UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2017 Petr Kopic

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Aplikace pro sdílení polohy pro platformu Android

Petr Kopic

Bakalářská práce

2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2015/2016

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Petr Kopic**
Osobní číslo:	**I13153**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Aplikace pro sdílení polohy pro platformu Android**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem bakalářské práce je vytvořit aplikaci pro platformu Android, která umožní uživatelům sdílet svoji polohu s dalšími uživateli.

V teoretické části práce bude představena platforma Android s ohledem na vývoj aplikace pro sdílení polohy a návrh vlastní aplikace.

V praktické části bude implementována aplikace pro sdílení polohy na platformě Android. Aktuální poloha uživatele a ostatních uživatelů bude v reálném čase zobrazena na mapě. Uživatelé mohou vytvářet skupiny a sdílet v rámci skupin svoji polohu. Aplikace umožní automatické sdílení a aktualizace polohy i v případě, že uživatel nebude mít aplikaci spuštěnou na popředí.

Rozsah grafických prací:

Rozsah pracovní zprávy: **30-40 stran**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

*** MEDNIEKS, Zigurd R. Programming Android. Beijing: O'Reilly, 2012, 12, 542 pages. ISBN 9781449316648**

*** HARWANI, B. M. Android Programming Unleashed. Indianapolis: Sams Publishing, 2012, 696 p. ISBN 9780133151749.**

Vedoucí bakalářské práce: **Ing. Roman Diviš**
Katedra softwarových technologií

Datum zadání bakalářské práce: **31. října 2015**

Termín odevzdání bakalářské práce: **13. května 2016**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Mgr. Josef Horálek, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2016

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 24. 4. 2017

Petr Kopic

PODĚKOVÁNÍ

Chtěl bych poděkovat panu Ing. Romanu Divišovi za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování bakalářské práce věnoval.

ANOTACE

Bakalářská práce se zaměřuje na vývoj aplikace pro platformu Android, která umožní sdílet uživatelům svoji polohu s ostatními uživateli. V první části práce jsou teoreticky popsány hlavní části systému Android s ohledem na vývoj aplikace. Ve druhé části je popsána implementace samotné aplikace pro sdílení polohy. Je popsána jak serverová část, tak i jednotlivé třídy aplikace.

KLÍČOVÁ SLOVA

Android, aplikace, poloha, sdílení

TITLE

Application for location sharing for the Android platform

ANNOTATION

This Bachelor's thesis is focused on the development of an Android-platform application which allows its users to share their location with other users. The first part of the thesis contains theoretical description of the primary components of the Android system from application development point of view. In the second part, the implementation process of the location-sharing application is described. The server part is described, along with the individual classes of the application.

KEYWORDS

Android, application, location, sharing

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

API	Application Programming Interface
IDE	Integrated Development Environment
HTTP	Hypertext Transfer Protocol
PDA	Personal Digital Assistant
PDO	PHP Data Object
PHP	PHP: Hypertext Preprocessor (původně Personal Home Page)
Sb.	Sbírka zákonů
SDK	Software Development Kit
SQL	Structured Query Language
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	eXtensible Markup Language

ÚVOD

Hlavním cílem této bakalářské práce je vytvořit aplikaci, která bude mezi skupinami uživatelů sdílet v reálném čase polohu zařízení, na kterém je spuštěna. Aplikace má být vyvinuta na zařízení s operačním systémem Android, hlavně tedy na mobilní telefony a tablety. Práce se věnuje popisu operačního systému Android a dále samotnému vývoji aplikace pro sdílení polohy. Tato práce je rozdělena do tří hlavních kapitol.

První kapitola je věnována popisu základních požadavků na aplikaci. Jsou zde obecně popsány funkce, které má zvládat a nabídnout uživateli. Je popsáno, jak by měla fungovat a navrhnuto několik způsobů řešení vybraných problémů.

Druhá kapitola popisuje operační systém Android. Jsou zde uvedeny základní informace a využití systému ve světě. Další části kapitoly se věnují základní architektuře systému a popisu vývoji aplikací na systém Android. Jsou uvedeny potřebné informace o přípravě na vytváření aplikací a základní části a komponenty, které je tvoří. V poslední části kapitoly jsou popsány možnosti testování aplikací.

Poslední kapitola uvádí konkrétní vývoj aplikace pro sdílení polohy na systém Android. Jsou uvedeny potřebné znalosti, instalace vývojového prostředí, balíků a programů pro vývoj a testování aplikace. Dále se kapitola věnuje sdílenému úložišti dat ve formě databáze a všech potřebných funkcionalit, které s ní souvisí, jako je způsob komunikace nebo předávání dat a informací. Jedna podkapitola je také věnována založení a orientaci v projektu aplikace. V poslední části kapitoly jsou popsány jednotlivé části aplikace (třídy, obrazovky), jejich funkce v aplikaci a jejich obsah.

1 INFORMACE O APLIKACI

Cílem práce je vytvořit aplikaci pro platformu Android, která uživateli umožní sdílet i sledovat svoji polohu. Uživatel si bude schopen vytvářet svoje skupiny, (např. pro své známé, příbuzné, kolegy). Ty potom bude moci sledovat na mapě, která bude zobrazovat polohu jednotlivých členů skupiny.

1.1 Základní funkce aplikace

Po otevření aplikace uživatel dostane možnost přihlásit se nebo registrovat. V případě přihlášení se zadané uživatelské jméno a heslo porovná s údaji v databázi, která bude uložená na serveru. Pokud údaje odpovídají, uživatel dostane oznámení, že je přihlášen. V případě registrace je také nutno vyplnit údaje jméno a heslo. Jelikož je nutné, aby uživatelské jméno bylo unikátní, porovná se s údaji v databázi a v případě shody s některým uživatelem, nebude nový účet zaregistrován. V případě úspěšné registrace se pokračuje na obrazovku přihlášení.

Aplikace by měla umožňovat sdílet polohu jen s určitými skupinami uživatelů. K tomuto účelu bude vytvořena obrazovka pro spravování skupin uživatele. Přihlášený uživatel bude schopen přidávat nebo odebírat skupiny z vlastního seznamu, který bude zobrazen na obrazovce. Dostane také možnost vytvořit si svoji vlastní skupinu. Pro vytvoření nové skupiny je zapotřebí vyplnit unikátní název skupiny a libovolné heslo. Skupiny i s hesly budou uloženy v databázi. Pro přihlášení do určité skupiny bude stačit najít skupinu v rozevírací nabídce a vyplnit její heslo. Pokud bude vše v pořádku, bude skupina přidělena přihlášenému uživateli.

Hlavní funkcí této aplikace je zobrazení mapy s polohou uživatelů. Na mapě budou zobrazeni všichni uživatelé, kteří jsou přítomni ve skupinách přihlášeného uživatele. Na poloze těchto uživatelů bude vykreslena značka s popiskem, kde bude zobrazeno jméno uživatele spolu s datem a časem, kdy u něj naposledy proběhla aktualizace polohy. Aktualizace polohy bude probíhat, ikdyž aplikace nebude běžet na popředí a za podmínek, že bude existovat přihlášený uživatel a bude zapnutá funkce GPS

K tomu, aby aplikace fungovala, bude uživatel potřebovat mobilní telefon nebo tablet s operačním systémem Android verze 4.0 a vyšší, stabilní připojení k internetu a zapnutou funkci GPS. Pokud zařízení nebude připojeno k internetu nebo nebude mít zapnutou funkci GPS, uživatel o tom bude informován pomocí dialogového okna.

Databázový server bude vytvořen na freehostingovém portále endora.cz, kde bude zaregistrována doména a k ní vytvořena jednoduchá databáze, která bude uchovávat uživatele (s jejich polohou) a jednotlivé skupiny [13].

Aplikace by mohla mít poměrně široké využití. Mohla by sloužit pro opatrné rodiče, kteří by kontrolovali své děti například při cestě ze školy, nebo pro zaměstnavatele, kteří by nutně chtěli kontrolovat polohu svých zaměstnanců. Také může sloužit čistě pro zábavu, kdy se skupina přátel rozhodne, že se budou navzájem sledovat.

2 SYSTÉM ANDROID

2.1 Obecné informace

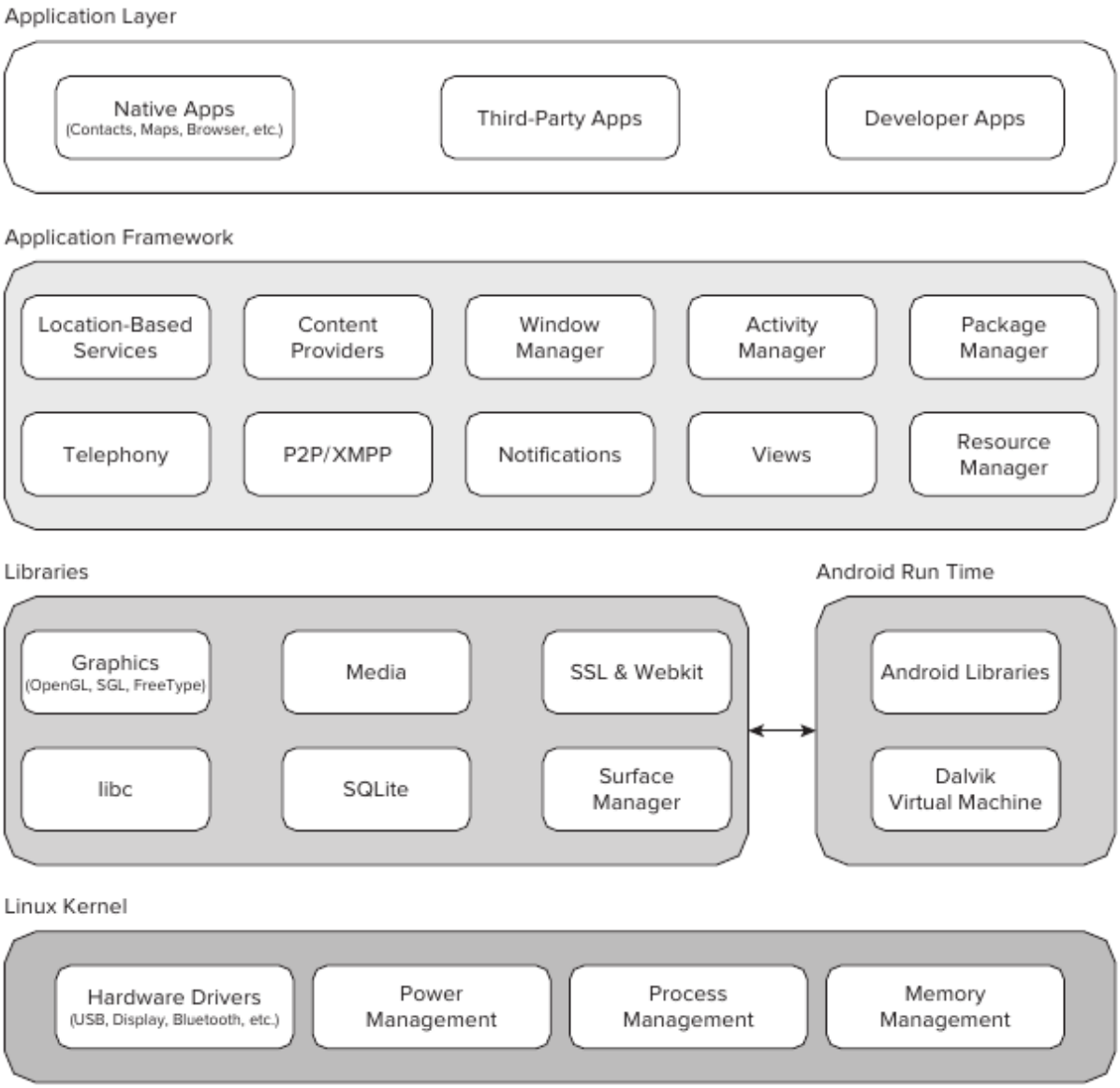
Android je rozsáhlá open-source platforma, která vznikla hlavně pro mobilní zařízení (smartphone, PDA, navigace, tablety, hodinky, atd.) [3]. Je to operační systém založený na Linuxovém jádře. Vyvíjí ho konsorcium Open Handset Alliance, kterého cílem je progresivní rozvoj mobilních technologií. Systém je vyvíjen s ohledem na omezení mobilních zařízení (výdrž baterie, menší výkon, menší paměťové možnosti), i když se tyto nevýhody s postupem doby snižují [4].

Platforma Android dává k dispozici nejen samotný systém s uživatelským rozhraním pro uživatele, ale i specifikaci ovladačů pro výrobce zařízení a mobilní operátory, ale i balík nástrojů pro vývojáře – Software Development Kit (SDK) [4].

