VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

MOBILNÍ ASISTENT PRO CESTOVÁNÍ MHD

THESIS TITLE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JAN TŮMA**
AUTHOR

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

BRNO 2017

Zadání diplomové práce/19847/2016/xtumaj02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Tůma Jan, Bc.**
Obor: Informační systémy
Téma: **Mobilní asistent pro cestování MHD**
Mobile Public Transportation Assistant
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s dostupnými službami IDS JMK pro zjišťování polohy zastávek a dopravních prostředků.
2. Prostudujte existující technologie pro tvorbu aplikací klient-server s mobilním klientem na platformě Android.
3. Navrhněte prototyp aplikace pro plánování trasy a přestupů v síti MHD v Brně s využitím dat o aktuální poloze z zpoždění vozidel.
4. Po dohodě s vedoucím implementujte serverovou část pomocí vhodných technologií a klientskou část na platformě Android.
5. Proveďte testování vytvořeného řešení v reálném prostředí.
6. Zhodnoťte dosažené výsledky a navrhněte možná další rozšíření.

Literatura:

- Lacko, L.: Vývoj aplikací pro Android, Computer Press, 2015
- Juneau, J.: Java EE 7 Recipes, Apress, 2013
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3

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: **Burget Radek, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce se zabývá kompletním návrhem a implementací mobilního asistenta pro cestování v městské hromadné dopravě v Brně. Výsledné řešení se skládá z mobilní a serverové části. Mobilní aplikace umožňuje uživateli pomocí aktuální polohy vozidel MHD a polohy mobilního zařízení navigovat a určit nejoptimálnější trasu mezi zastávkami. Serverová část zahrnuje webovou službu pro komunikaci s klienty.

Abstract

This thesis is a documentation covering complete design and implementation of a mobile public transportation assistant for Brno. The resulting solution consists of a mobile application and a server part. The mobile application allow user with actual position of public transport vehicles and positon of smart device navigate and determine optimal route. The server part includes web service for client-server communication.

Klíčová slova

mobilní aplikace, android, soap, wsdl, rest, webová služba, ids jmk, knihovna Retrofit

Keywords

mobile aplication, android, soap, wsdl, rest, web service, ids jmk, library Retrofit

Citace

TŮMA, Jan. *MOBILNÍ ASISTENT PRO CESTOVÁNÍ MHD*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Burget Radek.

MOBILNÍ ASISTENT PRO CESTOVÁNÍ MHD

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Tůma
23. května 2017

Poděkování

Tímto bych chtěl poděkovat panu Ing. Radku Burgetovi, Ph.D., vedoucímu této práce za odbornou pomoc, ochotu a čas, kterou mi při tvorbě této práce věnoval. Dále bych chtěl poděkovat centrálnímu dispečinku Jihomoravského kraje, za poskytnutí potřebných informací o poloze vozidel MHD a zastávek.

Obsah

Kapitola 1

Úvod

Dnešní svět se vyznačuje rychlým a hektickým tempem životního stylu. Každý neustále spěchá, šetří každou chvíli, snaží se co nejrychleji dostat k nejrůznějším informacím. Spoustu času člověk stráví v dopravních prostředcích, když se snaží dostavit na schůzku, dojet do práce, na nákup, kamkoliv. V dnešní době, kdy jsou mobilní zařízení každodenní součástí našeho života a chytré telefony jsou velice dostupné, nabízí se poměrně silný potenciál pro vytvoření mobilní aplikace k získávání aktuálních informací o pohybu dopravních prostředků. A nabídnout tak uživateli možnost zvolení co nejlepšího spojení na požadované místo a tím ušetřit, v dnešní době pro nás všechny, tak drahocenný čas.

Hlavním cílem této diplomové práce bylo navrhnout a vytvořit mobilního asistenta, na platformu Android, pro získávání aktuální polohy prostředků městské hromadné dopravy v Brně. Na základě těchto informací poskytnout uživateli aplikace nejpřesnější a nejvýhodnější odjezdy, spojení a trasy do zvolené destinace. K mobilní aplikaci bude vytvořena serverová část, která bude poskytovat potřebná data. Takováto aplikace může být užitečná uživatelům, kteří hledají odjezdy a spojení vozidel na základě aktuální dopravní situace.

Na začátku druhé kapitoly je provedena analýza zadání a specifikace požadavků. Je v ní zjednodušeně popsána architektura budoucího řešení, specifikováni uživatelé a popsány potřebné technologie a vlastnosti mobilních zařízení.

Třetí kapitola se věnuje analýze existujících mobilních aplikací, které jsou svým užitím podobné vyvíjené aplikaci. Nakonec je provedeno shrnutí jejich výhod a nedostatků.

V kapitole číslo čtyři je obecně popsána tvorba webových aplikací. Nejprve jsou definovány a popsány technologie tvořící webové služby. Následně je přiblížen jejich vývoj a představeno prostředí pro jejich vývoj.

Pátá kapitola se obecně věnuje tvorbě mobilních aplikací pro operační systém Android. Stručně je v ní popsána architektura tohoto operačního systému, základní principy, základní aplikační komponenty, prostředí pro jejich vývoj, vzhledové směrnice Material design a knihovny pro komunikaci s webovými službami.

Kapitola šest popisuje služby a data, které byla poskytnuta centrálním dispečinkem Kordis Jihomoravského kraje. V kapitole jsou podrobněji popsány poskytnuté webové služby a statická data.

V sedmé kapitole je proveden celkový návrh celé práce. Nejdříve je detailněji popsána architektura finálního řešení, poté je proveden návrh mobilní aplikace a jejího uživatelského rozhraní. Následně je specifikováno rozhraní pro komunikaci se serverovou částí a realizován její návrh.

Kapitola číslo osm se věnuje samotné implementaci mobilní aplikace i serverové části. Implementace je rozdělena na tři iterace. V prvních dvou iteracích je popsáno jejich zamě-

ření, co implementují a poté je provedeno testování a ověření jejich funkčnosti. Následně je detailně popsána implementace finální implementace mobilní aplikace a serverové části.

V poslední kapitole je popsáno testování finální implementace. Jsou zde definována testovací zařízení, popsán testovací scénář a provedeno vyhodnocení testovacího dotazníku.

Kapitola 2

Specifikace a analýza požadavků

Tato kapitola popisuje a analyzuje požadavky na aplikaci. Dále se zabývá vlastnostmi a technologiemi cílových zařízení, které jsou pro funkčnost výsledné aplikace zásadní.

2.1 Zadání

Zadání této diplomové práce vzniklo na základě impulzu Ing. Radka Burgeta, Ph.D. Cílem této diplomové práce je vytvořit mobilního asistenta pro cestování MHD v Brně. Pod tímto pojmem si můžeme představit aplikaci pro mobilní zařízení na platformě Android, která na základě informací o pohybu vozidel MHD v Brně a pozice mobilního zařízení, bude zobrazovat aktuální pozice vozidel, odjezdy vozidel, počítat a zobrazovat zvolenou trasu. Potřebné informace, které byly zmíněny výše, budou získávány ze serverové aplikace, která bude také vytvořena v rámci této práce. Poloha vozidel bude získána v rámci spolupráce s dopravním dispečinkem Kordis Jihomoravského kraje.

2.2 Architektura řešení

Ze zadání vyplývá, že požadované řešení je typu klient – server, jehož schéma je znázorněno na obrázku 2.1. Jednotlivé části budou popsány v následujícím textu.

Obrázek 2.1: Architektura aplikace.

Klient

V rámci této práce roli klienta představuje mobilní aplikace pro použití během cestování městskou hromadnou dopravou. Jejím hlavním účelem je zobrazení odjezdů a optimální trasy z bodu A do bodu B v rámci sítě zastávek a spojů v Brně. To znamená výpočet optimální trasy na základě zastávek, pohybu vozidel MHD a pozice uživatele aplikace. Trasa

bude v určitém intervalu přepočítávána a pokud dojde k nalezení vhodnější trasy bude uživatel aplikace informován o této možnosti. Uživateli bude zobrazena informace o možném přestupu do jiného vozidla, přesunu se na jinou zastávku apod. Kromě vyhledávání optimální trasy bude aplikace poskytovat aktuální polohu veškerých vozidel MHD na mapě. Aplikace bude také podporovat rozšířené možnosti výběru spoje jako filtrování podle typu, směru vozidel, jen bezbariérové spoje apod.

Pro spojení se serverem, tudíž získávání polohy vozidel, bude aplikace využívat mobilního internetového spojení a pro získání aktuální pozice uživatele použije satelitní GPS spojení. Aplikace bude vyvinuta pro mobilní operační systém Android.

Server

Serverová část bude shromažďovat data poskytnutá od IDS JMK. Konkrétně polohu zastávek, pozici vozidel MHD. Serverová část bude také implementovat výpočty tras vozidel, různé transformace dat a ta budou zabalena do vhodného formátu na přenos po sítí. Bude také definováno vhodné aplikační rozhraní, pro komunikaci s klienty.

Aktéři

Ze zadání a specifikovaných požadavků vyplývá existence dvou následujících aktérů (rolí):

- **Uživatel aplikace** – jeho interakce se systémem se omezuje pouze na použití mobilní aplikace. Nebude mít možnost přímého přístupu na serverovou část.
- **Administrátor** – bude mít přístup na server a možnost upravovat serverovou část, např. aktualizovat zastávky.

2.3 Potřebné technologie a vlastnosti mobilních zařízení

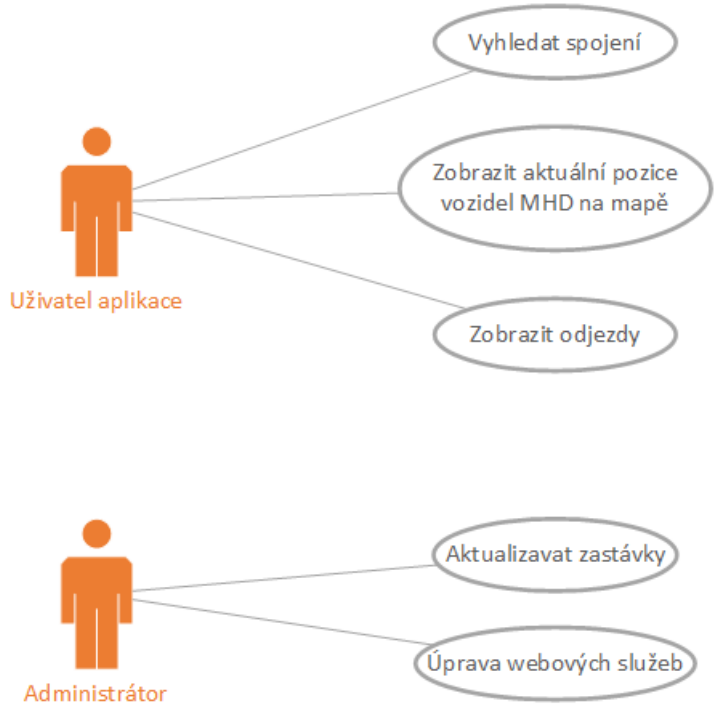
Cílová aplikace vyžaduje určité vlastnosti a technologie, které musí cílová mobilní zařízení podporovat. Tyto vlastnosti a technologie jsou popsány v následujícím textu. Dále je diskutována jejich použitelnost ve výsledné klientské aplikaci.

Určení polohy

Obsah této kapitoly vychází z [30], [20] a [27]. Systémy pro určování polohy jsou založeny na různých technologiích a mají v dnešní době v mobilním světě velmi významnou roli. Globální polohovací a navigační satelitní systémy (GNSS) jsou družicové radiové systémy, které slouží ke stanovení geografické pozice a navigování uživatelova přijímače kdekoliv na světě. V současné době jsou v činnosti tři systémy, a to americký GPS, ruský GLONASS a evropský GALILEO.

S pomocí GPS přijímače lze kdekoliv na planetě určit polohu, čas, rychlost a nadmořskou výšku. Kdysi vojenský systém GPS si po zpřístupnění k civilním účelům získal velkou popularitu a mnoho nových uživatelů. To hlavně díky těmto důvodům:

- vysoká míra přesnosti pro získání polohy
- bezplatné použití



Obrázek 2.2: Diagram případu použití specifikované aplikace.

- určení času a rychlosti
- dostupnost skoro po celém světě
- dostupnost GPS přijímačů miniaturních rozměrů

Systém GPS se skládá z 32 družic umístěných na oběžné dráze Země. Ke komunikace mezi družicemi a přijímačem dochází pomocí rádiových vln.

Určování polohy se počítá na základě tzv. zdánlivých vzdáleností od jednotlivých družic. Přijímač synchronizuje své hodiny s hodinami družic a pomocí dalších informací vysílaných z družic (jedná se hlavně o jejich dráhy), může zdánlivé vzdálenosti určit. Poté pomocí triletarace[1], lze určit aktuální polohu. Pro určení polohy je nutný současný příjem z minimálně čtyř družic. Dalším důležitým parametrem GPS přijímačů je čas pro získání polohy (TTFF).

Dnešní chytré telefony již standardně podporují systém GPS. Některá mobilní zařízení dokonce podporují kombinaci GLANSSu, GALILEA s GPS, což výrazně zvyšuje přesnost a snižuje čas TTFF. Díky tomuto můžeme s určováním polohy pomocí GPS počítat, při vývoji cílové aplikace.

Mapové podklady

Spolu s určováním polohy úzce souvisí dostupnost mapových podkladů. V dnešní době jsou běžně dostupné na každém mobilním zařízení a jsou, ve většině případů, používány jako součást navigačních aplikací. Společnosti vyvíjející mobilní operační systémy poskytují

[1] Trilaterace slouží k výpočtu vzdálenosti tří přijímačů resp. vysílačů. V případě systému GPS se počítá jako průnik povrchu koulí.

vlastní verze mapových podkladů[2]. K těmto podkladům poskytují vývojové nástroje, které jsou volně dostupné a pomáhají vývojářům k implementaci vlastní aplikace využívající mapové podklady, což je z pohledu této práce důležité.

Internetové připojení

Pro funkčnost navrhované mobilní aplikace je nutné datové připojení na server. Díky mobilnímu připojení bude získána aktuální pozice vozidel MHD a informace o odjezdech a spojích. V současné době je Česká republika z 99%[3] pokryta vysokorychlostním mobilním internetem LTE. Průměrná rychlost připojení pomocí technologie LTE je kolem 28Mbit/s. Primárně se počítá s fungováním aplikace výhradně v jihomoravské metropoli Brno a jeho přilehlém okolí, které je technologií LTE pokryto velmi dobře. Tudíž s komunikací mobilní aplikace a serveru by neměl být problém.

[2]GoogleMaps, Bing Maps, Apple Maps, Here
[3]Pokrytí LTE https://dotekomanie.cz/2017/02/pokryti-lte-cesku-dosahuje-az-99-lte/

Kapitola 3

Analýza existujících mobilních aplikací

V této kapitole budou popsány existující mobilní aplikace, které jsou svoji funkčností podobné vyvíjené aplikaci v rámci této práce. Kapitola komentuje jejich funkčnost a uživatelské rozhraní. Na tyto aplikace bude v rámci návrhu a implementaci brán zřetel.

3.1 iRIS

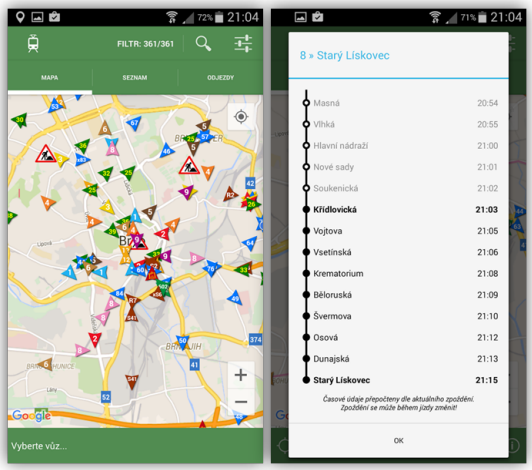
Aplikace zobrazuje aktuální polohu všech vozidel v systému integrovaného dopravního systému Jihomoravského kraje, včetně městské hromadné dopravy Brna, Blanska, Kyjova, Břeclavi, Vyškova, Adamova, Hodonína, Znojma, regionálních autobusů, vlaků a další informace o vozidlech. Poloha vozidel se přenáší pomocí dopravních dispečinků CEDRIS a iRIS a je aktualizována přibližně každých 10 sekund. Navigace aplikace je umístěna v horní části obrazovky, kde jsou 3 záložky. První záložka neboli hlavní obrazovka po startu aplikace se skládá z mapy, na které jsou zobrazeny ikony reprezentující vozidla a jejich směry. Druhá záložka obsahuje seznam všech aktuálně dostupných vozidel. Poslední, třetí, záložka slouží k vyhledavání odjezdů z vybrané zastávky. Aplikace je zdarma dostupná v obchodě Google Play [10].

3.2 Jízdní řády IDOS

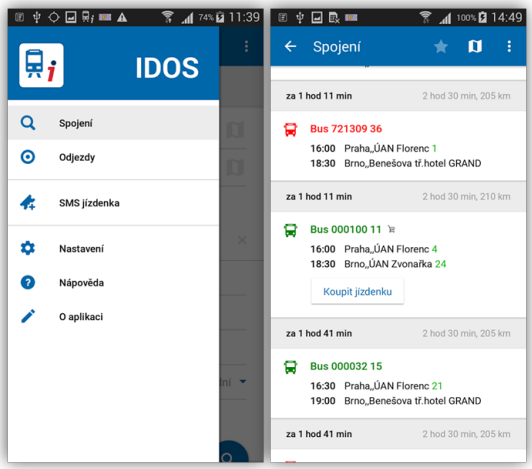
Jízdní řády IDOS umožňují snadné vyhledávání v jízdních řádech vlaků a MHD více než 60 měst ČR. Patří k nejoblíbenějším českým aplikacím pro vyhledávání spojů. Aplikace se vyznačuje intuitivním uživatelským rozhraním. Umožňuje vyhledání vlakových, autobusových a MHD spojů, online vyhledávání, inteligentní našeptávač zastávek, zakoupit SMS jízdenku, zobrazení spoje na mapě. Po startu aplikace je uživateli zobrazen jednoduchý formulář pro zadání počáteční, koncové stanice a dalších rozšiřujících možností hledání. Po vyhledání dostupných spojů je uživateli zobrazena obrazovka se seznamem možných tras. Aplikace je zdarma dostupná v obchodě Google Play [11].

3.3 Shrnutí analýzy

Tyto dvě aplikace budou výchozí pro návrh výsledné aplikace. První aplikace, iRIS, se snaží zobrazit aktuální pozice všech dostupných vozidel, což ve výsledku na malé obrazovce



Obrázek 3.1: Uživatelské rozhraní aplikace IRIS [10].



