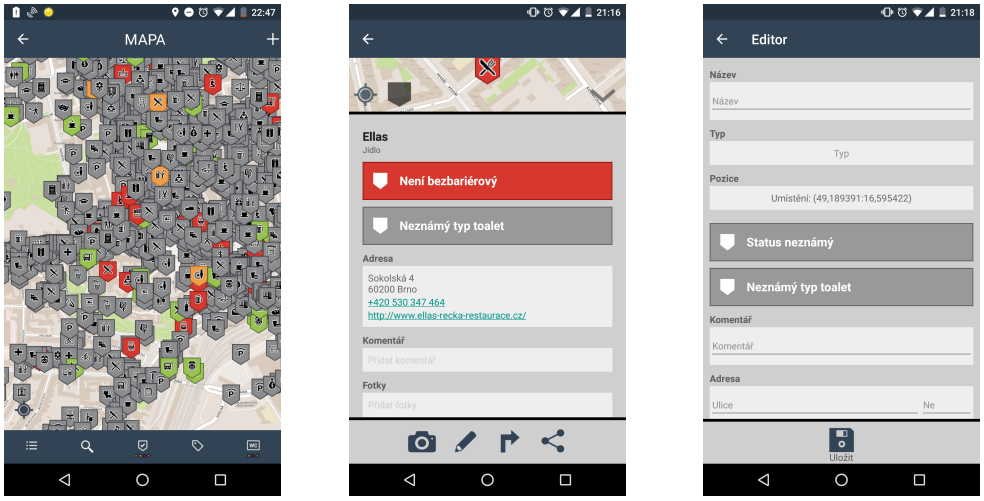
Obrázek 3.3: Ukázka uživatelského rozhraní Android aplikace VozejkMap

- *geometryType* — může nabývat dvou hodnot - *esriGeometryPoint*, resp. *esriGeometryPolygon* značící bod, resp. polygon (v tabulce 3.1 sloupec Typ geometrie).
- *spatialReference* — kódy EPSG a ESRI definující použitý souřadnicový systém. Kódy nabývají vždy stejné hodnoty značící Systém Jednotké Trigonometrické Sítě Katastrální (S-JTSK).
- *features* — obsahuje hlavní část odpovědi – pole objektů společně s geometrií (v případě *esriGeometryPoint* jedna souřadnice, jinak kolekce souřadnic), na které byl proveden dotaz. V našem případě *features* obsahuje kolekci objektů (budov) obsažené ve vrstvě Kultura.
- *fields* — dodatečné informace o objektech v kolekci *features* – název indexu, alias (stejný jako je tomu u *fieldAliases*) a datový typ.

Formát odpovědi se zdá býti dosti komplexní. V reálné situaci lze vystačit pouze s hodnotami obsažené v atributu *features*. Struktura odpovědi připomíná syntaxi GeoJSON, avšak nejedná se o ni. Výše zmíněný příklad předpokládá *geometryType* s hodnotou *esriGeometryPoint*. V případě *esriGeometryPolygon* bude formát hlavní části odpovědi *features* nabývat mírně odlišné struktury.

3.2 Podpora tělesně postižených

Podkapitola stručně představuje tři vybrané projekty, na jejichž principu staví většina mobilních aplikací pro podporu tělesně postižených. Během volby zástupných projektů bylo jako hlavní kritérium výběru zvoleno oblast působnosti, tj. nadnárodní, národní a lokální (městská) působnost. V úvahu byly brány pouze aplikace určené pro platformu Android přístupné prostřednictvím distribuční služby Google Play.



(a) Mapa míst (b) Detail místa (c) Přidání místa

Obrázek 3.4: Ukázka uživatelského rozhraní Android aplikace Wheelmap

3.2.1 VozejkMap – Svoboda pohybu na vozíku

Projekt VozejkMap[4] provozuje Česká asociace paraplegiků *CZEPA*[5]. K dispozici je webová i mobilní aplikace pro platformy Android a iOS, užívání libovolné je zcela zdarma. Aplikace si klade za cíl poskytovat informace o bezbarierových místech po celé České republice. Data vkládají samotní uživatelé, přičemž po založení nového záznamu tento prochází kontrolou administrátorem na možnou duplicitu a relevantnost obsažených informací.

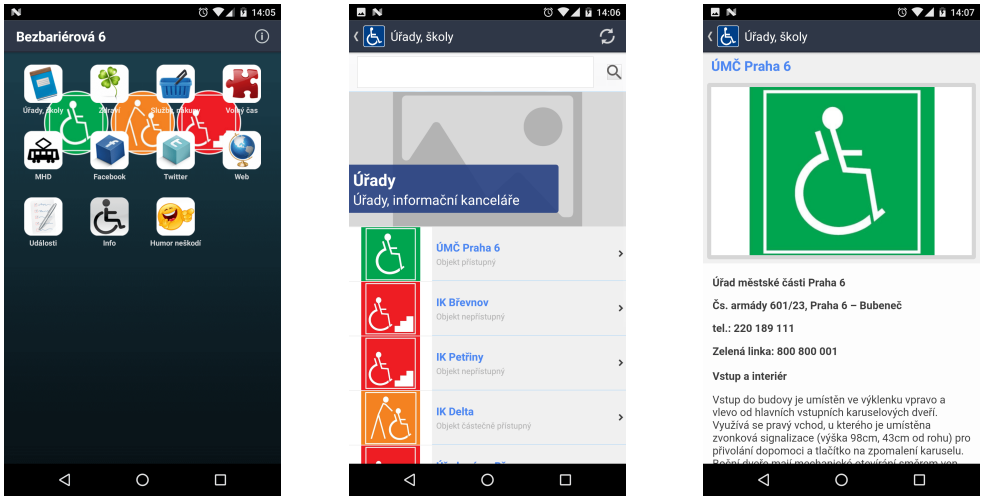
Uživatelské rozhraní obsahuje seznam nejbližších míst vůči aktuální poloze mobilního zařízení (obrázek 3.3a) nebo lze tato zobrazit na mapovém podkladu poskytované společností Google. V situaci vysokého počtu objektů na podkladové mapě jsou místa slučována a nahrazována zástupnými symboly, tzv. clustery, jak je patrné z obrázku 3.3c. Uživatel má možnost přidávat místa nová, komentovat a hodnotit stávající pomocí stupnice 1 až 5 reprezentované symboly hvězdičky. Obrázek 3.3b znázorňuje detail vybraného místa (ze seznamu nebo mapy), ve kterém uživatel mj. nalezne možnost spuštění externí aplikace plnící funkci navigace, která však na daném zařízení nemusí být dostupná.

Celý projekt využívá data získávaná od samotných uživatelů, navíc tato zpřístupňuje ve formátu JSON třetím osobám pod licencí *Creative Commons CC BY-NC-SA 4.0* při dodržení licenčních podmínek, zejména nekomerční účely s uvedení zdroje.

3.2.2 Wheelmap

Za projektem Wheelmap[6] stojí nezisková organizace *Sozialhelden*[7]. Aplikace Wheelmap je dostupná jako webová a mobilní (Android i iOS). Dokonce nechybí podpora i pro operační systém Windows. Princip je mírně odlišný oproti aplikaci VozejkMap (kapitola 3.2.1), ve které veškerá data vkládají samotní uživatelé. Aplikace Wheelmap čerpá potřebné informace o místech POI (zkratka anglického výrazu *Point Of Interest*) z již vytvořené databáze

[4]Domovská stránka VozejkMap – http://www.vozejkmap.cz
[5]Domovská stránka CZEPA – http://www.czepa.cz
[6]Domovská stránka Whellmap – https://wheelmap.org
[7]Domovská stránka Sozialhelden – http://sozialhelden.de/en



(a) Hlavní obrazovka (b) Detail podkategorie (c) Detail místa

Obrázek 3.5: Ukázka uživatelského rozhraní Android aplikace Bezbariérová 6

vlastněné *OpenStreetMap* pod licencí *Open Database Licence* s více než 130 druhy POI. VozejkMap se zaměřuje na území České republiky, kdežto Wheelmap cílí na uživatele z celého světa.

Android aplikace dovoluje zobrazit mapu s vyobrazenými místy, která jsou vykreslena nad aktuálně zobrazenou oblastí mapového díla. Vykreslování míst je dále omezeno od určité úrovně zoomu. Na obrázku 3.4a lze pozorovat nedostatek, a to chybějící slučování ikon mající za následek sníženou čitelnost mapy a také horší výkonnost. Obrázek 3.4b, resp. 3.4c ukazuje obrazovky detailu, resp. přidaní nového místa. Stejně jako u *VozejkMap* je dostupná funkce navigace pomocí externí aplikace s uvedenými možnými problémy.

3.2.3 Bezbariérová 6

Projekt *Šestka bez bariér*[8] dává základ pro webovou i mobilní aplikaci fungující na platformě Android Bezbariérová 6. Aplikace mapují přístupnost vybraných objektů městské části Prahy 6. Objekty mapovali specialisté z *Pražské organizace vozíčkářů*[9], která se této problematice věnuje téměř dvacet let.

Android aplikace umožňuje zobrazit vybrané objekty rozčleněné do čtyř kategorií obsahující úřady, školská i zdravotnická zařízení, lékárny a prodejny zdravotnických pomůcek, pošty a mnoho další. Situaci lze pozorovat na snímku hlavní obrazovky aplikace 3.5a. Každá kategorie je dále členěna do podkategorií. Příklad detailu podkategorie je zachycen na obrázku 3.5b. Po vybraní libovolného objektu z podkategorie bude zobrazen jeho detail (obrázek 3.5c) obsahující informaci o přístupnosti, adrese a detailnímu popisu vytvořeného zmíněnou Pražskou organizací vozíčkářů.

Nedostatkem může být chybějící možnost vykreslení objektů na mapovém podkladu, který částečně kompenzuje dostupné fulltextové vyhledávání procházející objekty všech kategorií. Webová aplikace trpí stejným nedostatkem.

[8] Domovská stránka Šestka bez bariér – http://www.bezbarierova6.cz
[9] Domovská stránka Pražské organizace vozíčkářů – http://www.pov.cz

3.3 Navigace v městech pomocí mobilních aplikací

V rámci zkoumání velmi malé části obsahu distribuční služby Google Play nebyla nalezena žádná aplikace specializující se výhradně na oblast navigace v rámci konkrétního města. Z této skutečnosti plyne popis vybraných aplikací fungující na území celých států či kontinentů s přihlédnutím k dostupným parametrům, jako je počet stažení[10], datum poslední aktualizace a celkové skóre dané hodnocením zadaných od uživatelů. Z prostorových důvodů nejsou k dispozici ukázky vzhledu aplikací.

3.3.1 Maps

Samotná společnost Google, která je provozovatelem služby Google Play, nabízí aplikaci Mapy, jejíž název byl až do letošního roku poněkud výstižnější, a to *Google Maps*. Aplikace se může pyšnit úctyhodným počtem stažení – 1 až 5 miliard. Důvodem tak vysokého počtu může být fakt, kdy zmíněná aplikace je již předinstalována v základní distribuci platformy Android.

Aplikace jako jediná z uvedených poskytuje informace o bezbariérovém přístupu k objektům, pokud je tato dostupná. Její dostupnost lze ověřit v libovolném detailu místa POI. Informace je získávána z výsledků odpovědí na sadu otázek, které poskytují uživatelé přihlášení v programu Místní průvodci[11]. Přispěvatelem se může stát každý uživatel, který nemusí mít potřebné znalosti k rozhodnutí o bezbariérovém přístupu (oproti aplikacím v naprosté většině využívající komunitou osob s tělesným postižením popsaných v podkapitole 3.2), proto pravdivost informace musí být brána s určitou rezervou.

Samotná navigace je dostupná pro více dopravních prostředků, jako automobil, veřejná doprava a nechybí také možnost volby chůze. V naprosté většině případů bývá před startem navigace uživateli nabídnuta hlavní trasa a maximálně dvě alternativní trasy, jejichž počet se liší v závislosti na vzdálenosti mezi startovním a koncovým bodem trasy. Samozřejmostí jsou aktuální dopravní informace částečně portované z aplikace *Waze*, jejíž popis následuje.

3.3.2 Waze

Funguje na principu komunitní sítě uživatelů, kteří poskytují aktuální informace o dopravní situaci formou hlášení dopravních nehod, uzavírek silnic, policejních radarů, nebezpečí na silnici a mnoho další. Uživatel vytvoří záznam, ostatní uživatelé projíždějící místem v jeho blízkosti jej mohou potvrdit či vyvrátit. Aplikace pak plánuje a optimalizuje trasy s ohledem na relevantní záznamy. V závislosti na možnostech hlášení událostí týkající se pouze automobilové dopravy není aplikace určena pro pěší a jiné druhy navigací. Z principu funkce se uživatel neobjede bez dostupného internetového připojení, přičemž užívání aplikace je zcela zdarma. Počet instalací (až půl miliardy) napovídá o velké popularitě.

3.3.3 HERE WeGo

Poslední popisovanou aplikací je *HERE WeGo* pyšnící se 10 až 50 miliony stažení. Umožňuje plánování trasy pomocí vlastního či cizího (sdílení) automobilu, městskou hromadnou dopravou, taxi, jízdou na kole nebo chůzí. Aplikace byla testována v centru města Brna, kde bohužel nebylo dostupné plánování trasy pomocí veřejné dopravy, sdílení automobilu

[10] Distribuční služba Google Play neposkytuje přesný počet stažení, tento má dostupný pouze vydavatel aplikace.

[11] Domovská stránka programu Místní průvodci – https://www.google.com/intl/cs/local/guides

a taxi. Podobně jako *Maps* popsaná v sekci 3.3.1 umožňuje funkci zobrazení aktuální dopravní situace pomocí dat poskytované aplikací *Waze*. Výsledky plánování pěší trasy jsou mírně odlišné oproti výsledkům z aplikace *Maps*. *HERE WeGo* jako jediná z uvedených poskytuje možnost stažení mapového podkladu do zařízení.

3.4 Shrnutí poznatků

V kapitole byly popsány stávající aplikace pro podporu tělesně postižených. Při zjednodušeném pohledu na problém existují aplikace dvojího druhu podle typu dat, se kterými manipulují a nabízí uživatelům k užitku. Prvním typem jsou aplikace pracující s daty zadávanými a spravovanými komunitou uživatelů, zaměřující se na území s různě velkou rozlohou. Představiteli jsou VozejkMap (podkapitola 3.2.1) nabízející informace o objektech na území převážně České republiky a Wheelmap (podkapitola 3.2.2) snažící se pokrýt území potenciálně celého světa (v závislosti na dostupných mapových podkladech projektu *OpenStreetMap*). Druhým typem jsou aplikace pracující pouze s krátkodobého hlediska neměnnými daty. Příkladem je *Bezbariérová 6* popsaná v podkapitole 3.2.3 poskytující informace o objektech na území městské části Praha 6 a *Bezbariérové objekty města Brna* popsané v podkapitole 3.1 dostupná pouze pro webové prohlížeče obsahující budovy města Brna. V okrajové části leží aplikace *Maps* (podkapitola 3.3.1), s jejíž pomocí uživatelé zapojeni do programu Místní průvodci odpovídají na nejrůznější množinu otázek vázající se k různorodým místům zájmu (POI). V této množině se může objevit otázka na bezbariérovost přístupu k danému místu.

Aplikace prvního typu obsahují větší množství dat na potenciálně velkém území, avšak jejich korektnost nemusí být vždy zaručena. Oproti tomu druhý typ aplikací poskytuje malé množství dat vázající se k menší rozloze území. Data jsou však přesná, autory jsou proškolení odborníci s kompetencemi takové informace vytvářet a publikovat. Aktuálnost dat však nemusí být vždy zaručena.

Část výsledného produktu diplomové práce si klade za cíl převzít výhody z obou zmíněných typů a minimalizovat možné negativní rysy taktéž převzaté.

Kapitola 4

Analýza požadavků a návrh

Pro účel získání části požadavků kladených na projekt byly použity jedny z tradičních technik, a to studium softwarových systémů (aplikací) popsaných v kapitole 3, dále pak několik spíše neformálních interview se zástupcem Odboru zdraví Magistrátu města Brna, společností T-Mapy a přiděleným technickým konzultantem z Odboru městské informatiky, oddělení geografických informačních systémů. Na základě získaných poznatků byla vytvořena textová specifikace požadavků kladených na budoucí produkt, byl vytvořen diagram případů použití následovaný stručným popisem jednotlivých případů. Taktéž nechybí návrh uživatelského rozhraní výsledného produktu.

4.1 Neformální specifikace

Primárními uživateli produktu budou osoby s tělesným postižením, kde pojem tělesní postižení redukujeme na postižení pohybového ústrojí. Zejména se jedná o osoby využívající invalidní vozík. Sekundárními uživateli mohou být osoby s lehčí formou pohybového postižení, téže dočasnou. Například se může jednat o uživatele berlí a holí, pro něž kvalita povrchu komunikací (viz dále) není nepodstatným faktem. Konečně poslední sekundární uživatel může být rodič s kočárkem. Rozlišení primárního a sekundárního uživatele nebude uvažováno, avšak bude kladen důraz na použitelnost převážně primárním (dále v textu označen jako uživatel).

Výsledný produkt by měl do jisté míry kopírovat část funkcionality webové aplikace Bezbariérové objekty města Brna popsané v kapitole 3.1. Produkt musí vhodným a přehledným způsobem zobrazit vybrané objekty nacházející se na území města Brna. Každý objekt bude ctít zavedenou grafickou konvenci (vůči stávajícímu řešení). Jako nejvhodnější zobrazení se jeví využití mapového díla, přičemž preferující, avšak ne nutná podmínka, je využití mapových podkladů *OpenStreetMap*. Použití mapového díla od společnosti Google rovněž není vyloučeno. Objektem se rozumí budova nacházející se pouze na území města Brna, jiné objekty není třeba uvažovat. Budoucí uživatelské rozhraní produktu zobrazující mapové dílo by mělo být vhodně omezeno, a to ve smyslu zamezení přístupu uživatele mimo katastrální území města Brna. Objekty tedy budou zobrazovány na zvoleném mapovém podkladu.

O každém objektu musí produkt uchovávat jeho parametry, kterými jsou přístupnost, poloha, úplná adresa, kategorie a typ objektu, adresu webové prezentace a kontaktní telefon. Během zobrazení objektů na mapovém podkladu bude uživateli přístupná funkcionalita filtrace na úrovni kategorie objektů. Jako rozšíření lze uvažovat jemnější filtraci, a to dle typu objektu nebo jakéhokoli jiného parametru. V neposlední řadě objekt může vlastnit parametr

informace, jehož hodnota nabývá detailního popisu přístupnosti. Doposud a v budoucnu uvedené parametry nejsou povinné s výjimkou názvu a polohy. Při absenci libovolného parametru musí produkt vhodně reagovat. Lokací se rozumí souřadnice nabývající hodnotu v zatím blíže nespecifikovaném souřadném systému. Parametr přístupnost může nabývat třech hodnot, a to přístupné, nepřístupné a přístupné s asistencí. Jiné přístupnosti nebudou a nemusí být uvažovány. Objekt dále vlastní atribut WC, jenž vždy nabývá pravdivostní hodnoty udávající umístnění bezbariérové toalety v rámci objektu. Posledním atributem jsou tzv. piktogramy (respektive kolekce piktogramů), kterých existuje právě třináct. Jsou jimi následující: Obtížný povrch, obtížný sklon, bezbariérový vstup hlavním vchodem, bezbariérový vstup bočním vchodem, shody, točité schodiště, výtah, plošina nebo výtah jen pro osoby s omezenou schopností pohybu, úzké dveře nebo průjezdy, přístupná toaleta, částečně přístupná toaleta, nepřístupná toaleta. Informace o piktogramech náležící objektu nemusí být vždy dostupné. V případě kladné dostupnosti budou piktogramy reprezentovány grafickou formou společně s textovým popiskem. Grafická forma znamená zobrazení ikony v jednom ze dvou odstínů, a to světlejší a tmavší. První odstín je reprezentován kladnou pravdivostní hodnotou, opačně tomu je v případě druhého odstínu. Vhodné zobrazení výše zmíněných parametrů označujeme jako detail objektu. Zdrojem pro získání objektů a jejich parametrů budou podpůrné služby popsané v kapitole 3.1.2. Podpora přidávání nových objektů není vyžadována.

Ke každému objektu bude umožněno přidávat libovolné množství komentářů. Uvažovanou formou je pouze čistý text. Komentář zadává uživatel aplikace, proto jsou tyto dále označovány jako uživatelské komentáře. Detail objektu bude obsahovat výpis případných již zadaných komentářů. Text komentáře bude dovoleno upravovat pouze zadavateli, kde zadavatel je autorem komentáře.

Projekt bude dále umožňovat práci s bariérami, kde bariérou rozumíme překážku nacházející se na veřejném prostranství města Brna (pojmy bariéra a překážka mohou být zaměňovány). O bariéře bude třeba uchovávat parametry, kterými jsou poloha, doba platnosti, překonatelnost, komentář a kategorie. Platnost bariéry může být dvojího druhu, a to dočasná nebo trvalá. V prvním případě je nutno uchovat časové ohraničení doby platnosti, přičemž začátek se uvažuje aktuální datum. Časová pásma není třeba uvažovat. Ve druhém případě stačí uchovat pouze informaci o platnosti trvalé. Budou rozlišovány dva druhy překonatelnosti bariéry (nepřekonatelné a překonatelné s asistencí). Komentářem bude, stejně jako v případě budovy, uvažována pouze textová hodnota. Poloha bariéry bude pro zjednodušení reprezentována jako bod umístěný v souřadném systému WGS-84. Jako rozšíření může být implementována podpora přidávání fotografie k bariéře. Možné kategorie bariéry jsou vyjmenovány v podkapitole 4.1.1. Práce s bariérami sestává z přidání, úpravě a zobrazení jejího detailu. Úprava parametrů bariéry je dovolena pouze zadavateli.

Ke každé bariéře mohou uživatelé přidávat textové komentáře, zadavatel jej může libovolně upravovat a není třeba uvažovat platnost komentáře, která je odvozena od platnosti bariéry.

Bude existovat možnost hodnotit existující a zároveň platnou bariéru. Uživatel může libovolnou překážku ohodnotit pouze jednou. Při opakovaném hodnocení téže překážky bude původní nahrazeno. Hodnocení může být kladné, resp. záporné označující potvrzení, resp. vyvrácení existence překážky. V zásadě se jedná o obdobu funkcionality poskytované aplikací Waze popsané v podkapitole 3.3.2.

V projektu by měla existovat možnost plánování tras na základně aktuálního stavu databáze s bariérami. Trasa jako taková sestává z počátečního a koncového bodu, není po-

žadováno přidávání průjezdných bodů. Základem mohou být trasy pro pěší. Bližší informace budou poskytnuty v kapitole popisující implementaci.

V zásadě není vyžadována registrace a přihlášení uživatele. Ten může libovolně procházet obsah projektu. Avšak jakoukoli činnost, v jejíž důsledku dochází ke změně dat, může provádět pouze přihlášený uživatel.

4.1.1 Kategorie překážek

Na základě výsledků diskuze se zástupcem Odboru zdraví (oddělení prorodinné politiky) Magistrátu města Brna dne 8. února 2017 byly specifikovány následující kategorie bariér dále členěné do podkategorií:

- Stavební — obsahující podkategorie díra v chodníku, lešení, úzký průjezd, velký sklon, nevhodný povrch, schody na chodníku.
- Předmět — obsahující podkategorie reklamní vývěska, kontejner na odpadky, odpadkový koš, poštovní schránka, lavička, zábora (ohrazení výkopu), prodejní stánek, hospodská zahrádka (letní), zaparkovaný automobil.

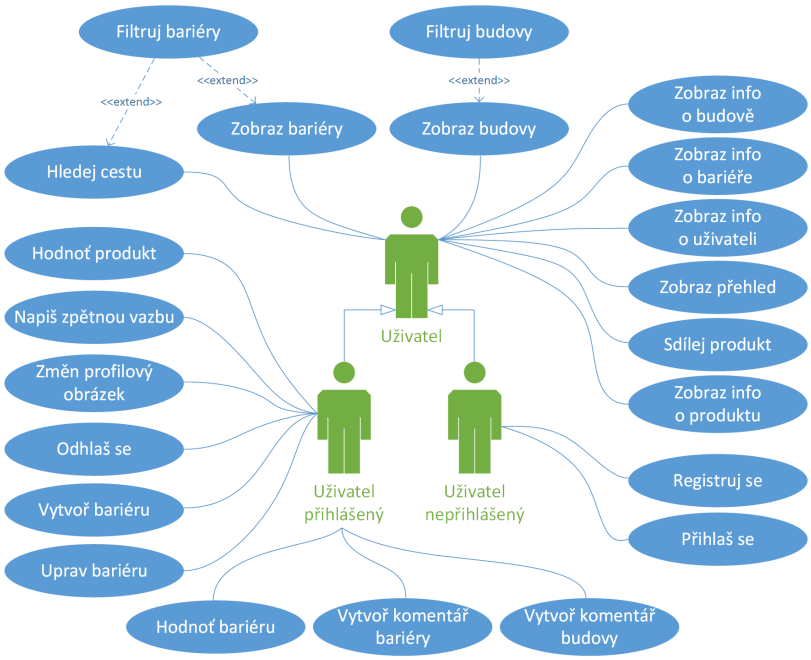
Každá kategorie obsahuje kromě výše uvedených i podkategorii „jiné“. Žádné další kategorie či podkategorie není třeba uvažovat.

4.2 Analýza požadavků

Podkapitola 4.1 obsahuje text neformální specifikace požadavků kladených na výsledný produkt. Produktem má být mobilní aplikace. Výběr optimální platformy souvisí s její rozšiřitelností na mobilních zařízeních a tabletech po celém světě, nevyjímaje území České republiky. Z důvodů pokrytí co možná největšího počtu potenciálních uživatelů byla vybrána platforma Android.

S přihlédnutím na obsah podkapitoly neformální specifikace 4.1 a dalších závěrů provedených interview plynou následující požadavky na výsledný produkt:

- Aktéři — primárními uživateli produktu mají být osoby s pohybovým postižením. Existují však i sekundární aktéři, jako např. rodič s kočárkem. Rozlišení primárního a sekundárního aktéra nemusí být uvažováno. Se systémem tedy bude pracovat pouze jediný aktér, kterým je uživatel. Z neformální specifikace dále plyne možnost použití produktu uživatelem přihlášeným a nepřihlášeným. Nepřihlášenému uživateli je umožněno prohlížet obsah, avšak modifikace záznamů bude povolena pouze uživateli v minulosti registrovanému a momentálně přihlášenému. Z uvedeného plyne zavedení abstraktního aktéra *Uživatel*, od kterého budou dědit společné případy použití konkrétní aktéři, tj. *Uživatel přihlášený* a *Uživatel nepřihlášený*.
- Funkční požadavky — blíže budou analyzovány v podkapitole 4.2.1.
- Výkonnostní požadavky — používání produktu aplikace by mělo být možné na obyčejném mobilním zařízení vlastnící operačním systémem Android s minimální verzí 4.0.3 (více tabulka 2.1). Z toho plyne v omezené míře zaručení zpětné kompatibility, avšak při případném subjektivním hodnocení výkonu produktu na starších zařízeních vlastnící nedostatečně výkonný hardware bude brán na tuto skutečnost patřičný zřetel.



Obrázek 4.1: Diagram případů použití

- Požadavky na podporovatelnost — produkt bude pracovat samostatně bez zapojení se do žádného většího systému. Pro produkt nebudou vyžadovány žádné automatické testy. Co možná nejjednodušší rozšiřitelnost o nové vlastnosti je vítaná.
- Požadavky na implementaci — využití objektově orientovaného jazyka Java pro klientskou část.

4.2.1 Detaily případů použití

Na obrázku 4.1 je znázorněn diagram případů použití zobrazující případy použití asociované s aktéry *Uživatel*, *Uživatel přihlášený* a *Uživatel nepřihlášený*. Systém, jakožto reprezentaci produktu, nebudeme uvažovat za aktéra, ačkoli v několika případech použití figurovat bude. Pro přehlednost byly z diagramu vypuštěny některé případy použití. Například „Hodnoť bariéru“ zahrnuje nalezení bariéry, ke které má být hodnocení přiřazeno. Následuje popis případů použití se zvolenou stručnou úrovní popisu, tzn. ve většině případů popis pouze hlavních toků. Popisy podobných případů použití budou z prostorových důvodů slučovány.