Ke dni 16. 8. 2016 bylo zjištěno, že operační systém Android využívá 86,2 procent obyvatel používajících mobilní zařízení [5]. Nejpoužívanější verze Androidu jsou v lednu roku 2017 Android 5 (verze 5.0 a 5.1) zvaný Lollipop, Android 6.0 zvaný Marshmallow a Android 4.4 zvaný KitKat. Tyto verze Androidu nyní využívá přibližně 86 procent aktivních zařízení [6].

2.2 Architektura

Architektura systému Android je rozdělena do pěti vrstev, které jsou uvedeny na obrázku (Obrázek 1 - Vrstvy systému Android).



Obrázek 1 - Vrstvy systému Android [14].

2.2.1 Linux Kernel

Linux Kernel je nejnižší vrstva architektury, která zajišťuje komunikaci mezi hardwarem a softwarem na vyšších vrstvách [3]. Mezi jeho hlavní součásti, které Android využívá, tedy patří ovladače, správa paměti, procesů a napájení. V systémech Android se využívá Linuxového jádra také z důvodu snadného sestavení na různých zařízeních, čímž je zaručena snadná přenositelnost.

2.2.2 Knihovny (Libraries)

Knihovny využívají další funkce systému (např.: zobrazování aplikací, práce s grafikou, práce se soubory nebo *SQLite*) [2]. Jsou napsány v jazycích C nebo C++ [8]. Tyto funkce jsou poskytovány pomocí *Android Application Framework* [8].

2.2.3 Android runtime

Tato vrstva slouží pro běh aplikací. Zajišťuje převod z kódu, ve kterém je napsaná aplikace (především z Javy) do nativního kódu. Pro tento účel se nyní používá dopředná kompilace (*AOT – Ahead-of-time compilation*). Z důvodů zpětné kompatibility vrstva obsahuje i *Dalvik VM*, který se staral o převod kódů[2]. Byl vyměněn kvůli zrychlení aplikací a úspoře energie. Vrstva dále obsahuje Java knihovny.

2.2.4 Aplication Framework

Je to nejdůležitější vrstva pro vývojáře. Umožňuje přístup k mnoha službám, které vývojáři používají v aplikacích. Jejich bližší popis bude popsán v další kapitole [7].

2.2.5 Aplikace (Application layer)

Nejvyšší vrstvu tvoří aplikace, které využívají uživatelé [7]. Jedná se jak o základní aplikace (SMS program nebo kontakty) i aplikace nainstalované z Google play, nebo vytvořené uživatelem (hry, e-mailový klient, webový prohlížeč) [2].

2.3 Application Framework

Poskytuje přístup k velkému počtu služeb. Tyto služby zpřístupňují data v jiných aplikacích, prvky uživatelského rozhraní, upozorňovací stavový řádek, aplikace běžící na pozadí, hardware používaného zařízení a mnoho dalších [7]. Tyto funkce jsou zahrnuty v rozhraních API, které jsou napsány v jazyce Java [8]. Dohromady tvoří základní stavební kameny pro tvorbu aplikací pro systém Android, které se dále dělí do skupin:

- **Sada prvků View -** Pomocí těchto prvků se sestavuje uživatelské rozhraní jako tlačítka, checkboxy, textová pole, seznamy a další [10].
- **Content providers -** Umožňuje přístup k obsahu a datům jiných aplikací (např. kontakty, kalendář, …) [10].
- **Resource manager -** Poskytuje přístup k dalším zdrojům jako jsou řetězce, grafika, přidané soubory [10].
- **Notification manager -** Umožňuje všem aplikacím zobrazit vlastní upozornění ve stavovém řádku [10].

- **Activity manager -** Řídí životní cyklus každé aplikace a poskytuje orientaci v zásobníku aplikací [10].

2.4 Vývoj aplikací

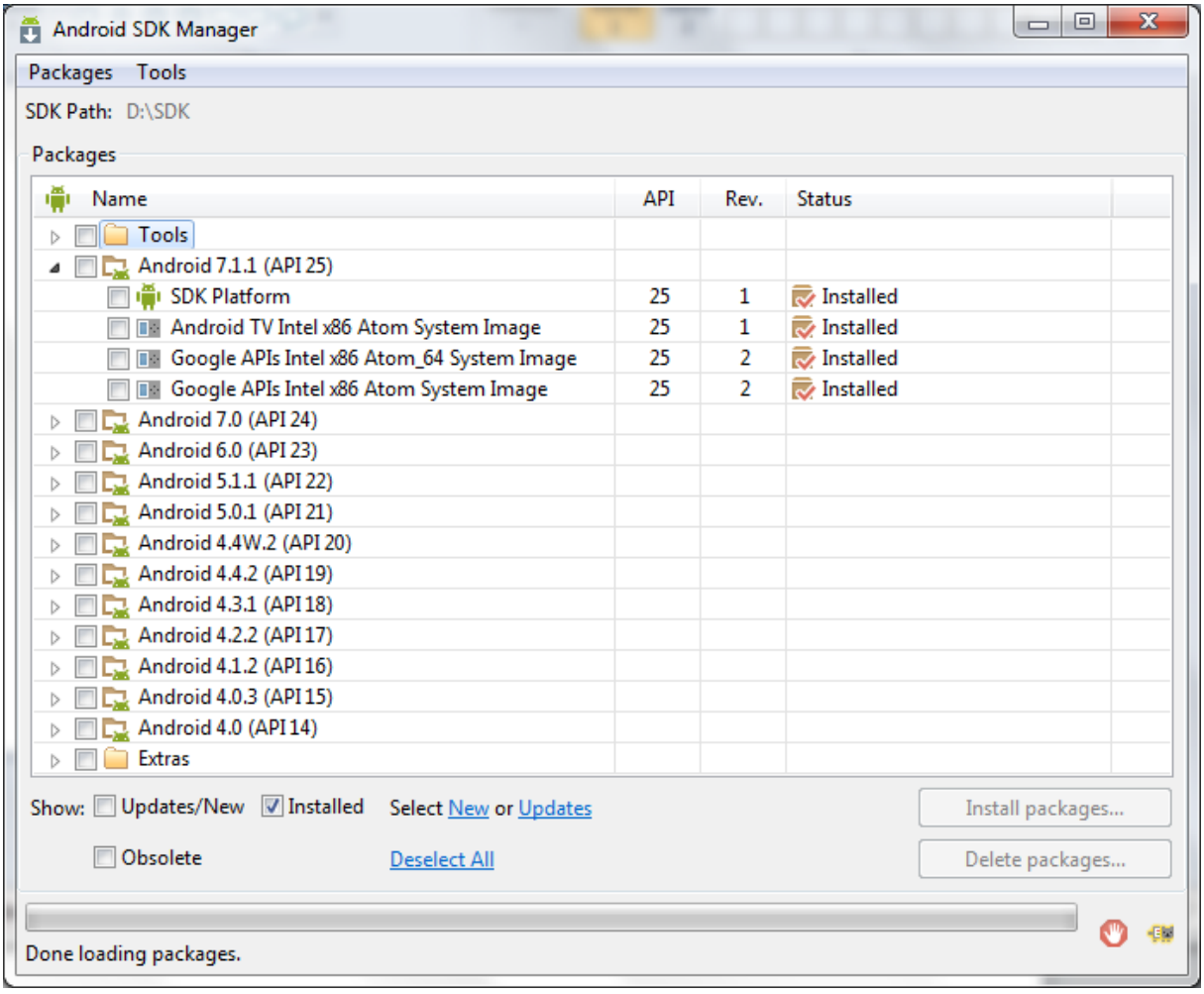
Hlavním programovacím jazykem, ve kterém se programují aplikace na Android je Java, tudíž je zapotřebí mít nainstalované JDK [3]. Dále je třeba zvolit vhodné vývojové prostředí (IDE). Hlavním podporovaným IDE je *Android Studio* od firmy Google. Toto IDE je volně dostupné na internetu, je velmi jednoduché na instalaci a ovládání a součástí instalace jsou balíčky potřebné pro vývoj aplikací. Součástí IDE je balík SDK, který obsahuje nástroje pro tvorbu aplikací [2]. Navíc obsahuje další důležitou část, a to emulátor. Tento balík je buď součástí instalace IDE, nebo je volně dostupný na internetu. Je dostupný, stejně jako IDE, pro operační systémy Windows, GNU/Linux i macOS.

2.5 Android SDK

Je sada vývojových nástrojů umožňující vytváření aplikací pro Android [1]. Pro vývoj aplikací na Android potřebujeme tři skupiny nástrojů.

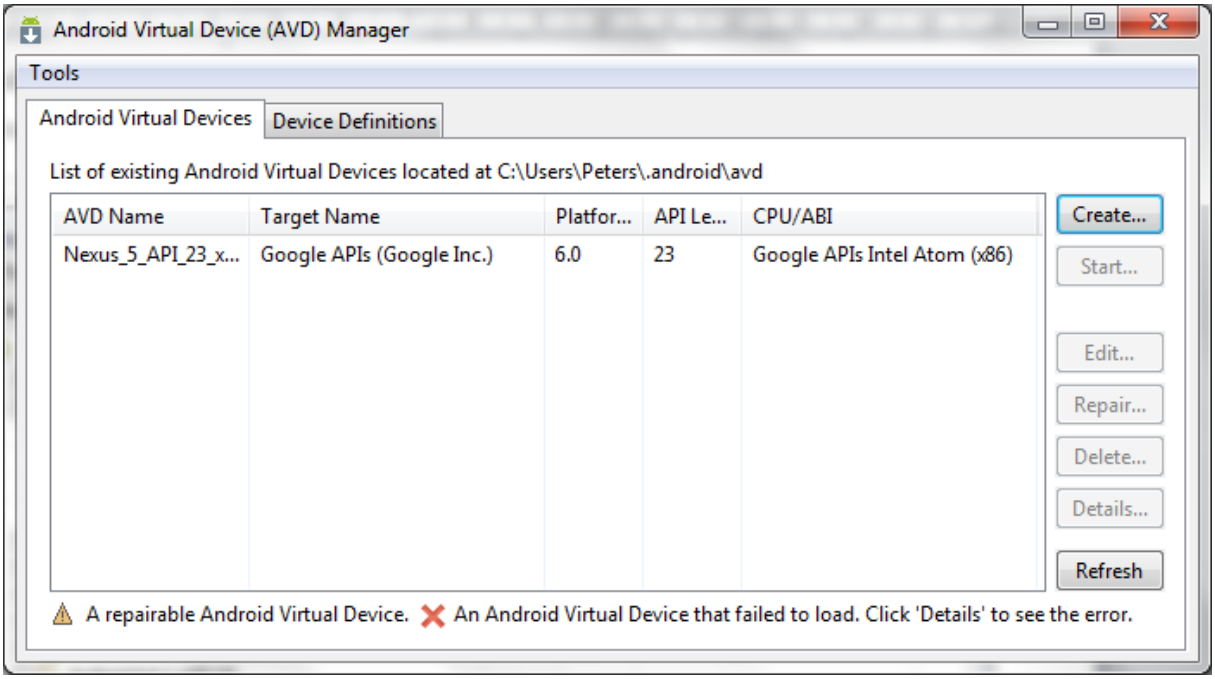
2.5.1 SDK Tools

Tyto nástroje jsou nainstalovány v základní konfiguraci balíčku a jsou pravidelně aktualizovány. Patří sem *SDK manager*, který umožňuje správu nainstalovaných nástrojů a verzí systému. [10]



Obrázek 2 - Ukázka Android SDK Manager

Dále tato skupina zahrnuje *AVD manager*, ve kterém můžeme vytvářet a spravovat virtuální zařízení. Pomocí těchto zařízení se dají ladit vytvořené aplikace v emulátoru. Následně tato skupina zahrnuje samotný emulátor. [2]



Obrázek 3 - Ukázka Android AVD Manager

Důležitou částí je také *Android monitor*, který je integrován do *Android studia* a umožňuje sledovat systémové zprávy nebo využití paměti, procesoru nebo grafického čipu [10]. Další nástroj, který stojí za zmínku je *Dalvik debug monitor server* (*ddms*) [2]. Tento nástroj nám dává možnost odladit naše aplikace.

Důležitou částí je také *Google repository*, která poskytuje řadu funkcí, například Google mapy [10].

2.5.2 SDK platform-tool

Hlavním zástupcem této skupiny je *Android Debug Bridge* (*adb*). Může se používat ke správě stavu instance emulátoru nebo k instalaci aplikace na zařízení [9]. Dalšími nástroji jsou například *aidl*, *aapt*, *dexdump* nebo *dx* [10]. Tyto nástroje nejsou zapotřebí tak často.

