VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

BEZPEČNOSTNÍ HROZBY INTERNETU
INTERNET SECURITY THREATS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Daniel Bolek**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Václav Zeman, Ph.D.**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Daniel Bolek ***ID:*** 146791
Ročník: 2 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Bezpečnostní hrozby internetu

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte a popište typy útoků využívající prostředí internetu a mající prvky kyberšikany. Z hlediska prevence kyberšikany je důležité vědět, jak se jednotliví uživatelé v prostředí internetu chovají, jaké služby a jak často využívají, jak dbají na ochranu osobních údajů apod. Cílem práce je navrhnout a realizovat systém, který by umožnil sběr a vyhodnocení údajů o chování uživatelů na internetu. Práce je řešena ve spolupráci s pracovníky Policie České republiky.

DOPORUČENÁ LITERATURA:

[1] JAMES, Lance. Phishing bez záhad. 1. vyd. Praha: Grada, 2007, 281 s. ISBN 978-80-247-1766-1.

[2] JIRÁSEK, P., NOVÁK, L., POŽÁR, J. Výkladový slovník kybernetické bezpečnosti: Cyber security glossary. Třetí aktualizované vydání. Praha: Policejní akademie ČR v Praze, 2015, 240 stran. ISBN 978-80-7251-436-6.

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 24.5.2017

Vedoucí práce: doc. Ing. Václav Zeman, Ph.D.
Konzultant: por. Mgr. Zdeňka Procházková

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Diplomová práce se zabývá vývojem systému, který slouží jako podpůrný nástroj pro zkvalitnění prevence kyberšikany. Realizovaný systém poskytuje kontinuální sběr a vyhodnocování informací o tom, kolik času uživatelé tráví v jednotlivých aplikacích určených pro mobilní zařízení s operačním systémem Android. Hlavní části systému jsou uživatelská Android aplikace a vzdálený server s webovou aplikací. Android aplikace zajišťuje průběžné získávání informací o využití zařízení a jejich pravidelné zasílání v týdenních intervalech na vzdálený server. Přístup k historií využívání mobilního zařízení je získán pomocí API `android.app.usage`. Webová aplikace je zodpovědná za příjem dat od jednotlivých mobilních zařízení a za jejich následné zpracování. Výsledky zpracování dat jsou prezentovány skrze uživatelské rozhraní webové aplikace. Pro vybudování webové aplikace je využit PHP framework Laravel.

KLÍČOVÁ SLOVA

Android, `android.app.usage`, Laravel, mobilní aplikace, prevence kyberšikany.

ABSTRACT

The diploma thesis deals with a development of a system, which serves as a supportive tool for improvement of cyberbullying prevention. The system provides continuous collection and evaluation of information about how much time users spend on mobile apps intended for mobile devices running Android operating system. The main parts of the system are the Android application and the remote server with the web application. The Android application provides continuous collection of information about device usage and sends it to the remote server at weekly intervals. Access to mobile device usage history is obtained by the API `android.app.usage`. The web application is responsible for receiving data from individual devices and for subsequent data processing. The results of data processing are presented through the user interface of the web application. The PHP framework Laravel is used to build the web application.

KEYWORDS

Android, `android.app.usage`, Laravel, mobile application, prevention of cyberbullying.

BOLEK, Daniel. *Bezpečnostní hrozby internetu*. Brno, 2017, 108 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Václav Zeman, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Bezpečnostní hrozby internetu“ jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno ....

podpis autora(-ky)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Václavu Zemanovi, Ph.D. za odborné vedení, konzultace a podnětné návrhy k práci. Rád bych také poděkoval paní por. Mgr. Zdeňce Procházkové za odborné konzultace a možnost účastnit se školení v oblasti kyberšikany a nebezpečných počítačových jevů.

Brno ....

podpis autora(-ky)

OBSAH

Úvod **14**

1 Nebezpečné jevy internetu **15**
1.1 Kyberšikana . . . 15
1.2 Kybergrooming . . . 17
1.3 Kyberstalking a stalking . . . 17

2 Požadavky na navrhovaný systém **19**

3 Operační systém Android **20**
3.1 Architektura systému . . . 21
3.2 Základní komponenty aplikace . . . 22
3.2.1 Aktivita . . . 23
3.2.2 Služba . . . 23
3.2.3 Příjemce vysílání . . . 23
3.2.4 Poskytovatel obsahu . . . 24
3.3 Životní cyklus aktivity . . . 24

4 Analýza problematiky **26**
4.1 Ověření řešitelnosti zadání . . . 26
4.1.1 Diskuze vzniklých úskalí . . . 27
4.2 Nalezení optimální varianty . . . 28
4.2.1 Popis nabízených vlastností . . . 28
4.2.2 Ověření funkčnosti . . . 30

5 Realizace systému **32**
5.1 Android aplikace . . . 32
5.1.1 Volba klíčových komponent . . . 33
5.1.2 Struktura aplikace . . . 36
5.1.3 Vnitřní logika . . . 40
5.1.4 Uživatelské rozhraní . . . 61
5.2 Vzdálený server . . . 69
5.2.1 Použité technologie . . . 69
5.2.2 Základní konfigurace . . . 70
5.2.3 Struktura webové aplikace . . . 72
5.2.4 Vytvořené panely . . . 74
5.3 Komunikace aplikace-server . . . 80
5.3.1 Registrace aplikace . . . 80

5.3.2 Zasílání týdenních statistik 82

6 Testování systému a závěrečná doporučení 83
6.1 Testování systému 83
6.1.1 Testování Android aplikace 83
6.1.2 Testování webové aplikace 85
6.2 Závěrečná doporučení 86

7 Závěr 88

Literatura 89

Seznam symbolů, veličin a zkratek 95

Seznam příloh 96

A Testování API `android.app.usage` 97

B Manifest Android aplikace 98

C Diagramy tříd Android aplikace 100

D Konfigurační soubor .env 103

E Uživatelské rozhraní web. aplikace 104
E.1 Administrativní panel 104
E.2 Panel vývojových trendů 105
E.3 Panely top aplikací 106

F Obsah přiloženého CD 108

SEZNAM OBRÁZKŮ

3.1 Zastoupení jednotlivých verzí OS Android [19] 20
3.2 Architektura OS Android [21] 21
3.3 Životní cyklus aktivity [24] 25
4.1 Ukázka získání informací ze třídy UsageStatsManager [31] 29
5.1 Schéma konceptu realizovaného systému 32
5.2 Třídy reprezentující vnitřní mechanismus aplikace 37
5.3 Třídy `UsageStatistic` a `UsageItem` 39
5.4 Třída `NetworkHelper` 39
5.5 Třídy reprezentující uživatelské rozhraní aplikace 40
5.6 Zjednodušený sekvenční diagram aktivace úlohy na registraci aplikace na vzdáleném serveru 43
5.7 Zjednodušený sekvenční diagram aktivace služby `DailyStatsSaver` za pomocí „půlnočního“ alarmu 49
5.8 Zjednodušený sekvenční diagram aktivace služby `DailyStatsSaver` za pomocí „denního“ alarmu 50
5.9 Zjednodušený sekvenční diagram reprezentující koordinační operace spojené s příjemcem vysílání `MiddayAlarmReceiver` 55
5.10 Vývojový diagram algoritmu vlákna `WeeklySyncRoutineThread` . . . 58
5.11 Fragmenty v rámci úvodní aktivity `IntroActivity` 63
5.12 Vybrané varovné zprávy úvodních fragmentů 64
5.13 Základní rozložení hlavní aktivity 65
5.14 Základní grafická struktura hlavní aktivity 66
5.15 Výsledný grafický vzhled fragmentu `FragmentForUsageStatistics` . 68
5.16 Laravel MVC architektura [47] 69
5.17 Rozbor zastoupení uživatelů dle jednotlivých věkových kategorií . . . 75
5.18 Zastoupení pohlaví v definovaných věkových kategoriích 76
5.19 Počet registrovaných a aktivních uživatelů v průběhu týdnů 76
5.20 Nejpoužívanější aplikace dle celkového času v průběhu týdnů 77
5.21 Nejpoužívanější aplikace dle celkového času 78
5.22 Panel „Top aplikace“ pro definované věkové kategorie 79
5.23 Ukázka procesu registrace aplikace na vzdáleném serveru 81
6.1 Ukázka testovacích záznamů v tabulce `DailyUsageStats` 84
6.2 Ukázka zachycené komunikace pomocí nástroje Wireshark 85
C.1 Diagram dědičnosti tříd příjemců vysílání – balíček `alarm_receivers` 100
C.2 Diagram dědičnosti tříd služeb – balíček `background_services` . . . 101
C.3 Diagram tříd – balíček `database` 102
E.1 Administrativní panel – pohled č. 1 104

E.2 Administrativní panel – pohled č. 2 104
E.3 Vývojové trendy – pohled č. 1 105
E.4 Vývojové trendy – pohled č. 2 105
E.5 Top aplikace – celkový souhrn 106
E.6 Top aplikace – dle pohlaví 106
E.7 Top aplikace – dle věkové kategorie 107

SEZNAM TABULEK

5.1 Tabulka pro denní statistiky 38
5.2 Tabulka pro týdenní souhrny 38

SEZNAM VÝPISŮ

5.1 Naplánování úlohy na registraci aplikace na vzdáleném serveru 42
5.2 Registrace alarmu pro `MidnightAlarmReceiver` 46
5.3 Registrace alarmu pro `DaytimeAlarmReceiver` 47
5.4 Registrace příjemce vysílání `DeviceBootreceiver` v manifestu aplikace 51
5.5 Registrace alarmu pro `MiddayAlarmReceiver` 53
5.6 Registrace aktivity `SplashActivity` v manifestu aplikace 62
5.7 Routy webové aplikace – `web.php` 73
5.8 Zkrácený výpis migrace `CreateInputTable` 74
5.9 Zkrácený výpis migrace `CreateUserTable` 74
5.10 Ukázka validace příchozích dat v rámci metody `saveData()` 82
A.1 Zjednodušená ukázka kódu stěžejní části testovací aplikace pro ověření funkčnosti balíčku `android.app.usage` 97
B.1 Soubor `AndroidManifest.xml` projektu Android aplikace 98
D.1 Ukázka nastavení konfiguračního souboru `.env` projektu webové aplikace 103

ÚVOD

S příchodem nových technologií přišla také celá řada nových hrozeb a nebezpečných jevů. Jedním z nich je rozšíření tradiční školní šikany do prostředí internetu a komunikačních technologií. Začíná se hovořit o problému kyberšikany, který dnes postihuje velké množství dětí základních a středních škol. Základním předpokladem boje s kyberšikanou a s ní souvisejícími jevy je důsledná prevence. Efektivní prevence zase vychází ze znalosti chování uživatelů ve virtuálním prostředí, tj. jakým způsobem spolu komunikují, jak často, jaké služby využívají apod. Hlavní cíl diplomové práce spočívá v navržení a realizaci systému, který by usnadnil získávání statistických dat pro účely prevence. Dílčím cílem je popsat typy útoků týkající se oblasti kyberšikany.

Problematika nebezpečných jevů na internetu spočívá mimo jiné v tom, že se jedná o velmi dynamické a proměnlivé prostředí, ve kterém se v průběhu času střídají různé trendy či fenomény a s nimi spojena bezpečnostní rizika. Tyto trendy je potřeba z hlediska prevence sledovat a průběžně na ně reagovat.

Jako řešení se nabízí návrh aplikace pro operační systém Android, jež bude kontinuálně zaznamenávat, které mobilní aplikace jsou nejčastěji využívány a kolik času v nich uživatelé tráví. Tyto informace budou propojeny s věkem a pohlavím uživatele a jako anonymní data budou zasílána na vzdálený server, jenž zajistí jejich zpracování a poskytne další potřebná vyhodnocení. Na základě zpracovaných výsledků bude Policie České republiky schopná sledovat, jaké jsou nejčastěji využívané mobilní aplikace dle pohlaví a věku uživatelů, a v rámci přípravy preventivních programů se tak efektivně zaměřit na rizikové oblasti.

Samotná práce je rozdělená na šest hlavních kapitol. První kapitola je věnována problematice kyberšikany a s ní souvisejícími nebezpečnými jevy jako je kybergrooming a kyberstalking. Druhá kapitola uvádí požadavky Policie ČR na realizovaný systém. Ve třetí kapitole je představen operační systém Android, jeho architektura, základní komponenty a princip fungování životního cyklu aktivity. Následující kapitola se zabývá analýzou problematiky měření aktivní doby běhu aplikací v operačním systému Android, uvádí jednotlivá úskalí a představuje volbu vhodného řešení. Pátá kapitola je věnována popisu realizovaného systému. Jsou zde představeny jeho dvě základní části, tj. Android aplikace a vzdálený server, resp. webová aplikace. U vyvíjené Android aplikace je pozornost zaměřena především na její vnitřní mechanismus, který je stěžejní části celého systému. Poslední kapitola popisuje princip testování realizovaného systému a uvádí další plánované kroky ve vývoji formou závěrečných doporučení.

1 NEBEZPEČNÉ JEVY INTERNETU

O kladných aspektech internetu není potřeba dlouze psát. Usnadňuje lidem práci a život v mnoha rovinách. Obdobných způsobem bohužel usnadňuje konání trestné činnosti, útočníkům poskytuje širší prostor pro jejich „tvůrčí“ práci a stává se moderní platformou pro šíření agresivity a násilí. Cílem této kapitoly je seznámit čtenáře s nebezpečnými jevy internetu, přičemž text bude omezen na problematiku kyberšikany a jevů s ní bezprostředně souvisejících jako je kybergroomig a kyberstalking. Vzhledem k technické povaze této práce je následující rozbor pouze stručným úvodem do dané problematiky.

1.1 Kyberšikana

Kyberšikana (angl. cyberbullying), nebo-li kybernetická šikana je označována za formu psychické šikany, při které jsou užívány prostředky elektronické komunikace (např. internet, mobilní telefony apod.) [1], [2]. Mezi nejčastější projevy kyberšikany patří zejména [3], [4]:

- verbální útoky, zesměšňování a ponižování (prostřednictvím sociálních sítí, SMS zpráv, e-mailů, apod.),
- zastrašování, vyhrožování a vydírání (pomocí internetu či mobilního telefonu obecně),
- tvorba webových stránek či blogů s cílem poškodit konkrétní osobu,
- zveřejnění fotografie, videa či zvukového záznamu oběti s ponižujícím podtextem,
- obtěžování a pronásledování (formou SMS zpráv, frekvetovaného volání či prozvánění),
- odkrývání cizích tajemství, odcizení elektronické identity a zneužití cizího účtu ke kyberšikaně.

Kyberšikana se od klasické šikany neliší pouze v samotném použití informačních a komunikačních technologií, ale především v důsledku použití těchto technologií. Uskalí spočívá v tom, že se děti přesouvají z klasického, resp. fyzického prostředí do prostředí virtuálního, ve kterém se objevují zcela nová nebezpečí a nástrahy. Tento fakt činí z kyberšikany mnohem závažnější problém než je šikana klasická. Definovat lze tyto nejzásadnější rozdíly [2], [5], [6], [7]:

A **Anonymita**

Ta v prvé řadě vytváří v agresorovi pocit beztrestnosti, čímž mu poskytuje odvahu jednat způsobem, kterým by se v běžném prostředí jednat neodvážil. V druhé řadě v něm umocňuje pocit moci nad obětí, která mnohdy neví,

kdo na ní útočí. Z praktického hlediska se jedná o anonymitu pouze zdánlivou, jelikož nalezení útočníka např. na základě použité IP adresy je běžnou praxí policie. Na druhou stranu, jedná-li se o technicky zdatného útočníka, pak např. zneužitím anonymizačních služeb (open proxy servery, síť TOR apod.) se útočník pro policii stává prakticky téměř nevystopovatelný.

B **Profil útočníka**

Útočníkem může být i fyzický slabý jedinec s nízkým postavením v sociální skupině, jenž by k šikanování v běžném prostředí neměl dostatečné prostředky. Rozdíly v pohlaví, věku, postavení apod. ve virtuálním prostoru mizí. Klíčem k úspěšnému provedení útoku ve virtuálním prostředí je znalost informačních a komunikačních technologií.

C **Profil oběti**

Ve virtuálním prostředí se oběti kyberšikany může snadno stát i člověk, který je v běžné prostředí velmi oblíbený. Největší riziko hrozí osobám, jež jsou na mobilních telefonech a internetu závislé.

D **Neomezenost útoku**

Skrze elektronické prostředky může být oběť zasažena kdekoliv a kdykoliv. Z hlediska prostoru není k útoku zapotřebí konkrétní místo či osobní kontakt. Z pohledu času se tak může dít 24 hodin denně, 365 dní v roce.

E **Délka trvání a opakovatelnost**

Zatímco klasická forma šikany trvá pouze omezenou dobu a pro její větší účinnost je potřeba jí opakovat, u kyberšikany se jediný útok může automaticky stát opakovatelným a z hlediska trvání neomezeným (např. vystavení ponižující fotografie oběti na sociálních sítích). Zvyšuje se tak dlouhodobé trauma oběti.

F **Nesnadná detekovatelnost**

Vzhledem k psychické povaze útoků zde chybí typické znaky šikany jako jsou modřiny, poškozené věci apod. Ani samotná poškozená osoba mnohdy nerozpozná, že se stala obětí kyberšikany a s problémem se tak nedokáže vypořádat.

G **Neomezené publikum**

Internet dává prostor ke značnému nárůstu pozorovatelů či účastníků kyberšikany, což vede k větší intenzitě útoků a následného dopadu na oběť. Dalším problémem je, že např. změna bydliště a kolektivu nemění virtuální publikum. Nový kolektiv se navíc může velmi snadno dostat k informacím o minulosti oběti a na předchozí útoky tak jednoduše navázat.

H **Neúmyslnost**

Virtuální komunikace eliminuje některé důležité prvky osobní komunikace, čímž dochází k riziku nezáměrného ublížení např. nevhodným žertem.

1.2 Kybergrooming

Pojmem kybergooming je nazváno jednání osoby, resp. uživatele internetu, jenž má za cíl vzbudit u oběti klamnou důvěru a dosáhnout tak osobní schůzky. Děje se tak za pomocí sociálních sítí, internetových seznamek, e-mailu či jiných online komunikačních kanálů. Důsledky této schůzky pak mohou být pro oběť fatální. Nejčastěji se hovoří o sexuálním zneužití, fyzickém násilí a v krajních případech o usmrcení oběti [1], [8]. Studie ukazují, že oběti nejčastěji spadají do skupiny nezletilých ve věku od 10 do 17 let, přičemž se velmi často jedná o dívky [9]. Profil útočníka je u kybergroomingu poměrně nesourodý. Může se jednat o osoby s různým postavením ve společenské skupině. Společným jevem mnohdy bývá diagnostika chorobného zájmu o děti. Společně se skutečností, že útoky jsou velmi promýšlené a sofistikované (útočníkem je v drtivé většině dospělá osoba), se jedná o nejvíce nebezpečnou formu útoku z oblasti problematiky nebezpečných komunikačních jevů virtuálního prostředí [3], [8].

Samotný útok probíhá v několika na sebe navazujících etapách, přičemž se jedná o poměrně dlouhodobou záležitost (od 3 měsíců až po řadu let). První etapa je charakteristická samotnou přípravou útočníka, ve které si zajišťuje falešnou identitu či identity, shromažďuje volně dostupné informace a vybírá si potencionální oběť. Ve druhé fázi s obětí navazuje virtuální kontakt. Pomocí různých manipulačních technik, jako je např. efekt zrcadlení[1], s ní prohlubuje svůj vztah a buduje emoční závislost, získává další citlivé údaje a snaží se oběť izolovat formou zastrašování či citového vydírání. Třetí etapou je plánovaní osobní schůzky a její samotná realizace. Není-li zmanipulované dítě ke schůzce svolné, děje se tak s použitím nátlaku. Náplní osobní schůzky je buďto opakování druhé etapy útoků (tentokrát již v běžném prostředí) anebo útočník přejde do poslední fáze útoku, jimž je sexuální obtěžování, zneužití dítěte nebo fyzické či psychické napadení [8], [10].

1.3 Kyberstalking a stalking

Jedná se o záměrné obtěžování a pronásledování určité cizí osoby s cílem narušit její bezpečnost, vyvolat u ní pocit strachu a omezit její kvalitu života. Stalking, resp. kyberstalking, se vyznačuje svou opakovatelností, trvalostí a nepředvídatelnosti. O termínu kyberstalking se hovoří v případě užívání prostředků elektronické komunikace útočníkem. Tyto prostředky realizaci útoků výrazně zjednodušují a zefektivňují [1], [8], [11]. Kyberstalking a stalking se opět označuje za velmi nebezpečný jev, jelikož téměř třetina všech případů končí násilným jednáním nebo usmrcením

[1] útočník napodobuje chování oběti, předstírá společné záliby apod.

oběti. Typologie oběti se zde často kategorizuje na základě jejího vztahu k útočníkovi. Obdobně je určována typologie útočníka, kdy se kromě vztahu k oběti (bývalý partner, obdivovatel apod.) posuzuje motivace jeho jednání [8], [12].

Existuje celá řada klasifikací projevu stalkingu. Základní varianta se rozděluje na tři fáze [11], jež mohou na sebe navazovat či se vzájemně prolínat. První fáze se označuje za nejmírnější verzi stalkingu, kdy se jedná o pronásledování na dálku neboli obtěžování. Spadá zde jednání jako je soustavné zasílání nevyžádaných vzkazů, fotografií apod., dále pokusy o vzdálený kontakt s obětí či poškozování její reputace. Již na této úrovni stalkingu mohou být následky obtěžování velmi vážné (psychické potíže, ztráta partnera či zaměstnání atd.). Druhá fáze se označuje jako pronásledování na blízko. Zde jsou nově patrné známky přítomnosti útočníka. Jsou jimi vzkazy u vozidla, domu či pracoviště. Objevují se stopy škod u věcí vlastněných obětí. Třetí fázi je tzv. nebezpečné pronásledování, kde se již objevuje přímé vyhrožování (újmou na zdraví a životě, únosem apod.) či fyzické násilí. Často se jedná o cílené poškozování majetku, zabití domácího zvířete, fyzické pronásledování oběti a v nejhorších případech následuje usmrcení oběti či jejich blízkých.

Závěrem je nutné poznamenat, že posuzování stalkingu z hlediska trestního práva je velmi komplexní a nelehká záležitost. Obecně se za nebezpečné pronásledování považuje jednání, jehož minimální trvání přesahuje období 4 týdnu, přičemž proběhlo alespoň 8-10 pokusů o nevyžádaný kontakt. Hlavním úskalím je rozeznání či stanovení hranice, kdy je již možné útočníka za tento přečin trestně stíhat [8], [13], [14].

2 POŽADAVKY NA NAVRHOVANÝ SYSTÉM

Jednotlivé požadavky na navrhovaný systém byly průběžně tvořeny a postupně rozšiřovány na základě konzultací s pracovníky Policie ČR. Stěžejní části systému byly v průběhu vývoje diskutovány s odborníky z komerční a akademické sféry.