Následující případy použití startuje abstraktní aktér *Uživatel*:

- Zobraz budovy, resp. bariéry — z nabízených možností vybere uživatel volbu Budovy, resp. Bariéry. Systém přichystá potřebná data a tato zobrazí na mapovém podkladu. Uživatel může budovy, resp. bariéry filtrovat dle dostupných kritérií. Po vybrání možnosti filtru dat a potvrzení volby budou data tříděna a ponechána pouze data splňující kritéria definované volbou filtru.
- Hledej cestu — z nabízených možností bude uživatelem zvolena volba Cesty. Uživatel zadá počáteční a koncový bod. Systém vhodným způsobem zobrazí nalezenou cestu mezi zadanými body. Pokud se v okolí cesty nacházejí bariéry, bude o této skutečnosti uživatel informován. Bude dostupná možnost filtru bariér jako v případě případu použití „Zobraz bariéry“.

- Zobraz informace o budově, resp. bariéře — uživatel nejprve provede kroky definující případ použití „Zobraz budovy“, resp. „Zobraz bariéry“. Na mapovém podkladu vybere vhodným způsobem budovu, resp. bariéru. Po tomto kroku bude uživateli zobrazen detail budovy, resp. bariéry obsahující detailní informace o budově, resp. bariéře, uživatelské komentáře společně s uživatelskými hodnoceními.

- Zobraz přehled — z nabízených možností bude zvolena volba Hlavní stránka. Uživatelovi bude zobrazen přehled o aktuálním dění, tj. poslední přidané komentáře a bariéry. Existuje alternativní tok, kdy samotný systém (téže aplikace) po svém startu zobrazí přehled.

- Zobraz informace o uživateli — pomocí interakce s profilovou fotografií nebo jménem uživatele systém zobrazí detail profilu sestávající ze seznamů bariér a komentářů od uživatele, jehož profil je zobrazován.

- Sdílej produkt, resp. zobraz informace o produktu — uživatel zvolí volbu Sdílet, resp. O aplikaci. Systém nabídne možnost sdílet produkt prostřednictvím, k tomuto účelu dostupné, aplikaci instalované na zařízení, resp. zobrazí textovou informaci o produktu společně s podmínkami použití.

Následující případy použití startuje aktér *Uživatel nepřihlášený*:

- Přihlas se, Registruj se — pro případ použití „Přihlas se“ systém po jeho startu nabídne možnost zadání přihlašovacích údajů a tlačítko pro potvrzení. Dále je nabídnuta možnost registrace, při které uživatel zadá potřebné údaje a tyto potvrdí. V obou případech systém oznámí výsledek prováděné akce, její úspěch či neúspěch. V případě úspěchu bude startován alternativní tok případu použití „Zobraz přehled“.

- Hodnoť produkt, Napiš zpětnou vaznu – uživatel zvolí volbu Hodnotit, resp. Zpětná vazba. Systém zobrazí uživateli dialog s tlačítkem, po jehož stlačení dochází k přesměrování na stránku produktu dostupné v distribuční službě Google Play. Pro případ použití „Napiš zpětnou vazbu“ bude zobrazeno pole pro textový vstup a tlačítko pro potvrzení.

Následující případy použití startuje aktér *Uživatel přihlášený*:

- Změň profilový obrázek — uživatel zvolí volbu „Nastavení“. Systém zobrazí aktuální profilový obrázek, pokud tento existuje (v minulosti byl zadán). Uživatel pomocí externí aplikace fotoaparát vytvoří fotografii. Po potvrzení bude fotografie přiřazena k profilu aktuálního uživatele. Systém vhodně zareaguje při nedostupnosti externí aplikace.

- Odhlaš se — uživatel zvolí volbu „Odhlásit“. Systém provede odhlášení a přesměruje aktéra na obrazovku pro nepřihlášeného uživatele.

- Vytvoř bariéru — případ použití na startu předpokládá před startem provedení případu použití „Zobraz bariéry“. Aktér vybere volbu Přidat novou bariéru. Systém zobrazí obrazovku s možností voleb vázající se k bariéře, a to zejména její umístění, datum platnosti, překonatelnost a volbu podkategorie, do které bariéra spadá. Alternativně bude umožněno vytvoření fotografie bariéry. Aktér potvrdí vytvoření bariéry, která bude systémem vytvořena.

(a) Uživatelské menu (b) Filtr budov

Obrázek 4.2: Grafické návrhy aplikace – část 1.

- Uprav bariéru — případ použití na startu předpokládá před startem provedení případu použití „Zobraz informace o bariéře“ a zároveň aktér musí být zadavatelem bariéry. Uživatel vybere možnost Upravit. Systém zobrazí totožnou obrazovku jako pro vytvoření bariéry v případu použití „Vytvoř bariéru“. Aktér provede změny, potvrdí je volbou „Upravit bariéru“ a systém provedené změny aplikuje.
- Hodnoť bariéru — případ použití na startu předpokládá provedení případu použití „Zobraz informace o bariéře“. Uživatel zvolí volbu Vytvořit vlastní hodnocení. Systém zobrazí obrazovku s možností hodnotit kladně, resp. záporně. Uživatel vybere jednu z nabízených možností. Systém volbu akceptuje a uloží nebo aktualizuje záznam o hodnocení.
- Vytvoř komentář budovy, resp. bariéry — případ použití předpokládá provedení případu použití „Zobraz informace o budově“, resp. „Zobraz informace o bariéře“. Uživatel zvolí volbu Přidat nový komentář. Systém zobrazí obrazovku s možností zadání textového komentáře. Po potvrzení provede systém uložení nebo aktualizaci záznamu.

4.3 Návrh uživatelského rozhraní

Podkapitola 4.1 se zaměřila na neformální specifikaci projektu, která byla následně transformována na diagram případů použití (obrázek 4.1). Veškeré případy byly stručně popsány v podkapitole 4.2.1. Byli nalezeni celkově tři aktéři, abstraktní aktér *Uživatel* a konkrétní *Uživatel přihlášený* společně s *Uživatel nepřihlášený*. Druhému jmenovanému, pokud nyní nebudeme uvažovat zděděné, náleží pouze dva případy použití, a to „Registruj se“ a „Přihlaš se“. Tyto by měli být viditelné co možná nejdříve po startu aplikace. Zároveň, s uvažováním dědění, musí být aktéru umožněno přeskočení registrace či přihlášení, díky čemuž získává

možnost provádění poměrně velkého počtu zděděných případů použití od abstraktního aktéra *Uživatel*.

Z uvedeného lze vyvodit nutnost existence globálního přístupového bodu pro účely navigace, ovšem s částečně odlišným obsahem pro přihlášeného a nepřihlášeného uživatele. Výsledkem v minulosti specifikovaného projektu má být mobilní aplikace pro platformu Android, tudíž vhodným řešením s přihlédnutím na poměrně velké množství případů použití se jeví využití menu dobře známé z většiny aplikací pro cílovou platformu. Konkrétně *Navigation Drawer*, které po většinu času práce s aplikací zůstává zasunuto (skryto) a tím pádem šetří prostor pro zobrazení požadovaného obsahu. Vysunutí se provede patřičným gestem nebo interakcí s ikonou umístěnou uvnitř prvku zvaném *Toolbar*[1]. Tímto rozumíme horní lištu zobrazenou ve všech obrazovkách navrhované aplikace.[2]

Návrh rozložení a vzhledu uživatelského menu je patrný z obrázku 4.2a. Horní sekce (hlavička menu) obsahuje profilovou fotografii doplněnou o uživatelské jméno a email aktuálně přihlášeného uživatele aplikace. Pod hlavičkou se nachází dostupné volby navigace, které budou později v textu rozebrány. Pokud aktuální uživatel nebude přihlášen, bude hlavička menu obsahovat možnost pro přesměrování na obrazovku přihlášení. Navigační volby budou patřičně omezeny, a to dle diagramu případu použití znázorňující obrázek 4.1. Uživatelské menu by mělo být přístupné vždy, vyjma obrazovek pro přihlášení a registraci uživatele, ve kterých dostupnost nedává smysl. Vzhled vysouvacího menu první verze aplikace, která obsahovala minimum požadované funkcionality a neuvažovala přihlášené či nepřihlášeného uživatele, je pro ilustraci patrný v příloze na obrázku A.1c.

Z popisu detailů případů použití v podkapitole 4.2.1 může být vyvozena existence několika případů použití, které pro své úspěšné provedení vyžadují jen malé množství vstupních informací poskytovaných uživatelem. Příkladem může být případ použití „Hodnoť bariéru“, ve kterém uživatel pouze zvolí jednu z nabízených možností, a to hodnocení kladné nebo záporné. Je tedy zbytečné zobrazovat možnost volby ze dvou možností tzv. „přes celou obrazovku“. Mnohem výhodnější bude využití dialogu, který se zobrazí přes obsah aktuální obrazovky, ze které bude tento vyvolán patřičnou akcí (např. stlačení tlačítka). Výsledný uživatelský prožitek tak bude jistě příjemnější. Mírně odlišnou situaci znázorňuje grafický návrh na obrázku 4.2b ukazující možný vzhled dialogu pro definici filtru budov použitých v případu použití „Zobraz budovy“. Zde uživatel volí kategorie, do nichž náleží objekty následně zobrazené na mapovém podkladu. Je využito klasických vstupních prvků, jako *CheckBox* a *RadioButton*. Pro tento případ použití můžeme konstatovat využití obrazovky formou dialogu jako hraniční mez.

Výsledkem provedení případu použití „Zobraz informace o budově“ má být zobrazení obrazovky detailu budovy. Její grafický návrh je viditelný na obrázku 4.3a. V horní části obrazovky budou zobrazeny obrázky vázající se k budově, pokud tyto budou zatím blíže nespecifikovaným způsobem dostupné. Následuje volba zobrazených informací formou vhodného přepínače, a to mezi základními, rozšířenými a uživatelskými. Na zmíněném obrázku je patrný obsah uživatelských informací, kterými ve skutečnosti jsou uživatelské komentáře vázající se k dané budově s možností přidání vlastního. Samotné komentáře by měly být pro lepší přehlednost vhodně ohraničeny.

Případ použití „Zobraz informace o bariéře“ končí zobrazením detailu bariéry (obrázek 4.3b), jehož základ uživatelského rozhraní bude podobný jako v případě detailu budovy.

[1] *Toolbar* je novější generace dobře známého prvku *ActionBar* poskytující větší možnost kontroly chovaní a celkově větší flexibilitu.

[2] Existuje nemalé množství aplikací, které *Toolbar* v určitých chvílích skrývají, například při posunu v delších seznamech prvků přesahující aktuální výšku obrazovky. Tato skutečnost však nebude brána v potaz.

(a) Detail budovy (b) Detail bariéry

Obrázek 4.3: Grafické návrhy aplikace – část 2.

Ve vrchní části obrazovky bude zobrazena mapa s označením polohy bariéry. Následuje možnost volby, pomocí přepínače mající stejný vzhled, mezi zobrazením informací o bariéře nebo výpisem uživatelských komentářů. V prvním případě, pokud je aktuální uživatel zadavatelem, bude zobrazeno akční tlačítko pro možnost úpravy bariéry. Následuje výpis jednotlivých informací, přičemž nadpis každé informace by měl být vhodně zvýrazněn, např. formou odsazení nebo tučného písma.

4.3.1 Shrnutí

Byly vytvořeny a popsány čtyři návrhy obrazovek (možno označit anglickým výrazem *wireframes*) uživatelského rozhraní výsledné aplikace (obrázky 4.2 a 4.3) zachycující možnosti navigace v aplikaci, vzhled dialogových oken a v neposlední řadě předpokládané rozmístění prvků obrazovek detailů budovy a bariéry. Z množství nalezených případů použití znázorněných diagramem na obrázku 4.1 lze vyvodit budoucí velké množství obrazovek výsledné aplikace, pro které je zbytečné vytvářet jednotlivé grafické návrhy. Jejich základem ve většině případů budou popsané elementy uživatelského rozhraní a budou stavěny na podobném principu pro zachování jednotného vzhledu uživatelského rozhraní.

Kapitola 5

Implementace serverové části

V kapitole 4 byly nalezeny a analyzovány požadavky kladené na výsledný produkt, kterým má být mobilní aplikace určená pro platformu Android. Z požadavků je zřejmá nutnost perzistentního uchování aplikačních dat. Tato navíc musí být přístupná globálně pro všechny uživatele aplikace. Zároveň je požadována patřičná aktuálnost dat. Uvedené znační nutnost existence globálního přístupového bodu uchovávajícího a spravujícího data. Bude se tedy jednat o klient-server architekturu, kdy veškerá aplikační data podléhající nutnosti synchronizace budou uložena na právě jednom místě (vzdálený server) a spotřebitelé dat při jejich potřebě budou kontaktovat globální přístupový bod, který požadavku vyhoví, nebo jej zamítne např. z autentizačních důvodů.

Serverová část bude rozdělena na dvě části. První bude poskytovat informace o vybraných budovách města Brna. Druhá část bude mít na starost uchování zbývajících aplikačních dat, např. informace o uživatelích, bariérách a komentářích.

5.1 Statická data

Jedním z důvodů rozdělení serverové logiky na dvě části je oddělení obsluhy dvojího druhu dat. Prvním jsou data potenciálně stálá, z krátkodobého hlediska neměnná. Taktéž se jedná o data, jejichž autorem je třetí strana (Magistrát města Brna). Druhá část dat je pravý opak, data z krátkodobého i dlouhodobého hlediska dynamická.

Aktuální podkapitola se zabývá uchování dat prvního typu, kterým jsou atributy definující vybrané budovy města Brna. Pro jejich získání lze využít podpůrné služby popsané v podkapitole 3.1.2. První verze aplikace tyto skutečně využívala, nicméně během vývoje nebyly služby vždy přístupné z důvodů plánovaných odstávek, nebo v horším případě neočekávaných výpadků. Nejjednodušší řešení spočívá v uchování textových kopií odpovědí, které vrací zmíněné služby, jejichž obsahem je vždy objekt ve formátu JSON. Struktura objektu byla detailněji popsána ve zmíněné podkapitole 3.1.2, přičemž jiné formáty odpovědi nebyly uvažovány.

Obsluhu požadavků pro získávání atributů popisující vybrané budovy města Brna má přidělenou jednoduchý skript implementovaný pomocí skriptovacího jazyka PHP. Požadavek je odeslán prostřednictvím HTTP protokolu metodou GET. Formát URL adresy je následující: `{api_url}/{script}?file={source}`, kde jednotlivé bude nahrazeno následovně:

- `{api_url}` — URL adresa serveru,
- `{script}` — název souboru PHP skriptu, vždy jím bude *getjson.php*,

- `{source}` — název požadovaného souboru, jehož obsah má být navrácen. Název je vždy formátu `{id}_{layer}`, kde `{id}`, resp. `{layer}` bude nahrazeno dvojciferným číselným identifikátorem, resp. upraveným názvem vrstvy – nahrazení velkých písmen za odpovídající malá, případná čárka s následující mezerou bude nahrazena podtržítkem. Názvy vrstev společně s identifikátory lze nalézt v tabulce 3.1.

Například požadavek na získání všech budov vrstvy Kultura s identifikátorem „1“ bude tvaru `http://dp.kopla.cz/getjson.php?file=01_kultura`.

5.1.1 Princip činnosti skriptu

Po příjmu požadavku na navrácení atributů budov náležící určité vrstvy je nejprve zkontrolováno, zda-li URL splňuje předepsaný formát. Především musí obsahovat název souboru, jehož obsah je požadován. V negativním případě bude zaslána prázdná odpověď s HTTP statusem 400 *Bad Request*. Jinak následuje kontrola existence textového souboru uloženého na disku. Jestliže žádný takový neexistuje, bude navrácena prázdná odpověď, nyní s nastaveným HTTP statusem 404 *Not Found*.

Pokud výše uvedené podmínky budou splněny, skript nejprve sestaví HTTP hlavičky. Jedná se například o velikost a typ přenášeného obsahu s doplněním informace o použitém kódování, kterým je vždy UTF-8. Poté přečte obsah souboru, vrátí jej klientovi a ukončí svou činnost.

Nutno podotknout absenci kontroly přístupových práv, která je v tomto případě nadbytečná. Data vracející skript reprezentují informace o budovách města Brna, která jsou jinak volně dostupná pomocí služeb popsaných v kapitole 3.1.2.

5.2 Dynamická data

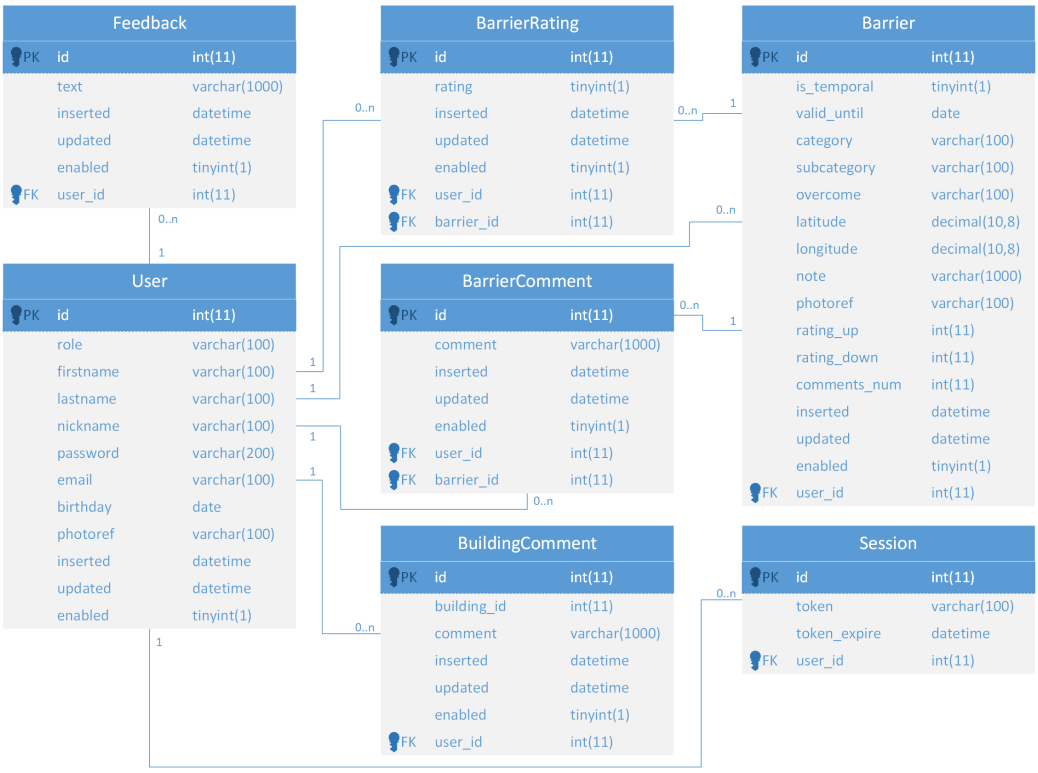
Z analýzy požadavků plyne nutnost uchovat kromě atributů budov města Brna popsaných v předcházející podkapitole i informace např. o uživatelích, bariérách a komentářích, které mají být dostupné všem uživatelům aplikace. Pro jejich uchovávání byla zvolena relační databáze, která svými možnostmi plně vyhovuje charakteru dat. Na obrázku 5.1 je znázorněno schéma výsledné databáze. Stručný popis tabulek následuje:

- *User* — uchovává veškeré informace o registrovaném uživateli aplikace – jméno, příjmení, zvolenou přezdívku, emailovou adresu, datum narození. Dále také heslo, které je pro zajištění alespoň nízké úrovně zabezpečení uloženo jako otisk hašovací funkce SHA-512.

- *Feedback* — slouží pro uchování textové zpětné vazby vázající se na konkrétního uživatele.

- *Barrier* — reprezentace bariéry (téže překážky). Sloupce tabulky jsou kategorie a podkategorie, do které bariéra spadá, dále překonatelnost překážky (překonatelná, nepřekonatelná, překonatelná s asistencí), souřadnice zeměpisné šířky a délky (umístění překážky uvažujeme pouze bodově), textová poznámka.

- *BarrierRating* — vazební tabulka mezi *Barrier* a *Rating* sloužící k uchování informací o uživatelském hodnocení bariéry, které může být kladné (hodnota 1) nebo záporné (hodnota 0).



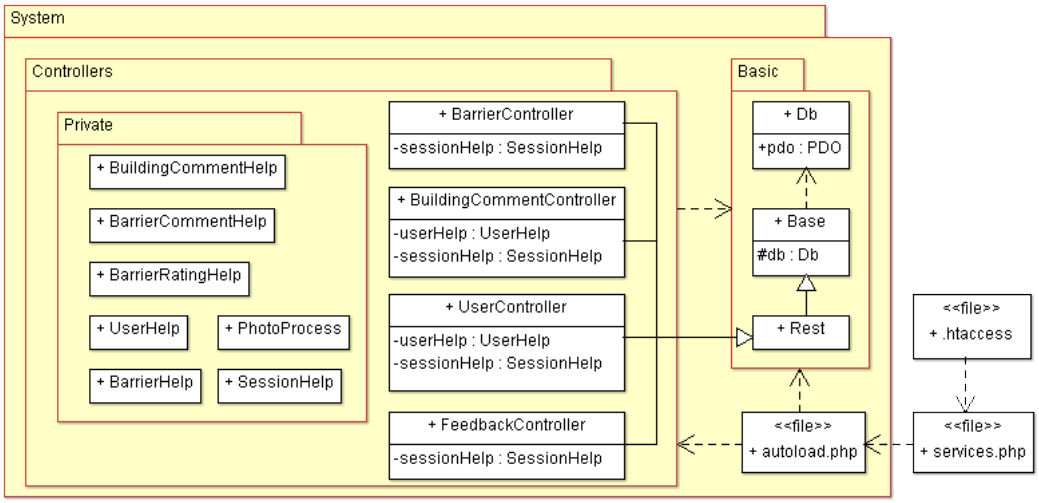
Obrázek 5.1: Návrh schématu databáze

- *BarrierComment* — druhá a zároveň poslední vazební tabulka, tentokrát mezi *Barrier* a *Comment*. Uchovává textový komentář vztažený k bariéře.
- *Session* — uchovává uživatelské *tokeny* společně s jejich datem expirace. Výsledkem akce přihlášení do systému je přidělení právě jednoho tokenu, který bude následně vyžadován pro operace editace či přidaní dat.
- *BuildingComment* — slouží pro uchování uživatelských komentářů k vybraným budovám města Brna. Každá budova je identifikována svým jednoznačným ID, kterému odpovídá sloupec *building_id*. Pro úplnost, nejedná se tedy o cizí klíč odkazující na sloupec žádné tabulky z aktuální databáze, nýbrž odkazuje na budovu uloženou „statickým způsobem“ (viz podkapitola 5.1).

Veškeré tabulky, vyjma *Session*, obsahují sloupce *inserted*, resp. *updated* uchovávající datum a čas vložení, resp. úpravy záznamu. Hodnoty sloupců jsou automaticky nastavovány během operace přidání, resp. úpravy. Aby byla zachována trvalost záznamů, operace mazání (SQL příkaz *DELETE*) nebude nikdy vykonán. Náhradou je sloupec *enabled*, který obsahuje každá tabulka (vyjma *Session*). Řádek libovolné tabulky s hodnotou „0“ sloupce *enabled* pak označuje virtuálně odstraněný záznam.

Pro možnost přiřazení fotografie záznamu v tabulce *User* nebo *Barrier* slouží sloupec *photoref*, jehož hodnota bude *NULL*, pokud záznamu není přiřazena fotografie. V opačném případě bude hodnotou část názvu souboru s fotografií. Princip bude podrobněji vysvětlen v podkapitole 5.3.2.

Tabulka *Barrier* obsahuje sloupce *rating_up*, resp. *rating_down*. Jedná se o agregované hodnoty počtu kladných, resp. záporných hodnocení konkrétní bariéry. Při každém



Obrázek 5.2: Zjednodušená struktura API obsluhující dynamická data

vložení či úpravě záznamu tabulky *BarrierRating* budou přepočítány hodnoty uvedených sloupců v tabulce *Barrier*. Jedná se o optimalizaci, díky které nemusí být během dotazu na záznamy z tabulky *Barrier* tato slučována s tabulkou *BarrierRating* za předpokladu výsledku obsahujícího počet kladných a záporných hodnocení bariéry. Stejný mechanismus platí pro sloupec *comments_num* tabulky *BarrierComments* uchovávající počet uživatelských komentářů bariéry.

5.3 API pro obsluhu dynamických dat

Pro manipulaci s relační databází, jejíž schéma bylo popsáno v předchozí podkapitole, bylo vytvořeno API implementované pomocí skriptovacího jazyka PHP.

Povolený formát URL adresy zasílaných požadavků na služby API je následující: `{api_url}/{controller}/{action}/{value}`, kde jednotlivé bude nahrazeno následovně:

- `{api_url}` — URL adresa serveru,
- `{controller}` — část názvu třídy plnící funkci kontroléru, tj. třída koordinující modifikaci obsahu databázových tabulek,
- `{action}` — akce prováděná kontrolérem,
- `{value}` — doplňující hodnota, která je potřebná pro vykonání specifikované akce (ne vždy je tato požadována, vždy záleží na charakteru akce).

Například požadavek na vrácení informací o uživateli s identifikátorem „1“ bude tvaru `http://dpapi.kopla.cz/User/GetById/1`.

5.3.1 Struktura implementace

Zjednodušená struktura tříd, artefaktů a balíčků použitých pro implementaci API je znázorněna na obrázku 5.2. Pro přehlednost nejsou v diagramu vyznačeny veškeré závislosti mezi třídami a balíčky[1]. Dále byly vynechány předpisy funkcí a většina atributů.

[1]Nejedná se o balíčky v pravém slova smyslu, ale spíše o znázornění adresářové struktury. Dále bude používán pojem balíček.

Controllers

Funkci kontrolérů zastávají třídy z balíčku *Controllers*, jejichž popis následuje:

- *FeedbackController* — jedná se o nejjednodušší kontrolér z celé implementace API. Manipuluje s tabulkou *Feedback*, do které je umožněno pouze vkládat nové záznamy, nikoli upravovat nebo mazat. Třída vlastní jedinou funkci, a tou je *Create* vkládající řádek do zmíněné tabulky reprezentující textovou zpětnou vazbu od uživatele.

- *UserController* — poskytuje funkce pro manipulaci s databázovou tabulkou *User*. Např. funkce *GetById* vyhledá uživatele v databázi pomocí číselného identifikátoru, který přijímá jako parametr. Tato je tedy využita pro zpracování požadavku z příkladu uvedeného na začátku této podkapitoly. Rovněž poskytuje funkci *Login*, která se pokusí podle vstupního emailu a hesla vyhledat záznam v tabulce. Pokud je takový nalezen, bude sestaven objekt reprezentující uživatele, který bude navrácen jako výsledek.

- *BuildingCommentController* — manipuluje převážně s tabulkou *BuildingComment*. Slouží pro vytváření a úpravu komentářů k vybraným budovám města Brna. Poskytuje např. metodu *GetLatestByUser*, která vyhledá a navrátí specifikovaný počet nejnovějších komentářů, jejímiž autorem je uživatel aplikace identifikující se pomocí *tokenu* zasílaného v hlavičce dotazu a v minulosti získaného pomocí služby *Login*, která je spravována již zmíněnou stejnojmennou funkcí *Login* třídy *UserController*.

- *BarrierController* — z pohledu počtu funkcí (téže služeb) poskytovaných touto třídou se jedná o nejsložitější kontrolér. Manipuluje s tabulkou *Barrier* reprezentující bariéru. Současně však také s vazebními tabulkami *BarrierRating* sloužící pro uchování uživatelských hodnocení bariéry a *BarrierComment* reprezentující komentáře k bariérám. Např. pro vytvoření, resp. úpravu uživatelského hodnocení bariéry poskytuje funkce *RatingCreate*, resp. *RatingUpdate*.

Kontroléry pro svou činnost využívají funkcí umístěných v pomocných třídách obsažených v balíčku *Private*. Diagram na obrázku 5.2 neobsahuje zakreslené závislosti mezi těmito třídami, jelikož jejich uvedení by vedlo ke značnému snížení čitelnosti samotného diagramu vzhledem k přinesenému užitku. Všechny dříve uvedené kontroléry vlastní instanci třídy *SessionHelp*, jenž je dostupná po celou dobu existence instance kontroléru (inicializace objektu je prováděna v konstruktoru třídy).

