Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Využití Google Awareness na platformě Android

Plzeň 2021 Radek Müller

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 28. června 2021

Radek Müller

Abstract

The present bachelor thesis aims to investigate Awareness API created by Google. The knowledge gained in the research will be used for implementation of the application in Android platform which will employ the Awareness API. The thesis comprises a theoretical analysis where themes such as options for development on Android platform, the Awareness API itself and, lastly, the existing apps already using the API are dealt with. Further, the practical part, its main concern being to develop an application and to describe the process of its implementation follows the previous analysis. Possible extensions of the application along with the testing scenarios examining the functionality of the application – but chiefly the functionality and reliability of the Awareness API itself – can be found in the conclusion.

Abstrakt

Tato bakalářská práce si klade za cíl prozkoumat Awareness API vytvořené společností Google. Následné znalosti poté budou využity k implementaci aplikace pro platformu Android, která bude Awareness API používat. Práce se skládá z teoretického rozboru, kde jsou prozkoumány možnosti vývoje na platformu Android, samotné Awareness API a již existující aplikace, které API využívají. Praktická část práce se zaobírá navržením aplikace a popisem její implementace. V závěru nalezneme možná rozšíření aplikace společně s testovacími scénáři, které prověřují funkčnost aplikace, ale především jak je funkční a spolehlivé Awareness API.

Poděkování

Rád bych tímto poděkoval svému vedoucímu bakalářské práce Ing. Ladislavu Pešičkovi za vedení, užitečné připomínky a především čas strávený konzultacemi nad touto prací.

Obsah

1 Úvod

V dnešní době skoro každý vlastní poměrně výkonný mobilní telefon, který má v podstatě neustále u sebe. Díky tomu se mobilní aplikace přizpůsobují a snaží se nám pomáhat v souladu s naším okolím nebo aktivitou, kterou zrovna provádíme. Takové aplikace jsou velmi známe a užitečné, například Google Maps, která v závislosti na místě, kde se nacházíme, dokáže správně navigovat na místo určení. Jiné sportovní aplikace jsou schopny změřit délku naší běžecké trasy. Podobných aplikací jistě najdeme mnoho, ale ve většině případů nám zajišťují funkcionalitu jedné kontextuální akce. Právě tehdy přichází řešení od Google v podobě Awareness API[1], které si klade za cíl sjednotit tyto kontextuální akce (akce, které jsou závislé na našem okolí a kontextu ve kterém se zařízení nachází) do jednoho nástroje, který mohou vývojáři dále využívat a postavit na něm své aplikace. Aplikace budou díky tomu pro uživatele přívětivější a zároveň mu ušetří čas, jelikož aplikace sama rozpozná kontextuální akci. API nabízí možnost sledovat lokaci, aktivitu, stav zapojených sluchátek (zapojené, odpojené), čas a pokud se uživatel nachází v blízkosti beaconů (viz 4.1). Důležité ovšem je, jak tyto nabízené funkce obstojí v praxi v reálné aplikaci.

Cílem této práce je vytvoření mobilní aplikace v podobě asistenta, který pomocí senzorů a API dokáže rozpoznat v jakém se nachází prostředí a je schopen na něj příslušně reagovat. Uvnitř aplikace uživatel příliš času nestráví, jelikož se bude jednat spíše o nastavování hodnot a určování podmínek, za jakých má být uživatel na jejich splnění upozorněn. Aplikace dále bude běžet na pozadí a v případě splněných podmínek bude zobrazena notifikace s konkrétní zprávou. Jednou z plánovaných funkcionalit je například nákupní seznam v určitém obchodě - při přiblížení uživatele k danému obchodu, bude uživatel upozorněn, aby na nákup nezapomněl. Funkce podobného typu jsou užitečné pro uživatele, ale zároveň i pro vývojáře, kteří se o Awareness API doslechli a mají nápad pro využití ve svých aplikacích. Díky této aplikaci si budou moci vývojáři vyzkoušet, jak jednotlivé věci fungují.

[1] API - Application programming interface, programové rozhrání

2 Možnosti vývoje pro Android

V následující kapitole jsou rozebrány možnosti, jak lze vyvíjet na platformu Android. Je zde zohledněn převážně operační systém (dále jen OS) Android, kvůli tomu, že s Awareness API na iOS[1] pravděpodobně pracovat nelze, jelikož v dokumentaci o tomto OS žádná zmínka není.

2.1 Typ vývoje

Při rozhodování, jakým směrem se vydáme ve vývoji, můžeme volit nativní nebo multiplatformní možnost. Úskalí jednotlivých možností vývoje jsou rozebrána v následujících podkapitolách.

2.1.1 Nativní vývoj

Nativní aplikace je taková, která je vyvíjena na jednu platformu, jazykem, který je k tomu určený. Pokud se bavíme o Androidu, je tímto jazykem Java, popřípadě Kotlin a vývojovým prostředím Android Studio (viz 2.3). Konkurenční iOS využívá jazyk Objective-C nebo Swift.

Z výše uvedeného odstavce pramení výhody a nevýhody tohoto přístupu. Fakt, že aplikace jsou vyvíjeny na konkrétní platformu znamená, že aplikace budou rychlejší a mohou snadno využít všechny nabízené možnosti platformy. Díky tomu je jednoduché přistupovat k hardwarovým zařízením telefonu, jako je například kamera či GPS.

Vývoj pro specifickou platformu také znamená, že je využíváno uživatelské rozhraní dané platformy. Uživateli tudíž není neznámé to, jak aplikace vypadá a to, jakým způsobem se v ní pohybuje.

Pokud chceme nativní aplikaci pro Android i iOS, pochopitelnou nevýhodou je nutnost tutéž aplikaci naprogramovat ve více jazycích pro jednotlivé platformy. Z čehož vyplívají vyšší náklady na vývoj.

2.1.2 Multiplatformní vývoj

Známou platformou pro multiplatformní vývoj je Xamarin, vyvíjený společností Microsoft [2], který využívá jazyk C-sharp. Multiplatformní vývoj

[1] OS vyvíjený společností Apple

nabízí možnost vývoje nezávisle na platformě. Vše je díky tomu zastřešeno pod jedním společným zdrojovým kódem (tvořící například 90%) a není nutnost vytvářet více projektů pro jednotlivé platformy. Zbylé procento jsou nuance, které jsou specifické pro cílové platformy, jakožto například prvky uživatelského rozhrání.

Díky platformové nezávislosti je aplikace rychleji vyvíjená a zároveň cílí na větší množství uživatelů z jednotlivých platforem.

Mezi nevýhody patří již výše zmíněné nuance a také například doba, kterou je nutno čekat, než je zajištěna podpora pro nově vydanou nativní funkcionalitu.

2.2 Programovací jazyk

Na platformu Android je možno vyvíjet ve více programovacích jazycích, avšak rozebrány zde budou dva nejznámější, jimiž jsou Java a Kotlin.

2.2.1 Java

Jave je objektově orientovaný programovací jazyk, který byl vyvinut a představen společností Sun Microsystems roku 1995 a je momentálně třetím nejpoužívanějším jazykem na světě [12]. Java vychází z jazyků C a C++, ale liší se způsobem, jakým je program zpracováván. Překlad totiž neprobíhá do jazyka relativních adres, nýbrž do jazyka, který je nazván jako bajtkód. Díky tomu je jazyk oddělen od závislosti na konkrétním počítači. Soubory přeložené do bajtkódu jsou poté zaváděny do paměti počítače a následně interpretovány speciálními programy JVM[2] [16]. Odlišností od jazyka C je také vlastní správa paměti, kdy neuvolňuje paměť programátor, ale provádí to za něj *garbage collector*.

Android nepoužívá JVM ale vlastní ART[3].

2.2.2 Kotlin

Kotlin je objektově orientovaný, staticky typovaný programovací jazyk, který byl vyvinut společností JetBrains v roce 2011. Díky tomu, že je jazyk překládán, aby mohl být interpretován na ART, není problém, aby aplikace kombinovala jazyky Java a Kotlin. Není tedy nutné přepisovat Java kód do Kotlinu, ale pouze přidat další části. Jednou z výhod Kotlinu je, že pokud

[2]Java Virtual Machine

[3]Android runtime - běhové prostředí systému Android, které nahradilo Dalvik

není proměnná typu *nullable*, nelze do ní přiřadit hodnotu *null*, to vede k menšímu počtu výjimek a pádům aplikace, až o 20% [6].

2.3 Android Studio

Android Studio je oficiální IDE [4] pro vývoj aplikací na Android a lze jej stáhnout pro platformy Windows, Linux a macOS. Android Studio obsahuje Android SDK[5], které obsahuje nezbytné nástroje nejen pro vývoj, ale i například pro testování[18].

Aplikace na Android jsou vyvíjeny na širokou škálu zařízení, kde se mimo hardwaru mění také velikost displeje. Proto existuje emulátor, který je obsažen v Android Studiu, díky kterému můžeme vytvořit virtuální přístroje (AVD[6]), kde můžeme vybrat různou verzi Android OS a také například velikost obrazovky.

2.4 Shrnutí

Nativní vývoj umožňuje použití Awareness API bez problému, pochybnost může vyvstat u platformy Xamarin. Ovšem i tam je možné Awareness API použít s využitím NuGet (správce balíčku pro .NET aplikace) [3].

Platformou pro tuto práci je pouze Android, proto bude aplikace vyvíjena nativně.

Android Studio podporuje jak Kotlin, tak Javu. Google spolupracuje s firmou JetBrains a snaží se z Kotlinu udělat číslo 1. na vývoj Android aplikací [7]. Přesto jsem se rozhodl pro Javu, jelikož moje zkušenosti s Kotlinem jsou nulové.

[4]IDE - Integrated Development Environment
[5]SDK - Software Development Kit - Sada vývojových nástrojů
[6]AVD - Android Virtual Device

3 Aplikace využívající Awareness API

Následující kapitola popisuje dvě vybrané aplikace, které jsou k dispozici zdarma na Google Play (obchod s aplikacemi pro platformu Android). Tyto aplikace byly v Google Play nalezeny na základě jejich popisu, kde je řečeno, že nějakým způsobem využívají Awareness API. Jakým způsobem je popsáno dále.

3.1 Conscient - Context Aware app

Aplikace umožňuje spustit libovolnou další aplikaci, pokud jsou splněny jisté podmínky. Tyto podmínky jsou určeny na základě aktivity nebo prostředí, ve kterém se uživatel s telefonem nachází. Právě pro definici a vyhodnocování podmínek je využíváno Awareness API.

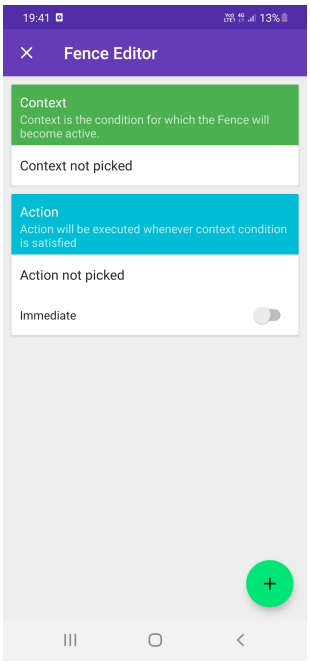
Uživatel vybere Context (podmínky) (viz 3.1a) a aplikaci, kterou chce spustit ve chvíli, kdy jsou podmínky splněny. Aplikace se po splnění podmínek spustí a zároveň je zobrazena notifikace s informací, která podmínka byla splněna.

V přehledu aplikace pak vidíme, které podmínky jsou s jakou aplikací spjaty a také, kdy byly naposledy splněny (viz 3.1b).