Obrázek 3.2: Uživatelské rozhraní aplikace IDOS [11].

mobilního zařízení je pro uživatele matoucí a nepřehledné. Aplikace IDOS patří po stránce uživatelského rozhraní a přívětivosti k tomu nejlepšímu v obchodě Google Play, ale zase nepracuje s aktuální pozicí vozidel.

Kapitola 4

Tvorba webových služeb

Tato kapitola popisuje a analyzuje nástroje a technologie, které budou použity pro vývoj webových služeb v rámci této práce.

Pro implementaci webových služeb byl zvolen jazyk Java, který je velmi rozšířený a podporovaný.

4.1 SOA

SOA (Service-Oriented Architecture)[23], [16] je paradigma předepisující způsob uspořádání a použití distribuovaných služeb, které mohou být spravovány různými vlastníky. Definuje jednotný způsob inzerce, hledání, vzájemné komunikace a spotřeby služeb s cílem dosáhnout požadovaných výsledku v souladu s měřitelnými předpoklady a očekáváním. SOA je tedy architektura orientovaná na služby, obecný koncept spolupracujících služeb.

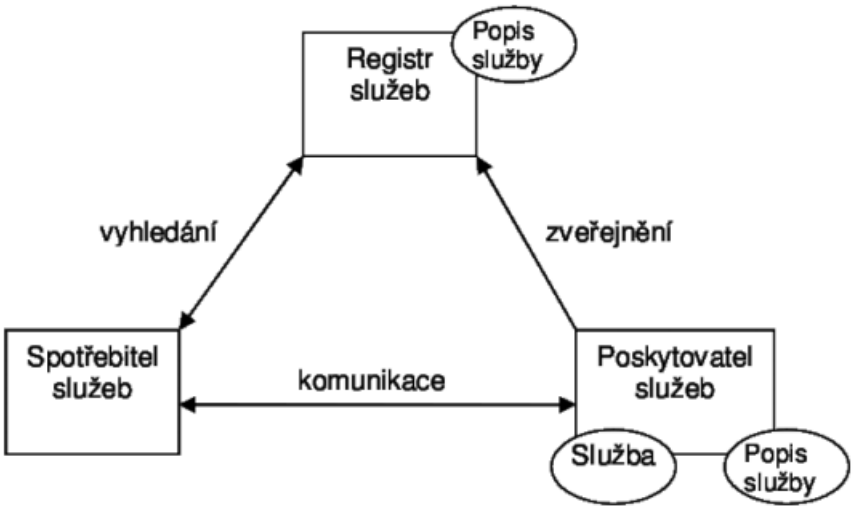
Vlastnosti SOA:

- SOA je paradigma – kompozice služeb
- SOA je distribuovaná – autonomní, ale spolupracující služby
- SOA je heterogenní prostředí – služba přístupná přes své rozhraní
- SOA je standardizovaná – popis rozhraní a způsob komunikace služeb
- SOA podporuje business – služby realizují konkrétní business procesy

4.2 Webové služby

Webové služby[9] jsou technologie pro implementaci SOA, které spravuje konsorcium W3C. Webové služby umožnují jednoduchou komunikaci mezi aplikacemi v heterogenních prostředích, protože je komunikace založena na platformě nezávislých standardů. A to konkrétně na jazyce XML a protokolu HTTP. Aplikace si mezi sebou posílají XML zprávy, které přenášejí dotazy a odpovědi jednotlivých aplikací. Základní technologie tvořící celou infrastrukturu webových služeb jsou následující:

- **SOAP** (Simple Object Access Protocol) – protokol pro výměnu zpráv založených na XML



Obrázek 4.1: Konceptuální model SOA-model interakce mezi poskytovatelem služeb a spotřebitelem služeb [22]

- **WSDL** (Web Services Description Language) – standardní formát pro popis rozhraní webové služby
- **UDDI** (Universal Description, Discovery and Integration) – registr webových služeb, standardní mechanismus umožňující registraci a vyhledávání webových služeb

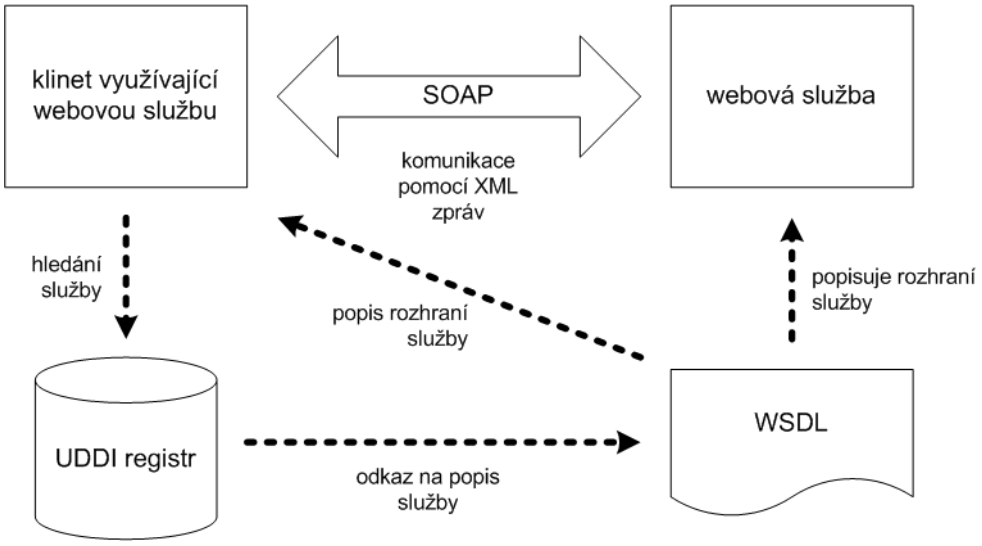
Princip použití webových služeb je následující. Klient z registru webových služeb (UDDI) vybere webovou službu, kterou hodlá využít a získá její popis (WSDL). Na základě toho popisu může zjistit bližší informace o využití webové služby, jako její adresu, popis metod, struktury dotazu i odpovědi. Podle těchto informací může klient posílat webové službě strukturované dotazy pomocí protokolu SOAP. Stejným protokolem dostane i zpracovanou odpověď odeslanou webovou službou. Odpověď klient dostane v XML formě, kterou lze jednoduše dále zpracovat.

SOAP

Obsah této kapitoly vychází z [12], [29] a [28]. SOAP (Simple Object Access Protocol) je základem webových služek. Jedná se o protokol pro výměnu strukturovaných zpráv ve formátu XML v decentralizovaném a distribuovaném prostředí. Definuje strukturu jednotlivých zpráv, které si aplikace vyměňují. Jedná se o protokol aplikační vrstvy, který však k samotnému odesílání jednotlivých zpráv používá jiný protokol aplikační vrstvy, převážně pak HTTP (metodou POST).

Struktura SOAP zprávy:

SOAP zpráva je definována v jazyku XML, to hlavně z důvodu snadné transformace XML dat do formátu potřebného pro specifickou aplikaci komunikující pomocí SOAP protokolu. Výhodou XML syntaxe je také její snadná čitelnost zpráv uživateli a snadná validace, tím lze předejít chybám. SOAP zpráva se skládá z následujících částí:



Obrázek 4.2: Vzájemné vztahy mezi technologiemi webových služeb [14].

- **Obálka (envelope)** - Jedná se o kořenový element SOAP zprávy. Obsahuje definice jmenných prostorů, které identifikují značky v XML. Element envelope uzavírá všechna přenášená data (kromě příloh) do obálky. Dále obsahuje další dva elementy: hlavičku (header) a tělo (body).
- **Hlavička (header)** - Hlavička je nepovinná a používá se pro přenos pomocných informací pro zpracování zprávy – například identifikaci uživatele, autentizační informace (jméno, heslo) apod. Hlavička může také obsahovat další atributy jako:
  - mustUnderstand – určuje, zda příjemce je povinen se zprávou zpracovat i hlavičku (musí jí rozumět)
  - actor – zpráva může cestovat od odesílatele k příjemci přes více koncových bodů a tento atribut umožňuje směřovat hlavičku právě některému z nich
  - encodingStyle - obsahuje odkaz na definici datových typů
- **Tělo (body)** - Tělo je povinné a obsahuje vlastní XML zprávu s požadavkem nebo odpovědí konkrétní webové služby.

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <env:Envelope
03.   xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
04.   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
05.   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
06.  <env:Header/>
07.  <env:Body>
08.   <jePrvocislo xmlns="urn:mojeURI">
09.     <cislo xsi:type="xsd:long">1987</cislo>
10.   </jePrvocislo>
11.  </env:Body>
12. </env:Envelope>
```

Obrázek 4.3: Ukázka jednoduché SOAP zprávy [15].

Obrázek 4.3 ukazuje velmi jednoduchou SOAP zprávu na zjištění informací o tom, zda je číslo 1987 prvočíslo. V těle zprávy je požadavek na vyvolání vzdálené funkce jePrvocislo

s parametrem pojmenovaným cislo s hodnotou 1987. Obrázek 4.4 ukazuje SOAP zprávu s odpovědí. Odpověď zobrazuje informaci o tom, že číslo je prvočíslo.

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <env:Envelope
03.   xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
04.   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
05.   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
06.  >
07.  <env:Body>
08.   <jePrvocisloResponse xmlns="urn:mojeURI">
09.    <vysledek xsi:type="xsd:boolean">true</vysledek>
10.   </jePrvocisloResponse>
11.  </env:Body>
12. </env:Envelope>
```

Obrázek 4.4: Ukázka SOAP odpovědi [15].

WSDL

WSDL[7] je jazyk založený na XML. Slouží k popisu konkrétní webové služby, jejího umístění, metod, parametrů a dalších informací nutných pro přístup a komunikaci. WSDL má přesně danou strukturu, a je dobře programově zpracovatelný. Pokud webová služba obsahuje korektní WSDL soubor je přístup ke službě velice jednoduchý. WSDL lze totiž považovat za jakousi dokumentaci pro SOAP klienta. Většina dnešních SOAP klientů je schopna při inicializaci zpracovat WSDL soubor a následně nabízet přímé volání funkcí vzdálené webové služby. Programátor proto nemusí znát detaily komunikace.

WSDL je tedy XML soubor definující rozhraní služby. Každý WSDL popis se skládá zejména z následujících elementů:

- **types** – Obsahuje definici datových struktur používaných ve zprávách. K definici lze použít teoreticky libovolný typový systém, ale nejčastěji se používají XML schémata. Nástroje pro webové služby se starají o mapování datových typů podle XML schémat na nativní datové typy použitého jazyka.
- **message** – Definuje formát předávaných zpráv pomocí dříve definovaných datových typů. Zprávy fungují jako vstupní anebo výstupní struktury pro operace. Každá zpráva se může skládat z několika logických částí s vlastním datovým typem. Při použití SOAPu pro RPC odpovídá jedna část zprávy jednomu parametru vzdálené metody.
- **operation** – Abstraktní definice operací, které jsou službou podporovány. U operace se definuje, jaké má vstupy a výstupy. Vstup a výstup je popsán již existující zprávou (message). V SOAP RPC modelu odpovídá operace metodě.
- **portType** – Sdružuje dohromady několik operací.
- **binding** – Slouží pro navázání určitého typu portu (portType) na konkrétní protokol a formát přenosu zpráv.
- **port** - Jeden koncový bod služby definovaný jako kombinace síťové adresy a dříve definované vazby (binding).
- **service** – Sdružuje několik koncových bodů (portů) do jedné služby.

Obrázek 4.5 ukazuje soubor s WSDL popisem rozhraní pro výše uvedené služby pro zjištění, zda je číslo prvočíslem. Struktura je shodná s výše uvedenou definicí.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="PrvniSluzba"
 targetNamespace="urn:mojeURI"
 xmlns:tns="urn:mojeURI"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns1="urn:mojeURI"
 xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

<!-- definice typů -->
<types>
 <schema targetNamespace="urn:mojeURI"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="unqualified"
      attributeFormDefault="unqualified">
  <element name="cislo" type="xsd:long"/>
  <element name="vysledek" type="xsd:boolean"/>
 </schema>
</types>

<!-- komunikační zprávy -->
<message name="jePrvocisloRequest">
 <part name="cislo" element="ns1:cislo"/>
</message>
<message name="jePrvocisloResponse">
 <part name="vysledek" element="ns1:vysledek"/>
</message>

<!-- dostupné operace -->
<portType name="Cisilka">
 <operation name="jePrvocislo">
  <documentation>Operace jePrvocislo()</documentation>
  <input message="tns:jePrvocisloRequest"/>
  <output message="tns:jePrvocisloResponse"/>
 </operation>
</portType>

<!-- volatelné přes HTTP -->
<binding name="PrvniSluzba" type="tns:Cisilka">
 <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="jePrvocislo">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
   <SOAP:body use="literal" namespace="urn:mojeURI"/>
  </input>
  <output>
   <SOAP:body use="literal" namespace="urn:mojeURI"/>
  </output>
 </operation>
</binding>