Třída *SessionHelp* poskytuje funkce pro manipulaci s pomyslným sezením. Provádí tedy operace nad tabulkou *Session*. Zejména poskytuje funkci *CreateToken*, která pro daného uživatele vytvoří přístupový *token*. Klientská aplikace si tento uchovává po celou dobu sezení, které začíná přihlášením pomocí služby *Login* a končí odhlášením, nebo po vypršení časové platnosti *tokenu*, který má při vytváření nastavenu platnost 48 hodin. *SessionHelp* dále poskytuje veřejné funkce *DeleteToken*, *FindToken*, *CheckTokenValidity*, *RefreshToken* sloužící postupně pro ukončení sezení, nalezení uživatele dle tokenu, ověření a obnovu platnosti tokenu.

Další pomocnou třídou je *BarrierHelp*. Za zmínku stojí funkce *recalcRatingUpDown* upravující hodnoty sloupců tabulky *Barrier* agregující počet kladných a záporných hodnocení překážky, a dále *recalcCommentsNum* přepočítávající počet uživatelských komentářů vázající se k dané překážce. Existence a účel sloupců byla diskutována dříve.

Třídy balíčku *Private* dále obsahují statické funkce plnící funkcionalitu často se vyskytujících operací nad tabulkami databáze. Důvodů, proč tyto nejsou přímo ve třídách kontrolérů můžeme nalézt několik:

1. Třídy kontrolérů by měly pouze koordinovat postupné provedení několika operací tvořící v konečném součtu poskytovanou službu implementovaného API. Např. úprava komentáře zadaného uživatelem sestává z načtení upravovaného, ověření oprávnění, samotní úpravy a promítnutí změn do databáze.

2. Redukce opakujícího se kódu, s tím souvisí jeho sdílení. Např. v minulosti vysvětlený princip třídy *SessionHelp*, jenž je využívána všemi kontroléry.

3. Třídy kontrolérů by měly obsahovat pouze veřejné metody. Tyto budou později reprezentovat výsledné služby (princip bude vysvětlen v podkapitole 5.3.3). Zmíněné je důvodem, proč *SessionHelp* není implementována jako kontrolér, ale pouze jako pomocná třída.

Basic

Pro účely oddělení základních tříd od zbytku implementace aplikace byl vytvořen balíček *Basic* obsahující následující třídy:

- *Db* — inicializuje spojení s relační databází za pomoci třídy PDO[2].

- *Base* — bázová třída, od které dědí vlastnosti specifičtější třídy vyžadující práci s databází. Jako atribut uchovává objekt třídy PDO získaný prostřednictvím výše uvedené třídy *Db*.

- *Rest* — v současné době dědí od *Base* pouze tato třída. Pro manipulaci s databází má tedy k dispozici patřičný objekt.

Třídy kontrolérů z balíčku *Controllers* sdílí základní, avšak o to důležitější vlastnosti. Jedná se zejména o funkce získávající parametry zasílané prostřednictvím HTTP metod GET a POST, zpracování hlaviček požadavku a příprava společně s odesláním samotné odpovědi na požadavek. Jejich výčet následuje:

- *get_request_data* — získá a uchová parametry z URL dotazu. Taktéž zpracuje tělo dotazu zasílané jako *form-data*, jestliže aktuální požadavek je poslán prostřednictvím HTTP metody POST.

- *get_headers_data* — zkontroluje existenci hlaviček zasílaných v dotazu. Pokud existují, uchová je pro budoucí použití. Veškeré služby poskytované implementovaným API požadují v hlavičce dotazu maximálně jeden klíč *User-Token* s hodnotou uživatelského tokenu. Při nutnosti více párů klíč-hodnota obsažených v hlavičce je implementace API již nachystaná.

- *clear* — odstraňuje HTML a PHP tagy ze vstupního textového řetězce, který pak může být použit v SQL příkazech manipulující s databází. Jedná se o zamezení útoku SQL injection. Tato funkce je využívána při provádění dvou výše zmíněných funkcí.

[2]Dokumentace třídy *PDO* – http://php.net/manual/en/class.pdo.php [Online; navštíveno 09.05.2017]

- *response* — přichystá odpověď na zpracovaný dotaz. Nastavuje hlavičky, vytváří výsledný objekt, jenž bude před koncem provádění funkce zaslán klientovi ve formě textového řetězce obsahující serializovaný objekt ve formátu JSON.

- *not_acceptable* — zasílá klientovi informaci o nevalidním požadavku, který značí HTTP status 405. Nevalidním požadavkem rozumíme např. chybějící parametr URL dotazu, který je vyžadován pro úspěšné provedení určité služby (např. pro získání detailu uživatele je vyžadováno jeho ID).

Výše zmíněné funkce potřebují pro svou činnost všechny kontroléry. Proto byly umístěny do třídy *Rest*, od které dědí všechny třídy kontrolérů z balíčku *Controllers*.

5.3.2 Uchování fotografií

Tabulka *User* obsahuje sloupec *photoref*, totožné platí pro tabulku *Barrier*. Jinými slovy, uživatel bude mít možnost nastavení profilové fotografie a dále během vytváření či editace bariéry tuto bude moci vyfotografovat (viz podkapitola 4.2).

Z uvedeného plynou požadavky na zpracování a uchování fotografií. Pro tento účel slouží pomocná třída *PhotoProcess* umístěná v balíčku *Private*. Obsahuje jedinou statickou funkci, a tou je *resize* přijímající cestu k umístění zdrojového a cílového souboru, maximální výšku a šířku výsledné fotografie a číselnou hodnotu udávající kvalitu převodu. Funkce dle původní a požadované velikosti tuto zmenší tak, aby zůstalo zachování poměru stran a výsledná fotografie nebyla zdeformována. Popsaným způsobem je zaručena úspora diskového prostoru, ale především při požadavku na fotografii nebude nutné přenášet neúměrně velké množství dat, čímž se zkrátí doba potřebná k přenosu fotografie.

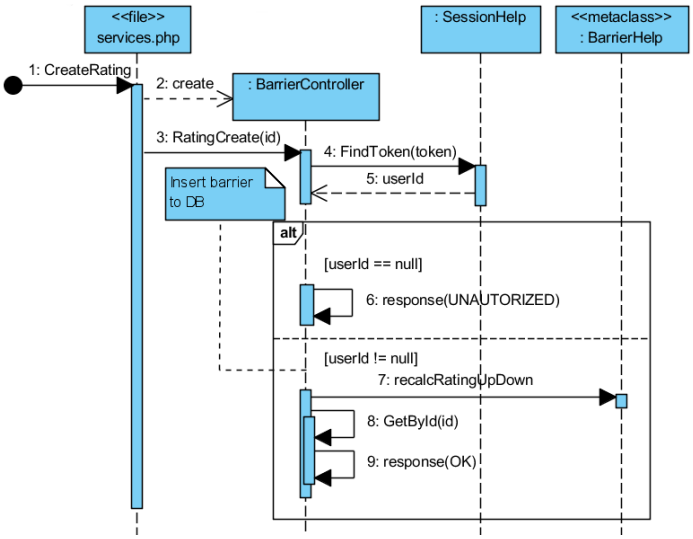
Funkce *resize* fotografii zmenšuje a zároveň výsledek ukládá ve formátu JPEG na disk, tj. nikoli do databáze. Jedná se o vhodnější způsob, jelikož aktuální webhosting má omezenou kapacitu databáze, která by s postupem času nemusela stačit.

Název uložené fotografie sestává z textové hodnoty identifikátoru záznamu v tabulce a přípony. Například uživatel z tabulky *User* reprezentovaný číselným identifikátorem 11 bude mít profilovou fotografii uchovanou na disku s názvem *11.jpg*. Popisované analogicky platí i pro záznamy z tabulky *Barrier*. Uvedené naznačuje omezení, které však pro účely vyvíjené aplikace není třeba uvažovat. Uživatel a bariéra mohou mít přiřazenu maximálně jednu fotografii. V případě uživatele situace nevadí, pokud by však bylo požadavkem přiřazovat bariéře libovolné množství fotografií, musela by se zavést pomocná tabulka uchovávající cesty k fotografiím a cizím klíčem odkazující na řádek tabulky *Barrier*.

5.3.3 Princip funkce API

Implementované API poskytuje celkově 26 služeb pro získávání a manipulaci s daty uloženými v relační databázi. Jejich výčet je uveden v příloze B prostřednictvím sedmi tabulek. Každý kontrolér poskytuje služby zpracovávající HTTP požadavky GET a POST. Výjimkou je kontrolér *FeedbackController*, který poskytuje jedinou službu *Create* prostřednictvím požadavku POST (viz tabulka B.7). Například výčet služeb kontroléru *UserController* je pro HTTP metodu GET, resp. POST je uveden v tabulce B.1, resp. B.2.

Popis každé služby a její implementace není z prostorových důvodů možný a nedává ani smysl. Služby fungují a jsou implementovány na podobném principu. Proto byl vybrán jeden reprezentant, jímž je služba pro vytvoření hodnocení existující bariéry *RatingCreate*



Obrázek 5.3: Princip provedení služby *CreateRating*

kontroléru *BarrierController*, uvedená v tabulce B.6. Požadavek zaslaný klientem na vytvoření hodnocení bariéry s identifikátorem „3“ bude proveden pomocí následující URL: `http://dpapi.kopla.cz/Barrier/RatingCreate/3`.

V následujícím popisu se budou vyskytovat názvy skriptů a souboru, které doposud nebyly zmíněny. Jsou však znázorněny na obrázku 5.2 ukazující zjednodušenou strukturu implementace API.

Požadavek přicházející na subdoménu *dpapi* je souborem *.htaccess* zachycen a přesměrován na zpracování prostřednictvím skriptu *services.php*, kterému jsou zaslány GET parametry určující část názvu kontroléru (Barrier) a akce (RatingCreate). V popisovaném případě je zaslána i doplňující hodnota (3). Skript prostřednictvím *autoload.php* načte veškeré soubory s příponou *.php* z balíčků *Basic* a *Controllers* (vyjma balíčku *Private*). Dále *services.php* sestaví název třídy kontroléru (*BarrierController*) a zkontroluje její dostupnost. V kladném případě vytvoří instanci zmíněného kontroléru. Následuje kontrola existence funkce reprezentující akci (RatingCreate). Jestliže tato existuje, pokračuje zavoláním metody s parametrem (v našem případě je parametr roven hodnotě „3“). Pokud neexistuje třída kontroléru, resp. funkce, je vrácena textová hodnota „Třída neexistuje“, resp. „Metoda neexistuje“ a skript následně ukončí činnost.

Hlavní kroky úspěšného provedení služby *CreateRating* ukazuje diagram sekvence na obrázku 5.3. Skript *services.php*, jak bylo zmíněno výše, přijímá přesměrovaný požadavek. V diagramu je tato skutečnost znázorněna pomocí nalezené zprávy 1 (dále značeno bez uvedení v závorce). Po popsaných kontrolách vytváří instanci třídy *BarrierController* (2) a volá patřičnou akci (3) s číselným identifikátorem bariéry, ke které je vytvářeno hodnocení. Požadavek v hlavičce obsahuje uživatelský token. Tento je zaslán objektu třídy *SessionHelp* (4), která vrací identifikátor uživatele, kterému token patří (5). Pokud nebyl identifikátor nalezen, bude klientovi zaslána odpověď *UNAUTORIZED* značící neautorizovaný přístup, který je způsoben vypršením platnosti tokenu nebo jeho neexistencí (6). V opačném případě pokračuje zpracování vložením záznamu o hodnocení bariéry do tabulky *BarrierRating*. Následně jsou pomocí statické metody třídy *BarrierHelp* přepočítány hodnoty sloupců *rating_up* a *rating_down* agregující počet kladných a záporných hodnocení bariéry, ke které bylo právě přiděleno nové hodnocení (7). Jako výsledek akce *CreateRating*

má být navrácen objekt ve formátu JSON reprezentující bariéru s aktualizovaným počtem kladných a záporných hodnocení. Zmíněné je provedeno pomocí služby *GetById* (8), která zasílá klientovi požadovaný výsledek se statusem *OK* značící úspěšné provedení služby (9).

5.4 Shrnutí kapitoly

Kapitola pojednávala o existenci dvou typů dat, které je třeba v kontextu vyvíjené mobilní aplikace a serverové části uvažovat. Prvním typem jsou statická data uchovávající informace o vybraných budovách města Brna. Tato jsou uložena přímo v textových souborech, které obsahují textové odpovědi z REST API Magistrátu města Brna popsané v podkapitole 3.1.2. Druhým typem jsou data často se měnící, dynamická, jako např. záznamy o uživatelích, bariérách a komentářů. Pro jejich uchování byla zvolena relační databáze, jelikož plně vyhovuje charakteru dat. Pro účely manipulace s databází bylo implementováno REST API, jehož struktura byla popsána v podkapitole 5.3. Toto API poskytuje celkem 26 služeb pro účely získání a manipulaci s daty uloženými v relační databázi. Z množiny služeb byl vybrán jeden zástupce, jehož použití společně s principem implementace bylo popsáno v podkapitole 5.3.3. Tento popis nezahrnuje vysvětlení všech částí implementace celého REST API, účelem bylo nastínění principů, na kterých implementace staví. Pro další informace a detaily jsou k dispozici zdrojové kódy na přiloženém CD (kapitola F) obsahující snad dostatečné množství komentářů pro případné pochopení zbylé logiky.

Kapitola 6

Implementace klientské části

Existuje více přístupů k vývoji aplikací pro platformu Android z hlediska výběru frameworku, na kterém bude aplikace postavena. Například *Phonegrap*[1] je vhodným výběrem pro vývojáře se znalostí HTML, CSS a JavaScript. Výhodou je pouze jedna sada zdrojových kódů, která je s využitím frameworku použitelná na více platformách, a to iOS, Android a Windows. V případě platformy Android bude, zjednodušeně řečeno, vytvořeno *WebView*, které bude manipulováno pomocí skriptovacího jazyka JavaScript.

Teoretická část diplomové práce popisuje platformu Android, které se zabývala kapitola 2. S přihlédnutím na její jednotlivé části lze vyvodit využití prostředků pro vývoj (podkapitola 2.3), tj. použití Android SDK (podkapitola 2.3.1), jenž svými možnostmi dalece přesahuje požadavky kladené vyvíjenou aplikací. Dále ostatní dostupné frameworky není třeba uvažovat.

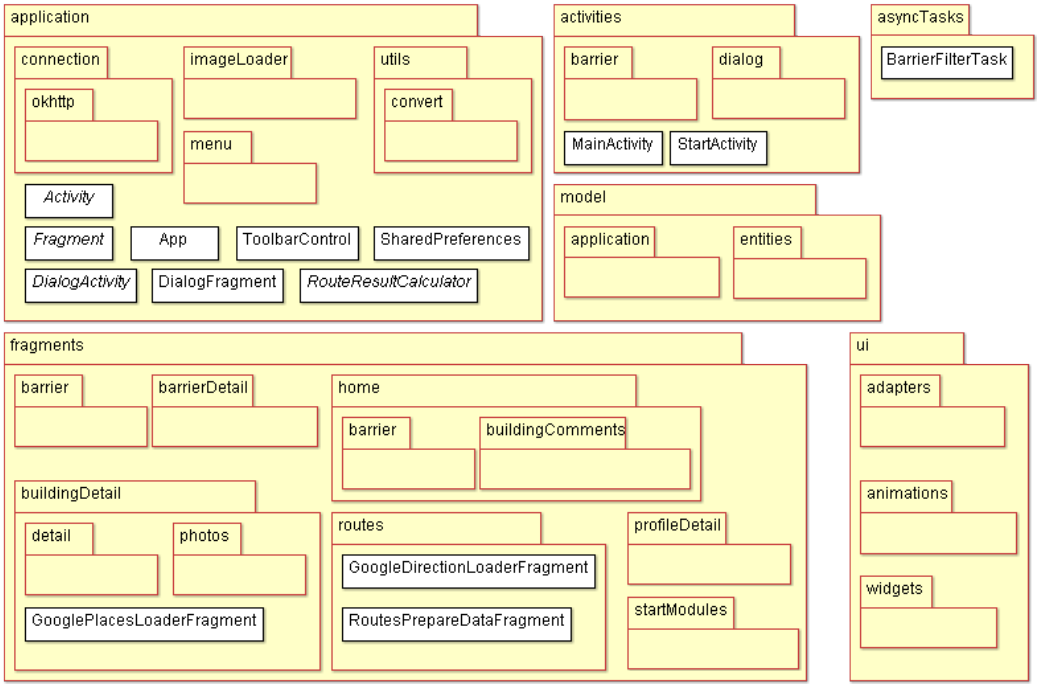
V jednotlivých částech aktuální kapitoly budou uváděny názvy knihoven či webových služeb, které byly využity pro účely implementace výsledné mobilní aplikace. Výčet těchto je uveden v příloze E, přičemž v budoucnu nebude uváděn explicitní odkaz na zmíněnou přílohu.

6.1 Architektura implementace

Programová část implementace byla kompletně napsána v objektově orientovaném jazyce Java, pokud pomineme popisy vizuálního rozložení prvků na obrazovkách (tzv. definice layoutu), které byly napsány prostřednictvím značkovacího jazyka XML, což je při použití Android SDK zcela běžnou a pohodlnou záležitostí. Druhým způsobem definice rozloží je instanciace elementů za běhu aplikace bez využití funkcionality nabízené třídou *LayoutInflater*. Tento přístup je velmi nepraktický a používá se jen ve specifických případech.

Architekturu programové části tvoří několik samostatných balíčků, které jsou znázorněny na obrázku 6.1. Pro přehlednost je, podobně jako v případě popisu struktury implementovaného REST API serverové části na obrázku 5.2, tentokrát úplně vynecháno znázornění závislostí jednotlivých balíčků. Taktéž předpisy metod a atributů tříd nejsou uvedeny. Jedná se tedy o zjednodušený diagram nejdůležitějších tříd s důrazem znázornění jejich náležitosti do programových balíčků. Jak je zvykem v případě UML diagramu, abstraktní třídy jsou od ostatních odlišeny zápisem jejich názvu kurzívou. Následuje popis jednotlivých kořenových balíčků.

[1] Domovská stránka — http://phonegap.com/



Obrázek 6.1: Zjednodušená struktura implementace klientské aplikace

6.1.1 Balíček Application

Hlavním balíčkem je bezesporu *Application* obsahující nejdůležitější třídy využité v prakticky celé implementaci aplikace. Obsahuje zejména abstraktní třídy *Activity*, resp. *Fragment*, jenž jsou základem pro většinu tříd aktivit, resp. fragmentů.

Activity

Od abstraktní třídy *Activity* dědí vlastnosti veškeré ostatní aktivity vyskytující se v kořenu balíčku *Activities* (podkapitola 6.1.3). Samotná dědí od *AppCompatActivity*. Jedná se o třídu z knihovny *Android Support v7*. Důvodem je zachování zpětné kompatibility, aby mohla být výsledná aplikace funkční na zařízeních vlastnící API 15 a vyšší (viz tabulka 2.1), což by mělo znamenat dostupnost aplikace, dle aktuální situace, pro více než 99 % uživatelů navštěvující distribuční službu *Google Play*. Ve skutečnosti s pomocí zmíněné knihovny lze cílit i na zařízení s nižší verzí API, avšak dle popsané situace je toto zbytečné a pravděpodobně v konečném součtu i nemožné, jelikož ostatní komponenty využívané aplikací (např. knihovny třetích stran) zavádí taktéž jistá omezení, díky kterým cílí ve většině případů rovněž na API verze 15 a vyšší. Minimální verze API lze nastavit např. prostřednictvím atributu *minSdkVersion* v souboru *AndroidManifest* popsaného kapitolou 2.3.4. Ve stromu dědičnosti třídy *AppCompatActivity*[2] se nachází *android.app.Activity*, která byla podrobněji popsána v kapitole 2.2.1 obsahující především její životní cyklus, který z podstaty věci zůstává zachován.

[2]Dokumentace — https://developer.android.com/reference/android/support/v7/app/AppCompatActivity.html [Online; navštíveno 14.05.2017]

Abstraktní třída *Activity* nastavuje a spravuje definici rozložení prvků na obrazovce ze souboru *activity_layout.xml* (dále bude především používán anglický výraz „layout“). Tento obsahuje následující prvky (možno nazvat elementy):

1. *Toolbar* — nahrazující a ve starších verzí platformy Android hojně používaný prvek *ActionBar*, jehož použití muselo být vždy definováno prostřednictvím stylu uvedeného u elementu specifikující aktivitu v souboru *AndroidManifest.xml*. Prvek *Toolbar* tento nedostatek odstraňuje. Může být definován až v layoutu aktivity, tak jako v našem případě. Díky tomu vyniká lepší využitelností a přizpůsobitelnosti aktuálním potřebám aktivity, respektive celé aplikace.

2. Kontejner obsahu — element *FrameLayout*, do něhož se bude umísťovat layout podřízené aktivity vzhledem k třídě *Activity*.

3. Elementy obsahující layouty pro zobrazení aplikačních chyb, jako nedostupnost internetového připojení nebo chyba provádění libovolné akce, např. navrácený chybový kód služby a z toho plynoucí nemožnost zobrazit požadovaný obsah, avšak nutnost upozornění uživatele o této situaci. V neposlední řadě element zobrazující průběh zpracování (anglicky *loading*) během dlouhotrvajících operací.

4. *DrawerLayout* — kořenový element souboru *activity_layout.xml*, v jehož těle se mohou nacházet právě dva elementy. Prvním je element obalující výše zmíněné prvky. Druhým je kontejner pro obsah vyjíždějícího menu aplikace, které bude diskutováno níže.

Aby platilo výše uvedené, musí třída *Activity* přepsat metody *setContentView* a *findViewById*, jejichž funkcionalita bude pozměněna. První metoda bude přijímat identifikátor layoutu od podřízené aktivity, ten pomocí třídy *android.view.LayoutInflater* vytvoří a umístí jej do kontejneru obsahu (výčet č. 2). Při požadavku podřízené aktivity na vyhledání elementu metodou *findViewById* bude dle identifikátoru prohledáván pouze obsah kontejnerového obsahu, nikoli celého layoutu třídy *Activity*. Popsaný mechanismus zaručuje vyhnutí se případným kolizím hodnot identifikátorů elementů uvedených v layoutech nadřazené a podřazené aktivity, jelikož unikátnost identifikátorů není frameworkem požadována.[3]

Dalším úkolem třídy *Activity* je kompletní správa uživatelského menu, které je umístěno jako druhý element prvku *DrawerLayout* (výčet. č. 4). Menu je vytvořeno dynamicky za běhu aplikace, přičemž jeho obsah je odlišný v případech přihlášeného a nepřihlášeného uživatele. Strukturu menu tvoří hlavička, kterou následují položky složené z ikony a textové hodnoty. Podkapitola 4.3 částečně rozebírala budoucí podobu aplikačního menu, jehož grafický návrh byl znázorněn na obrázku 4.2a. Finální podoba menu pro nepřihlášeného, resp. přihlášeného uživatele je viditelná na snímcích C.2a, resp. C.2b. Třída *Activity* pro vytváření obsahu menu využívá abstraktní třídy *MenuBuilder* umístěné v balíčku *application.menu*, která poskytuje veřejnou statickou metodu *buildMenu*. Ta podle aktuálního stavu aplikace rozhodne, zda je uživatel přihlášen či nikoli. Podle této informace využije seznam dostupných položek deklarovaných pomocí výčtového typu enum *MenuItems*. Každá položka pak obsahuje informaci o jejím názvu, použité ikoně, třídy aktivity spuštěnou po výběru položky společně s informací o vyprázdnění zásobníku aktivit během spouštění a konečně, zda má být položka dostupná nepřihlášenému uživateli. Pokud uživatel vybere položku nebo interaguje s hlavičkou menu, bude pomocí zpětného volání o této skutečnosti informován

[3] Unikátnost z principu ani vyžadovanou být nemůže, uvážíme-li např. situaci množiny prvků umístěných v *ListView* a spravovaných pomocí adaptéru, tudíž sdílející tentýž layout.

posluchač, v našem případě abstraktní třída *Activity*, která implementuje rozhraní *MenuBuilderCallback*, jehož instance je předávána jako jeden z parametrů metody *buildMenu*.

ToolbarControl

Předchozí popsaná abstraktní třída *Activity* má ve svém layoutu umístěn prvek *Toolbar* reprezentující vrchní navigační lištu umístěnou téměř ve všech dostupných obrazovkách. Pro manipulaci se zmíněným prvkem byla vytvořena třída *ToolbarControl*, jejíž instanci spravuje abstraktní třída *Activity* a poskytuje ji podtřídám na jejich případné vyžádání.

Třída *ToolbarControl* umožňuje provádět následující operace:

- Nastavení hlavního nadpisu okna zarovnaného na střed obrazovky. Situaci lze pozorovat např. na obrázku C.1. Standardně bývá nadpis v aplikacích pro platformu Android zarovnáván vlevo, avšak zarovnání na střed může vzhledem k ostatnímu obsahu okna vypadat lépe.
- Zobrazení ikony šipky značící možnost navigace o krok zpět (např. obrázek C.1b). V případě interakce s ikonou šipky bude tato událost oznámena případnému posluchači prostřednictvím rozhraní *ToolbarControlListener*.
- Zobrazení ikony značící dostupnost vysouvacího menu (např. snímky C.3).
- Zprůhlednění celé vrchní navigační lišty. Hlavní obsah aktivity se pak přesune blíže k prvku *Status Bar*[4]. Situaci lze pozorovat na obrázku C.4.
- Správa případného menu, obsahujícího akční tlačítka, umístěného v samotném prvku *Toolbar*, a to na jeho pravé straně (viditelné rovněž na obrázku C.4).

DialogActivity

Implementovaná aplikace nabízí uživateli několik obrazovek, jejichž obsah lze vždy zcela umístit na displej zařízení. Zobrazení aktivity formou dialogu pak může znamenat zlepšení uživatelského prožitku během práce s aplikací. Pokud obsah bude přesahovat aktuální výšku zobrazení, bude použit prvek *ScrollView*, který umožní posuv přesahujícího obsahu.

Pro zobrazení dialogu slouží abstraktní třída *DialogActivity*, která je odvozena (podobně jako *Activity*) od třídy *AppCompatActivity*. Zobrazení aktivity formou dialogu je zajištěno prostřednictvím nastavení příslušného schématu[5] u elementu aktivity v souboru *AndroidManifest.xml*. Příklady použití lze pozorovat např. na obrázku C.10.

Většina vytvořených aktivit zobrazených formou dialogu jsou umístěny v balíčku *activities.dialog*. Tyto dědí vlastnosti od popisované třídy *DialogActivity*, jejíž implementace je z malé části podobná implementaci třídy *Activity*, zejména přepis metod *setContentView* a *findViewById*. Z podstaty chybí správa uživatelského menu, které v případě dialogu nedává smysl.

Třída *DialogActivity* má ve svém layoutu připravený element pro výpis textového nadpisu a dále dvě akční tlačítka umístěné naspod, které již mají vyplněny textové hodnoty, jak je znázorněno např. na obrázku C.5. Podtřídám pak poskytuje metody pro manipulaci jak textového nadpisu, tak i akčních tlačítek, pokud jejich textový popis není vhodný při aktuálním použití. Dialogů v celé aplikaci s uživatelským rozhraním postaveném na totožném základu (nadpis, dvě tlačítka) existuje celá řada, proto byl zvolen právě popisovaný způsob.

[4] Jedná se o vrchní lištu zobrazující hodiny, stav baterie a další.

[5] Bylo použito schéma *Theme.AppCompat.Light.Dialog* dostupné v knihovně *Android Support v21*.

App

V podkapitole 2.2.3 byla představena třída *android.app.Application* včetně jejího životního cyklu. Instance této třídy vznikne ihned po startu aplikace a s využitím vhodného mechanismu pak může být přístupná odkudkoli v rámci aktuálního jmenného prostoru. Aby bylo popsaného docíleno, byla vytvořena třída *App*, která je odvozena právě od třídy *Application* a registrována v souboru *AndroidManifest.xml* prostřednictvím atributu *name* elementu *application*.

Třída *App* přepisuje metodu životního cyklu *onCreate*, ve které provádí počáteční inicializace privátních proměnných, jenž jsou prostřednictvím *getterů*[6] předmětem globálního přístupu. Především se jedná o inicializaci proměnných pro následující využití:

- Aplikační kontext — uchování instance třídy *android.content.Context*. Existují dva typy kontextů. Prvním je kontext aplikační. Tento se váže k životnímu cyklu třídy *android.app.Application*. Druhým typem je kontext aktivity s návazností na životní cyklus třídy *Activity* popsanou v kapitole 2.2.1. Ve většině případů dostačuje prvně jmenovaný. Použitím druhého jmenovaného nese rizika, při nesprávném použití, vzniku úniků paměti (anglicky *Memory Leaks*). Proto je uchováván aplikační kontext redukující možnost vzniku popsaného problému.
- Načítání obrázků — inicializace instance třídy *ImageLoaderPicasso* z balíčku *application.imageLoader*, starající se o načítání a zpracování obrázků pomocí knihovny *Picasso*.
- Internetová komunikace — třída *App* plní funkci globálního přístupového bodu pro komunikaci aplikačních komponent s prostředím internet, princip bude představen v podkapitole 6.2.
- *GoogleApiClient* — pro celou aplikaci existuje pouze jedna instance třídy *GoogleApiClient*. Pro inicializaci je třeba instance třídy *Context* (aplikační nebo kontext aktivity). Pokud byla inicializace prováděna v rámci aktivity s použitím aplikačního kontextu s cílem průběžného získávání aktuální lokace zařízení, tak i přes využití aplikačního kontextu vznikal únik paměti. Problém má společnost Google nahlášen již od verze 8.1.0 knihovny *Google Play Services*, tento však doposud nebyl vyřešen. Proto bylo zvoleno popsané řešení.