2.5.3 Android SDK platforms

Ke kompilaci vlastních programů je zapotřebí mít nainstalovanou alespoň jednu Android platformu, což je vlastně verze operačního systému Android. Každá verze obsahuje systémové obrazy, které specifikují architekturu procesoru, knihovny, ukázkové kódy, nebo Google API. Jednotlivé verze a jejich nainstalované nástroje lze nalézt v *SDK manageru*, kde můžeme tyto funkce aktualizovat nebo mazat. [10]

2.6 Android studio

Je vývojové prostředí založené firmami Google a JetBrains [11]. Nejnovější verze je v této době *Android Studio 2.3.1* [10]. Jak už název napovídá, dají se v něm vyvíjet aplikace pouze pro operační systém Android (mobilní telefony, tablety, *Android Wear* nebo TV) a je zcela zdarma. *Android Studio* je založeno na *IntelliJ IDEA* a poskytuje špičkovou práci s kódem (navigace, refaktoring, našeptávání, analýza, atd.). V *Android Studiu* se programuje pomocí programovacího jazyka Java.

Při zakládání projektu můžeme svou aplikaci směřovat na určitý druh zařízení (mobil, tablet, atd.) i na verzi systému. Verzi systému zvolíme výběrem API (rozhraní pro programování) [10]. Po vybrání určitého rozhraní půjde aplikace spustit pouze na zvoleném a vyšším rozhraní.

V *Android Studiu* se příjemně tvoří i design jednotlivých částí aplikace. Máme na výběr psaní v XML kódu, při kterém se rovnou zobrazuje náhled obrazovky v rozlišení, které si zvolíme. Existuje i možnost zobrazit náhled na všech možných rozlišeních najednou (*Preview all screens sizes*), nebo jen ty nejdůležitější rozlišení (*Preview Representative Sample*). Druhou možností tvoření layoutů je design mód, který je velmi přesný a jednoduchý. Lze jednoduše vkládat a upravovat jednotlivé prvky, jejich velikost, tvar, pozadí, akce po kliknutí apod. [11]

Android studio má v sobě zabudovanou i správu virtuálních zařízení, která je přímo napojená na *AVD manager*, takže provedené změny se promítnou i v druhém programu. Lze nastavit spoustu parametrů jako například API, velikost RAM, velikost paměťové karty, kamery a další. [10]

Pro uživatele, kteří by chtěli změnit vývojové prostředí z *Eclipse* (jedno z nejpoužívanějších prostředí v minulých letech), existuje možnost exportu projektu z *Eclipse* a importu do *Android Studia*.

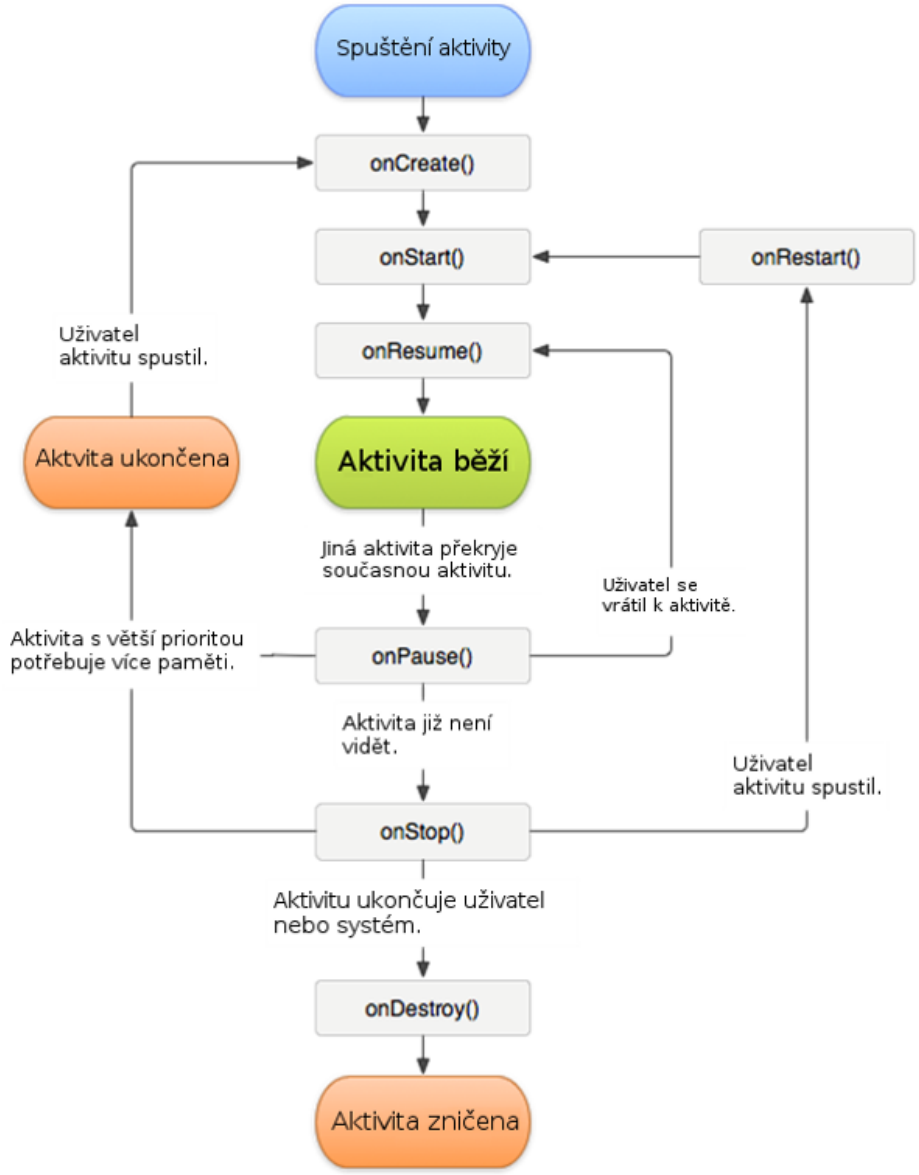
2.7 Základní časti aplikace

Každá aplikace napsaná pro operační systém Android se může skládat ze základních prvků *aktivit, services, content providers* a *broadcast recievers* [2]. Každá z použitých komponent a jejich názvy musí být uvedeny v souboru *AndroidManifest.xml*, který je uložený v kořenovém adresáři projektu [10]. Výhodou jednotlivých komponent, kromě *content provider*, je možnost vzájemné komunikace mezi sebou pomocí zpráv (*Intent*).

2.7.1 Activity

Každá aktivita odpovídá jedné obrazovce. Aktivita obsahuje uživatelské rozhraní, pomocí kterého aplikace komunikuje s uživatelem [20]. Každá aplikace obvykle obsahuje více aktivit, mezi kterými se dá přepínat, a aktivity si mohou mezi sebou předávat informace. Aktivita je většinou prezentována jako okno přes celou obrazovku, ale nemusí tomu tak být. Může se nacházet i jako plovoucí obrazovka nebo část jiné aktivity. Každá aktivita musí být uvedena v souboru *AndroidManifest.xml* (<activity android:name=".Prihlaseni" />) [10].

Jelikož zahájení aktivity zabere dost výpočetních prostředků (nový proces, paměť), stará se o rušení, vytváření a správu aktivit *Activity Manager* [10]. *Activity Manager* pracuje se zásobníkem, ve kterém se uchovávají informace o běžících aktivitách a na vrcholu se nachází právě spuštěná a viditelná aktivita. Každá aktivita má svůj životní cyklus (Obrázek 4 - Životní cyklus aktivity [20].) a může se nacházet v jednom ze čtyř stavů, a o to ve kterém se nachází se stará *Activity Manager*.



Obrázek 4 - Životní cyklus aktivity [20].

První ze stavů, aktivita je spuštěna, nastává při inicializaci aktivity. Další stav, aktivita běží, nastává, když je aktivita na vrcholu zásobníku, tedy aktivní a uživatel s ní může pracovat [20]. V tomto stavu se vždy může nacházet pouze jediná aktivita a ta je viditelná na obrazovce zařízení [10]. Další stav může nastat při kritickém nedostatku paměti, tím pádem *Activity Manager* ukončí aktivitu. Ve většině případů se jedná o aktivitu, která není viditelná, ale může nastat i situace, kdy zruší viditelnou aktivitu, se kterou uživatel nemůže navázat interakci (dialogová hláška). Posledním stavem je zničení aktivity. Tento stav znamená, že *Activity Manager* ukončil a zničil aktivitu a ta již nezabírá žádnou paměť. Tento stav může nastat například při ukončení aplikace.