<!-- adresy komunikačních bodů -->
<service name="PrvniSluzba">
 <documentation>Sluzba pocitajici prvocisla</documentation>
 <port name="PrvniSluzba" binding="tns:PrvniSluzba">
  <SOAP:address location="http://localhost:10000"/>
 </port>
</service>
</definitions>
```

Obrázek 4.5: Ukázka WSDL popisu webové služby [15].

UDDI

Poslední součástí, která patří k webovým službám založeným na protokolu SOAP je UDDI[25]. Zjednodušeně řečeno, UDDI je katalog webových služeb. Ukládají se zde především informace o umístění webových služeb, jejich popis, dále lze v registru nalézt také informace o majiteli služby, případně geografické informace a oblast působnosti podniku, která webové služby nabízí. Základní smysl UDDI je třídění webových služeb podle různých parametrů a umožnit tak snadné nalezení požadovaných informací o konkrétní službě. Většina velkých softwarových firem má svůj UDDI registr. Samotný registr pracuje rovněž jako webová služba a komunikace s ní tedy opět probíhá pomocí SOAP protokolu. UDDI registr obsahuje následující čtyři druhy entit:

- **usiness entity** – popisuje firmu dodávající služby,
- **service servide** – popisuje skupiny služeb dodávaných firmou,
- **service type** – je technický model popisující typ služby (odkaz na WSDL) nebo použitý protokol
- **binding template** – popisuje informaci nezbytnou pro použití služby

REST

REST (Representational State Transfer)[21], [17] je architektonické rozhraní, navržené pro distribuované prostředí. Někdy je REST zařazován do skupiny protokolů společně se SOAP, což je špatně, protože se nejedná o protokol ani formát. Jak již bylo řečeno, REST je architektonický vzor. Definuje základní metody zvané CRUD, pomocí kterých lze přistupovat k datovým zdrojům rozhraní. Tyto metody jsou mapovány na metody protokolu HTTP podle následujícího rozdělení: Create (POST) - slouží k přidávání dat, Retrieve (GET) - získání dat, Update (PUT) - editace dat, Delete (DELETE) - slouží k odstranění dat. Odpověď na požadavek může být v libovolném formátu (např.: XML, JSON, RSS, plain-text). REST je bezstavový, proto je nutné u každého požadavku zadávat všechny parametry, které jsou potřeba. REST je zaměřený spíše na data než na operace.

REST je tedy architektonický styl a abychom byli schopni rozhodnout, zda aplikace dodržuje tento vzor, je nutné definovat seznam omezení:

- klient-server (client-server)
- bezstavovost (stateless)
- možnost využití vyrovnávací paměti (cacheable)
- kód na vyžádání (code on demand)
- vrstvený systém (layered system)
- jednotné rozhraní (uniform interface)

Dodržením těchto omezeních nám potom dává tyto výhody:

- škálovatelnost komunikace mezi komponentami
- jednotné rozhraní pro komunikaci mezi komponentami

- možnost použití prostředníků v komunikaci (proxy)
- robustnost a efektivita – možnost použití vyrovnávací paměti
- možnost nezávislého vývoje jednotlivých komponent (volně vázané systémy)
- jednodušší implementace komponent – není nutné řešit problém komunikace, všichni mají stejnou sadu metod a stejné rozhraní

4.3 Java EE

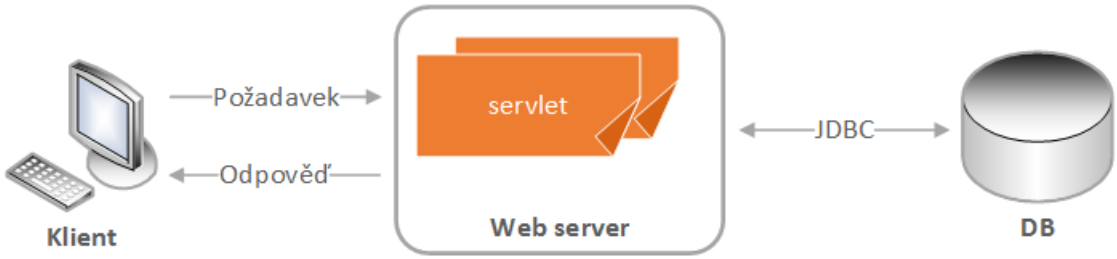
Java Enterprise Edition[22], [13] je standardizovaná platforma určená pro vývoj přenosných, robustních, škálovatelných a dobře zabezpečených serverových aplikací v jazyce Java schopných běhu ve specializovaných prostředích označovaných jako aplikační servery. Java EE poskytuje vývojářům infrastrukturu, která jim usnadní jejich vývoj. Rozšiřuje platformu Java SE o podporu:

- pro tvorbu webových aplikací
- webových služeb
- distribuovaných vícevrstvých aplikací

V rámci této práce bude vytvořeno RESTful API, proto v následující části budou představeny nástroje pro jazyk Java, které tvorbu tohoto API usnadňují.

Java Servlet

Pomocí Java Servlet API[18] a třídy servlet lze implementovat jednoduché REST API. Servlet je standardní třída, která je součástí každého webového kontejneru v Javě a poskytuje rozhraní pro práci s HTTP požadavky. Při použití toho rozhraní je nutné definovat několik metod, jako jsou: `doGet`, `doPost` a další. Každému servletu je v rámci webové aplikace přiděleno URI pro jednoznačnou identifikaci. Servlety implementují princip požadavku / odpověď, typický pro architekturu klient-server. Java Servlet API definuje standardní rozhraní pro obsluhu těchto požadavků a odpovědí mezi klientem a serverem. Následující obrázek zjednodušeně ilustruje procesy mezi klientem a serverem.



Obrázek 4.6: Základní procesy mezi klientem a serverem.

1. Klient pošle požadavek na server.
2. Server přepošle požadavek na konkrétní servlet.

3. Servlet vytvoří odpověď a předá ji serveru. (Odpověď je vytvořená dynamicky a její obsah závisí od požadavku, který poslal klient.)

4. Server pošle klientovi odpověď.

JAX-RS

JAX-RS[24] je specifikace popisující API poskytující podporu pro budování REST služeb na platformě Java. Tato specifikace vznikla v rámci JSR 311, před vznikem této specifikace neexistoval žádný standard pro tvoru REST služeb. Cílem JAX-RS je poskytnutí standardizovaného programového rozhraní pro tvorbu REST aplikací, které je jednoduché na použití a je nezávislé na dalších použitých technologiích. Tuto ideu se snaží naplnit pomocí řady anotací, které umožnují webovému serveru zpřístupňovat zdroje. Základními anotacemi jsou:

- @Path - umožňuje nastavení relativní cesty ke zdroji (část URI)
- @GET, @PUT, @POST, @DELETE a @HEAD - určují použití HTTP metod
- @Produces - určuje MIME typ, který metoda produkuje
- @Consumes - určuje MIME typ, který metoda umí přijmout

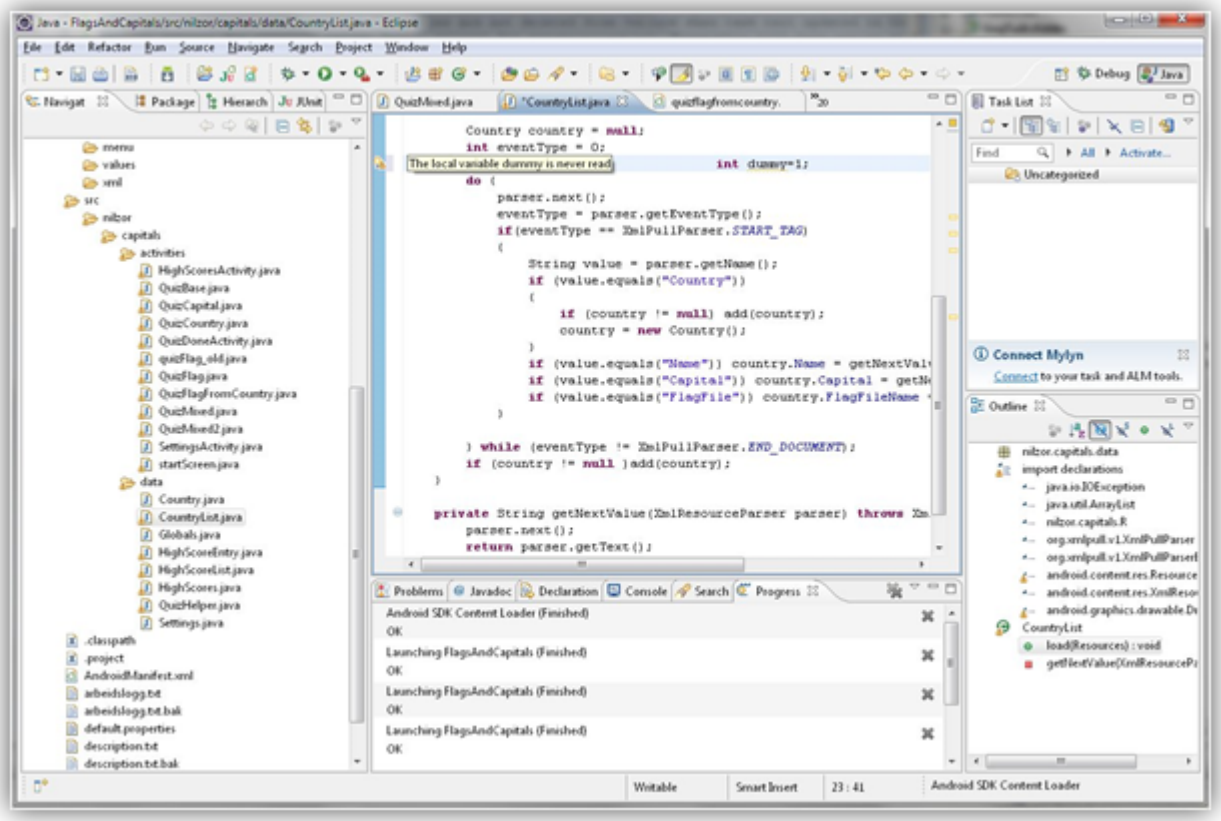
4.4 Vývoj webových služeb

Pro vývoj webových služeb v jazyce Java je ideální nástroj Eclipse[1], který poskytuje vývojovou platformu pro různé programovací jazyky. Jeho největší výhodou je možnost instalace zásuvných modulů, které umožňují rozšířit funkčnost pro různé potřeby.

V základní verzi obsahuje Eclipse pouze integrované prostředky pro vývoj standardní Javy jako kompilátor, debugger atd., ale neobsahuje například nástroj pro vizuální návrh grafických uživatelských rozhraní desktopových aplikací nebo aplikační server – všechna taková rozšíření je potřeba dodat formou pluginů.

Pro běh webových služeb je potřeba aplikační server, což je vrstva mezi operačním systémem a webovou aplikací. Jedním z nejpopulárnějších aplikačních serverů je Apache Tomcat[2], který je vyvíjen jako open source projekt a poskytován pod licencí Apache License verze 2[3]. Je jednoduchý, nenáročný a dobře poslouží na menší projekty. Pro Tomcat je k dispozici základní management rozhraní, kde může administrátor najít statistiky serveru a spravovat nahrané aplikace

[1] Eclipse IDE https://www.eclipse.org/downloads/?
[2] Apache Tomcat http://tomcat.apache.org/
[3] Apache License verze 2 https://www.apache.org/licenses/LICENSE-2.0



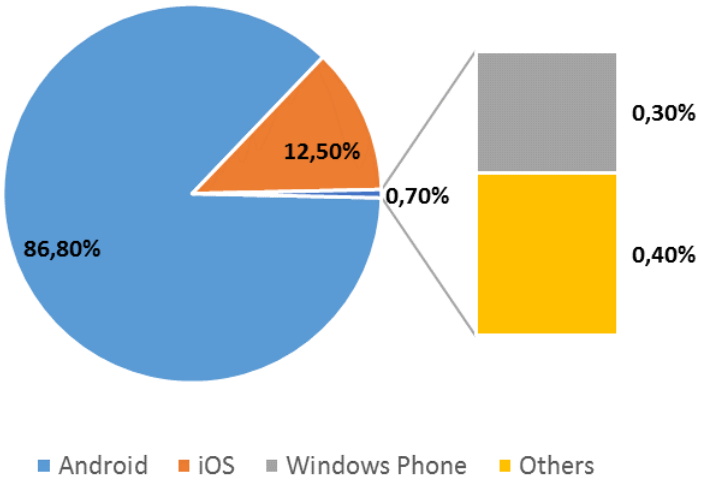
Obrázek 4.7: Ukázka rozhraní vývojového prostředí Eclipse[1].

Kapitola 5

Tvorba mobilních aplikací pro Android

Tato kapitola popisuje a analyzuje nástroje a technologie, které budou použity pro vývoj mobilní aplikace v rámci této práce.

Pro vývoj mobilní aplikace byl zvolen operační systém Android[8]. Hlavním důvodem této volby byla rozšířenost a dostupnost zařízení s tímto systémem viz. obrázek 5.1, díky tomu by měla být aplikace dostupná co největšímu počtu uživatelů.



Obrázek 5.1: Porovnání rozšířenosti mobilních zařízení s mobilními operačními systémy.

Android je rozsáhlá open source platforma primárně určená pro chytré mobilní telefony a tablety. V dnešní době se začíná dostávat i do chytrých hodinek, notebooků a dalších zařízení. Tato platforma v sobě zahrnuje operační systém, který je založen na linuxovém jádře, uživatelské rozhraní, aplikacích a middleware (softwarová vrstva ležící mezi aplikacemi a operačním systémem). Většina zdrojového kódu je dostupná jako open-source pod licencí Apache. Historie Androidu se začala psát v roce 2003, kdy Andy Rubin a jeho společníci založili firmu Android Inc. V roce 2005 tuto firmu odkoupila společnost Google a vytvořila

z ní svoji dceřinou společnost. Po třech letech vývoje byla oficiálně přestavena první verze systému: Android 1.0 na zařízení HTC G1 'Dream'.

5.1 Architektura systému Android

Jak již bylo zmíněno Android je open source, založený na Linuxovém jádře vytvořený pro širokou škálu zařízení. Následující diagram ukazuje hlavní vrstvy platformy Android, kde každá vrstva má svůj účel a nemusí být přímo oddělena od ostatních vrstev[5].

- **Linuxové jádro (Linux Kernel)** – Tvoří nejnižší vrstvu, která je základem platformy Android. Použití jádra systému Linux umožňuje systému Android využívat výhod zabezpečení Linuxového jádra a usnadňuje vývoj ovladačů. Linuxové jádro zajišťuje základní správu paměti, procesů, bezpečnosti.
- **Hardwarová abstraktní vrstva (HAL)** – HAL poskytuje standardní rozhraní, která umožňuje komunikaci hardwaru s vyšší úrovní API rozhraní. HAL se skládá z několika modulů knihoven, z nichž každá implementuje rozhraní pro konkrétní typ hardwarové komponenty, jako je například modul fotoaparátu nebo bluetooth. Když rozhraní Java API Framework vyvolá přístup k hardwaru zařízení, systém Android načte knihovní modul pro tuto hardwarovou komponentu.
- **Android Runtime (ART)** – ART je vrstva pro řízení běhu aplikací a některých systémových služeb. Využívá zpracování AOT (Ahead-Of-Time) to znamená, že se aplikace překládá z byte-kódu na strojový kód už při instalaci. Výhodou je menší náročnost, tudíž větší výdrž baterie, lepší multitasking a celkově větší plynulost prostředí. Před Androidem verze 5.0 (úroveň 21 API) se o řízení běhu aplikací staral virtuální stroj Dalvik.
- **Nativní knihovny C/C++** - Mnoho základních komponent a služeb systému Android, jako například ART a HAL, je postaveno na nativním kódu, který vyžaduje nativní knihovny napsané v C a C ++. Platforma Android poskytuje rozhraní pro přístup k některým těmto nativním knihovnám (např. OpenGL).
- **Java API Framework** - Celá sada funkcí operačního systému Android je k dispozici prostřednictvím tohoto rozhraní napsaného v jazyce Java. Toto rozhraní tvoří stavební bloky, které jsou potřeba k vytváření aplikací pro Android, a to zjednodušením opětovného použití základních, modulárních součástí a služeb systému.
- **Systémové aplikace** - Android obsahuje sadu hlavních aplikací pro e-maily, SMS zprávy, kalendáře, prohlížení internetu, kontakty a další. Systémové aplikace umožňují uživateli využít svoje funkce při tvorbě vlastních aplikací, jako například zpřístupnění SMS zpráv apod.

5.2 Základní principy Androidu

Aplikace pro Android[26] jsou napsány v programovacím jazyce Java. Nástroje SDK pro Android kompilují váš kód spolu s veškerými daty a zdroji do souboru APK, balíčku pro Android, který je archivačním souborem s příponou .apk. Jeden soubor APK obsahuje veškerý

Obrázek 5.2: Diagram architektury operačního systému Android, ukazující pět základních [5].

obsah Android aplikace a představuje soubor, který zařízení Android používá k instalaci aplikace.

Každá aplikace pro Android má vlastní bezpečnostní obálku (sandbox) chráněnou těmito funkcemi zabezpečení Androidu:

- každá aplikace má jedinečné uživatelské jméno systému Linux
- každý proces má svůj vlastní virtuální počítač (VM), takže kód aplikace běží izolovaně od ostatních aplikací
- každá aplikace je spuštěna ve vlastním procesu

Systém Android implementuje zásadu nejméně privilegií. To znamená, že každá aplikace ve výchozím nastavení má přístup pouze ke komponentám, které potřebuje k tomu, aby vykonávala svoji práci a ne více. To vytváří velmi bezpečné prostředí, ve kterém aplikace nemůže přistupovat k částem systému, pro které nemá povolení. Existují však způsoby, jak aplikace může sdílet data s jinými aplikacemi a přistupovat k systémovým službám.

5.3 Aplikační komponenty

Aplikace pro operační systém Android jsou rozdělené do aplikačních komponent[1]. Ty tvoří základní stavební prvky každé Androidí aplikace. Každá komponenta je vstupní bodem skrze ní může systém nebo uživatel pracovat s aplikací. Některé komponenty jsou na sobě závislé. Hlavní výhodou je znovu použitelnost těchto komponent v jiných aplikacích. Android se skládá z následujících komponent:

1. Aktivity (Activity)

Aktivity v aplikaci reprezentují prezentační vrstvu. Jedná se o základní vizuální komponentu, která reprezentuje jednu obrazovku aplikace. Prostřednictvím aktivit mohou uživatelé komunikovat. Aplikace je většinou složena z více aktivit, které mezi sebou mohou sdílet data. Každá aktivita má vlastní životní cyklus viz. obrázek 5.4 a je vždy v jednom z následujících stavů:

- **Běžící (running)** – Aktivita je spuštěná a běží na popředí.
- **Pozastavená (paused)** – Aktivita je viditelná, ale již není na popředí. Je většinou překryta nějakým upozorněním (např. příchozí hovor).
- **Zastavená (stopped)** – Aktivita není viditelná a běží na pozadí, je překryta jinou aktivitou. Je stále uložena v paměti, ale při jejím nedostatku může být systémem ukončena.
- **Ukončená (destroyed)** – Aktivita, která byla ukončena násilně nebo z nedostatku paměti. Po ukončení je odstraněna z paměti.

Každá aktivita pro přechod mezi uvedenými stavy reaguje na následující metody:

- **onCreate()** - Metoda je volána při vytvoření aktivity.
- **onStart()** - Metoda je volána, pokud se uživatel rozhodne vrátit do aktivity.
- **onResume()** - Metoda je volána, když aktivita komunikuje s uživatelem.
- **onPause()** - Metoda je volána, pokud dojde uživatelem k přesunu do jiné aktivity.
- **onStop()** – Metoda je volána v době, kdy je aktivita pro uživatele déle neviditelná.
- **onDestroy()** - Metoda je volána, pokud dojde v rámci aplikace k ukončení aktivity.
- **onRestart()** – Metoda je volána, pokud je aktivita zastavena a je jí potřeba obnovit.

2. Fragmenty (Fragments)

Fragment představuje chování nebo část uživatelského rozhraní v aktivitě. Více fragmentů lze kombinovat v jediné aktivitě a vytvořit víceúčelové uživatelské rozhraní. Fragment lze znovu použít ve více aktivitách. Fragment si lze představit jako modulární část aktivity, která má svůj vlastní životní cyklus, přijímá vlastní vstupní události a lze ho dynamicky přidávat nebo odstraňovat, zatímco je aktivita spuštěna.

3. Služby (Services)

Služby jsou součástí aplikace, které neposkytují uživateli grafické uživatelské rozhraní. Slouží k provádění dlouhotrvajících operací na pozadí. Službu může spustit komponenta aplikace a zůstává stále aktivní na pozadí, i když uživatel přejde do jiné aplikace.

4. Poskytovatelé obsahu (Content providers)

Poskytovatelé obsahu pracují s daty (ukládají, načítají) a zpřístupňují je pro všechny aplikace. Jelikož neexistuje žádné běžné uložiště, do kterého by měly možnost přistupovat všechny balíčky systému Android, proto poskytovatelé obsahu představují jediný způsob, jak sdílet data v rámci aplikace.

5. Přijímače (Broadcast receiver)

Přijímač je komponenta, která slouží k naslouchání zprávám ze systému nebo z jiných aplikací. Podle typu zprávy následuje reakce, například výpis do stavového řádku, nebo lze jejich prostřednictvím spouštět jiné komponenty.

6. Záměry (Intents)

Záměry slouží ke komunikaci a přemisťování mezi jednotlivými komponentami aplikace. Aplikace se v podstatě skládá z komponent (aktivity) a ze zpráv mezi komponentami (záměry). Záměr se obecně skládá z činnosti, která se má vykonat, parametru, nad kterým má být tato činnost vykonána, a aplikace, která má tuto akci provést.

6. Záměry (Intents)

Jsou základní elementy uživatelského rozhraní (tlačítka, seznamy, texty). Jejich definice se zapisuje do XML souboru v podobě stromové struktury.

8. Zdroje (Resources)

Pro lepší udržovatelnost a správu aplikace je vhodné veškeré externí zdroje, jako například obrázky a řetězce apod., uchovávat nezávisle. Externalizace zdrojů také umožňuje poskytnout alternativní zdroje, které podporují specifická konfigurace zařízení, jako jsou různé jazyky nebo velikosti obrazovky. To se stává stále důležitější, jelikož stále více zařízení se systémem Android je k dispozici v různých konfiguracích. Aby byla zajištěna kompatibilita s různými konfiguracemi, je třeba organizovat zdroje v adresáři `res/` projektu pomocí různých podadresářů, které seskupují zdroje podle typu a konfigurace.

5.4 Vývoj aplikace

Aplikace pro Android se většinou programují v jazyce Java, je proto nutné mít nainstalován Java Development Kit (JDK)[1]. JDK poskytuje základní soubor nástrojů a knihoven pro platformu Java. Dále je třeba mít nainstalován Android Software Development Kit (SDK)[2], jenž obsahuje balíček nástrojů pro vývoj aplikací pro Android. Android SDK obsahuje knihovny API, dokumentaci, ukázkové příklady, a hlavně virtuální mobilní zařízení, které lze při vývoji a ladění použít. Pro samostatný vývoj aplikací je vhodné použít vývojové prostředí Android Studio[3], které nahradilo dříve používaný plugin Android Development Tools pro prostředí Eclipse.

Android Studio je oficiální integrované vývojové prostředí (IDE) pro vývoj Android aplikací, které je založené na IntelliJ IDEA prostředí. Vedle výkonného editoru a vývojářských nástrojů IntelliJ nabízí Android Studio ještě další funkce, které zvyšují produktivitu při vytváření aplikací, například:

- Flexibilní systém založený na systému Gradle
- Rychlý emulátor s bohatou funkcí
- Jednotné prostředí, ve kterém se můžete vyvíjet pro všechna zařízení Android
- Okamžité spuštění, bez vytváření nového souboru APK
- Šablony kódu a integrace GitHub
- Rozsáhlé testovací nástroje a rámce
- Lint nástroje k zachycení výkonu, použitelnosti, kompatibility verze
- Podpora C ++ a NDK
- Vestavěná podpora pro platformu Google Cloud Platform

5.5 Material Design

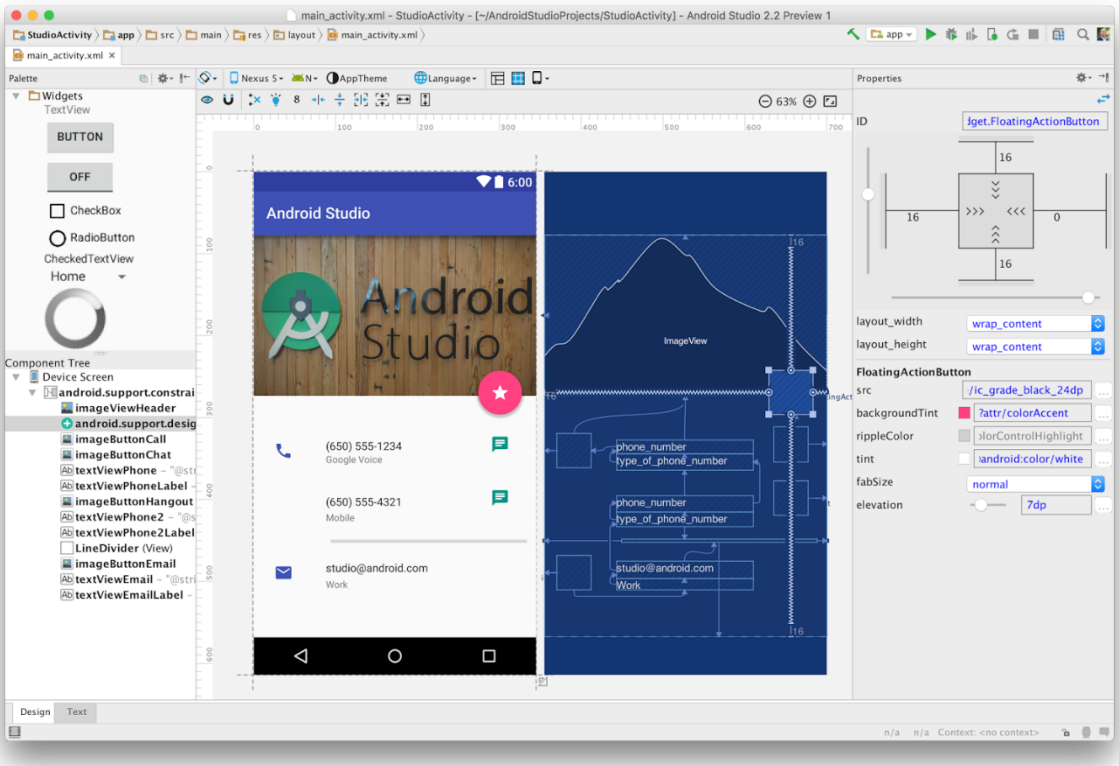
Material Design[2] je komplexním návodem pro vizuální, pohybový a interakční design mezi platformami a zařízeními. Tento nový předpis pro tvorbu aplikací přišel společně s verzí systému 5.0 Lollipoop (API 21) a společnost Google tento styl postupně přenesla do svých mobilních aplikací a později i do rodiny svých webů. Cílem Material Designu je sjednotit vzhled a chování aplikací na různých platformách. Material Design staví na následujících principech:

- Materiál je metafora – teorie sjednocující prostor a pohyb. Je inspirován studiem papíru a inkoustu.
- Důraz na grafiku, typografii – záměrná volba barev, písma, hran objektů
- Pohyb podporuje význam – pohyb je smysluplný a vhodný, slouží k soustředění pozornosti uživatele, je jemný a plynulý

[1] Java Development Kit (JDK) http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

[2] Android Software Development Kit (SDK) https://developer.android.com/studio/index.html

[3] Android Studio https://developer.android.com/studio/index.html



Obrázek 5.3: Ukázka vývojového prostředí Android Studio[3].

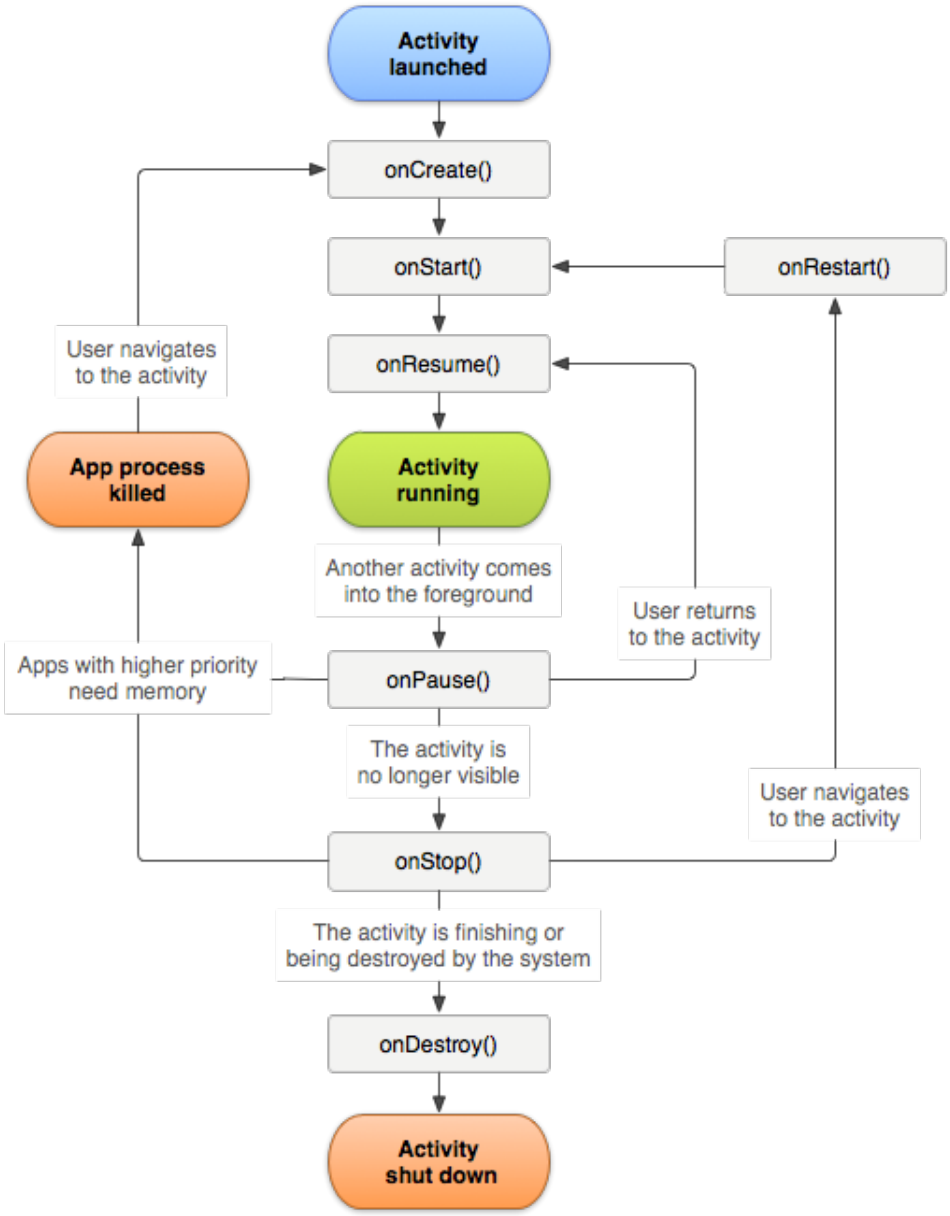
5.6 Knihovny pro komunikaci s webovými službami

Pro operační systém Android existuje několik knihoven pro zjednodušení komunikace mezi klientem a serverem. Mezi tyto knihovny patří: Volley[4], Retrofit[5] a další.

Volley je síťová knihovna vyvinutá inženýry společnosti Google. Byla představena na Google IO 2013. Nabízí skvělé funkce, jako jsou synchronní a asynchronní požadavky, stanovení priorit, několik požadavků najednou, řazené požadavky a samozřejmě ukládání do mezipaměti. Jedním z hlavních problémů čekající na vývojáře, kteří tuto knihovnu používají, je absence podrobné oficiální dokumentace.

Retrofit je čistá, jednoduchá a lehká knihovna pro Android od společnosti Square, Inc. Jinými slovy, Retrofit je REST klient pro Android, díky němuž můžete snadno používat rozhraní. Retrofit se liší tím, že může provádět asynchronní a synchronizační požadavky s automatickým analyzováním a parsování JSONu bez jakékoli námahy a zásahu uživatele. To díky externím knihovnám jako Gson[6].

[4]Knihovna Volley https://developer.android.com/training/volley/index.html
[5]Knihovna Retrofit http://square.github.io/retrofit/
[6]Gson https://github.com/google/gson



Obrázek 5.4: Diagram zobrazují životní cyklus aktivity [6].

Kapitola 6

Služby a data IDS JMK

Tato kapitola popisuje data a služby poskytnuté od centrálního dispečinku Kordis Jihomoravského kraje. Bez této spolupráce by bylo velmi náročné až neproveditelné tuto práci vypracovat. Dispečink poskytuje informace jako: poloha vozidel, definici služeb, návaznosti spojů, polohu zastávek a další.

6.1 Popis webových služeb

V rámci zmíněné spolupráce byl poskytnut přístup k rozhraní webové služby, která poskytuje většinu potřebných informací. Konktrétně se jedná o webovou službu typu SOAP s WSDL popisem, tyto pojmy byly popsány v kapitole 4. Na základě poskytnutého WSDL popisu webové služby je možné, pomocí například vývojového prostředí Eclipse, vygenerovat webového klienta a skrze něj přistupovat k implementovaným metodám zpřístupňujícím potřebné data. Proto není třeba navrhovat a implementovat celé webové služby úplně od začátku, ale stačí analyzovat poskytnuté metody a vybrat jen potřebné. Nad nimi potom provést různé transformace, výpočty a navrhnout jednotné rozhraní pro jejich volání.

Pro prvotní analýzu poskytnutých webových služeb je vhodné použít specializované nástroje pro to určené jako například SoapUI. SoapUI[1] je nástrojem společnosti SmartBear Software pro tvorbu komplexních funkčních a nefunkčních testů. SoapUI podporuje řadu technologií a standardů: SOAP a REST webové služby, JMS nebo RIA. Umožňuje na základě WSDL popisu vygenerovat testovací scénáře, připravit modelové požadavky a celou komunikaci zobrazit v grafickém rozhraní. Z poskytnutého WSDL popisu lze po analýze získat následující třídy a metody:

ActualTrafficPerformance

Tato třída poskytuje jedinou metodu `GetActualTrafficPerformance`, která vrací procentuální statistiky o zpožděních veškerých spojů.

ElpInfo

Tato třída poskytuje bezparametrovou metodu `GetPanelInfo`, která vrací informace o veškerých panelech.

[1] SoapUI https://www.soapui.org/

LineRouteDelay

Tato třída poskytuje jedinou metodu `GetLineRouteDelay` specifikovanou parametry, číslem linky a číslem trasy, která vrátí objekt s informacemi o aktuální poloze vozidla, aktuálním zpožděním, poslední zastávce a zda je vozidlo bezbariérové.

NearDeparturesService

Tato třída poskytuje pět metod, které na základě kombinací parametrů vrací odjezdy vozidel ze zastávky. Jedná se o metody:

- `GetNearDepartures` – má parametry: číslo zastávky a číslo sloupku (směr jízdy)
- `GetNearDeparturesWithCount` - má stejné parametry jako výše zmíněná metoda plus navíc počet záznamů, které má vrátit
- `GetNearDeparturesAcrossPoints` – má parametr: číslo zastávky
- `GetNearDeparturesByTime` - má parametry: číslo zastávky a číslo sloupku, hodinu a minutu
- `GetNearDeparturesByTimeWithCount` – má stejné parametry jako výše zmíněná metoda plus navíc počet záznamů, které má vrátit

Všechny zmíněné metody vrací informace zabalené v objektu, který obsahuje základní informace o odjezdu jako například: jméno cílové stanice, číslo linky, čas odjezdu, číslo sloupku a zda je vozidlo bezbariérové.

StopInfo

Tato třída poskytuje jedinou metodu `GetStopInfo` s parametrem číslo zastávky a vrací informace o zastávce (její pozice a jméno).

TrafficState

Tato třída reprezentuje veškerá dostupná vozidla, která jsou aktuálně v terénu. Poskytuje celkem tři bezparametrové metody:

- `GetTrafficState` – vrátí seznam všech dostupných vozidel MHD
- `GetTrafficStateDPMBWoService` – vrátí seznam všech služebních jízd
- `GetTrafficStateVLDBusService` – vrátí seznam vozidel ostatních dopravců v Jihomoravském kraji, jsou to například dálkové spoje

Všechny zmíněné metody vrací informace zabalené v objektu, který obsahuje základní informace o vozidle jako například jsou: číslo vozu, zpoždění, číslo cílové zastávky, číslo linky, číslo trasy, pozici vozidla v GPS koordinátorech a zda je vozidlo bezbariérové.

6.2 Popis souborů

V rámci spolupráce s centrálním dispečinkem Kordis Jihomoravského kraje byl také získan přístup k FTP serveru. Na serveru jsou uložena statická data. Jednotlivé soubory s daty budou popsány v následující podkapitole.

Soubor zastávky

Tento soubor obsahuje všechny zastávky v IDS JMK. Zastávka je jedno místo, které se v jízdním řádu stejně jmenuje. Na jednom místě může být více označníků (reprezentují směr jízdy). Struktura toho souboru je následující:

```
01001 101 'Achtelky' 'Achtelky' 'Achtelky' 'Achtelky' 'Achtelky'
// číslo zastávky, zóna, 5 variant názvů, používat 1. variantu
  S01 +099252582 +295112496 M 000 '>Kam.vrch'
  S02 +099260496 +295111644 S 000 '>Mysl,Pisárky'
  S03 +099240792 +295114782 M 000 '>Koh.od K.v.'
  S04 +099252582 +295112496 S 000 '>Koh.od Mys.'
// číslo označníku, koordináty, informační popis
```

Obrázek 6.1: Ukázka struktury souboru se zastávkami

Přepočet koordinátu
Jak je vidět ze struktury souboru, tak je poloha zadána ve vlastním formátu. Pro přepočet do GPS souřadnic slouží následující výpočet.
$X/100000 = P$, *X je část souřadnic (např. +099252582)*
$^\circ = P/60$ *(výsledek zaokrouhleně dolů př.: 16,98 na 16)*
$' = ((P/60) - ^\circ) * 60$ *(výsledek zaokrouhleně dolů př.: 16,98 na 16)*
$'' = ((((P/60) - ^\circ) * 60) - ') * 60$ *(výsledek zaokrouhleně dolů př.: 16,98 na 16)*
$''' = ((((((P/60) - ^\circ) * 60) - ') * 60) - '') * 60$ *(výsledek zaokrouhleně dolů př.: 16,98 na 16)*

Soubor služby

Tento soubor slouží pro definování služeb (spojů). V jednom dnu může být jenom jedna služba stejného čísla. Struktura toho souboru je následující:

```
#002004801 02 19 3 D
// uvození nové služby, číslo služby 002 = den, 04801 = kurz, oběh,
02 = Dopravce, 19 = Provozovna, 3 = druh dopravy, D = denní služba N
= noční služba - specifikuje, zda začíná v daném nebo následujícím
dni.
L048 C00002 032
//Linka 048, 00002=cíl, 032=číslo spoje
 Z01 16143 002 04:34 04:34 A MN NN PN
//zastávka Z01, číslo zastávky, číslo sloupku, příjezd, odjezd, A -
veřejný odjezd, MN - milník, NN - návaznost, PN
 Z02 16139 002 04:35 04:35 A MN NN PN
 Z03 16141 002 04:36 04:36 A MN NN PN
```

Obrázek 6.2: Ukázka struktury souboru se službami.

Další soubory

Dalšími dostupnými soubory byly soubor `PDDMMYY.txt` a soubor `navaznosti.txt`. Soubor `PDDMMYY.txt` obsahuje seznam služeb, které v daném dni jezdí, kde dd značí den, mm měsíc a yy rok. V souboru `navaznosti.txt` je definováno, které spoje navazují na konkrétní spoj v příslušné zastávce a jak dlouho mají čekat.

Kapitola 7

Návrh

Tato kapitola se zabývá návrhem výsledného řešení na základě požadavků specifikovaných v kapitole 2. Popisuje návrh společných prvků, mobilní aplikace a webových služeb.

Na tomto místě je vhodné zopakovat, že výsledná aplikace má sloužit jako mobilní asistent pro pohyb v MHD ve městě Brno. Uživateli bude poskytovat jakousi nápovědu, jak se dostat z bodu A do bodu B co nejrychleji.

7.1 Architektura

Architektura musí být navrhnuta tak, aby z pohledu mobilní aplikace poskytovala uživatelům čisté a intuitivní rozhraní a ze serverového pohledu definovala jasné rozhraní pro komunikaci.

Pro propojení aplikace a serveru byl model typu klient – server. Klientská část, aplikace, bude implementována pro mobilní operační systém Android, který byl představen v kapitole 4. Pro komunikaci se serverovou částí bude využita knihovna Retrofit, která výrazně zjednodušuje implementaci komunikace se serverovou částí. Serverová část bude implementována v jazyku Java a bude využit architektonický styl REST pro definování čistého a jednoznačného rozhraní pro komunikaci.

Obrázek 7.1: Schéma architektury řešení diplomové práce.

Obrázek 7.1 zobrazuje schéma architektury, která byla použita v rámci této diplomové práce. Pod pojmem webové služby si představme webovou službu poskytnutou v rámci spolupráce s dispečinkem Kordis Jihomoravského kraje. Serverová část bude implementována během této práce a bude využívat zmíněnou webovou službu. Klientem je v této práci mobilní aplikace běžící na systému Android.

7.2 Mobilní aplikace

Následující text diskutuje části návrhu týkajícího se výsledné mobilní aplikace. Tedy klientské části, která bude uživateli zobrazovat informace o jeho trase, odjezdech a poloze vozidel.

Případy užití

Jak již bylo zmíněno výše v kapitole 2, mobilní aplikace bude vyhledávat trasu a informovat uživatele o případné možnosti přestupu. Dále bude poskytovat informace o poloze vozidel MHD v Brně a také zobrazovat odjezdy vozidel ze zvolené zastávky. Pro lepší představu je vhodné výslednou funkcionalitu popsat pomocí případů užití:

1. Zadat / upravit údaje o spojení
2. Zadat / upravit rozšiřující informace o spojení
3. Zobrazit výsledné spojení
4. Zadat / upravit místo odjezdu
5. Zobrazit odjezdy
6. Zobrazit mapu s polohou vozidel
7. Zobrazit detail vozidla
8. Zobrazit trasu vozidla
9. Nastoupení do vozidla
10. Sledování trasy vozidla
11. Zobrazení nejbližších vozidel
12. Vyhledání zastávky

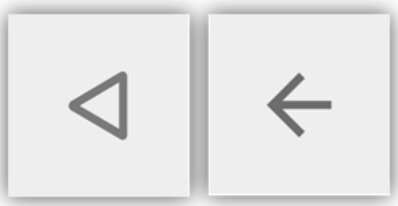
Obrázek 7.2: Diagram případů užití mobilní aplikace

Uživatelské rozhraní

Při návrhu uživatelského rozhrání je potřeba přijmout několik zásad, které je vhodné respektovat, jelikož uživatelé jsou již zvyklí na nějaký styl androidích aplikací. Uživatelské rozhraní musí umožnovat jednoduché, intuitivní a komfortní ovládaní celé aplikace. Proto vytvoření kvalitního uživatelského rozhraní, navigace a prezentace obsahu je velmi důležité dodržovat zásady, které budou následně popsány. Při návrhu uživatelského rozhraní aplikace bude vycházeno z vizuální koncepce Material Design[2], který byl přiblížen v kapitole 5.

Jak bylo zmíněno výše, navigace[3] je v rámci aplikace stěžejní částí, nejenže vede uživatele přes různé části aplikace, ale také organizuje obsah tak, aby bylo snadnější najít důležité místa v aplikaci. Material Design pro tvorbu kvalitní navigace poskytuje několik konceptů a návrhových vzorů, které budou popsány v následujícím textu.

- **Tlačítko nahoru a zpět (Button Up and Button Back)** – Tlačítko „Nahoru“ vrátí uživatele na předchozí obrazovku. Naviguje v hierarchii aplikace až do dosažení domovské obrazovky aplikace. Tlačítko „Zpět“ se pohybuje v obráceném chronologickém pořadí přes historii nedávno prohlížených obrazovek. Zatímco tlačítko „Nahoru“ zajišťuje, že uživatel zůstane ve vaší aplikaci, tlačítko „Zpět“ může vzít uživatele zpět na nedávné obrazovky mimo vaši aplikaci. Tyto tlačítka jsou základním prvkem uživatelské navigace v aplikaci. Příklad tlačítka nahoru a zpět zobrazuje obrázek 7.3.



Obrázek 7.3: Vpravo tlačítko "zpět"a vlevo tlačítko "nahoru[2].

- **Vestavěná navigace (Embedded navigation)** - Aplikace s jednoduchou navigací mohou zahrnout navigaci do obsahu aplikace, ale tím se zmenší dostupný prostor pro zobrazení tohoto obsahu. Vestavěná navigace je vhodná pro aplikace se silnou primární obrazovkou a minimem alternativních zobrazení a dále pro aplikace, které v hlavním zobrazení provádějí většinu úloh. Příklad vestavěné navigace zobrazuje obrázek 7.4.
- **Záložky a dolní navigační panel** (Tabs and Bottom navigation bar) - Záložky a dolní navigační panel umožňují uživatelům se rychle pohybovat mezi malým počtem stejně důležitých obrazovek. Tato navigace je vhodná pro aplikace s několika zobrazeními na nejvyšší úrovni a pro podporu budování povědomí o alternativních obrazovkách. Příklad vestavěné navigace zobrazuje obrázek 7.5.
- **Boční navigace (Navigation drawer)** - Pokud není k dispozici dostatek místa k podpoře záložek, boční navigace je dobrou alternativou. Boční navigace může zobrazovat mnoho navigačních cílů najednou. Navigace zůstává skrytá, dokud ji uživatel nevyvolá. Vhodné pro aplikace s mnoha obrazovkami na nejvyšší úrovni, rychlou navigaci mezi nesouvisejícími obrazovkami. Příklad vestavěné navigace zobrazuje obrázek 7.6.

Obrázek 7.4: Příklad aplikace s vestavěnou navigací. Běžné úkony jsou prováděny v hlavní obrazovce [2].

Návrh uživatelského rozhraní

Na základě výše popsaných návrhových vzoru pro tvorbu aplikace a na základě specifikace, účelu a případů užití byly identifikovány hlavní obrazovky aplikace a ty jsou:

- Obrazovka pro vyhledání odjezdů
- Obrazovka pro vyhledání spoje
- Obrazovka s mapou a polohou vozidel
- Obrazovka pro sledování trasy

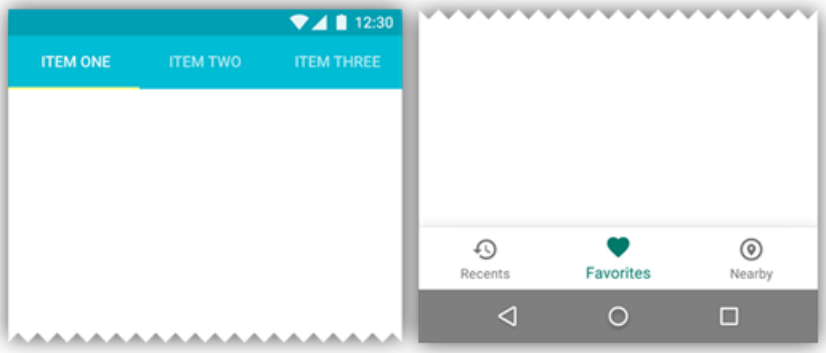
A také pomocné obrazovky jako:

- Obrazovka pro zobrazení výsledků
- Obrazovka pro vyhledání zastávky

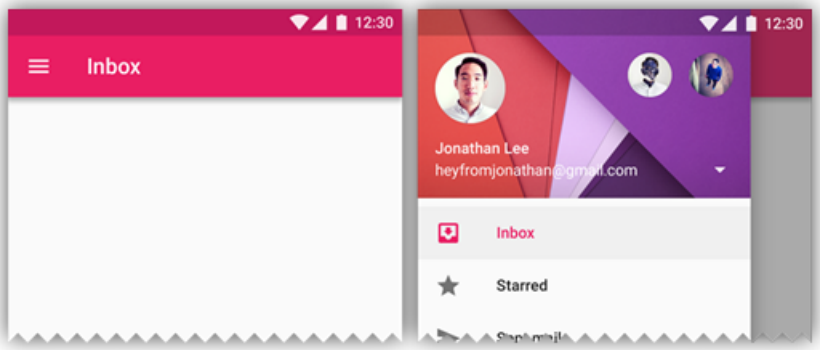
Pro lepší demonstraci bude využito mock-upových prototypů. Ve výrobě a designu je mock-up[19] měřítkem, plnohodnotným modelem designu nebo zařízení, používaného pro výuku, demonstraci, hodnocení návrhu, propagaci a jiné účely.

Jak bylo zmíněno, jedním z nejdůležitějších prvků aplikace je navigace. Z analýzy byly identifikovány čtyři hlavní obrazovky. V tom případě je vhodné pro navigaci použít panel se záložkami nebo dolní navigační panel. Obě tyto možnosti navigace poskytnou uživateli rychlou a přehlednou možnost přepínání mezi obrazovkami. Na obrázku 7.7 lze vidět návrh obou možností navigace.

Dalším důležitým úkolem při návrhu je ujasnění si toho, co chceme uživateli zobrazit. Zobrazení musí být jasné a jednotné, uživatel se musí v aplikaci zorientovat v co nejkratší



Obrázek 7.5: Vpravo příklad aplikace se třemi záložkami jako navigací. Vlevo příklad aplikace s dolním navigačním panelem [2].



Obrázek 7.6: Příklad aplikace se skrytou a zobrazenou boční navigací [2].

době. Na obrázku 7.8 je návrh hlavních obrazovek pro vyhledání odjezdu a spojení. Obrazovka pro vyhledávání odjezdů je velmi jednoduchá, poskytuje uživateli pole pro výběr odjezdové zastávky, filtr zda chce zobrazit odjezdy podle jízdního řádu nebo podle aktuálních vozidel v terénu a tlačítko pro vyhledávání. Obrazovka pro vyhledávání spojení je koncipována podobně jako obrazovka pro vyhledávání odjezdů, jen je rozšířená o pole pro výběr cílové zastávky a také o vysouvací nabídku pro rozšíření parametrů vyhledávání. Rozšířená nabídka poskytuje uživateli možnost si vybrat přes jakou zastávku má spoj jet, výběr vozidel, zda hledá pouze přímé spojení a zda vozidla mají být bezbariérová. Po vybrání některého z rozšířených parametrů bude uživateli dáno na vědomí, že je vybrána specifikace vyhledávání a také mu bude zobrazena možnost rychlého smazání rozšířených parametrů.

Obrázek 7.9 zobrazuje návrh obrazovky s mapou a polohou vozidel. Dále návrh obrazovky pro zobrazení nejbližších vozidel a módu jízdy. Obrazovka s mapou bude zobrazovat klasickou Google mapu s ikonami jednotlivých vozidel, při kliku na vozidlo bude zobrazeno okno s informacemi o vozidle a vykreslena jeho trasa. Obrazovka s nejbližšími vozidly bude zobrazovat seznam nejbližších vozidel a po nastoupení do vozidla se zobrazí obrazovka, na které bude naznačena trasa spoje (předchozí, aktuální a následující zastávka) a seznam spojů odjíždějících z následující zastávky.

Obrázek 7.7: Návrh hlavní navigace v aplikaci. Vlevo ukázka navigace se záložkami v horní části obrazovky. Vpravo ukázka navigace s dolním navigačním panelem.

Obrázek 7.8: Návrh obrazovek pro odjezd a spojení. Vlevo je obrazovka pro vyhledání spojení s rozbalenou nabídkou pro rozšířené zadání. Vpravo je obrazovka pro vyhledání odjezdů ze zvolené zastávky.

7.3 Serverová část

V rámci této práce je nutné implementovat i serverovou část. Definovat její rozhraní, aby byla možná komunikace serveru s klienty. Na serveru bude probíhat výpočet spojení, vyhle-

Obrázek 7.9: Návrh obrazovek. Vlevo obrazovka s mapou a polohou vozidel. Uprostřed obrazovka s nejbližšími vozidly s možností nastoupení. Vpravo obrazovka pro režim jízdy se sledováním trasy a možnými přestupy.

dání odjezdů, zastávek vozidel a to díky využití klienta vygenerovaného na základě WSDL popisu poskytnutého od dispečinku Kordis Jihomoravského kraje.

Pro návrh rozhraní webové služby se v dnešní době ve velké míře využívá architektonický styl REST, který byl popsán v kapitole 4. V architektuře REST se rozhraní definuje pomocí zdrojů, které jsou identifikovány pomocí URI adresy a manipulace s nimi probíhá pomocí protokolu HTTP (pomocí metod GET, POST, PUT, DELETE).

Serverová část v této práci bude implementována v jazyku Java a využitím architektonického stylu REST. Na základě specifikace a analýzy je zřejmé, že serverová část bude poskytovat pouze data, proto bude použita jen metoda GET z protokolu HTTP. Bude tedy možné implementovat požadovanou webovou službu tím nejjednodušším způsobem a to například pomocí Java Servletů.

Návrh API

Serverová část na základě specifikace a analýzy požadavků musí minimálně podporovat následující operace, aby splnila požadovanou funkčnost. Na zřetel byly vzaty metody vygenerované na základě WSDL popisu. Tyto metody a získaná statická data jsou blíže popsána v kapitole 6.

- **Získání všech pozic zastávek** – k implementaci bude použit soubor se zastávkami, tento soubor se bude muset rozparsovat a získat z něho potřebné informace
- **Získání polohy vozidel MHD** – k implementaci bude použita některá z metod třídy `TrafficState`, která uchovává informace o pohybu veškerých vozidel
- **Získání informací o trase** - k implementaci bude vhodné použít třídu `TrafficState` a soubor s polohou zastávek

- **Získání informací o vozidle** – k implementaci bude vhodné použít třídu `TrafficState`
- **Získání informací o pozici nejbližších vozidel MHD** – k implementaci bude vhodné použít třídu `TrafficState` a aktuální pozici klieta. Na základě tohoto provést výpočet.
- **Získání odjezdů** - k implementaci bude vhodné použít třídu `NearDeparturesService`, která uchovává informace o odjezdech vozidel
- **Získání spojení** – k implementaci bude vhodné použít soubor se službami a třídu `TrafficState`, která vrací informace o poloze vozidel.

Kapitola 8

Implementace

V této kapitole je popsána implementace mobilní aplikace a serverové části. Implementace vychází ze specifikace požadavků popsané v kapitole 2 a z návrhu provedeného v kapitole 7.

Implementace probíhala v několika iteracích. Po každé iteraci byl hotov funkční prototyp mobilní aplikace a serverové části. Každý prototyp se zaměřoval na jinou část výsledné práce a postupně na sebe nabaloval funkce z předešlé iterace. Po každé iteraci byl prototyp předán k otestování. Na základě zpětné vazby byla mobilní aplikace a serverová část vylepšována. Prvotní iterace budou popsány stručně, bude v nich vysvětleno na jakou část byly zaměřeny, jaké možnosti implementace pro realizování této funkčnosti jsou k dispozici a na závěr každé iterace bude hotový prototyp předán několika testerům. Nakonec bude podrobně popsána finální implementace.

8.1 1. Iterace

První iterace se zaměřila na implementaci mapy a zobrazení aktuální polohy vozidel MHD v Brně. Zobrazení a získání aktuální polohy vozidel je základní část celé práce. Tato část byla proto zpracována hned v první iteraci, aby se odhalila případná rizika co nejdříve.

8.1.1 Serverová část

Serverová část v rámci této iterace musí poskytovat data s aktuální polohou všech vozidel. Na začátku byl na základě WSDL popisu (viz. kapitola 4), pomocí vývojového prostředí Eclipse, vygenerován webový klient, který zpřístupňuje metody poskytnuté centrálním dispečinkem Kordis Jihomoravského kraje. Poté byla implementována třída, která volá metody třídy `TrafficState` (viz. kapitola 6), které vrací objet s daty o poloze vozidel. Následně je tato třída instanciována v Java Servletu, kde je použita pro generování JSON modelu.