Hlavním cílem navrhovaného systému je umožnit kontinuální sběr a vyhodnocování informací o využívání aplikací určených pro mobilní zařízení s operačním systémem Android. Důležitým aspektem systému je schopnost informace analyzovat na základě věku a pohlaví uživatele. Systém jako celek se skládá z uživatelské aplikace pro operační systém Android a vzdáleného serveru. Konkrétní požadavky na uživatelskou aplikaci jsou následující:

- Získání informací ohledně roku narození a pohlaví uživatele skrze uživatelské rozhraní vyvíjené aplikace.
- Schopnost získat informace o celkové aktivní době využívání jednotlivých aplikací, jež se nacházejí v mobilním zařízení uživatele, tzn. navrhnout a zrealizovat mechanismus zajišťující tuto funkcionalitu.
- Schopnost automatického a pravidelného zasílání získaných dat na vzdálený server za účelem dalšího zpracování (spolu s informacemi o uživateli).
- Navrhnout způsob komunikace se vzdáleným serverem s ohledem na spotřebu baterie zařízení a optimální využití síťových prostředků.

Požadavky na vzdálený server jsou tyto:

- Zajistit příjem dat od jednotlivých mobilních zařízení a jejich uložení do SQL databáze.
- Vytvoření webového rozhraní za účelem vizualizace výsledků zpracování dat dle následných požadavků:
  - Schopnost zobrazit celkový počet uživatelů, zastoupení mužů a žen, zastoupení uživatelů dle věkových kategorií a zastoupení pohlaví v jednotlivých věkových kategorií.
  - Vyhodnotit nejpoužívanější aplikace z hlediska celkového času stráveného v příslušné aplikaci a také z hlediska zastoupení dané aplikace mezi uživateli (četnost jejího výskytu v týdenních souhrnech).
  - Stejné informace vyhodnotit pro jednotlivé věkové kategorie a také pro jednotlivá pohlaví.
  - Možnost zobrazení nejpoužívanějších aplikací dle celkového času v průběhu jednotlivých týdnů.

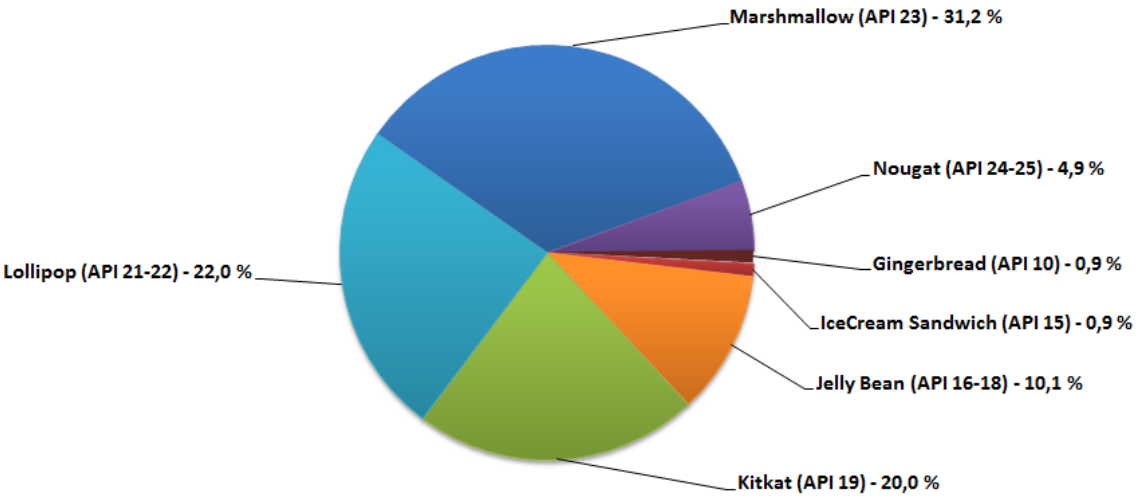
Případná upřesnění uvedených požadavků jsou zmíněna v kontextu kapitol či sekcí věnovaných příslušným částem daného systému.

3 OPERAČNÍ SYSTÉM ANDROID

Jedná se o operační systém (dále jen OS) primárně určený pro mobilní zařízení, jako jsou chytré telefony či tablety. Jeho použití se ovšem rozšiřuje také na nositelnou elektroniku (např. chytré hodinky), automobily, televize, fotoaparáty apod. OS Android je vybudován na jádře Linuxu a od roku 2007 je k dispozici jako tzv. svobodný software (angl. open-source software). V současnosti je vyvíjen firmou Google, resp. v širším měřítku spadá pod vývoj Open Handset Alliance [15].

Hlavním důvodem výběru OS Android je jeho velká rozšířenost u mobilních zařízení a tabletů. Jsou to právě tato zařízení, která dnes děti používají pro komunikaci na sociálních sítích či ve virtuálním prostoru obecně nejvíce. Celosvětově má OS Android 86,8% podíl[1] na trhu s chytrými telefony [16]. V České republice, ke konci roku 2016, využívalo OS Android přibližně 74 % všech uživatelů [17], [18]. Díky této skutečnosti je zde potenciál cílit na velký počet uživatelů a získat tak rozsáhlý a poměrně přesný vzorek informací o chování (nejen) dětí ve virtuálním prostředí.

Z hlediska vývoje aplikace je důležité také znát, které verze OS Android jsou mezi uživateli nejvíce rozšířené. Cílem je efektivní využití nově dostupných tříd a jejich metod u nových verzí OS Android a současně zajištění kompatibility pro starší, avšak doposud hojně využívané verze systému. Na obrázku 3.1 je graficky znázorněno současné zastoupení[2] jednotlivých verzí OS Android[3].



Obr. 3.1: Zastoupení jednotlivých verzí OS Android [19]

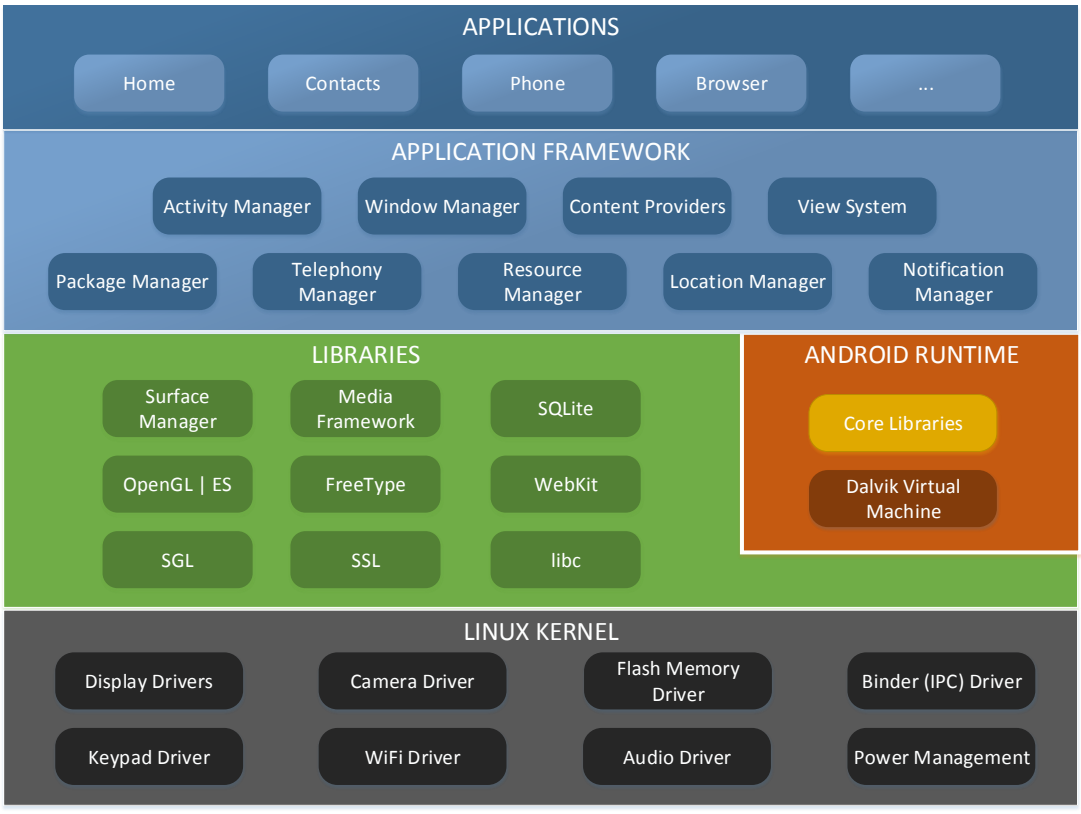
[1] Údaj je platný pro třetí kvartál roku 2016.
[2] Platné ke dni 24.4.2017.
[3] Verze systému s nižším zastoupením jak 0,1 % nejsou v grafu uvedeny.

Jednotlivé verze systému se odlišují nejen svým názvem, ale také úrovní API (Application Programming Interface), která je pro danou verzi jednoznačným identifikátorem. Úroveň API je významná především z hlediska funkcionality, kompatibility, bezpečnosti systému, apod. Současnou nejnovější verzi[4] je Android 7.1 s úrovní API 25.

3.1 Architektura systému

OS Android je rozdělen přibližně na 5 sekcí, resp. do 4 hlavních vrstev. Jejich uspořádání je přehledně zobrazeno na obrázku 3.2. Informace uvedené v této části kapitoly vycházejí ze zdrojů [20] a [21].

Nejnižší vrstvou architektury je *Linux Kernel*, neboli jádro Linuxu. Android jej využívá k řízení procesů a paměti, ke správě ovladačů hardwaru zařízení a integrovaných periferií, a také k řízení síťových a dalších potřebných služeb systému. Díky vybudování Androidu na jádře Linuxu je tento operační systém snadno distribuovatelný na různé platformy. Přímý přístup k této vrstvě není koncovému uživateli,



Obr. 3.2: Architektura OS Android [21]

[4]Údaj je platný ke dni 28.4.2017.

resp. instalované aplikaci umožněn. Přístup je zprostředkován skrze vyšší vrstvu architektury.

Nad linuxovým jádrem se nacházejí C/C++ knihovny související s mediálními kodeky (MPEG4, H.264, MP3 atd.), databázemi (SQLite), zabezpečením (OpenSSL), vykreslováním grafiky apod. Nachází se zde také tzv. *Android Runtime*, jenž obsahuje vlastní aplikační virtuální stroj Dalvik a specifické knihovny programovacího jazyka Java určené pro vývoj Androidu. Ve výchozím stavu běží každá aplikace ve svém vlastním linuxovém procesu a každý proces má přidělenou svou vlastní instanci virtuálního stroje Dalvik. Díky tomuto přístupu jsou jednotlivé aplikace od sebe navzájem bezpečně izolované a mohou v systému provádět jen operace, ke kterým mají přidělená práva.

Následnou vrstvou architektury je *Application Framework*. Z hlediska vývoje aplikací je tato část nejvýznamnější. Formou tříd poskytuje vývojářům velkou řadu služeb, jež mohou do své aplikace implementovat. Mezi nejpoužívanější třídy patří např.:

- `Activity Manager` – zodpovědná za správu životního cyklu aplikace,
- `Contents Provider` – zodpovědná za manipulaci s daty a jejich zprostředkování pro ostatní aplikace,
- `View System` – poskytuje prostředky pro tvorbu uživatelského rozhraní,
- `Resource Manager` – zajišťuje přístup ke zdrojům jako jsou textové řetězce, ikony, barvy apod.,
- `Notification Manager` – umožňuje zobrazení notifikací či upozornění aplikace,
- `Package Manager` – poskytuje různé druhy informací o instalovaných aplikacích.

Samotné aplikace jsou součástí poslední, nejvyšší vrstvy. Zde se odehrává interakce mezi zařízením a uživatelem. Spadají zde předinstalované aplikace výrobce zařízení či aplikace dodatečně instalované (získané např. z obchodu *Google Play*).

3.2 Základní komponenty aplikace

V Android aplikaci je možné se setkat s celkem čtyřmi základními komponentami. Jsou to aktivity, služby, příjemci vysílání a poskytovatelé obsahu. Jsou to unikátní stavební bloky, kdy každý blok slouží k odlišnému účelu a má odlišný životní cyklus, který určuje jakým způsobem je daná komponenta v rámci aplikace vytvořena nebo naopak zrušena. Na základě toho, jaké bloky použijeme a jak budou na sobě záviset, definujeme chování dané aplikace. Aby bylo možné tyto komponenty v aplikaci využívat, je zapotřebí je definovat v souboru `AndroidManifest.xml`, který je uložen v kořenovém adresáři projektu [22].

Důležité je poznamenat, že jednotlivé komponenty (kromě poskytovatele obsahu) jsou aktivovány prostřednictvím asynchronní zprávy označované jako tzv. záměr. Záměr je vytvořen jako instance třídy `Intent`. Zjednodušeně řečeno se jedná o žádost jedné komponenty o spuštění jiné, čímž dochází k jejich vzájemné kooperaci. Následný popis jednotlivých komponent vychází ze zdrojů [22] a [23].

3.2.1 Aktivita

Aktivita představuje jednu obrazovku, která zprostředkovává uživatelské rozhraní aplikace. Samotná aplikace se tedy může skládat z několika na sobě nezávislých aktivit (jedna, jež slouží jako úvodní obrazovka, druhá sloužící pro čtení dat, další určená ke změně nastavení apod.) a tvořit tak jednotné uživatelské prostředí. Ačkoliv jsou jednotlivé aktvity na sobě nezávislé, mohou si navzájem předávat informace.

V souvislosti s aktivitou se často hovoří také o tzv. fragmentech. Fragment představuje jakousi modulární sekci aktivity, jež má svůj vlastní životní cyklus v rámci životního cyklu aktivity. Jedna aktivita může obsahovat jeden a více fragmentů, které lze do aktivity dynamicky přidávat či odebírat. Výhodou je, že jednotlivé fragmenty lze opakovaně využít v rámci různých aktivit a lze díky ním vytvářet dynamická a flexibilní uživatelská rozhraní, jež podporují různé velikosti obrazovek zařízení s OS Android.

Ve zdrojovém kódu je aktivita implementována jako podtřída třídy `Activity` a fragment je implementován jako podtřída třídy `Fragment`.

3.2.2 Služba

Jedná se o komponentu, jež nezprostředkovává uživatelské rozhraní, ale běží na pozadí za účelem provádění dlouhodobých operací či obsluhy vzdálených procesů (např. aktualizace databáze aplikace z příslušného serveru). Služba může být spuštěna jinou komponentou aplikace (např. aktivitou) a běžet na pozadí i v případě, že uživatel již aplikaci nepoužívá či začne používat aplikaci odlišnou (např. přehrávání hudby, zatímco si uživatel prohlíží webovou galerii). Službu lze také svázat s danou komponentou a zajistit tak vzájemnou interakci a předávání zpracovaných informací.

Služba je ve zdrojovém kódu implementována jako podtřída třídy `Service`.

3.2.3 Příjemce vysílání

Příjemce vysílání je komponenta zodpovědná za naslouchání oznámením a hlášením systému či jiné aplikace. Typickým systémovým hlášením je např. oznámení o nízkém stavu baterie zařízení či vyčerpání volné kapacity datového úložiště. Příkladem oznámení pocházejícího z aplikace je hlášení o provedení aktualizace databáze či

stažení požadovaných dat. Na základě oznámení pak příjemce vysílání reaguje určitou akcí. Může jít o spuštění jiné komponenty nebo se velmi často jedná o zobrazení stavového řádku s určeným sdělením, jelikož samotný příjemce vysílání nemá uživatelské rozhraní (obdobně jako služba).

Implementace ve zdrojovém kódu aplikace je realizována formou podtřídy třídy `BroadcastReceiver`.

3.2.4 Poskytovatel obsahu

Poskytovatel obsahu má na starosti správu a zprostředkování sdílených aplikačních dat (sms, kontakty, webové záložky apod.) mezi jednotlivými aplikacemi. Data mohou být uložena v souborovém systému, SQLite databázi, na internetu či na jiných úložištích. Mají-li přidělená příslušná práva, mohou jednotlivé aplikace skrze poskytovatele obsahu k daným datům přistupovat (ať už pro čtení či zápis). Využití této komponenty najdeme také u manipulace s privátními daty v rámci jedné aplikace.

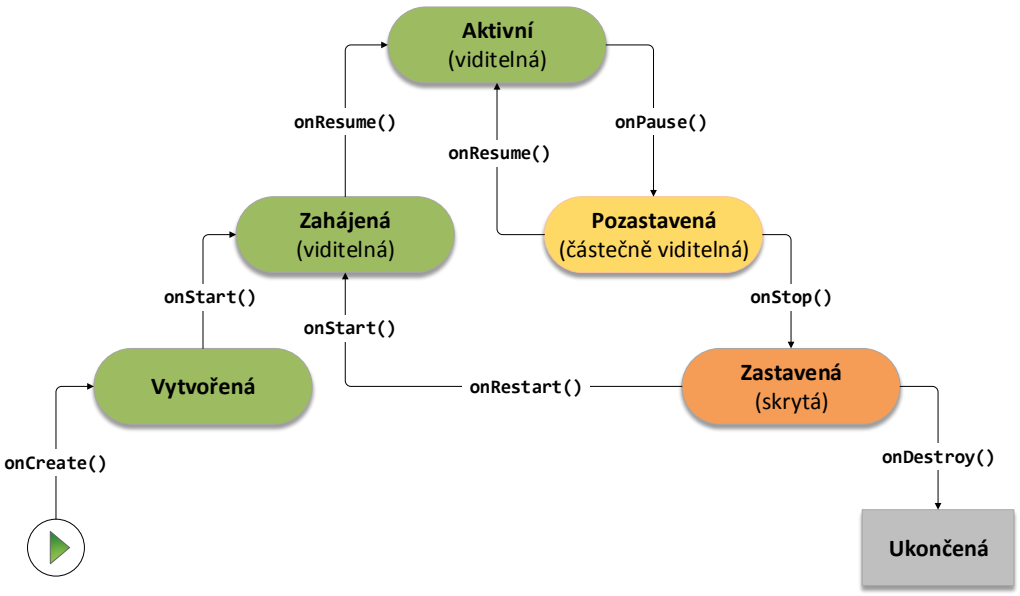
Ve zdrojovém kódu aplikace je komponenta implementována jako podtřída třídy `ContentProvider`.

3.3 Životní cyklus aktivity

Správné řízení životních cyklů dílčích komponent pomocí jejich klíčových metod je zcela stěžejní pro vývoj efektivní aplikace. Na rozdíl od jiných známých přístupů, kdy je program či aplikace spuštěna pomocí hlavní metody `main()`, OS Android spouští kód aplikace pomocí speciálních metod se zpětným voláním, které odpovídají konkrétní fázi životního cyklu komponenty. Voláním těchto metod v určené posloupnosti se docílí spuštění či ukončení dané komponenty [24], [25].

Obecně lze říci, že nejpodstatnějším životním cyklem aplikace je cyklus aktivity, neboť ta jest zodpovědná za zprostředkování uživatelského rozhraní. Správnou implementací jednotlivých metod se zpětným voláním lze zamezit situacím jako je např. pád aplikace v případě jejího přerušení jinou aplikací, čerpání systémových prostředků i v případě, že aplikace není využívána, či ztrátě prováděných činností při rotaci displeje zařízení.

Zjednodušené schéma životního cyklu aktivity je vidět na obrázku 3.3. Pro lepší ilustraci je jeho uspořádání ve schodovitém pyramidovém tvaru. Dílčí fáze životního cyklu aktivity představují jednotlivé schody pyramidy (ve formě barevných bloků) a jednotlivé kroky (ve formě šipek) označují volané metody. Po počátečním vytvoření instance třídy `Activity` se voláním jednotlivých metod dostáváme vždy o jeden schod pyramidy výše. Vrchol pyramidy (blok *Aktivní*) představuje stav, ve kterém se aktivita nachází na popředí obrazovky a uživatel s ní může manipulovat. V případě,



Obr. 3.3: Životní cyklus aktivity [24]

že uživatel aktivitu opouští, jsou postupně volány metody pro její odvolání z popředí obrazovky. Dojde-li až na metodu `onDestroy()`, aktivita je zcela zrušena a jsou ji odebrány veškeré systémové prostředky [24].

Důležité je poznamenat, že pouze tři fáze tohoto diagramu představují fáze statické, čili pouze v těchto třech stavech může aktivita setrvat po delší dobu [24]:

- *Aktivní* – jak již bylo zmíněno, jedná se o fázi, ve které aktivita běží na popředí a má pozornost uživatele;
- *Pozastavená* – jedná se o stav, kdy pozornost uživatele přebírá jiná aktivita, která ovšem překrývá pouze určitou část obrazovky (či je částečně průhledná). Původní aktivita tak stále zůstává částečně viditelná. V tomto stavu ovšem nemůže původní aktivita spouštět žádný kód či reagovat na vstupy uživatele;
- *Zastavená* – v tomto stavu již není aktivita pro uživatele viditelná a běží na pozadí, čili stále má přidělenou paměť, udržuje stavové informace atd., ale nemůže spouštět žádný kód.

V ostatních fázích diagramu se pouze vykoná předepsaný kód a dochází k okamžitému přesunu do další fáze životního cyklu komponenty.

Pokud se aktivita nachází ve stavu *Pozastavená* nebo *Zastavená*, může být systémem v případě nedostatku systémových prostředků požádána o ukončení nebo ji systém ukončí bez varování. V takovém případě je ji pro opětovné spuštění potřeba znovu vytvořit zavoláním metody `onCreate()` [24].

4 ANALÝZA PROBLEMATIKY

Stěžejní části vyvíjené aplikace, jejíž úkolem je získávat informace o tom, které mobilní aplikace jsou uživatelem nejčastěji využívány, je mechanismus měření aktivní doby běhu instalovaných aplikací v OS Android. Pojmem aktivní doba je zde myšlen čas, po který uživatel danou aplikaci aktivně využívá, tj. aplikace běží na popředí obrazovky. Jedná se o stav *Aktivní* dle životního cyklu aktivity zobrazeného v části 3.3 na obrázku č. 3.3.

Náplní této kapitoly je analýza problematiky týkající se výše uvedeného mechanismu. Cílem první části kapitoly je ověřit, zda je možné takovýto mechanismus v OS Android prakticky zrealizovat či zda je dostupné již existující řešení, které by bylo možné pro účely vyvíjené aplikace využít. Součástí je také diskuze úskalí spojených s realizovatelností daného mechanismu. Druhá část kapitoly se věnuje popisu poskytovaných vlastností nalezené optimální varianty a ověření její funkčnosti.

4.1 Ověření řešitelnosti zadání

Úvodním krokem k nalezení způsobu řešení uvedeného mechanismu byla analýza, zda existují mobilní aplikace pro OS Android, jež poskytují funkcionalitu na měření aktivní doby strávené v instalovaných aplikacích. Pomocí online distribuční služby *Google Play* bylo nalezeno několik aplikací, jež se touto problematikou zabývá a prakticky ji řeší.

Pro testovací účely byly zástupně vybrány aplikace, jež funkci na měření aktivní doby běhu aplikací poskytují bezplatně. Vybranými aplikacemi jsou: *QualityTime*, *App Usage Tracker*, *App Usage*, *OFFTIME* a *RescueTime*. Uvedené aplikace byly naistalovány na mobilní zařízení značky Samsung v modelu Nexus S[1]. Po dobu jednoho týdne byly vybrané aplikace testovány z hlediska správnosti poskytovaných informací o aktivní době běhu aplikací. Referenční hodnoty byly získány na základě manuálního stopování doby strávené v aplikacích daného zařízení. Správné výsledky měření odpovídající skutečné aktivitě na mobilním zařízení poskytly všechny testované aplikace kromě aplikace *RescueTime*, jejíž výsledky měření byly velmi nepřesné a neodpovídaly realitě.