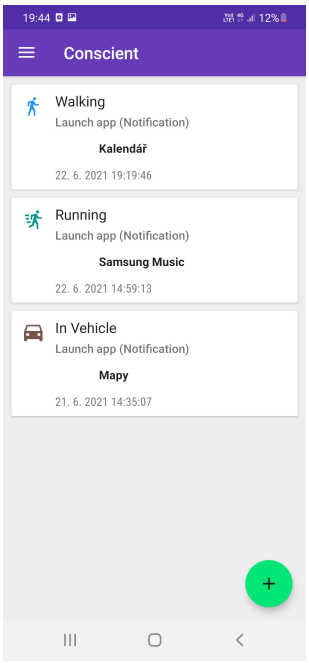
3.2 Vortex - Data Driven Live Wallpaper

Jak už z názvu aplikace vyplývá, jedná se o tapetu na pozadí telefonu, která se mění v čase v závislosti na datech, které jsou získávána pomocí Awareness API.

Aplikace umožňuje prohlédnout si aktuální stav měnící se tapety a také ji nastavit jako tapetu telefonu. Uvnitř aplikace nalezneme kartu s názvem "Awareness Data". Tato položka nám umožňuje nahlédnout na hodnoty, které jsou poskytovány Awareness API. Na základě těchto hodnot je poté modifikován vzhled měnící se tapety. Tlačítkem "REFRESH DATA"je možné aktualizovat hodnoty. Mezi položkami najdeme mimo jiné Activity - nabývá hodnot "still / walking / running", nebo třeba Coordinate - nabývá hodnot souřadnic, kde se s telefonem nacházíme (viz 3.2).

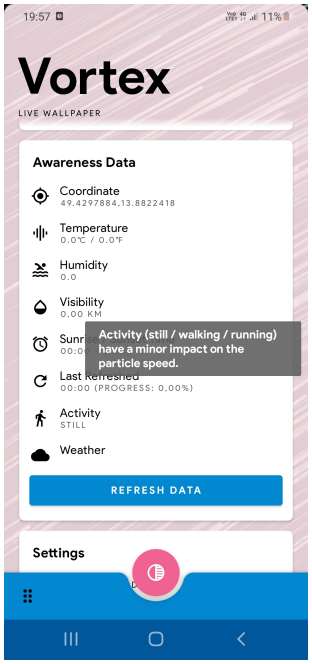


(a) Výběr kontextu a aplikace



(b) Seznam aplikací a podmínek

Obrázek 3.1: Aplikace Conscient



Obrázek 3.2: Aplikace Vortex

4 Analýza Awareness API

Awareness API bylo představeno v roce 2016 společností Google s nabídkou rozpoznávání 7 různých kontextových akcí [13]. Pro svoje fungování zastřešuje další dvě API a těmi jsou Fence API a Snapshot API.

Data v podobě signálů získaných ze senzorů umístěných v zařízení, jsou pomocí algoritmů zpracovávána pro co nejpřesnější výsledky. Tyto signály je možné kombinovat a zajistit tak reakci na detailnější popis situace, ve které se uživatel nachází. API zároveň bere v potaz spotřebu baterie a snaží se jí zmírnit na nezbytně nutné minimum. Společnost Google už podobné API vytvořila, například Geofencing API [5], které dokáže po přidání do projektu zjistit lokaci zařízení a následně na ní reagovat. Druhým příkladem je Activity Recognition API, které zjistí uživatelovu aktivitu, například zda uživatel s telefonem běží (viz 4.1.5). Problém ovšem nastává, pokud požadovaná aplikace má za cíl kombinaci těchto API dohromady. Kombinaci musíme totiž naprogramovat samostatně. V tom smyslu, že máme různá data (zpracované ze senzorů zařízení) od různých API a až naše aplikace musí zařizovat to, jak spolu API budou spolupracovat. Pokud bychom přidávali další API, uživatel nainstalované aplikace poté může čelit problémům s rychle klesající baterií nebo zpomaleným výkonem mobilu. Tyto problémy za nás řeší Awareness API.

4.1 Seznam kontextových akcí

Kontextovou akcí se rozumí událost, která je rozpoznána na základě dat, které získá zařízení ze senzorů telefonu. Akce je možné získávat jak ve Fence API, tak i v Snapshot API, liší se v tom, jakým způsobem akce získáváme (viz 4.3 a 4.2). Následuje popsání jednotlivých kontextuálních akcí.

4.1.1 Location

Aktuální poloha zařízení, možnost získání zeměpisné šířky, zeměpisné délky a také nadmořské výšky. Díky tomu je možné reagovat na to, kde se uživatel se zařízením zrovna nachází.

4.1.2 Headphones

Stav o zapojených či odpojených sluchátkách. Rozpoznávání funguje pouze na drátová sluchátka, na Bluetooth sluchátka nereaguje. Nefunkčnost Bluetooth sluchátek popsána v dokumentaci Awareness API není, tvrzení vychází z mé vlastní zkušenosti, kdy jsem zkoušel jedny Bluetooth sluchátka se dvěma telefony. Awareness API nezaznamenalo připojení ani odpojení sluchátek na žádném z telefonů.

4.1.3 Beacons

Informace o tom, zda se uživatel nachází poblíž určitého beaconu. Beacon (v překladu maják), je malé zařízení, které v krátkých intervalech vysílá signál Bluetooth, který je možné, například pomocí mobilního telefonu, zachytit. Beacony jsou využívány převážně ve vnitřních prostorech, jejich dosah je v řádu desítek metrů. Výhodou využití beaconů pro určování lokace je skutečnost, že zapnutý bluetooth signál spotřebovává méně baterie telefonu, než například GPS [19].

4.1.4 Time

Informace o aktuálním čase, podle toho, v jaké časové zóně se uživatel se zařízením nachází. Vhodné využití spíše ve Fence API (viz 4.3), kde je možné nadefinovat interval, podle kterého bude aplikace příslušně reagovat. Takový interval může být například každý den v určitou hodinu.

4.1.5 Activity

Aktivita zařízení je sdružena s číslem přesnosti rozpoznání [15]. Toto číslo značí s jak velkou pravděpodobností API odhaduje aktivitu (například 70/100). Typy aktivit jsou popsány v tabulce 4.1.

4.1.6 Place

Tato akce ode dne 30.10.2019 nefunguje [13]. Google nabízí alternativu prostřednictvím Places SDK, které dovoluje získávat informace o veřejně známých místech. Lze získat například obrázky z místa, nebo detailnější informace jako je například otevírací doba místa. To by bylo možné využít v kombinaci s Awareness API a upozorňovat tak uživatele, který bude blízko obchodu kde by chtěl něco pořídit, pouze pokud by byl tento obchod otevřený.

Typ	Popis
IN_VEHICLE	Zařízení je v dopravním prostředku, například auto.
ON_BICYCLE	Zařízení je u uživatele, který jede na kole.
ON_FOOT	Zařízení je u uživatele, který běží nebo jde.
RUNNING	Zařízení je u uživatele, který běží.
STILL	Zařízení se nehýbe.
TILTING	Zařízení značně změnilo svůj úhel oproti působení gravitace.
UNKNOWN	Nelze určit aktivitu zařízení.
WALKING	Zařízení je u uživatele, který jde.

Tabulka 4.1: Jednotlivé aktivity.

4.1.7 Weather

Tato akce ode dne 31.1.2020 nefunguje [13]. Jednalo se o možnost získání aktuálního počasí v místě, kde se zařízení nacházelo. V současné době Google nenabízí žádnou alternativu. Alternativa, která není od Google může být AccuWeather API [17].

4.2 Snapshot API

Pomocí Snapshot API je možné získat aktuální informace o kontextových akcích zařízení, když si to aplikace vyžádá. Tyto informace jsou ukládány do paměti RAM pro zvýšení rychlosti. Jednotlivé kontextové akce mají svoje přidružené metody, které jsou k tomuto účelu určeny. U detekce aktivit je poté možnost získání aktivity s nejvyšším číslem přesnosti rozpoznání, nebo si vyžádat seznam aktivit s jejich jednotlivými přesnosti, například: typ akce: ON_FOOT, číslo přesnosti 65/100.

4.3 Fence API

Fence API vychází z Geofencing API, kde definujeme lokaci spolu s jejím rádiusem a pokud se zařízení do tohoto ohraničeného "plotu"dostane, je zavolán callback a my v aplikaci můžeme reagovat. Fence API je tedy jakousi nadstavbou Geofencing API, jelikož tento koncept se dá přenést i na další kontextové akce.

Všechny tyto akce vrací výsledek ve formě `true` nebo `false`. Díky tomuto faktu je možné jednotlivé fence kombinovat Booleovými operátory OR, AND a NOT. Z toho nám také vychází dělení fenců na primitivní a kombinované

[14], kde primitivní může být například zda uživatel má zapojená sluchátka a kombinované zda uživatel má zapojená sluchátka a k tomu se nachází v určitém místě.

4.4 Nutná povolení

Následující tabulky 4.2 a 4.3 ukazují, pro jaké metody/typ fence jsou nutná povolení přidělené uživatelem pro správné fungování Awareness API. V tabulkách 4.2 a 4.3 je také ukázáno, že ve verzi Android 9 a níže je povolení pro rozpoznání aktivity lehce rozdílné.

Od Android verze 10 (API level 29), je možnost přidělit oprávnění o poloze aplikaci, která běží na pozadí. Pomocí dialogového okna uživatel zvolí možnost "Allow all the time", která je nezbytná pro správné fungování Awareness API. Tím je verze 10 specifická, jelikož tato možnost ve vyšších verzích Android OS již není a je nutné navigovat uživatele do nastavení aplikace se záložkou o povoleních dané aplikace.

Metoda	Vyžadované povolení
getDetectedActivity()	android.permission.ACTIVITY_RECOGNITION com.google.android.gms.permission. ACTIVITY_RECOGNITION(<=Android 9 (API level 28))
getBeaconState()	android.permission.ACCESS_FINE_LOCATION
getLocation()	android.permission.ACCESS_FINE_LOCATION

Tabulka 4.2: Nutná povolení pro Snapshot metody.

Typ fence	Vyžadované povolení
DetectedActivityFence	android.permission.ACTIVITY_RECOGNITION com.google.android.gms.permission. ACTIVITY_RECOGNITION(<=Android 9 (API level 28))
BeaconFence	android.permission.ACCESS_FINE_LOCATION
LocationFence	android.permission.ACCESS_FINE_LOCATION

Tabulka 4.3: Nutná povolení pro typy fence.

4.5 Způsob fungování Awareness API

Nejprve je nutné přidat API do projektu. Awareness API je součástí Google Play Services, je tedy nutné tuto službu stáhnout a nainstalovat do našeho projektu pomocí SDK Manageru ve vývojovém prostředí Android Studio. Dále je potřeba aktualizovat Gradle[1] závislosti. Poté je nutné přidat API key do manifestu naší aplikace a deklarovat nutná povolení, která zajistí správnou funkčnost API

Snapshot API funguje zavoláním příslušné metody, která nám poskytne aktuální informaci, s kterou dále můžeme pracovat a reagovat na ni. Můžeme například volání metody navěsit na tlačítko a získávat aktuální informace o právě prováděné aktivitě či lokaci kliknutím na toto tlačítko. Oproti Fence API zde nedefinujeme podmínky, které mají být splněny, ale získáváme aktuální informace o kontextuálních akcích.

U Fence API je třeba nejdříve vytvořit objekt typu `AwarenessFence`, kde definujeme o jaký typ fence se jedná, případně vytvoříme kombinaci fenců (viz 6.3). Nově vytvořenému objektu zvolíme jedinečný klíč pro jeho následnou identifikaci. Dále je potřeba naimplementovat `BroadcastReceiver`, kde získáme stav vyslané fence přes přijatý broadcast. Z tohoto stavu můžeme získat klíč a porovnat s nějakým naším jedinečným registrovaným klíčem u již existujících a nadefinovaných fenců. Pokud se nějaký klíč rovná našemu registrovanému klíči, ověříme, zda stav fence je požadovaný `true`, pokud ano, nyní jsou naše předešle definované podmínky fencu splněny a my můžeme příslušně reagovat.