Druhou a poslední přepsanou metodou životního cyklu je *onLowMemory*. Ta bude operačním systémem volána jako žádost o uvolnění nepotřebných aplikačních zdrojů, zpravidla během doby, kdy je aplikace minimalizována, a to pro redukci nebezpečí jejího ukončení. Vyvíjená aplikace používá dvě relační databáze (budou v budoucnu krátce popsány), jenž jsou obsluhovány knihovnou *SQLite* vysvětlenou v podkapitole 2.4.4. Pokud je otevřené databázové spojení, bude metodou *onLowMemory* uzavřeno společně s provedením úklidu souvisejících zdrojů.

6.1.2 Balíček Ui

Samotný balíček *Ui* přímo neuchovává žádnou třídu. Jeho obsah tvoří další balíčky spravující vzhled a manipulaci s uživatelským rozhraním. Jedná se o třídy plnící funkci adaptérů, které pro svou činnost vyžaduje třída *RecyclerView*, dále správa tříd poskytující implementaci jednoduchých animací a konečně samotné prvky uživatelského rozhraní.

[6] Pro metody getters a setters nejspíše vhodný český překlad neexistuje.

Adaptéry

Ve výsledné aplikaci se vyskytuje poměrně velké množství obrazovek zobrazující seznamy určitých prvků. Například zobrazení bariér je možné na mapovém podkladu (obrázek C.8a), nebo prostřednictvím vertikálního seznamu (snímek C.8b). Zároveň se tyto seznamy mohou vyskytovat v rámci aplikace na více obrazovkách. Ve zmíněném příkladu zobrazení seznamu bariér se tento vyskytuje i na hlavní obrazovce, jak ukazuje obrázek C.3b. Pro zamezení duplicity kódu implementace byly vytvořeny třídy adaptérů umístěné v balíčku *ui.adapters*.

Prostřednictvím funkcionality konkrétního adaptéru je pak umožněno zobrazit různé datové množiny prvků se stejným datovým typem pomocí totožného layoutu. Díky tomu je tedy odstraněna zmíněná duplicita kódu (s tím souvisí i jeho udržovatelnost), která by vznikala bez použití adaptéru a zároveň je zajištěn požadovaný konzistentní vzhled zobrazení prvků stejného datového typu.

Pro tyto účely byla vytvořena abstraktní třída *RecyclerViewAbstractAdapter* v konstruktoru přijímající instanci třídy aplikačního kontextu, díky které je nepřímo schopna poskytovat objekt třídy *LayoutInflater* sloužící pro možnost vytvoření layoutu reprezentující řádek seznamu. Dále poskytuje abstraktní třídu *ViewHolder*, kterou povinně používají konkrétní adaptéry. Tato slouží pro uchování layoutu řádku seznamu, který je pak recyklován. Recyklace řádků je hlavní funkcionalita adaptérů, jelikož naplnění layoutu prostřednictvím instance třídy *LayoutInflater* společně s nalezením všech požadovaných UI elementů je výpočetně náročnou operací. Instance třídy *ViewHolder*, resp. layout který reprezentuje, je pak naplňována konkrétními hodnotami vždy dle aktuálního vertikálního posuvu seznamu.

Jak již z názvu abstraktní třídy *RecyclerViewAbstractAdapter* implementující funkcionalitu adaptéru plyne, je tato určena pouze pro použití s elementem uživatelského rozhraní *RecyclerView*. Využití konkrétních adaptérů bude v budoucích podkapitolách zmíněno.

Widgety

Element uživatelského rozhraní platformy Android se v anglické literatuře obvykle označuje jako *widget* (většina elementů je obsažena v balíčku frameworku *android.widget*). Doposud byl tento v textu označován jako element, což by při uvážení definice rozložení layotu prostřednictvím XML souboru mělo dávat smysl. Dále mohou být pojmy zaměňovány.

Pro zobrazení textových hodnot ve výsledné aplikaci není použit přednastavený font, místo něj byl zvolen *TitilliumWeb*. Změnu fontu nelze vynutit prostřednictvím stylu, obyčejně definovaného v souboru *res/values/styles.xml* (viz podkapitola Adresářová struktura projektu 2.3.3), nýbrž v programové části. Proto třídy veškerých widgetů, jejichž obsah zcela nebo částečně tvoří textová hodnota, byly přepsány a pomocí metody *android.widget.setTypeface* lze vlastní font vynutit. Tímto způsobem je zaručen konzistentní vzhled textových elementů napříč všemi podporovanými verzemi platformy Android.

Třídy widgetů, rozšiřující vlastnosti původních poskytovaných frameworkem, jsou umístěny v balíčku *ui.widgets*. Některé obsažené třídy nerozšiřují žádné vlastnosti, avšak z pohledu budoucího vývoje se jedná o vhodnou praktiku.

Animace

V současné době většina aplikací, včetně mobilních, využívá nejrůznější animace pro zlepšení uživatelského prožitu během používání aplikace. Pro tento účel slouží následující abstraktní třídy z balíčku *ui.animations*:

- *AnimationExpandCollapse* — pomocí metody *expand*, resp. *collapse* virtuálně rozbalí, resp. sbalí widget ve vertikálním směru. Animace funguje na principu postupné změny výšky widgetu. Je použita např. v obrazovce volby filtru zobrazených budov (obrázky C.5a a C.5b).

- *Animations* — poskytuje několik metod pro přesun objektu na pozici relativní vůči jeho rodiči (s uvažováním stromové hierarchie widgetů rodič–potomek). Animace jsou použity např. na obrazovce zobrazení bariér během přepínání mezi mapovým zobrazením a seznamem (obrázky C.8a a C.8b).

6.1.3 Ostatní balíčky

Podkapitola 6.1.1 se detailně zabývala vybranými třídami, obsaženými v balíčku *Application*, tvořícími základ implementace celé aplikace, jako např. *Activity*, *DialogActivity* a *App*. Dalším popsaným balíčkem byl *Ui*, kterému se věnovala podkapitola 6.1.2 popisující princip implementovaných adaptérů, widgetů a tříd poskytujících jednoduché animace. Pro úplnost následuje alespoň stručný popis zbývajících hlavních balíčků:

- Activites — obsahuje veškeré třídy, jejichž základem je abstraktní třída *Activity*. Vzdáleným předkem je tedy třída *android.app.Activity* popsaná v podkapitole 2.2.1. Balíček *activities.dialog* obsahuje podtřídy třídy *DialogActivity* implementující funkcionalitu zobrazení dialogu.

- Model — tvořen dvěma balíčky. Prvním je *model.entities* zahrnující větší množství tříd reprezentující entity, které slouží pro reprezentaci výsledku internetové komunikace. Dalším je balíček *model.application*, taktéž zapouzdřující entity, tentokrát určené pro lokální využití aplikací, např. ukládání nastavení filtru bariér. Obojí bude podrobněji diskutováno později.

- Fragments — zahrnuje více balíčků, které převážně obsahují třídy založené na abstraktní třídě *Fragment* z balíčku *application*. Nepřímým předkem zmíněných je třída *android.app.Fragment*[7], jenž byla společně s životním cyklem představena v podkapitole 2.2.2.

- AsyncTask — balíček původně obsahoval veškeré podtřídy třídy *android.os.AsyncTask* popsané v podkapitole 2.2.4. Byl však zvolen přístup opačný. Patřičné třídy jsou umístěny blíže místu použití, nebo jsou třídami privátními, pokud jsou využívány pouze lokálně.

Entity obsažené v balíčku *model.entities* implementují rozhraní *Parcelable* (podkapitola 2.4.3). Jejich instance tak mohou být umístěny do objektu třídy *Bundle* (podkapitola 2.4.2). Této skutečnosti se dá využít pro uchování stavu aktivity a jejího opětovné vytvoření, během něhož je volána metoda životního cyklu *onCreate* přijímající objekt třídy *Bundle*, jehož obsahem může být v minulosti uchovaná entita. Podobný způsob je využit i pro předávání entity nově vytvářené aktivity. Tyto skutečnosti nebudou dále v textu explicitně jmenovány.

[7] Přesněji řečeno, předkem je třída z knihovny Android Support v4, a to android.support.v4.app.Fragment. Důvodem je zachování zpětné kompatibility pro zařízení s nižší verzí API. Životní cyklus fragmentu a další obecné informace uvedené v teoretické části zůstávají platné.

6.2 Požadavky na vzdálené zdroje

Kapitola 5 popisovala data dvojího charakteru (statická a dynamická), která jsou uložena vzdáleně na serverové části. Klientská aplikace tak musí mít možnost v případě potřeby se na data dotázat a přijatou odpověď ve formátu JSON vhodně zpracovat (jiné formáty odpovědi, vyjma fotografií, nejsou uvažovány).

Byla popsána třída *App*, která má plnit funkci globálního přístupového bodu pro komunikaci s vnějším prostředím. Pro tyto účely poskytuje následující metody:

- *performRequest* — přijímá objekt třídy *Request* z balíčku *application.connection* zapouzřující URL dotazu, třídu entity reprezentující odpověď na dotaz a instanci třídy implementující rozhraní *ResponseListener* sloužící pro asynchronní oznámení konce provádění dotazu.

- *performRequestBackend* — podobná předcházející, avšak přijímá objekt třídy *RequestBackend*, která rozšiřuje třídu *Request* o možnost přidaní do HTTP dotazu metody POST tělo ve formě *form-data* včetně případných souborů. V neposlední řadě taktéž možnost umístění hlaviček dotazu. Pro asynchronní oznámení konce provádění dotazu nyní slouží rozhraní *ResponseBackendListener*.

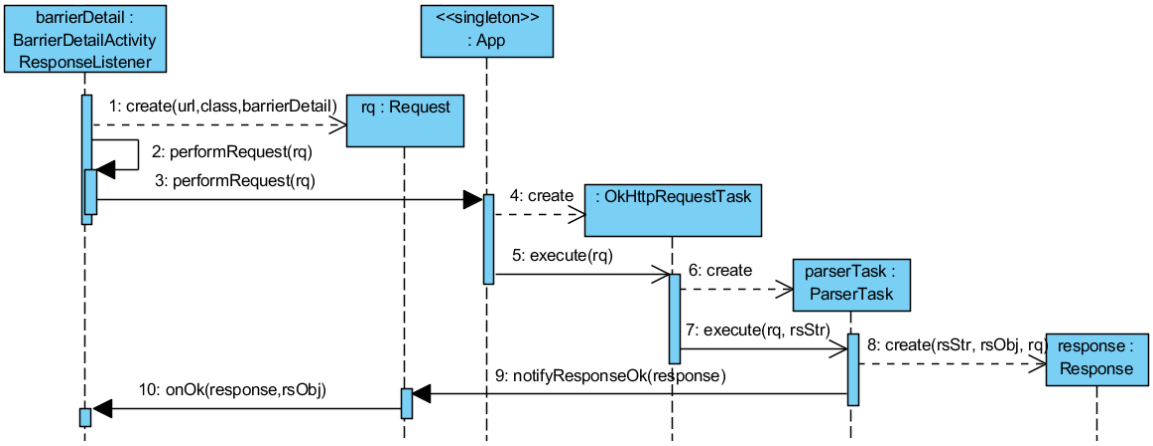
Rozdílem mezi výše uvedenými metodami je jejich univerzálnost z hlediska použití. Metoda *performRequest* je určena pro požadavek na jakýkoli zdroj, jehož výsledek ve formátu JSON lze deserializovat na objekt předávané třídy (reprezentující entitu). Používá se např. pro získání metadat služby *Street View*, která značí jejich dostupnost pro specifickou lokaci. Metoda *performRequestBackend* je specifičtější, slouží pouze na provádění požadavků zasílaných na implementované REST API obsluhující dynamická data nebo na požadavky dat statických (rozdíl uveden v kapitole 5). Samozřejmostí je použití první jmenované (obecnější) namísto druhé (specifičtější), ovšem opačný přístup platit nemůže.

6.2.1 Princip provedení požadavku

Klientská aplikace sestává z mnoha klasických aktivit a dialogů umístěných v balíčku *activities*. Navíc, každá aktivita nebo dialog může obsahovat libovolný počet fragmentů založených na abstraktní třídě *Fragment* (popsané v podkapitole 2.2.2) z balíčku *fragments*, nebo může být fragment navázán na aktivitu bez nutnosti poskytovat uživatelské rozhraní. Všechny uvedené aplikační komponenty by měly mít možnost zasílat požadavky a přijímat odpovědi. Z toho důvodu vyjmenované komponenty poskytují stejnojmenné metody (*performRequest* a *performRequestBackend*). Jejich implementace pouze získá instanci třídy *App*, které následně předá řízení požadavku.

Příklad úspěšného provedení obecnějšího požadavku na metadata služby *Street View* aktivity *BarrierDetailActivity* (její význam bude vysvětlen později) je znázorněn na obrázku 6.2 zobrazující zjednodušený sekvenční diagram. Označení volání metod bude uvedeno pořadovým číslem v závorce.

V aktivitě je nejprve vytvořen objekt třídy *Request* (1), jehož konstruktor přijímá URL dotazu, třídu specifikující entitu určenou pro deserializaci JSON odpovědi na dotaz (označené jako *class*) a instanci třídy implementující rozhraní *ResponseListener* (v uvedeném příkladu aktivita sama toto implementuje) sloužící pro předání výsledku nebo oznámení o případné chybě během provádění. Následně je volána metoda abstraktní třídy *Activity* (*BarrierDetailActivity* je podtřídou zmíněné) *performRequest* (2), která pouze předá požadavek pomocí volání stejnojmenné metody ve třídě *App* (3). Ta vytvoří instanci třídy



Obrázek 6.2: Princip provedení úspěšného požadavku

OkHttpRequestTask (4), předá jí objekt třídy *Request* (5), která následně v samostatném vlákně provede pomocí knihovny *OkHttp* dotaz na vzdálený server a příjme textovou odpověď ve formátu JSON (*rsStr*). Pokud nenastala žádná chyba, vytvoří (6) a spustí (7) v novém vlákně provádění deserializace prostřednictvím třídy *ParserTask*. Po úspěšné deserializaci tato vytvoří objekt třídy *Response* (8) obsahující původní odpověď (rsStr), deserializovaný objekt (rsObj) a originální instanci třídy *Request* (*rq*), kterou zároveň upozorní o výsledku (9). Třída *Request* pak pomocí uchované instance třídy implementující rozhraní zašle zprávu o úspěchu celé operace (10).

Byl uveden princip provedení úspěšného obecného požadavku. Specifický úspěšný požadavek (metoda *performRequestBackend*) bude proveden totožnými kroky s menšími odchylkami (odlišné metody v krocích 1 až 3).

Na první pohled se může jevit princip provedení požadavku zbytečně složitý. Avšak s ohledem na udržovatelnost se nemusí jednat o špatné řešení. Příkladem může být změna knihovny provádějící dotazy (internetovou komunikaci mezi klientem a serverem). V původním řešení byla použita knihovna *Volley*. Později byl zjištěn zásadní nedostatek, a to nemožnost nativního zasílání souborů pomocí HTTP metody POST zmíněnou knihovnou. Řešení spočívalo v záměně použité knihovny, a to z *Volley* na *OkHttp*, která je obecnější a přizpůsobitelnější. Tato záměna znamenala zaměnění pouze jednoho článku popsaného řetězce (*OkHttpRequestTask*). Ostatní zůstalo nedotčeno, neboť princip provedení a způsob doručení odpovědi prostřednictvím zavedeného rozhraní zůstalo stejné.

6.2.2 Požadavky na obrazové zdroje

Požadavky na načtení fotografie a její následné zobrazení do widgetu *ImageView* přijímá třída *App*. Ta pro tento účel poskytuje celkově tři metody, z nichž dvě jsou přetížené:

- *loadImage* — jedná se o nejpoužívanější metodu. Vždy přijímá instanci třídy *ImageView* reprezentující prvek uživatelského rozhraní, do kterého se má získaný obrázek vykreslit. Dalším parametrem je jedna z následujících možností: přímá URL zdroje, objekt třídy *User* reprezentující uživatele aplikace nebo objekt třídy *Barrier* reprezentující bariéru. V případě zmíněných objektů bude automaticky sestavena příslušná URL dotazu. Posledním parametrem může být instance třídy implementující rozhraní *LoadImageListener*, která informuje posluchače o úspěchu operace nebo nastalé chybě.

Provedení dotazu, jeho zpracování a zobrazení výsledku má na starosti knihovna *Picasso*.

- *loadImageVolley* — metoda není přetížena. Přijímá vždy instanci *ImageView*, URL zdroje a instanci *LoadImageListener*. Dotaz a zpracování je prováděno za pomoci knihovny *Volley*. Tato metoda je použita pouze v aktivitě *ZoomImageActivity* zobrazující obrázek a podporující funkci přiblížení a oddálení obrázku.[8]
- *loadImageFromFile* — slouží pro načtení obrázku z úložiště zařízení pomocí knihovny *Picasso*.

6.3 Inicializace aplikace během startu

V souboru *AndroidManifest.xml* (podkapitola 2.3.4) musí být definována právě jedna aktivita jako hlavní, spouštějící se bezprostředně po startu aplikace. Takovou byla zvolena *StartActivity* z balíčku *activities*, jejímž úkolem je inicializace většiny datových struktur potřebných pro následující běh aplikace. K tomuto účelu využívá startovací moduly implementované pomocí fragmentů, které neposkytují žádné uživatelské rozhraní, pouze provádí specifikované akce. Aktivita zobrazí informaci informující uživatele o průběhu inicializačních akcí, jak znázorňuje obrázek C.15c. Po dokončení potřebných operací bude uživateli animováno oznámení za pomocí třídy *Animations* z balíčku *ui.animations* (podkapitola 6.1.2).

Aktivita *StartActivity* tedy pouze vytvoří inicializační fragmenty (moduly), spustí provádění a vyčká na jejich dokončení. Pokud nastane v průběhu inicializace jakákoli chyba, např. nedostupnost internetového připojení, zobrazí chybovou obrazovku, díky kterému může uživatel akci opakovat.[9]

Dále bude následovat popis jednotlivých inicializačních modulů, přičemž bude vynechán modul *Location*, který zjišťuje přibližnou polohu zařízení dle aktuální IP adresy. Takto zjištěná poloha nakonec nebyla v implementaci využita, je však připravena pro budoucí případné použití.

6.3.1 AutoLogin

Třída *App* poskytuje metody pro uchování vybraných objektů perzistentním způsobem prostřednictvím *SharedPreferences* (podkapitola 2.4.1), se kterou je vždy manipulováno pomocí stejnojmenné třídy umístěné v balíčku *application*. Třída *SharedPreferences* serializuje objekt pomocí knihovny *Gson* do textové reprezentace. Taktéž poskytuje metodu pro budoucí deserializaci.

Jedním z vybraných objektů uložených v *SharedPreferences* je objekt třídy *AppUser* reprezentující uživatele aplikace (přihlášeného i nepřihlášeného). Modul *AutoLogin* zjistí, zda-li je uživatel přihlášen a v kladném případě zašle požadavek na REST API pro obnovení platnosti uživatelského tokenu a následně ukončuje činnost. V záporném případě nastává ukončení činnosti ihned po startu modulu.

[8] ZoomImageActivity s metodami *loadImage* využívající knihovnu *Picasso* nepracovala korektně.

[9] *StartActivity* jako všechny aktivity aplikace dědí od abstraktní třídy *Activity*, která již má připraveny layouty reprezentující obrazovky i pro chybové stavy, což bylo diskutováno v podkapitole 6.1.1 věnující se struktuře balíčku *application*.

6.3.2 SyncStart

Dalším modulem provádějící činnost během startu aplikace je synchronizační modul. Tento spravuje statická data (podkapitola 5.1) reprezentující vybrané budovy města Brna.

Aplikace, dle analýzy požadavků, musí umožnit zobrazení a přístup k detailním informacím části vybraných budov. Startovací modul naplňuje lokální databázi *SQLite* (podkapitola 2.4.4), ve které osm tabulek obsahuje řádky reprezentující budovy jednotlivých vrstev. Jedná se o vrstvy s názvem počínaje Kultura a konče Zdravotnictví, jejíž kořenem je vrstva Objekty. Rozdělení je patrné z tabulky 3.1.

Samotná manipulace s databází *SQLite* je prováděna prostřednictvím knihovny *ORMLite*, která poskytuje funkcionalitu objektově relačního mapování řádků tabulek na entity reprezentované třídami, které jsou umístěny v balíčku *model.entities.gisBrno*. Byla zmíněna existence osmi tabulek reprezentující jednotlivé vrstvy, ve skutečnosti existuje dalších pět, které např. uchovávají informace o dostupných piktogramech a textových hodnotách popisu budovy v českém a anglickém jazyce. Výsledné databázové schéma přibližně odpovídá struktuře JSON odpovědi získané pomocí podpůrných služeb *ArcGIS REST API* města Brna, popsané v kapitole 3.1.2. Přesná podoba schématu databáze není pro budoucí text podstatná, proto není uvedena.

Startovací modul na začátku své činnosti zkontroluje dostupnost databáze. Pokud byla v minulosti vytvořena a naplněna, prověří stáří obsažených dat. Jestliže data byla uložena před více než měsícem, databáze bude zahozena a opětovně vytvořena. Dále bude zaslán dotaz pro získání statických dat jednotlivých vrstev obsahující budovy (struktura dotazu s odpovědí byla popsána v podkapitole 5.1). Těmito pak bude aktuálně prázdná databáze naplněna.

Poslední činností (s možným přeskočením opětovného naplnění databáze) startovacího modulu je inicializace seznamů budov, kde každý seznam (myšleno jako instance třídy *java.util.ArrayList*) reprezentuje budovy jedné vrstvy. Seznamy jsou uchovány v objektu třídy *AppResources* umístěné v balíčku *application*. Objekt zmíněné třídy je pak možné získat během celé doby běhu aplikace prostřednictvím třídy *App*. Díky tomu již nemusí být prováděny potenciálně pomalé dotazy na databázi.

6.3.3 CheckGooglePlaces

Detail budovy (podkapitola 6.4.3) obsahuje kartu tzv. rozšířených informací, jak je patrné z obrázku C.6b. Tyto informace jsou získávány prostřednictvím služby *Google Places* s jistými omezeními. Jedná se zejména o počet možných dotazů za časové rozmezí 24 hodin. Poté se doposud bezplatná služba stane nepřístupnou a pro zachování její funkcionality je třeba uhrazení poplatků.

Samotná společnost Google doporučuje odpovědi získané uvedenou službou uchovávat lokálně s platností maximálně jeden měsíc, což se také v případě vyvíjené aplikace děje. Data jsou uchovávána, podobně jako v případě modulu *SyncStart*, prostřednictvím *SQLite* databáze. Modul *CheckGooglePlaces* pak provádí kontrolu stáří dat. Všechny řádky tabulky s časovým razítkem starším než jeden měsíc, tudíž nesplňující podmínku, budou smazány.

6.4 Budovy města Brna

S přihlédnutím na specifikované detaily případů použití (podkapitola 4.2.1) výsledná aplikace poskytuje možnost zobrazení vybraných budov města Brna. Další funkcionalitou je

jejich filtrování a konečně zobrazení detailu budovy. Popis jednotlivých funkcionalit následuje.

6.4.1 Zobrazení budov

Funkci zobrazení budov poskytuje aktivita *BuildingsActivity*. Tato, stejně jako zbytek aplikace, využívá pro zobrazení mapového podkladu služby balíčku *Google Play Services API*, resp. jeho části *Google Maps*.

Ukázka vzhledu obrazovky *BuildingsActivity* lze pozorovat na obrázku C.4a. Tento znázorňuje mapu při vysoké úrovni přiblížení. Jednotlivé budovy jsou reprezentovány ikonou, kde budovu nepřístupnou, přístupnou nebo přístupnou s asistencí značí postupně ikona červené, zelené nebo žluté barvy. Každá ikona zároveň obsahuje miniaturu značící kategorii, do které budova náleží (může náležet právě do jedné kategorie). Jako nedostatek lze hodnotit špatnou kvalitu obrázků reprezentující ikony. Tyto byly získány z webové aplikace Bezbariérové objekty popsané v podkapitole 3.1. Ikony jsou dostupné pouze v nízké úrovni rozlišení a nemohou být tedy umístěny do patřičných *drawable* složek adresářové struktury projektu (podkapitola 2.3.3). Poté by bylo frameworkem zajištěno použití správné verze ikony dle aktuální hustoty obrazovky (viz podkapitola 2.1). Uvedené má za následek vyšší nároky na výpočetní výkon a s tím související problémy. Obrázek totiž musí být uměle zmenšován (nebo zvětšován) dle aktuální hustoty displeje.

Mapový podklad by se za předpokladu současného zobrazení vyššího počtu ikon stal nepřehledný, proto byla za pomocí knihovny *Google Maps Android API utility* implementována funkcionalita shlukování sousedních ikon, tzv. „clustering“. Popsanou situaci znázorňuje snímek C.4b. Byl tedy odstraněn nedostatek uvedené webové aplikace, která zobrazovala ikony jen při velmi vysoké úrovni přiblížení.

Aktivita *BuildingsActivity*, jako rozšíření, podporuje změnu podkladové vrstvy. Uživatel má na výběr jednu ze čtyř možností (normální, hybridní, satelitní a terénní). Situaci lze pozorovat na snímku C.5c zobrazující okno dialogové aktivity spuštěné pomocí patřičného tlačítka menu v prvku *Toolbar*.

Atributy budov (např. poloha, název, kategorie) jsou přístupné kdekoli v programové části aplikace v rámci aktuálního jmenného prostoru. Tyto atributy spravuje třída *AppResources*, která byla inicializována na startu aplikace pomocí modulu *SyncStart* popsaného v podkapitole 6.3.2. Jejich získání je tedy jednoduché, a to prostřednictvím třídy *App*.

Nutno podotknout, že atribut lokace budovy v originále obsahuje souřadnice v souřadnicovém systému S-JTSK, se kterým pracuje i zmíněná webová služba. Avšak *Google Maps* pracují se souřadnicovým systémem WGS-84. Synchronizační modul *SyncStart* tedy mj. provádí i převod souřadnic do druhého jmenovaného s pomocí metod třídy *ConvertJTSK* umístěné v balíčku *application.utils.convert.cts*. Převod mezi souřadnicovými systémy není triviální záležitostí. Pro tento účel bylo prozkoumáno více knihoven, nakonec byla použita knihovna *CTS* napsaná v programovacím jazyce Java s přesností převodu plně vyhovujícímu danému použití.

6.4.2 Filtrování budov

Na mobilní aplikaci byl dále specifikován požadavek možnosti filtrování budov. K tomuto účelu slouží dialog implementovaný pomocí aktivity *BuildingsFilterDialogActivity*, jehož vzhled je zobrazen na obrázku C.5a a který je spuštěn prostřednictvím ikony umístěné v horní liště obrazovky zobrazující budovy na mapovém podkladu.

Dialog umožňuje zadaní parametrů pro filtrování na úrovni celých kategorií budov (např. Ubytování, Kultura). Tuto funkcionalitu taktéž nabízela webová aplikace. Z pohledu autora práce se jedná o nedostatečné řešení. Uživateli je tedy nabídnuta možnost zadat parametry filtru na nižší úrovni, a to dle přístupnosti (fakticky úroveň podkategorií). Uvedeným způsobem může uživatel zvolit pouze budovy přístupné, nepřístupné nebo přístupné s asistencí, nebo libovolnou kombinaci uvedeného. Navíc je umožněno vybrat pouze budovy s přístupným bezbariérovým WC. Zmíněné volby jsou viditelné na obrázcích C.5a a C.5b.