Námi vytvořená aktivita je vlastně třída, která dědí z třídy *Activity* [10]. K vytvoření aktivity slouží metoda *onCreate(Bundle)*, ve které se pomocí metody *setContentView(R.layout.nazev_layout)* připojí grafické rozvržení a prvky aktivity, které jsou uloženy v XML souboru dané aktivity, ve složce layout. Jednotlivé prvky (tlačítka, textová pole, …) se musí v metodě *onCreate* také definovat a dále s tímto prvkem můžeme pracovat. Například na tlačítko nastavit *listener* a v něm požadované akce po kliknutí. V *listeneru* můžeme vyvolat jinou aktivitu pomocí metody *startActivity()* [1]. Například pro přejití z hlavní aktivity do formuláře pro přihlášení. Příklad metody *onCreate()* s jedním tlačítkem:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    uzivatelText = (TextView) findViewById(R.id.id_text);
    butt = (Button) findViewById(R.id.id_tlacitko);
    butt.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        startActivity(new Intent(MainActivity.this, Prihlaseni.class));
      }
    });
}
```

Další metoda, která stojí za zmínku je *onPause()*. Tato metoda se vyvolá, když se systém chystá zahájit obnovení předešlé aktivity. Můžeme v ní nastavit, aby se některá data v aktuální aktivitě, která chceme uložit, neztratila. Obvykle se nastaví *ContentProvider*, do kterého se data uloží. Řešení této metody by mělo být velmi rychlé, aktivita se totiž spustí až po ukončení této metody. [20]

Existují i další metody, které se vyvolávají při změně životního cyklu aktivity, a to jsou *onStart(), onRestart(), onDestroy(), onResume()* a *onStop()*. Kompletní dokumentaci o nich naleznete na stránce *https://developer.android.com/reference/android/app/Activity.html* [10].

Jelikož existuje mnoho druhů zařízení s operačním systémem Android a každé z nich má jinou velikost displeje, je někdy nutné zobrazovat komponenty patřící jedné aktivitě jinak na telefonu a například na tabletu. K řešení tohoto problému slouží fragmenty. Fragment slouží jako mezivrstva mezi aktivitou a konečným zobrazením. V aktivitě se rozhodne (například podle velikosti displeje), jaký fragment bude vykreslen. K tomu je ovšem zapotřebí vytvořit několik fragmentů a několik layoutů, které bude aktivita využívat podle potřeby. [21]

Každý fragment má svůj životní cyklus, který je podobný jako u aktivity. Podobné jsou také metody, které jsou volány, při změně stavu cyklu [10]. Životní cyklus fragmentu je však ovlivněn cyklem aktivity, která ho obaluje. Pokud je například aktivita zrušena, jsou zrušeny i všechny její fragmenty.

Pro výsledné zobrazení fragmentu je nutné ho vložit do určitého layoutu, nebo lze vložit i programově. Při vkládání do layoutu se vloží značka *<fragment>* a vyplní parametry *android:name* a *android:id,* nebo *android:tag*, které slouží k určení fragmentu v kódu a práci s jeho daty. V případě programového přidání je jsou využity objekty tříd *FragmentManager* a *FragmentTransaction.* [10]

Fragmenty se využívají, protože při jejich používání můžeme vytvořit aplikaci pro zařízení s různými rozměry displeje, bez duplikování kódu. Pro různé velikosti se fragmenty poskládají jinak, nebo se rozloží do více různých aktivit (v případě malého displeje) a jejich kód zůstává napsaný v aplikaci jen v jednom fragmentu. [21]

2.7.2 Service

Tato komponenta představuje službu běžící na pozadí. Nemá žádné svoje uživatelské rozhraní a nemusí to být ani samostatné vlákno nebo proces. Používá se hlavně pro dlouhodobě se opakující úkony, nebo pro přístup ke vzdáleným zdrojům s nejistou dobou odezvy (server). Každý service musí být uveden v souboru *AndroidManifest.xml* stejně jako activity *(<service android:enabled="true" android:name=".services.Nazev/">).* [10]

Service běží v hlavním vlákně aplikace, proto je vhodné, pokud service řeší výpočetně náročnější operace, přesunout ho do samostatného vlákna. Na to existuje jednoduché řešení a to třída *IntentService*, která zpracovává službu v samostatném vlákně. Pro normální službu běžící v hlavním vlákně aplikace se používá třída *Service* [1].

Existují dva způsoby spuštění služby. První je pomocí volání *Context.startService().* Tento způsob slouží k operacím, které se mají provádět na pozadí i v případě, že uživatel nemá aplikaci otevřenou [2]. Ukončuje se sama po dokončení operací nebo ji může uživatel ukončit prostřednictvím jiné komponenty. Druhou možností je metoda *Context.bindService()*, která umožnuje vystavit některé funkčnosti služby do jiných aplikací. Služba je vyvolána jinou komponentou a tou musí být také ukončena. Je možné na službu připojit více komponent, potom se služba zruší po odpojení všech [10].

Pokud službu vytvoříme pomocí *Context.startService()*, potom systém načte službu (vytvoří a v případě potřeby zavolá *onCreate()*), poté volá metodu *onStartCommand(Intent, int, int)* s parametry zadanými uživatelem [10]. Služba běží, dokud není zastavena zavoláním metody *Context.stopService()* nebo *stopSelf()*. V tomto případě nezáleží, jestli je služba ještě využívána a zastaví se. Existuje však metoda *stopSelf(int)*, která počká, až se ukončí všechny započaté procesy se službou a pak se ukončí.

Klient také může využít *Context.bindService()* k vytvoření trvalého připojení ke službě [10]. Tímto způsobem se také spustí služba, pokud již není spuštěna, ale nevyvolá se metoda *onStartCommand()*. Služba pomocí metody *onBind()* vrátí klientovi objekt *IBinder*, který umožňuje klientovi nadále být ve spojení se službou. Služba tedy běží, dokud je držen odkaz na objekt *IBinder* [2].

Při rušení služby je volána metoda *onDestroy()*, která vše ukončí. Když je tato metoda ukončena měla by být zastavena všechna vlákna používaná službou i zrušení registrace přijímačů. [10]

Každý service se může nacházet v jednom ze tří stavů. Prvním stavem je *Component call*. Tento stav nastává při inicializaci zavoláním nebo při navázáním komponenty na službu. Druhým stavem je *Service is running*. V tomto stavu služba vykonává svou činnost na pozadí. Posledním stavem je *Service is shut down*, který nastává po ukončení služby buď komponentou, nebo se ukončí sám. To záleží na způsobu spuštění služby. [10]

Kompletní dokumentaci a způsob vytvoření služby různými způsoby lze najít na webové stránce *https://developer.android.com/reference/android/app/Service.html* [10].

2.7.3 Content provider

Je blok programu sloužící k uchovávání dat buď v rámci jedné aplikace (přenos mezi aktivitami), nebo může poskytovat data i jiným aplikacím [2]. Data je možné uchovávat v souborech, na webu nebo v *SQLite* databázi.

K vytvoření je zapotřebí vytvořit třídu *ContentProvider* a v ní metodu *onCreate()* [2]. Pro práci s daty *ContentProvider* nabízí funkce *insert()* pro vkládání dat do zvoleného uložiště, *update()* pro aktualizaci dat, *delete()* pro smazání dat a *query()* pro vrácení určitých dat. Ještě existuje metoda, která vrací typ dat uložených v *ContentProvider*, a to *getType()*. Každá metoda pro práci s daty musí mít jako parametr URI konkrétního ContentProvider a další parametry pro upřesnění práce s daty [10]. Oddělení dat od aplikace má výhody. Typickým

použitím *Content Provideru* je například telefonní seznam kontaktů. Jelikož jsou data oddělena od výchozí aplikace seznamu kontaktů, není problém změnit výchozí seznam za svůj vlastní nebo jinde stažený.

Na druhu sdílení dat (v rámci jedné nebo více aplikací) záleží druh poskytovatele obsahu. Pokud sdílíme data v rámci jedné aplikace, stačí použít databázi *SQLite*, která je v systémech Android vestavěná. Pokud, ale chceme sdílet data v rámci více aplikací, je nutné zvolit rozhraní *ContentResolver*. Pokud je zvolen *ContentResolver* systém automaticky kontroluje oprávnění daného URI a předá požadavek na určitý *ContentProvider*. [10]

Kompletní dokumentaci třídy Content provider lze nalézt na webové stránce *https://developer.android.com/reference/android/content/ContentProvider.html* [10].

2.7.4 Broadcast receiver

Je komponenta sloužící k vytváření reakcí na různé oznámení (například oznámení o nízkém stavu baterie, stažení souboru nebo doručení SMS zprávy). Reakcí může být například jenom vypsání na stavový řádek anebo spuštění jiné komponenty. Tato komponenta nemá uživatelské rozhraní a je nutné ho uvést v souboru *AndroidManifest.xml* pod tagem <*receiver*>. Podobně jako u jiných komponent lze rozlišit, zda bude *Broadcast receiver* fungovat napříč různými aplikacemi nebo jen v rámci jedné. [10]

V rámci jedné aplikace je lepší použít třídu *LocalBroadcastManager*, která je jednodušší. Není v ní řešeno bezpečnostní opatření přijímání nebo odesílání vysílání z jiných aplikací. Při použití v rámci více aplikací se používá třída *BroadcastReceiver*. [10]

Existují dva hlavní druhy vysílání, které se dají přijmout. První je normální broadcast vysílání, které je odeslané metodou *Context.sendBroadcast()* [2]. Toto vysílání je zcela asynchronní, což znamená, že všechny přijímače mohou přijmout vysílání najednou a v neuspořádaném pořadí. Tato metoda je sice efektivnější, ale přijímač nemůže použít výsledek nebo vypnout používané API. Druhý druh vysílání je *Ordered broadcast*, který se odesílá pomocí *Context.sendOrderedBroadcast()*. Vysílání přijme pouze jeden přijímač. Potom se může poslat na další přijímač nebo může být vysílání v přijímači úplně zrušeno a nikam se neposílá [10]. Pořadí přijímačů je možné kontrolovat podle priority nastavené u každého přijímače v souboru *AndroidManifest.xml*.

Kompletní dokumentaci a více informací o *Broadcast Receiver* lze nalézt na stránce *https://developer.android.com/reference/android/content/BroadcastReceiver.html* [10].

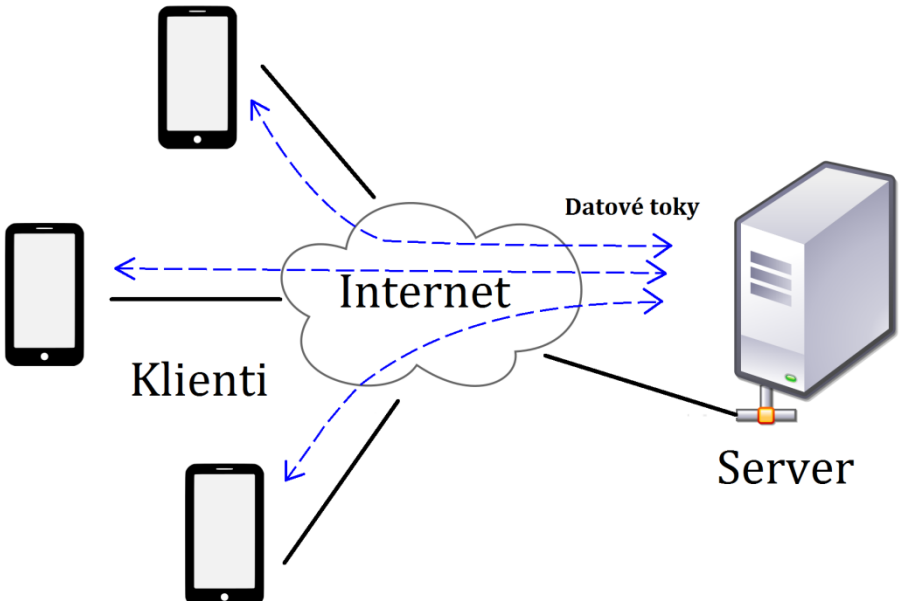
2.8 Testování aplikací

Při testování vlastních aplikací máme dvě možnosti, emulátor a reálné zařízení. Emulátor je obsažen v *Android SDK*, takže není potřeba stahovat další balíčky a programy [21]. Stačí vytvořit nové virtuální zařízení a zapnout. Většina aplikací pracuje stejně, jak na virtuálním, tak na reálném zařízení, ale existují výjimky, například přijímání hovorů nebo funkce bluetooth. Takové aplikace na virtuálním zařízení fungovat nebudou. U reálného zařízení je zapotřebí povolit na zařízení možnosti pro vývojáře a povolit ladění přes USB. Dále je nutné stáhnout *USB Driver* pro dané zařízení a nainstalovat jej na počítač.

3 APLIKACE PRO SDÍLENÍ POLOHY

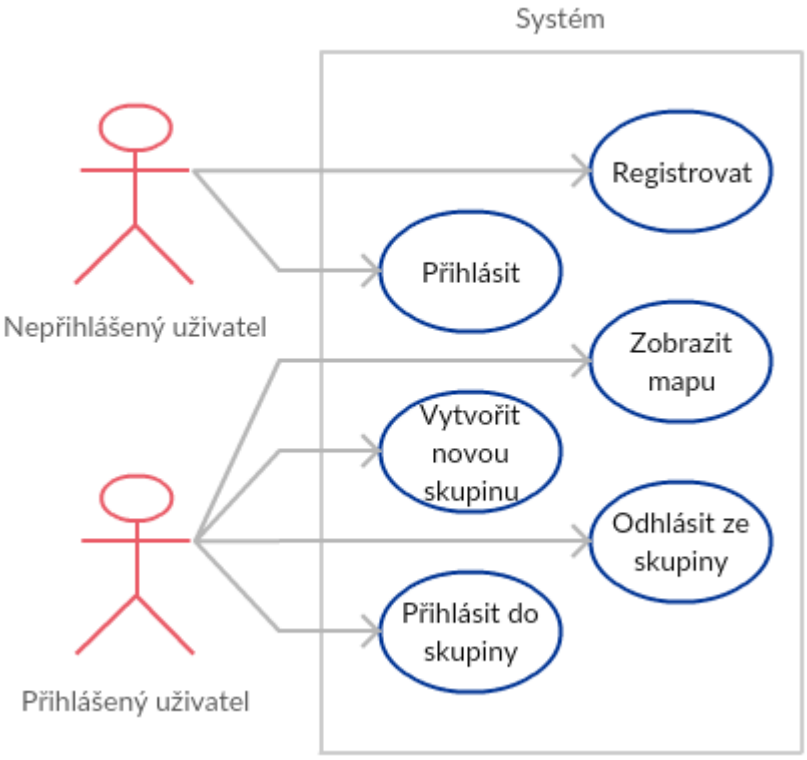
Tato část práce je věnována návrhu aplikace pro sdílení polohy a její následné implementaci. První kapitola je zaměřena na potřebné znalosti, které jsou nutné k vytvoření aplikace. Dále se zabývá výběrem vývojového prostředí a všech potřebných doplňkových programů a balíků, které jsou nutné pro vývoj a testování aplikace.

Hlavní myšlenkou aplikace je sdílet polohu mezi více uživateli, proto je nutné ukládat data na sdílené uložiště, ke kterému budou mít všichni přístup. Pro tuto funkci je zvolena databáze, která je přístupná z internetu pomocí architektury klient/server (Obrázek 5 - Architektura klient/server). Pokud má aplikace správně fungovat, musí být zařízení, na kterém je spuštěna, připojeno k internetu. Další kapitola je tedy věnována popisu této databáze a možnostem získávaní a ukládání potřebných dat.



Obrázek 5 - Architektura klient/server

Poslední kapitola se věnuje popisu samotné aplikace. Popisu tříd, aktivit a služby pro aktualizaci polohy. Věnuje se možnostem, které má uživatel aplikace. Hlavní funkcionality aplikace, které může uživatel využívat, jsou zobrazeny v use case diagramu aplikace (Obrázek 6 - Use case diagram).



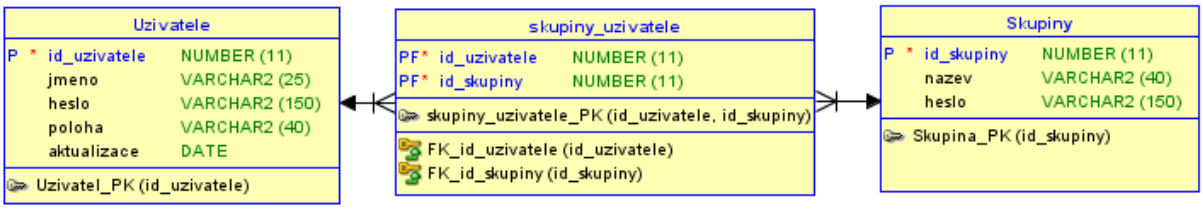
Obrázek 6 - Use case diagram

3.1 Příprava

K vývoji aplikace je zapotřebí mít znalosti programovacího jazyka Java, základy jazyka PHP a základní práce s databázemi. Dále je nutné mít nainstalovanou sadu nástrojů *Android SDK*. V této sadě nástrojů je třeba nainstalovat příslušné platformy, na které bude aplikace vyvíjena. Jelikož bude v aplikaci přítomna mapa, je nutno u platformy nainstalovat, kromě základních balíčků *SDK Platform* a *Samples for SDK* ještě balíček *Google API*. Dalším krokem k vývoji aplikace je instalace vývojového prostředí *Android Studio*, které je volně ke stažení na internetu stejně jako balík *Android SDK*. Dále je nutné se rozhodnout, jak budeme aplikaci testovat. Pro testování na reálném zařízení je nutné stáhnout a nainstalovat *USB Driver* pro určitý druh zařízení, povolit ladění přes USB a připojit zařízení k počítači v režimu PTP. Pro testování na virtuálním zařízení je nutné vytvořit virtuální stroj v *AVD manageru*.

3.2 Serverová část aplikace

Protože aplikace sdílí polohu mezi více uživateli, kteří jsou rozděleni do skupin, je nutné udržovat některé informace na úložišti, které je pro všechny přístupné. Adekvátním řešením je databáze uložená na serveru. Pro tuto aplikaci je využit server na freehostingovém portále Endora.cz [13]. Na serveru je vytvořena databáze, která uchovává informace o uživatelích a skupinách. U uživatele je uchováváno id uživatele, jméno, heslo, poloha a datum poslední aktualizace polohy. U skupiny se ukládá id, jméno a heslo. Hesla uložená v databázi jsou zašifrována pomocí metody PBKDF2 [18]. V databázi je vytvořena ještě tabulka *skupiny_uzivatele*, která obsahuje sloupce *id_uzivatele* a *id_skupiny*. Každému uživateli je zde přiřazen seznam skupin, ve kterých je přihlášen. Databáze se tedy skládá ze tří tabulek, kde každý uživatel může být přihlášen v N skupinách a každá skupina může být přiřazena M uživatelům. Mezi tabulkami *Uzivatele* a *Skupiny* je tedy relace M:N.



Obrázek 7 - Relační model databáze.

Na serveru je také uložený PHP skript, který zajišťuje komunikaci s databází. Tento skript se skládá z několika bloků, z nichž každý blok představuje jednu akci prováděnou s databází (ověření přihlašovacích údajů, vložení nového uživatele nebo skupiny, aktualizace polohy, atd.). Když potřebuje aplikace odeslat nebo přijmout data z databáze vytvoří HTTP požadavek na adresu, kde je PHP skript uložený. Podle GET parametrů připojených v požadované URL adrese se určí, který blok skriptu bude vykonán. Určení tohoto bloku probíhá pomocí parametru *action*. Podle požadované akce ve skriptu jsou potřeba další parametry, jako *id_uzivatele*, *id_skupiny*, *jmeno*, *heslo*, nebo *nazev*. Skript následně provede požadovaný blok a vypíše výsledek. Aplikace následně přečte a vyhodnotí výsledek bloku PHP skriptu.

PHP skript se připojuje k dané databázi pomocí objektu PDO (*PHP Data Object*), který nabízí metody k vytvoření a vykonání SQL dotazu [16]. Použití tohoto objektu zamezuje možnosti napadení databáze metodou SQL injection [17]. Tato metoda spočívá v napadení neošetřených vstupů. Uživatel může, například místo přihlašovacího jména, vepsat nebezpečný dotaz, který změní data nebo dokonce smaže tabulku v databázi.

Objekt PDO nabízí metodu *prepare(sql_query)*, která připraví dotaz, ve kterém jsou místo zadaných parametrů zástupné názvy (:jméno, :heslo). Vykonání dotazu se provede voláním *stmt->execute(array)*, kde v poli *array* jsou k zástupným názvům přiřazeny skutečné parametry [16]. Použitím tohoto postupu je zamezeno zneužití databáze pomocí SQL injection.

Příklad vytvoření objektu PDO:

```
$pdo = new PDO('mysql:host='.$servername.';dbname='.$dbname, $username,
$password);
```

Příklad URL adresy s parametry posílanými na server:

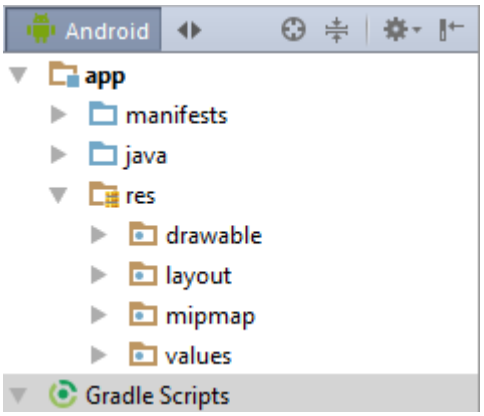
http://www.sledujpolohu.9e.cz/db_work.php?action=registrace&jmeno=jmeno&heslo=heslo

Příklad bloku PHP skriptu pro registraci nového uživatele s vytvořením dotazu a přidáním parametrů pomocí metod objektu PDO je uveden v Příloha A – *Příklad zdrojového kódu v PHP skriptu*.

PHP skript obsahuje dotazy pro přihlášení, registraci, výpis skupin, vytvoření skupiny, přidání skupiny k uživateli, aktualizaci polohy, načtení uživatelů a jejich polohy z určitých skupin, odebrání skupiny ze seznamu uživatele a výpis skupin přihlášeného uživatele.

3.3 Založení a orientace v projektu

Vytvoření projektu v *Android studiu* je velmi jednoduchá záležitost. Při spuštění programu stačí zvolit *Start a new Android Studio project*. Dále bude uživatel vyzván k zadání názvu projektu. Po kliknutí na *Next* je třeba vybrat nejnižší platformu, na kterou bude aplikace vyvíjena. Aplikace pro sdílení polohy bude vyvíjena na mobil nebo tablet a zvolená nejnižší platforma je třeba vybrat tak, aby pokryla co možná nejvíce zařízení. V tomto případě je zvoleno API 14, což znamená, že aplikace bude fungovat na systémech Android 4.0 a vyšších. [10]



Obrázek 8 - Základní složky projektu

Složky projektu lze zobrazit v několika rozděleních. Nejpřehlednější zobrazení pro programování je zobrazení Android, ve kterém se lze nejlépe orientovat ve složkách projektu a nalézt potřebné soubory. Projekt je rozdělen do několika hlavních složek, ve kterých je zapotřebí se orientovat. Základní dvě složky jsou *app* a *Grandle.Scripts*. Složka *Grandle.Scripts* obsahuje skripty pro sestavení aplikace a generování *apk* souboru. Ve složce *app* se nacházejí složky *manifest*, která obsahuje soubor *AndroidManifest.xml*. V tomto souboru jsou popsány jednotlivé komponenty aplikace (aktivity, service, …) [10]. Tyto komponenty budou popsány v dalších částech práce. Manifest dále obsahuje oprávnění pro přístup do chráněných částí API (*user-permissions*). V případě aplikace pro sdílení polohy je třeba připojit hned několik těchto oprávnění, a to například pro přístup na internet, zveřejňování polohy nebo přístup k síti.

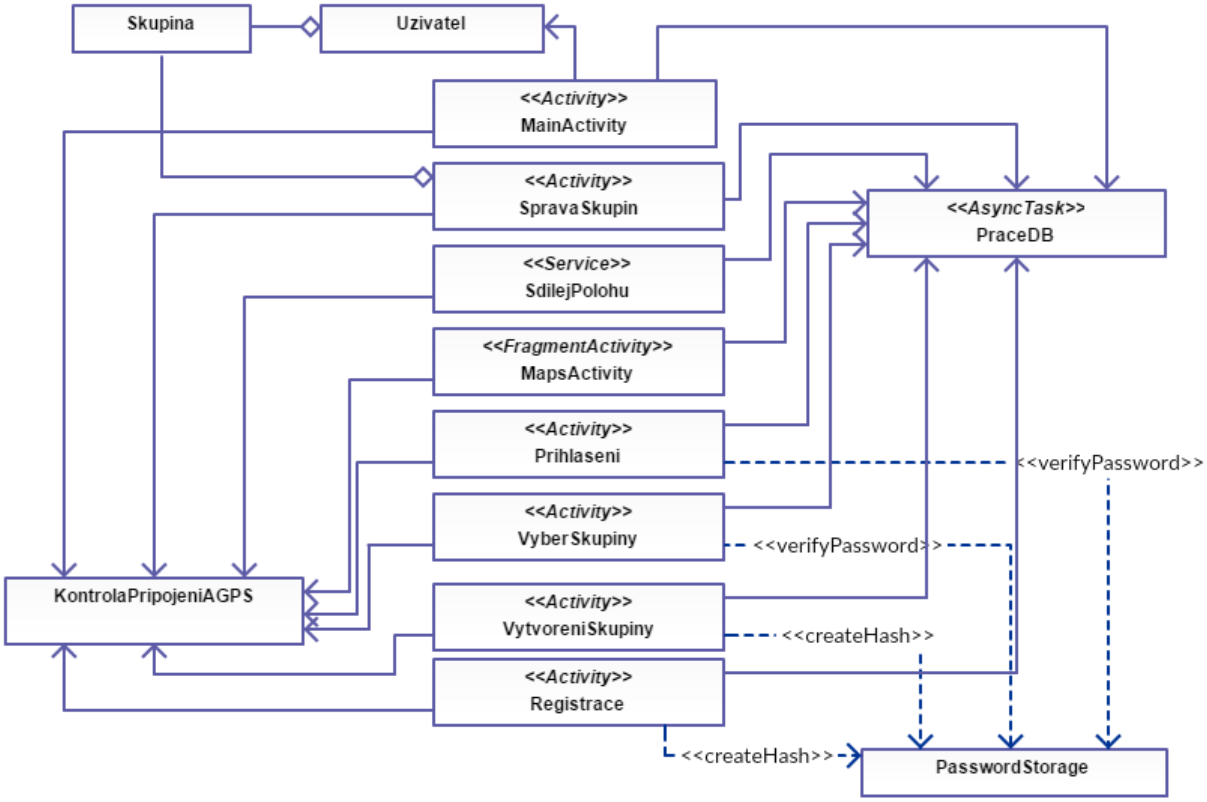
```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Obrázek 9 - Oprávnění vložená v souboru AndroidManifest.xml

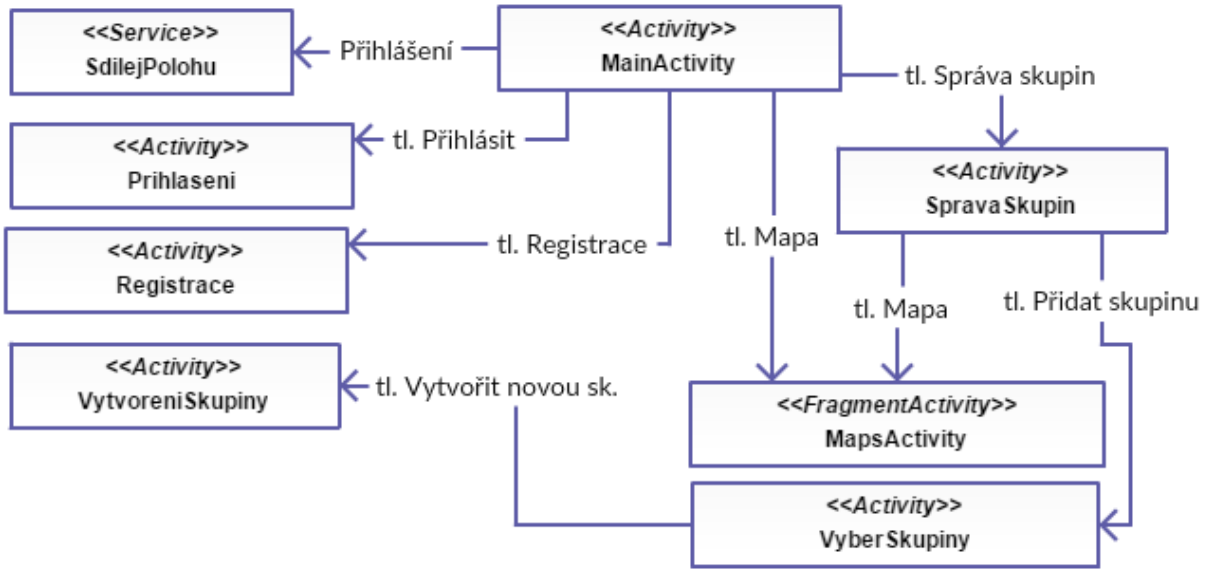
Další složkou je *Java*. Jak už název napovídá, v této složce se nachází naprogramované třídy, popřípadě také jejich testy. Poslední je složka *res*, ve které se nacházejí podsložky obsahující obrázky používané v aplikaci (*drawable*), grafické rozvržení jednotlivých obrazovek v XML souborech (*layout*), obrázek ikony aplikace (*mipmap*) a složka sloužící k uchovávání například řetězců z důvodu rychlejšího překladu aplikace (*values*). [10]

3.4 Hlavní třídy aplikace

Aplikace se skládá z několika obrazovek, jedné služby a několika dalších pomocných tříd, které budou v této kapitole popsány. Jednotlivé třídy a relace mezi nimi jsou zobrazeny na UML diagramu tříd (Obrázek 10 - UML diagram tříd). Jednotlivé obrazovky zobrazují tlačítka, které umožňují uživateli přecházet na jiné obrazovky. Možnosti těchto změn obrazovek jsou zobrazeny na diagramu přechodů mezi aktivitami (Obrázek 11 - Diagram přechodů mezi aktivitami).