[1]Gradle je plugin, který se stará o sestavování Android aplikací

5 Funkcionalita aplikace

V následujícím textu je popsáno, jaké funkce jsou plánované k implementaci. Dané funkcionality byly vybrány, aby ulehčily za pomoci Awareness API každodenní aktivity. Zároveň také k ušetření práce a času, jelikož uživatel nemusí mít telefon ani odemknutý k tomu, aby dostal notifikaci s příslušnými informacemi.

5.1 Nákupní seznam

Funkcionalita, která je spjata s lokací uživatele. V aplikaci plánuji uživateli umožnění přidání nové lokace pomocí souřadnic a také využití aplikace Google Maps k vybrání místa na mapě, což je rozhodně pohodlnější volba. Poté, co si uživatel uloží název lokace (obchodu), bude mít možnost lokaci smazat, či editovat její název. Po uložení zároveň bude umožněno přidávat jednotlivé položky nákupního seznamu. Uživatel bude také moci stávající seznam editovat (mazat a měnit). Uživatel bude mít k dispozici seznam svých lokací a po rozkliknutí se dostane k seznamu svého nákupu. Až se uživatel se zařízením dostane do kruhu okolo lokace definovaného rádiusem (hodnota bude volitelná, ale pro všechny lokace stejná) a nákupní seznam této lokace bude neprázdný, bude uživatel na tuto skutečnost upozorněn notifikací.

5.2 Todo seznam

Podobnou funkcionalitu spjatou s lokací plánuji rovněž se seznamem úkolů ke splnění. Uživatel zde bude mít rovněž možnost přidání lokace a k ní jednotlivé úkoly, které má v plánu splnit. Rovněž bude moci předem zadané lokace mazat a upravovat. Rozdíl je zde v seznamu jednotlivých úkolů. Zde bude vhodné, aby uživatel mohl mít možnost dalšího popisu, než jen názvu jako to je plánované u nákupního seznamu. Dalším rozdílem bude možnost zaškrtnutí tlačítka "done", aby uživatel mohl vidět úkoly, které již splnil, ale zároveň na ně nebyl při vstupu do lokace upozorňován.

5.3 Hudba

Pro reakci na připojená sluchátka bude možností přidání nového hudebního alba. Uživateli bude umožněno přidat nové hudební album s názvem, jmé-

nem interpreta a datem vydání jeho hudebního počinu. Všechny tyto položky bude opět možné editovat. Po připojení sluchátek bude uživatel upozorněn notifikací o novém albu, pokud sluchátka připojí po datu vydání alba. Aby uživatel nedostával notifikaci po datu vydání opakovaně a nemusel album mazat, bude přidáno zaškrtávací tlačítko "seen". Tím bude zajištěno, že uživatel nezapomene na alba, které již byla vydána.

5.4 Aktivita

V plánu reakce na aktivitu bude možnost získávání času, po jaký uživatel provádí aktivitu běhu, (viz 4.1, položka ON_FOOT). Uživatel pomocí tlačítka spustí svůj běžecký trénink a následně bude pomocí API rozpoznávána aktivita, zda uživatel běží nebo pouze jde. V této závislosti bude měřen čas aktivity. Pokud uživatelovo aktivita bude chůze, časovač nebude přidávat další sekundy, aby uživatel věděl čistý čas, kdy skutečně pouze běžel. Sekundy se tedy budou přičítat pouze, pokud uživatel poběží. Uživateli bude umožněno nastavit čas, který bude určovat, po jaké době bude uživateli ukončeno stopování tréninku a bude poslána notifikace. Pokud například uživatel nastaví 10 minut, zahájí trénink a jeho aktivita bude jiná než běh po dobu více jak 10 min v kuse, znamená to, že jeho trénink už je u konce a dostane notifikaci. V notifikaci jsou plánované informace o tom, jak dlouho běžel a také doporučení, aby doplnil tekutiny.

Problém nastal ve zkoušení a testování na dvou referenčních zařízeních (viz 9.2), zda jsou aktivity správně rozpoznávány. Je možné to rozdělit na rozpoznávání se Snapshot API a rozpoznávání pomocí Fence API.

Začínal jsem s testováním Snapshot API, které pošle rozpoznané výsledky u aktivit, pokud jej o to požádáme. Stejně tak je možnost získat seznam všech aktivit s přidruženým číslem přesnosti rozpoznání. První aktivitou k testování byla aktivita ON_FOOT, která je nadmnožinou a má se zobrazovat společně, pokud uživatel jde (WALKING) nebo také pokud běží (RUNNING). Vznikne nám tedy například dvojice ON_FOOT a k tomu RUNNING, které mají stejné číslo přesnosti. Problém ovšem je, jak moc přesné rozpoznávání je. Při testování Snapshot API jsem navěsil akci získání nejpravděpodobnějších aktivit a jejich následné zobrazení na obrazovku. Výsledky nebyly vůbec přívětivé a neshodovaly se s realitou. Pokud jsem testoval chůzi, častým problémem bylo, že doba na rozpoznání chůze mohla trvat až několik minut po reálném chození a také to, že chůze několikrát rozpoznána nebyla vůbec nebo s nízkým číslem přesnosti. To ale úplně vysoké být nemusí, dokud bude stále nejvyšší, ze všech rozpoznaných aktivit.

Pokud jsem se zaměřil na běh (RUNNING), problém v podstatě přetrvával a nebál bych se říct, že rozpoznání na tom bylo ještě hůře než u chůze. Pravděpodobně nejlépe rozpoznávanou aktivitou je aktivita STILL, kdy se telefon nehýbe. I přesto občas trvá dlouho (dle mého testování například i 4 minuty), než telefon rozpozná, že je ve stavu STILL.

Pokud jsem se zabýval testováním ve Fence API, bylo to velice obdobné. `BroadcastReceiver` často nebyl vůbec přijat. Dokonce jsem se dostal i do situace, kdy Snapshot API správně vrátilo aktivitu chůze s přesnosti 89/100, `BroadcastReceiver` přijal zprávu, ale stav fence byl `false` čili špatně vyhodnocen. U jízdy autem (IN_VEHICLE) je vyhodnocování také nekonzistentní a někdy je `BroadcastReceiver` přijat a správně vyhodnocen, jindy zase ne. Podobné problémy byly zaznamenány i u jiných vývojářů [9] [10].

Na základě zmíněných zkušeností není úplně vhodné používat Awareness API pro původní plán stopování běhu. Proto jsem se rozhodl pro nahrazení v podobě zobrazování seznamu aktivit s číslem pravděpodobnosti odhadu a s tlačítkem, které umožní obnovení a znovuzískání seznamu. Díky tomu uživatelé vidí data, která nejsou mnou blíže uprvována, pouze zobrazeny tak jak je poskytuje Awareness API.

6 Návrh aplikace

Cíl aplikace je, aby dokázala uživateli pomoci v závislosti na jeho lokaci, či jeho aktivitě. K tomu používá Awareness API, a díky tomu aplikace nese název Aware Assistant. Snaží se jednak být uživatelovým pomocníkem, například s nákupním seznamem (viz 5.1) a zároveň ukázat slabé a silné stránky Awareness API, což využijí hlavně vývojáři, kteří si chtějí vyzkoušet, jak API funguje a na kolik jsou jednotlivé součásti spolehlivé.

V následujícím textu jsou rozebrány nezbytné pojmy a komponenty pro fungování aplikace využívající Awareness API a dále jak tyto komponenty spolupracují s aplikací k dosažení výsledku.

6.1 Service

Service je komponenta, která provádí dlouhodobější úkony, které nepotřebují interakci od uživatele, čili neposkytuje žádné UI[1]. V tomto smyslu běží na pozadí, ale nevytváří nové vlákno[18]. Může dokonce běžet i v případě, že je aplikace, která Service spustila zavřená, či pokud je přepnuto na jinou aplikaci. Následující text osvětlí, jak lze Service rozdělit.

6.1.1 Foreground Service

Service, která stále zobrazuje notifikaci, díky které lze interagovat s aplikací, která Foreground Service spustila. Díky notifikace je uživatel upozorněn na to, že aplikace stále běží na pozadí. Pro představu můžeme Foreground Service použít například pro stahovaní souboru, kdy bude zobrazovat, kolik procent ke stažení daného souboru ještě zbývá.

Původně jsem zamýšlel, že v mé aplikaci bude Background Service (viz 6.1.2). Tím ale není zajištěna funkčnost, kterou jsem předpokládal a to je, aby Service běžela i když je aplikace zavřená po delší dobu, nebo je zavřená vymazáním ze seznamu naposledy otevřených aplikací. Android od API 26 a vyšší omezil používání procesů, které běží na pozadí, za účelem příjemnější uživatelské zkušenosti, jelikož tyto procesy spotřebovávají prostředky jako je paměť RAM a také baterii[4]. Pokud byla aplikace a její Background Service spuštěna a telefon byl následně zamknut, Service byla po nějaké době neaktivity zastavena. To ovšem u aplikace, která potřebuje uživatele

[1] User Interface - Uživatelské rozhraní

informovat, zda jsou splněny určité podmínky, které jsou závislé na kontextu, ve kterém se uživatel nachází, není žádoucí. Proto jsem zvolil Foreground Service a zároveň také proto, že v dnešní době je kladen stále větší důraz na to, aby uživatel věděl, jaké aplikace mu běží na pozadí v jeho mobilním telefonu. Uživatel je jasně informován, že aplikace běží a zároveň systém nechává Foreground Service běžet, proto je tento typ Service implementován v mé aplikaci (viz 7.3.7).

6.1.2 Background Service

Service, která nevyžaduje žádnou interakci s uživatelem a ani ho o svém působení nikterak neinformuje. Takové Service může sloužit například pro uspořádání dat v aplikaci. Skutečnost, že tento typ Service běží není pro uživatele na první pohled znatelný. Proto má tato Service vyšší omezení, jak je vysvětleno v předchozím odstavci.

6.1.3 Bound Service

Bound Service slouží k svázání komponenty aplikace jako je Aktivita se Service. Bound Service poté provádí svojí práci, dokud je s ní nějaká Aktivita svázaná. Bound Service si lze představit jako serverovou část v client-server architektuře [18]. Svázáním je myšleno to, že uživatel vytvoří můstek, po kterém může Aktivita a Bound Service komunikovat pomocí zasílání zpráv.

6.2 Broadcast Receiver

Broadcast je typ zprávy, která je poslána všem, kteří jí naslouchají. Jednotlivé aplikace se mohou registrovat k poslechu zpráv a následně na tyto zprávy reagovat. Aplikace zároveň mohou posílat vlastní broadcasty. Jinými slovy se jedná o systém posílání zpráv napříč aplikacemi. Součástí OS Android jsou také systémové broadcasty, které jsou posílány aplikacím, pokud nastane nějaké systémová událost, jako je třeba nízký stav baterie.

Jak již bylo zmíněno v kapitole 4.5, je nutné nadefinovat třídu, která bude dědit od třídy `BroadcastReceiver` k tomu, aby mohla přijímat vyslané broadcasty, pokud je nějaká podmínka předešle zaregistrované fence splněna. V aplikaci je udržovaný seznam fenců a jejich jedinečných klíčů, které se následně v této třídě porovnávají a hledá se, jestli nějaký z přijatého fencu není shodný s nějakým ze seznamu. Pokud ano, je odeslána příslušná notifikace.

6.3 Registrace fence

Pokud se jedná o Snapshot API, žádné registrování fenců nutné není, jelikož odešleme požadavek a na něj dostaneme příslušnou odpověď, zda nějaká kontextová akce nastává. Ale pokud chceme dostat zprávu, až když jsou nějaké podmínky fence splněny, znamená to nejdříve nutnost fence registrovat, až poté může `BroadcastReceiver` přijímat zprávy.

Registrace znamená nejdříve fence vytvořit. Vytvořením nadefinujeme o jakou fence (případně kombinaci fenců) se jedná (viz 4.3). Při registraci ještě navíc potřebujeme k samotné fence přidat unikátní klíč a také `PendingIntent`, skrze který může naše aplikace reagovat na změny stavů registrovaného fence. Nyní pomocí metod, které jsou součástí Awareness API registrujeme fence. Registrovat je možné i více fenců najednou a pokud registrujeme fence, jejíž klíč už je registrován, fence bude přepsána.