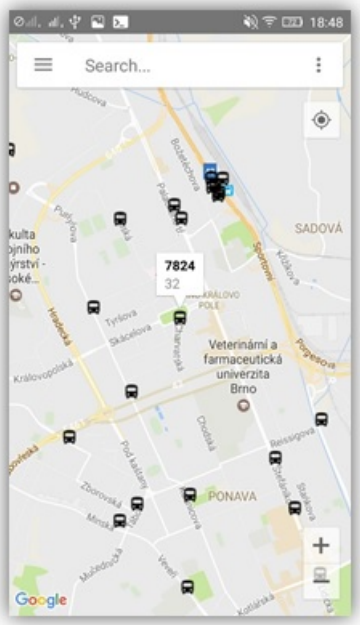
8.1.2 Mobilní aplikace

Mobilní aplikace v této iteraci, jak bylo zmíněno v úvodu, byla zaměřena na implementaci mapy a zobrazení aktuální polohy vozidel. Nutné bylo implementovat zobrazení mapy, REST rozhraní pro komunikaci s webovou službou a následné zobrazení aktuální pozice vozidla.

Společnost Google poskytuje API rozhraní pro vytvoření plnohodnotné aplikace s mapovými podklady. Google Maps API je dostupné prostřednictvím Google Play služeb. Pro

zprovoznění Google map je potřeba si nejprve vygenerovat klíč [1], který vložíme do aplikace. Poté byl implementován fragment s mapou.

Pomocí vývojového prostředí Android Studio byl vytvořen prototyp aplikace implementující výše zmíněné požadavky.



Obrázek 8.1: Ukázka mobilní aplikace po dokončení první iterace. Implementována je Google mapa a zobrazeny jsou aktuální polohy vozidel, které se automaticky obnovují.

8.1.3 Testování prototypu

Po dokončení prototypu jsem výsledek práce ověřoval osobně, protože pro testování testery nebyla implementována dostatečná funkčnost. Testování bylo zaměřeno na ovládání mapy a správnosti polohy vozidel. Dalším důležitým faktorem k otestování byla správná aktualizace polohy vozidel.

8.2 2. Iterace

V druhé iteraci byla aplikace rozšířena o výběr zastávky, vyhledávání odjezdů a spojení na základě počáteční a cílové zastávky. Vyhledávání spojení bude možné specifikovat pomocí rozbalovací nabídky s možnostmi jako specifikování průjezdového bodu, výběru vozidel MHD, apod. V neposlední řadě byla implementována navigace celé aplikace. Webové služby byly rozšířeny tak, aby vracely odjezdy vozidel a požadované spojení na základě zastávek.

8.2.1 Serverová část

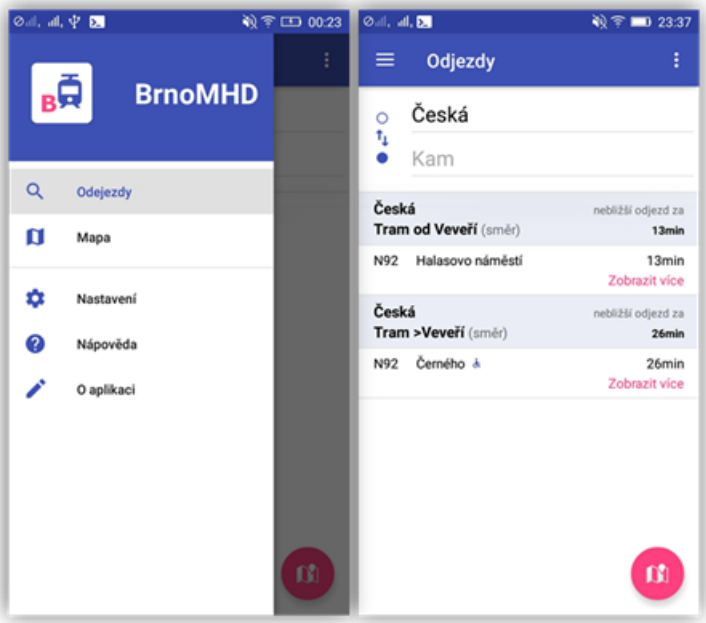
Serverová část v této iteraci byla rozšířena o vyhledání odjezdů vozidel na základě parametru s číslem zastávky. Vyhledávání odjezdů je rozděleno na dva typy: vyhledávání podle jízdního

[1] Google Map Key https://developers.google.com/maps/documentation/android-api/signup

řádu a vyhledávání podle vozidel v terénu. Ke zpracování odjezdů podle jízdního řádu byly využity metody třídy `NearDeparturesService` a výsledek zpracován v Java Servletu do JSON modelu.

8.2.2 Mobilní aplikace

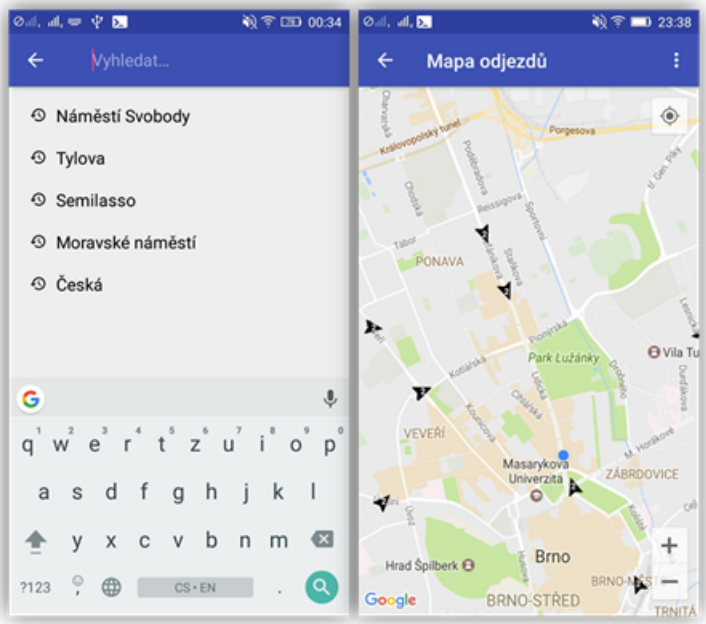
Mobilní aplikace byla rozšířena o jednu z nejdůležitějších částí a tou je navigace. V této iteraci byla, jako navigace, implementována boční navigace (viz. kapitola 7.2). Po jejím zobrazení se uživateli ukáže seznam možností pro přepínání mezi hlavními obrazovkami. Hlavní obrazovka je složená ze dvou částí. V horní část je uzpůsobena pro vyhledávání odjezdů a spojení. Zbytek obrazovky je rezervován pro zobrazení výsledků. Pro vyhledávání zastávky je implementována další obrazovka, která má v horním panelu implementované vyhledávací pole. Při psaní jsou uživateli zobrazovány zastávky začínající na zadané znaky. Než uživatel začne psát je mu zobrazena historie posledních pěti vyhledávaných zastávek, po vybrání zastávky je uživatel vrácen zpět na hlavní obrazovku. V případě, že je zadána pouze odjezdová stanice jsou zobrazeny jen odjezdy. Pokud je zadána pouze cílová stanice jsou z nejbližších zastávek vybrány odjezdy vozidel jedoucí tímto směrem. Při zadání počáteční a cílové stanice je klasicky vyhledán spoj mezi těmito zastávkami. Vpravo dole je umístěno plovoucí tlačítko pro rychlé zobrazení mapy. Obrazovka s mapou je rozšířena o ikony vozidel zobrazující směr vozidla a při vybrání vozidla je vykreslena jeho trasa a informační okno s dalšími užitečnými informacemi o něm.



Obrázek 8.2: Ukázka mobilní aplikace po dokončení druhé iterace. Vlevo je zobrazena rozbalená boční navigace. Vpravo je náhled na hlavní obrazovku s odjezdy ze zastávky Česká.

8.2.3 Testování prototypu

Pro testování druhého prototypu bylo pozváno dohromady 6 testerů. Ve věku od 22 do 50 let s různou zkušeností s užíváním mobilního zařízení. Test prototypu probíhal podle ná-



Obrázek 8.3: Ukázka mobilní aplikace po dokončení druhé iterace. Vlevo je zobrazena obrazovka pro vyhledávání, je na ní zobrazena i historie posledních vyhledávání. Vpravo je zobrazena mapa s polohou vozidla a ikonami podle směru vozidla.

sledujícího scénáře. Uživateli byl představen koncept aplikace a implementovaná funkčnost v prototypu. Poté mu bylo zadáno několik úkolů, které vycházely z hlavních případů užití a zaměření prototypu. Uživatel se nejdříve úkol pokusil splnit samostatně. Pokud se mu nedařilo splnit zadaný úkol, byla poskytnuta nápověda. Po dokončení zadaných úkolů uživatel vyplnil dotazník. Testování tohoto prototypu bylo především zaměřeno na následující úlohy:

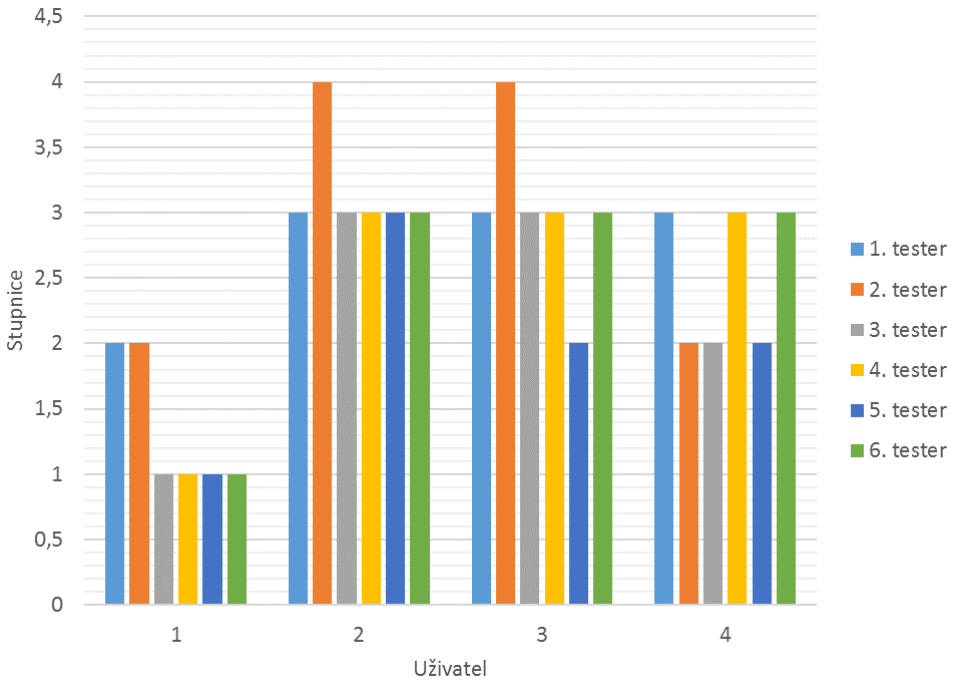
- Vyhledání odjezdů
- Vyhledání spoje
- Zobrazení mapy
- Otestování navigace v aplikaci

Závěrečný dotazník byl složen z následujících otázek, se stupnicí od jedné (nejlepší) do pěti (nejhorší) plus polem pro krátký komentář.

- Obtížnost vyhledávání – tento dotaz byl zaměřen na validaci vyhledávací obrazovky a relevantnost zobrazených dat
- Přehlednost vozidel na mapě – zaměření tohoto dotazu bylo na velikost ikon, množství zobrazovaných vozidel
- Navigace v aplikaci – cílem tohoto dotazu byla verifikace navrhnuté uživatelské navigace

- Informační hodnota – tento dotaz se zaměřoval na přehlednost a relevantnost zobrazených výsledků vyhledávání

Výsledky dotazníku byly zpracovaný a zobrazeny v následujícím grafu 8.4.



Obrázek 8.4: Graf zobrazující výsledky dotazníku k testování druhé iterace vývoje. Na ose x jsou zobrazeny otázky dotazníku a na ose y jsou vyneseny známky.

Z grafu 8.4 je zřejmé, že na první otázku „Obtížnost vyhledávání“ testeři odpovídali velmi kladně, čtyři ze šesti testerů ohodnotili vyhledávání známkou jedna, tedy nejvyšším možným. Jedna z jejich reakcí na vyhledávání bylo, že je přehledné, jednoduché a snadno použitelné.

Druhá otázka „Přehlednost vozidel na mapě“ dopadla o poznání hůře, pět ze šesti testerů jí hodnotilo známkou tři, jeden dokonce známkou čtyři. Většina námitek se týkala množství zobrazovaných vozidel na mapě.

Ve třetí otázce testeři hodnotili navigaci v aplikaci. Průměr známek se ustálil na čísle tři, tedy spíše podprůměru. Většina kritiky se týkala toho, že pro zobrazení navigace muselo být stlačeno tlačítko v horním panelu. Navigace by měla být viditelná hned.

Poslední, čtvrtá, otázka byla zaměřená na informativní hodnotu zobrazených výsledků po vyhledání. Polovina testerů tento úkol oznámkovala číslem dva a druhá půlka číslem tři. V komentáři bylo většinou uvedeno, že zobrazené výsledky jsou dostatečně informativní. Jen horní vyhledávací panel je zbytečně neustále zobrazen.

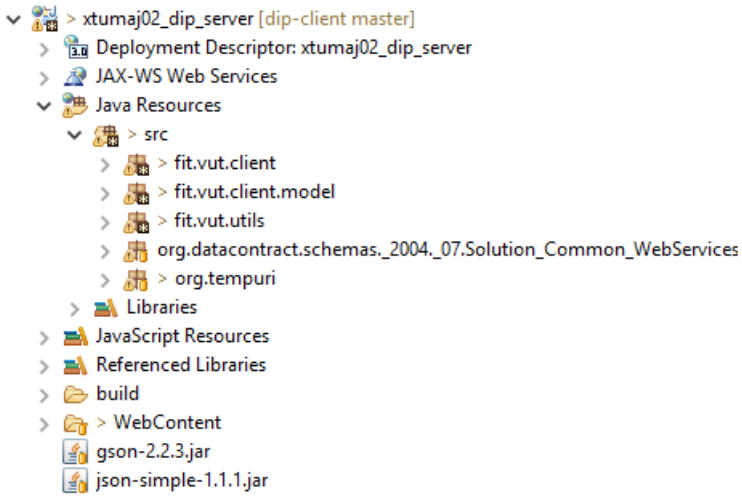
Z výsledků dotazníku je jasné, že musí být celkově přepracována navigace v aplikaci, hlavní obrazovka s výsledky a zobrazováno méně vozidel na mapě.

8.3 Finální implementace

Tato kapitola popisuje finální implementaci této diplomové práce. Při zhotovení finální verze byly vzaty na zřetel prvotní prototypy a výsledky jejich testování. Finální verze již poskytuje veškerou funkcionalitu.

8.3.1 Serverová část

Stejně jako prvotní prototypy byl pro tvorbu finální serverové části použit jazyk Java a jeho Servlety. Finální serverová část byla umístěna na školní server pcuifs2, na kterém běží Apache Tomcat verze 7.0.56 a Java verze 7. Z toho serveru byla dostupná webová služba od dispečinku Kordis.



Obrázek 8.5: Náhled na strukturu projektu pro serverovou část.

Obrázek 8.5 zokbrazuje strukturu finální serverové části. Její součásti jsou následující:

- Složka `fit.vut.client` - obsahuje třídy pro komunikaci s mobilní aplikací. Většina jich je implementována jako Java Servlet a vrací data zabalená v modelových třídách ve formátu JSON.
- Složba `fit.vut.client.model` obsahuje veškeré modelové třídy serverové části
- Složky `org.data.contact.schemas` a `org.tempuri` obsahují třídy vygenerované z WSDL popisu sloužící pro komunikaci s webovou službou Kordis.

Než budou popsány hlavní části implementace serverové části, je třeba nejdříve popsat nejdůležitější modelové třídy. Informace o zastávkách pro vyhledávání v mobilní aplikaci uchovává třída `StationSuggestion`. Obsahuje parametry jako jméno a číslo zastávky. Třída `DepartureItem` reprezentuje jeden záznam odjezdu nebo spojení, obsahuje několik parametrů relevantních pro odjezd jako například jméno a identifikátor počáteční zastávky, čas odjezdu, číslo linky a další. `VehicleInfo` je třída, která uchovává informace o vozidle, obsahuje informace jako číslo vozidla, číslo trasy, GPS koordináty a další. `RouteWrapper` je obalující třída pro informace o trase vozidla a obsahuje list s objekty třídy `RouteItem`, identifikátor linky a číslo poslední známe zastávky. Další modelová třída `DriveMode` uchovává

data pro režim jízdy v mobilní aplikaci, obsahuje tedy informace jako předešlá, aktuální a následující zastávka, GPS pozici vozidla, číslo vozidla a další.

Hlavní třída se jmenuje `Client` a implementuje metody, které volají metody komunikující s webovou službou poskytnutou v rámci spolupráce s dispečinkem Kordis Jihomoravského kraje. Obsahuje metody jako například je `getVehicleMap`, která vrátí hashmapu s objekty třídy `Vehicle`, tedy se všemi vozidly. Dále obsahuje metody `findNearestStops`, pro nalezení nejbližších zastávek. A metodu `getNearDepartures`, která vrací list s odjezdy. Třída `Departure` je rozšířením třídy `HttpServlet` a implementuje metody pro získání odjezdů nebo spojení mezi zastávkami. Nejprve je v metodě `init` volána metoda `getRouteFromFile`, která získá trasu vozidla, a poté metoda `getStationsFromFile` pro získání hashmapy zastávek. Později v metodě `doGet` jsou procházeny a filtrovány trasy na základě vozidel a zastávek. V rámci této třídy probíhá vyhledávání optimální trasy spojení, které je počítáno následovně. Na začátku jsou zadaný parametry jako počáteční a cílová zastávka. Poté je načtena hashmapa s trasami vozidel a list s vozidly. List vozidel je procházen a pro každé vozidlo je získána jeho trasa. Tato trasa je testována jestli projíždí počáteční a cílovou stanicí. Na zřetel je brán směr, přestup a další parametry. Pokud trasa odpovídá požadovaným parametrům je sestaven objekt `DepartureItem` a přidán do listu. Nakonec je nad tímto listem provedeno řazení na základě času odjezdu a celkové doby jízdy. Následně je výsledek tohoto vyhledávání převeden pomocí knihovny Gson do formátu JSON.

Třída `NearestVehicle` také rozšiřuje třídu `HttpServlet` a slouží k vracení všech vozidel, která jsou nejblíže zadané polohy. Nejdříve je získána instance třídy `Client` a poté je v metodě `doGet` volána metoda `findNearest`, třídy `Client`, pro získání nejbližších vozidel. Implementace metody `findNearest` spočívá ve volání metody `getTrafficState`, webové služby, která vrátí list všech vozidel. Nad těmito vozidly je vypočítána vzdálenost od zadaných GPS koordinátů a nakonec je celý tento list seřazen podle vzdálenosti. Nakonec je výsledek převeden do formátu JSON pomocí knihovny Gson.

Třída `StopSearch` rozšiřuje třídu `HttpServlet` a na základě parametru, reprezentující vyhledávaný řetězec, vrací seznam zastávek, které začínají na tento parametr. V metodě `doGet` je nejdříve načten a parsován soubor se zastávkami a poté je nad tímto seznamem provedeno vyhledávání. Výsledek je poté pomocí knihovny Gson převeden do formátu JSON.

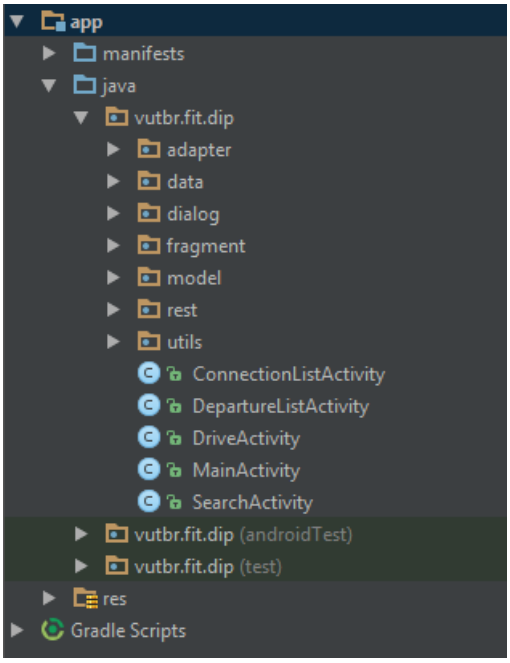
Třída `Route` také rozšiřuje třídu `HttpServlet` a slouží k získání trasy na základě parametru s číslem trasy, linky, počáteční a koncové zastávky. Nejdříve je rozparsován soubor `DPMD.txt` pospaný v kapitole 6 do hashmapy. Mapa má jako klíč číslo trasy a jako hodnotu objekt `RouteWrapper`, reprezentující trasu vozidla. Na základě vstupního parametru je poté pomocí metody get nad mapou s trasami provedeno získání požadované trasy. Nakonec je výsledek převeden to formátu JSON.

8.3.2 Mobilní aplikace

Finální mobilní aplikace byla vytvořena pro minimální verzi Android API 15, odpovídající verzi operačního systému 4.0.3 IceCreamSandwich, díky čemuž bude dostupná na více než 90% zařízení. Mobilní aplikace kopíruje standardní strukturu Gradle projektu.

Obrázek 8.6 ukazuje strukturu projektu výsledné mobilní aplikace. Její součásti jsou následující:

- Složka manifest – obsahuje soubor manifest, ve kterém jsou uvedeny definice všech komponent aplikace. Indikuje také, která aktivita je hlavní a deklaruje oprávnění, které aplikace vyžaduje.



Obrázek 8.6: Náhled na strukturu projektu výsledné mobilní aplikace.

- Složka java – obsahuje podsložky s veškerými třídami použitými v aplikaci, zmíním třeba třídu MainActivity, která je startovací třídou celé aplikace.
- Složka res – obsahuje veškeré externí zdroje jako jsou obrázky, XML definice vzhledu obrazovek, textu, definice stylu a podobně.
- Složka Gradle – obsahuje soubory se závislostmi použitými v aplikaci, jako jsou například externí knihovny.

Při implementaci aplikace byly využity následující externí knihovny a služby:

- `appcompat-v7`[2] - pro zpětnou kompatibilitu vzhledu a fungování prvků uživatelského rozhraní
- `recyclerview-v7`[3] – obsahuje prvek uživatelského rozhraní RecyclerView, který slouží pro zobrazování seznamů (nahrazuje dříve používaný ListView)
- `design support library`[4] – knihovna pro podporu prvků Material design
- `gson`[5] – knihovna pro práci s JSON objekty, usnadňuje jejich transformace a konverzi na Java objekty