Na základě výsledků testování lze konstatovat, že mechanismus na měření aktivní doby běhu aplikací lze v OS Android prakticky realizovat. Žádná z těchto aplikací ovšem není vyvíjená jako svobodný software, čili způsob řešení daného mechanismu je know-how vývojářů jednotlivých aplikací.

[1] Ve verzi OS Android 4.1.2. s označením Jelly Bean.

4.1.1 Diskuze vzniklých úskalí

Při návrhu mechanismu měření aktivní doby běhu instalovaných aplikací v OS Android a hledání vhodného řešení se naráží na několik zásadních úskalí. Prvním z nich je detekce, ve kterém stavu či životním cyklu se instalovaná aplikace nachází. Druhým úskalím je problém s odlišnou úrovní bezpečnosti a tím i funkčností určitých části systému u jednotlivých verzí OS Android.

Problém detekce stavu aplikace

Fáze životního cyklu, ve kterém se aplikace, resp. aktivita, nachází na popředí obrazovky a uživatel s ní může manipulovat, je označena jako *Aktivní* (viz část 3.3). Doba běhu aktivity v této fázi životního cyklu představuje aktivní dobu využívání aplikace uživatelem. Pro účely měření je tedy potřeba detekovat začátek aktivní fáze (vytvoření aktivity nebo přechod z fáze *Zahájená*) a konec aktivní fáze (přechod do fáze *Pozastavená* či *Zastavená*) u cizí aplikace. Problémem je, že přístup k informacím, ve které fázi životního cyklu se jednotlivé komponenty cizí aplikace nacházejí, má z bezpečnostního hlediska pouze samotný operační systém. Neexistuje žádné veřejné hlášení systému, které by oznamovalo, v jaké fázi se cizí aplikace nacházejí, a jež by mohlo být jednoduše zachyceno vyvíjenou aplikací např. pomocí příjemce vysílání.

Jednou z možnosti, jak se dostat k potřebným informacím, je získání oprávnění administrátora zařízení, neboli požádat uživatele o tzv. „rooting“ zařízení. Z praktického i bezpečnostního hlediska je ovšem podobný postup nepřípustný.

Druhou možností, zjištěnou na základě analýzy potřebných oprávnění u aplikací zabývající se obdobnou problematikou, je využití veřejné třídy `ActivityManager` [26]. Tato třída je zodpovědná za celkovou správu životního cyklu aktivity a má informace o všech vytvořených aktivitách v systému. Tyto informace jsou uchovávány v zásobníku, jehož vrchol představuje aktivita, která se aktuálně nachází ve fázi *Aktivní*, čili je aktivně využívána uživatelem. Data ze zásobníku lze získat zavoláním metody `getRunningTasks(int maxNum)`, kde `maxNum` představuje počet požadovaných záznamů (jeden záznam reprezentuje jednu aktivitu). Přistoupením k nultému záznamu (vrchol zásobníku) můžeme načíst atribut `topActivity` a prostřednictvím vnořené metody `getPackageName()` získat název aplikace, jejíž aktivita běží na popředí obrazovky. Voláním metody `getRunningTask(int maxNum)` ve vhodně určených intervalech si lze udržovat přehled o tom, která aplikace je uživatelem aktuálně využívaná a na základě této znalosti měřit její aktivní dobu běhu. Při snaze využít tento přístup se ovšem naráží na další problém (kompatibilita napříč úrovněmi Android API – viz dále).

Problém u novějších verzí OS Android

Jak bylo uvedeno v úvodu 3. kapitoly, jednotlivé verze s odlišnou úrovni API disponují různou vybaveností a funkčností. Ve většině případů jsou změny v nové verzi systému aditivní, čili verze s vyšší úrovni API zůstává kompatibilní s verzí, jež má úroveň API nižší. Existují ale i výjimky, kdy je funkcionalita určitých částí systému potlačena, modifikována či odstraněna. Nejčastějším důvodem těchto změn je zajištění vyšší bezpečnosti u nové verze systému [27].

Obdobnou změnu zaznamenala verze OS Android 5.0 s názvem Lollipop[2], u které došlo ke zvýšení bezpečnosti systému a tím k omezení funkčnosti některých zranitelných části. Tato změna postihla také výše uvedenou metodu `getRunningTasks(int maxNum)` [28], která již není dostupná pro aplikace třetích stran, čili pro aplikace instalované uživatelem. Přístup k zásobníku s informacemi o spuštěných aktivitách v systému je vyvíjené aplikaci zamezen. Výše uvedený příklad měření aktivní doby běhu aplikací u této a vyšší verze systému tak nelze použít. Verze OS Android 5.0 tímto rozděluje možnosti měření aktivní doby běhu aplikací do dvou různých oblastí řešení – pro systémy s úrovní API nižší než 21 a pro systémy s úrovní API rovnu či větší než 21.

Na základě diskuze těchto úskalí s pracovníky Policie ČR bylo hledání optimálního řešení z hlediska perspektivy zaměřeno na verzi OS Android 5.0 a vyšší.

4.2 Nalezení optimální varianty

Ačkoliv byla funkčnost některých části systému z bezpečnostního hlediska potlačena, přibyla ve verzi OS Android 5.0 celá řada nových balíčků, jež přidávají do systému funkce nové. Výsledkem hledání vhodné alternativy k řešení mechanismu měření aktivní doby běhu pomocí třídy `ActivityManager` a metody `getRunningTasks(int maxNum)` je nalezení API, resp. balíčku `android.app.usage` [29]. Jedná se o nový balíček dostupný ve verzích systému s úrovni API 21 a vyšší.

4.2.1 Popis nabízených vlastností

Nalezené API disponuje několika třídami, přičemž pro účely této práce jsou nejdůležitější třídy `UsageStats` a `UsageStatsManager`.

První z nich sdružuje a reprezentuje dílčí informace o využívání konkrétního aplikačního balíčku (např. časové razítko prvotního či posledního spuštění aplikace) a poskytuje celou řadu veřejných metod, pomocí kterých lze tyto dílčí informace

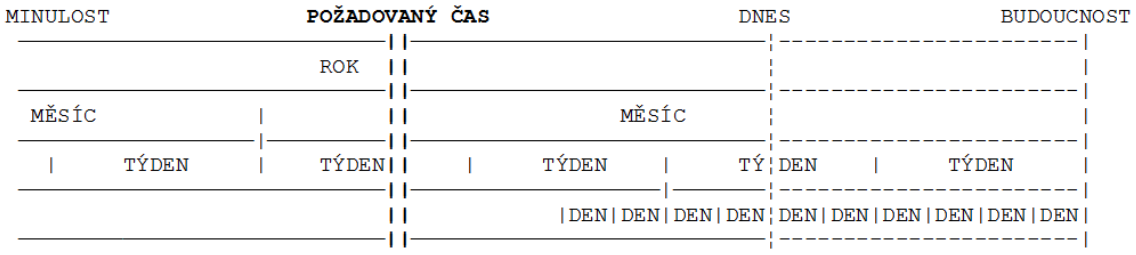
[2]Odpovídá úrovni API 21.

získat. Z hlediska získání informací o aktivní době běhu určité aplikace je nejvýznamnější metodou metoda `getTotalTimeInForeground()`, která v jednotkách milisekund poskytuje celkový čas, po který aplikace běží na popředí obrazovky.

Druhá ze zmíněných tříd poskytuje přístup k historii a statistikám využívání jednotlivých aplikací nacházejících se v mobilním zařízení. Jednotlivé údaje jsou seskupeny dle daných časových intervalů (dny, týdny, měsíce a roky), přičemž pro každý časový interval existuje maximální doba, po kterou jsou statistiky skrze danou třídu v systému dostupné. Maximální doba uchovávání dat o využívání jednotlivých aplikací dle časových intervalů je následující [30]:

- denní interval: 7 dnů,
- týdenní interval: 4 týdny,
- měsíční interval: 6 měsíců,
- roční interval: 2 roky.

Nejzajímavější veřejnou metodou třídy `UsageStatsManager` pro účely této diplomové práce je metoda `queryUsageStats(int intervalType, long beginTime, long endTime)`. Parametr `intervalType` označuje zvolený časový interval, podle kterého budou statistiky využití agregovány, a parametry `beginTime` a `endTime` označují začátek a konec časového rozsahu, ze kterého chceme data získat. Návratovou hodnotou této metody je list obsahující objekty třídy `UsageStats`, ze kterých je následně možné získat např. aktivní dobu běhu konkrétních aplikací. Princip získávání informací z daného časového úseku je dle oficiální dokumentace znázorněn na obrázku 4.1[31].



Obr. 4.1: Ukázka získání informací ze třídy UsageStatsManager [31]

Příklad získání statistik, kdy je časový rozsah menší než zadaný časový interval, je následující: pokud je zadán časový rozsah daný parametry `beginTime` a `endTime` (např. 2 po sobě jdoucí dny z předminulého týdne) s určitým časovým intervalem daným parametrem `intervalType` (např. týdenní interval), metoda `queryUsageStats()` vrátí statistiky za celý časový interval – v tomto případě za celý předminulý týden. Pokud je naopak časový rozsah (např. 3 po sobě jdoucí týdny)

větší než zadaný časový interval (např. týdenní interval), metoda vrátí list obsahující tři jedno-týdenní statistiky pro jednotlivé týdny z časového rozsahu (seřazeno od nejstaršího týdne).

Alternativně lze využít také např. metodu `queryAndAggregateUsageStats(long beginTime, long endTime)`, jejíž návratovou hodnotou je mapa, kde klíčem je název balíčku konkrétní aplikace (objekt typu `String`) a hodnotou je opět objekt třídy `UsageStats`. Na rozdíl od předchozí zmiňované metody není u metody `queryAndAggregateUsageStats()` parametr `intervalType`, jelikož časový interval je zde určen automaticky na základě zvoleného časového rozsahu zadaným parametry `beginTime` a `endTime`.

Pro správnou funkčnost a použití tříd `UsageStatsManager` a `UsageStats` je vyžadováno oprávnění `android.permission.PACKAGE_USAGE_STATS`. Uvedené oprávnění je potřeba deklarovat v souboru `AndroidManifest.xml`. Současně je potřeba, aby uživatel po nainstalování aplikace obsahující dané API manuálně povolil přístup k historií zařízení skrze nastavení systému (*Nastavení* > *Zabezpečení* > *Aplikace s přístup. k využívání*) [30].

4.2.2 Ověření funkčnosti

Cílem této části je ověřit funkčnost poskytovaných vlastností daného API. Pro tyto účely byla vytvořená jednoduchá testovací aplikace[3], ve které bylo dané API implementováno (včetně deklarace vyžadovaného oprávnění v manifestu aplikace a také povolení potřebného oprávnění v nastavení zařízení). Úkolem testovací aplikace je vypisovat do konzole výstupy volání metody `queryUsageStats()` třídy `UsageStatsManager` pro různě zadané hodnoty parametrů dané metody. Zjednodušený kód stěžejní části testovací aplikace je uveden v příloze A.

Na základě testování byly odhaleny nedostatky, které lze rozdělit na dva zásadní problémy. Prvním problémem je nemožnost načtení statistik z předchozích časových intervalů, ačkoliv by dle dokumentace měly být k dispozici. Získat lze pouze statistiky, které se vztahují k aktuálně probíhajícímu časovému intervalu, tzn. je možné načíst statistiky z dnešního dne (statistiky jsou spočtené do okamžiku načtení dat), aktuálně probíhajícího týdne, měsíce či roku. Při snaze získat statistiky za předchozí den, týden, měsíc či rok vrátí metoda `queryUsageStats()` prázdný list. Při analýze tohoto problému byly nalezeny zdroje, které poukazují na totožný problém (např. [32], [33]). Oprava chyby není v současnosti k dispozici. Druhý problém se týká počátků časových intervalů, které z neznámého důvodu začínají v jiné dny, než by se intuitivně očekávalo a není respektováno konkrétní nastavení daného

[3]Testovací aplikace není součástí příloh diplomové práce – sloužila pouze k ověření funkčnosti daného API.

zařízení (počátek týdne a časová zóna). Problémem je to zejména proto, že API `android.app.usage` neposkytuje možnost nastavit či určit počátek daných časových intervalů dle konkrétní potřeby vývojáře.
Po několika týdenním testování poskytovaných vlastností třídy `UsageStatsManager` bylo v této věci zjištěno, že:

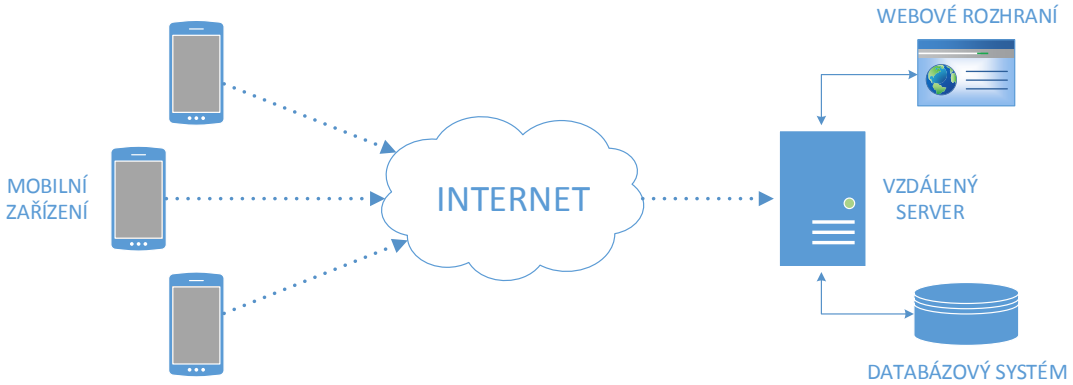
- počátek nového týdne nepřipadá na pondělí či neděli (dle nastavení operačního systému), ale na čtvrtek;
- začátek nového měsíce nepřipadá na první den v měsíci, ale na některý z posledních dnů měsíce předcházejícího, např. na 26. či 27. den předchozího měsíce;
- v časové zóně UTC+1 (Coordinated Universal Time) jsou statistiky z daného dne dostupné i ve dni následujícím a to v čase 00:00 až 00:59, tzn. v čase mezi půlnoci a jednou hodinou ranní vrací metoda `queryUsageStats()` statistiky, které se týkají dne předchozího.

Navzdory všem zmíněným nedostatkům daného balíčku, se prakticky jedná o jediné dostupné API operačního systému Android, které poskytuje přístup k historii využívání mobilního zařízení a umožňuje získat informace o aktivní době běhu jednotlivých aplikací, jež se v mobilním zařízení nacházejí. Z tohoto důvodu bude vybrané API využito při realizaci systému na zlepšení prevence kyberšikany z pohledu uživatelské aplikace. K samotnému API bude ovšem potřeba navrhnout a dodat logiku, která zjištěné nedostatky nahradí vlastními prostředky. Pomocí kombinace fungujících částí balíčku `android.app.usage` (získání statistik z aktuálně probíhajícího dne) a vlastní obslužné logiky bude možné řešit stěžejní část aplikace, kterou je získání informací o tom, jaké mobilní aplikace uživatelé využívají a kolik času v nich tráví.

5 REALIZACE SYSTÉMU

Systém jako celek se skládá z uživatelské aplikace pro operační systém Android a vzdáleného serveru. Hlavním úkolem uživatelské aplikace je sběr dat o využívání jednotlivých aplikací nacházejících se v daném mobilním zařízení a jejich pravidelné zasílání na vzdálený server v určených intervalech. Součástí zasílaných dat jsou také informace o věku a pohlaví uživatele. Hlavním úkolem vzdáleného serveru je příjem dat od jednotlivých mobilních zařízení a jejich zpracování a vyhodnocení dle požadavků Policie ČR.

Zjednodušený koncept realizovaného systému je znázorněn na obrázku 5.1. Bližšímu popisu jednotlivých částí systému se věnují následující podkapitoly.



Obr. 5.1: Schéma konceptu realizovaného systému

5.1 Android aplikace

Hlavním zdrojem informací potřebných pro vývoj aplikace je online dokumentace určená vývojářům mobilních aplikací OS Android [34] a s ní související instruktážní průvodci [35], [36]. V uvedených zdrojích je v případě potřeby možné nalézt doplňující informace potřebné k hlubšímu pochopení jednotlivých části vyvíjené aplikace.

Při návrhu a realizaci uživatelské aplikace byl kladen důraz na:

- minimální množství potřebných oprávnění,
- minimální spotřebu baterie,
- úsporu síťových prostředků (jak z hlediska množství přenesených dat, tak z hlediska četnosti zasílání dat),

- pokrytí co možná největšího množství scénářů, které mohou nastat (vypnutí či restart mobilního telefonu, nedostupné síťové připojení, zařízení v režimu spánku, apod.).

Základní myšlenkou mechanismu na sběr dat[1], s ohledem na výše uvedené důrazy, je získat statistiky využití z aktuálně probíhajícího dne za pomocí balíčku `android.app.usage` a na konci každého dne je uložit do privátní databáze aplikace. Následně, po určitém časovém intervalu, je sumarizovat a jakožto celek zaslat na vzdálený server. Jako časový interval pro sumarizaci statistik získaných z jednotlivých dnů byl zvolen interval týdenní. Z hlediska úspory síťových prostředků a spotřeby baterie se jedná o velmi šetrnou variantu, jelikož ke spojení se vzdáleným serverem dochází pouze jednou týdně. Z pohledu vyhodnocování dat na straně serveru pro účely Policie ČR se taktéž jedná o optimální interval.

Dále bylo snahou docílit nezávislosti vnitřní logiky aplikace na uživatelském rozhraní, tzn. zajištění kontinuálního fungování mechanismu na sběr statistik bez ohledu na to, zda je vyvíjená aplikace uživatelem aktivně využívána či nikoliv.

5.1.1 Volba klíčových komponent

Cílem této části práce je stručně představit klíčové komponenty, pomoci kterých je realizována vnitřní logika vyvíjené aplikace a zdůvodnit jejich výběr.

V části 4.2 byly popsány vlastnosti API `android.app.usage` spolu s jeho nedostatky. Navzdory všem zmíněným nedostatkům bylo toto API vybráno jako základní nástroj pro získání informací ohledně aktivní doby běhu aplikací instalovaných v mobilním zařízení – konkrétně k získání statistik z aktuálně probíhajícího dne. Aby bylo možné získat sumarizované statistiky za uplynulý týden, jež se následně automaticky zašlou na vzdálený server, je zapotřebí implementovat i jiná API či komponenty OS Android.

AlarmManager

Vzhledem ke snaze docílit nezávislosti vnitřní logiky aplikace na uživatelském rozhraní je potřeba zajistit, aby mechanismus na sběr statistik využití zařízení fungoval i nad rámec životního cyklu aplikace, resp. aktivit nacházejících se ve vyvíjené aplikaci.

K zajištění fungování mechanismu na sběr statistik bez ohledu na využívání vyvíjené aplikace uživatelem bude sloužit třída `AlarmManager`. Třída poskytuje přístup ke službám alarmů OS Android a umožňuje spuštění a vykonání určité operace na základě časových požadavků mimo životní cyklus aktivit vyvíjené aplikace. Nabízí

[1] Jako data jsou zde myšleny informace, resp. statistiky o využívání mobilního zařízení z pohledu celkové aktivní doby běhu instalovaných aplikací v OS Android.

možnost nastavit a vyslat žádost o spuštění jiné komponenty (třída `PendingIntent`), která je následně odchycena příjemcem vysílání (třída `BroadcastReceiver`), jenž zajistí spuštění komponenty uvedené v zaslané žádosti. Danou komponentou může být např. služba, jež poběží na pozadí za účelem provádění definovaných úkonů. V případě vyvíjené aplikace se jedná např. o úkony získávání a ukládáni statistik využití zařízení do privátní databáze. Spouštět libovolný kód pomocí alarmů je navíc možné nejen v případě, kdy vyvíjená aplikace neběží, ale i během režimu spánku mobilního zařízení.

Třída `AlarmManager` poskytuje celou řadu veřejných metod a možností, jak nastavit potřebný alarm. Nejpodstatnějšími metodami (bez uvedení jejich parametrů) jsou:

- `set()`: nastaví jednorázový alarm na určený čas,[2]
- `setWindow()`: nastaví jednorázový alarm ke spuštění pro definované časové okno,
- `setRepeating()` a `setInexactRepeating()`: nastaví opakující se alarm s definovanou periodou opakování.

Společné parametry uvedených metod jsou: typ alarmu (viz níže) a objekt žádosti o spuštění jiné komponenty (třída `PendingIntent`). V případě opakujícího se alarmu je navíc k dispozici parametr určující čas spuštění alarmu v jednotkách milisekund a interval mezi jednotlivým spouštěním alarmu opět v jednotkách milisekund. Pro alarm s definovaným časovým oknem určují předchozí zmíněné parametry začátek a konec časového okna.

Typy alarmů, které je možné využít jsou následující:

- `ELAPSED_REALTIME`: spouštění úkonu na základě množství uplynulého času od doby spuštění zařízení (včetně času po který bylo zařízení v režimu spánku), např. spuštění alarmu po 30 minutách,
- `ELAPSED_REALTIME_WAKEUP`: obdoba předchozího s rozdílem schopnosti vzbudit mobilní zařízení z režimu spánku,
- `RTC`: spouštění úkonu na základě specifikovaného času, např. spuštění alarmu ve 12:00 hod.,
- `RTC_WAKEUP`: obdoba předchozího s rozdílem schopnosti vzbudit mobilní zařízení z režimu spánku.

Nastavené alarmy je možné v případě potřeby zrušit pomocí metody `cancel()`. Při restartu zařízení dochází automaticky ke zrušení všech nastavených alarmů.

[2]Od verze Android API 19 není určený čas alarmu striktní a ke spuštění může dojít se zpožděním, jelikož snahou operačního systému je seskupit více alarmů registrovaných na podobný čas a spustit je v jedné dávce za účelem snížení spotřeby baterie zařízení. Pro striktní spuštění alarmu je potřeba použít metodu `setExact()`.

Podrobnější informace o třídě `AlarmManager` je možné získat z online dokumentace OS Android [37], [38].

JobScheduler

Jedná se o API Android frameworku sloužící k plánování úloh, jež jsou vykonávány na pozadí systému jako proces dané aplikace.[3] Na rozdíl od třídy `AlarmManager`, která umožňuje na pozadí systému spouštět kód na základě specifického času či časového intervalu, třída `JobScheduler` poskytuje možnost spouštění kódu na základě splnění definovaných podmínek. Mezi tři základní podmínky patří:

- požadavek na dostupné síťové připojení;
- požadavek na stav, kdy zařízení nevykonává žádné náročné operace, tj. stav nečinnosti zařízení;
- požadavek na stav, kdy je zařízení připojeno ke zdroji elektrické energie.