Obrázek 10 - UML diagram tříd



Obrázek 11 - Diagram přechodů mezi aktivitami

3.4.1 Třída *PraceDB*

Tato třída slouží k propojení s databází. Je potomkem třídy *AsyncTask*, která slouží k provedení určité úlohy na pozadí a ve vlastním vlákně [2]. Jelikož v systémech Android je připojení na server moc náročná úloha, která v hlavním vlákně aplikace není povolena, je nutné tuto činnost odsunout do jiného vlákna. K tomu je využití třídy *AsyncTask* nejvhodnější řešení. V třídě je nutné implementovat metodu *doInBackground(String... arg)*. Jako vstupní parametr se do metody posílá část URL adresy s GET parametry, pomocí které se aplikace vyšle požadavek na server. Pomocí vloženého PHP skriptu se určí daná operace a vypíše výsledek, který aplikace zpětně přečte. Návratová hodnota metody je řetězec s výsledkem, se kterým se nadále pracuje v aplikaci. Následující ukázka kódu zobrazuje obsah metody *doInBackground*.

```
//Poslání požadavku na server a následné přečtení výsledku
URL url = new URL(link);
HttpURLConnection con = (HttpURLConnection) url.openConnection();
bufferedReader = new BufferedReader(new
InputStreamReader(con.getInputStream()));
String result = bufferedReader.readLine();
```

Pomocí *HttpURLConnection* je otevřena komunikace se serverem [2]. V PHP skriptu se provede určitý dotaz, podle zadaných parametrů a výsledek daného bloku skriptu je jednoduše vypsán pomocí příkazu echo. Následně je tento výsledek načten do aplikace pomocí vstupního proudu dat.

Pro spuštění metody, která pošle požadavek, je nutné v aplikaci vytvořit instanci třídy a zavolat metodu *execute(String)* s parametrem adresa. Tím se provede vytvoření nového vlákna a v něm provedení obsahu metody *doInBackground()*. Aby bylo možné zjistit výsledný řetězec, je nutné zavolat ještě metodu *get()*, pro vrácení výsledku metody *doInBackground()*. Ke zjištění výsledku dotazu je tedy nutné provést následující operace.

```
PraceDB db = new PraceDB();
String vysledekDotazu = db.execute(adresa).get();
```

3.4.2 Třída *KontrolaPripojeniAGPS*

Tato třída obsahuje dvě metody. Metoda *checkInternetConn(ConnectivityManager)* slouží ke kontrole připojení zařízení k internetu pomocí mobilního internetu, nebo přes wifi. Tato metoda je v aplikaci použita při každém připojení na server. Následující metoda *checkGPS(LocationManager)* slouží ke zjištění, zda je zapnuté určování polohy pomocí funkce GPS a v aplikaci je použita při spuštění. Při vypnuté funkci se zobrazí informativní dialogové okno, které říká, že aplikace s vypnutou funkcí GPS nebude aktualizovat polohu uživatele.

3.4.3 Třída *PasswordStorage*

Tato třída slouží k zašifrování hesla metodou PBKDF2. Její nejdůležitější metody jsou pro zašifrování a ověření hesla *createHash(String)* a *verifyPassword(String, String)* [18].

3.4.4 Třída *Uživatel*

Tato třída uchovává informace o uživateli, jako je id uživatele, jméno, heslo, poloha a seznam skupin. Instance této třídy jsou v aplikaci použity na dvou místech. První je statická proměnná, která udržuje informace o přihlášeném uživateli. Další je seznam uživatelů načítaný z databáze, který je následně zobrazen na mapě.

3.4.5 Třída *Skupina*

Třída obsahuje atributy id skupiny, název a heslo. Instance této třídy jsou vytvářeny při načítání skupin z databáze. Skupiny jsou následně ukládány například do seznamu skupin přihlášeného uživatele, nebo do komponent v různých aktivitách (*ListView*, *Spinner*).

3.4.6 Služba *SdilejPolohu*

Tato služba slouží ke zjištění a odeslání aktuální polohy přihlášeného uživatele do databáze. Třída je potomkem třídy *Service*, tím pádem je nutné, aby byla přítomna metoda *onBind()*. Pro příklad této aplikace je však tato metoda nevyužita a je implementována metoda

onStartCommand(Intent, int, int). Tato metoda je spuštěna vždy, když je v aplikaci zavolána metoda *startService(Intent)*. Parametr *Intent* je zpráva, která směřuje do třídy *SdilejPolohu*. Ve zprávě je přidána hodnota, která určuje, zda se má služba spustit nebo zastavit. V aplikaci je služba zastavována pomocí metody *stopSelf()* přímo zevnitř služby. Příklad spuštění a zastavení služby:

```
Intent i = new Intent(getApplicationContext(), SdilejPolohu.class);

//Spuštění služby
i.putExtra(INTENT_PARAMETER_NAME, CREATE_LOCATION_LISTENER);
startService(i);

//Zastavení služby
i.putExtra(INTENT_PARAMETER_NAME, DELETE_LOCATION_LISTENER);
startService(i);
```

Zjišťování polohy zajišťuje *LocationListener*, který je vytvořen při spuštění služby a zastaven při jejím zrušení. *LocationListener* obsahuje metodu *onLocationChange(Location)*, která je automaticky volána při změně polohy zařízení [2]. Parametr *Location* obsahuje nově zjištěnou polohu, která je odeslána do databáze a do statické proměnné přihlášeného uživatele. Ke zjištění souřadnic polohy slouží metody *Location.getLatitude()* a *Location.getLongitude()*. Spolu s polohou je obnoven i datum a čas poslední aktualizace polohy. *LocationListener* obsahuje další metody, které jsou volány například při výpadku funkce GPS a musejí být implementovány, i když pro tuto aplikaci nemají smysl. Mezi tyto metody patří *onStatusChange(String, int, Bundle)*, která je volána v případě, když je možno načíst polohu po delší době nedostupnosti této možnosti [2]. Další metody jsou *onProviderEnabled(String)* a *onProviderDisabled(String)*, které jsou volány při zapnutí, nebo vypnutí funkce GPS. Pokud aktualizace polohy běží a GPS bude vypnuto, tato vyvolá se tato metoda a aktualizace je pozastavena [10]. Potom je možné změnit způsob aktualizace polohy, nebo počkat až bude funkce GPS znova zapnuta [10]. Příklad spuštění *LocationListener*:

```
manager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
UPDATE_INTERVAL, MIN_DISTANCE, mLocationListener);
```

Proměnná *manager* je instance třídy *LocationManager*, která poskytuje přístup ke službám zjišťování polohy [10]. Příklad vytvoření proměnné *LocationManager*:

```
manager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

3.4.7 Aktivita *MainActivity*

Toto je úvodní obrazovka, která se zobrazí po spuštění aplikace. Po spuštění aplikace se zkontroluje připojení k internetu a funkce GPS. Pokud zařízení není připojeno k internetu, zobrazí se dialogové okno, kde je uživatel vyzván k připojení k internetu a potom je aplikace ukončena. Když v zařízení není zaplá funkce GPS, bude uživatel vyzván k jejímu zapnutí, ale aplikace ukončena nebude. Bez této funkce bude aplikace bezchybně pracovat, jenom nebude aktualizována poloha přihlášeného uživatele.

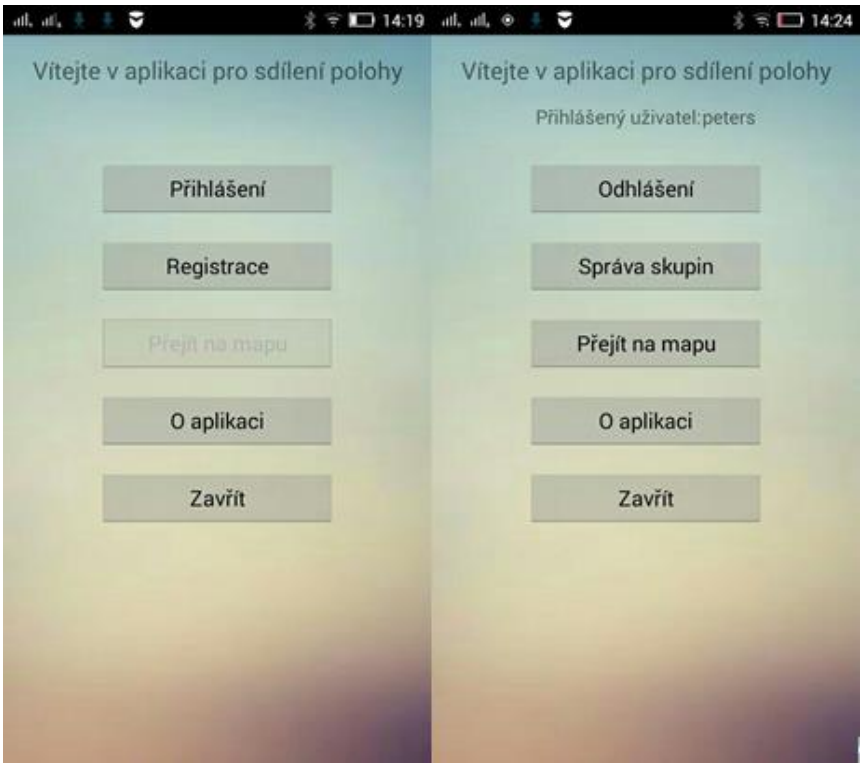
Jak už bylo řečeno v předchozích kapitolách, aplikace obsahuje oprávnění (*permissions*), které poskytují přístup ke chráněným částem systému. Ve starších verzích systémů Android (do verze 6.0, neboli API 23) stačí tyto oprávnění vepsat do souboru *AndroidManifest.xml*. Od verze systému Android 6.0 je však nutné tyto oprávnění navíc potvrdit při prvním spuštění aplikace na zařízení (*Runtime Permissions*). Po kontrole připojení k internetu tedy aplikace rozpozná, jaká verze systému je nainstalována na zařízení a podle toho, buď zobrazí dialogové okno pro potvrzení oprávnění, nebo bude aplikace pokračovat dál. Při odmítnutí oprávnění bude aplikace ukončena. [19]

Aplikace dokáže také rozpoznat naposledy přihlášeného uživatele. Tento problém je řešen pomocí *SharedPreferences*, což je třída, která dokáže udržet informace, ikdyž je aplikace úplně vypnutá a při jejím spuštění tyto data využít [10]. Pokud je tedy uživatel přihlášen a aplikace je vypnuta, uloží se do této proměnné data o uživateli. Při opětovném spuštění aplikace, jsou tyto data obnovena a uživatel je automaticky přihlášen. Když se uživatel rozhodne odhlásit, data z této proměnné jsou smazána. Každá sdílená preference je rozpoznána podle svého jména. Pro změnu obsahu této preference je nutné ji otevřít v módu editor, potom lze ukládat potřebná data. Uložení probíhá pomocí metod *editor.put(název,proměnná)*. Každá uložená proměnná je určena svým názvem, podle kterého jsou následně data načítána. Konečné uložení dat ve sdílené preferenci je uskutečněno pomocí metody *editor.commit()*. Příklad vytvoření a uložení dat do sdílené preference.