6.4 Notifikace

V aplikaci jako takové uživatel příliš času nestráví. Naopak smyslem aplikace je to, aby v ní uživatel pouze nastavil podmínky, na které bude následně upozorněn, i přesto, že aplikaci nebude mít otevřenou. Vhodným upozorněním na splnění podmínek je v tomto případě notifikace.

Notifikace je zpráva, která se objeví mimo uživatelské rozhraní aplikace. Zpravidla se zobrazuje v horní liště telefonu, ve stejném místě, kde se například zobrazuje i stav baterie telefonu. Po roztažení horní lišty máme možnost vidět dodatečné informace k notifikaci. Pokud je telefon zamčený, je možné notifikaci spatřit i na zamčené obrazovce. V této aplikaci bude vždy alespoň jedna notifikace, která je vždy vidět a ta indikuje, že Aware Assistant běží na pozadí. Ostatní notifikace se zobrazují jako reakce na splnění definované podmínky u fence. Například pokud se jedná o nákupní seznam, zpráva notifikace je, že máme produkty ke koupi spolu s názvem lokace a počtem produktů.

6.5 Ukládání a čtení ze souborů

Aplikace potřebuje ukládat pouze soubory s předem vytvořenými fenci a soubor pro uživatelské nastavení. Aplikace je navržena tak, že funguje se zdroji na konkrétním telefonu. Tudíž není nutný nějaký typ externí databáze.

Každý fence je nutné nějakým způsobem uložit, aby uživatel po restartování aplikace o tyto fence nepřišel a byly mu opět zobrazovány v příslušných

Aktivitách v podobě seznamu. Zároveň je také důležité, aby po vynuceném zastavení, kdy je zastavena i Foreground Service, apllikace po spuštění znovu registrovala fence, které se v souborech nacházejí.

Každý typ fence má svůj vlastní soubor, kde jsou uloženy všechny fence příslušného typu, které jsou registrované. Pro příklad lze uvést fence typu `TodoLocation`, který je ukládán do souboru *todoloc.fc*. Při spuštění jsou nejprve fence ze souborů načteny a následně registrovány. Pokud uživatel například přidá novou lokaci s nákupním seznamem, fence je registrována a zároveň zapsána do příslušeného seznamu. Pokud už zde takový fence je, je nejprve smazán, viz 7.3.5.

6.6 Oprávnění aplikace

Aby se uživatel cítil bezpečně a mohl mít kontrolu nad svým soukromím, existují v systému Android oprávnění aplikací. Oprávnění můžeme rozdělit na dva typy, jimiž jsou oprávnění v době instalace a oprávnění získávané za běhu aplikace. V prvním případě se jedná o oprávnění, které systém automaticky aplikaci schválí a jsou uživateli sdělena při instalaci aplikace z Obchodu Play, kde si je uživatel může přečíst. Ve druhém případě je uživatel tázán až v aplikaci, během jejího běhu, pokud zrovna dojde do bodu, kdy aplikace potřebuje pro své fungování oprávnění. Tento typ oprávnění je také považován jako nebezpečný, jelikož se může jednat o oprávnění, která zahrnují soukromé informace, jako je například lokace [8]. Pro svojí funkčnost je nutné oprávnění definovat v souboru `AndroidManifest.xml`.

Seznam vyžadovaných oprávnění pro správnou funkčnost aplikace je uveden v následujícím seznamu.

- ACCESS_FINE_LOCATION - Dovoluje aplikaci zjistit lokaci zařízení. Nutné pro funkčnost fenců spjatých s lokací, například nákupní seznam, který je spojen s lokací obchodu. Dále také pro Google Maps SDK.
- ACCESS_BACKGROUND_LOCATION - Dovoluje aplikaci zjistit lokaci zařízení, pokud aplikace běží na pozadí. Nutné, aby zároveň ACCESS_FINE_LOCATION bylo povoleno a aby fence dokázaly reagovat i v případě, že aplikace nebude běžet v popředí.
- ACTIVITY_RECOGNITION - Dovoluje aplikaci zjistit typ fyzické aktivity. Původně potřebné pro plánované aktivity spojené s během, nyní nutné pro zjištění seznamu aktivit (viz 5.4).

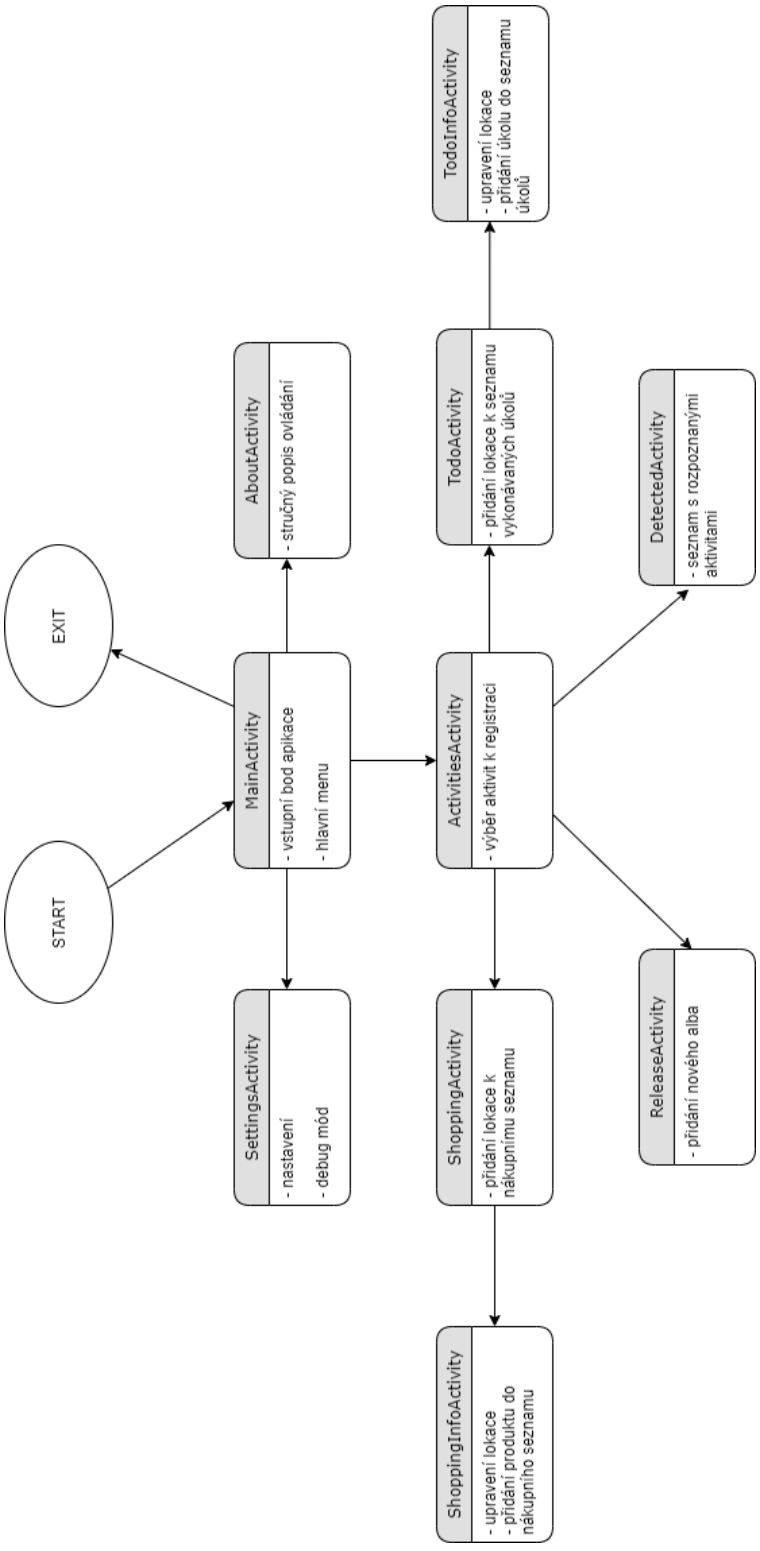
- FOREGROUND_SERVICE - Dovoluje aplikaci použít Foreground-Service, která je nutná pro základní fungování aplikace (viz 6.1.1).

6.7 Aktivity aplikace

Aktivitu si můžeme představit jako obrazovku mobilní aplikace, kterou zrovna vidíme a můžeme s ní manipulovat. Z toho vyplývá, že aktivita je nezbytnou součástí každé Android aplikace. Aktivita obsahuje grafické rozložení, které obsahuje prvky uživatelského rozhraní, díky kterým může uživatel aplikaci ovládat.

První co uživatel uvidí po spuštění aplikace Aware Assistant je `MainActivity`. Tato aktivita reprezentuje vstupní prvek a hlavní menu aplikace. Uživatel zde vidí několik tlačítek, skrze která se může pohybovat po aplikaci. Nejdůležitějším tlačítkem pro uživatele pravděpodobně bude tlačítko "activities", kde se následně může dostat k registraci jednotlivých typů fenců. Zvolil jsem zde název "activities", jelikož mi to přijde pro neznalého uživatele intuitivnější, než označení "fences". Pokud by uživatel nevěděl, co má v aplikaci provádět, pomocí tlačítka "about"na hlavní stránce zjistí nápovědu.

Aplikace se skládá z 10ti aktivit. Jak na sebe navazují a možnost pohybu po aplikaci je k nahlédnutí na obrázku (viz 6.1).



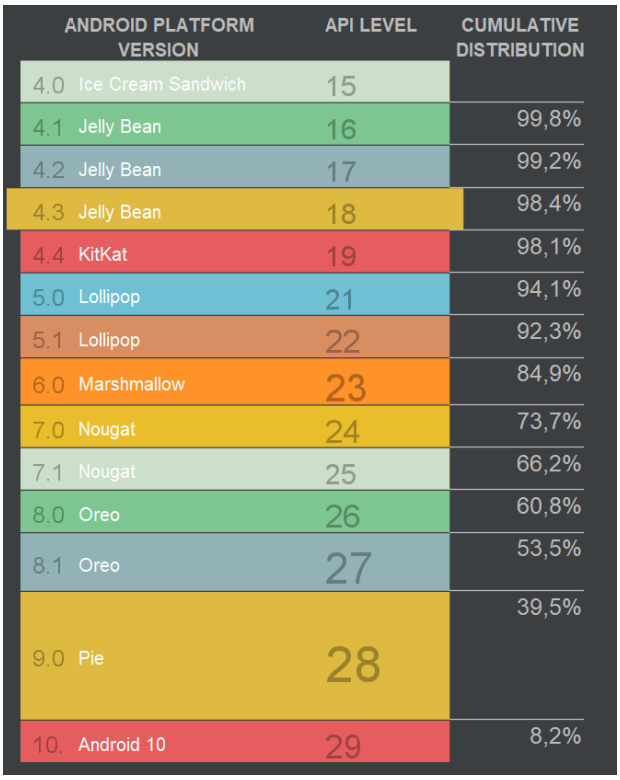
Obrázek 6.1: Architektura aplikace

7 Popis implementace

Aplikace je naprogramována v jazyce Java, důvody jsou vysvětleny v kapitole 2. Pro vývoj bylo využito prostředí *Android Studio v4.1.2*.

7.1 Minimální verze Android API

Zastoupení jednotlivých API na trhu je vidět na obrázku 7.1. Zvolené minimální Android API je 18 a to ze dvou důvodů. Prvním je, že pokud by aplikace v budoucnu využívala Beacon pomocí Snapshot API, je nutné, aby verze API byla 18 a vyšší [1]. Druhým důvodem je co nejširší možné publikum.