Naplánovaná operace či úkon je spuštěn pouze v případě splnění požadované podmínky. Pokud podmínka splněna není, zůstává požadovaná operace v seznamu naplánovaných úloh do doby naplnění definované podmínky (s výjimkou nastavení kritéria na nejzazší časovou hranici vykonání operace – viz níže). Mimo tři výše uvedená základní kritéria je dále možné nastavit:

- pozdržení spuštění úkonu o definovaný čas;
- způsob zopakování spuštění úkonu při jeho neúspěšném vykonání, tzn. pokud v průběhu vykonávání operace dojde k porušení definovaných kritérií (např. dojde k vypnutí připojení k internetu při nastaveném požadavku na dostupné síťové připojení);
- opakované spouštění úkonu s definovanou periodou;
- nastavení nejzazší časové hranice, při jejíž překročení bude úkon vykonán bez ohledu na stanovené podmínky;
- předání dodatečných informací pro následně spuštěnou službu.

Jednotlivé podmínky jsou konstruovány pomocí objektu třídy `JobInfo.Builder`, jehož součástí je naplánovaná úloha, tj. služba, která je definována třídou `JobService`. Bližší popis fungování API na příkladu konkrétní implementace v rámci vyvíjené aplikace se nachází v části 5.1.3.

Díky schopnosti automaticky detekovat síťové připojení a následně spustit požadovanou službu, bude třída `JobScheduler` využita jako nástroj ke spuštění registrace aplikace na vzdáleném serveru a ke spuštění operací pro zpracování statistik z uplynulého týdne a jejich zaslání na vzdálený server. Podrobnější informace o API

[3] Obdobně jako balíček `android.app.usage` je toto API dostupné od verze OS Android 5.0. (tj. od úrovně Android API 21 a vyšší).

`JobScheduler` je možné nalézt v online dokumentaci OS Android [39] či souvisejícím instruktážním průvodci [40].

SQLite

Vzhledem k tomu, že pomocí API `android.app.usage` lze spolehlivě získat pouze statistiky využití z aktuálně probíhajícího dne, je potřeba tato data průběžně ukládat do privátního úložiště aplikace za účelem uchovávání statistik z uplynulých dnů a možnosti jejich následné sumarizace do týdenního souhrnu. K účelům ukládání statistik bude využito SQLite databáze, jež je součástí Android frameworku. Potřebná API pro obsluhu databáze jsou dostupná v balíčku `android.database.sqlite` [41].

SharedPreferences

Pro ukládání informací jako je rok narození a pohlaví uživatele, jedinečné identifikační číslo aplikace, či jiné hodnoty potřebné pro správnou funkci vnitřní logiky vyvíjené aplikace bude použito API `SharedPreferences`. Jedná se o privátní či sdílené perzistentní úložiště (dle nastavení) pro ukládání dat typu klíč-hodnota. Bližší informace lze nalézt v online dokumentaci [42].

5.1.2 Struktura aplikace

Cílem této části je stručně přiblížit strukturu vyvíjené aplikace. Základní kořenová struktura projektu vyvíjené Android aplikace respektuje standartní strukturu projektů vytvářených v rámci vývojového prostředí *Android Studio*. Nejpodstatnějším adresářem z pohledu vytvořeného kódu je adresář `KYBER_apka/app/src/main/`, jehož obsahem je:

- `AndroidManifest.xml`: jedná se o XML soubor, jenž poskytuje operačnímu systému Android nezbytné informace o vyvinuté aplikaci. Nachází se zde např. deklarace vytvořených komponent aplikace (aktivit, služeb, příjemců vysílání apod.), deklarace potřebných práv pro správnou funkčnost aplikace, určení minimální úrovně Android API atp.;
- `res/`: adresář určený pro prostředky aplikace. Patří zde soubory obsahující definice rozvržení jednotlivých částí uživatelského rozhraní, použité ikony či obrázky, textové řetězce apod.;
- `java/`: adresář určený pro zdrojové kódy jazyka JAVA.

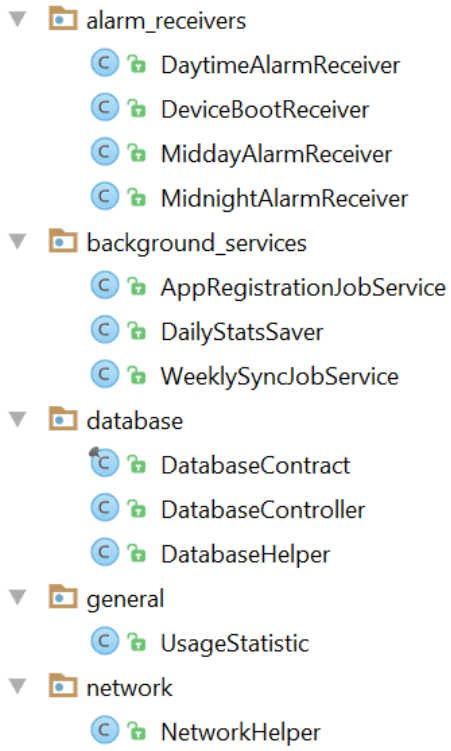
Výpis souboru `AndroidManifest.xml` lze nalézt v příloze B. Nejpodstatnější soubory adresáře `res/` jsou zmíněny v části 5.1.4 věnované uživatelskému rozhraní aplikace. V rámci adresáře `java/`[4] je vytvořena dílčí struktura, která jednotlivé JAVA

[4]Resp. `java/com/xbolek00/kyberapka`.

soubory logicky rozděluje do jednotlivých balíčků. Jednotlivé balíčky je možné prakticky rozdělit do dvou skupin. První skupina reprezentuje kód vnitřní logiky aplikace, tj. mechanismu na získávání statistik využití aplikací nacházejících se v mobilním zařízení uživatele (včetně jejich zpracování a zasílání na vzdálený server). Druhá skupina reprezentuje kód týkající se fungování uživatelského rozhraní aplikace.

Třídy vnitřní logiky aplikace

Na obrázku 5.2 je vyobrazena struktura vytvořených balíčků a jejich tříd. Popisu vzájemné interakce nejpodstatnějších tříd a jejich účelu se věnuje část 5.1.3 zabývající se fungováním vnitřní logiky aplikace. Následující text je pouze stručným přehledem jednotlivých balíčků.



Obr. 5.2: Třídy reprezentující vnitřní mechanismus aplikace

alarm_receivers Jedná se o příjemce vysílání, jež jsou aktivováni v různou denní dobu pomocí nastavených alarmů. Jednotliví příjemci vysílání mají v rámci vnitřního mechanismu různou úlohu (koordinace dílčích operací, spouštění příslušných komponent, plánování úloh, obnova nastavení potřebných alarmů atp.) Diagram tříd z pohledu jejich dědičnosti se nachází v příloze na obrázku C.1.

background_services Jedná se o služby, jež jsou při splnění definovaných podmínek spuštěny na pozadí systému za účelem vykonání definovaných operací. Jedná se například o zajištění registrace aplikace na vzdáleném serveru, získávání denních statistik využití jednotlivých aplikací a jejich ukládání do privátní databáze či zpracování týdenních sumarizovaných statistik a jejich zaslání na vzdálený server. Diagram tříd z pohledu jejich dědičnosti se nachází v příloze na obrázku C.2.

database Jedná se o třídy zprostředkovávající obsluhu SQLite databáze ve vyvíjené aplikaci. Třída `DatabaseController` zapouzdřuje přístup k databázi a obsahuje metody pro načítání a ukládání záznamů z/do jednotlivých tabulek databáze. Třída `DatabaseHelper` poskytuje metody potřebné pro vytvoření, aktualizaci, otevření či zavření databáze, definuje její název, verzi apod. Třída `DatabaseContract` specifikuje názvy tabulek databáze a jejich sloupců.

Ve vyvíjené aplikaci jsou vytvořeny celkem dvě tabulky. První tabulka má název `DailyUsageStats` a slouží k ukládání denních statistik využití jednotlivých aplikací mobilního zařízení uživatele (viz tab. 5.1). Druhá tabulka `WeeklyUsageStats` (viz tab. 5.2) slouží k ukládání sumarizovaných týdenních statistik spočtených na základě záznamů v tabulce `DailyUsageStats`. Odpovídající diagram tříd se nachází v příloze na obrázku C.3.

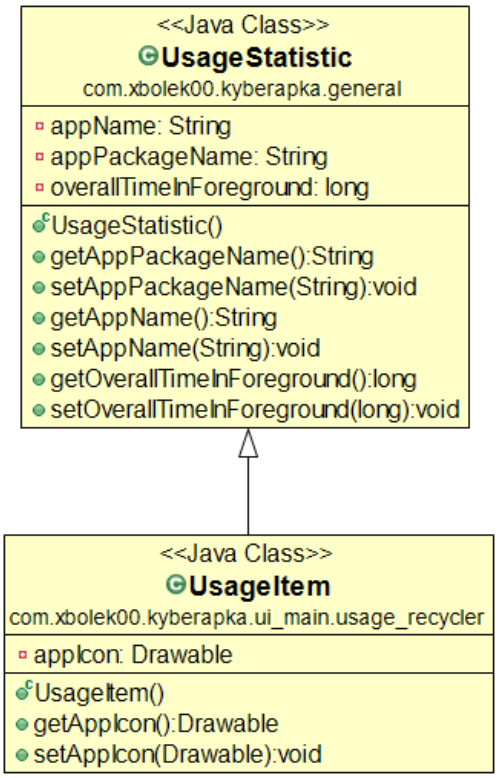
Tab. 5.1: Tabulka pro denní statistiky

`DailyUsageStats`
`month` (měsíc)
`weekOfYear` (týden v roce)
`dayOfWeek` (den v týdnu)
`packageName` (jméno balíčku aplikace)
`appName` (jméno aplikace)
`timeInForeground` (aktivní doba běhu)
`timeStamp` (časové razítko)

Tab. 5.2: Tabulka pro týdenní souhrny

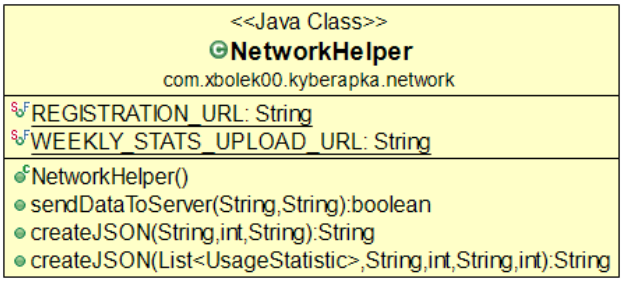
`WeeklyUsageStats`
`weekOfYear` (týden v roce)
`packageName` (jméno balíčku aplikace)
`appName` (jméno aplikace)
`timeInForeground` (aktivní doba běhu)
`timeStamp` (časové razítko)

general Zde se nachází jednoduchá pomocná třída `UsageStatistic`, jež reprezentuje instalovanou aplikaci v zařízení uživatele spolu s dobou jejího aktivního běhu. Objekt dané třídy obsahuje informace jako je jméno balíčku aplikace, název aplikace a celková doba jejího běhu na popředí obrazovky. Ukázka třídy je na obrázku 5.3 (podtřída `UsageItem` je používána v kontextu uživatelského rozhraní aplikace).



Obr. 5.3: Třídy `UsageStatistic` a `UsageItem`

network Pomocná třída `NetworkHelper` zapouzdřuje síťové operace spolu s metodami pro tvorbu zasílaných JSON[5] řetězců. Obsahuje také statické proměnné s příslušnými URI[6] potřebnými pro komunikaci se vzdáleným serverem. Ukázka třídy je na obrázku 5.4.

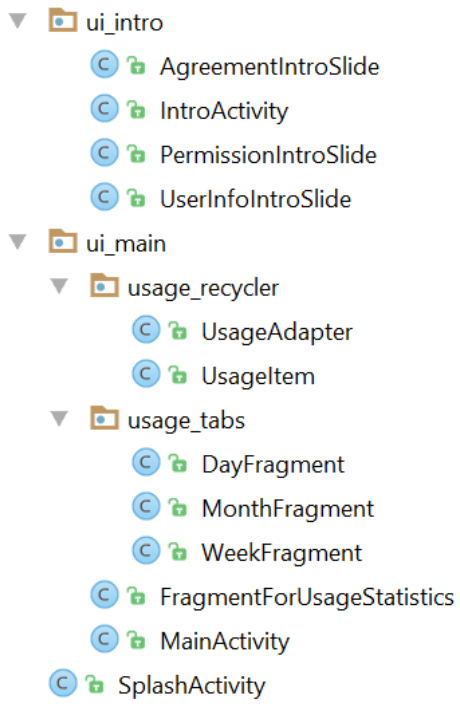


Obr. 5.4: Třída `NetworkHelper`

[5]JavaScript Object Notation.
[6]Uniform Resource Identifier.

Třídy uživatelského rozhraní aplikace

Struktura vytvořených balíčků týkajících se tříd uživatelského rozhraní je na obrázku 5.5. Balíček `ui_intro` obsahuje třídy potřebné pro fungování té části uživatelského rozhraní, jež je zobrazena pouze při prvotním spuštění aplikace. Zároveň se jedná o jedinou část uživatelského rozhraní, jež souvisí s fungováním vnitřního mechanismu aplikace (v rámci třídy `IntroActivity` dochází k jeho aktivaci – viz dále). Balíček `ui_main` obsahuje třídy reprezentující hlavní uživatelské rozhraní vyvíjené aplikace. Účel jednotlivých tříd je vysvětlen v kontextu části 5.1.4 věnované uživatelskému rozhraní.



Obr. 5.5: Třídy reprezentující uživatelské rozhraní aplikace

5.1.3 Vnitřní logika

Fungování mechanismu na získávání statistik využití aplikací nacházejících se v mobilním zařízení uživatele je možné rozdělit na tři hlavní fáze: prvotní spuštění aplikace, průběžné získávání statistik využití a detekci nového týdne spolu se zasláním dat na vzdálený server. Z pohledu těchto tří hlavních fází bude vysvětlen princip fungování vnitřní logiky aplikace.

Prvotní spuštění aplikace

První fáze reprezentuje prvotní spuštění aplikace, což je situace, kdy je aplikace nově nainstalována anebo ji byla uživatelem vymazána uživatelská data skrze nastavení systému. Při prvotním spuštění aplikace dochází k zobrazení úvodní aktivity (více viz 5.1.4), která slouží k získání informací o pohlaví a roku narození uživatele, potvrzení souhlasu o sběru anonymních dat týkajících se statistik využití zařízení a dále je uživatel přesměrován do nastavení systému za účelem povolení potřebného oprávnění (přístup k historií využívání zařízení). Získané informace jsou po jejich zadání ihned uloženy pomocí API `SharedPreferences` do privátního souboru typu klíč-hodnota. Na konci úvodní aktivity, tzn. před přesměrováním uživatele do hlavní aktivity, je provedeno několik operací (bez vědomí či zásahu uživatele):

- vygenerováni jedinečného identifikačního čísla aplikace, tzv. UUID (Universally Unique IDentifier), které bude sloužit k registraci aplikace na vzdáleném serveru – `uniqueID`;
- získání čísla aktuálně probíhajícího týdne, které je součástí fungování logiky na detekci nového týdne – `weekNumber`;
- získání čísla týdne prvotního spuštění aplikace, které slouží k vyhodnocování návratnosti statistik na straně vzdáleného serveru – `firstRunWeek`;
- uložení uvedených proměnných do privátního souboru typu klíč-hodnota;
- registrace potřebných alarmů, resp. příjemců vysílání, jež zajišťují správné fungování mechanismu na sběr statistik využívání zařízení a jejich zasílání na vzdálený server;
- naplánování úlohy na registraci aplikace na vzdáleném serveru;
- uložení informace o úspěšném prvotním spuštění aplikace (vytvoření klíče `firstRun` s hodnotou `false` v privátním úložišti typu klíč-hodnota).

Po provedení těchto operací dochází k přesměrování z úvodní aktivity do hlavní aktivity aplikace. Nastavení jednotlivých alarmů spolu s účelem jejich použití bude blíže popsáno v nasledujících částech věnujících se druhé a třetí fázi fungování vnitřní logiky aplikace.

Registrace aplikace na vzdáleném serveru slouží k identifikaci konkrétní instance vyvíjené aplikace, od které bude server následně přijímat zasílané sumarizované statistiky. Komunikace mezi aplikací a serverem je navržená tak, že bez úspěšné registrace není možné na vzdálený server zasílat data, resp. server přijímá data pouze od registrovaných aplikací. Samotná registrace probíhá na pozadí systému a děje se bez vědomí uživatele. Podrobnější informace o účelu této registrace se nacházejí v části 5.3 věnující se komunikaci mezi aplikací a vzdáleným serverem. Následující řádky jsou věnovány popisu registrace z pohledu operací, jež jsou spojeny s její aktivací.

Úloha, resp. služba na registraci aplikace na vzdáleném serveru je naplánována pomocí API `JobScheduler`. Reprezentována je třídou `AppRegistrationJobService`, která rozšiřuje třídu `JobService`, což je speciální služba daného API. Jedinou nastavenou podmínkou pro naplánovanou úlohu je požadavek na dostupné síťové připojení. Zkrácená ukázka naplánování dané úlohy je ve výpisu kódu 5.1.

Výpis 5.1: Naplánování úlohy na registraci aplikace na vzdáleném serveru

```
// Určení komponenty, jež specifikuje operace úlohy
ComponentName component = new ComponentName(this,
    AppRegistrationJobService.class);

// Nastavení požadavků úlohy
JobInfo jobInfo = new JobInfo.Builder(MainActivity.
    APP_REGISTRATION_JOB_SERVICE_ID, component).
    setRequiredNetworkType(JobInfo.NETWORK_TYPE_ANY).build();

// Naplánování úlohy
JobScheduler jobScheduler = (JobScheduler) this.getSystemService(
    this.JOB_SCHEDULER_SERVICE);
jobScheduler.schedule(jobInfo);
```

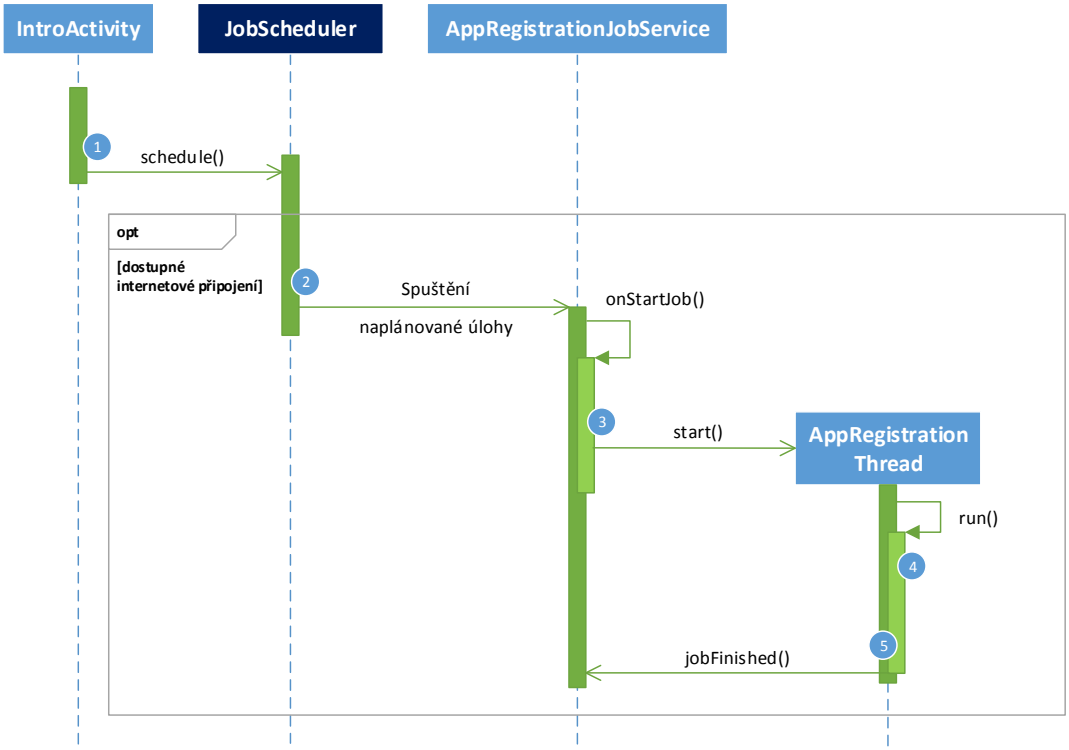
Pokud je bezprostředně po naplánování úlohy k dispozici síťové připojení, může dojít k jejímu spuštění. Za jak dlouho dobu ke spuštění skutečně dojde je závislé na vnitřním mechanismu API, které se snaží vyhodnotit optimální okamžik ke spuštění s ohledem na spotřebu baterie a efektivní využití výpočetních prostředků. Pokud síťové připojení k dispozici není, úloha zůstává v seznamu naplánovaných úloh API `JobScheduler`, dokud nenastane splnění nastavené podmínky.

Pomocí sekvenčního diagramu na obrázku 5.6 je ve zjednodušené podobě zakreslen průběh interakce hlavních komponent, jež zajišťují obsluhu mechanismu na registraci aplikace na vzdáleném serveru (z pohledu prvotního spuštění aplikace).[7] Celý průběh je možné shrnout do několika kroků:

1. Nastavení úlohy pro registraci aplikace na vzdáleném serveru pomocí objektu třídy `JobInfo.Builder`.[8] Po nastavení úlohy následuje její naplánování pomocí objektu třídy `JobScheduler` metodou `schedule()`.
2. Je-li detekováno dostupné internetové připojení, je pomocí API `JobScheduler` spuštěna naplánována úloha.
3. Jakmile je komponenta `AppRegistrationJobService` aktivována, začne se vykonávat její metoda `onStartJob()`. Vzhledem ke skutečnosti, že kód metody `onStartJob()` je vykonáván v hlavním vlákně aplikace, je v rámci této metody

[7] Světle modrou barvou jsou označeny vytvořené komponenty, tmavě modrou barvou je označena komponenta, jež je součástí samotného Android frameworku.

[8] Pro lepší přehlednost diagramu není tato operace na obrázku 5.6 zakreslena.



Obr. 5.6: Zjednodušený sekvenční diagram aktivace úlohy na registraci aplikace na vzdáleném serveru

vytvořeno vedlejší pracovní vlákno, ve kterém budou provedeny operace potřebné k registraci aplikace na serveru. Vedlejší pracovní vlákno reprezentuje privátní třída `AppRegistrationThread`, jež rozšiřuje třídu `Thread`. Pomocí metody `start()` objektu dané třídy je vlákno aktivováno. Aby bylo zajištěno aktivní CPU (Central Processing Unit) mobilního zařízení během doby běhu vedlejšího pracovního vlákna, je zapotřebí nastavit návratovou hodnotu metody `onStartJob()` na `true`. Tímto je API `JobScheduler` informováno, že úloha nebyla ještě dokončena a poběží v rámci jiného vlákna. V opačném případě by byla služba označena za vykonanou.

4. Po vytvoření a aktivaci pracovního vlákna dojde ke spuštění metody `run()`. Zde jsou pomocí objektu třídy `SharedPreferences` načtena data z privátního úložiště typu klíč-hodnota. Jedna se o: věk[9] a pohlaví uživatele, jedinečné identifikační číslo aplikace UUID a číslo týdne, ve kterém byla aplikace poprvé spuštěna. Dále je pomocí metody `createJSON(String ID, int age, String sex, int firstWeek)` vytvořen JSON řetězec, který je následně zaslán na

[9]Věk je vypočtený na základě roku narození.

vzdálený server za pomocí metody `sendDataToServer(String urlString, String jsonString)`. Na základě úspěšnosti či neúspěšnosti přenosu dat na server vrací metoda hodnotu `true`, či `false`. Obě metody jsou volány nad objektem pomocné třídy `NetworkHelper`. Po úspěšné registraci aplikace na vzdáleném serveru je do privátního úložiště typu klíč-hodnota uložena proměnná `appRegistered` s hodnotou nastavenou na `true`.