Nutností bylo vytvoření speciálního widgetu – *CheckBoxThreeStates*. Jak možná z názvu plyne, vychází tento z klasického widgetu *CheckBox*, který rozšiřuje o třetí stav. Celkově jsou tedy dostupné stavy vybráno, nevybráno, částečně vybráno. Použití je viditelné na obrázku C.5b, kde kategorie Ubytování má vybrané všechny podkategorie, kdežto kategorie Kultura jen některé.

Existuje několik kořenových kategorií (např. zmíněná Kultura) obsahující vyšší počet podkategorií. Pro zvýšení přehlednosti jsou tyto na počátku skryty (obrázek C.5a). Jejich zobrazení se provádí až na požádání společně s jednoduchou animací, kterou poskytuje třída *AnimationExpandCollapse* (podkapitola 6.1.2).

Pro účel uchování nastavení filtru byla vytvořena třída *LayerFilter*, jejíž serializovaná reprezentace je ukládána perzistentním způsobem pomocí *SharedPreferences*. Nastavení filtru je tak zachováno i v případě opětovného spuštění aktivity, resp. celé aplikace.

6.4.3 Detail budovy

Po výběru budovy reprezentované ikonou na mapovém podkladu je spuštěna aktivita *BuildingDetailActivity* umožňující prohlížení detailu zvolené budovy. Sestává z hlavičky zobrazující jednu či více fotografií a těla, jehož obsah generují následující fragmenty (obrázek C.6 zobrazuje postupně náhledy obrazovek):

- *BasicDetailSectionFragment* — vykresluje základní statická data (podkapitola 5.1) vynikající svou potenciální vysokou přesností, neboť nepřímým poskytovatelem je samotné město Brno. Obsahují nejdůležitější údaje – přístupnost budovy (objektu) a dostupnost bezbariérového WC. Eventuálně budou zobrazeny piktogramy detailněji upřesňující přístupnost objektu, jejichž seznam lze pozorovat na obrázku A.2b. V případě dostupnosti jsou vypsány velmi detailní informace o budově užitečné především pro osoby na invalidním vozíku (spodní část obrázku A.2a). Textové hodnoty statických dat jsou dostupné v anglické a české verzi. V případě české a slovenské lokalizace zařízení bude použita česká verze, v opačném případě vždy anglická.

- *ExtendSectionFragment* — na požádání zobrazí rozšířené informace o budově získané prostřednictvím služeb *Google Places*, podrobněji bude vysvětleno níže.

- *UsersSectionFragment* — zobrazí seznam uživatelských komentářů k aktuální budově. Aktuálně přihlášenému uživateli aplikace umožní přidat vlastní komentář nebo upravit stávající (snímek C.7b). Uživatelé tak získávají možnost sdílet vlastní zkušenosti k danému místu nebo reagovat na nepřesnost uvedených statických dat (mohou být zastaralá).

Google Places

Společnost Google poskytuje množinu služeb *Google Places API Web Services* navracející místa zájmu (POI) v blízkosti specifikované lokace.

Jinými slovy, dotaz na webovou službu sestává ze souřadnic pozice v souřadném systému WGS-84, poloměrem virtuální kružnice udávající maximální vzdálenost od specifikované pozice, preferovanou jazykovou lokalizaci textových řetězců výsledku a konečně klíčová slova pro účely vyhledávání. Odpovědí je potenciálně neprázdný seznam POI, kde jednotlivé jsou jednoznačně identifikovány textovým identifikátorem (*placeId*). Další službou lze získat detailní informace o konkrétním místě specifikovaném uvedeným identifikátorem.

Množina atributů objektu popisující budovu města Brna neobsahuje atribut *placeId* služby *Google Places*, rozšířené informace tedy nemohou být získány přímo. Byl zvolen postup sestávající z následujících kroků:

1. Seznam potenciálních POI — získání všech POI v bezprostřední blízkosti budovy, jejíž lokace je známá. Během volby maximální vzdálenosti od budovy musí být brán zřetel na vzniklé nepřesnosti, zejména potenciálně nepřesná originální lokace budovy, nepřesnost převodu ze souřadného systému S-JTSK na WGS-84 (není jednoznačný) a poslední případná nepřesnost samotných mapových dat společnosti Google. Z přihlédnutím na zmíněné byla zvolena maximální vzdálenost 50 metrů, která je jistým kompromisem mezi počtem výsledků (s tím související velikost odpovědi) a pravděpodobnosti budoucího nalezení potenciálního výsledku.

2. Získání *placeId* budovy — předchozím bodem byl získán seznam POI potenciálně obsahující záznam o budově. Jak bylo řečeno, *placeId* není doposud známé. S přihlédnutím na dostupná data byla zvolena triviální metoda (a zároveň nejspíše jediná možná) spárování budovy a POI. Textové řetězce názvů budovy a POI jsou zbaveny vybraných speciálních znaků (např. znak hvězdičky, pomlčky) a rozděleny na jednotlivá slova. Jako výsledek párování je označeno POI s nejvyšším (a unikátním) počtem shodných slov originální budovy.

3. Získání detailu POI — pokud předchozí krok vedl k úspěšnému spárování, bylo nalezeno *placeId*, které je předmětem dotazu na detail POI reprezentující požadovaný výsledek, tj. rozšiřující informace o budově.

Funkcionalitu uvedeného postupu implementuje fragment *GooglePlacesLoaderFragment*, který neposkytuje žádné uživatelské rozhraní. Ke své činnosti využívá třídy entit umístěné v balíčku *model.entties.google.placesAPIWebServices* (všechny odpovědi služby *Google Places* jsou poskytovány ve formátu JSON).

Pro nalezení rozšířených informací o jedné budově je tedy třeba dvou dotazů (nalezení seznamu POI a detail POI) na službu *Google Places*. V podkapitole 6.3.3 byla zmíněna existence startovacího modulu *CheckGooglePlaces* kontrolující databází uchovávající data získaná službou *Google Places*. Jedná se o JSON odpovědi na dotazy získání detailu POI uvedené ve 3. kroku postupu. Pokud budou požadovány rozšiřující informace, které v minulosti (platnost jeden měsíc) byly získány, budou přeskočeny všechny kroky postupu a bude rovnou navrácen deserializovaný výsledek z lokální databáze. Ušetření dotazů je tak patrné, neboť *Google Places* je pro bezplatné využití omezeno.

Fotografie budovy

Hlavička detailu budovy obsahuje prostor pro umístění fotografií (viz snímky C.6). Mezi případnými fotografiemi je možný horizontální posuv prostřednictvím widgetu *ViewPager*, který spravuje množinu fragmentů, kde každý uchovává právě jednu fotografii.

První fotografie je získána prostřednictvím služby *Google Street View*. Bude zobrazena pouze tehdy, pokud je tato dostupná pro lokaci budovy (dotaz na dostupnost provádí koordinující aktivita *BuildingDetailActivity*). V kladném případě bude navíc zpřístupněna funkce *Street View*, kterou implementuje aktivita *StreetViewActivity*, prostřednictvím které uživatel může získat přehled o přístupnosti budovy před jejím samotným navštívením (snímek C.7c).

Ostatní fotografie budou načítány až na přání uživatele (snímek C.7a), přičemž tyto jsou získávány pomocí popsané služby *Google Places*. Fotografie jsou automaticky knihovnou *Picasso* ukládány do lokální *cache*, není nutno řešit optimalizaci pro zamezení opakovaných dotazů na službu *Google Places*.

Uživateli je přístupná funkce zobrazení fotografie přes celou obrazovku, a to pomocí aktivity *ZoomImageActivity*, která podporuje funkci přiblížení a oddálení.

6.5 Správa bariér

Pro podporu správy bariér slouží aktivita *MapListBarrierActivity*, která koordinuje dva následující fragmenty:

- *BarrierMapFragment* — na mapovém podkladu znázorňuje bariéry dvojího typu. Prvním je bariéra nepřekonatelná, reprezentovaná pinem zabarveným do červené barvy, druhým pak bariéra překonatelná s asistencí (pin žluté barvy). Situaci znázorňuje snímek C.8a. Samozřejmostí je, stejně jako v případě budov, shlukování sousedních pinů při nižších úrovní přiblížení pro zachování vyšší přehlednosti mapy.
- *BarrierListFragment* — zobrazuje vertikální seznam bariér (snímek C.8b) pomocí prvku *RecyclerView*. K tomuto účelu využívá služeb adaptéru *BarrierAdapter* (základ adaptérů byl obecně popsán v podkapitole 6.1.2).

Prvek seznamu obsahuje informace o uživateli, který překážku zadal, dále její platnost, textovou reprezentaci data vytvoření nebo úpravy záznamu překážky, přiřazenou kategorii a počet uživatelských komentářů. Následuje celkové hodnocení bariéry doplněné o počty kladných a záporných hlasů (princip hodnocení a bude krátce uveden později). Poslední je vykreslení mapy společně s umístěním překážky. Pro lepší výkonost je tato použita v tzv. *Lite* módu, který omezuje funkcionalitu prvku *MapView*, jenž je v tomto případě pro zobrazení mapy využit. Avšak omezená funkcionalita bohatě vystačuje aktuálnímu použití, při kterém je vyžadována především nízká výpočetní náročnost během vykreslování mapy v průběhu vertikálního posuvu seznamu prvků.

Fragmenty ke své činnosti vyžadují seznam bariér, který je předmětem zobrazení. V případě potřeby pomocí zpětného volání kontaktují rodičovskou aktivitu *MapListBarrierActivity*, která přepošle požadavek fragmentu *BarrierLoaderFragment* starající se o načtení seznamu bariér z implementovaného REST API. Poté následuje filtrace dle uložených kritérií, po které je seznam připraven k předání fragmentům pro účel zobrazení.

6.5.1 Filtrování bariér

Pro účely filtrování bariér byla vytvořena třída *BarrierFilterTask*, která je založena na třídě *AsyncTask*[10], provádějící požadovanou operaci v rámci samostatného vlákna.

[10] V implementaci celé aplikace pro provádění výpočetně náročnějších operací je využita třída *AsyncTask* popsaná v kapitole 2.2.4

Ke své činnosti získává pomocí zpětného volání výchozí seznam bariér a hodnoty nastavení filtru, reprezentované objektem třídy *BarrierFilter*. Tento zapouzdřuje informace zadané pomocí dialogové aktivity *BarrierFilterDialogActivity*, jejíž vzhled je znázorněn snímkem C.8c). Možné hodnoty nastavení filtru jsou následující:

- Platnost — definuje dobu platnosti bariér. Ta může být dočasná nebo trvalá. V prvním případě je nutnost specifikovat datum, do kterého mají být bariéry minimálně platné. Během definice filtru musí být zvolena alespoň jedna doba platnosti.
- Hodnocení — uživatelé mohou hodnotit konkrétní bariéru kladně nebo záporně. Rozdíl mezi počtem všech kladných a záporných hodnocení udává celkové skóre bariéry. Lze tedy zobrazit bariéry s libovolným, nebo pouze s kladným skórem (hodnocením).
- Překonatelnost — bariéry mohou být překonatelné nebo překonatelné s asistencí. Ve filtru musí být zvolena alespoň jedna překonatelnost.
- Kategorie — volba rodičovských kategorií bariér. Každá bariéra je přiřazena do právě jedné podkategorie, která má jako rodičovskou kategorii buďto Stavební nebo Objekt (viz podkapitola 4.1.1). Pro účely filtru musí být zvolena alespoň jedna rodičovská kategorie.

Pokud možnosti filtru nebyly uživatelem doposud definovány, použije se výchozí nastavení. Rovněž je pro uživatele dostupné tlačítko pro resetování hodnot do výchozího nastavení.

6.5.2 Vytvoření bariéry

Funkce vytvoření a úpravu bariéry poskytuje aktivita *BarrierCreateUpdateActivity*. Pokud je požadována úprava, přijímá zmíněná objekt třídy *Barrier* reprezentující bariéru určenou k úpravě, v opačném případě bude objekt bariéry vytvořen a inicializován počátečními hodnotami. Dále veškeré změny provedené prostřednictvím prvků uživatelského rozhraní budou okamžitě reflektovány do vytvořeného (nebo přijatého) objektu. Výsledný dotaz na REST API pro vytvoření (nebo úpravu) bariéry se pak vytváří z atributů uvedeného objektu.

Obrázek C.11 ukazuje jednotlivé snímky aktivity během úpravy bariéry. Již vybraná lokace je znázorněna pomocí pinu umístěného na Google mapě reprezentované widgetem *MapView* v *Lite* módu (podobně jako v seznamu bariér). Pokud lokace bariéry (např. během vytváření) doposud nebyla specifikována, bude zobrazena zástupná ikona ohraničená obdélníkem značící možnost výběru souřadnic. Výběr se provede interaktivně, a to prostřednictvím aktivity *BarrierSelectLocationActivity*. Uživatel zadá souřadnice formou umístění pinu na mapový podklad a zadání potvrdí. Výsledek (souřadnice) je zaslán zpět aktivitě *BarrierCreateUpdateActivity*, která lokaci zobrazí popsaným způsobem (viz snímek C.11a).

Následuje výběr platnosti překážky. Pokud bude zvolena platnost dočasná, s animací se zobrazí vstupní pole pro zadání data platnosti s přednastaveným datem (aktuální datum plus jeden rok). Pokud přednastavená hodnota uživateli nevyhovuje, může prostřednictvím kalendáře zadat datum vlastní. Kalendář spravuje samostatná dialog aktivita *DatePickerDialogActivity* (snímek C.15d). Tato přijímá a vrací jako výsledek objekt své veřejné vnitřní třídy *DatePickerData* obsahující inicializační hodnoty pro prvek *android.widget.DatePicker*, zejména počáteční vybrané, maximální a minimální datum. Inicializační hodnoty nemusí být specifikované, v takovém případě aktivita provede inicializaci sama. V objektu vnitřní třídy je rovněž přítomný atribut pro uchování výsledku. Výsledkem výběru data tedy bude celý objekt, na konci zaslaný volající aktivitě.

Dalšími požadovanými informacemi o bariéře je její překonatelnost (nepřekonatelná, překonatelná s asistencí), vložení povinného textového komentáře a výběru podkategorie překážky (snímek C.11c). Po provedeném výběru budou vypsané podkategorie, z důvodu ušetření místa obrazovky, skryty (s použitím animace).

Konečně, uživatel má možnost překážku vyfotografovat pomocí fotoaparátu zařízení. K této akci je třeba získání (formou žádosti) nebezpečného systémového oprávnění (viz podkapitola 2.5). Pokud uživatel poprvé žádost zamítne, bude před provedeném budoucích zobrazen vysvětlující dialog. Jestliže uživatel provede celkový zákaz žádostí, bude rovněž zobrazen dialog s patřičným vysvětlením nastalé situace. Tento princip provádění žádostí o systémová oprávnění je použit i v ostatních případech (např. žádost o nebezpečné systémové oprávnění aktuální polohy) a nebude již explicitně uveden.

6.5.3 Detail bariéry

Po výběru konkrétní bariéry z libovolného seznamu nebo prostřednictvím mapy bude uživateli zobrazen její detail, implementovaný aktivitou *BarrierDetailActivity* (snímky na obrázku C.9). Tento sestává z hlavičky obsahující mapu reprezentující lokaci bariéry a těla, ve kterém jsou obsaženy informace o bariéře nebo seznam uživatelských komentářů (obojí formou samostatných fragmentů).

Po interakci se statickou mapou bude spuštěna aktivita *MapMarkerActivity*. Její obsah, zobrazený přes celou obrazovku, tvoří pouze mapové dílo s vyznačenou pozicí bariéry, doplněný o číselnou hodnotu souřadnic zeměpisné šířky a délky.

Informace o bariéře jsou přibližně totožné s informacemi zadávanými během vytváření či úpravy bariéry (viz předcházející podkapitola). Navíc je uveden uživatel aplikace, který překážku vytvořil, reprezentovaný profilovou fotografií, přezdívkou a případně plným jménem (snímek C.9a). Následuje výpis uživatelských hodnocení (obrázek C.9b) s možností vytvoření nebo úpravy vlastního pomocí dialogové aktivity *BarrierRatingDialogActivity* (snímek C.10a).

Výpis uživatelských komentářů znázorňuje obrázek C.9c. Uživatelé tak mají možnost přidat vlastní názor či doplňující informace o bariéře. Manipulace komentáře bariéry (založení, úprava) je patrná ze snímku C.10b.

6.6 Hledání cesty

Podkapitola 6.5 se věnovala správě bariér, tj. možnosti jejich vytváření a úpravy, zobrazení a filtrování. Výsledná aplikace by měla mít možnost vyhledat trasy v rámci bližšího centra města Brna a rozhodnout, zda jsou tyto bariérové nebo bezbariérové. Rozhodnutí provádí s využitím aktuální množiny bariér (téže překážek), které zadávají samotní uživatelé mobilní aplikace a tím přímo ovlivňují funkcionalitu rozhodování a v konečném součtu i výsledku vyhledávání tras.

K účelu vyhledání potenciálně bezbariérových tras byla využita služba *Google Maps Directions API*, která bude zkráceně popsána níže. Ve stručnosti, tato vrací maximálně tři trasy. Musí pak existovat mechanismus, který určí, zda je trasa bariérová nebo bezbariérová. V prvním případě pak bude provedena částečná optimalizace, neformálně řečeno, s cílem přeměny bariérové trasy na bezbariérovou.

V textu se vyskytovaly (a budou vyskytovat) pojmy bariérová a bezbariérová trasa. Pro úplnost následuje jejich neformální definice. Trasu označíme za bariérovou, pokud se

v její dostatečné blízkosti nachází alespoň jedna překážka. Opačně, trasu označíme za bezbariérovou, pokud v její dostatečné blízkosti neexistuje žádná překážka.

6.6.1 Google Maps Directions API

Společnost Google poskytuje webovou službu, zjednodušeně řečeno, vracející trasu mezi specifikovaným startovním a cílovým bodem. Trasa může být optimalizována pro právě jeden dopravní režim, a to automobilový, veřejná doprava, jízda na kole, chůze. Tato webová služba může být dále zkráceně označována jako *Google Directions*. Požadavky je možno zasílat prostřednictvím HTTP nebo HTTPS protokolu. Syntaxe požadavku je v obou případech totožná, vyjma absence (v případě HTTP protokolu) tzv. API klíče (v některých případech označován jako *Server KEY*), který jednoznačně identifikuje volající stranu pro účely monitorování počtu požadavků a s tím související účtování. Odpověď na dotaz nabývá, dle výběru, formátu JSON nebo XML. Dotaz může obsahovat velké množství volitelných parametrů, jejichž charakterizaci lze nalézt v dokumentaci služby. Následuje popis povinných a volitelných parametrů dotazu podstatných z hlediska implementace aplikace:

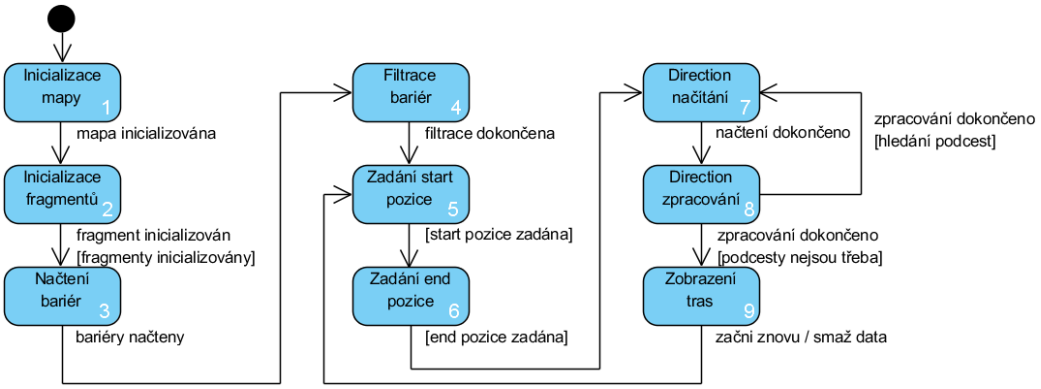
- *Origin* — souřadnice zeměpisné šířky a délky startovního bodu trasy.
- *Destination* — souřadnice zeměpisné šířky a délky koncového bodu trasy.
- *Mode* — dopravní režim výsledné trasy, pokud není uveden, bude použit režim automobilový.
- *Alternatives* — požadavek na vyhledání alternativních tras vzhledem k trase originální.
- *Key* — zmíněný API klíč pro účely účtování (HTTPS protokol).

Odpověď na dotaz z výše uvedenými parametry je dosti komplexní. V případě požadavku alternativních tras bude navráceno pole obsahující jednotlivé, které jsou dále strukturovány. Během experimentů byly vždy navráceny současně maximálně 3 trasy, jedna hlavní, dvě alternativní (v uvedeném pořadí). Trasa sestává především z atributů definující ohraničení trasy MBB[11], pole obsahující jednotlivé kroky trasy doplněné o textový popis. Dalším, z pohledu implementace aplikace nejdůležitějším atributem, je tzv. přehledová křivka, obsahující seznam souřadnic interpretující křivku, ze které se skládá výsledná trasa. Seznam souřadnic není uveden přímo, ale je s využitím ztrátové komprese kódován, jejíž výsledkem je textový řetězec. Díky této skutečnosti je redukována velikost celkové textové odpovědi JSON. Kódování a dekódování přehledové křivky není předmětem diplomové práce, pro tyto účely byla využita metoda z dříve uvedené knihovny *Google Maps Android API utility*. Podrobný popis a postup komprese lze nalézt v dokumentaci[12].

Aplikace bude využívat službu *Google Directions* s parametry specifikující trasy založené na režimu chůze. Zároveň budou požadovány trasy alternativní, odpověď bude akceptována ve formátu JSON.

[11] Anglický výraz *Minimal Bounding Box*

[12] Encoded Polyline Algorithm Format — https://developers.google.com/maps/documentation/utilities/polylinealgorithm [Online; navštíveno 15.05.2017]



Obrázek 6.3: Stavový diagram aktivity *RoutesActivity*

6.6.2 Koordinace hledání cesty

Aktivitou poskytující uživateli možnosti hledání bariérových a bezbariérových tras je *RoutesActivity*. Jedná se spíše o koordinátora posloupnosti akcí vedoucí ke splnění požadovaného cíle. Poskytuje uživatelské rozhraní pro zadávání souřadnic startovního a cílového bodu s možností opakování, dále funkcionalitu přepínání mezi nalezenými trasami a jejich zobrazení na mapovém podkladu. V případě bariérové trasy bude tato vykreslena, včetně startovního a koncového bodu, společně s překážkami náležící této trase. Ukázka *RoutesActivity* zobrazující bariérovou, resp. bezbariérovou trasu je znázorněna snímkem C.12a, resp. C.12b.

Na obrázku 6.3 jsou znázorněny jednotlivé očíslované stavy (dále bude v závorce uváděno číslo stavu), kterými během své existence aktivita *RoutesActivity* prochází, přičemž tato se v každém časovém okamžiku nachází v právě jednom stavu.

Po vytvoření aktivity systémem je spuštěna inicializace Google mapy (1), o jejímž dokončení bude aktivita informována registrovaným zpětným voláním. Následně nastává inicializace pomocných fragmentů (2), kterými jsou právě tři fragmenty sloužící postupně pro načtení bariér, obsluhu požadavků na *Google Direction* a zpracování odpovědí od *Google Direction*. Po dokončení inicializace všech fragmentů aktivita přechází do následujícího stavu (3), ve kterém je provedeno načtení všech bariér z REST API (fragment *BarrierLoaderFragment* uvedený v podkapitole 6.5), následuje jejich filtrace (4) pomocí třídy *BarrierFilterTask* uvedené v podkapitole 6.5.1.

Dále se očekává vstupní akce od uživatele, a to definování startovního bodu cesty (5). Pokud bylo přiděleno nebezpečné systémové oprávnění lokace, může uživatel zvolit startovní bod totožný s aktuální polohou zařízení, nebo tento zvolit interaktivně prostřednictvím mapy formou umístění a následném volitelném přetažení pinu. Druhá možnost je jedinou možnou, pokud oprávnění lokace nebylo uděleno. Po potvrzení volby je akce opakována pro specifikaci koncového bodu trasy (6). Následně budou prostřednictvím *Google Direction* získány trasy (7) následujíc jejich jednotlivé zpracování, během kterého je rozhodnuta otázka bezbariérovosti konkrétní trasy (8). Pokud bude nalezena alespoň jedna trasa bariérová, budou se stavy (7) a (8) za účelem optimalizace opakovat (princip bude vysvětlen později). Následuje vykreslení výsledných tras (9) z možností opakování procesu pro odlišný startovní a koncový bod, tj. návrat do stavu (5) předcházející zahození dočasných dat.

Uvedený princip popisoval koordinaci akcí bez výskytu chyb. Například nedostupné internetové připojení vede k přechodu aktivity do jednoho z chybových stavů, které pro přehlednost nejsou uvedeny.

Během provádění filtrace přijatých bariér (4) se bere v potaz nastavení filtru (podkapitola 6.5.1) uložené v objektu třídy *BarrierFilter* a uchovávaném perzistentním způsobem v *SharedPreference*. Pro rychlou změnu nastavení filtru je dostupná v rámci aktivity *RoutesActivity* ikona spouštějící patřičný dialog.

6.6.3 Získání a zpracování odpovědi Google Directions

Aktivita *RoutesActivity*, nacházející se ve stavu (7), pro získání tras využije fragment *GoogleDirectionsLoaderFragment*. Ten od aktivity příjme souřadnice startovního a koncového bodu, následně sestaví URL a zašle dotaz na službu *Google Directions*. Parametry dotazu byly popsány v podkapitole 6.6.1. Nepřímým výsledkem je pak seznam objektů třídy *GoogleDirectionRoute* reprezentující nalezené cesty volanou službou (jedna hlavní cesta a maximálně dvě alternativní). Seznam je jako odpověď zaslán zpět aktivitě *RoutesActivity*, která následně přechází do stavu (8), ve kterém přijatý seznam zasílá fragmentu *RoutesPrepareDataFragmet* ke zpracování.

Fragment pro každou cestu ze seznamu vytvoří a spustí výpočetní vlákno *PrepareRouteTask* a vyčká na jejich dokončení. Výsledkem každého bude naplněný objekt třídy *Route* reprezentující původní cestu s atributy obsahujícími dodatečné informace. Tyto objekty jako výsledek zasílá rodičovské aktivitě *RoutesActivity*, která rozhodne o dalším postupu.

Inicializace třídy Route

Třída *GoogleDirectionRoute* obsahuje atribut přehledové křivky (viz podkapitola 6.6.1). Pomocí knihovny *Google Maps Android API utility* lze tuto dekódovat a získat tak seznam souřadnic, pomocí kterých je výsledná trasa sestavena. Tato je tedy reprezentována lomenou čarou, která může být dále rozdělena na jednotlivé úsečky.

Uvažujme nyní trasu reprezentovanou lomenou čarou, resp. skládající se z jednotlivých úseček. Dále předpokládejme existenci bariér na této trase. Příkladem nechť je trasa na obrázku D.1a. Této náleží právě tři bariéry, jedná se tedy o trasu bariérovou.

Pro účely optimalizace trasy (podkapitola 6.6.4) třída *Route* obsahuje následující:

- Originální trasa *GoogleDirectionRoute* získaná prostřednictvím služby *Google Directions*, resp. fragmentu *GoogleDirectionsLoaderFragment*.
- Úsečky reprezentují lomenou čáru, resp. dekódovaný seznam souřadnic.
- Bariéry potenciálně spadající do MBB originální trasy.
- Bariéry dle úseček, jedné úsečce může být přiřazeno libovolný počet bariér, bariéře maximálně jedna úsečka.
- Informaci o celkové bezbariérovosti trasy, případné rodičovské a synovské cesty.

Na základě předchozího výčtu lze sestavit uspořádané seznamy úseček, které jsou graficky znázorněny a odlišeny na obrázku D.1b. Výčet uspořádaných seznamů následuje:

1. Celková trasa — úsečky trasy mezi startovním a koncovým bodem, reprezentované červenou barvou (viditelná pouze mezi první a poslední bariérou trasy, jinak skryta).

2. Před první překážkou — úsečky trasy mezi startovním bodem a první překážkou nacházející se na trase, reprezentované modrou a růžovou barvou.

3. Za poslední překážkou — úsečky trasy mezi poslední překážkou a koncovým bodem trasy, reprezentované zelenou a hnědou barvou.

4. Zkrácený úsek před první překážkou — úsečky mezi startovním bodem trasy a první překážkou se zvolenou maximální vzdáleností od první překážky, reprezentované růžovou barvou.

5. Zkrácený úsek za poslední překážkou — úsečky mezi poslední překážkou a koncovým bodem trasy se zvolenou maximální vzdáleností od poslední překážky, reprezentované zelenou barvou.