Obrázek 7.1: Architektura aplikace

7.2 Struktura projektu

Android Studio se postará o vygenerování nezbytně nutné struktury se všemi potřebnými soubory a umožní se v nich jednoduše orientovat. Pro vývoj je připraven modul s výchozím názvem *app*, do kterého ukládáme naše zdrojové kódy. Na disku je uloženo hodně složek a souborů, ale ty pro ukázku struktury nejsou příliš podstatné. Následující seznam ukazuje nejdůležitější složky a soubory, které najdeme ve složce s názvem modulu:

- **build/** - obsahuje výstupní soubory sestavení aplikace.
- **libs/** - obsahuje knihovny
- **src/main/java/** - obsahuje Java zdrojové kódy aplikace
- **src/main/AndroidManifest.xml** - soubor, který obsahuje informace o aplikaci jako jsou komponenty aplikace, oprávnění a další.
- **src/main/res/** - obsahuje zdroje, jako ikony, UI řetězce a soubory rozložení UI
- **src/test/** - obsahuje kód testů
- **build.gradle** - obsahuje konfigurace sestavení pro moduly

7.3 Třídy a balíky

Jednotlivé třídy jsou rozdělovány do balíků pro větší přehlednost zdrojového kódu a zapouzdření tříd, které spolu souvisí. V následujícím textu je popsané dělení a funkčnost jednotlivých balíků. Aplikace obsahuje celkem 31 tříd a 8 balíků (jeden sloužil k testovacím účelům a nebude zde popsán).

7.3.1 `activities`

Třída obsahující všechny Aktivity, které slouží k zobrazování jednotlivých obrazovek a k interakci s uživatelem. Celkem zde nalezneme 10 aktivit.

7.3.2 `adapters`

Balík, obsahující všechny třídy `Adapterů`, které dědí od třídy `ArrayAdapter<>`, kde je definováno jaké data se mají zobrazovat v jednotlivých `ListView` čili seznamech, jako je například nákupní seznam. Toho je docíleno pomocí metody `getView()`, kterou je nutno překrýt.

7.3.3 dialogs

Jak už z názvu vyplývá, v balíku nalezneme všechny dialogy, které aplikace využívá. Jednotlivé třídy dědí od třídy `AppCompatDialogFragment`. Pokud si pro příklad vezmeme lokaci, můžeme získat dva trochu jiné `LocationDialogy`, jelikož v jednom případě může jít o dialog, který edituje existující záznam, kde jsou `EditTexty` vyplněné dopředu, nebo může jít o dialog, který nový záznam vytváří. K tomu slouží překrytá metoda `onCreateDialog()`. Dále je zde definováno rozhraní, které je využíváno pro předání informací do Aktivity, z které byl dialog zobrazen.

7.3.4 fences

V balíku nalezneme třídy, které reprezentují fence a rozhraní `IFence`, které je implementováno každou třídou uvnitř balíku `fences`. Existuje zde jedna rodičovská třída `Location`, ze které dědí dvě třídy `ShoppingLocation` a `TodoLocation`. Třídy mají díky rozhraní implementovanou akci pro poslání notifikace nebo například pro získání jedinečného klíče a jejich další atributy. Například u `ShoppingLocation` takový atribut je `ArrayList` produktů.

7.3.5 io

Balík obsahující třídu obstarávající čtení a zapisování do souborů pro ukládání jednotlivých fenců nebo uživatelského nastavení. V této třídě je také implementována registrace fenců. Třída je jedináčkem (návrhový vzor singleton) a slouží také pro distribuování dat mezi aktivitami. Jsou zde totiž udržovány jednotlivé `ArrayListy` fenců, ke kterým poté aktivity přistupují pomocí getterů.

7.3.6 receivers

Balík, ve kterém leží také pouze jedna třída, jménem `Receiver`, která dědí od třídy `BroadcastReceiver` a musí tedy implementovat metodu `onReceive()`. V této metodě je přijatý fence získaný z přijatého `Intentu` porovnáván se seznamem registrovaných fenců. Shoda je definována klíčem. Pokud shoda nastane, je nutno ještě ověřit stav přijatého fence metodou `getFenceState()` a dle výsledku je odeslána notifikace.

7.3.7 `services`

Balík obsahuje třídu `ReceiverService` oddděnou od třídy `Service`. Tato Service je typu Foreground, důvody vysvětleny zde 6.1.1. Service vytvoří základní notifikaci, která je v ní užívána a spustí se. Dále je zde vytvořena instance `Receiveru` a následně registrována. Jako poslední jsou zde přečteny a registrovány fence ze souborů.

7.4 Google Maps SDK

Pro lepší uživatelský zážitek je v aplikace využito Google Maps SDK, které umožňuje integrovat do naší aplikace mapy, které jsou známe z aplikace Google Maps. Integrované mapy je poté možné ovládat stejnými gesty a také mimo jiné umožňují používat červené označení pro vybrání místa (tzv. marker), kam uživatel zrovna klikl prstem. To je vhodné pro výběr lokace v naší aplikaci, kde uživatel vybírá lokaci. Tímto při zadávání lokace nemusí psát souřadnice, ale jednoduše vybere místo na mapě, která se mu v aplikaci po rozkliknutí zobrazí. Následně dojde k vyplnění souřadnic v dialogovém okně, z kterého bylo k mapě přistupováno.

Dalším využitím Maps SDK je přidání tlačítka pro aktuální polohu, kdy uživatel klikne na příslušnou ikonku a díky tomu se kamera mapy přesune na polohu, kde se momentálně nachází.

Pro správnou funkčnost SDK musí uživatel přidělit aplikaci oprávnění o jeho poloze.

8 Možná rozšíření aplikace

V této kapitole budou popsány návrhy, jak by bylo možné dál aplikaci rozšiřovat.

8.1 Seznam todo úkolů

Aplikace nyní umožňuje vytvářet jednoduchý todo seznam úkolů, které jsou připomínány uživateli, pokud je v lokaci, která je s těmito úkoly spjata. Možným rozšířením by v tomto ohledu bylo například možnost vytvoření kategorií, ve kterých by se následně nacházely úkoly. Dále by zde bylo možné uvažovat otevření nové aktivity po kliknutí na úkol ze seznamu, kde by byl detailnější popis, než jenom možnost názvu a popsání úkolu. Bylo by takto možné udělat hierarchii úkolů, kde by uživatel mohl úkol rozdělit na jeho menší části.

8.2 Využití beaconů

Aplikace nevyužívá žádným způsobem beacony, beacon totiž nevlastním a co se týče lokace jsem se soustředil hlavně na získávání dat skrze GPS. Během testování jsem došel k závěru, že pro správné vyhodnocení fence, která je určená lokací je vhodná hodnota rádiusu přibližně 200 metrů. Což dělá z lokace poměrně velký prostor, ačkoliv by mohlo jít například o malý krám.

Beacony dokáží rozpoznávání s větší přesností. Proto by se hodily na případné rozpoznávání například uvnitř bytu v jednotlivých místnostech. Většina z nás se probouzí díky telefonu, tudíž ho mají hned od rána u sebe. Kdyby se poté například uživatel vydal do kuchyně, kde by měl nainstalovaný beacon, mohl by jeho telefon signál přijmout a vyslat jiný signál k automatické přípravě kávy, či podobnému nápoji.

8.3 Využití time fence

Čas ve zpracované aplikaci roli hraje, ale není využito časové fence. Aplikace s ním pracuje pouze, aby určila, zda album, které si uživatel uložil, již nebylo vydané (viz 5.3).

Jak bylo řečeno v kapitole o způsobu fungování Awareness API (viz 4.5), je možné definovat specifický interval, který se opakuje například během

pracovního týdne. To by bylo možné využít například s kombinací s todo seznamem, kde by uživatel eventuelně mohl přidat i časový rámec, během kterého by určité úkoly měl splnit. Další možností by mohlo být nastavení intervalů, kdy by uživatel měl jíst, kdyby se řídil podle určité diety.

8.4 Sluchátka a Spotify

V dnešní době se těší stále větší oblibě sluchátka, které pro přenos hudby používají technologii Bluetooth, s těmi ovšem Awareness API nepočítá. Stále je ale hodně lidí, kteří využívají klasická drátová sluchátka. Mezi uživateli je populární streamovací služba Spotify, kde je možné vytvářet playlisty s hudbou či podcasty. Podle dostupných zdrojů je proveditelné přehrání playlistu na Spotify pomocí Spotify SDK [11]. Díky tomu by uživatel mohl přehrávat hudbu pouze zapojením sluchátek.

8.5 Sportovní rozšíření

Pokud by se situace s Awareness API zlepšila co se týče rozpoznávání aktivit, bylo by možné také rozšířit aplikaci o můj původně zamýšlený plán ke stopování běhu, kdy by uživatel věděl, kolik času přesně uběhl. Pokud by uživatel do aplikace uložil svůj BMI [1], mohlo by mu být doporučeno, po jeho sportovní aktivitě, jaké doplňky stravy a v jaké míře jich má zkonzumovat.

[1]BMI - Body Mass Index - Index tělesné hmotnosti

9 Testování aplikace

Součástí práce je otestovat jak Awareness API a realizovaná aplikace fungují na různých zařízeních. Postupně je zde uveden výčet zařízení, na kterých testování probíhalo a následně pomocí jakých testovacích scénářů.

9.1 Testovací zařízení

Testování probíhalo na virtuálních a fyzických zařízeních s rozdílnými verzemi OS. Fyzická zařízení jsou k nahlédnu v tabulkách 9.1 a 9.2. Virtuální zařízení je specifikováno v tabulce 9.3. Vzhledem k tomu, že aplikace reaguje na okolí pomocí senzorů zařízení, virtuálním zařízením byla testována hlavně responzivita.

Samsung Galaxy A40	
Android OS	11 (API level 30)
CPU	Exynos 7 Octa 7404
RAM	4GB

Tabulka 9.1: Tabulka specifikací prvního fyzického zařízení

Samsung Galaxy J3	
Android OS	9 (API level 28)
CPU	Exynos 7570 Quad
RAM	2GB

Tabulka 9.2: Tabulka specifikací druhého fyzického zařízení

Nexus S	
Android OS	4.3 (API level 18)
Velikost displeje	4,0"
Rozlišení	480x800

Tabulka 9.3: Tabulka specifikací virtuálního zařízení

9.2 Testovací scénáře

Ve scénářích je popsána posloupnost dílčích úkolů a zhodnocení výsledků, zda aplikace reagovala dle očekávání.

9.2.1 Reakce na sluchátka

Test, který prověří, zda je správně registrovaný fence spojený s hudebním albem a dále, zda uživatel dostane notifikaci se správnými údaji.

Postup:

1. Kliknutí na tlačítko "activities"
2. Kliknutí na tlačítko "music"
3. Kliknutí na tlačítko "add release"
4. Vyplnění dialogového okna s názvem alba, jménem interpreta a datem vydání hudebního alba, které již proběhlo
5. Kliknutí na tlačítko "ADD"
6. Opakování kroku 3. - 5. 3x, kdy jednou v kroku 4., zvolíme nadcházející datum, podruhé po přidání zaškrtneme položku "seen"a potřetí vzniklý záznam vymažeme
7. Aplikaci zavřeme
8. Zapojíme drátová sluchátka

Po testu na obou dostupných fyzických zařízení se v liště objeví pouze první přidané album, což je korektní chování.

9.2.2 Nákupní seznam - editace

Test přidání nové lokace k nákupnímu seznam, její editaci a notifikaci na lokaci, která byla editována.
Postup:

1. Kliknutí na tlačítko "activities"
2. Kliknutí na tlačítko "shopping"
3. Kliknutí na tlačítko "add location"

4. Vyplnění položky "name"a vybrání lokace, ve které se nenacházíme, kliknutí na "ADD"
5. Vybrání nově vzniklé lokace ze seznamu
6. Přidání dvou produktů
7. Kliknutí na šipku zpět (registruje se fence)
8. Vybrání stejné položky jako v bodě 5.
9. Kliknutí na ikonku editu vedle názvu lokace
10. Upravení názvu lokace a souřadnic, ve kterých se nacházíme a kliknutí na "SAVE"
11. Kliknutí na tlačítko zpět

Test prošel na obou fyzických zařízeních, jelikož první fence notifikaci nezobrazila, druhá už ano a se správnou změnou v názvu.