5. Před samotným ukončením vlákna je v rámci metody `run()` zavolána metoda `jobFinished(JobParametrs params, boolean needsReschedule)`, která informuje vnitřní komponentu `JobManager`[10] o jejím ukončení skrze službu zodpovědnou za spuštění vlákna. První parametr metody identifikuje naplánovanou úlohu a pomocí druhého parametru lze určit, zda má být úloha znovu naplánována. V případě, kdy nedošlo k úspěšnému přenosu registračních dat na vzdálený server (návratová hodnota metody `sendDataToServer()` je `false`), je parametr metody `needsReschedule` nastaven na `true`.

Průběžné získávání statistik

Druhá fáze reprezentuje mechanismus na průběžné získávání statistik využití, tj. informací o aktivní době běhu jednotlivých aplikací nacházejících se v mobilním zařízení. Pro účely získávání statistik využití slouží třída s názvem `DailyStatsSaver`, jež rozšiřuje třídu `IntentService`. Jedná se o službu, jež se při zavolání spouští mimo hlavní vlákno vyvíjené aplikace a po vykonání definovaného kódu v rámci metody `onHandleIntent()` je automaticky zastavena, resp. ukončena. Hlavním úkolem třídy `DailyStatsSaver` je získat informace o aktivní době běhu aplikací z aktuálně probíhajícího dne pomocí API `android.app.usage`, konkrétně pomocí metody `queryUsageStats()` třídy `UsageStatsManager`, a uložit tato data do privátní databáze SQLite pro možnost dalšího zpracování. Postup provádění jednotlivých operací v rámci metody `onHandleIntent()` lze shrnout do několika základních kroků:

- Vytvoření spojení na privátní databázi `UsageStatistics.db` a získání přístupu k objektu třídy `UsageStatsManager`.
- Pomocí metody `queryUsageStats(int intervalType, long beginTime, long endTime)` je získána kolekce typu `List`, jejíž položkami jsou objekty třídy `UsageStats` s informacemi o využívání jednotlivých aplikací. Parametry metody jsou nastaveny tak, aby byly získány informace z aktuálně probíhajícího dne, tzn.:
  - `intervalType` je nastaven na denní interval (`INTERVAL_DAILY`),
  - `beginTime` označuje počátek aktuálního dne v jednotkách milisekund (nastaveno pomocí objektu třídy `Calendar`),

[10] Jedná se o součást API `JobScheduler`.

  - `endTime` je nastaven na aktuální čas v době volání této funkce.
- Jednotlivé položky získané kolekce jsou ukládány do tabulky `DailyUsageStats`. Před uložením jednotlivých položek je vždy metodou `checkDailyStatistic()` zkontrolováno, zda se již záznam pro danou aplikaci a aktuálně probíhající den v databázi vyskytuje, či nikoliv. V případě, že záznam v tabulce neexistuje, je do ní přidán pomocí metody `addDailyStatistic()`. V opačném případě je záznam aktualizován pomocí metody `updateDailyStatistic()`, kdy dochází k aktualizaci informace o aktivní době běhu dané aplikace.
- V posledním kroku je detekováno, která komponenta provedla spuštění služby `DailyStatsSaver`. V případě, že se jednalo o `MidnightAlarmReceiver`, je danému příjemci vysíláni zaslána informace o dokončení potřebných operací (více viz níže).

Služba definovaná třídou `DailyStatsSaver` zajišťuje získání statistik využití a jejich uložení do databáze. Aby mechanismus jako celek ovšem fungoval správně, je potřeba zařídit inteligentní a pravidelné spouštění kódu této služby. Pro tyto účely jsou ve vyvíjené aplikaci vytvořeny a nastaveny dva alarmy pomocí třídy `AlarmManager`, které obsahují žádost o spuštění odpovídajících příjemců vysílání. Jedná se o třídy `MidnightAlarmReceiver` a `DaytimeAlarmReceiver`. Jediným úkolem obou zmiňovaných příjemců vysílání je spouštění služby `DailyStatsSaver` pomocí metody `startService()`.

První alarm, reprezentovaný příjemcem vysílání `MidnightAlarmReceiver`, je navržen s cílem získat finální a co nejpřesnější statistiky využití zařízení pro aktuálně probíhající den. Z tohoto důvodu je alarm spuštěn těsně před koncem dne, kdy získané statistiky obsahují celodenní aktivitu uživatele. Navržený alarm má následující konfiguraci:

- metoda pro aktivaci alarmu: `setWindow()`,
- čas spuštění alarmu: 23:50 až 23:59,
- typ alarmu: `RTC_WAKEUP`.

Zkrácená ukázka nastavení alarmu v případě prvotního spuštění vyvíjené aplikace je vidět ve výpisu kódu 5.2.

Důvodem použití metody `setWindow()` je poskytnutí určité flexibility pro určení doby spuštění alarmu operačním systémem Android, který má možnost seskupit více alarmů registrovaných v zařízení do jednoho bloku a zajistit tak šetrnější spotřebu baterie mobilního zařízení. Důvodem volby typu alarmu `RTC_WAKEUP` je zajištění spuštění služby `DailyStatsSaver` i v případě, že je dané mobilní zařízení v režimu spánku. S tímto se také váže problém ohledně doby, kdy je CPU mobilního zařízení aktivní. U standartního příjemce vysílání definovaného třídou `BroadcastReceiver` je CPU mobilního zařízení aktivní pouze po dobu vykonávání metody `onReceive()`. Po jejím ukončení zařízení přechází automaticky zpět do režimu spánku. Problém

Výpis 5.2: Registrace alarmu pro `MidnightAlarmReceiver`

```
// Určení komponenty, jež bude spuštěna po aktivaci alarmu
Intent midnightIntent = new Intent(this, MidnightAlarmReceiver.
    class);
PendingIntent midnightPendingIntent = PendingIntent.getBroadcast(
    this, 100, midnightIntent, 0);

// Nastavení začátku časového okna - 23:50
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.set(Calendar.HOUR_OF_DAY, 23);
calendar.set(Calendar.MINUTE, 50);
calendar.set(Calendar.SECOND, 00);

// Délka časového okna - 9 minut
int windowSize = 1000 * 60 * 9;

// Registrace alarmu
alarmMngr.setWindow(AlarmManager.RTC_WAKEUP, calendar.
    getTimeInMillis(), windowSize, midnightPendingIntent);
```

nastává v případě, kdy je v rámci metody `onReceive()` spuštěná jiná komponenta (v případě kontextu vyvíjené aplikace se jedná o službu `DailyStatsSaver`), jejíž délka doby běhu je delší, než doba běhu metody `onReceive()`. V takovém případě je přechod do režimu spánku iniciován dříve, než jsou úspěšně dokončeny veškeré probíhající operace definované v rámci spuštěné komponenty, resp. služby.

Za účelem ošetření uvedeného problému je pro realizaci uvedeného příjemce vysílání využita třída `WakefulBroadcastReceiver`. Tato třída rozšiřuje abstraktní třídu `BroadcastReceiver` a poskytuje potřebné prostředky pro zajištění aktivního CPU mobilního zařízení během doby provádění definovaných operací v rámci spuštěné služby. Ke spuštění služby s garantováním aktivního CPU během délky doby jejího běhu slouží metoda `startWakefulService()`. Jakmile spuštěná služba dokončí potřebné úkony, zavolá metodu `completeWakefulIntent(Intent intent)` (jedná se o statickou metodu příjemce vysílání, jenž byl zodpovědný za spuštění služby), kde parametrem je objekt typu `Intent`. Jedná se o totožný objekt, který služba přijala při svém spuštění v rámci metody `onHandleIntent (Intent intent)`. Tímto je daný příjemce vysílání informován o úspěšném dokončení prováděných operací služby a zařízení je umožněno opět přejít do režimu spánku. Pro využití třídy `WakefulBroadcastReceiver` je vyžadována registrace oprávnění typu `WAKE_LOCK`. Jedná se o typ oprávnění nacházející se ve skupině tzv. běžných oprávnění,[11] jež jsou

[11] Daná klasifikace oprávnění je uváděna pro systémy od úrovně API 23 a vyšší.

aplikaci operačním systémem přiděleny automaticky (na základě registrace v souboru `AndroidManifest.xml`).

Druhý alarm, reprezentovaný příjemcem vysílání `DaytimeAlarmReceiver`, je navržen za účelem doplnění funkce alarmu prvního. Jedná se především o pokrytí scénářů, kdy není možné zaručit spuštění předchozího alarmu. Při reálném nasazení vyvíjené aplikace do mobilního zařízení uživatele může nastat případ, kdy uživatel mobilní zařízení pravidelně vypíná v určený čas (např. během noci). V takovémto případě ke spuštění předchozího alarmu nedojde a z proběhlého dne nejsou získány žádné statistiky využití. Z tohoto důvodů je druhý alarm navržen tak, aby bylo možné zajistit pravidelné a optimální spouštění služby `DailyStatsSaver` již v průběhu dne. Konfigurace alarmu je následující:

- metoda pro aktivaci alarmu: `setInexactRepeating()`,
- čas spuštění alarmu: 5 minut po jeho nastavení,
- perioda opakování alarmu: cca 10 minut,
- typ alarmu: `ELAPSED_REALTIME`.

Zkrácená ukázka nastavení alarmu v rámci prvotního spuštění vyvíjené aplikace je ve výpisu kódu 5.3.

Výpis 5.3: Registrace alarmu pro `DaytimeAlarmReceiver`

```
// Určení komponenty, jež bude spuštěna po aktivaci alarmu
Intent daytimeIntent = new Intent(this,DaytimeAlarmReceiver.class);
PendingIntent daytimePendingIntent = PendingIntent.getBroadcast(
    this, 200, daytimeIntent, 0);

// Nastavení počátečního spuštění alarmu a periody opakování
long triggerTime = 1000 * 60 * 5; // 5 minut
long repeatInterval = 1000 * 60 * 10; // 10 minut

// Registrace alarmu
alarmMngr.setInexactRepeating(AlarmManager.ELAPSED_REALTIME,
    triggerTime, repeatInterval, daytimePendingIntent);
```

Zvolený typ alarmu `ELAPSED_REALTIME` zajišťuje spouštění alarmu pouze v případě, kdy je mobilní zařízení uživatelem aktivně využíváno. V případě, kdy je zařízení v režimu spánku a tudíž nejsou generovány žádné nové statistiky, není alarm aktivní a nedochází tak ke zbytečnému spouštění celého mechanismu zajišťující jejich sběr. Díky použití metody `setInexactRepeating()` je operačnímu systému umožněno optimalizovat okamžik spuštění alarmu v rámci definované periody za účelem nižší spotřeby baterie mobilního zařízení.

Při uvážení výše uvedeného scénáře, kdy uživatel mobilní zařízení pravidelně vypíná v určený čas (např. během noci), je díky druhému alarmu[12] zajištěn sběr dat až do doby vypnutí mobilního zařízení. Pokud je zařízení vypnuto ihned po procesu uložení dat do databáze, nedochází k žádné ztrátě informací o aktivitě uživatele. Pokud je zařízení vypnuto těsně před začátkem dalšího spuštění alarmu v rámci definované periody, pak je maximální ztráta dat v nejhorším možném případě menší nebo rovná dvojnásobku definované periody opakování alarmu, tj. $2 \times 10 = 20$ minut. Důvodem dvojnásobku periody je skutečnost, že alarm může být spuštěn na začátku první periody a následně na konci druhé periody. Pokud zařízení není vypnuto, ale pouze uvedeno do režimu spánku, pak jsou díky prvnímu alarmu[13], získány přesné statistiky z daného dne. Výjimkou je pouze scénář, kdy uživatel aktivně využívá zařízení v době spuštění tohoto alarmu, tj. v čase od 23:50 do 23:59. Nejhorší případ může nastat tehdy, kdy k aktivaci alarmu dojde v čase 23:50, ale uživatel zařízení používá i po dobu zbývající do konce dne. V takovémto případě je maximální ztráta dat menší nebo rovná 10 minut. Vzhledem k tomu, že k výše uvedeným scénářům dochází velice vzácně (jak z pohledu limitních aktivací alarmu, tak z pohledu chování uživatele), jedná se o tolerovanou ztrátu informací o aktivitě uživatele, jež je kompenzována efektivnějším využitím prostředků kapacity baterie mobilního zařízení.

Celý proces od nastavení příslušného alarmu, přes spuštění příjemce vysílání až po vykonání kódu definované služby, je ve zjednodušené podobě zakreslen pomocí sekvenčního diagramu[14] na obrázku 5.7 a lze jej shrnout do několika kroků (uvažován případ s příjemcem vysílání `MidnightAlarmReceiver`):

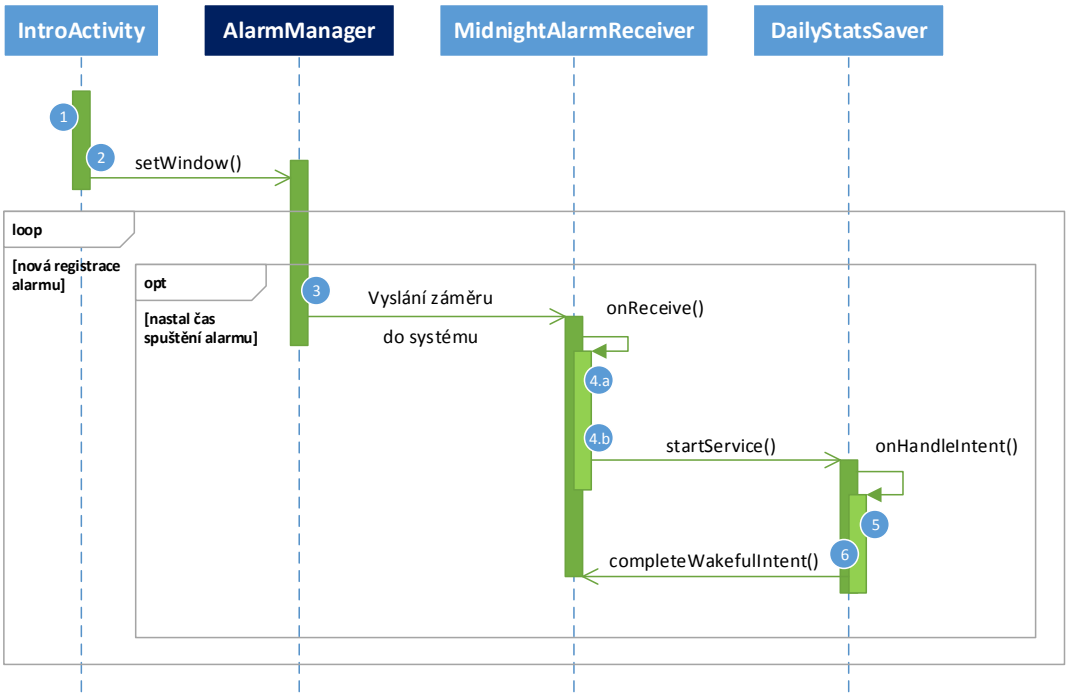
1. Vytvoření záměru (objekt typu `PendingIntent`), jenž obsahuje příslušného příjemce vysílání `MidnightAlarmReceiver`[15].
2. Nastavení alarmu a jeho registrace pomocí metody `setWindow()`.
3. Jakmile nastane definovaný čas spuštění alarmu, je do systému vyslán registrovaný záměr, jenž spustí odpovídajícího příjemce vysílání.
4. Příjemce vysílání v rámci metody `onReceive()` následně:
   (a) provede opětovné nastavení alarmu (ke spuštění pro následující den) s obdobným nastavením jako v bodě 1 a 2,
   (b) spustí službu `DailyStatsSaver` pomocí metody `startService()`.
5. Spuštěná služba provede stanovené operace v rámci metody `onHandleIntent()` a automaticky se ukončí.

[12] Reprezentován příjemcem vysílání `DaytimeAlarmReceiver`.

[13] Reprezentován příjemcem vysílání `MidnightAlarmReceiver`.

[14] Světle modrou barvou jsou označeny vytvořené komponenty, tmavě modrou barvou je označena komponenta, jež je součástí samotného Android frameworku.

[15] Pro lepší přehlednost diagramu není tato operace na obrázku 5.7 zakreslena.

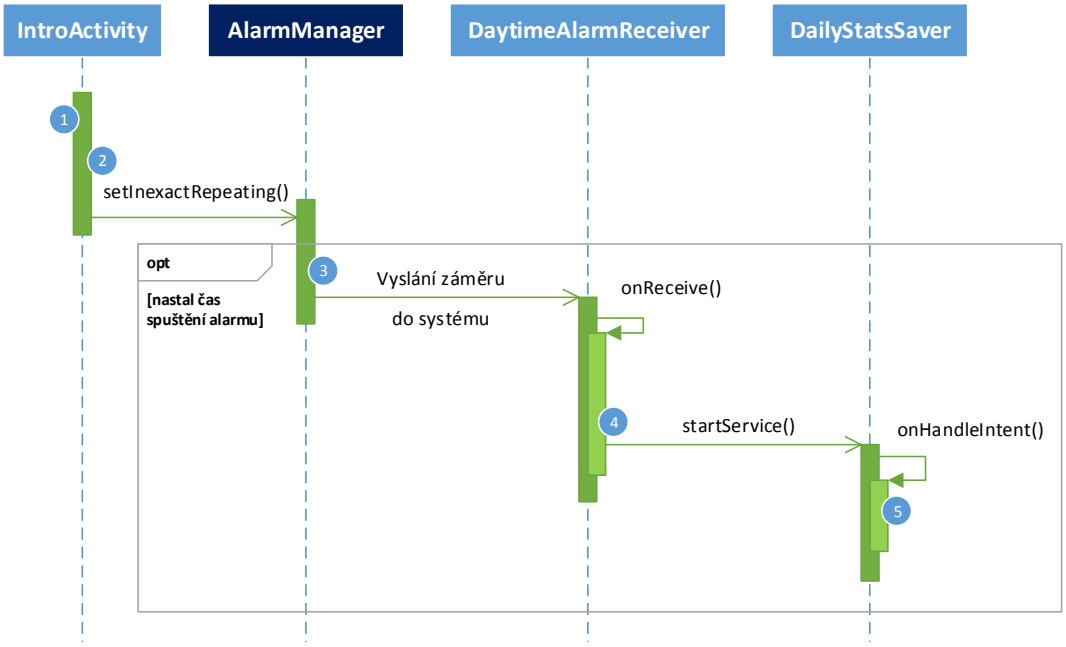


Obr. 5.7: Zjednodušený sekvenční diagram aktivace služby `DailyStatsSaver` za pomocí „půlnočního“ alarmu

6. Před samotným ukončením je příjemce vysílání informován o dokončení všech potřebných operací služby pomocí metody `completeWakefullIntent()` a CPU zařízení tak může přejít zpět do režimu spánku.

Bodem číslo 4.a) je zajištěno pravidelné každodenní opakování vykonávání alarmu aktivovaného metodou `setWindow()` a s ním související spouštění dané služby (naznačeno zakreslenou smyčkou, která se opakuje pro nově nastavený alarm). Série kroků 3 až 6 se tak s časovým rozestupem jednoho dne pravidelně opakuje.

Sekvenční diagram pro případ s příjemcem vysíláni `DaytimeAlarmReceiver` je na obrázku 5.8. Jednotlivé kroky jsou obdobné jako u předchozího případu s rozdílem použité registrační metody alarmu, která je zde `setInexactRepeating()`. Je možné si také povšimnout absence bodu č. 4.a) a zakreslené smyčky, jelikož alarm je nastaven jako automaticky se opakující a tudíž jej není potřeba opětovně nastavovat (vyjímkou je pouze restart či vypnutí a zapnutí mobilního zařízení – viz dále). Posledním rozdílem je absence bodu č. 6 (jedná se o standartního příjemce vysílání). Série kroků 3 až 5 se v případě, že je mobilní zařízení uživatelem aktivně využíváno, opakuje v intervalu cca 10 minut.



Obr. 5.8: Zjednodušený sekvenční diagram aktivace služby `DailyStatsSaver` za pomocí „denního“ alarmu

Jak již bylo dříve uvedeno, veškeré nastavené alarmy v systému jsou automaticky zrušeny při restartování operačního systému mobilního zařízení (tzn. i při jeho vypnutí a opětovném zapnutí). Tato skutečnost má přímý dopad na fungování mechanismu na průběžné získávání statistik, jelikož po restartu zařízení se v systému již nenachází žádný z nastavených alarmů, který by umožnil spuštění kódu třídy `DailyStatsSaver`. Pro zachování funkčnosti daného mechanismu je potřeba alarmy opětovně nastavit. K tomuto účelu je ve vyvíjené aplikaci vytvořen a zaregistrován dodatečný příjemce vysílání `DeviceBootReceiver`, který je schopen detekovat restart zařízení či jeho opětovné zapnutí. Aby byla detekce možná, je potřeba do souboru manifestu aplikace přidat oprávnění `RECEIVE_BOOT_COMPLETED`. Dané oprávnění umožní vyvíjené aplikaci získat přístup k informaci `ACTION_BOOT_COMPLETED`, která je do operačního systému vyslána ve chvíli, jakmile je systém opět připraven k používání. Při registraci daného příjemce vysílání v souboru `AndroidManifest.xml` je dále nutné specifikovat tzv. filtr záměru, jenž filtruje výše uvedenou informaci o úspěšném naběhnutí systému. Konfigurace manifestu aplikace pro příjemce vysílání `DeviceBootreceiver` je na obrázku 5.4.

Výpis 5.4: Registrace příjemce vysílání `DeviceBootreceiver` v manifestu aplikace

```
<receiver
    android:name=".alarm_receivers.DeviceBootReceiver"
    android:enabled="true">
    <intent-filter>
        <action
            android:name="android.intent.action.BOOT_COMPLETED">
        </action>
    </intent-filter>
</receiver>
```

V rámci samotného kódu příjemce vysílání `DeviceBootReceiver` jsou po detekci příslušné akce opětovně nastaveny všechny potřebné alarmy zajišťující správnou funkčnost mechanismu na průběžné získávání statistik využití zařízení.

Detekce nového týdne a zaslání statistik na server

Poslední fáze fungování vyvíjené aplikace reprezentuje schopnost vnitřního mechanismu získat sumarizovaná data o využívání mobilního zařízení za uplynulý týden, která jsou následně uložena do privátní databáze a zaslána na vzdálený server pro jejich další zpracování a vyhodnocení.

S návrhem a realizací uvedené funkcionality se váže celá řada úskalí, otázek a scénářů, které mohou nastat. Zde je několik z nich:

- Jak určit, zda nastal nový týden a je tak možné začít se sumarizaci denních statistik do týdenního souhrnu?
- V jaký čas či den budou data zaslána na vzdálený server?
- Co se bude dít v případě, kdy v určeném čase či dni nebude k dispozici internetové připojení?
- Co se bude dít v případě, kdy v určeném čase či dni bude mobilní zařízení vypnuté?
- Co se bude dít v případě, kdy v určeném čase či dni bude vzdálený server nedostupný?
- V případě, že mobilní zařízení bude vypnuté déle než jeden týden, jak se naloží se statistikami, jež byly získávány v týdnu před vypnutím mobilního zařízení? Bude možné data zpětně sumarizovat a dodatečně zaslat na vzdálený server? Nebo budou data ztracena, jelikož se bude vyhodnocovat již jiný, aktuálně uplynulý týden?
- Jak vyřešit rozložení zátěže na vzdálený server ze strany mobilních zařízení?