```
//uloženi přihlášeného uživatele
SharedPreferences.Editor editor = getSharedPreferences(MY_PREFS_NAME,
MODE_PRIVATE).edit();
// uloženi potřebných dat
editor.putInt(ID, uzivatel.getId());
editor.putString(JMENO, uzivatel.getJmeno());
editor.putString(HESLO, uzivatel.getHeslo());
editor.putFloat(LAT, (float) uzivatel.getLat());
editor.putFloat(LON, (float) uzivatel.getLon());
editor.putString(AKTUALIZACE, uzivatel.getAktualizace());
editor.commit();
```

Načtení dat ze sdílené proměnné neprobíhá v módu editor a používají se k tomu metody *get* (*pref.getString(**JMENO**, null)*) [10]. Pokud je uživatel přihlášen a jsou načteny data ze sdílené proměnné, je nutné načíst do seznamu skupiny tohoto uživatele. Načtená data i skupiny jsou nadále uloženy ve statické proměnné *uzivatel*.

Podle toho, zda je, nebo není uživatel přihlášený, budou zobrazena různá tlačítka, která může uživatel použít. Pokud není přihlášený, má uživatel možnost se přihlásit nebo registrovat. Po kliknutí na jedno z těchto tlačítek se vyvolá příslušná obrazovka registrace pomocí metody *startActivity(new Intent(MainActivity.this, Prihlaseni.class))* [2]. Když je uživatel přihlášený má místo tlačítek přihlášení a registrace na výběr se odhlásit nebo spravovat svoje skupiny. Další tlačítka jsou společná pro oba případy. Jedním z těchto tlačítek je Přejít na mapu, na které lze kliknout pouze, pokud je uživatel přihlášený. V opačném případě je kliknutí znemožněno. Další možnosti jsou tlačítka o aplikaci, při kterém se zobrazí dialogové okno s informacemi o aplikaci a tlačítko zavřít, které schová aplikaci na pozadí pomocí metody *finish()*.



Obrázek 12 - Hlavní obrazovka aplikace, pokud je/není přihlášený uživatel

3.4.8 Aktivita *Registrace*

Obrazovka nabízí uživateli možnost vytvořit nového uživatele. Potřebné informace k vytvoření je jméno, které musí být unikátní a heslo. K zapsání těchto informací a k registraci nového uživatele nabízí obrazovka příslušné komponenty.

Po otevření se objeví textová pole pro jméno, heslo a potvrzení hesla a tlačítko pro registraci. Když se klikne na tlačítko registrovat, tak se v aplikaci zkontroluje, zda jsou všechna pole vyplněna, a zda jsou vyplněná hesla shodná. Dále se zkontroluje připojení k internetu. Jak již bylo řečeno, tato kontrola probíhá při startu aplikace, ale je možně, že za běhu se připojení přeruší, nebo je přerušeno uživatelem. Jelikož je nutné data odesílat na server a existuje možnost ztráty připojení k internetu, je nutné ho před odesláním dat znovu zkontrolovat. Pokud jsou tyto podmínky splněny, je údaj hesla zašifrován pomocí metody PBKDF2 [18] a společně se jménem poslány s využitím třídy *PraceDB* na server.

V PHP skriptu je provedena část programu určená pro registraci nového uživatele, což obsahuje kontrolu, zda již uživatel s tímto jménem neexistuje. Pokud existuje, je aplikaci vrácená chyba a uživateli zobrazena hláška, že je třeba zvolit jiné přihlašovací jméno. Když jméno zatím neexistuje, je vytvořen nový uživatel. Potom je aktivita ukončena a je vyvolána aktivita pro přihlášení.

Obrázek 13 - Obrazovka pro registraci

3.4.9 Aktivita *Prihlaseni*

Zde je uživateli nabídnuta možnost se přihlásit a tím odesílat svoji polohu na server, prohlížet mapu s uživateli ve vybraných skupinách, či vytvářet vlastní skupiny. Aby se uživatel mohl přihlásit, musí být již registrován a do příslušných komponent na obrazovce vyplnit správné uživatelské jméno a heslo.

Po otevření se zobrazí textové pole pro jméno a heslo a tlačítko pro přihlášení. Když se klikne na tlačítko přihlásit, je nutné zkontrolovat několik věcí. Jelikož je nutné zadané jméno a heslo zkontrolovat s databází je nutné překontrolovat připojení k internetu. Dále následuje podmínka, která zjistí, zda jsou vyplněna textová pole. Pokud jsou tyto podmínky splněny, odešle se vyplněné jméno na server. Když uživatel s tímto jménem v databázi existuje, jsou jeho data (id, zašifrované heslo, poloha, aktualizace) odeslána zpět do aplikace, pokud neexistuje, je vrácen a zobrazen chybový řetězec oznamující neplatnost přihlašovacích údajů.

Dalším krokem, pokud uživatelské jméno v databázi existuje, je ověření hesla. K tomu je zapotřebí volat metodu *PasswordStorage.verifyPassword(password, hash)*. Této metodě se předá zadané heslo z textového pole a zašifrované heslo přijaté z databáze. Pokud metoda vrátí *false*, je také zobrazena chybová hláška oznamující neplatnost přihlašovacích údajů. Za stavu, kdy uživatelské jméno existuje a ověření hesla proběhne v pořádku, jsou uživatelská data (id, jméno, zašifrované heslo, poloha, aktualizace) uložena do statické proměnné *uzivatel*. Pro případ neočekávaného pádu aplikace jsou data také pro jistotu uložena do sdílené preference. Pokud všechny tyto operace proběhnou v pořádku je uživateli sděleno, že je úspěšně přihlášen a kontext aplikace se vrací na hlavní obrazovku, ovšem již s přihlášeným uživatelem.

Obrázek 14 - Obrazovka přihlášení

3.4.10 Aktivita *SpravaSkupin*

Hlavní komponentou této obrazovky je seznam skupin, ve kterých je uživatel přihlášený. Dále je uživateli, prostřednictvím příslušných tlačítek, nabídnuto skupiny do seznamu přidávat nebo z něj odebírat. Uživatel také může prostřednictvím této obrazovky zobrazit mapu s polohou uživatelů ve svých skupinách.

Po vyvolání aktivity jsou inicializovány všechny komponenty. Je zde tlačítko pro přechod na mapu, které zobrazí mapu s uživateli ve skupinách. Dále tlačítko pro přidání skupiny do seznamu, které vyvolá další aktivity s formulářem pro přidání skupiny. Poslední tlačítko slouží k odebrání označené skupiny ze seznamu. Hlavní komponentou obrazovky je však prvek *ListView*, který zobrazuje skupiny uživatele. Skupiny jsou uloženy ve statické proměnné *uzivatel*, v proměnné typu seznam (*List*). Pomocí tohoto seznamu je vytvořen adaptér a ten je nahrán a zobrazen v komponentě *ListView* [1]. Příklad zobrazení skupin v komponentě:

```
listSkupinUzivatele = MainActivity.uzivatel.getSkupina();

ArrayAdapter<Skupina> adapter = new ArrayAdapter<Skupina>(this,
android.R.layout.simple_list_item_single_choice, listSkupinUzivatele);

listSkupiny.setAdapter(adapter);
```

K vytvoření adaptéru jsou nutné tři věci. Kontext aplikace, což určuje, na které obrazovce se cílová komponenta nachází. V tomto případě je to aktuální instance třídy. Dalším důležitým parametrem je způsob (layout) zobrazení jednotlivých prvků a možnost práce s nimi. Zvolený layout je *simple_list_item_single_choice*, který zobrazuje na každém řádku jednu skupinu a pouze jedna skupina lze označit kliknutím [10]. U každé skupiny je přepínač, který ukazuje, která skupina je označená. Tento mód je v aplikaci zvolen z důvodu odebírání skupin ze seznamu. Po kliknutí na tlačítko odebrat skupinu aplikace zjistí, zda je označená nějaká skupina a pokud ano, tak je odstraněna ze seznamu a proběhne zobrazení aktualizovaného seznamu do komponenty *ListView*. Jelikož má uživatel přidělené skupiny i v databázi je nutné toto spojení s vybranou skupinou smazat i tam. Na server je tedy odesláno id uživatele a skupiny a z tabulky *skupiny_uzivatele* je smazán příslušný řádek, který spojuje uživatele se skupinou.

Obrázek 15 - Obrazovka pro správu skupin

3.4.11 Aktivita *VyberSkupiny*

Tato obrazovka uživateli slouží hlavně k přidání skupiny do jeho seznamu. Obrazovka obsahuje rozevírací nabídku s dostupnými skupinami a textové pole pro vyplnění hesla skupiny. Dále obsahuje tlačítko pro vytvoření nové skupiny a tlačítko pro přidání skupiny do seznamu.

Po kliknutí na tlačítko vytvořit novou skupinu se vyvolá nová aktivita, které bude věnována následující kapitola. Další komponentou na obrazovce pro přidání skupiny je *Spinner* (rozevírací seznam), který zobrazuje všechny volné, dosud vytvořené skupiny. Volné znamená, že tyto skupiny uživatel zatím nemá ve svém seznamu. Po vyvolání aktivity je nutné nahrát tyto volné skupiny do *spinneru*. Skupiny se načítají z databáze, proto je nutné zkontrolovat připojení k internetu a až potom načítat skupiny. Načítaná data (id_skupiny, název a zašifrované heslo), jsou nadále ukládána do proměnných třídy *Skupina* a ukládána do seznamu skupin (*List*). V tomto seznamu jsou dále skupiny abecedně seřazeny podle názvu a nahrány do rozevíracího seznamu. K nahrání je zapotřebí nejprve vytvořit adaptér, který se následné aplikuje na *spinner*. Zdrojový kód pro nahrání skupin do rozevíracího seznamu [1].

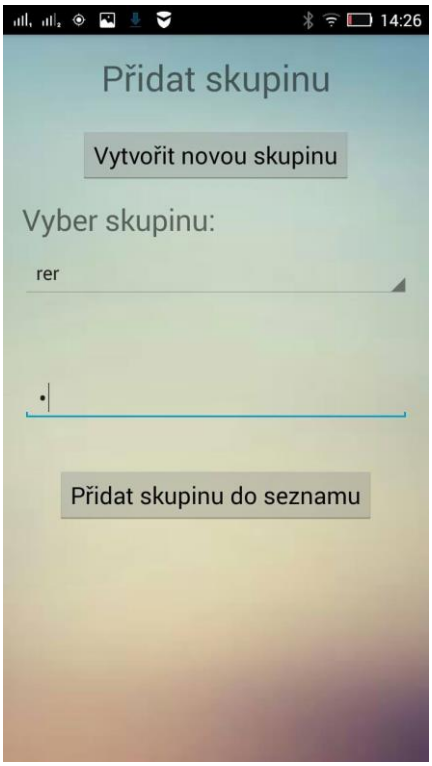
```
ArrayAdapter<Skupina> dataAdapter = new ArrayAdapter<Skupina>
        (this, android.R.layout.simple_spinner_item, vsechnySkupiny);

dataAdapter.setDropDownViewResource
        (android.R.layout.simple_spinner_dropdown_item);

spin.setAdapter(dataAdapter);
```

Dalšími komponentami na obrazovce je textové pole pro vyplnění hesla vybrané skupiny a tlačítko pro přidání skupiny do seznamu uživatele. Po kliknutí na tlačítko přidat je zkontrolováno, zda je vyplněné textové pole pro heslo a zda je zařízení připojené k internetu. Když jsou tyto kontroly v pořádku, je nutné ověřit správnost hesla. Aplikace z rozevíracího seznamu získá vybranou skupinu se všemi jejími daty a zavolá metodu *PasswordStorage.verifyPassword(heslo, hash)* [18]. Do metod je odesláno heslo, které uživatel vyplnil a zašifrované heslo získané z vybrané skupiny.

Pokud je heslo ověřeno, je skupina přidána do seznamu skupin uživatele. Tuto změnu je nutné zaevidovat i do databáze. Na server je tedy odesláno id uživatele a právě přidané skupiny, kde příslušný blok scriptu přidá nový záznam do tabulky *skupiny_uzivatele*.

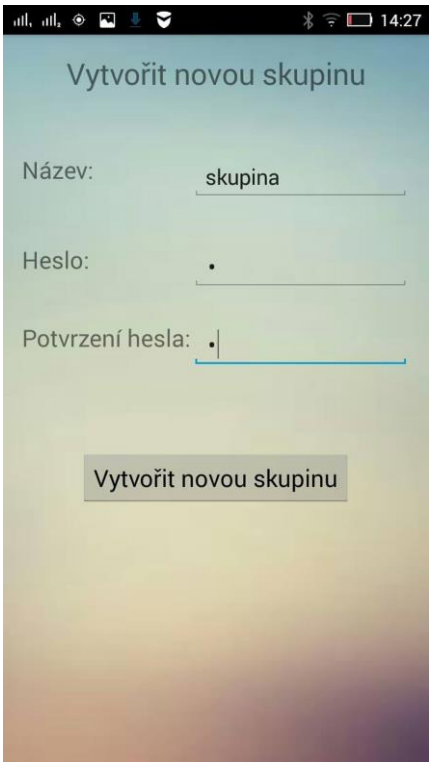


Obrázek 16 - Obrazovka přidání skupiny

3.4.12 Aktivita *VytvoreniSkupiny*

Jak už název napovídá, hlavním úkolem této obrazovky je možnost vytvoření nové skupiny a její vložení do databáze. Tato aktivita je zobrazena po kliknutí na tlačítko vytvořit novou skupinu na obrazovce pro přidání skupiny. K vytvoření nové skupiny je potřeba zadat unikátní název a heslo. Pro tyto údaje jsou v aktivitě vloženy potřebné komponenty.

Po otevření se objeví textová pole pro zadání názvu, hesla a potvrzení hesla skupiny a tlačítko, které skupinu vytvoří. Po kliknutí na tlačítko se zkontroluje, zda jsou všechna pole vyplněna, a zda jsou obě zadaná hesla stejná. Pokud jsou podmínky splněny, heslo se zašifruje a společně s názvem se odešle na server. Tyto údaje zpracuje příslušný blok v PHP skriptu, který zkontroluje, jestli skupina s tímto názvem zatím neexistuje. Když existuje, pošle zpět do aplikace chybovou hlášku a uživatel je o situaci informován. Pokud je vše v pořádku, je nová skupina vytvořena a aktivita je ukončena. Kontext se vrací na aktivitu pro přidání skupiny do seznamu, v jejímž rozevíracím seznamu je nyní přítomna i nově vytvořená skupina.



Obrázek 17 - Obrazovka pro vytvoření nové skupiny

3.4.13 Aktivita *MapsActivity*

Tato obrazovka je žádaný výsledek aplikace. Zobrazuje mapu a na ní značky s uživateli, kteří jsou přidáni ve stejných skupinách jako přihlášený uživatel. Po kliknutí na značku (*Marker*) je zobrazen popisek se jménem uživatele a časem jeho poslední aktualizace polohy. Tato značka je zobrazena na poloze daného uživatele. Pro přihlášeného uživatele je zvolena červená barva značky a pro ostatní uživatele barva zelená.

Při vytváření této aktivity je nutné udělat několik kroků. Pro zobrazení mapy na obrazovce není možné vytvořit klasickou aktivitu, která dědí ze třídy *Activity*. Je nutné vytvořit *Google Maps Activity*, která zobrazí mapu [15]. Aby bylo možné google mapu zobrazit ve své vlastní aplikaci, je nutné získat *API key*. K získání tohoto klíče je nutné zaregistrovat aplikaci do *Google API Console*. Po zaregistrování je vygenerován řetězec znaků, který je nutný zkopírovat do aplikace, do souboru *AndroidManifest.xml*. Je nutné klíč obalit do tagu <*meta-data*>. Místo, kam je nutné kód zkopírovat, je označen komentářem, který je vygenerován při vytvoření aktivity. Přidání *API key*:

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="@string/google_maps_key" />
```