9.2.3 Reakce na fence registrovaných za běhu aplikace

Test prověří, reakci na fence, které byly registrovány za běhu aplikace čili neregistrovaly se ze souboru při prvním spuštění aplikace.

Sluchátka

Zadání data, které již uplynulo a nezaškrtnutí položky "seen", aby notifikaci nic nebránilo k zobrazení.

Postup:

1. Kliknutí na tlačítko "activities"
2. Kliknutí na tlačítko "music"
3. Kliknutí na tlačítko "add release"
4. Vyplnění dialogového okna s názvem alba, jménem interpreta a datem vydání hudebního alba, které již proběhlo
5. Kliknutí na tlačítko "ADD"
6. Zapojení drátových sluchátek

Po testu se správně zobrazí notifikace s názvem alba a interpretem.

Lokace - nákupní seznam a todo seznam

Přidání lokací nákupního seznamu a todo seznamu. V todo seznamu nebude zaškrtnuta položka "done", která by bránila notifikaci k zobrazení.

Postup:

1. Kliknutí na tlačítko "activities"
2. Kliknutí na tlačítko "shopping"/"to do"
3. Kliknutí na tlačítko "add location"
4. Vyplnění položky "name"a vybrání lokace, která není v naší blízkosti (např. 300 m od nás), kliknutí na "ADD"
5. Vybrání nově vzniklé lokace ze seznamu
6. Přidání dvou produktů/úkolů
7. Kliknutí na šipku zpět (registruje se fence)
8. Zavření aplikace (volitelné)
9. Dostavení se na zadanou lokaci

Výsledky testu jsou různorodé. Ve většině případů, po dostavení se na zadanou lokaci, je notifikace u obou typů (todo seznam, nákupní seznam) správně zobrazena. Rozdíl je, jak rychle se notifikace zobrazí. Někdy se stalo, že se na jednom zařízení zobrazila ihned a na druhém to trvalo přibližně 3 minuty od dostavení se na místo a rozpoznání druhým telefonem. Obecně je tento interval v průměru přibližně od 1 minuty až do 8 min. Takto to probíhá, když uživatel na místo dorazí nějakou dobu (například 5-10 minut) na něm posečká.

Jiná situace nastává, pokud uživatel místem pouze prochází rychlostí průměrné chůze. Zde je důležité, na kolik metrů je nastavený rádius fenců. Pokud je nastavený například na 100 metrů, stalo se mi, že jeden telefon zareagoval správně a včas a druhý nestihl zareagovat vůbec. Pokud je hodnota vyšší, například 400 metrů, nastala situace, kdy telefon, který zareagoval ihned zobrazil notifikaci až příliš brzy, jelikož jsem byl stále od místa daleko. I v tomto případě je ale možné, že horní hranice intervalu, kdy telefon rozpozná může být 10 minut, což má za následek, že se notifikace nedostaví vůbec.

9.2.4 Reakce na fence registrovaných ze souboru

Test prověří, reakci na fence, které jsou opětovně registrovány ze souboru poté, co je proces aplikace ukončen a poté je aplikace opět zapnuta. Takové ukončení je možné pomocí tlačítka "exit"v hlavním menu aplikace, nuceným zastavením aplikace v nastavení aplikace telefonu a také vypnutím telefonu.

Dílčí úkoly zadáme stejně jako v předchozí podkapitole 9.2.3 a následně provedeme ukončení jednou z metod popsaných výše. Poté znovu aplikaci zapneme.

Sluchátka

Po zapnutí aplikace provedeme pouze jediný krok a tím je zapojení drátových sluchátek. Po zapojení se správně zobrazují hudební záznamy v notifikaci. Chování je tedy korektní.

Lokace - nákupní seznam a todo seznam

Takto přečtené a registrované fence ze souboru nefungují správně. Pokud jsou registrovány (stejným způsobem jako když se registrují za běhu) ze souboru a uživatel se nachází mimo lokalitu a následně se do lokality dostane, nic se nestane ani po delší době. Pokud se uživatel v lokalitě již nachází, je přijatý fence rozpoznán a správně srovnán s fencem ze souboru ale stav jeho výsledku je `false` viz 4.5.

Všechny fence jsou registrovány stejným způsobem a u fenců se sluchátky se tento problém neděje, jakmile jsou sluchátka připojeny po spuštění aplikace, správně se přečte soubor a porovná sluchátkový fence a jeho hodnota je `true`. Pokud je stejný fence (ve smyslu stejná lokace) registrovaný za běhu a přístroj se nachází v této lokaci, fence je správně porovnán a jeho hodnota je správně `true`.

9.2.5 Rozpoznání aktivity

V testu je prověřeno, jak dobře jsou rozpoznávány aktivity uživatele, který má telefon u sebe.

1. Kliknutí na tlačítko "activites"
2. Kliknutí na tlačítko "detected activities"
3. Kliknutí na tlačítko "start"
4. Provedení aktivity chůze

5. Sledování obrazovky s výsledky

Úkolem testu je ukázat, jak dobře jsou jednotlivé aktivity detekovány. Po kliknutí na tlačítko "start"bude uživateli zobrazováno, které aktivity jsou rozpoznávány a jaké mají číslo přesnosti rozpoznání. Seznam je aktualizován každé tři sekundy. Test tedy ukazuje, nespolehlivost v rozpoznání aktivit, která je popsána v kapitole 5.4. Bod 4. v testovacím scénař lze zaměnit za libovolnou aktivitu, která dokáže být skrze Awareness API rozpoznáná (viz 4.1) a následně sledovat výsledky.

Podobně lze testovat Awareness API i v druhé referenční aplikaci Vortex (viz 3.2), kde pomocí tlačítka "REFRESH DATA"uživatel získá aktuální informaci o prováděné aktivitě vyhodnocenou pomocí Awareness API.

9.2.6 Uložení a načtení souborů

Jednoduchý test, který ověří uložení zadaných fenců a následně jejich načtení ze souboru a zobrazení v aplikaci.

1. Kliknutí na tlačítko "activites"
2. Kliknutí na tlačítko "shopping"/"to do"
3. Kliknutí na tlačítko "add location"
4. Vyplnění položky "name"a vybrání libovolné lokace, kliknutí na "ADD"
5. Vybrání nově vzniklé lokace ze seznamu
6. Přidání dvou produktů/úkolů
7. Kliknutí na šipku zpět
8. Kliknutí na tlačítko "music"
9. Kliknutí na tlačítko "add release"
10. Vyplnění dialogového okna s názvem alba, jménem interpreta a datem vydání hudebního alba
11. Kliknutí na tlačítko "ADD"
12. Dvakrát kliknutí na tlačítko zpět
13. Kliknutí na tlačítko "exit"

Po otevření aplikace jsou v jednotlivých aktivitách správné záznamy, které byly testem do souboru uloženy. Test tedy proběhl v pořádku.

10 Použitelnost Awareness API

Pravděpodobně nejdůležitějším výstupem této práce je skutečnost, do jaké míry fungují jednotlivé části Awareness API. Této problematice se bude věnovat následující text.

Sluchátka jsou na tom se spolehlivostí pravděpodobně nejlépe. Spolehlivostí je zde myšleno, jestli Awareness API vyhodnocuje správně stav zapojených sluchátek vzhledem ke skutečnému zapojení do telefonu. Pokud byl fence se sluchátky úspěšně registrován - ať už ze souboru nebo za běhu aplikace - byl přijat broadcast a stav fence byl vždy správně vyhodnocen, tedy `true`, pokud byly sluchátka zapojeny (viz 4.5). Zde je jen nutné opět zmínit, že fence spojené se sluchátky fungují pouze na drátová sluchátka, nikoliv na sluchátka s technologií Bluetooth.

Lokace je spojena s rádiusem okolo ní, který ovlivňuje to, jak je poznána. Nesmí být příliš malý a zároveň ani příliš velký. Dle mého testování je rozumná hodnota přibližně 200 metrů. Spolehlivost ale rozhodně není stoprocentní, konkrétnější rozbor je vysvětlen na testovacím scénáři zde 9.2.3.

Nejhorší výsledky jsem zpozoroval během testování aktivit. Předmětem mého testování byly nejčastěji aktivity "WALKING"a "RUNNING", jelikož ty jsem chtěl využít ve své aplikaci. Obě tyto aktivity sdílí stejné problémy a těmi jsou za jak dlouho a zda jsou vůbec rozpoznány. Podrobnější popis těchto problémů je rozebrán v kapitole 5.4. Zde je důležité porovnání s ostatními aplikacemi (viz 3), obě totiž pracují s aktivitami. V aplikaci Conscient je možné zvolit aktivitu, například "Walking"a následně přidružit aplikaci, kterou chceme spustit. Další aplikace nese název Vortex a pomocí tlačítka je možné získat momentální aktivitu. Obě aplikace se potýkají se stejnými problémy jako moje aplikace čili rozpoznání aktivity a doba, kdy je aktivita rozpoznána. Stala se mi i situace s aplikací Conscient, kdy byla rozpoznána aktivita "IN_VEHICLE"během toho, co telefon ležel na místě.

Z těchto poznatků lze říci, že pouze drátová sluchátka jsou stoprocentně spolehlivá, ostatní rozpoznávání mohou občas fungovat správně, jindy s velkou rezervou a někdy naopak vůbec.

11 Závěr

Cílem této práce bylo prozkoumat Google Awareness API a následně vytvořit aplikaci pro OS Android, která API využije a ukáže jeho funkčnost v reálných podmínkách.

Než započal samotný vývoj, byl učiněn teoretický rozbor, kde bylo rozhodnuto, jakým vývojem aplikace projde a pomocí kterého programovacího jazyka bude implementována. Byl proveden průzkum aplikací, které Awareness API využívají a následovalo teoretické prostudování možností samotného Awareness API. Poté byla navržena a následně implementována aplikace, která bude Awareness API využívat a přenese teoretické poznatky do praxe. Byly vytvořeny testovací scénáře, kterými byla ověřena funkčnost aplikace a především funkčnost Awareness API. Testování probíhalo na dvou fyzických zařízení a také na AVD. V závěru byly navrženy možné rozšíření aplikace.

Výsledná aplikace nese název Aware Assistant a je určena pro běžného uživatele, kterému se snaží pomáhat v závislosti na jeho prostředí či vykonávané aktivitě. Druhou skupinou, které tato aplikace může být nápomocná jsou vývojáři, kteří chtějí Awareness API využít ve svých projektech.

Během implementování a testování bylo dosaženo výsledků, které ověřují spolehlivost Awareness API. Bylo zjištěno, že dílčí části fungují různorodě - některé bez problému a jsou spolehlivé, jiné zase naopak s menší spolehlivostí či občas vůbec. Tyto poznatky jsou důležité pro uživatele, ale v první řadě pro vývojáře, kteří pomocí výsledné aplikace vidí reálnou použitelnost Awareness API pro své projekty.

Seznam zkratek

- API - Application programming interface - Programové rozhraní
- ART - Android Runtime - Virtuální stroj pro vytváření běhového prostředí pro programy napsané v Javě
- AVD - Android Virtual Device - Virtuální zařízení s OS Android
- BMI - Body Mass Index - Index tělesné hmotnosti
- GPS - Global positioning system - Globální družicový polohový systém
- IDE - Integrated Development Environment - Software pro pohodlnější psaní a vývoj programů
- JVM - Java Virtual Machine - Virtuální stroj ke spuštění programů a skriptů napsaných v Javě
- OS - Operating system - Operační systém
- RAM - Random Access Memory - Paměť s přímým přístupem
- SDK - Software Development Kit - Sada nástrojů pro vývoj aplikací
- UI - User interface - Uživatelské rozhraní

Literatura

[1] *SnapshotClient* [online]. Android developers, 2020. [cit. 2021/05/15]. Google Awareness API Documentation. Dostupné z: https://developers.google.com/android/reference/com/google/android/gms/awareness/SnapshotClient?hl=es-419.