Na základě těchto a celé řady dalších obdobných otázek či scénářů byly specifikovány a navrženy základní požadavky na funkčnost algoritmu, resp. mechanismu, jenž má zajistit potřebnou funkcionalitu. Požadavky jsou následující:

- Schopnost efektivní detekce začátku nového týdne.
- Zajistit detekci, zda již byla aplikace registrována na vzdáleném serveru (bez úspěšné registrace není možné zasílat data na vzdálený server).
- Vyhnout se definování konkrétního času a dne pro spuštění operací na sumarizaci týdenních statistik, jejich uložení a zaslání na vzdálený server. Poskytnout možnost zaslat data na vzdálený server kdykoliv během týdne, pokud jsou již k dispozici kompletní statistiky za předcházející týden.
- Schopnost určit, zda data z jednotlivých dnů již byla sumarizována a zaslána na vzdálený server.
- Možnost načíst data z předcházejících týdnů a zpětně je zaslat na vzdálený server i v případě, kdy mobilní zařízení bylo dlouhodobě vypnuto nebo bylo dlouhodobě bez dostupného internetového připojení.

Veškeré zmíněné požadavky splňuje navržený a realizovaný mechanismus (kooperace několika komponent), jehož popisu jsou věnovány následující řádky.

Hlavní koordinační komponentou realizovaného mechanismu je příjemce vysílání `MiddayAlarmReceiver`. Jeho dva hlavní úkoly jsou:

1. Kontrola, zda již byla aplikace registrována na vzdáleném serveru, včetně případného opětovného naplánovaní úlohy, pokud aplikace stále není zaregistrována.
2. Detekce začátku nového týdne a s tím související nastavení a naplánování úlohy na sumarizaci statistik z minulého týdne do týdenního souhrnu, jejich uložení do privátní databáze a zaslání na vzdálený server.

Spouštění příjemce vysílání je zajištěno alarmem pomocí třídy `AlarmManager`. Konfigurace alarmu je následující:

- metoda pro aktivaci alarmu: `setWindow()`,
- čas spuštění alarmu: 12:00 až 13:00,
- typ alarmu: `RTC_WAKEUP`.

Zkrácená ukázka nastavení alarmu v rámci prvotního spuštění vyvíjené aplikace je vidět ve výpisu kódu 5.5.

Důvody použití uvedené konfigurace jsou obdobné jako u výše uvedeného příjemce vysílání `MidnightAlarmReceiver`. Metoda `setWindow()` s nastavenou délkou okna 60 minut poskytuje operačnímu systému Android dostatek prostoru pro určení optimální doby spuštění alarmu za účelem šetrnější spotřeby baterie mobilního zařízení. Prakticky je možné délku okna navýšit i na několik hodin, jelikož v tomto případě není doba spuštění alarmu podstatná, ale jde především o jeho každodenní vykonáni.

Výpis 5.5: Registrace alarmu pro `MiddayAlarmReceiver`

```
// Určení komponenty, jež bude spuštěna po aktivaci alarmu
Intent middayIntent = new Intent(this, MiddayAlarmReceiver.class);
PendingIntent middayPendingIntent = PendingIntent.getBroadcast(this
    , 0, middayIntent, 0);

// Nastavení začátku časového okna - 12:00
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.set(Calendar.HOUR_OF_DAY, 12);
calendar.set(Calendar.MINUTE, 00);
calendar.set(Calendar.SECOND, 00);
calendar.add(Calendar.DATE, 1);

// Délka časového okna - 60 minut
int middayWindowSize = 1000 * 60 * 60;

// Registrace alarmu
alarmMngr.setWindow(AlarmManager.RTC_WAKEUP, calendar.
    getTimeInMillis(), middayWindowSize, middayPendingIntent);
```

Obdobně je potřeba zajistit spuštění alarmu i v případě, kdy je zařízení v režimu spánku (`RTC_WAKEUP`). Na rozdíl od příjemce vysílání `MidnightAlarmReceiver` ovšem není potřeba využívat prostředků třídy `WakefullBroadcatReceiver`, jelikož v rámci metody `onReceive()` není spouštěná žádná služba. Dochází pouze k registraci potřebných komponent a naplánování definovaných úloh pomocí API `JobScheduler`[16]. Zajištění aktivního CPU po dobu trvání metody `onReceive()` je tedy postačující.

Opakování alarmu, resp. opakované spouštění příjemce vysílání je zajištěno stejným způsobem jako v případě třídy `MidnightAlarmReceiver`, tzn. v rámci metody `onReceive()` je alarm znovu nastaven ke spuštění pro následující den.

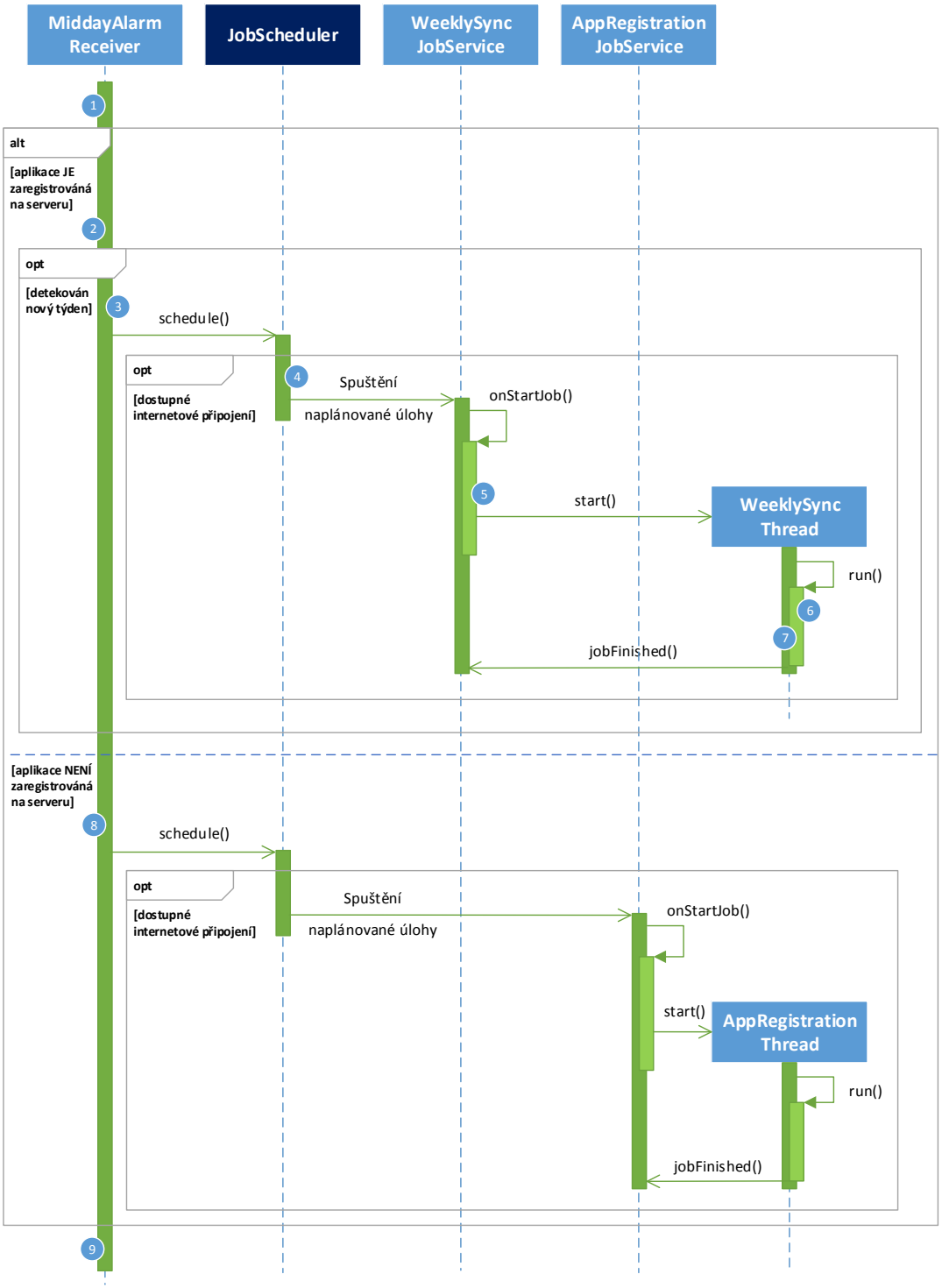
Z výpisu kódu 5.5 je možné si také povšimnout, že doba prvotního spuštění alarmu je nastavena až na následující den. Díky tomu je zajištěno, že alarm nebude zaregistrován na dobu, která již proběhla (možný scénář, kdy k prvotnímu spuštění aplikace dojde až po 13. hodině). V případě registrace tohoto alarmu v rámci příjemce vysílání `DeviceBootReceiver` (tzn. po restartu mobilního zařízení) je alarm nastaven tak, aby byl spuštěn již v rámci aktuálně probíhajícího dne (jsou tak pokryty další dílčí scénáře, jejichž bližší popis je již nad rámec tohoto textu).

[16]Třída `JobScheduler` má již své vlastní prostředky pro zajištění aktivního CPU za účelem vykonání naplánované úlohy.

Na základě jednotlivých kroků sekvenčního diagramu na obrázku 5.9 je blíže popsán princip fungování příjemce vysílání `MiddayAlarmReceiver`, potažmo mechanismu, jenž zajišťuje detekci nového týdne a vykonávání operací spojených se zasláním informací o využívání zařízení na vzdálený server. Sekvenční diagram je zakreslen z pohledu interakcí mezi nejdůležitějšími komponentami realizovaného mechanismu. Dílčí interakce mezi dalšími souvisejícími komponentami nejsou z důvodu lepší přehlednosti sekvenčního diagramu plně zakresleny, ale pouze označeny akčním bodem, jenž je popsán v textu práce. Výchozí stav sekvenčního diagramu je okamžik, kdy je příjemce vysílání spuštěn nastaveným alarmem pomocí třídy `AlarmManager` a začne provádění metody `onReceive()` třídy `MiddayAlarmReceiver`. Sled akcí je následovný:

1. Určení, zda byla aplikace již zaregistrována na vzdáleném serveru či nikoliv na základě proměnné `appRegistered` načtené z privátního uložiště typu klíč-hodnota pomocí API `SharedPreferences`.
2. V případě, že je aplikace již zaregistrována, je provedena detekce začátku nového týdne. Ta pracuje na principu porovnání hodnoty aktuálně probíhajícího týdne (získána pomocí třídy `Calendar`) s hodnotou proměnné `weekNumber` uloženou v privátním uložišti typu klíč-hodnota. Proměnná `weekNumber` je vytvořena při prvotním spuštění aplikace a indikuje hodnotu týdne, v němž aktuálně probíhá každodenní sběr statistik využití zařízení. Pokud se hodnota aktuálně probíhajícího týdne shoduje s hodnotou proměnné `weekNumber`, stále trvá týden, v němž jsou aktivně získávány statistiky pro jednotlivé dny. Ve chvíli, kdy se hodnoty neshodují, je detekován začátek nového týdne, jelikož hodnota proměnné `weekNumber` zůstává stále stejná, ale hodnota aktuálního týdne se logicky s příchodem nového týdne změní. V tomto stavu proměnná `weekNumber` označuje číslo týdne, u kterého je možné provést operace spojené se zasláním dat na vzdálený server.
3. Je-li detekován nový týden, dochází pomocí objektu třídy `JobInfo.Builder` k nastavení úlohy pro zaslání dat na vzdálený server a sní spojenými operacemi. Úloha je reprezentována komponentou `WeeklySyncJobService`, která rozšiřuje třídu `JobService`[17]. Nastavení úlohy obsahuje pouze jedinou podmínku, a to požadavek na dostupné internetové připojení. Úloha je následně naplánována pomocí objektu třídy `JobScheduler` metodou `schedule()`.
   Z hlediska běhu komponenty `MiddayAlarmReceiver` dochází již pouze k vykonání bodu č. 9 a k jejímu ukončení. Další kroky jsou vykonány asynchronně na základě splnění definované podmínky pro naplánovanou úlohu.

[17]Speciální služba API `JobScheduler`.



Obr. 5.9: Zjednodušený sekvenční diagram reprezentující koordinační operace spojené s příjemcem vysílání `MiddayAlarmReceiver`

4. Je-li detekováno dostupné internetové připojení, je pomocí API `JobScheduler` spuštěna naplánována úloha, resp. služba `WeeklySyncJobService`.
5. Po aktivaci komponenty se začne vykonávat metoda `onStartJob()`. Vzhledem k tomu, že kód metody `onStartJob()` je vykonáván v hlavním vlákně aplikace, je v rámci této metody vytvořeno vedlejší pracovní vlákno, ve kterém se budou provádět operace související se zasláním dat na vzdálený server. Vedlejší pracovní vlákno reprezentuje privátní třída `WeeklySyncRoutineThread`, jež rozšiřuje třídu `Thread`. Pomocí metody `start()` objektu dané třídy je vlákno aktivováno. Aby bylo zajištěno aktivní CPU mobilního zařízení během doby běhu vedlejšího pracovního vlákna, je zapotřebí nastavit návratovou hodnotu metody `onStartJob()` na `true`. Tímto je API `JobScheduler` informováno, že úloha nebyla ještě dokončena a poběží v rámci jiného vlákna. V opačném případě by byla služba označena za vykonanou.
6. Po vytvoření a aktivaci pracovního vlákna dojde ke spuštění metody `run()`. V této fázi je již možné začít s operacemi, jako je sumarizace denních statistik do týdenního souhrnu z týdne, jenž je definován hodnotou proměnné `weekNumber`, dále uložení sumarizovaných statistik do privátní databáze aplikace a jejich zaslání na vzdálený server ve formátu JSON. Jednotlivé operace v rámci aktivovaného vlákna budou blíže vysvětleny na základě vývojového diagramu níže v textu.
7. Před samotným ukončením vlákna je v rámci metody `run()` zavolána metoda `jobFinished(JobParametrs params, boolean needsReschedule)`, která informuje vnitřní komponentu `JobManager` o jejím ukončení skrze službu zodpovědnou za spuštění vlákna. První parametr metody identifikuje naplánovanou úlohu a pomocí druhého parametru lze určit, zda má být úloha znovu naplánována (např. v případě, kdy nedošlo k úspěšnému zaslání dat na vzdálený server).

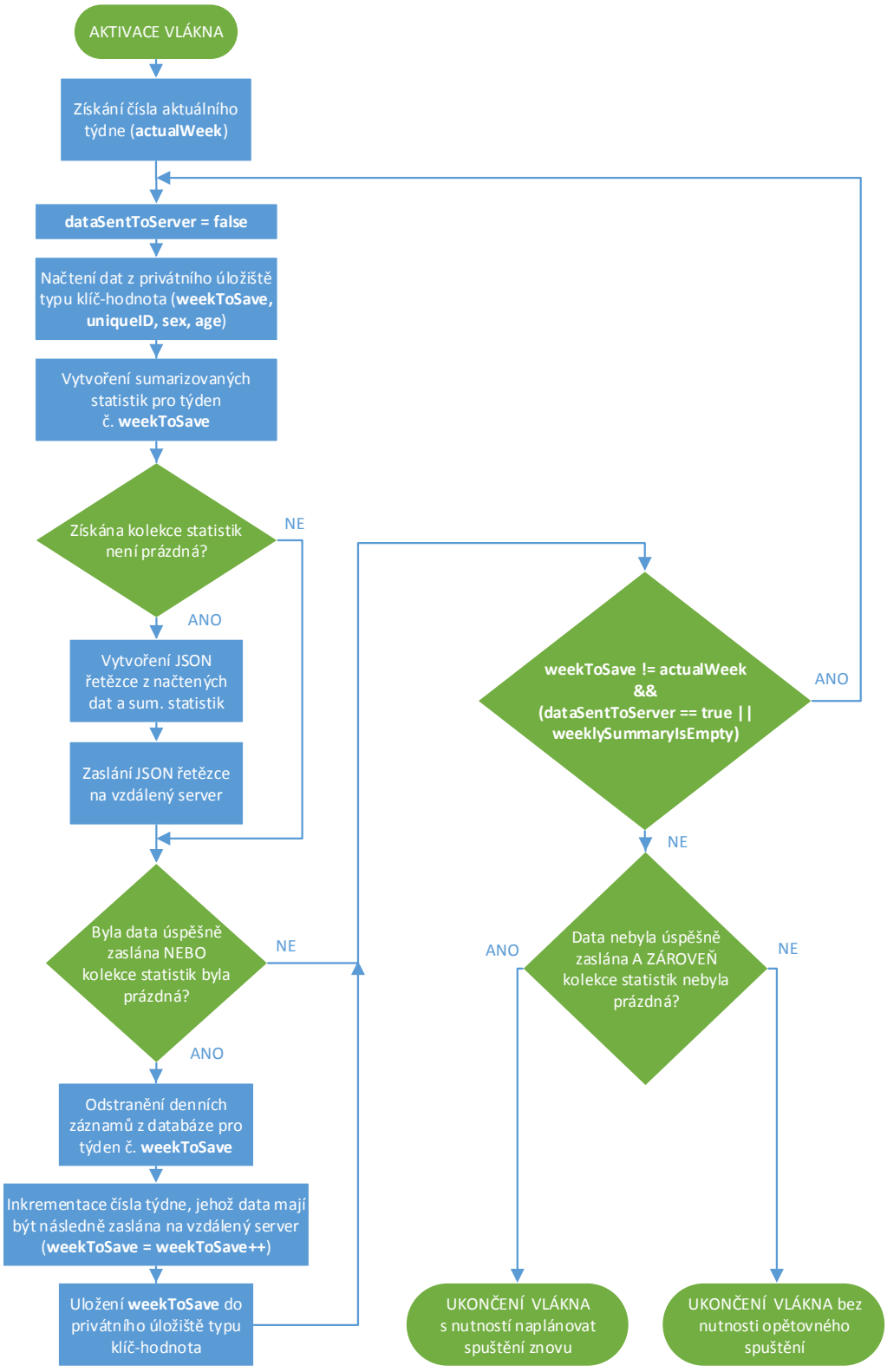
   Pokud byla v této fázi data úspěšně uložena na vzdáleném serveru, pokračuje standartní každodenní sběr statistik z jednotlivých dnů a každodenní spouštění příjemce vysílání `MiddayAlarmReceiver`, který provádí kroky č. 1 a 2. V okamžiku, kdy je detekován nový týden, dochází k provedení kroků č. 4 až 7. Uvedený proces se tak neustále opakuje.
8. V případě, kdy bylo v bodě č. 1 detekováno, že aplikace stále není zaregistrována na vzdáleném serveru, nelze na server zasílat žádná data a je nutné znovu naplánovat úlohu na její registraci. Jedná se o totožné kroky, které byly popsány na obrázku 5.6 v rámci sekce věnované prvotnímu spuštění aplikace. Rozdílné je pouze to, že naplánování úlohy neprobíhá v rámci třídy `IntroActivity`, ale v rámci příjemce vysílání `MiddayAlarmReceiver`.

9. Poslední bod reprezentuje registraci alarmu pro opětovné spuštění příjemce vysílání `MiddayAlarmReceiver` pro následující den. Jedná se o totožné nastavení jako u prvnotního spuštění aplikace (viz předcházející výpis kódu 5.5).

Významnou výhodou použití API `JobScheduler` pro účely naplánování úlohy je vyhnutí se definování konkrétního času a dne pro její spuštění. V součinnosti s algoritmem realizovaným v rámci vlákna `WeeklySyncRoutineThread` (viz níže) tak může zaslání týdenních statistik na server proběhnout kdykoliv v průběhu nového týdne či dokonce i v týdnech následujících (díky schopnosti algoritmu zaslat data i zpětně). Vedlejší výhodou tohoto přístupu je náhodné rozložení zátěže na vzdáleném serveru ze strany mobilních zařízení.

Bloky operací, jež jsou prováděny v rámci vlákna `WeeklySyncRoutineThread` (bod č. 6 sekvenčního diagramu 5.9) jsou zakresleny pomocí vývojového diagramu na obrázku 5.10. Postup vykonávání algoritmu je následující:

- Po aktivaci vlákna jsou deklarovány (případně definovány) potřebné proměnné a pomocí třídy `Calendar` je získáno číslo aktuálního týdne `actualWeek`.
- Následně se vstupuje do smyčky do-while, kde je v úvodu nastavena výchozí hodnota proměnné `dataSentToServer`, jež reprezentuje (ne)úspěšnost přenosu dat na vzdálený server, na `false`.
- Z privátního uložiště typu klíč-hodnota jsou pomocí API `SharedPreferences` načtená data jako je pohlaví a věk uživatele (vypočítán na základě roku narození), jedinečné identifikační číslo aplikace UUID a proměnná `weekNumber`, jež byla vytvořena při prvotním spuštění aplikace a v kontextu tohoto vlákna reprezentuje číslo týdne, jehož denní statistiky jsou již k dispozici ke zpracování a zaslání na vzdálený server. Hodnota této proměnné je uložena do lokální proměnné `weekToSave`.
- Následně dochází k vytvoření sumarizovaných týdenních statistik pro číslo týdne, jenž je definované proměnnou `weekToSave`. Statistiky jsou vytvořeny pomocí metody `createWeeklyStats()`, jejíž návratovou hodnotou je kolekce typu `List` s objekty třídy `UsageStatistic`. Metoda obsahuje dílčí algoritmus, který na základě denních statistik uložených v tabulce `DailyUsageStats` vygeneruje seznam aplikací, jež byly uživatelem v daném týdnu využívány, a pro jednotlivé aplikace vytvoří celkový týdenní souhrn doby, po kterou byly aktivně využívány. Sumarizované týdenní statistiky jsou následně uloženy do tabulky `WeeklyUsageStats` a v rámci návratové hodnoty metody předány k dalšímu zpracování.
- Pokud získaná kolekce statistik není prázdná (tzn. mobilní zařízení bylo uživatelem alespoň nějakou dobu v daném týdnu využíváno) je z položek dané kolekce a dat získaných z privátního úložiště typu klíč-hodnota vytvořen JSON řetězec pomocí metody `createJSON()`, který je následně zaslán na vzdálený



Obr. 5.10: Vývojový diagram algoritmu vlákna `WeeklySyncRoutineThread`

server pomocí metody `sendDataToServer()`. Na základě úspěšnosti či neúspěšnosti přenosu dat na server vrací metoda hodnotu `true`, či `false`. Obě metody jsou volány nad objektem třídy `NetworkHelper`. V případě, kdy získána kolekce je prázdná (tzn. v daném týdnu bylo mobilní zařízení vypnuto), jsou uvedené kroky vynechány.