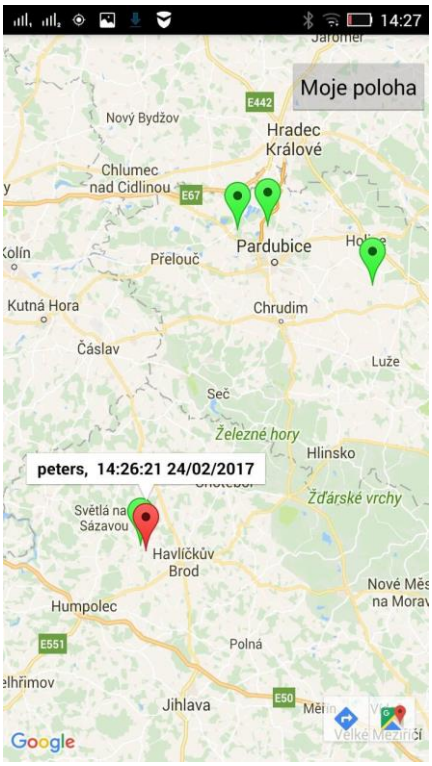
Po zadání klíče se po vyvolání této aktivity zobrazí mapa. Ještě před zobrazením mapy jsou spuštěny dvě defaultní metody a to metoda *onCreate()* a *onMapReady()*. V první uvedené metodě je získán mapový fragment. Tato část kódu je automaticky vygenerována při vytvoření aktivity. Dále je zde vytvořen *Handler*, který zajišťuje zobrazení uživatelů na mapě. Druhá zmíněná metoda slouží k získání objektu *GoogleMap*. Tento objekt je získán v momentě, kdy je mapa připravena k použití a je potřebný k vykreslování například značek (*Marker*) atd. Dále je v této metodě spuštěn *Handler*, který zobrazuje uživatele.

V *handleru* je opakovaně, po určitém časovém intervalu, volána metoda, která načte a zobrazí uživatele na mapě. Na server je odesláno identifikační číslo přihlášeného uživatele a podle toho jsou do aplikace vráceny jména, polohy a časy poslední aktualizace všech uživatelů, kteří jsou přihlášeni ve stejných skupinách, jako přihlášený uživatel. Před jejich zobrazením na mapě je ovšem nutné vyřadit stejné záznamy, které mohou být načteny, když jsou uživatelé přihlášeni ve stejných skupinách. *Handler* vypnut při schování aplikace na pozadí, protože není třeba mapu aktualizovat, když není vidět mapa. Zastavení handleru je tedy implementováno v metodě *onStop()*. Při opětovném zobrazení mapy je handler znovu spuštěn v metodě *onRestart()*. Příklad zdrojového kódu handleru:

```
Runnable mStatusChecker = new Runnable() {
    @Override
    public void run() {
        try {
            nahraj();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            e.printStackTrace();
        } finally {
            mHandler.postDelayed(mStatusChecker, mInterval);
        }
    }
};
```

Zastavení a spuštění handleru probíhá pomocí metod *mStatusChecker.run()* a *nHandler.removeCallbacks(mStatusChecker)*.

Dále tato aktivita obsahuje tlačítko Moje poloha. Po kliknutí se kurzor mapy přesune na polohu přihlášeného uživatele.



Obrázek 18 - Mapa s uživateli.

4 ZÁVĚR

Hlavním cílem této práce bylo zhotovit funkční aplikaci pro zařízení s operačním systémem Android, která uživatelům umožní sdílet mezi sebou svoji polohu v reálném čase. Dalšími úkoly bylo nadále popsat operační systém Android vzhledem k vývoji aplikace, což je uvedeno v první polovině práce a popsat samotnou aplikaci a její funkcionality, které jsou popsány v druhé části. Druhá část práce dále uvádí způsoby implementace jednotlivých částí aplikace, ať už se jedná o grafické ztvárnění aplikace nebo vnitřní implementaci důležitých pasáží aplikace. Je zde také detailně popsána serverová část aplikace, která je zapotřebí ke sdílení polohy mezi více uživateli.

Aplikace byla testována na několika různých zařízeních s různými verzemi systému Android. Při testování jsem nenarazil na chyby nebo nedostatky, které by omezovaly použití nebo funkčnost aplikace. Všechny funkcionality pracovaly tak, jak bylo požadováno, a nenarazil jsem na situaci, kdy by aplikace bezdůvodně přestala pracovat. To ovšem nevylučuje, že tato situace na některém zařízení nemůže nastat.

Na této aplikaci by se dále mohlo pracovat na vylepšení grafického rozhraní. Další navrhované úpravy by spočívaly hlavně v úpravě layoutů a chování jednotlivých aktivit pro různé velikosti displejů s využitím fragmentů. Samozřejmě se nabízejí i další rozšíření jako například ukládání oblíbených míst, kde se nacházím, ale to už nekoresponduje se zadáním práce.

Aplikace by mohla mít v reálném životě poměrně značné využití. Mohli by jí využívat například starostliví rodiče, kteří chtějí hlídat svoje potomky na cestě do a ze školy. Dále by mohla mít využití u zaměstnavatelů, kteří chtějí všude hlídat svoje zaměstnance. Další využití a také jeden z důvodů proč aplikaci vyvíjím, je pro zábavu, abychom se mohli sledovat se skupinou přátel.

5 POUŽITÁ LITERATURA

[1] MEDNIEKS, Zigurd R. *Programming Android*. Beijing: O'Reilly, 2012, 12, 542 pages. ISBN 9781449316648.

[2] HARWANI, B. M. *Android Programming Unleashed*. Indianapolis: Sams Publishing, 2012, 696 p. ISBN 9780133151749.

[3] *Android* [online]. 2014 [cit. 2017-04-26]. Dostupné z: https://www.android.com/

[4] Android slovník: Otevřená komunita pro otevřenou platformu. *Androidforum.cz* [online]. [cit. 2016-11-13]. Dostupné z: http://slovnik.androidforum.cz/

[5] ZAVŘEL, Roman. IOS a Android vymazaly Windows Phone ze světa. In: *Letem světem Applem: Magazín o společnosti Apple a produktech Apple* [online]. [cit. 2017-03-29]. Dostupné z: https://www.letemsvetemapplem.eu/2016/08/19/ios-android-vymazaly-windows-phone-ze-sveta/

[6] KOČÍ, Mirek. Podíl Androidu Nougat nepřekročil ani procento. In: *Svět aplikací: Vše o aplikacích pro chytré telefony a tablety* [online]. [cit. 2017-03-29]. Dostupné z: http://svetaplikaci.tyden.cz/podil-androidu-nougat-zatim-neprekrocil-procento/

[7] HODLOVÁ, Lenka. *Web o operačním systému android* [online]. 2012 [cit. 2016-11-13]. Dostupné z: http://home.zcu.cz/~hodlova/

[8] Jak vypadá Android uvnitř aneb co je ROM, kernel, bootloader a další? In: *Android Market: Stránky věnované operačnímu systému Android* [online]. [cit. 2017-03-29]. Dostupné z: http://androidmarket.cz/android/jak-vypada-android-uvnitr-aneb-co-je-rom-kernel-bootloader-a-dalsi/

[9] Android software development. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-03-29]. Dostupné z: https://en.wikipedia.org/wiki/Android_software_development

[10] *Android Developers* [online]. [cit. 2017-03-13]. Dostupné z: https://developer.android.com/index.html

[11] SEMECKÝ, Vojtěch. Android Studio: nové vývojové prostředí. In: *Zdroják: o tvorbě webových stránek a aplikací* [online]. Devel.cz Lab [cit. 2017-03-29]. ISSN 1803-5620. Dostupné z: https://www.zdrojak.cz/clanky/android-studio-nove-vyvojove-prostredi/

[12] JAMAL, Eason. Android Studio 2.0. In: *Android Developers Blog* [online]. [cit. 2017-03-29]. Dostupné z: https://android-developers.googleblog.com/2016/04/android-studio-2-0.html

[13] *Endora.cz: Freehosting, webhosting - neomezeně domén, multihosting* [online]. [cit. 2017-03-13]. Dostupné z: https://www.endora.cz/

[14] NOVÁK, Lukáš. *Aplikace na objednávání jídel pro platformu Android* [online]. 2012 [cit. 2017-03-29]. Dostupné z: http://docplayer.cz/1072156-Prirodovedecka-fakulta-univerzity-palackeho-katedra-informatiky-aplikace-na-objednavani-jidel-pro-platformu-android.html

[15] Google Maps Android API. *Google developers* [online]. [cit. 2017-03-13]. Dostupné z: https://developers.google.com/maps/documentation/android-api/

[16] Práce s Mysql v PHP: Použití ovladače PDO. In: *Itnetwork.cz: Ajťácká sociální síť a materiálová základna pro C#, Java, PHP, HTML, CSS, JavaScript a další.* [online]. [cit. 2017-03-29]. Dostupné z: http://www.itnetwork.cz/php/ostatni/php-prace-s-mysql-pouziti-ovladace-pdo

[17] What is SQL Injection (SQLi) and How to Fix It. *Website security: keep in check with Acunetix* [online]. 2017 [cit. 2017-04-26]. Dostupné z: https://www.acunetix.com/websitesecurity/sql-injection/

[18] HORNBY, Taylor. Salted Password Hashing: Doing it Right. In: *Code Project: For those who code* [online]. 1999 [cit. 2017-03-29]. Dostupné z: https://www.codeproject.com/Articles/704865/Salted-Password-Hashing-Doing-it-Right

[19] Everything every Android Developer must know about new Android's Runtime Permission. In: *The Cheese Factory: Blog* [online]. [cit. 2017-04-07]. Dostupné z: https://inthecheesefactory.com/blog/things-you-need-to-know-about-android-m-permission-developer-edition/en

[20] HLAVÍK, Jiří. Android Programování: Životní cyklus a nový projekt. In: *Itnetwork.cz: Ajťácká sociální síť a materiálová základna pro C#, Java, PHP, HTML, CSS, JavaScript a další* [online]. [cit. 2017-03-29]. Dostupné z: http://www.itnetwork.cz/java/android/tutorial-programovani-pro-android-v-jave-zivotni-cyklus-a-novy-projekt

[21] KONEČNÝ, Matěj. Dej Androidu tablety!. In: *Zdroják: o tvorbě webových stránek a aplikací* [online]. Devel.cz Lab [cit. 2017-03-29]. ISSN 1803-5620. Dostupné z: https://www.zdrojak.cz/clanky/dej-androidu-tablety/

6 PŘÍLOHY

6.1 Příloha A – *Příklad zdrojového kódu v PHP skriptu*

```
//REGISTRACE NOVÉHO UŽIVATELE
if($action == "registrace"){
	$jmeno = $_GET['jmeno'];
	$heslo = $_GET['heslo'];
	$sql = "SELECT count(*) as pocet from uzivatel where jmeno=:jmeno";
	$stmt = $pdo->prepare($sql);
	$stmt->execute(array(':jmeno' => $jmeno));
	$row = $stmt->fetch(PDO::FETCH_ASSOC);
	if($row["pocet"] == "0"){
		$sql = "INSERT INTO uzivatel (jmeno, heslo, poloha) VALUES
		(:jmeno, :heslo, '50;15')";
		$stmt = $pdo->prepare($sql);
		if ($stmt->execute(array(':jmeno'=> $jmeno,':heslo'=>$heslo))){
			//Uživatel byl vložen do databáze.
			echo "ok";
		}else{
			//Uživatel nebyl vložen do databáze.
			echo "error";
		}
	}else{
		//Uživatel s tímto jménem již existuje.
		echo "exist";
	}
}
```