[2] *Xamarin* [online]. Microsoft, 2020. [cit. 2021/05/28]. Dokumentace pro Xamarin. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/.

[3] *Google Awareness API for Android* [online]. Microsoft, 2016. [cit. 2021/06/20]. Xamarin devblog. Dostupné z: https://devblogs.microsoft.com/xamarin/google-awareness-api-for-android-query-and-react-to-signals/.

[4] *Background Execution Limits* [online]. Android Developers, 2021. [cit. 2021/04/25]. Background Execution Limits. Dostupné z: https://developer.android.com/about/versions/oreo/background.

[5] *Geofencing API* [online]. Android developers, 2021. [cit. 2021/05/15]. Geofencing API. Dostupné z: https://developers.google.com/location-context/geofencing.

[6] *Develop Android apps with Kotlin* [online]. Android developers, 2021. [cit. 2021/05/1]. Kotlin. Dostupné z: https://developer.android.com/kotlin.

[7] *Android's Kotlin-first approach* [online]. Android developers, 2021. [cit. 2021/05/1]. Kotlin. Dostupné z: https://developer.android.com/kotlin.

[8] *Permissions on Android* [online]. Android Developers, 2021. [cit. 2021/04/25]. Permissions Overview. Dostupné z: https://developer.android.com/guide/topics/permissions/overview.

[9] *Activity Fence not working in Awareness API* [online]. Stackoverflow, 2017. [cit. 2021/04/10]. Stackoverflow question 43283337. Dostupné z: https://stackoverflow.com/questions/43283337/activity-fence-not-working-in-awareness-api.

[10] *Android activity recognition with awareness API inconsistent* [online]. Stackoverflow, 2017. [cit. 2021/04/10]. Stackoverflow question 43833052. Dostupné z: https://stackoverflow.com/questions/43833052/android-activity-recognition-with-awareness-api-inconsistent.

[11] *Android SDK Quick Start* [online]. Spotify, 2021. [cit. 2021/05/1]. Spotify for Developers. Dostupné z: `https://developer.spotify.com/documentation/android/quick-start/`.

[12] *TIOBE Index for May 2021* [online]. TIOBE, 2021. [cit. 2021/05/1]. Tiobe Index. Dostupné z: `https://www.tiobe.com/tiobe-index/`.

[13] *Awareness API* [online]. Android developers, 2016. [cit. 2021/04/10]. Google Awareness API Documentation. Dostupné z: `https://developers.google.com/awareness`.

[14] *Create a Fence* [online]. Android developers, 2019. [cit. 2021/04/10]. Google Awareness API Documentation. Dostupné z: `https://developers.google.com/awareness/android-api/fence-create`.

[15] *Class DetectedActivity* [online]. Android developers, 2016. [cit. 2021/04/10]. Google Play Services Documentation. Dostupné z: `https://developers.google.com/android/reference/com/google/android/gms/location/DetectedActivity`.

[16] Herout, P. *Učebnice Jazyka Java*. KOPP, 2008. ISBN 978-80-7232-355-5.

[17] Inc, A. *AccuWeather APIs* [online]. AccuWeater Inc, 2020. [cit. 2021/06/21]. Dostupné z: `https://developer.accuweather.com/`.

[18] Lacko, *Mistrovství - Android*. Computer Press, 2017. ISBN 978-80-251-4875-4.

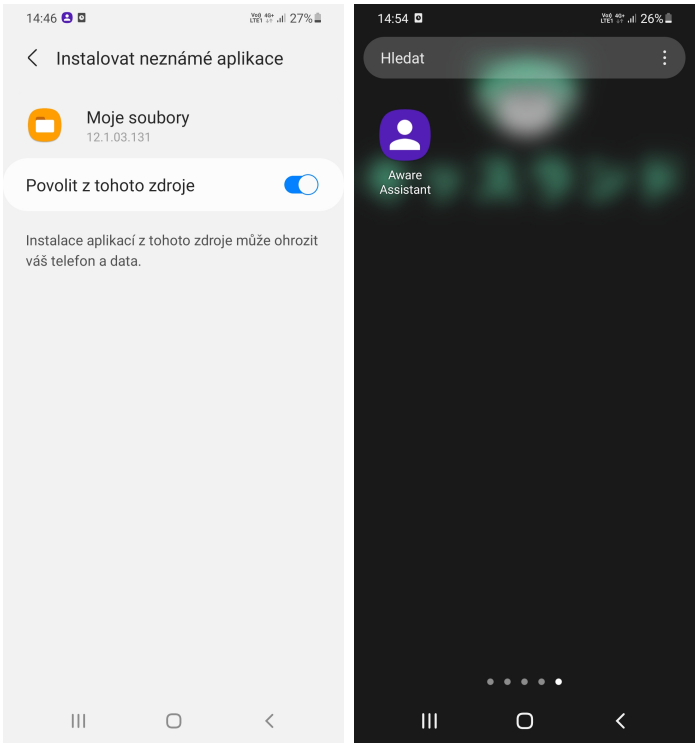
[19] Liu, S. – Striegel, A. *Accurate Extraction of Face-to-Face Proximity Using Smartphones and Bluetooth* [online]. Researchegate, 2011. [cit. 2021/06/20]. Dostupné z: `https://www.researchgate.net/figure/Energy-consumption-of-Bluetooth-WiFi-and-GPS_fig1_252039238`.

A Přílohy

A.1 Instalační příručka

Instalační příručka poskytne potřebné instrukce k nainstalování aplikace do mobilního zařízení. Telefon s minimální verzí Android API 18 připojíme k počítači pomocí USB. Na přiloženém CD se nachází soubor `awareassistant.apk` ve složce `/aplikace`. Tento soubor zkopírujeme do telefonu, například do složky Stažené soubory. Následně otevřeme aplikaci Moje soubory a nalezneme požadovaný soubor `awareassistant.apk`. Naše zařízení musí mít zaškrtnutou možnost instalace neznámé aplikace. Pokud to tak není, budeme na to upozornění a přesměrování do příslušného nastavení, kde možnost povolíme viz A.1a.

Poté následuje průvodce instalací, kde postupujeme dle pokynů. Po úspěšné instalaci se aplikaci objeví v menu, ze kterého ji můžeme spustit viz A.1b.



(a) Udělení povolení (b) Ikona aplikace v menu

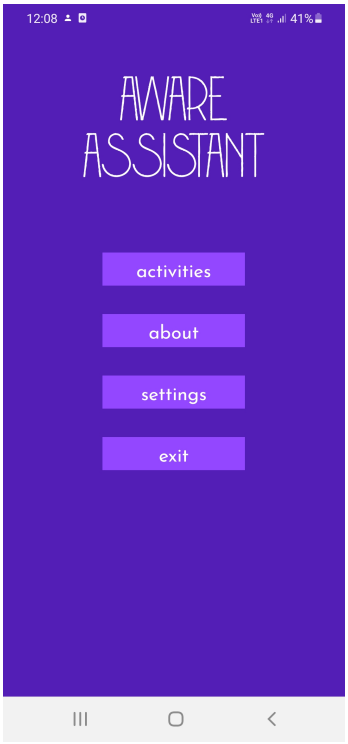
Obrázek A.1: Instalace

A.2 Uživatelská příručka

V následující příloze je uvedena uživatelská příručka, která osvětlí, jak s aplikací zacházet.

A.2.1 Hlavní menu

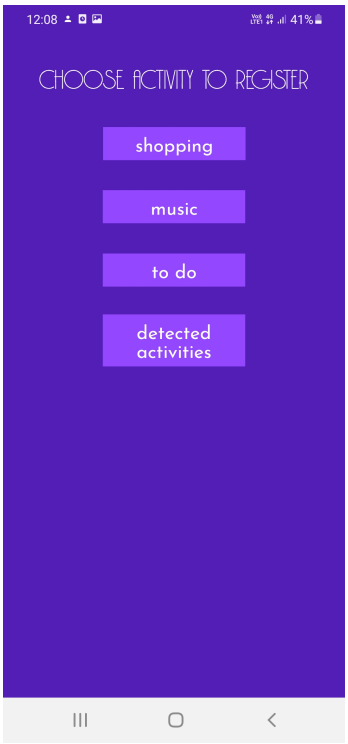
Po otevření aplikace se uživateli jako první objeví hlavní menu aplikace viz obrázek A.2. Je zde několik možností, jak postupovat aplikací dále. Jednotlivé možnosti budou popsány v následujícím textu s výjimkou tlačítka "exit", které pouze ukončuje aplikaci.



Obrázek A.2: Hlavní menu aplikace

A.2.2 Vytvoření a registrace fence

Pokud klikneme v hlavním menu na tlačítko "activites", dostaneme se na obrazovku A.3. Zde se nachází výčet aktivit, které je možné registrovat jako fence. Výjimkou je poslední položka "detected activities", kde žádné aktivity neregistrujeme, nýbrž pouze sledujeme, zda jsou nějaké aktivity detekovány. Budou zde postupně popsány jednotlivé možnosti registrace aktivit.

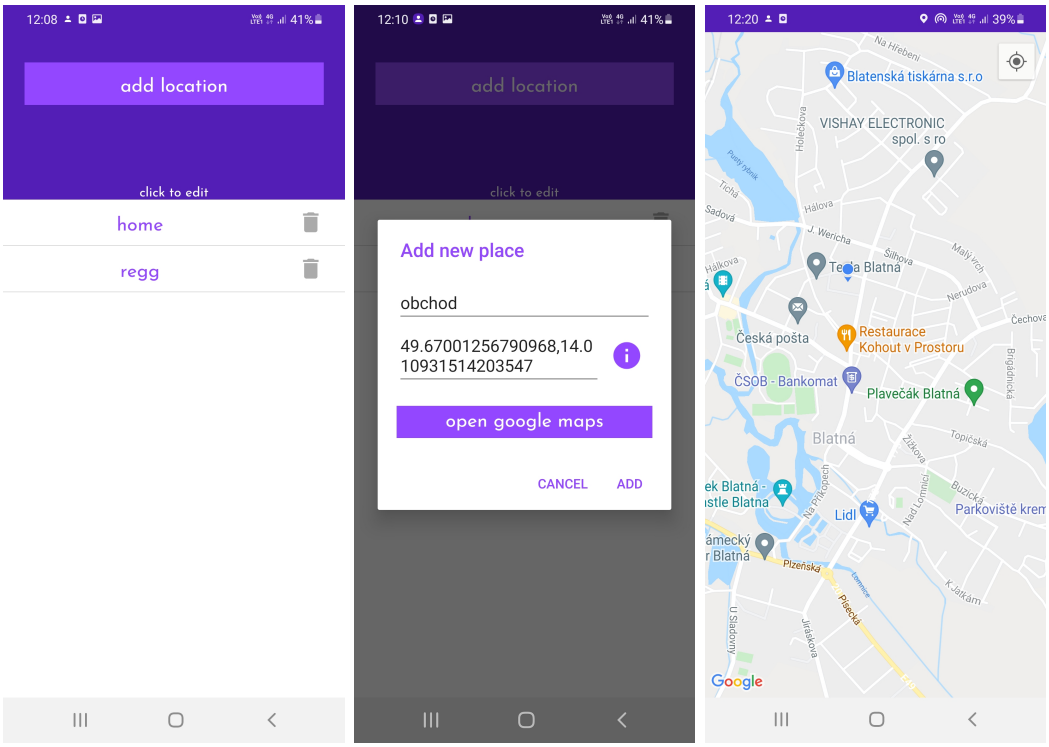


Obrázek A.3: Aktivity k registraci

Shopping

Po kliknutí na tlačítko "shopping"se dostáváme na obrazovku A.4a, kde najdeme tlačítko pro přidání nové lokace a seznam již vytvořených lokací. Aktivitu definuje dvojice lokace a nákupní seznam složený z produktů. Po kliknutí na tlačítko "add location"se zobrazí dialogové okno, kde vyplňujeme příslušné údaje viz A.4b, zde si také můžeme všimnout tlačítka "open google maps", díky kterému můžeme v aplikaci využít Google Maps a kliknutím na místo na mapce vyplnit souřadnice do formuláře viz A.4c.