V předcházejícím výčtu byla uvedena položka 4, resp. 5 reprezentující seznam úseček, v jehož popisu se vyskytoval pojem maximální vzdálenosti. Tento udává součet délek úseček seznamu, který nabývá hodnoty maximálně jedné třetiny součtu délek úseček seznamu 2, resp. 3 nebo minimálně 50 m (délkové údaje se předpokládají v jednotkách metr).

Náležitost bariéry k úsečce

V předcházejícím textu byla popsána inicializace třídy *Route* (podkapitola 6.6.3), ve které bylo vyjmenováno několik uspořádaných seznamů úseček (reprezentující části trasy), např. seznam před první překážkou. Pro účely vytváření uvedených seznamů musí existovat mechanismus přiřazení překážky ke konkrétní úsečce.

Během vytváření bariéry uživatel definuje její umístnění, konkrétně formou virtuálního umístění pinu na mapový podklad. Většina překážek se tedy bude vyskytovat v blízkosti středů silnic a ulic zobrazených na mapě během procesu zadávání umístění (pokud uvážíme kategorie a podkategorie překážek definované v podkapitole 4.1.1). Widget *MapView* neposkytuje mechanismus, který by omezoval umístnění pinu pouze na souřadnice bodů náležící vybrané linii uliční sítě[13]. Taktéž není dostupný mechanismus označení konkrétní ulice či její části. Z těchto důvodů muselo být zavedeno zjednodušení, kdy je překážka uvažována jako bod a její umístnění je tedy specifikováno souřadnicemi zeměpisné šířky a délky.

Uvedené dále naznačuje existenci problému rozhodnutí náležitosti bariéry ke konkrétní úsečce. Příklad lze pozorovat na obrázcích D.5a a D.5b, kde překážky nejsou přímo umístěny na žádné úsečce tvořící výslednou křivku, nýbrž v její těsné blízkosti.

Uvažujme tedy nyní překážku umístěnou v blízkosti úsečky. Pseudo-algoritmus rozhodující, zda překážka náleží úsečce sestává z následujících kroků:

- Vypočti vzdálenosti bariéry od startovního a koncového bodu úsečky.

- Jestliže je libovolná z vypočtených vzdáleností menší nebo rovna hraniční hodnotě, překážka náleží úsečce a skonči. Jinak, jestliže je libovolná z vypočtených vzdáleností větší než délka úsečky, překážka nenáleží úsečce a skonči. Jinak pokračuj.

- Uvažuj pomyslný trojúhelník, jehož množina vrcholů je tvořena krajními body úsečky a bodem bariéry. V trojúhelníku vypočti výšku na stranu tvořenou z krajních bodů úsečky. Jestliže je, resp. není výška menší nebo rovna hraniční hodnotě, překážka náleží, resp. nenáleží úsečce. Skonči.

[13] Za předpokladu, kdy je uliční síť reprezentována neorientovaným grafem složeným z jednotlivých linií.

Byl uveden pojem hraniční hodnota. Tento značí maximální možnou vzdálenost překážky od úsečky, kde úsečkou předpokládáme osu ulice, resp. komunikace. Normy[14] stanovují minimální šířku jízdního pruhu 2,75 m, možnou existenci bezpečnostního odstupu 0,5 m a chodníku nespecifikované šíře. Z uvedeného byla zvolena hraniční hodnota 5 m, která je samozřejmě kompromisem (nelze získat informaci o počtu jízdních pruhů, možnou existenci bezpečnostního odstupu či autobusového nebo trolejbusového pásu).

Dále byl uveden pomyslný trojúhelník s požadavkem výpočtu výšky na specifikovanou stranu. Pokud označíme strany trojúhelníku jako a,b,c a výšku na stranu a označíme H, pak s využitím *Heronova vzorce* lze odvodit jednoduchý vzorec výpočtu výšky H, tj. vzdálenosti bariéry od úsečky, která reprezentuje část osy komunikace:

$$H = \frac{2}{a}\sqrt{s(s-a)(s-b)(s-c)}, s \ = \frac{a+b+c}{2}$$

6.6.4 Princip optimalizace

Aktivita nacházející se ve stavu (7) (viz obrázek 6.3) vyčkává na doručení seznamu objektů třídy *GoogleDirectionRoute* reprezentující odpovědi od služby *Google Directions*. Po jejich přijetí přechází do stavu (8) a očekává od fragmentu *RoutesPrepareDataFragmet* výsledek ve formě seznamu inicializovaných objektů třídy *Route*. Následně existují dvě varianty pokračování. Jestliže jsou všechny trasy jednotlivě reprezentované objektem třídy *Route* bezbariérové, přechází aktivita do stavu (9) zobrazujícího výsledné trasy. V opačném případě započne fázi optimalizace.

Každá bariérová trasa obsahuje neprázdný seznam zkráceného úseku před první, resp. za poslední překážkou, které byly uvedeny číslovaným výčtem v podkapitole 6.6.3 pod čísly 4, resp. 5 a graficky znázorněné na obrázku D.1b růžovou, resp. zelenou barvou.

Aktivita, aktuálně nacházející se ve stavu (8), všechny přijaté bariérové trasy umístí do tzv. seznamu čekajících tras. Posléze z něj vyjme první trasu a určí startovní, resp. koncový bod pro nové vyhledávání, který je roven koncovému, resp. počátečnímu bodu poslední, resp. první úsečky neprázdného seznamu zkráceného úseku před první, resp. poslední překážkou. Neformálně řečeno, startovní bod je roven začátku růžové lomené čáry, koncový pak konci zelené lomené čáry (obrázek D.1b). Následně aktivita přechází do stavu (7) a kontaktuje fragment *GoogleDirectionsLoaderFragment*, kterému předá získané body startu a cíle. Uvedený postup se opakuje do vyprázdnění zmíněného tzv. seznamu čekajících tras.

Nebyla zmíněna důležitá podmínka. Bariérová trasa je přidána do seznamu čekajících tras tehdy a jen tehdy, když její seznamy 4 a 5 (číslovaný výčet v podkapitole 6.6.3) jsou neprázdné.

Ukázkový příklad

Uvedené principy lze pozorovat na grafickém příkladu optimalizace bariérové trasy. První dotaz na službu *Google Directions* vrátí tři trasy, znázorněné na obrázcích D.4a, D.4b a D.4c. Dále budeme předpokládat navrácení vždy maximálního počtu tras službou *Google Directions*, tzn. právě tři trasy. Pro účely příkladu vybereme trasu první, zobrazenou zároveň i obrázkem D.1a. Dále pokračujeme neformálním výčtem akcí, v závorce uvedeno označení patřičného obrázku:

- Zpracování vstupní trasy (D.2a), je bariérová. Následuje dotaz na „pod-trasy“.

[14] ČSN 73 6100, ČSN 73 6110

- Příjem odpovědi na „pod-trasy“, zpracování, vybrána jedna (D.2b), je bariérová. Následuje dotaz na „pod-pod-trasy“.
- Příjem odpovědi na „pod-pod-trasy“, zpracování, vybrána jedna (D.2c), je bezbariérová, není nutný další dotaz, konec.

Pro uvedený příklad lze vyvodit maximální možný počet provedených dotazů. Jestliže bylo uvažováno navrácení maximálního počtu tras v každém dotazu, bude celkově provedeno právě 13 dotazů. První dotaz navrátí 3 trasy. Pro každou pak bude proveden samostatný dotaz, jehož výsledkem budou opět tři trasy. Situace se následně bude opakovat pro všechny přijaté trasy. Popsané si lze představit jako plný ternární strom, kde počet dotazů bude roven počtu uzlů při dané výšce.

6.6.5 Vytvoření výsledných cest

Byl uveden ukázkový příklad (podkapitola 6.6.4) optimalizace vybrané bariérové trasy, jehož výsledkem je několik „pod-tras“ s různou úrovní zanoření. Pokud navážeme na ukázkový příklad s přihlédnutím na zbylé dvě bariérové trasy (D.4b a D.4c), výsledkem budou právě tři plné ternární stromy, jejichž uzly tvoří „pod-trasy“, vyjma kořenů. Každý kořen je původní bariérová trasa, která je předmětem optimalizace.

Aktivita *RoutesActivity*, která je koordinátorem celého procesu optimalizace, uchovává všechny původní trasy a „pod-trasy“, vždy reprezentované třídou *Route*, v seznamu. Každý objekt třídy *Route* obsahuje odkazy na případné rodičovské a synovské cesty (viz nečíslovaný výčet v podkapitole 6.6.3). Respektive, pokud je prováděn dotaz na synovské cesty, ty budou přijaty, ihned jim bude přiřazena cesta rodičovská (uchování odkazu v atributu na zmíněnou), synovské cesty zůstanou prozatím neinicializovány (nejsou nyní známy).

Po dokončení všech dotazů, související s procesem optimalizace, aktivita za pomoci statické metody abstraktní třídy *RouteResultCalculator* získá výsledné trasy, přičemž metodě zašle veškeré doposud získané trasy (seznam objektů třídy *Route*). Proces získání výsledných tras probíhá následujícím způsobem:

1. Veškeré originální trasy (kořeny ternárních stromů, získané prvním dotazem na *Google Directions*) zařadí do výsledného seznamu. V globále se tedy může jednat o bezbariérové, tak bariérové, které podléhaly procesu optimalizace.
2. Následně bude všem trasám přiřazen seznam potomků, téže synovské cesty (viz nečíslovaný výčet v podkapitole 6.6.3). K tomuto účelu je využit fakt, kdy každá trasa má nastaven atribut odkazující na svého rodiče (pokud se jedná o kořen, rodič není definovaný).
3. Dále bude nalezen seznam tras, které jsou listy všech stromů (nemají definovány synovské cesty) a zároveň jsou bezbariérové.
4. Následně proběhne proces vytvoření výsledné cesty od všech listů (výčet č. 3) směrem k rodiči (bude uveden příklad).

Aktivita *RoutesActivity* předává metodě třídy *RouteResultCalculator* objekty třídy *Route*. Jako výsledek však předpokládá seznam objektů třídy *RouteResult*. Tento, zjednodušeně řečeno, obsahuje úsečky pro účel vytvoření lomené čáry reprezentující výslednou cestu, bariéry navázané k této cestě (v případě bariérové trasy) a vypočítanou celkovou délku cesty. Pouze tyto informace budou potřeba pro následné vykreslení tras na podkladovou mapu,

doplněné o možnost přepínání mezi trasami, uskutečňující se v posledním stavu (9) aktivity *RoutesActivity*.

Ukázkový příklad

Ukázka principu vytvoření výsledné trasy, která byla předmětem optimalizace, bude provedena pouze graficky. Uvažujme tedy bariérovou trasu, jejíž část je znázorněna na obrázku D.2a a její „pod-trasy“ D.2b a D.2c získané v minulosti uvedeným postupem (dále trasy reprezentované obrázky budou uvedeny v závorce).

Postup získání výsledné trasy je pak přímočarý. Trasa (D.2c) je listem, veškeré její úsečky proto budou přidány do výsledného seznamu. Jejím přímým rodičem je trasa (D.2b), ze které budou do seznamu přidány úsečky označené barvou modrou a zelenou. Rodičem trasy (D.2b) je pak (D.2a), která je zároveň kořenovou. Do výsledného seznamu budou umístěny úsečky modré a hnědé barvy.

Spojení úseček obsažených ve výsledném seznamu lze pozorovat na detailu výsledné trasy (D.3a), případně jejího celého zobrazení (D.3b). Během procesu spojování je využit fakt, kdy veškeré seznamy reprezentující úsečky uchované ve třídě *Route* (podkapitola 6.6.3) jsou uspořádány ve směru od startovního bodu k cílovému.

Závěrem, z původně bariérové trasy (D.1a) se třemi překážkami, byla procesem optimalizace získána trasa bezbariérová (D.3b).

6.7 Ostatní funkcionalita

Aktuální kapitola popisovala celkovou architekturu klientské části, princip provádění požadavků na vzdálené zdroje (např. serverová část REST API popsaná v kapitole 5), dále se věnovala implementaci týkající se zobrazení budov města Brna, bariér a konečně způsobem hledání cesty s možnou optimalizací.

Výsledná mobilní aplikace však poskytuje další funkcionalitu. Tato bude pro úplnost popsána, avšak z prostorových údajů pouze okrajově, bez větších implementačních detailů.

6.7.1 Registrace a přihlášení

Po startu aplikace a její inicializaci (podkapitola 6.3) bude v případě nepřihlášeného uživatele nastartována aktivita *NoActiveUserActivity*, jejíž obsah je znázorněn snímkem C.1a. Poskytuje základní textové informace o aplikaci s možností přihlášení prostřednictvím přístupových údajů, kterými jsou emailová adresa a heslo. Přihlášení je rovněž umožněno přeskočit a pokračovat jako nepřihlášený uživatel, který je charakteristický jistými omezeními (viz obrázek 4.1 znázorňující diagram případů použití pro různé aktéry).

Pro účely registrace slouží aktivita *RegisterActivity* s obsahem znázorněným obrázky C.1b a C.1c, přičemž jsou vyžadovány povinné údaje – přezdívka, emailová adresa a heslo. Volitelnými pak jméno, příjmení a datum narození, které je umožněno zadat interaktivně prostřednictvím v minulosti zmíněném dialogu *DatePickerDialogActivity* (obrázek C.15d). Po zadání registračních údajů a odeslání požadavku na REST API mohou být navráceny chybové kódy značící např. již existujícího uživatele se zadanou přezdívkou nebo emailovou adresou.

6.7.2 Domovská obrazovka

Výsledkem přihlášení, registrace nebo přeskočení uvedeného bude start aktivity *MainActivity* zobrazující domovskou obrazovku. Tato sestává z následujících sekcí (v uvedeném pořadí):

1. komentáře k budovám vytvořené aktuálním uživatelem,
2. komentáře k budovám vytvořené ostatními uživateli včetně aktuálního,
3. bariéry manipulované aktuálním uživatelem,
4. bariéry manipulované ostatními uživateli včetně aktuálního.

Uvedené sekce jednotlivě obsahují maximálně 5 prvků, přičemž tyto jsou řazeny dle data vytvoření, případě úpravy (obojí znační manipulaci), přičemž druhé jmenované má vyšší prioritu. Jestliže aktuální uživatel aplikace není přihlášen, sekce 1 a 3 budou vynechány. Ukázky vzhledu aktivity, resp. jednotlivých sekcí jsou uvedeny snímky C.3a a C.3b.

Všechny sekce jsou implementovány pomocí fragmentů a dynamicky přidávány rodičovskou aktivitou. Vzdálenější nadtřídou uvedených je abstraktní fragment *HomeSectionFragment*, dále fragmenty zobrazující bariéry, resp. komentáře jsou podtřídou abstraktní třídy *BarrierAbstractFragment*, resp. *BuildingCommentsAbstractFragment*. Uvedeným objektovým návrhem je zajištěna poměrně nízká duplicita kódu a snadná rozšířitelnost. Fragmenty pro zobrazení prvků v seznamech využívají již existující adaptéry z balíčku *ui.adapters* (viz podkapitola 6.1.2).

6.7.3 Detail profilu

Z několika míst aplikace lze spustit aktivitu *ProfileDetailActivity* zobrazující seznam záznamů typu bariéra, resp. komentáře, znázorněné na snímku C.13a, resp. C.13b. Jedná se o seznamy záznamů uvedených typů, jejichž autorem je právě jeden, vybraný, uživatel. Aktivita tyto seznamy zobrazuje zvlášť, s možností přepínání prostřednictvím dvou akčních tlačítek umístěných těsně pod vrchní lištou, tj. widgetem *Toolbar*. Seznamy jsou implementovány formou fragmentů, které mají nadřazenou třídu *ProfileDetailSectionFragment*, jenž je odvozena od třídy *HomeSectionFragment*. Tato je, jak bylo zmíněno, základní třídou pro jednotlivé sekce domovské obrazovky (podkapitola 6.7.2).

Přepínání mezi fragmenty, uchovávající seznamy bariér a komentáře, je prováděno za pomoci vybraných statických metod třídy *Animations* poskytující funkcionalitu jednoduchých animací (podkapitola 6.1.2).

6.7.4 Ostatní dialogová okna

Aplikace dále poskytuje vedlejší funkcionality dostupné pouze (až na jednu výjimku) přihlášenému uživateli. Tyto jsou zobrazeny formou dialogu, spouštěného patřičným prvkem menu aplikace (snímky C.2). Následuje jejich výčet (v závorce odkaz na snímek):

- Zpětná vazba (C.14a) — umožňuje uživateli zaslat textovou zpětnou vazbu. Tato bude reprezentována záznamem v tabulce *Feedback* relační databáze (viz obrázek 5.1).
- Sdílení aplikace (C.14b) — na požádání vytvoří text obsahující zkrácené informace o aplikaci společně s URL adresou odkazující na detail výsledné aplikace v distribuční službě *Google Play*. Text pak uživatel může prostřednictvím instalovaných aplikací

na aktuálním zařízení zaslat potenciálnímu novému uživateli, nebo sdílet na libovolné sociální síti.

- Hodnocení aplikace (C.14c) — podobné jako předcházející bod, avšak k přesměrování do distribuční služby *Google Play* dochází přímo, uživatel pak může ohodnotit aplikaci mechanismem poskytovaný distribuční službou.
- O aplikaci (C.14d) — zobrazení textových informací ohledně aplikace, licenční podmínky aj. Pouze tato funkcionalita z uvedených je dostupná nepřihlášenému uživateli.

6.7.5 Změna profilové fotografie

Uživatel během registrace povinně zadává přezdívku. Na základě této je pak generována, např. v seznamech komentářů, zástupná profilová fotografie. Tato sestává z kruhu vyplněného barvou právě jednoho odstínu z několika dostupných a počátečním písmenem přezdívky. Zástupná profilová fotografie je generována s využitím knihovny *TextDrawable*. Výběr konkrétního odstínu je ovlivněn hodnotou identifikátoru řádku tabulky *User* relační databáze, reprezentujícího uživatele. Příklad je patrný na snímku C.15a, který zároveň ukazuje vzhled aktivity *SettingsActivity*, umožňující změnu již existující, nebo zadání nové profilové fotografie, získané prostřednictvím fotoaparátu zařízení. Pro jmenované akce musí být uděleno nebezpečné systémové oprávnění využití fotoaparátu, v negativním případě libovolná není možná. Výsledek změny profilové fotografie ukazuje snímek C.15b.

6.8 Zhodnocení výsledků

První verze výsledné aplikace byla zveřejněna prostřednictvím online distribuční služby *Google Play* dne 28. dubna 2017. Pro účely propagace aplikace bylo kontaktováno prostřednictvím elektronické pošty následující:

- Liga vozíčkářů — hnutí sídlící ve městě Brno.
- ParaCENTRUM Fenix — občanské sdružení (rovněž město Brno).
- Zadavatel diplomové práce — statutární město Brno, které v průběhu řešení diplomové práce zastupoval přidělený technický konzultant.

I přes výše uvedenou propagaci bylo, po více než třech týdnech[15] veřejné dostupnosti mobilní aplikace, zaznamenáno pouze 11 stažení[16], jediné netextové hodnocení (pozitivní) a jeden registrovaný uživatel. Aplikace poskytuje (snad i trochu nabádá) zadání uživatelské textové zpětné vazby dvěma způsoby, buďto přímo v aplikaci nebo prostřednictvím distribuční služby *Google Play* (viz podkapitola 6.7.4). Ani v jednom případu nebylo zadáno žádné hodnocení. Z těchto důvodů nebyla získána žádná uživatelská zpětná vazba, krom zpětné vazby poskytnuté technickým konzultantem diplomové práce, jejíž závěry budou krátce uvedeny v podkapitole 6.8.2.

Pro účely sběru informací o případných pádech aplikace byla využita služba *Crashlytics*, která poskytuje i sběr základních statistik o uživatelské aktivitě. Výsledkem je celkově 21 sezení s průměrnou délkou přesahující 2 minuty. Z toho vyplývá, že téměř každý uživatel spustil aplikaci za období třech týdnů průměrně dvakrát. Uvádění dalších statistik nemá význam, neboť nejsou dostupná relevantní data.

[15] Text této podkapitoly psán k datu 20. května 2017.
[16] Z celkových 11 stažení připadá 9, resp. 2 na uživatele z Česka, resp. Slovenska.

Název	Verze platformy	API	Rozlišení (px)	Hustota (dpi)
LG Nexus 5X	7.1.2	25	1080x1920	420 (xxhdpi)
AVD	7.1.1	25	1080x1920	420 (xxhdpi)
AVD	7.0	24	1080x1920	420 (xxhdpi)
AVD	6.0	23	1080x1920	420 (xxhdpi)
Samsung SM-G130HN	4.4.2	19	320x480	165 (mdpi)
AVD	4.4	19	768x1280	320 (xhdpi)
HTC Desire X	4.1.1	16	480x800	233 (hdpi)

Tabulka 6.1: Použité zařízení pro účely testování

6.8.1 Testování

Pro účely testování nebylo vyžadováno vytvoření žádných automatických testů (viz podkapitola 4.2). Testování aplikace probíhalo „ruční“ formou, a to vždy po dokončení implementace většího celku, např. zobrazení budov města Brna včetně filtrace, detail budovy, správa bariér apod. Před vydáním finální verze aplikace byla tato testována na třech fyzických zařízení a několika virtuálních. Jejich seznam je uveden v tabulce 6.1, kde hodnota sloupce *AVD* značí *Android Virtual Device* (viz podkapitola 2.3.1). Uvedeným postupem bylo odhaleno několik implementačních chyb způsobující pády aplikace, které byly odstraněny. Dále byl odhalen problém vyskytující se u posledních dvou fyzických zařízení uvedených v tabulce, vlastnící displej s nižšími hodnotami rozlišení a hustoty. Problém se týká uživatelského rozhraní, konkrétně dialog aktivity *BarrierFilterDialogActivity* popsané v podkapitole 6.5.1. Tento se nezobrazuje korektně, některé textové hodnoty obsažené ve widgetu *CheckBox* se, neformálně řečeno, nevlezou na užitečnou šířku displeje. Řešení spočívá ve vytvoření optimalizované definice vzhledu *layout file* aktivity pro zařízení vlastnící displej s nižší hodnotou hustoty, tzn. *hdpi* a nižší.

Celkově byly vydány tři verze, poslední k datu 30. dubna 2017 s označením „0.2“ zpřístupňující funkcionalitu optimalizovaného vyhledávání (podkapitola 6.6.4) a opravující jediný pád aplikace zaznamenaný službou *Crashlytics*, související s neočekávaným chováním aktivity *BarrierDetailActivity* (podkapitola 6.5.3) na pomalejších zařízeních.

Závěrem, aplikace byla otestována na fyzických zařízeních s nejnovější veřejnou stabilní verzí platformy 7.1.2 a téměř nejstarší podporovanou verzí 4.1.1 (viz zmíněná tabulka 6.1), přičemž nejstarší podporovaná je 4.0.3 (API 15). Současně aplikace potenciálně podporuje téměř 9595[17] typů zařízení. Je tedy vysoce pravděpodobné, že v případě dosažení vyššího počtu stažení bude implementace aplikace obsahovat více doposud skrytých chyb projevujících se na různě upravených verzí platformy.[18]

6.8.2 Možnosti dalšího vývoje

Pro možnosti dalšího vývoje jsou již nyní připraveny obě části architektury, serverová i klientská. U první jmenované jsou dynamická data uložena v relační databázi, kterou manipuluje implementované REST API. Formát URL adresy dotazu byl popsán v podkapitole 5.3, ve které byla uvedena subdoména *dpapi* domény *kopla.cz*. Tato slouží pro obsluhu dotazů zasílaných ostrou verzí klientské aplikace, tzn. aplikace dostupné veřejnosti prostřednictvím *Google Play*.

[17] Údaj poskytnutý prostřednictvím nástroje *Google Play Console*.
[18] Výrobci mobilních zařízení mohou (alespoň prozatím) libovolně platformu upravovat.

Během probíhajícího vývoje aplikace ostrá verze databáze není vhodná. Proto, krom výše zmíněné subdomény *dpapi*, existuje další subdoména *dpapidev*, ve které je umístěna kopie implementovaného REST API obsluhujícího odlišnou databázi. Díky této skutečnosti lze provádět změny v implementaci API, schématu a daty relační databáze, a to bez ovlivnění dat a funkcionality veřejné verze aplikace.

Klientská část na výše popsané musí patřičně reagovat. Proto bylo využito nabízené funkcionality tzv. *Product Flavors*[19], pomocí kterých lze, zjednodušeně řečeno, přizpůsobit proces a výsledek překladu odlišných verzí aplikace založené na jediném projektu, resp. modulu (projekt a modul stručně popsán v podkapitole 2.3.3). Konkrétně byly specifikovány dvě varianty překladu:

- *flavDev* — definuje hodnoty konstant přizpůsobující běh aplikace v režimu vývoje. Jedná se zejména o konstanty zakazující službu *Crashlytics*, dále definici API klíčů pro využité služby poskytované společností Google (*Google Places*, *Google Directions*, *Google Maps*). Konečně definice URL adresy REST API a specifikaci jedinečného identifikátoru aplikace *package name*.

- *flavRelease* — platí výše uvedené, avšak pro běh aplikace v ostrém (veřejném) režimu.

Zásluhou využití *Product Flavors* jsou odstraněna rizika vzniku chyb během překladu veřejné verze aplikace, které by mohly vzniknout bez využití zmíněného (opomenutá změna některé konstanty). Velkou výhodou je pak možnost definice *package name* pro rozdílené verze aplikace, neboť touto skutečností je umožněno instalování rozdílných verzí téže aplikace na jednom zařízení.

Rozšíření nabízené funkcionality a logiky aplikace

Dále bude uveden seznam možných rozšíření funkcionality přesahující rámec požadavků v minulosti specifikovaných na výslednou aplikaci a možné modifikace implementace a architektury. Výčet následuje:

- Vyhledávání budov — nyní lze budovy zobrazit na mapovém podkladu, s možností jejich filtrace. Tyto jsou po první inicializaci aplikace uloženy lokálně v *SQLite* databázi. Navíc jsou vždy přístupné prostřednictvím objektu *AppResources* (diskutováno v podkapitole 6.3.2). Samotný proces vyhledávání (např. dle názvu, adresy) by neměl být implementačně náročný.

- Fulltextové vyhledávání POI — umožnění vyhledávat místa zájmu za pomoci služby *Google Places* v bezprostředním okolí dané aktuálním umístění zařízení. Pro účely získávání POI je dostupný fragment *GooglePlacesLoaderFragment* (podkapitola 6.4.3). Zbývá tedy implementovat patřičné uživatelské rozhraní umožňující navrhovanou funkcionalitu.

- Práce s fotografiemi — nyní je umožněno pouze vytváření nové fotografie pro účely přiřazení k uživateli nebo bariéře. Vhodným rozšířením by mohlo být umožnit výběr fotografie z lokálního úložiště zařízení, následované možností provádění jednoduchých úprav (ořezání, přetočení).

[19] Dokumentace – https://developer.android.com/studio/build/build-variants.html [Online; navštíveno 19.05.2017]

- Registrace a přihlášení — současný stav se může jevit dosti zdlouhavý. Implementace registrace a přihlášení za využití služeb třetích stran by celý proces urychlilo. Například sociální sítě *Facebook*, *Google Plus* a *Twitter* tuto funkcionalitu nabízejí prostřednictvím poskytovaného API. Uvedené řešení by mohlo vyřešit současnou neochotu registrace uživatelů a zlepšit poměr mezi počtem stažení aplikace a provedenými registracemi – nyní 11 : 1.

- Komunikace HTTPS protokolem — nyní je použit protokol HTTP pro komunikaci s vlastním REST API, který neřeší otázku bezpečnosti.

- Manipulace vybraných služeb — obsluha *Google Places* a *Google Directions* de facto patří na serverovou stranu, nikoli klientskou. Přesunem by se redukoval problém limitů služeb, které je nutno dodržet pro bezplatné využívání. Získaná data by se uchovávala na straně serveru s možností sdílení mezi klienty. Zároveň by bylo odstraněno nebezpečí možného vyzrazení (v současné době z logiky věci) nechráněných API (spíše *Server*) klíčů za prostředků reverzního inženýrství aplikovaných na publikovaný soubor *.apk*.

- Zavedení role správce — aktuálně jedinou možností globální správy dat (např. mazání nevhodných komentářů) je prostřednictvím přímého přístupu k záznamům databáze (*phpMyAdmin*). Zavedení role správce a zpřístupnění patřičných funkcionalit v rámci aplikace je z uživatelského hlediska samozřejmě vhodnější.