- `retrofit`[6] – knihovna pro síťovou komunikaci
- `material-dialogs`[7] – knihovna usnadňující práci s dialogy

[2] appcompat-v7 https://developer.android.com/topic/libraries/support-library/features.html
[3] recyclerview-v7 https://developer.android.com/topic/libraries/support-library/features.html
[4] design support library https://developer.android.com/topic/libraries/support-library/features.html
[5] gson https://github.com/google/gson
[6] retrofit http://square.github.io/retrofit/
[7] material-dialogs https://github.com/afollestad/material-dialogs

- `smoothprogressbar`[8] – knihovna pro zobrazení progres linky pro indikaci načítání
- `play-services`[9] – služby pro získávání aktuální polohy mobilního zařízení

Výsledná aplikace pro svoji funkci dále potřebuje povolení (permission viz. [4]) následujících funkcí operačního systému:

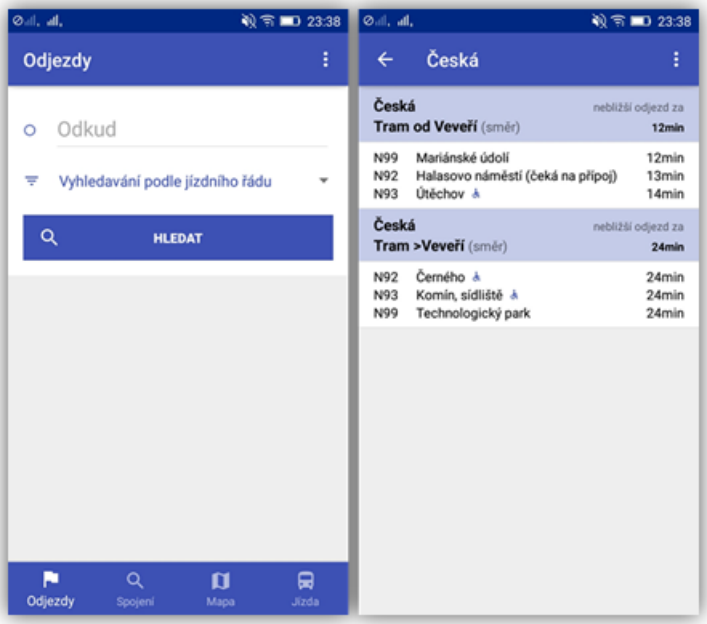
- `android.permission.ACCESS_FINE_LOCATION` - umožňuje aplikaci přístup k přesné poloze.
- `android.permission.ACCESS_COARSE_LOCATION` - umožňuje aplikaci přístup k přibližné poloze.
- `android.permission.INTERNET` - umožňuje aplikacím otevřít síťové spojení pomocí socketů.

Než budou vysvětleny další části aplikace, bude vhodné nejprve popsat implementaci modelových tříd a tříd obstarávajících komunikaci se serverovou částí, které jsou podobné jako v serverové části 8.3.1. Třída obsahující model pro odjezdy a spojení se jmenuje `DeparturesWrapper`. Třída `DeparutureItem` reprezentuje jeden záznam odjezdu nebo spojení. `NearestVehicle` je modelová třída uchovávající informace o vozidle. `RouteWrapper` je obalující třída pro informace o trase vozidla. `RouteItem` obsahuje informace o jedné položce trasy. Další modelová třída `DriveMode` uchovává data pro mód jízdy. Poslední představenou třídou bude třída `StopSuggestion`, která uchovává informace o posledních zastávkách.

Ke komunikaci se serverovou částí byla využita knihovna Retrofit, její implementace se nachází ve třídě `ApiClient`, která implementuje rozhraní `ApiInterface`. Ve třídě je volána třída `RetrofitBulider`, která volá definovanou adresu serverové části. Dále je nutné specifikovat továrnu pro deserializaci odpovědi ze serveru, v našem případě byla použita knihovna Gson. Metody pro obsluhu webové služby jsou definovány uvnitř rozhraní pomocí speciálních anotací knihovny Retrofit. Tyto anotace slouží pro dekódování podrobností o parametrech a způsobu volání požadavku. Návratová hodnota je vždy parametrizována Call<T> objektem, kde T je libovolná modelová třída.

Základem celé aplikace je třída `MainActivity`, které je spuštěna ihned po startu aplikace. Hlavní funkcí této třídy je tvorba navigace v celé aplikaci, stará se tedy o zobrazování různých obrazovek aplikace. Třída `MainActivity` implementuje funkcionalitu pro BottomNavigationView 7.2, spodní navigační panel, tudíž pro přepínání mezi obrazovkami aplikace, které jsou implementovány jako fragmenty. `BottomNavigationView` musí být nejdříve definován v XML souboru s definicí vzhledu, v tak zvaném layout souboru. Tento XML soubor také obsahuje view group prvek pro identifikaci kam se mají fragmenty načítat. Pro změnu fragmentu, obrazovky, byl použit `FragmentManager` a třída `FragmentTransaction`, která se stará o operace související s fragmenty. Nový fragment se přidá do specifikované view group na základě volání metody `beginTransaction`, která vytvoří novu transakci pro manipulaci s fragmentem. Aplikace obsahuje celkem čtyři hlavní fragmenty, které zastupují hlavní obrazovky aplikace. Jsou to tyto fragmenty: `DepartureFragment`, `ConnectionFragment`, `MapFragment` a `DriveFragment`. Fragment je normální třída implementující rozšíření třídy `Fragment`, to poskytne implementující třídě aplikační rozhraní třídy `Fragment`. Fragment je podobný komponentě `Activity`, také má vlastní životní cyklus, ale trochu odlišný od životního cyklu `Activity`.

[8] smoothprogressbar https://github.com/castorflex/SmoothProgressBar
[9] play-services https://developers.google.com/android/guides/overview



Obrázek 8.7: Vlevo je zobrazen náhled na navigaci aplikace a na první fragment, který je zobrazen po startu aplikace. Vpravo je zobrazena obrazovka s odjezdy ze zastávky Česká.

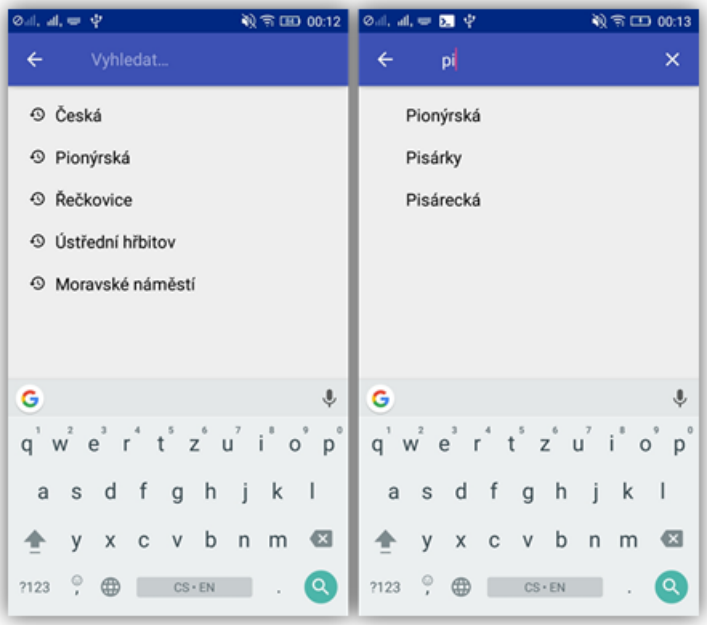
První zobrazený fragment po startu aplikace je `DeparturesFragment`, reprezentující obrazovku pro vyhledávání odjezdů. Jeho layout soubor obsahuje jen textové pole pro výběr zastávky, vysouvací menu s výběrem možností mezi vyhledávání podle klasického jízdního řádu nebo podle aktuálních vozidel v terénu. Dále obsahuje tlačítko pro začátek vyhledávání. Výběr zastávky probíhá tak, že při kliku na textové pole reprezentující zastávku se zavolá, pomocí `Intentu`, třída `SearchActivity`. Po návratu z vyhledávací třídy jsou v přetížené metodě `onActivityResult`, naplněny lokální proměnné. Vrácená data nesou informace o číslu zastávky a jejím jménu. Poté stiskem tlačítka dojde k zavolání třídy `DepartureListActivity` a zobrazení obrazovky se seznamem odjezdů. Pokud není vybraná žádná zastávka a uživatel stiskne tlačítko hledat, dojde k zobrazení dialogového okna, třídy `MaterialDialog`, s varováním.

Třída `DepartureListActivity` se stará o načítání dat o odjezdech a jejich zobrazování. Na začátku v metodě `onCreate` jsou z `Intentu` pomocí metody `getExtras` získány hodnoty parametrů pro volání serverové části. V metodě `onCreate` jsou také inicializovány všechny potřebné elementy pro zobrazení výsledků, konkrétně `RecycleView`. Na základě typu vyhledávání je volána metoda `getDepartures`, pro vrácení odjezdů vozidel na základě jízdního řádu, nebo je volána metoda `getDeparturesByVehicles` pro získání odjezdů vozidel na základě jejich aktuálních pozic. Vyplnění `RecycleView`, seznamu výsledků, zajišťuje třída `DepartureAdapter`, která je implementací třídy `RecyclerView.Adapter`. Pro neustálou aktualizaci výsledků jsou metody pro získání odjezdů každých 10 sekund volány v separátním vláknu. K tomu je využito rozhraní `Runnable` a jeho metoda `run`.

Třída `SearchActivity` slouží pro vyhledávání zastávek. Vyhledávácí pole je umístěno v horním panelu. Než uživatel začne hledat zastávku je mu zobrazena historie posledních pěti vyhledávání. Při vyhledání zastávky je uživateli dynamicky zobrazována nabídka zastávek, která odpovídá vyhledávanému textu. Vyhledávání funguje na základě komu-

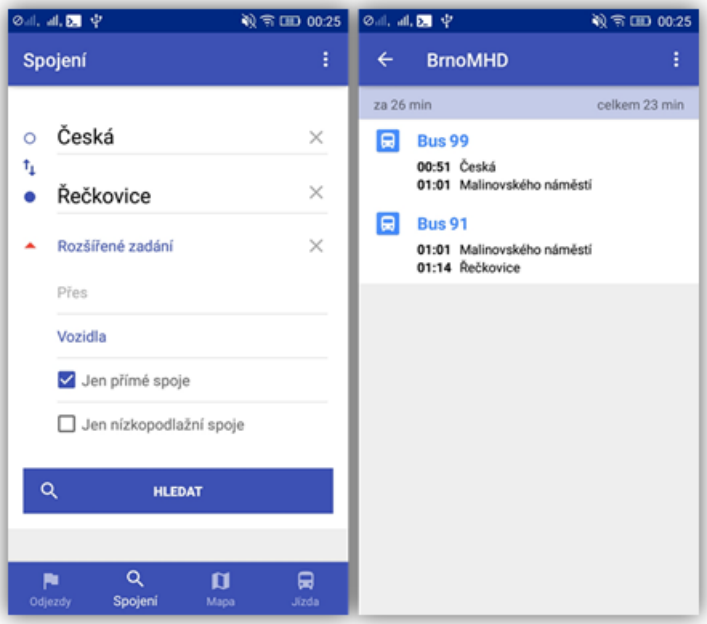
nikace se serverovou částí. Využita je k tomu třída `ApiClient` a konkrétně její metoda `getStopsSuggestions`, která vrátí list s objekty třídy `StopSuggestion`. Získaný list je poté vložen do `RecycleView` pro zobrazení výsledků. Jejich přímé zobrazení obsluhuje třída `SearchAdapter`, jenž je implementací třídy `ArrayAdapter`. Tato třída pomocí metody `getView` nastaví grafickou podobu řádku seznamu a naplní je daty. Při jakékoli změně vyhledávaného řetězce dojde k aktualizaci seznamu s využitím zmíněného principu. Po výběru z nalezených zastávek dojde k návratu do fragmentu, který tuto vyhledávací třídu volal. Při návratu jsou do `Intentu` vloženy pomocí metody `putExtra` informace o vybrané zastávce, jako jsou její číslo a jméno.

Historie vyhledávání je implementována ve třídě `SuggestionHistory`, která poskytuje dvě metody `writeHistory` pro uložení a `readHistory` pro čtení. Třída `SuggestionHistory` využívá pro ukládání dat třídu `SharedPreferences`, tato třída umožňuje uložení privátních dat ve formátu klíč – hodnota. Reference na třídu `SharedPreferences` byla získána pomocí metody `Context.getSharedPreferences`. Pro čtení / zápis byly využity metody `getString() / putString()`. Jelikož do `SharedPreferences` lze uložit jen primitivní typy jako `string, integer, boolean` apod a uživateli je potřeba zobrazit minimálně pět záznamů posledního vyhledávání, tak aby byl list s jednotlivými objekty historie převeden pomocí knihovny gson do formátu JSON. Potom je tento formát převeden do stringu a následně uložen do `SharedPreferences`. Metoda `readHistory` ze `SharedPreferences` načte string reprezentující pole s historií ve formátu JSON a s pomocí knihovny gson tento string převede na list objektů třídy `StopSuggestion`. Metoda `whiteHistory` nejdříve zavolá metodu `readHistory` pro načtení listu s historií a pokud list již neobsahuje stejnou zastávku dojde vložení záznamu na první pozici a polední pozice v listu je smazána, pokud list obsahuje více než pět záznamů.



Obrázek 8.8: Vlevo je zobrazen náhled na vyhledávací obrazovku s historií posledního hledání. Vpravo je zobrazen náhled s vyhledáváním řetezce "pi".

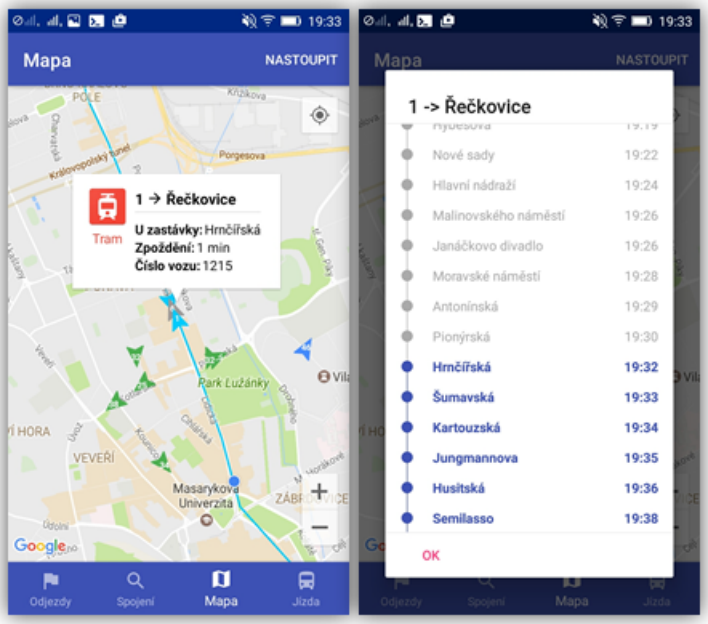
Fragment `ConnectionFragment` reprezentuje obrazovku pro vyhledávání spojení. Jeho layout soubor obsahuje textová pole pro zadání počáteční a cílové zastávky, potom tlačítko, které rozbaluje nebo zabaluje nabídku s dalšími možnostmi pro specifikaci vyhledávaného spojení. Při kliku na pole pro výběr zastávky dojde s pomocí `Intentu` k volání třídy `SeachActivity` pro vyhledávání zastávek. Rozšířená nabídka obsahuje možnost pro zadání průjezdného místa, volby vozidel (tramvaj, autobus a trolejbus) a dva chechboxy pro výběr zda vozidlo má být bezbariérové a zda se mají vyhledávat pouze přímé spoje. Při jakékoliv změně v rámci této rozšířené nabídky výhledávání dojde zobrazení tlačítka pro vymazání zadané specifikace a to hned vedle upraveného pole a také globálně u tlačítka pro skrytí rozšířené nabídky. Dále je změněna ikona indikující rozbalenou a skrytou rozšířenou nabídku na červenou barvu. Indikace změny je implementována pomoci privátní globální proměnné, která obsahuje hodnotu true nebo false na základě stavu specifikujících parametrů je volána metoda `updateShowElementsButtons`. Nakonec po stisku tlačítka vyhledat se aplikace přepne do třídy `ConnectionListActivity`. Tato třída se stará o načítání dat o spojeních a jejich zobrazování. Pracuje podobně jako třída `DepartureListActivity`, která je určená pro odjezdy. Vykreslení výsledků zajišťuje třída `DepartureAdapter`, která je implementací třídy `RecyclerView.Adapter`. Spojení je každých deset sekund aktualizováno.



Obrázek 8.9: Vlevo je zobrazen náhled obrazovky pro vyhledávání spojení. Vpravo je zobrazen náhled na obrazovku s výsledky vyhledávání spojení.

`MapFragment` třída zajišťuje zobrazení mapy a interakci s ní. To je umožněno přes implementaci `GoogleMap` objektu a jeho metod. V tomto fragmentu se nastavují parametry mapy, jako způsob ovládání a zobrazování aktivních prvků. Fragment pro zobrazení vozidel v okolí potřebuje získat pozici uživatele. Získání polohy umožnuje služba `LocationService`, ta k sestavení požadavku cyklického získávání polohy potřebuje instanci objektu `GoogleApiClient` a objektu `LocationRequest`, kde se nastavuje interval aktualizace polohy. Po získání polohy je možné volat metodu `getNearestVehicles` třídy `ApiClient`, která vrátí list objektů `NearestVehicle` s nejbližšími vozidly. Vozidla jsou poté

vykreslena do mapy a každých 10 vteřin je aktualizována jejich poloha. Při kliku na ikonu vozidla se zobrazí informační okno se základními informacemi o vozidle a na mapě je vykreslena čára značící jeho trasu. K zobrazení toho okna je implementována vlastní třída `VehicleInfoWindowAdapter`, která se stará o jeho plnění daty. Při dalším kliku na otevřené okno dojde k zavolání metody `getRoute`, pro získání trasy, a zobrazení okna s trasou vozidla. O plnění toho okna se stará třída `RouteAdapter`. Pokud je vybráno nějaké vozidlo, tak je v horním panelu zobrazeno tlačítko „Nastoupit“ po jeho stisknutí bude aplikace přepnuta do módu jízdy.

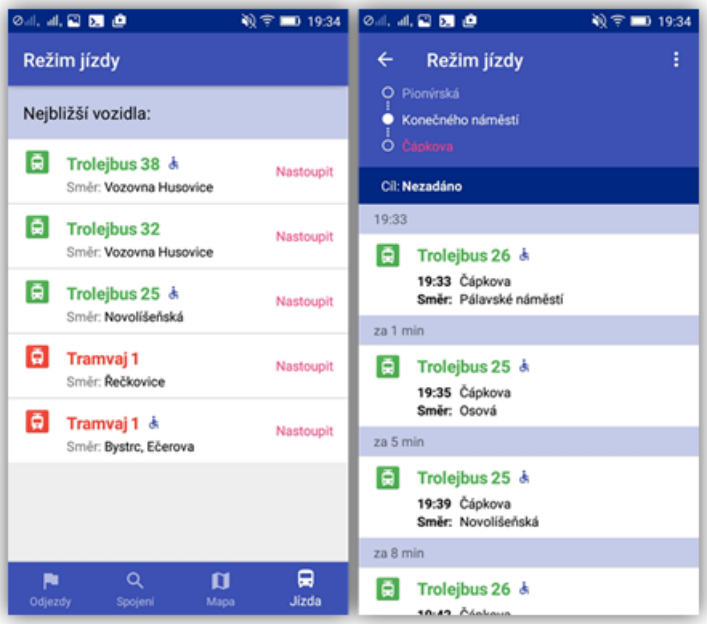


Obrázek 8.10: Vlevo je zobrazen náhled na obrazovku s mapou a vybraným vozidlem. Vpravo je náhled na obrazovku s dialogovým oknem s trasou vozidla.

Třída `DriveFragment` zobrazuje nejbližší vozidla k aktuální poloze uživatele. Jedná se o před krok režimu jízdy. Získání nejbližších vozidel probíhá voláním metody `getNearestVehicles` třídy `ApiClient`, která vrátí list objektů `NearestVehicle` s nejbližšími vozidly. Po získání těchto vozidel je `RecycleView` naplněn daty pomoci třídy `DriveNearestVehicleAdapter`. Po vybrání vozidla k přepnutí aplikace do režimu jízdy.

`DriveActivity` je třída zobrazující režim jízdy. Třída sleduje trasu vozidla, do kterého uživatel nastoupil. Zobrazuje také odjezdy z následující zastávky. Dále zobrazuje předešlou, aktuální a následující zastávku. Pokud je zadaná cílová stanice, je plánována optimální trasa a uživateli jsou zobrazeny jen relevantní informace. Výběr cílové stanice probíhá pomocí `SearchActivity` podobně jako v třídě `DepartureFragment`. Třída `DriveActivity` pro poskytnutí těchto informací potřebuje znát aktuální pozici uživatele a číslo vozidla, do kterého uživatel nastoupil. Číslo vozidla se získá pomocí metody getExtras hned při startu třídy v metodě `OnCreate`. Získaní aktuální pozice je obdobné jako ve třídě `MapFragment` pomocí `LocationService`. Informace se aktualizují každých deset vteřin nebo při změně pozice. Pokud se pozice uživatele a vozidla neshoduje, je zobrazeno dialogové okno s varováním.

Další důležitou částí při tvorbě aplikace byla volba jejího jména a ikony. Platí, že i sebelepší aplikace bez pořádné ikony a jména bude jen další aplikací s malým uživatelským



Obrázek 8.11: Vlevo je zobrazen náhled na obrazovku s nejbližšími vozidly. Vpravo je zobrazen náhled na obrazovku s režimem jízdy.