- Následně je vyhodnocena podmínka, zda byla data úspěšně zaslána na server *nebo* byla kolekce statistik prázdná, tj.:

  ```
  (dataSentToServer == true || weeklySummaryIsEmpty == true).
  ```

  Pokud podmínka platí, jsou z tabulky pro denní statistiky `DailyUsageStats` smazány záznamy ze dnů, které již byly úspěšně sumarizované do týdenního souhrnu a zaslány na vzdálený server. Nejdůležitějším krokem je následná inkrementace proměnné `weekToSave` (včetně ošetření, kdy dochází k přelomu roku a číslování týdne začíná opět od č. 1) a uložení její hodnoty do proměnné `weekNumber` privátního uložiště typu klíč-hodnota. V případě, kdy výše uvedená podmínka neplatí, jsou zmíněné kroky vynechány.
- V tomto bodě dochází k vyhodnocení podmínky cyklu do-while. Podmínka je následující:

  ```
  (weekToSave != actualWeek && (dataSentToServer == true ||
     weeklySummaryIsEmpty == true)).
  ```

  První část podmínky, tj. `weekToSave != actualWeek`, prakticky určuje, zda existují statistiky z předešlých týdnů, které ještě nebyly zaslány na vzdálený server (např. vlivem dlouhodobého vypnutí mobilního zařízení či nedostupnosti internetového připojení). Celkem mohou nastat čtyři scénáře:

  1. `(true && (true || false))` – cyklus do-while se bude opakovat. Existují další statistiky z předminulých týdnů, které ještě nebyly zpracovány a zaslány na vzdálený server. Zaslání statistik z týdne, jenž předcházel týdnu č. `weekToSave`, proběhlo úspěšně. Cyklus se bude opakovat pro zpracování statistik z týdne č. `weekToSave`.
  2. `(true && (false || true))` – cyklus do-while se bude opakovat. Existují další statistiky z předminulých týdnů, které ještě nebyly zpracovány a zaslány na vzdálený server. Získaná kolekce statistik z týdne, jenž předcházel týdnu č. `weekToSave`, byla prázdná (tzn. mobilní zařízení bylo vypnuto) a žádná data nebyla zasílána. Cyklus se bude opakovat pro zpracování statistik z týdne č. `weekToSave`.
  3. `(true && (false || false))` – cyklus do-while se nebude opakovat. Existují další statistiky z předminulých týdnů, které ještě nebyly zpracovány a zaslány na vzdálený server. Zaslání statistik z týdne č. `weekToSave`

ovšem proběhlo neúspěšně.[18] Cyklus se nebude opakovat a je potřeba naplánovat nové spuštění tohoto vlákna za účelem opětovného pokusu o zaslání dat z týdne č. `weekToSave`.

4. `(false && (/ || /))`[19] – cyklus do-while se nebude opakovat.
   Pro týden č. `weekToSave` ještě nejsou k dispozici kompletní statistiky, které by bylo možné zpracovat a zaslat na vzdálený server (tzn. stále probíhá jejich sběr). Cyklus se nebude opakovat.

- V rámci poslední podmínky je vyhodnocováno, zda je či není potřeba znovu naplánovat spuštění vlákna, resp. služby `WeeklySyncJobService`, jež dané vlákno spouští. Pokud data nebyla na server úspěšně zaslána *a zároveň* získaná kolekce se statistikami nebyla prázdná, tj.:

  ```
  (dataSentToServer == false && weeklySummaryIsEmpty == false),
  ```

  je potřeba spuštění vlákna naplánovat znovu.

[18] Jedná se o původní číslo týdne, jež bylo načteno z privátního úložiště – nebylo inkrementováno!
[19] Na stavu druhé části podmínky nezáleží.

5.1.4 Uživatelské rozhraní

Cílem této části je představit význam uživatelského rozhraní vyvíjené aplikace. Nejdříve je představen kontext, ze kterého vychází návrh uživatelského rozhraní, a dále následuje popis jednotlivých aktivit, ze kterých se uživatelské prostředí skládá.

Díky skutečnosti, že výše popsaný vnitřní mechanismus vyvíjené aplikace běží na pozadí operačního systému Android a je zcela nezávislý na interakci uživatele s vyvíjenou aplikací,[20] je možné celou vnitřní logiku nasadit, resp. integrovat do jakékoliv mobilní aplikace OS Android s úrovní API 21 a vyšší. S pracovníky Policie ČR jsou diskutovány dvě základní varianty pro nasazení realizovaného mechanismu do konkrétní aplikace. Snahou je poskytnout určitou přidanou hodnotu pro uživatele, jenž si aplikaci obsahující daný mechanismus nainstaluje a bude tak poskytovat anonymní údaje o využívání svého zařízení pro účely zkvalitnění prevence kyberšikany.

První možnou variantou je integrovat mechanismus do aplikace z oblasti volného času a zábavy, např. mobilní hry. Jednalo by se o buďto vytvoření nové aplikace tohoto typu anebo by bylo možné využít již existující aplikace, která je např. součástí jiného přidruženého projektu, a do ní daný mechanismus integrovat.

Druhou uvažovanou variantou je integrace mechanismu do aplikace, která poskytuje přidanou hodnotu v oblasti vzdělávání. Jednalo by se o aplikaci, jež obsahuje edukativní obsah různorodého typu (texty, komiksy, audio/video apod.) či poskytuje funkce užitečné v kontextu školní docházky žáků a studentů jako jsou např. úkolníček, budík, rozvrh hodin atp.

Vzhledem k tomu, že otázka nasazení daného mechanismu je stále předmětem diskuze, bylo pro účely této diplomové práce stanoveno připravit strukturu uživatelského rozhraní pro druhou diskutovanou variantu.

Cílem je poskytnout flexibilní uživatelské rozhraní, do kterého bude možné dle potřeby snadno a efektivně přidávat další obsah a rozšiřovat jej o novou funkcionalitu. Vývoj jednotlivých funkcí aplikace poskytovaných skrze uživatelské rozhraní či integrace určitého edukativního obsahu není součástí této diplomové práce. Vytvořené uživatelské rozhraní je doplněno pouze u ukázkový fragment, jehož účelem je poskytování zpětné vazby uživateli o tom, jak využívá své mobilní zařízení z hlediska času stráveného v jednotlivých aplikacích. Prakticky se jedná o demonstraci prostředků balíčku `android.app.usage`.

Vytvořené uživatelské rozhraní se skládá z celkem tří aktivit. Jedná se o třídy `SplashActivity`, `IntroActivity` a `MainActivity`. Jejich popisu jsou věnovány následující řádky.

[20] Jedinou podmínkou je, aby aplikace byla alespoň jednou spuštěna a mohlo tak dojít k aktivaci vnitřního mechanismu.

SplashActivity

Hlavním účelem této aktivity je vyplnit časovou mezeru mezi okamžikem, kdy je aplikace uživatelem spuštěna, a stavem, kdy je aplikace plně načtena a zobrazena na popředí obrazovky. V závislosti na tom, zda je či není konfigurace spuštění aplikace dostupná v cash paměti, může plné načtení aplikace obecně trvat až jednotky sekund. Z tohoto důvodu je během této doby vhodné zobrazit např. logo dané aplikace či krátký uvítací slogan.

Aktivita je v rámci souboru `AndroidManifest.xml` zaregistrována jako hlavní spouštěcí aktivita, tzn. je pro ní nastaven filtr záměru pro akci typu `MAIN` kategorie `LAUNCHER`. Současně je zde nastaveno schéma `SplashScheme`, které odkazuje na soubor `splash_screen.xml`, kde je definován obsah určený k zobrazení při spuštění dané aktivity. Registrace aktivity v rámci manifestu aplikace je zobrazena ve výpisu kódu 5.6.

Výpis 5.6: Registrace aktivity `SplashActivity` v manifestu aplikace

```
<activity
    android:name=".SplashActivity"
    android:theme="@style/SplashTheme">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
```

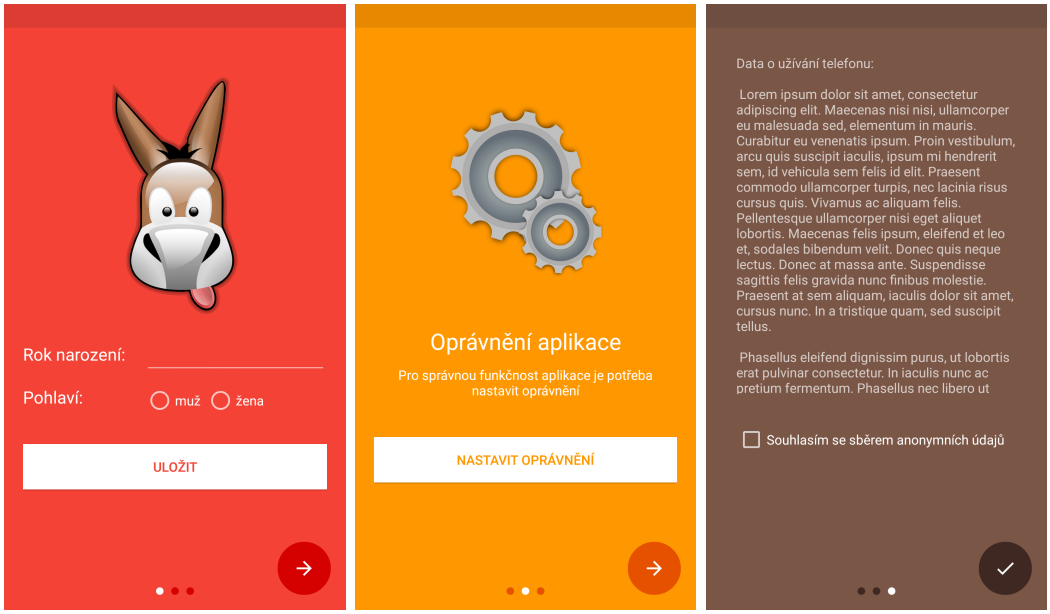
Dalším podstatným úkolem aktivity je detekce, zda se jedná o prvotní spuštění aplikace[21] či nikoliv. V případě, kdy je aplikace spuštěná poprvé, dochází k načtení a zobrazení úvodní aktivity (třída `IntroActivity`). V opačném případě dochází k přesměrování do aktivity reprezentující hlavní uživatelské rozhraní aplikace (třída `MainActivity`). Detekce funguje na principu zjištění hodnoty proměnné `firstRun` načtené z privátního úložiště typu klíč-hodnota pomocí objektu třídy `SharedPreferences`. V případě, kdy již byla aplikace v minulosti spuštěná, obsahuje proměnná `firstRun` hodnotu `false` (její uložení proběhlo v rámci posledního kroku úvodní aktivity), čímž dochází ke spuštění hlavní aktivity. Pokud je aplikace spuštěná poprvé, daná proměnná v privátním úložišti neexistuje a funkce na její načtení vrací výchozí hodnotu `true`, čímž dochází k přesměrování do úvodní aktivity.

[21] Aplikace je nově nainstalována anebo ji byla uživatelem vymazána uživatelská data skrze nastavení systému.

IntroActivity

Cílem úvodní aktivity je získání informací o pohlaví a roku narození uživatele, přesměrování uživatele do nastavení systému za účelem povolení potřebného oprávnění (přístup k historii využívání zařízení) a potvrzení souhlasu o sběru anonymních dat. Před ukončením této aktivity dochází k aktivaci vnitřního mechanismu zodpovědného za sběr, zpracování a zasílání statistik využívání zařízení na vzdálený server. Jedná se tak o jedinou část uživatelského rozhraní, jež souvisí s fungováním vnitřního mechanismu aplikace.

Pro vybudování úvodní aktivity je využitá open-source knihovna *Material-intro-screen* [43]. V rámci aktivity jsou vytvořeny tři vlastní fragmenty. Grafický vzhled jednotlivých fragmentů je na obrázku 5.11. Vektorové obrázky použité při tvorbě jednotlivých fragmentů spadají pod CCO licenci[22] a byly získány z webové stránky *pixabay.com* [44].



Obr. 5.11: Fragmenty v rámci úvodní aktivity `IntroActivity`

První fragment (třída `UserInfoIntroSlide`) slouží k získání informací ohledně věku a pohlaví uživatele. Součástí vnitřní logiky tohoto fragmentu je validace zadaného roku narození. Povolený rozsah odpovídá věku od 6 do 99 let a je dynamicky určen na základě aktuálně probíhajícího roku. Pokud jsou zadaná data nekorektní či není zvoleno pohlaví uživatele, je přechod k dalšímu fragmentu znemožněn. Obdobně je tomu v případě, kdy data nejsou uložena stisknutím tlačítka „ULOŽIT“. Definice rozložení fragmentu se nachází v souboru `intro_fragment_user_info.xml`.

[22]Licence umožňuje volné nakládání s daným dílem.

V rámci druhého fragmentu (třída `PermissionIntroSlide`) je uživatel naveden k povolení potřebného oprávnění. Před samotným načtením fragmentu je v rámci aktivity detekováno, jaká verze OS Android je v mobilním zařízení naistalována. Pokud je k dispozici verze API 23 a vyšší, dochází k načtení fragmentu, jenž je optimalizovaný pro tzv. *run-time permission* (přidělování práv aplikaci až za jejího běhu). V opačném případě je načten fragment, v němž je uživatel skrze tlačítko „NASTAVIT OPRÁVNĚNÍ“ přesměrován do sekce nastavení systému určené ke správě přístupu jednotlivých aplikací k historii využívání zařízení. Pokud je při pokusu o přechod k dalšímu fragmentu detekováno, že požadované oprávnění nebylo aplikaci přiděleno, není možné pokračovat na další fragment. Definice rozložení fragmentu se nachází v souboru `intro_fragment_permission.xml`.

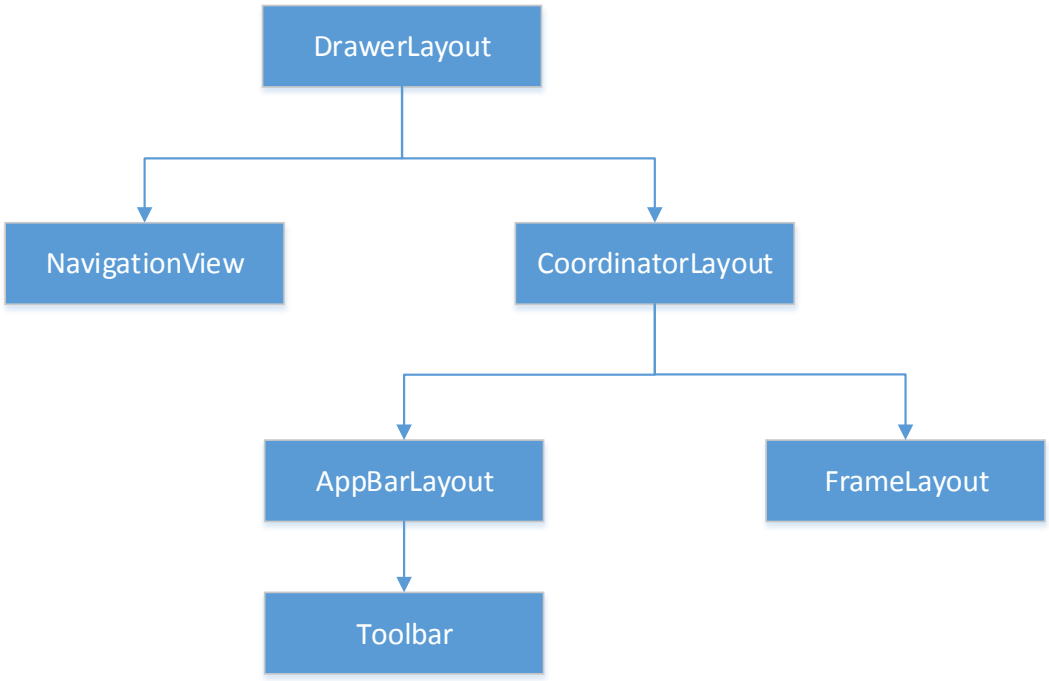
Poslední fragment úvodní aktivity (třída `AgreementIntroSlide`) slouží k potvrzení souhlasu o sběru anonymních dat. Konkrétní znění textu související se souhlasem sběru anonymních dat bude v pozdější fázi projektu dodáno pracovníky Policie ČR. V rámci diplomové práce je tato část vyplněna textem *Lorem Ipsum*. Po potvrzení souhlasu je uživatel přesměrován do aktivity reprezentující hlavní uživatelské rozhraní aplikace. Obdobně jako u předchozího fragmentu je v případě nepotvrzení souhlasu se sběrem anonymních dat přechod do další části aplikace, tj. hlavní aktivity, znemožněn. Definice rozložení fragmentu se nachází v souboru `intro_fragment_agreement.xml`.

U všech zmíněných fragmentů je uživateli při znemožnění přechodu k dalšímu kroku zobrazena odpovídající varovná zpráva upřesňující důvod zamítnutí (viz obrázek 5.12).

Obr. 5.12: Vybrané varovné zprávy úvodních fragmentů

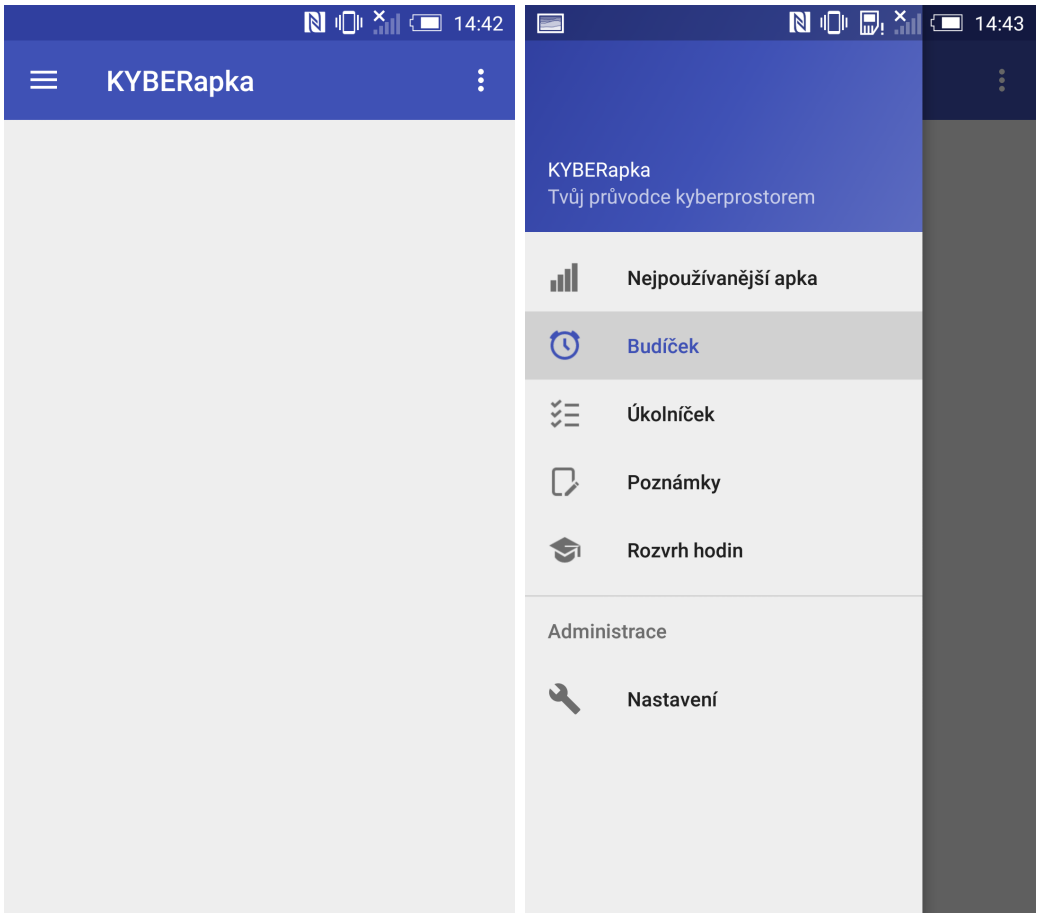
MainActivity

Aktivita reprezentovaná třídou `MainActivity` reprezentuje hlavní uživatelské rozhraní aplikace. Při jeho budování bylo snahou respektovat doporučení a principy uvedené v oficiální specifikaci *Material Design* určené pro vývoj uživatelských rozhraní OS Android [45]. Pro účely splnění požadavku na poskytnutí flexibilního uživatelské rozhraní, do kterého bude možné dle potřeby snadno a efektivně přidávat další obsah a rozšiřovat jej o novou funkcionalitu, je rozložení aktivity definováno použitím kontejneru `DrawerLayout` s vnořeným kontejnerem `CoordinatorLayout` a widgetem `NavigationView`. V rámci `CoordinatorLayout` je následně použit kontejner `FrameLayout` a kontejner `AppBarLayout` s widgetem `Toolbar`. Specifikace vzájemných vztahů jednotlivých widgetů a příslušných kontejnerů jsou definovány v rámci souboru `activity_main.xml` a `app_bar_main.xml` v adresáři `res/layout/`. Na obrázku 5.13 je formou stromové struktury zobrazeno základní rozložení hlavní aktivity.



Obr. 5.13: Základní rozložení hlavní aktivity

Princip vytváření a doplňování nového obsahu a s ním spojené funkcionality spočívá ve vytváření jednotlivých fragmentů, jež mohou být do hlavní aktivity dynamicky vkládány skrze kontejner `FrameLaylout`. Odkazy na aktivaci jednotlivých fragmentů jsou vkládány do widgetu `NavigationView`. Základní grafická struktura je zobrazena na obrázku 5.14.



(a) Hlavní zobrazení (b) Otevřený widget `NavigationView`

Obr. 5.14: Základní grafická struktura hlavní aktivity

Výhod použití jedné hlavní aktivity a dynamického vkládáni jednotlivých fragmentů je celá řada. Za zmínku stojí např.:

- menší náročnost na potřebnou operační paměť zařízení pro zobrazení uživatelského rozhraní (načtena je pouze jedna aktivita a v ní jsou dynamicky vkládány jednotlivé fragmenty namísto načítání nových vnořených aktivit),
- možnost opakovaného využití konkrétního fragmentu napříč různými poskytovanými funkcemi aplikace,
- možnost optimalizovat uživatelské rozhraní pro různé velikosti obrazovky (např. zobrazení více fragmentů současně u větších obrazovek).

Ukázkový fragment v rámci hlavní aktivity

Vytvořena struktura uživatelského rozhraní je doplněna o funkcionalitu na poskytování zpětné vazby uživateli o tom, jak využívá své mobilní zařízení z hlediska času stráveného v jednotlivých aplikacích. Jedná se o fragment, jenž obsahuje seznam

nejčastěji využívaných aplikací seřazen v sestupném pořadí, tzn. od nejčastěji používané aplikace po nejméně využívanou aplikaci. Seznam nejčastěji využívaných aplikací může být zobrazen pro tři časové intervaly: aktuálně probíhající den, týden či měsíc. Jednotlivé položky seznamu obsahují ikonu dané aplikace, její název a celkovou dobu aktivního běhu aplikace za zvolený časový interval. Pro funkci poskytování informací o aktivní době využívání jednotlivých aplikací je využito API `android.app.usage`.