- Využití dat aplikace *VozejkMap* — zmíněná byla popsána podkapitole 3.2.1. Autoři poskytují data pro nekomerční účely třetí straně. Výsledná aplikace uvedené splňuje. Tato by pak mohla zobrazovat objekty nacházející se na katastrálním území města Brna.

- Uživatelské přidávání budov — umožnit registrovaným uživatelům přidat nové záznamy reprezentující budovy města Brna a tím rozšiřovat databázi.

Rozšíření specifikované dvěma posledními body stojí za úvahu. V případě jejich implementace by byl porušen část konceptu aplikace. Ta si klade za cíl poskytovat informace o vybraných budovách města Brna v co možná nejlepší kvalitě a přesnosti. V detailu budovy (podkapitola 6.4.3) jsou poskytnuty velmi detailní informace o přístupnosti, jako šířky vstupních dveří, sklony povrchu příjezdových cest a mnoho další. Tyto informace nejsou schopni samotní uživatelé aplikace zadávat.

Technický konzultant poskytl zpětnou vazbu obsahující následující rozšíření:

- Značení průjezdných tras — uživateli by bylo, kromě zadávání překážek, umožněno označovat části komunikace jako bezbariérové. Se současným využitím *Google Map* uvedené není možné.

- Zobrazení budov a bariér v rámci jedné mapy — v současné době je tato funkcionalita oddělena. Splnění zmíněného požadavku je možné.

Kapitola 7

Závěr

Hlavním cílem diplomové práce bylo vytvořit mobilní aplikaci pro platformu Android, která je určena pro úzkou skupinu uživatelů, tvořenou osobami s různými úrovněmi tělesného postižení, a to zejména se zaměřením na osoby využívající invalidní vozík.

Samotné implementaci předcházelo seznámení se s platformou Android z hlediska tvorby mobilních aplikací, včetně oficiálního vývojového prostředí, jenž je v současné době Android Studio. Této problematice se věnovala teoretická část diplomové práce, přičemž byl kladen důraz na uvedení a vysvětlení základních principů a komponent platformy Android, následně využité v průběhu řešení programové části diplomové práce.

Dále byl proveden průzkum stávajících mobilních a webových aplikací. Jako hlavní kritérium pro výběr reprezentantů byla zvolena oblast působnosti. Z řad nadnárodních byla vybrána aplikace *Wheelmap*, národních pak *VozejkMap*, která se zaměřuje na uživatele z celé České republiky. Reprezentantem aplikace s lokální působností byla vybrána *Bezbariérová 6* zaměřená na území městské části Praha 6.

Pro účely návrhu byla sestavena obsáhlejší neformální specifikace požadavků, s pomocí které byly nalezeni aktéři a detaily případů použití. Tyto byly popsány stručnou formou. Následoval návrh uživatelského rozhraní pro několik konkrétních obrazovek výsledné mobilní aplikace, avšak obsahující většinu komponent následně využitých v ostatních obrazovkách uživatelského rozhraní.

Následně je popsána implementace serverové části, která obsluhuje statická a dynamická data. Prvními jsou atributy definující vybrané budovy města Brna, u kterých není předpoklad časté aktualizace. Dynamickými daty rozumíme ostatní aplikační data, jako informace o uživatelích, bariérách, komentářích a hodnocení, mající charakter časté změny. Dynamická data jsou uchovávána v relační databázi a manipulována implementovaným REST API, poskytující sadu 26 služeb, ze kterých byl vybrán jeden zástupce určený k popisu principu implementace celé sady.

Velkou část diplomové práce tvoří popis implementace klientské části, kterou je mobilní aplikace pro platformu Android. Z počtu nalezených případů použití a rozsahu programové části plyne obsáhlost patřičné kapitoly, která nejprve popisuje celkovou strukturu implementace se zaměřením na popis hlavních tříd a balíčků. Nechybí vysvětlení provedení požadavku na obecný zdroj, který je téměř schodný s požadavkem na implementované REST API. Dále je uveden důvod nutnosti provádění inicializačních úkonů během startu aplikace. Následuje detailnější popis implementace správy vybraných budov města Brna a manipulace s bariérami. Detailněji je popsán princip nalezení bariérových a potenciálně bezbariérových tras, společně s uvedením postupu optimalizace bariérové trasy. Ostatní funkcionalita aplikace byla popsána zkrácenou formou.

Bylo uvedeno celkově 11 možných rozšíření mobilní aplikace, např. umožnění vyhledávání v budovách pomocí jejich názvu nebo adresy a urychlení procesu přihlášení za pomoci prostředků poskytnutých třetí stranou. Celkově však implementovaná mobilní aplikace splňuje požadavky nalezené analýzou požadavků, potažmo zadáním diplomové práce.

Mobilní aplikace byla zveřejněna v českém a anglickém jazyce prostřednictvím online distribuční služby Google Play. V současné době zaznamenala pouze 11 stažení, a to i přes provedené kontaktování dvou organizací se zaměřením na pomoc osobám s tělesným postižením. Současně byl uvědoměn zadavatel diplomové práce, který je zastupován prostřednictvím technického konzultanta.

Závěrem, osoby s tělesným postižením získávají nový nástroj umožňující plánování potenciálně bezbariérových tras a zobrazení přesných informací o vybraných budovách, obojí týkající se města Brna. Z pohledu konkurenčních řešení žádné nenabízí funkcionalitu plánování bezbariérových tras. V tomto ohledu výsledná aplikace vyniká.

Literatura

[1] Allen, G.: *Beginning Android, Fifth Edition*. Apress, 2015, ISBN 978-1-4302-4687-9.

[2] D. Richard Hipp: *About SQLite*. [Online; navštíveno 14.03.2017].
URL https://www.sqlite.org/about.html

[3] D. Richard Hipp: *SQL As Understood By SQLite*. [Online; navštíveno 04.03.2017].
URL http://sqlite.org/lang_altertable.html

[4] Deitel, P.; Deitel, H.; Wald, A.: *Android 6 for Programmers: An App-Driven Approach*. Pearson Education, Inc., 2015, ISBN 0-13-428936-6.

[5] Google inc.: *Activity*. [Online; navštíveno 02.01.2017].
URL https://developer.android.com/reference/android/app/Activity.html

[6] Google inc.: *ADT Plugin (DEPRECATED)*. [Online; navštíveno 03.01.2017].
URL https://developer.android.com/studio/tools/sdk/eclipse-adt.html

[7] Google inc.: *Android Developers Blog - An update on Eclipse Android Developer Tools*. [Online; navštíveno 03.01.2017].
URL https://android-developers.googleblog.com/2015/06/an-update-on-eclipse-android-developer.html

[8] Google inc.: *Android Platform Reference - Services*. [Online; navštíveno 02.05.2017].
URL https://developer.android.com/reference/android/app/Service.html

[9] Google inc.: *App Install Location*. [Online; navštíveno 02.05.2017].
URL
https://developer.android.com/guide/topics/data/install-location.html

[10] Google inc.: *Application*. [Online; navštíveno 03.01.2017].
URL https://developer.android.com/reference/android/app/Application.html

[11] Google inc.: *App Manifest*. [Online; navštíveno 05.01.2017].
URL
https://developer.android.com/guide/topics/manifest/manifest-intro.html

[12] Google inc.: *App Manifest - uses-sdk*. [Online; navštíveno 05.01.2017].
URL https:
//developer.android.com/guide/topics/manifest/uses-sdk-element.html

[13] Google inc.: *AsyncTask*. [Online; navštíveno 02.01.2017].
URL https://developer.android.com/reference/android/os/AsyncTask.html

[14] Google inc.: *Dashboards*. [Online; naposledy navštíveno 01.05.2017].
URL https://developer.android.com/about/dashboards/index.html#Platform

[15] Google inc.: *Fragments*. [Online; navštíveno 02.01.2017].
URL https://developer.android.com/guide/components/fragments.html

[16] Google inc.: *Getting Started with the NDK*. [Online; navštíveno 04.01.2017].
URL https://developer.android.com/ndk/guides/index.html

[17] Google inc.: *Hardware Acceleration*. [Online; navštíveno 02.05.2017].
URL
https://developer.android.com/guide/topics/graphics/hardware-accel.html

[18] Google inc.: *Intents and Intent Filters*. [Online; navštíveno 03.05.2017].
URL https://developer.android.com/guide/components/intents-filters.html

[19] Google inc.: *Parcelable*. [Online; navštíveno 03.05.2017].
URL https://developer.android.com/reference/android/os/Parcelable.html

[20] Google inc.: *Processes and Threads*. [Online; navštíveno 03.01.2017].
URL https:
//developer.android.com/guide/components/processes-and-threads.html

[21] Google inc.: *Projects Overview*. [Online; navštíveno 05.01.2017].
URL https://developer.android.com/studio/projects/index.html

[22] Google inc.: *Requesting Permissions*. [Online; navštíveno 03.05.2017].
URL
https://developer.android.com/guide/topics/permissions/requesting.html

[23] Google inc.: *Saving Data in SQL Databases*. [Online; navštíveno 19.03.2017].
URL https:
//developer.android.com/training/basics/data-storage/databases.html

[24] Google inc.: *Saving Key-Value Sets*. [Online; navštíveno 04.05.2017].
URL https://developer.android.com/training/basics/data-storage/shared-preferences.html

[25] Google inc.: *Services*. [Online; navštíveno 06.01.2017].
URL https://developer.android.com/guide/components/services.html

[26] Google inc.: *SharedPreferences.Editor*. [Online; navštíveno 03.05.2017].
URL https://developer.android.com/reference/android/content/SharedPreferences.Editor.html

[27] Google inc.: *SQLiteOpenHelper*. [Online; navštíveno 19.03.2017].
URL https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html

[28] Google inc.: *Supporting Multiple Screens*. [Online; navštíveno 01.01.2017].
URL https://developer.android.com/guide/practices/screens_support.html

[29] Google inc.: *Version Your App*. [Online; navštíveno 05.01.2017].
URL https://developer.android.com/studio/publish/versioning.html

[30] Google inc.: *View*. [Online; navštíveno 02.01.2017].
URL https://developer.android.com/reference/android/view/View.html

[31] Horton, J.: *Android Programming for Beginners*. Packt Publishing Ltd., 2015, ISBN 978-1-78588-326-2.

[32] Luboslav Lacko: *Vývoj aplikací pro Android*. Computer Press, 2015, ISBN 978-80-251-4347-6.

[33] Lombardo, J.; aj.: *Android Application Development*. O'Reilly Media, Inc., 2009, ISBN 978-0-596-52147-9.

[34] Meier, R.: *Professional Android 2 Applicaton Development*. Wiley Publishing, Inc., 2010, ISBN 978-0-470-56552-0.

[35] Phillips, B.; Steward, C.; Marsicano, K.: *Android Programming: The Big Nerd Ranch Guide*. Big Nerd Ranch, LLC., 2017, ISBN 0-13-470607-2.

[36] Vávrů, J.; Ujbányai, M.: *Programujeme pro Android*. Grada Publishing, a.s., 2013, ISBN 978-80-247-4863-4.

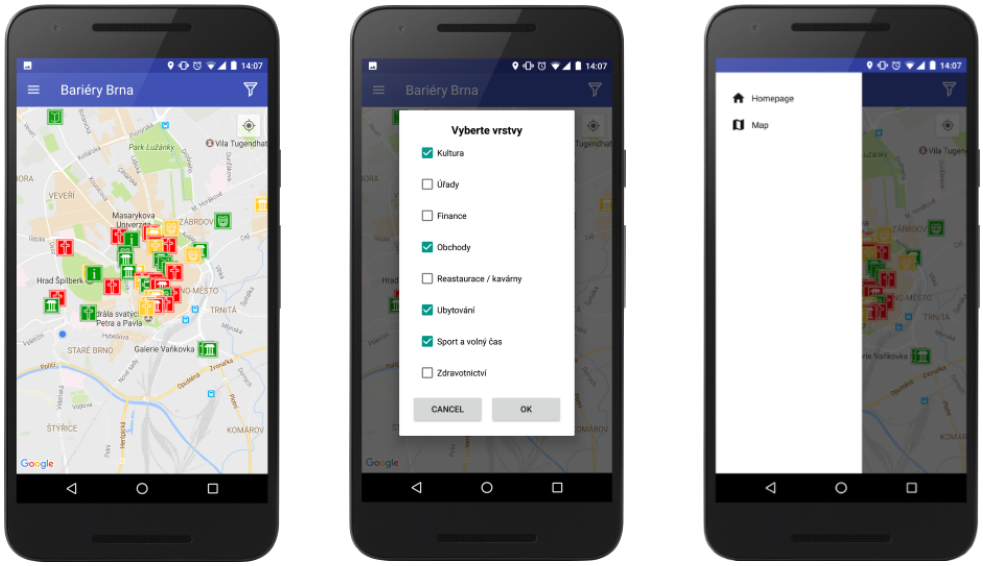
[37] Wilson, J.: *Creating Dynamic UIs with Android Fragments, 2nd Edition*. Packt Publishing Ltd., 2016, ISBN 978-1-78588-959-2.

Přílohy

Příloha A

Ukázky vzhledu první verze aplikace

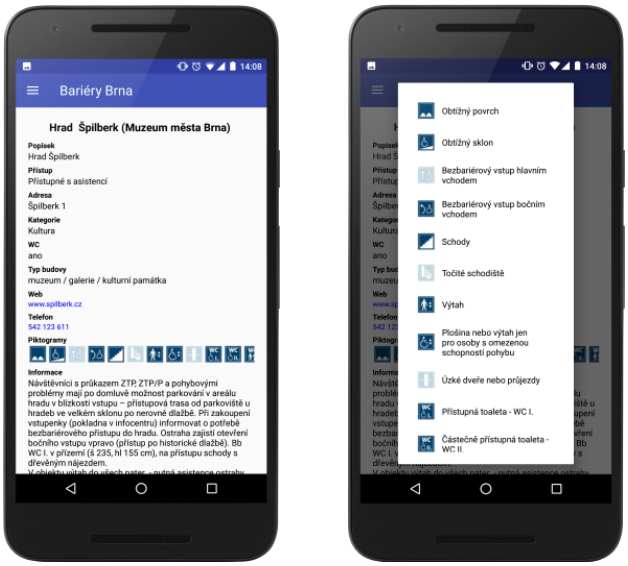
Následující obrázky znázorňují grafické uživatelské rozhraní první verze aplikace. Obrázek A.1a zobrazuje mapový podklad s ikonami reprezentujícími budovy z vrstvy Kultura. Již nyní je patrná nepřehlednost mapového díla při zobrazení budov pouze z jedné vrstvy dané množstvím vykreslených grafických elementů. Obrázek A.1b zachycuje původní vzhled filtru budov. Ukázka počátečního obsahu menu je patrná z obrázku A.1c.



(a) Zobrazení budov (b) Výběr vrstev (c) Menu aplikace

Obrázek A.1: Snímky první verze aplikace – 1.

Snímek A.2a zobrazuje uživatelské rozhraní původního detailu budovy. Výpis piktogramů společně s jejich popisem pomocí dialogového okna zachyceném na obrázku A.2b následuje. Jedná se o jedinou obrazovku téměř nezměněnou, oproti stávajícímu stavu pouze chybí akční tlačítko pro uzavření dialogu.



(a) Detail objektu (b) Piktogramy

Obrázek A.2: Snímky první verze aplikace – 2.

Příloha B

Služby REST API serverové části

Služba	Hlavičky	Popis služby
GetById/user_id	-	Vrací uživatele dle ID
GetAll	-	Vrací všechny uživatele
Logout	User-Token	Odhlašuje uživatele dle tokenu
RefreshToken	User-Token	Vrací uživatele dle tokenu s aktualizovaným tokenem

Tabulka B.1: Služby třídy *UserController* - metoda GET

Služba	Hlavičky	Form-data	Popis služby
Create	-	firstname, lastname, nickname, password, email, birthday	Vytvoření uživatele
Login	-	email, password	Přihlášení uživatele
Photo	User-Token	file	Změna profilové fotografie

Tabulka B.2: Služby třídy *UserController* - metoda POST

Služba	Hlavičky	Služba vrací
GetById/comment_id	-	Komentář dle ID
GetByBuilding/building_id	-	Všechny komentáře k budově
GetLatest?Limit=limit	-	Maximálně *limit* nejnovějších komentářů
GetLatestByUser?Limit=limit	User-Token	Maximálně *limit* nejnovějších komentářů aktuálního uživatele
GetByUserId/user_id	-	Všechny komentáře k budovám dle uživatele *user_id*

Tabulka B.3: Služby třídy *BuildingCommentController* - metoda GET

Služba	Hlavičky	Form-data	Popis služby
Create	User-Token	building_id, text	Vytvoření komentáře budovy
Update/comment_id	User-Token	text	Úprava komentáře budovy

Tabulka B.4: Služby třídy *BuildingCommentController* - metoda POST

Služba	Hlavičky	Služba vrací
GetById/barrier_id	-	Bariéru dle ID
GetAll	-	Všechny bariéry
GetByUserId/user_id	-	Všechny bariéry zadané uživatelem *user_id*
GetLatest?Limit=limit	-	Maximálně *limit* nejnovějších bariér
GetLatestByUser?Limit=limit	User-Token	Maximálně *limit* nejnovějších bariér zadaných uživatelem

Tabulka B.5: Služby třídy *BarrierController* - metoda GET

Služba	Hlavičky	Form-data	Popis služby
Create	User-Token	is_temporal, valid_until, category, subcategory, overcome, latitude, longitude, note, file (optional)	Vytvoření bariéry
Update/ barrier_id	User-Token	is_temporal, valid_until, category, subcategory, overcome, latitude, longitude, note, file (optional)	Úprava bariéry
RatingCreate/ barrier_id	User-Token	rating	Vytvoření hodnocení
RatingUpdate/ barrier_rating_id	User-Token	rating	Úprava hodnocení
CommentCreate/ barrier_id	User-Token	comment_text	Vytvoření komentáře
CommentUpdate/ barrier_comment_id	User-Token	comment_text	Úprava komentáře

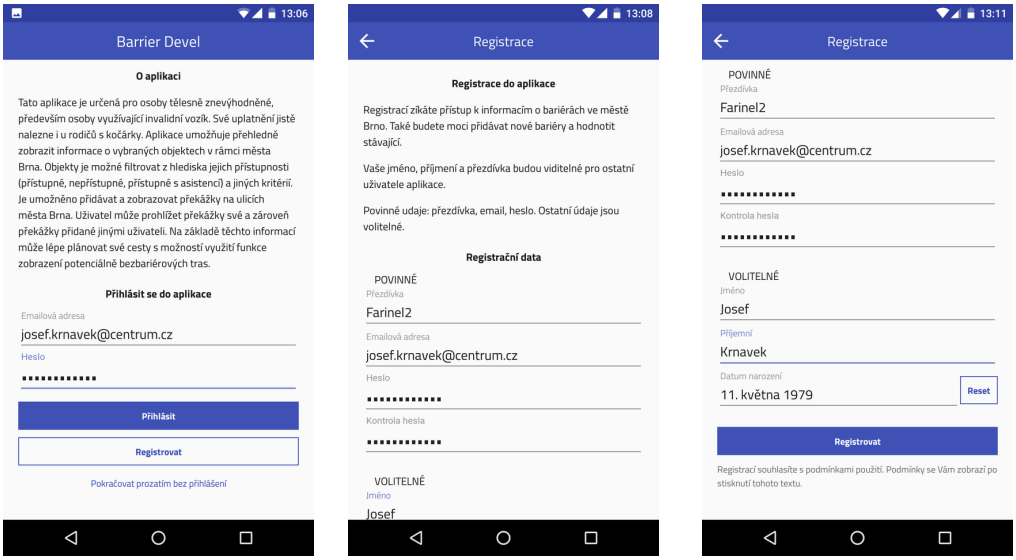
Tabulka B.6: Služby třídy *BarrierController* - metoda POST

Služba	Hlavičky	Form-data	Popis služby
Create	User-Token	text	Vytvoření zpětné vazby

Tabulka B.7: Služby třídy *FeedbackController* - metoda POST

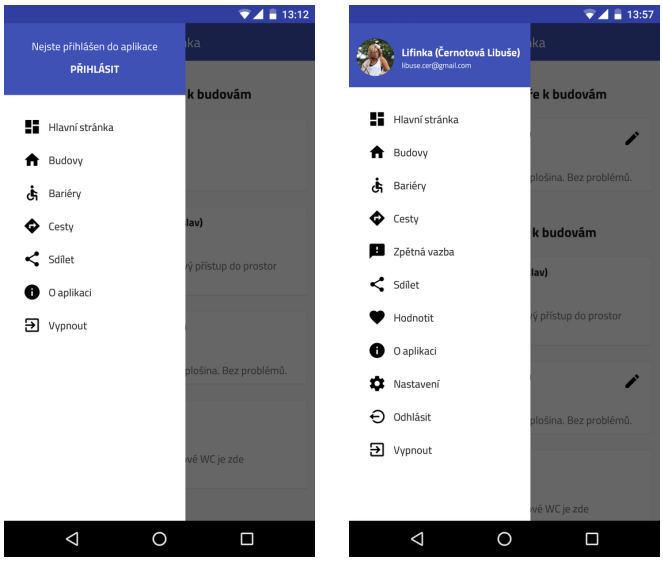
Příloha C

Ukázky vzhledu finální aplikace



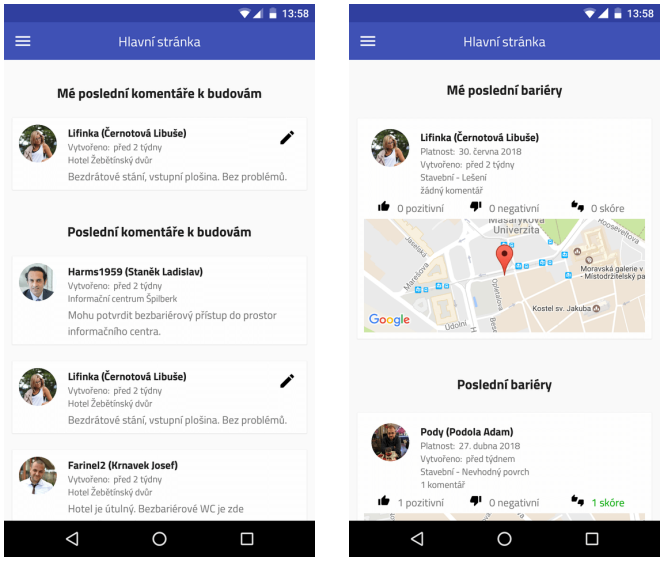
(a) Přihlášení (b) Registrace (c) Registrace

Obrázek C.1: Přihlášení a registrace



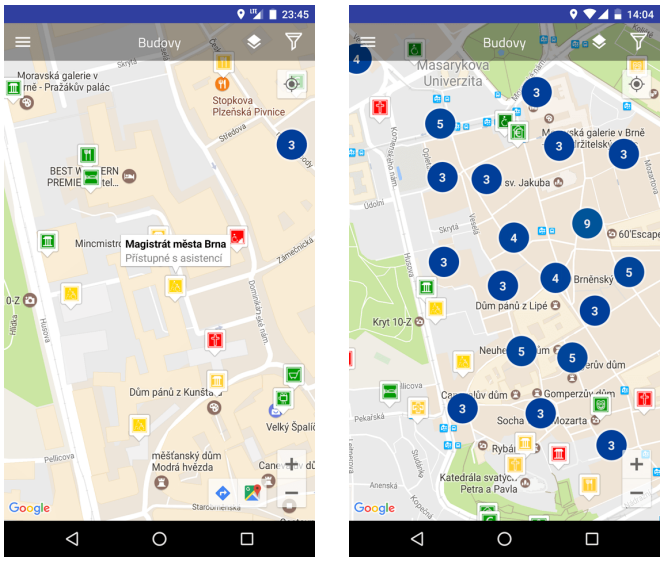
(a) Nepřihlášený (b) Přihlášený

Obrázek C.2: Menu aplikace dle uživatele



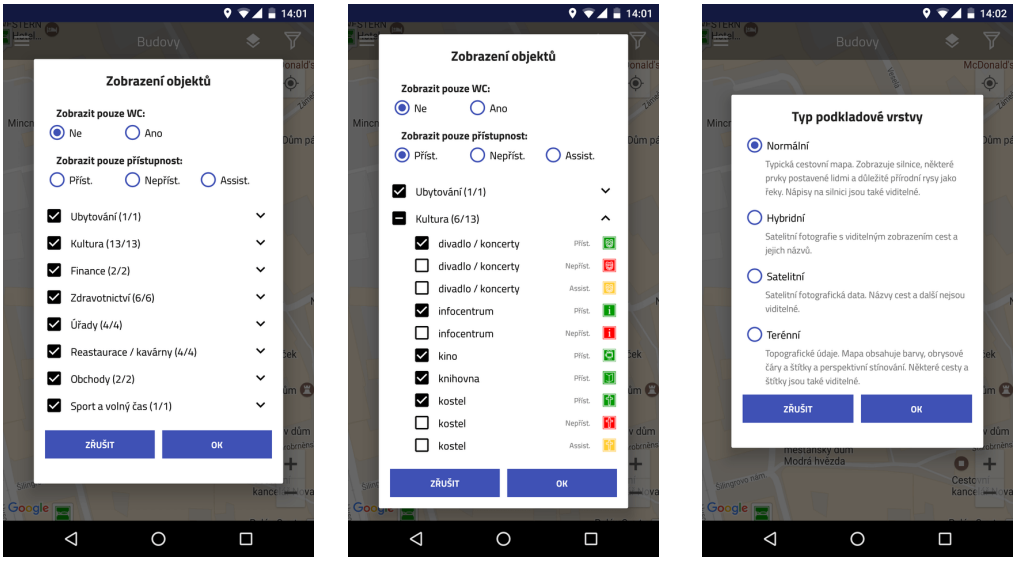
(a) Nejnovější komentáře (b) Nejnovější bariéry

Obrázek C.3: Hlavní stránka



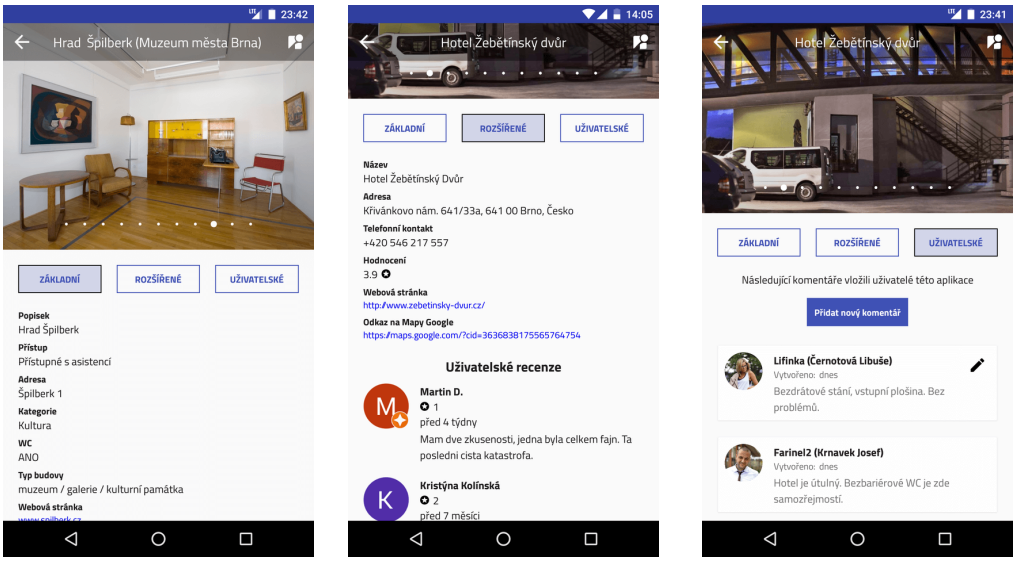
(a) Vyšší úroveň přiblížení (b) Nižší úroveň přiblížení