potenciálem. Nejdříve bylo důležité si uvědomit k čemu je aplikace určená a kdo a kde jí bude užívat. V případě této práce to bylo jasné. Aplikace bude usnadňovat cestování v Brněnské MHD a z toho jsem vycházel při volbě názvu aplikace "BrnoMHD". Volba ikony rovněž vycházela z účelu aplikace. Byla nalezena ikona prostředku městské hromadné dopravy a ta byla skombinována s názvem. Jelikož načítání aplikace nějaký okamžik trvá, je vhodné tento čas využít k zobrazení značky aplikace, tak zvanému Branded launch neboli obrazovky s logem a názvem aplikace. Výsledek je zobrazen na obrázku 8.12.

Obrázek 8.12: Návrh loga aplikace a loga pro startovací obrazovku.

Kapitola 9

Testování

Testování patří mezi velmi důležité části vývoje software. Jeho cílem je odhalit chyby a ověřit správnost implementace. Tato kapitola popisuje zařízení, na kterých proběhlo testování. Dále popisuje scénář jakým byla výsledná aplikace testována a na závěr diskutuje výsledky.

Testování probíhalo na několika mobilních zařízeních a také na emulátoru GenyMotion[1]. Výhodou emulátoru GenyMotion je rychlost, plynulost a hlavně možnost libovolné konfigurace emulovaného zařízení. Mezi nevýhody patří omezená dostupnost, složitá instalace knihoven a nemožnost získání aktuální pozice. Testování na emulátoru se tedy ve větší míře týkalo pouze uživatelského rozhraní. Další užitečný nástroj pro testování je Logcat, který poskytuje Android SDK. Pomocí Logcatu je možné sledovat výstup aplikace a debugovat případné chyby. Následující tabulka zobrazuje zařízení, která byla použita při testování.

Zařízení	Verze systému	Uhlopříčka	Rozlišení displeje
Samsung Galaxy S II	4.1.2	4.3	800x420
Huawei P7 mini	4.4.4	4.3	800x480
Lenovo S90	5.0.2	5.0	1280x720

Obrázek 9.1: Výčet testovacích zařízení.

Testování finální verze mobilní aplikace se zúčastnilo celkem šest testerů ve věku od 22 do 50 let s různou zkušeností s užíváním mobilního zařízení. Testování probíhalo hlavně na Lenovu S90, které bylo uživatelům poskytnuto a bylo provedeno podle následujícího scénáře. Uživateli byl nejprve představen koncept a cíle této aplikace. Poté mu bylo zadáno několik úkolů, které vycházely z hlavních případů užití popsaných v kapitole 7. Uživatel se nejdříve úkol pokusil splnit samostatně. Pokud se mu nedařilo splnit zadaný úkol, byla mu poskytnuta nápověda. Po dokončení zadaných úkolů uživatel vyplnil dotazník. Testování aplikace bylo zaměřeno na základní úkony, které budou v aplikaci prováděny nejvíce. Mezi ně patří:

- Vyhledávání odjezdů
- Vyhledávaní spojů
- Zobrazení mapy

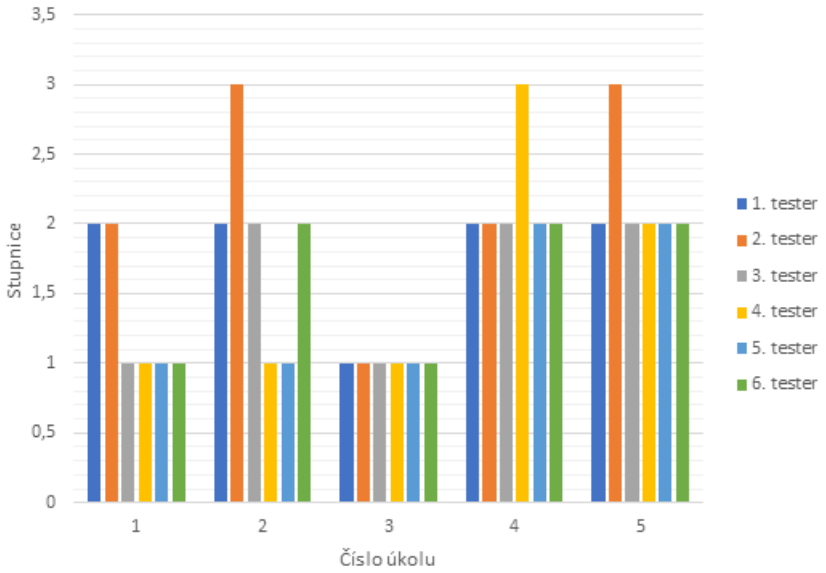
[1] GenyMotion https://www.genymotion.com/

- Zobrazení trasy vozidla
- Zobrazení informací o vozidle
- Režim jízdy

Závěrečný dotazník byl složen na základě výše zmíněných úkolů. Uživatel po provedení úkolu hodnotil jeho náročnost pomocí stupnice od jedné (nejlepší) do pěti (nejhorší) a také byl požádán o napsání krátkého komentáře. Dotazník se skládal z následujících otázek:

- Obtížnost vyhledávání – tento dotaz byl zaměřen na validaci vyhledávací obrazovky a relevantnost zobrazených dat. Uživatel dostal za úkol vyhledat odjezdy ze zastávky Semilasso. Dalším jeho úkolem bylo vyhledat spojení mezi zastávkami Česká a Řečkovice, vyhledaná vozidla měla být pouze nízkopodlažní.
- Přehlednost vozidel na mapě – zaměření toho dotazu bylo na velikost ikon, množství zobrazovaných vozidel. Uživatel měl za úkol zjistit trasu libovolného vozidla v jeho blízkosti.
- Navigace v aplikaci – cílem tohoto dotazu byla verifikace navrhnuté uživatelské navigace.
- Informační hodnota – tento dotaz se zaměřoval na přehlednost a relevantnost zobrazených výsledků vyhledávání. Uživateli byly předloženy výsledky vyhledávání, zobrazena mapa s vozidly a měl posoudit hodnotu těchto výsledků.
- Funkčnost režimu jízdy – úkolem tohoto dotazu bylo zjistit užitečnou hodnotu tohoto módu a jeho přehlednost. Uživatel měl přejít do režimu jízdy a sledovat svoji trasu.

Výsledky testování byly zpracovaný a zobrazeny v následujícím grafu 9.2.



Obrázek 9.2: Graf zobrazující výsledky dotazníku finálního testování. Na ose x jsou zobrazeny otázky dotazníku a na ose y jsou vyneseny známky.

Z grafu 9.2 je zřejmé, že první úkol „Obtížnost vyhledávání“ uživatelé hodnotili velmi kladně, čtyři ze šesti uživatelů ohodnotili vyhledávání známkou jedna. Průměrná známka pro tento úkol byla 1,3. Pro splnění úkolu nemusela být nikomu podána nápověda. V jednom komentáři k tomuto úkolu bylo napsáno: „Vyhledávání je intuitivní, srozumitelné a zobrazené výsledky jsou dobře čitelné“. Na základě výsledků a komentářů se dá říct, že se povedlo vytvořit jednoduché a jasné rozhraní pro vyhledávání zastávek.

Hodnocení druhého úkolu „Přehlednost vozidel na mapě“ dopadlo relativně dobře, dva uživatele dali známku jedna , tři uživatelé známku dva a jediný hodnotil číslem tři. Průměrná známka druhého úkolu byla 1,83. Většina námitek se týkala množství zobrazovaných vozidel na mapě a velikosti ikony vozidla.

Třetím úkolem bylo ohodnotit „Navigaci v aplikaci“, kterou všichni uživatelé shodně obodovali známkou jedna. Z výsledků je zřejmé, že se povedlo přepracovat navigaci, oproti druhé iteraci, do příjemné a funkční podoby. Uživatelé okomentovali splnění tohoto úkolu pozitivně. Navigace je dobře viditelná a v aplikaci je jednoduché se snadno zorientovat.

Čtvrtý úkol byl zaměřen na informativní hodnotu zobrazených výsledků po vyhledání. Pouze jeden uživatel hodnotil známkou tři, ostatní uživatelé známkou dva. Uživatelé ve většině případů hned poznali čeho se zobrazená obrazovka týká.

Poslední úkol se zabýval testováním módu jízdy, kde se většina uživatelů zajímala o jeho účel a využití. Po vysvětlení funkce většina hodnotila tento mód známkou dva a pouze jeden uživatel známkou tři. Z komentářů vyplynulo, že se jedná o zajímavý a pro někoho velmi užitečný nápad.

Na základě zpracovaných výsledků a neformálních rozhovorů s uživateli bylo v aplikaci provedeno několik drobných úprav, jejichž cílem bylo zvýšit užitnou hodnotu výsledné aplikace. Byl snížen počet zobrazovaných vozidel, přidán indikátor pro načtení výsledků a zvýšen počet některých informačních dialogových oken. Celkově se dá říct, že navigace, vyhledávání a poskytnuté výsledky jsou na dobré úrovni a poskytují uživateli hodnotné informace.

Kapitola 10

Závěr

Cílem této práce, k jejímuž vzniku dal prvotní impulz Ing. Radek Burget PhD., bylo vytvořit mobilní aplikaci pro operační systém Android, která bude sloužit jako asistent pro cestování v městské hromadné dopravě v Brně. Data pro mobilní aplikaci měla být získávána pomocí webové služby.

V rámci této práce byla provedena analýza a specifikace požadavků cílové aplikace, popsány existující řešení související s cestováním v městské hromadné dopravě a definovány technologie pro vývoj webových služeb a mobilních aplikací pro operační systém Android. Následně byl proveden návrh mobilní aplikace a serverové části. Na základě těchto poznatků byla provedena implementace. Na závěr bylo realizováno uživatelské testování.

Výsledkem diplomové práce je mobilní aplikace BrnoMHD a serverová část řešení. Mobilní aplikace umožňuje vyhledávání odjezdů a spojení mezi zastávkami a zobrazuje mapu s aktuální polohou vozidel hromadné městské dopravy v Brně. Dále poskytuje informace o režimu jízdy tím, že sleduje trasu vozidla, zobrazuje zastávky na trase a upozorňuje uživatele o případné vhodnější trase. Mobilní aplikace je implementována pro operační systém Android od verze 4.0.3, z důvodu dostupnosti co největšímu počtu uživatelů. Serverová část řešení slouží k poskytování dat pro mobilní aplikaci. Využívá statická data a webové služby poskytnuté v rámci spolupráce s centrálním dispečinkem Kordis Jihomoravského kraje. Webová služba poskytuje informace o aktuální dopravní situaci, lze z ní získat aktuální polohu vozidel, informace o zastávce a další data o hromadné dopravě. Statická data obsahují informace o veškerých zastávkách a jízdním řádu vozidel městské hromadné dopravy. Serverová část je postavena na zmíněných datech a webových službách. Byly nad nimi provedeny transformace a výpočty pro poskytnutí odjezdů, spojení vozidel a dalších užitečných dat. Serverová část je implementována v jazyku Java a umístěna na školním serveru, na kterém běží aplikační server Tomcat.

Každá z funkcí výsledné aplikace má svůj přínos pro cestování městskou hromadnou dopravou v Brně, jelikož vždy pracuje s aktuálními daty o poloze vozidel a dopravní situaci. Funkce vyhledávání odjezdů a spojení zobrazuje aktuální odjezdy nejbližších vozidel. Mapa poskytuje informace o aktuální poloze vozidel. Režim jízdy zobrazuje průběh trasy a informuje uživatele aplikace o možných výhodnějších spojeních.

Do budoucna je možné aplikaci umístit na obchod Google Play. Zde by byla veřejně přístupná, mohla by být hodnocena a získávat recenze od uživatelů pro případné další inovace. Možné další vylepšení aplikace vidím v implementaci pokročilejšího vyhledávacího algoritmu a doplnění lokalizace pro další světové jazyky.

Literatura

[1] Android Developer: *App Components*. 2017, [Online; navštíveno 10.4.2017].
URL https://developer.android.com/guide/components/index.html

[2] Android Developer: *Material Design*. 2017, [Online; navštíveno 18.4.2017].
URL https://material.io/

[3] Android Developer: *Navigation*. 2017, [Online; navštíveno 12.4.2017].
URL https://material.io/guidelines/patterns/navigation.html

[4] Android Developer: *Permissions*. 2017, [Online; navštíveno 18.4.2017].
URL https://developer.android.com/guide/topics/permissions/index.html

[5] Android Developer: *Platform Architecture*. 2017, [Online; navštíveno 18.4.2017].
URL https://developer.android.com/guide/platform/index.html

[6] Android Developer: *The Activity Lifecycle*. 2017, [Online; navštíveno 18.4.2017].
URL https://developer.android.com/guide/components/activities/activity-lifecycle.html

[7] Chinnici, R.; aj.: *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. 2007, [Online; navštíveno 8.1.2017].
URL https://www.w3.org/TR/wsdl20/

[8] Cinar, O.: *Android Quick APIs Reference*. Apress, 2015, ISBN 978-1-4842-0523-5.

[9] David Booth, H. H.; aj.: *Web Services Architecture*. 2004, [Online; navštíveno 8.1.2017].
URL https://www.w3.org/TR/ws-arch/

[10] Google Play: *IRIS*. 2017, [Online; navštíveno 18.7.2017].
URL https://play.google.com/store/apps/details?id=cz.zolex.iris&hl=cs

[11] Google Play: *Jízdní řády IDOS*. 2017, [Online; navštíveno 18.7.2017].
URL
https://play.google.com/store/apps/details?id=cz.mafra.jizdnirady&hl=cs

[12] Gudgin, M.; aj.: *SOAP*. 2007, [Online; navštíveno 8.1.2017].
URL http://www.w3.org/TR/soap12-part1/#intro

[13] Jendrock, E.; aj.: *The Java EE 6 Tutorial*. 2013, [Online; navštíveno 9.1.2017].
URL http://java.sun.com/javaee/6/docs/tutorial/doc/JavaEETutorial.pdf

[14] Kosek.cz: *Využití webových služeb a protokolu SOAP při komunikaci*. 2017, [Online; navštíveno 11.1.2017].
URL http://www.kosek.cz/diplomka/html/websluzby.html

[15] Kuba, M.: *Tutoriál Web Services*. 2017, [Online; navštíveno 12.1.2017].
URL http://dior.ics.muni.cz/~makub/soap/tutorial.html

[16] Linthicum, D.: *Service Oriented Architecture (SOA)*. 2017, [Online; navštíveno 7.1.2017].
URL https://msdn.microsoft.com/en-us/library/bb833022.aspx

[17] Malý, M.: *REST: architektura pro webové API*. 2009, [Online; navštíveno 7.1.2017].
URL https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[18] Mordani, R.: *Java Servlet Specification*. 2009, [Online; navštíveno 9.1.2017].
URL http://download.oracle.com/otn-pub/jcp/servlet-3.0-fr-eval-oth-JSpec/servlet-3_0-final-spec.pdf?AuthParam=1495277691_28c317355fa43d21c4ff2d2cb7dc7e33

[19] Ninjamock.com: *Mock-up design*. 2017, [Online; navštíveno 12.5.2017].
URL https://ninjamock.com/

[20] Rapant, P.: *Družicové polohové systémy*. Ostrava: Vysoká škola báňská - Technická univerzita, 2002, ISBN 80-248-0124-8.

[21] Richardson, L.; Ruby, S.: *RESTful web services*. O'Reilly Media, 2007, ISBN 0-596-52926-0.

[22] Rychlý, M.: *Webové služby v Java EE (JAX-WS)*. 2007, [Online; navštíveno 11.1.2017].
URL http://www.fit.vutbr.cz/~zendulka/grants.php?file=%2Fproj%2F378%2Fslides-webservices.pdf&id=378

[23] Rychlý, M.: *Servisně orientovaná architektura a její aplikace v systémech sledování a řízení výroby*. 2011, [Online; navštíveno 7.1.2017].
URL http://www.fit.vutbr.cz/~rychly/public/docs/arap11.soa-a-jeji-apl-pri-rizeni-vyroby/arap11-rychly.pdf

[24] Santiago Pericas-Geertsen, M. P.: *JAX-RS: Java API for RESTful Web Services*. 2013, [Online; navštíveno 9.1.2017].
URL http://download.oracle.com/otn-pub/jcp/jaxrs-2_0_rev_A-mrel-eval-spec/jsr339-jaxrs-2.0-final-spec.pdf?AuthParam=1495277840_b30e8d94ba21de931c015f5f4087d2e4

[25] Tutorialspoint.com: *UDDI - Overview*. 2017, [Online; navštíveno 8.1.2017].
URL https://www.tutorialspoint.com/uddi/uddi_overview.htm

[26] Ujbányai, M.: *Programujeme pro Android*. Apress, 2012, ISBN 978-80-247-3995-3.

[27] U.S. Coast Guard Navigation Center: *NAVSTAR GPS User Equipment Introduction*. 2017, [Online; navštíveno 10.1.2017].
URL http://www.navcen.uscg.gov/pubs/gps/gpsuser/gpsuser.pdf

[28] W3schools.com: *SOAP Tutorial*. 2017, [Online; navštíveno 8.1.2017].
URL https://www.w3schools.com/xml/xml_soap.asp

[29] Wennerstrom, T.; Jespersen, J.; Lundquist, M.: *Simple Object Access Protocol*. 2002, [Online; navštíveno 8.1.2017].
URL http://user.it.uu.se/~hsander/Courses/DistributedSystems/Reports/soap_report_2.pdf

[30] Wikipedia: *Global Positioning System*. 2017, [Online; navštíveno 11.1.2017].
URL https://cs.wikipedia.org/wiki/Global_Positioning_System

Přílohy

Příloha A

Obsah CD

text - zdrojove TEX soubory textu prace a z nich vygenerovane soubory .pdf

android - zdrojové kody mobilní aplikace

server - zdrojove kody serverove části

readme.docx - soubor obsahuje informace o spuštění aplikace a další užitečné informace