Po vybrání lokace ze seznamu se dostáváme na obrazovku A.5a, kde můžeme přidávat jednotlivé produkty zadáním jejich názvu do pole "product"a následným stisknutím tlačítka "add product". Můžou být také mazány kliknutím na ikonu s košem. Po kliknutí na tlačítko zpět proběhne registrace a fence nyní může být spuštěn.

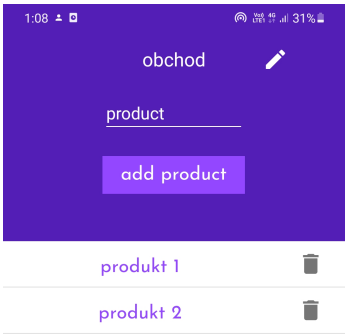


(a) Přehled registrovaných lokací

(b) Vytváření nové lokace

(c) Otevřená mapa s aktuální polohou

Obrázek A.4: Obrazovka shopping

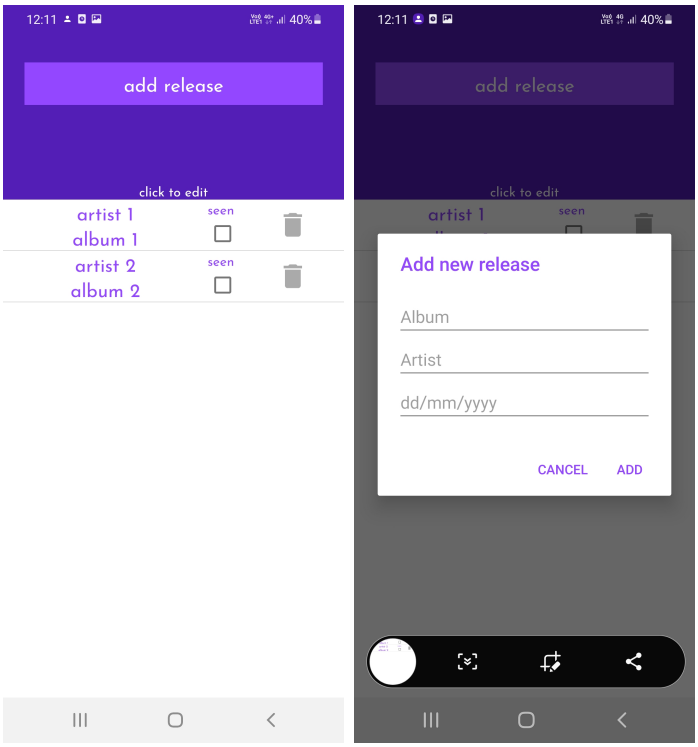


(a) Přehled registrovaných lokací

Obrázek A.5: Obrazovka shopping - detail lokace

Music

Kliknutím na tlačítko "music"se dostáváme na přehled registrovaných hudebních alb, viz A.6a. Checkboxem "seen"je nastaveno, zda uživatel již o vydání alba ví a nechce na něj být nadále upozorňován. Tlačítkem "add release"zobrazíme dialog, ve kterém je možné vyplnit údaje pro přidání nového alba A.6b, kde vyplňujeme název alba, jméno interpreta a datum vydání alba. Poté je hudební fence registrován.



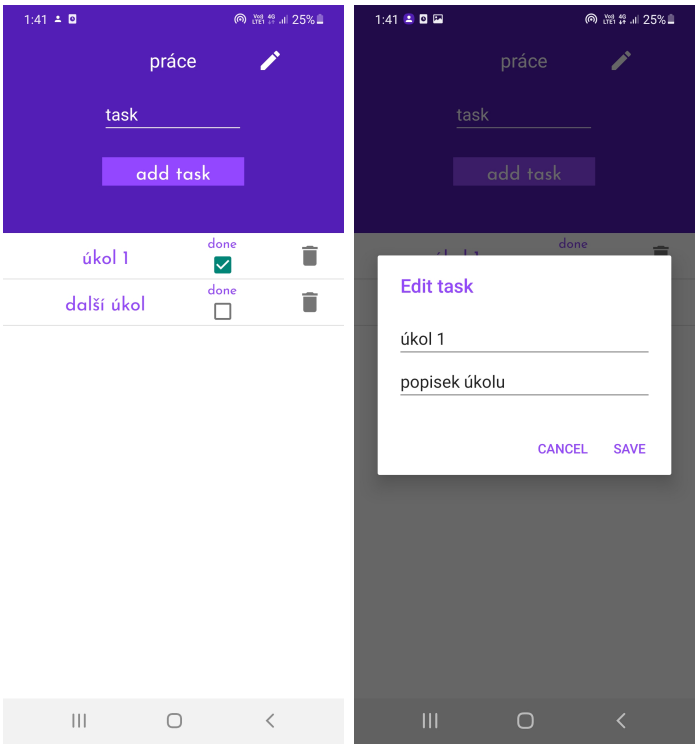
(a) Přehled registrovaných hudebních alb

(b) Vytváření nového alba

Obrázek A.6: Obrazovka music

To do

Kliknutím na tlačítko "to do"se dostaneme na seznam lokací, které jsou spojené s úkoly v to do seznamu. Přehled s lokacemi je totožný s přehledem lokací nákupního seznamu. Rozdíl je pochopitelně v cílovém seznamu s úkoly to do. Detail lokace s názvem "práce"je k nahlédnutí na obrázku A.7a, kde také můžeme vidět tlačítko "add task"pro přidání nového úkol. Checkbox "done"označuje již hotový úkol. Po kliknutí na úkol ze seznamu můžeme měnit jeho název a zároveň měnit popisek k úkolu, viz A.7b.

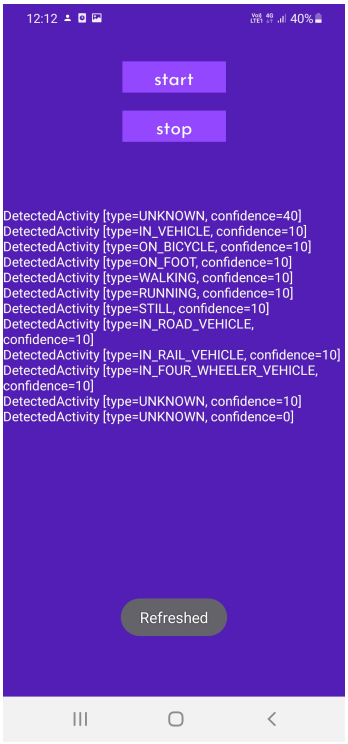


(a) Seznam úkolů (b) Editace úkolu

Obrázek A.7: Obrazovka to do - detail lokace

Detected activities

Jak již bylo zmíněno, zde žádné fence neregistrujeme, obrazovka pouze ukazuje detekované aktivity a jejich číslo přesnosti rozpoznání, viz A.8. Tlačítky "start"a "stop"ovládáme spuštění a zastavení obnovování seznamu, které probíhá každé tři sekundy.



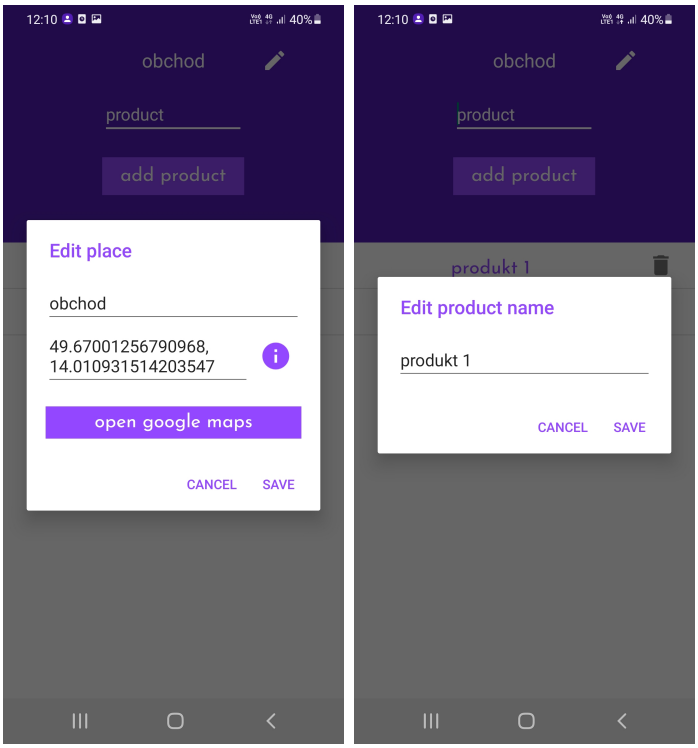
Obrázek A.8: Detekované aktivity

A.2.3 Editace vytvořených fenců

Na obrazovku s přehledem lokací pro nákup, viz A.4a, se dostaneme z hlavního menu kliknutím na položku "activities"a následně na tlačítko "shopping". Pokud klikneme na lokaci ze seznamu, zobrazí se nám její detail se seznamem produktů, viz A.5a. Zde můžeme kliknutím editovat jednotlivé položky nákupního seznamu, viz A.9b. Pokud chceme editovat souřadnice, či název obchodu, klikneme na ikonku tužky, která značí editaci, viz A.9a. Tento princip je aplikován pro všechny fence (shopping, to do, music).

A.2.4 Mazání vytvořených fenců

Tím, že fence smažeme ho zároveň odregistrujeme. K tomu nám slouží ikonka koše, která je vidět například na obrazovce A.4a. Vymazáním jednotlivých



(a) Editace lokace (b) Editace produktu

Obrázek A.9: Editace fenců

produktů či úkolů se registrace fence nemění.

A.2.5 Nastavení

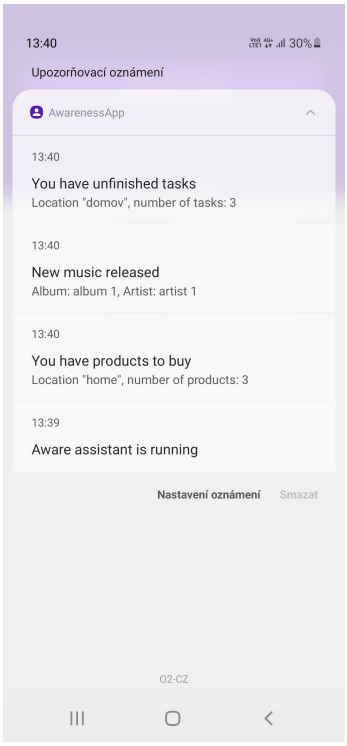
Kliknutím na tlačítko "settings"z hlavního menu se dostaneme na obrazovku A.10. Jedná se o jednoduché nastavení aplikace. Můžeme zde měnit dvě hodnoty. Jednou z nich je zapnutí či vypnutí tzv. "Debug mode", který ukazuje, informace spojené s přijatými fency v `BroadcastReceiver`. Podrobnější vysvětlení najdeme po kliknutí na informační ikonku, která se nachází vedle přepínače. Druhou hodnotou je rádius, který je použit pro fence, které jsou spjaty s lokací. Tento údaj je číselný a udává se v metrech. V nastavení najdeme ještě dvě tlačítka, "save values"a "set default". Tlačítko "save values"uloží stav přepínače a hodnotu vyplněného rádiusu. Druhé tlačítko "set default"nastaví stav přepínače na vypnuto a hodnotu rádiusu na 100 metrů.

Obrázek A.10: Detekované aktivity

A.2.6 Notifikace

Při splnění podmínek některé z našich definovaných fenců se zobrazí notifikace, která nám ukazuje, která fence byla splněna a také dodatečné informace. Jak jednotlivé notifikace vypadají je k nahlédnutí na obrázku A.11.

Po rozkliknutí notifikace se dostáváme na příslušný přehled registrovaných fenců.



Obrázek A.11: Notifikace se splněnými fenci