Obrázek C.4: Zobrazení budov na mapovém podkladu



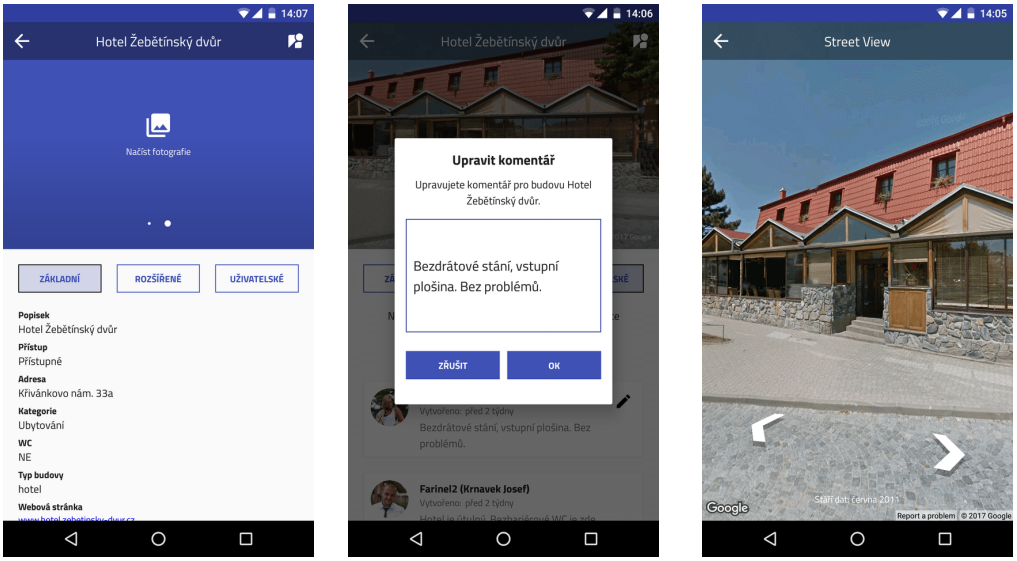
(a) Filtr kompaktní (b) Filtr detailní (c) Změna podkladu mapy

Obrázek C.5: Filtrování budov a změna podkladové mapy



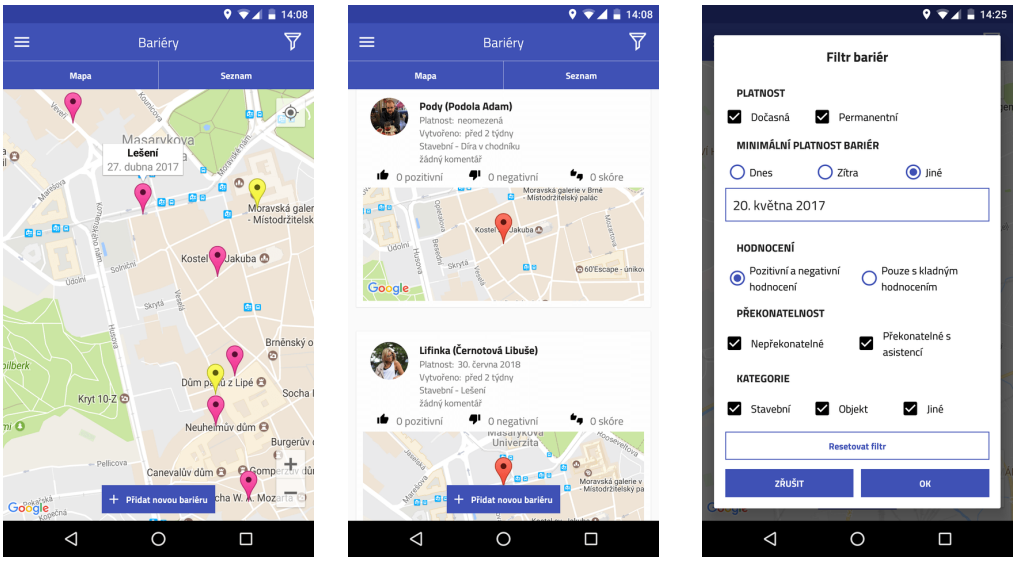
(a) Základní (b) Rozšířené (c) Uživatelské recenze

Obrázek C.6: Detail budovy s různými typy informací



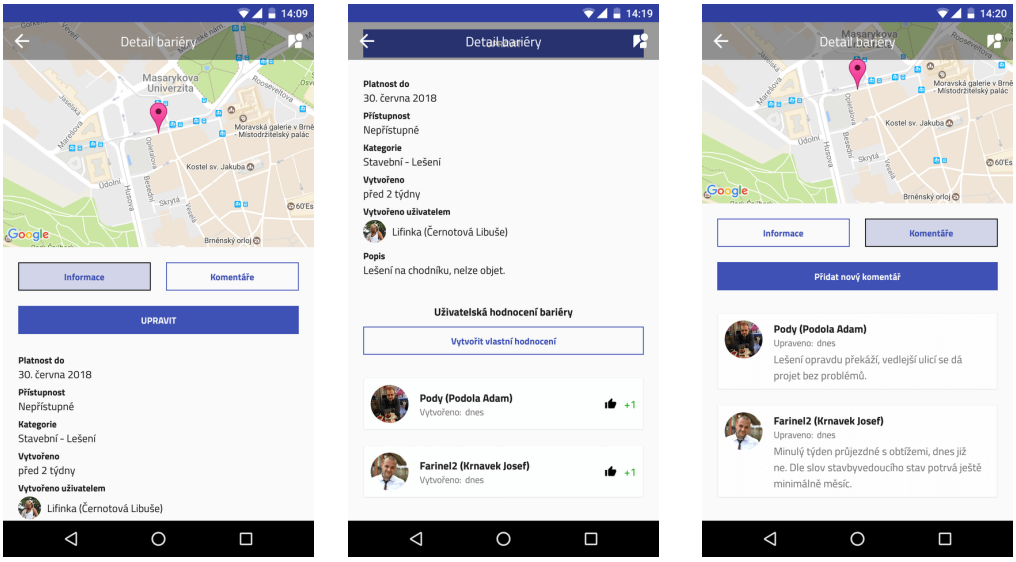
(a) Načtení fotografií (b) Manipulace s komentáři (c) Street View

Obrázek C.7: Detail budovy



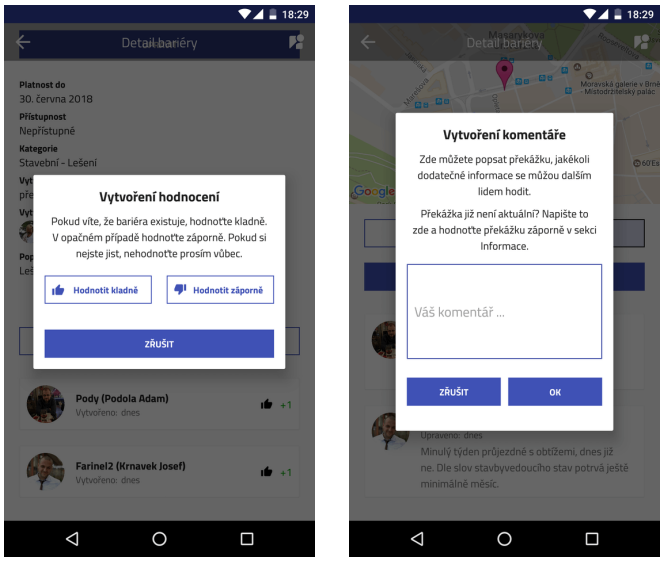
(a) Mapový podklad (b) Seznam (c) Filtrování

Obrázek C.8: Zobrazení bariér s možností filtrace



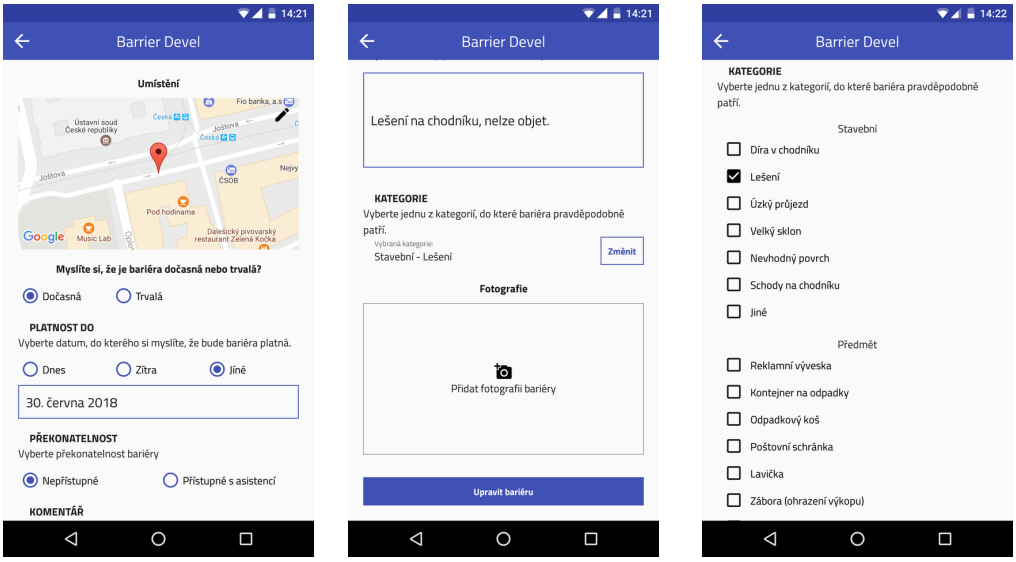
(a) Základní informace (b) Uživatelské hodnocení (c) Uživatelské komentáře

Obrázek C.9: Detail bariéry



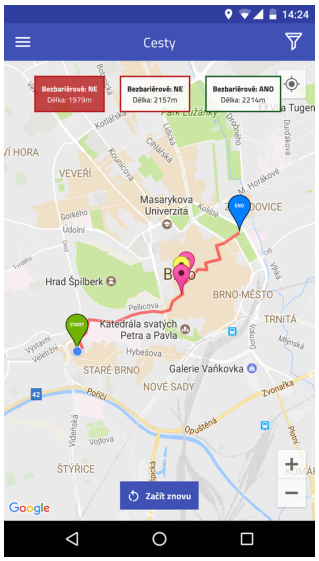
(a) Vytvoření hodnocení (b) Vytvoření komentáře

Obrázek C.10: Manipulace s hodnocením a komentářem bariéry

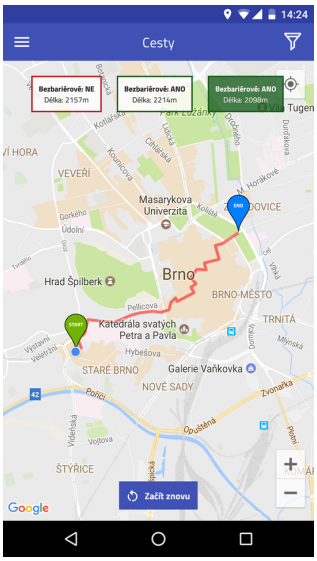


(a) Základní informace (b) Možnost přidání fotografie (c) Výběr kategorie

Obrázek C.11: Vytváření nové bariéry

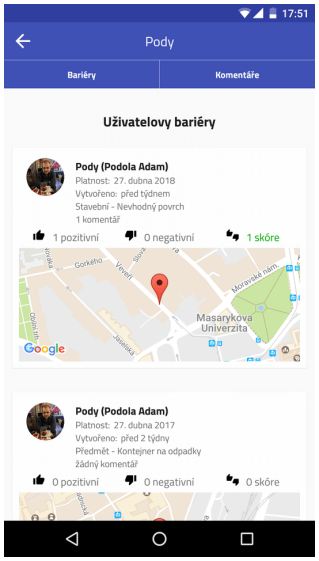


(a) Bariérová trasa

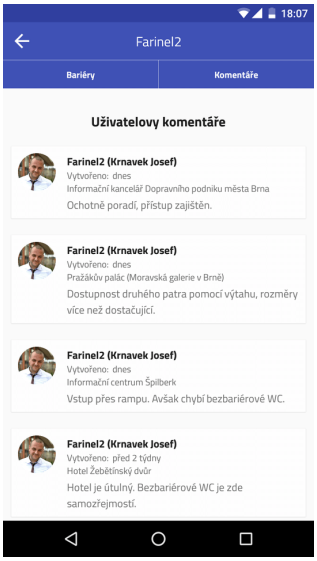


(b) Bezbariérová trasa

Obrázek C.12: Zobrazení výsledku plánování trasy

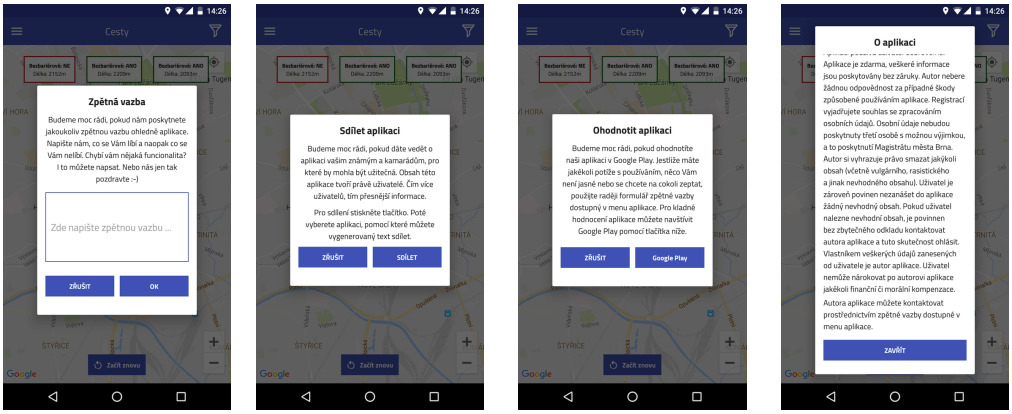


(a) Uživatelovy bariéry



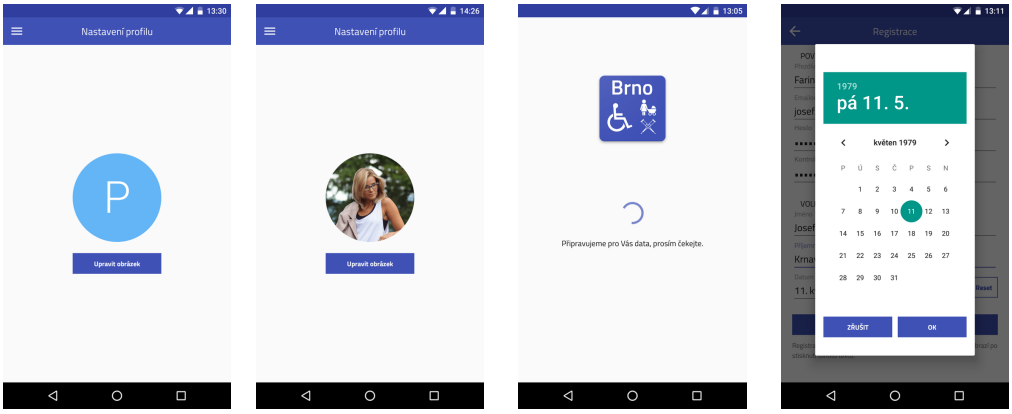
(b) Uživatelovy komentáře

Obrázek C.13: Detail profilu uživatele



(a) Zpětná vazba (b) Sdílet aplikaci (c) Hodnoť aplikaci (d) O aplikaci

Obrázek C.14: Dialogová okna s možností spuštění prostřednictvím menu aplikace

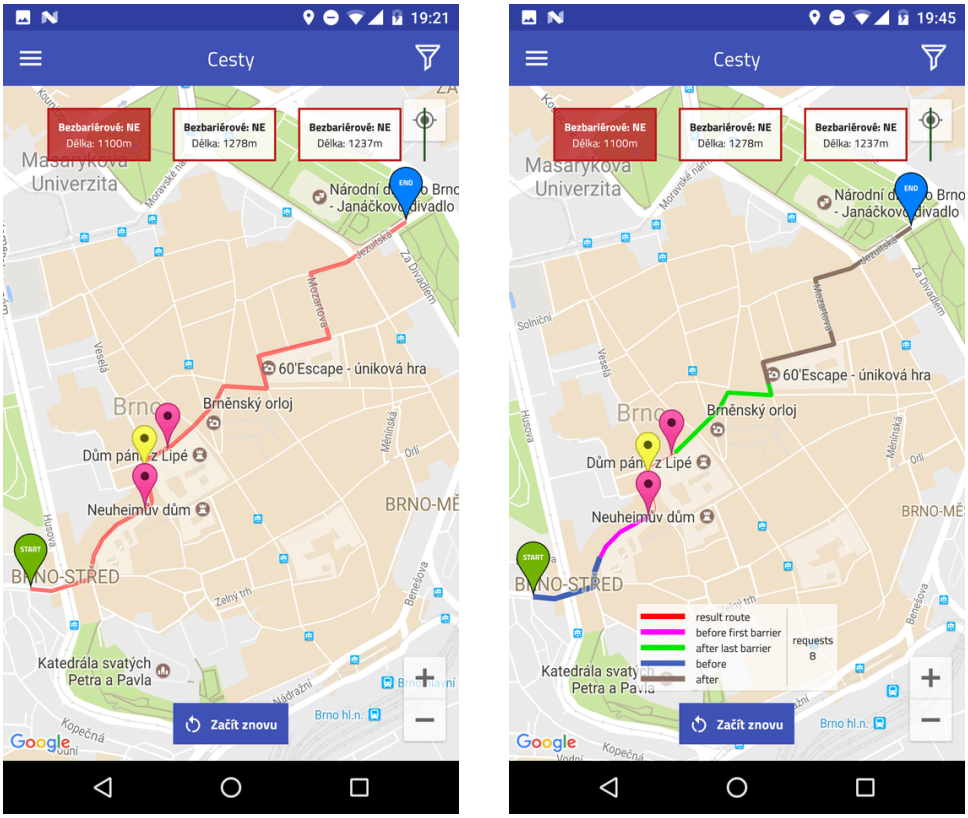


(a) Změna foto 1 (b) Změna foto 2 (c) Průběh startu (d) Výběr data

Obrázek C.15: Změna fotografie uživatele, průběh startu aplikace a výběr data.

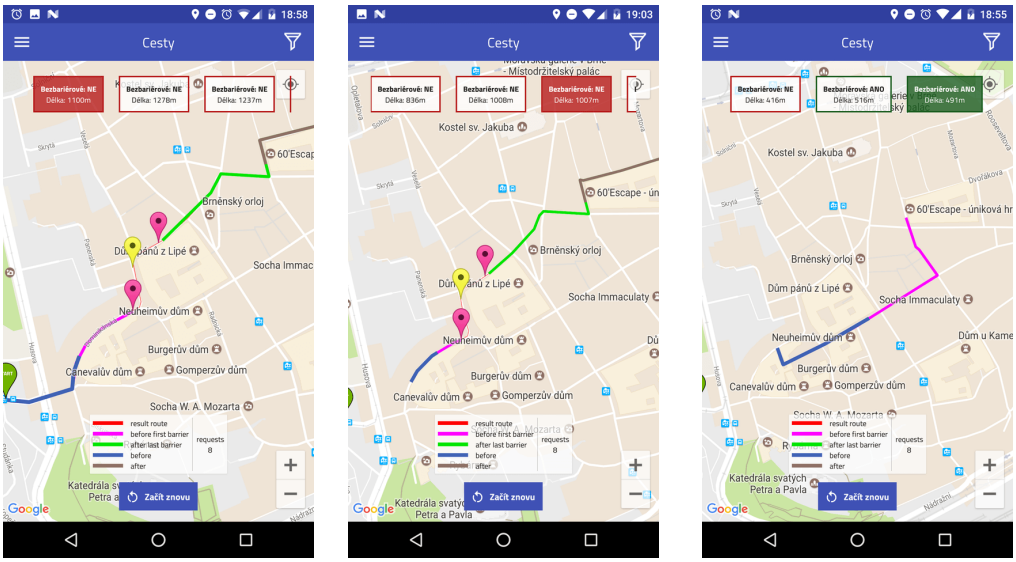
Příloha D

Podpůrné snímky hledání cesty



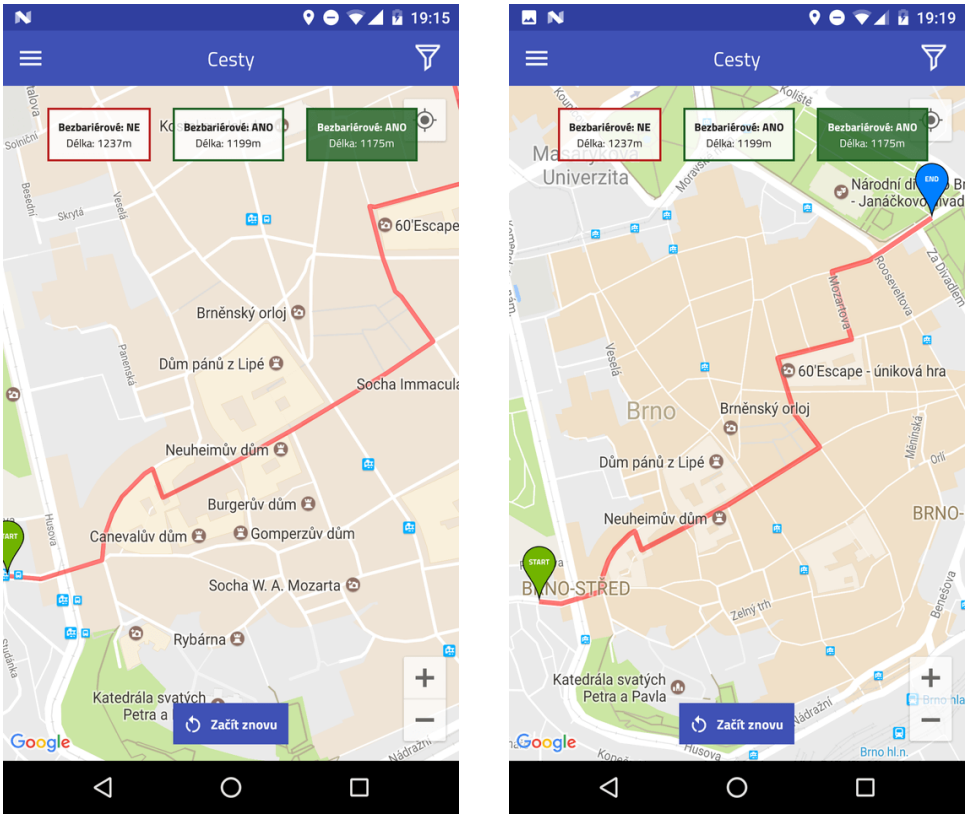
(a) Původní trasa (b) Rozdělená trasa

Obrázek D.1: Rozdělení bariérové trasy na úseky



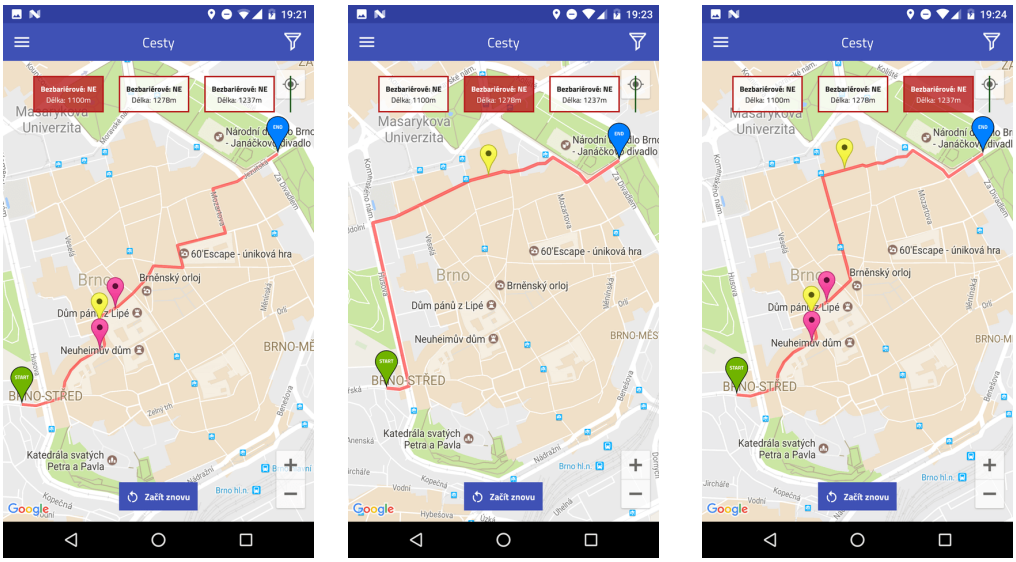
(a) Hloubka 0 (b) Hloubka 1 (c) Hloubka 2

Obrázek D.2: Trasy *Google Direction* s různou hloubkou zanoření



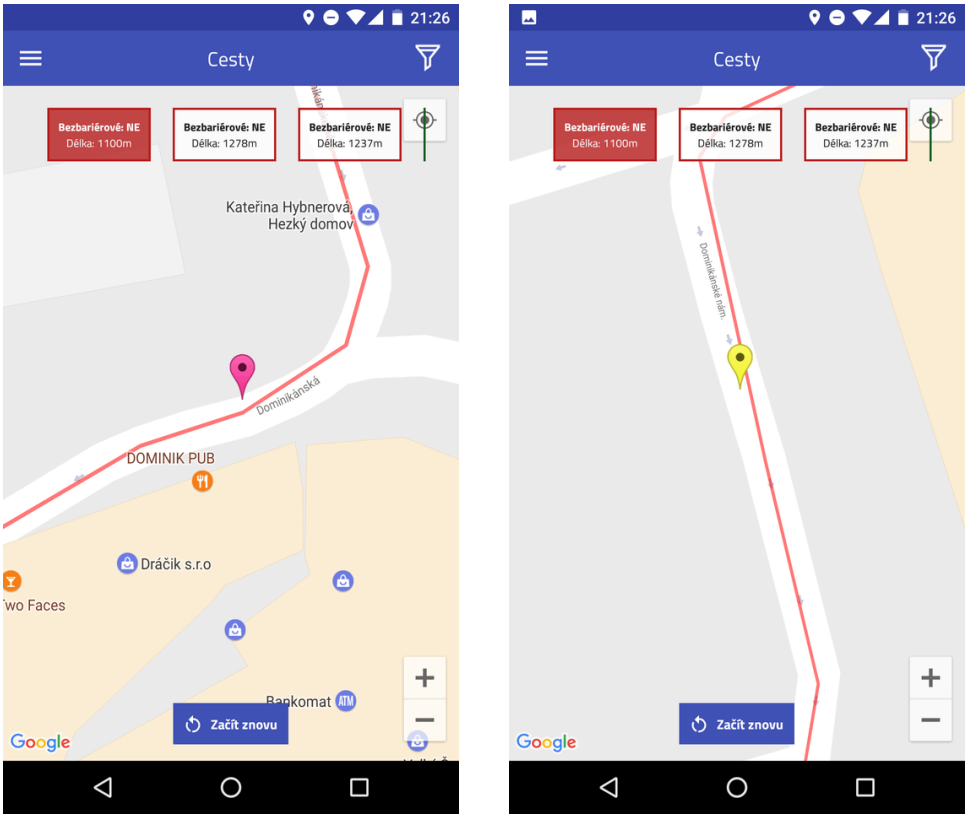
(a) Detail optimalizované části trasy (b) Zobrazení celé trasy

Obrázek D.3: Optimalizovaná trasa



(a) Trasa 1 (b) Trasa 2 (c) Trasa 3

Obrázek D.4: Původní bariérové trasy *Google Direction*



(a) Příklad 1 (b) Příklad 2

Obrázek D.5: Problém náležitosti bariéry k úseku trasy

Příloha E

Knihovny a služby třetích stran

Následuje výčet vybraných knihoven, nástrojů a webových služeb využitých pro implementaci klientské části. Většina byla zmíněna v kapitole 6, proto dále bez bližšího vysvětlení.

Knihovny a nástroje

- Google Play Services — maps, location
  https://developers.google.com/android/guides/setup
- Google Maps Android API utility library
  https://github.com/googlemaps/android-maps-utils
- Volley
  https://github.com/google/volley
- OkHttp
  http://square.github.io/okhttp
- Picasso
  http://square.github.io/picasso
- Jackson — core, annotation, databind
  https://github.com/FasterXML
- CTS
  https://github.com/orbisgis/cts
- Gson
  https://github.com/google/gson
- ORMLite — core, Android
  http://ormlite.com
- TextDrawable
  https://github.com/amulyakhare/TextDrawable
- Crashlytics
  https://fabric.io/kits/android/crashlytics
- Street View
  https://developers.google.com/maps/documentation/android-api/streetview

Webové služby

- Google Places API Web Services
  https://developers.google.com/places/web-service
- Google Maps Directions API Web Services
  https://developers.google.com/maps/documentation/directions/

Příloha F

Obsah přiloženého paměťového média

Přiložené CD paměťové médium obsahuje následující komprimované soubory:

- 01_latex — zdrojové soubory diplomové práce pro LaTeX.
- 02_android — zdrojové soubory mobilní aplikace.
- 03_android_doc — generovaná dokumentace ke zdrojovým kódům mobilní aplikace.
- 04_backend_static — PHP skript obsluhující statická data.
- 05_backend_api — zdrojové soubory REST API obsluhující dynamická data.
- 06_sql_init — SQL soubor pro inicializaci relační databáze obsluhovanou REST API.