Za účelem efektivního zobrazení jednotlivých seznamů nejčastěji využívaných aplikací pro konkrétní časové intervaly je využito kontejneru `TabLayout` a `ViewPager`. Oba kontejnery jsou vnořeny do kontejneru `LinearLayout` s vertikální orientací. V souboru `fragment_for_usage_statistics.xml` se nachází definice rozložení hlavního fragmentu. Danému fragmentu odpovídá třída `FragmentForUsageStatistics`, ve kterém se nachází logika pro konfiguraci potřebných adaptérů a nastavení vzájemného propojení kontejneru `TabLaylout` a `ViewPager` spolu s načtením dílčích fragmentů, jež zobrazují seznamy nejčastěji využívaných aplikací.

Celkem jsou vytvořeny tři dílčí fragmenty (pro každý časový interval jeden):

- `fragment_day.xml` s odpovídající třídou `DayFragment`,
- `fragment_week.xml` s odpovídající třídou `WeekFragment`,
- `fragment_month.xml` s odpovídající třídou `MonthFragment`.

Kořenovým kontejnerem těchto fragmentů je `RelativeLayout`, v němž je použít widget `RecyclerView`. Tento widget slouží jako kontejner pro zobrazování velkého množství dat, jež mohou být na obrazovce rolována či listována. Hlavní výhodou je, že načtená data se zobrazují dynamicky dle potřeby, čili není vytvořen grafický obsah pro všechna načtená data, ale jen pro tu část, jež se vměstná na obrazovku. Při nutnosti zobrazit větší množství dat tak nevzniká nežádoucí odezva a zpomalení chodu aplikace. Pomocí daného widgetu je tak vytvořen vertikální rolovací seznam. Definice grafického rozložení pro jednotlivé položky seznamu se nachází v souboru `usage_item_for_recycler.xml`.

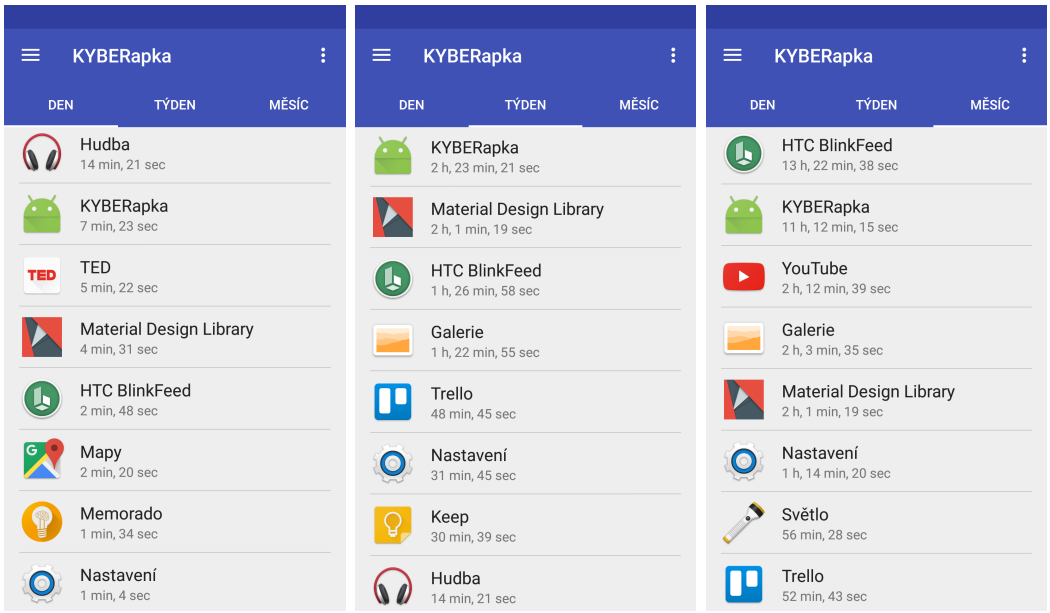
Vnitřní logika fungování widgetu se nachází ve třídě `UsageAdapter`, jež rozšiřuje třídu `RecyclerView.Adapter`. Nejpodstatnějšími metodami dané třídy jsou:

- `onCreateViewHolder()`: v rámci metody je adaptéru přiřazen soubor s definicí grafického rozložení položky seznamu (`usage_item_for_recycler.xml`),
- `onBindViewHolder()`: slouží k načtení konkrétních dat (tj. ikona, jméno a celkový čas běhu aplikace) z kolekce nejčastěji využívaných aplikací a přiřazení těchto dat do grafické struktury položky seznamu,
- `setUsageItemList()`: slouží k vytvoření kolekce typu `List` obsahující seřazený seznam nejčastěji využívaných aplikací pro konkrétní časový interval. Kolekce je vytvořena za pomocí API `android.app.usage`.

V rámci třídy `UsageAdapter` se nachází také statická třída `ViewHolder` (dědí ze třídy `RecyclerView.ViewHolder`), jež obsahuje reference na jednotlivé elementy položky seznamu.

Úkolem tříd obsluhující jednotlivé dílčí fragmenty (`DayFragment`, `WeekFragment` a `MonthFragment`) je v rámci metody `onCreateView()` načíst definici grafického rozložení příslušného fragmentu a přiřadit objektu třídy `RecyclerView` objekt třídy `UsageAdapter`, který obsahuje kolekci s nejčastěji využívanými aplikacemi pro časový interval odpovídající načtenému fragmentu.[23]

Výsledný grafický vzhled fragmentu, jenž obsahuje funkcionalitu na poskytování zpětné vazby uživateli o tom, jak využívá své mobilní zařízení z hlediska času stráveného v instalovaných aplikacích, je na obrázku 5.15. Vyobrazeny jsou všechny tři dílčí fragmenty reprezentující jednotlivé časové intervaly.



Obr. 5.15: Výsledný grafický vzhled fragmentu `FragmentForUsageStatistics`

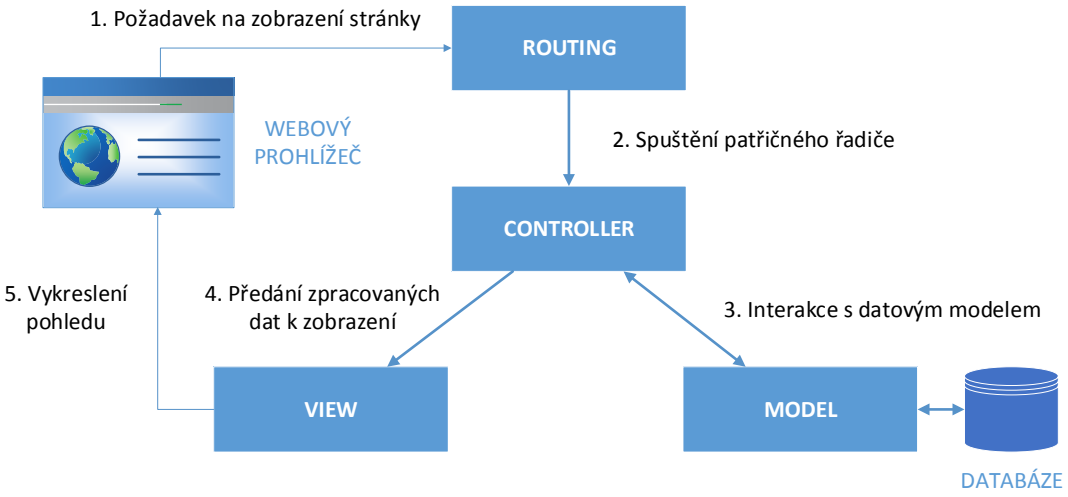
[23] Vzhledem k tomu, že pro získání seznamu nejčastěji využívaných aplikací bylo použito pouze samotné API `android.app.usage`, obsahují výsledné zobrazené seznamy nedostatky, jež jsou uvedené v části 4.2.2 této práce (např. nový týden začíná ve čtvrtek; začátek nového měsíce nepřipadá na 1. den v měsíci, apod.).

5.2 Vzdálený server

Hlavním úkolem vzdáleného serveru je příjem dat (tj. sumarizovaných týdenních statistik o nejčastěji využívaných mobilních aplikacích) od jednotlivých mobilních zařízení, jejich uložení do privátní databáze serveru a jejich následné zpracování dle požadavků Policie ČR uvedených v kapitole č. 2. Vyhodnocení dat je prezentováno skrze webové rozhraní serveru, které je primárně určeno pro potřeby pracovníků Policie ČR, resp. osob zapojených v rámci příslušného projektu.

5.2.1 Použité technologie

Technologie či prostředky potřebné pro vybudování vzdáleného serveru byly vybrány s cílem zajistit snadnou rozšiřitelnost serveru o nový obsah a novou funkcionalitu. Snahou bylo také poskytnout přehlednou správu vytvořeného kódu pro účely budoucího rozšiřování spolu se snadnou znovu použitelností jednotlivých komponent. Za tímto účelem byl využit open-source PHP framework *Laravel* [46], který slouží k vytváření webových aplikací. Laravel framework následuje návrhový vzor, resp. softwarovou architekturu MVC (Model View Controller)[24], kdy jsou jednotlivé části architektury, tzn. datový model, uživatelské rozhraní a řídicí logika vnímány jako nezávislé komponenty. Vzájemná interakce mezi jednotlivými komponentami je ve zjednodušené podobě zobrazena na obrázku 5.16. Obrázek je doplněn i o prvek routování, jenž je součástí fungování Laravel frameworku.



Obr. 5.16: Laravel MVC architektura [47]

[24] Česky: model, pohled, řadič.

V rámci diplomové práce byl jako webový server vybrán *Apache HTTP Server* a jako systém pro řízení báze dat byl zvolen *MySQL*. Vytvořenou webovou aplikaci lze nasadit i na odlišný webový server, např. *Nginx*, či je možné použít odlišný relační databázový systém, např. *PostgreSQL*. Díky vybudování webové aplikace s použitím frameworku Laravel je změna databázového systému otázkou úpravy příslušného konfiguračního souboru[25] bez nutnosti zásahu do již vytvořeného kódu.

Pro účely vybudování grafického rozhraní webové aplikace bylo využito prostředků open-source šablony *AdminLTE* [48]. Šablona je postavena na front-end[26] webovém frameworku *Bootstrap*, který mimo jiné podporuje tzv. responzivní design, jenž umožňuje vývojáři definovat rozložení webové stránky tak, aby se dynamicky přizpůsobila zařízení, na němž je zobrazena.

5.2.2 Základní konfigurace

Cílem této části je uvedení základních kroků potřebných ke zprovoznění vytvořené webové aplikace, resp. jejímu nasazení na konkrétní fyzické zařízení. Jedná se o projekt `KYBER_server` nacházející se na přiloženém CD. Jednotlivé kroky jsou uváděny v souvislosti s nasazením webového serveru a webové aplikace na OS Windows. V případě linuxových systémů je však většina kroků prakticky totožná.

Nejdříve je nutné provést instalaci webového serveru Apache, dále relačního databázového systému MySQL a také podporu PHP. Nejjednodušším řešením je využití např. softwarového balíčku *WampServer*,[27] jenž obsahuje uvedené komponenty a poskytuje možnost jejich základní konfigurace skrze jednoduché uživatelské rozhraní. V rámci souboru `httpd-vhosts.conf` webového serveru Apache je možné definovat vlastní virtuální server, resp. virtuálního hostitele, tzv. *VirtualHost* a nastavit odpovídající údaje jako je kořenový adresář (`DocumentRoot`), název domény (`ServerName`), IP adresu a port, nastavení pro přístup z internetu atp. Důležité je také nastavení práv jednotlivých adresářů webového serveru či nastavení pravidel firewallu. Popis zabezpečení samotného webového serveru je ovšem nad rámec tohoto textu.

Druhým krokem je instalace aplikace *Composer*. Jedná se o nástroj pro správu závislostí v rámci PHP projektů. Pomocí nástroje Composer jsou vždy staženy odpovídající verze balíčků či knihoven potřebných pro fungování vytvořené webové aplikace.

Třetím krokem je naimportování samotné webové aplikace na webový server, tj. do kořenového adresáře příslušného virtuálního hostitele. Projekt webové aplikace je

[25] Soubor `config/database.php`.

[26] Část webové aplikace, kterou vidí její uživatel.

[27] Pro linuxové prostředí je možné využít např. *LAMP Server*.

možné zkopírovat z přiloženého CD či ji stáhnout z online úložiště *GitHub*[28] pomocí příkazu:

```
# git clone https://github.com/xbolek00/KYBER_server.git
```

Pro použití uvedeného příkazu je zapotřebí mít nainstalovaný verzovací systém *Git*. Z pohledu webového serveru je zapotřebí v rámci souboru `httpd-vhosts.conf` upravit kořenový adresář virtuálního hostitele tak, aby byl zanořen na adresář `public` webové aplikace, tj.:

```
DocumentRoot <původní_kořenový_adresář>/KYBER_server/public
```

Následně je potřeba nainstalovat závislosti projektu pomocí nástroje Composer:

```
# composer install
```

Příkaz se spouští v rámci kořenového adresáře projektu webové aplikace, tzn. v adresáři `KYBER_server` a po jeho vykonání je vytvořen adresář `vendor` s odpovídajícími balíčky a knihovnami.

Dalším krokem je vytvoření databáze MySQL a příslušných tabulek. Databázi je možné vytvořit např. pomocí MySQL konzole či pomocí nástroje *phpMyAdmin*. Tabulky databáze není potřeba vytvářet manuálně, ale vytvářejí se na základě tzv. migrací, jež jsou uloženy v projektu webové aplikace. Použitím příkazu:

```
# php artisan migrate
```

dojde k vytvoření schématu databáze se všemi definovanými tabulkami na základě jednotlivých migrací.

Posledním krokem je vytvoření a nastavení souboru `.env` v kořenovém adresáři projektu. Jedná se o soubor, který není běžně verzován, jelikož obsahuje nastavení pro konkrétní prostředí včetně citlivých informací jako je např. heslo k databázi či aplikační klíč webové aplikace. Konfigurační soubor je možné buďto vytvořit nový anebo lze použít soubor `.env.example` a po jeho editaci jej uložit jako soubor `.env`. Ke zprovoznění webové aplikace je zapotřebí především vytvořit konfiguraci pro vybraný databázový systém. Jedná se o tyto položky:

- `DB_CONNECTION`: typ použité databáze,
- `DB_HOST`, `DB_PORT`: IP adresa a port běžícího databázového serveru,
- `DB_DATABAZE`: název vytvořené databáze,
- `DB_USERNAME`: jméno uživatele databáze,
- `DB_PASSWORD`: heslo pro přístup k databázi.

Důležité je také vygenerovat aplikační klíč, který je frameworkem využíván pro celou řadu operací spojených např. s šifrováním dat či uživatelských realací.

```
# php artisan key:generate
```

[28]Nutno požádat o přidělení práv autorem práce.

Ukázka nastavení konfiguračního souboru `.env` je v příloze D.

V této chvíli je vhodné restartovat webový server. Pokud je povolen přístup k webovému severu z internetové sítě, je webová aplikace připravena ke sběru dat od jednotlivých mobilních zařízení. Uživatelské rozhraní serveru je přístupné skrze internetový prohlížeč po zadání příslušné IP adresy či doménového jména (dle nastavení webového serveru).

5.2.3 Struktura webové aplikace

Účelem následujícího textu je představení základní struktury projektu webové aplikace, resp. částí, jež reprezentují uživatelské rozhraní, řídíci logiku a databázový model. Bližší informace týkající se ostatních části adresářové struktury projektu lze dohledat v online dokumentaci frameworku Laravel [49].

View – pohled

Pro optimální strukturování uživatelského rozhraní je využit šablonový systém *Blade*, jenž je součástí frameworku Laravel. Hlavní výhodou jeho použití je možnost definování jednotlivých sekcí webové stránky a jejich vkládání do navržené struktury. Díky tomuto přístupu je dosaženo vysoké úrovně znovu použitelností jednotlivých části uživatelského rozhraní.

Jednotlivé pohledy se nacházejí v adresáři `resources/views/`. Základní struktura uživatelského rozhraní je reprezentována souborem `app.blade.php` a lze ji rozdělit na čtyři základní části:

- záhlaví – `main_header.blade.php`,
- boční nabídka – `main_sidebar.blade.php`,
- panel s hlavním obsahem (celkem 5 různých panelů),
- zápatí – `main_footer.blade.php`.

Panely s hlavním obsahem se nacházejí ve vnořeném adresáři `pages/` a zprostředkovávají vizualizaci zpracovaných dat dle požadavků Police ČR. Jedná se o soubory:

- `admin_panel.blade.php`,
- `trends_panel.blade.php`,
- `topapps_panel.blade.php`,
- `topapps_sex_panel.blade.php`,
- `topapps_ageGroups_panel.blade.php`.

Součástí jednotlivých panelů je sekce určená pro definování jejich grafické struktury (`@section('content')`) a také sekce určená pro kód zajišťující interaktivitu panelu a vizualizaci dat (`@section('scripts')`).

Controller – řadič

Jednotlivé řadiče se nacházejí v adresáři `app/Http/Controllers/`. Celkem jsou vytvořeny čtyři řadiče:

- `AdminController`,
- `TrendsController`,
- `TopAppsController`,
- `InputController`.

První tři řadiče jsou určeny pro jednotlivé panely a čtvrtý řadič slouží k obsluze události při zaslání dat ze strany mobilních zařízení na vzdálený server. Třídy jednotlivých řadičů dědí ze třídy `Controller` frameworku Laravel.

Při vytváření nových řadičů není potřeba vytvářet nový soubor, ale je možné využít konzole *Artisan* frameworku Laravel. Pomocí příkazu:

```
# php artisan make:controller NewController --plain
```

je vygenerována třída pro nový řadič, ve které lze vytvářet potřebnou logiku.

Rozhodnutí o tom, který řadič má být spuštěn (a která z jeho metod), je určováno na základě definice rout. Definice jednotlivých rout se nachází v souboru `web.php` v adresáři `routes/`. Ukázka vytvořených rout je ve výpisu kódu 5.7.

Výpis 5.7: Routy webové aplikace – `web.php`

```
1  //Routy pro jednotlivé panely, resp. stránky webové aplikace
2  Route::get('/', 'AdminController@index');
3  Route::get('trends', 'TrendsController@index');
4  Route::get('topapps', 'TopAppsController@index');
5  Route::get('topapps_sex', 'TopAppsController@sex');
6  Route::get('topapps_age', 'TopAppsController@ageGroups');
7
8  //Routy pro zasílání dat z mobilních zařízení
9  Route::post('register', 'InputController@registerApp');
10 Route::post('uploadStatistics', 'InputController@saveData');
```

Model – model

Součástí frameworku Laravel je *Eloquent ORM*, jenž zajišťuje konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem PHP. Webová aplikace disponuje dvěma modely, jež využívají prostředky Eloquent ORM. Jedná se o model `Input` a `User`. Soubory reprezentující tyto modely jsou umístěny v adresáři `app/`. Pomocí vytvořených modelů je následně manipulováno se záznamy tabulek databáze (tvoření dotazů, vkládání nových záznamů, mazání záznamů atp.).

Odpovídající tabulky pro uvedené modely jsou `input` a `users`. Tabulka `input` slouží pro ukládání týdenních souhrnů nejčastěji využívaných aplikací od jednotlivých uživatelů a tabulka `users` je určena k uložení registračních záznamů. Jednotlivé tabulky jsou definovány pomocí migrací `CreateInputTable` a `CreateUsersTable`, jež se nacházejí v adresáři `database/migrations/`. Soubor s novou migrací lze vytvořit pomocí příkazu:

```
# php artisan make:migration <název_migrace>
```

a v rámci nově vytvořeného souboru definovat potřebnou strukturu tabulky. Definice atributů a jejich datových typů jednotlivých tabulek jsou uvedeny ve výpisu kódu 5.8 (tabulka `input`) a 5.9 (tabulka `users`). Tabulky byly navrženy na základě požadavků Police ČR a v současné verzi projektu webové aplikace není mezi uvedenými tabulkami definován žádný vztah.

Výpis 5.8: Zkrácený výpis migrace `CreateInputTable`

```
Schema::create('input', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->unsignedInteger('week');
        $table->string('user_id');
        $table->string('sex');
        $table->unsignedInteger('age');
        $table->string('package_name')->unique();
        $table->string('app_name');
        $table->unsignedBigInteger('total_time');
        $table->timestamp('added_on');
});
```

Výpis 5.9: Zkrácený výpis migrace `CreateUserTable`

```
Schema::create('users', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('user_id')->unique();;
        $table->string('sex');
        $table->unsignedInteger('age');
        $table->timestamp('added_on');
        $table->unsignedInteger('firstRunWeek');
});
```

5.2.4 Vytvořené panely

Následující text je věnován popisu jednotlivých panelů webové aplikace, jež byly vytvořeny za účelem splnění požadavků Policie ČR na zpracování dat získaných z mobilních zařízení uživatelů.

Administrativní panel

Účelem administrativního panelu je poskytnutí základních informací o zastoupení uživatelů v rámci realizovaného systému. Jedná se o výchozí panel, jenž je zobrazen při navštívení webové aplikace. Ukázka uživatelského rozhraní panelu se nachází v příloze E.1.

Administrativní panel reprezentuje pohled `admin_panel.blade.php`. Pro vizualizaci dat jsou využity knihovny *ChartJS*[29] a *jQuery Knob*. Odpovídajícím řadičem je `AdminController`. Pro práci se záznamy databáze jsou využity oba modely, tj. `Input` a `User`. Příslušná routa je následující:

```
Route::get('/', 'AdminController@index').
```

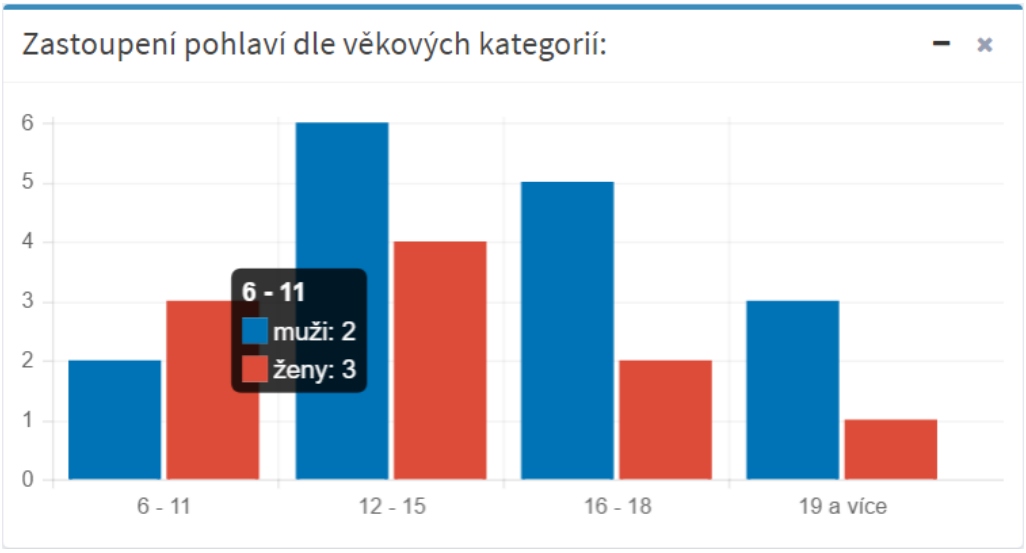
Administrativní panel poskytuje informace o celkovém počtu registrovaných uživatelů[30] a procentuálním zastoupení můžů a žen. Dále je zobrazena tabulka s rozborem zastoupení uživatelů dle jednotlivých věkových kategorií včetně početního zastoupení pohlaví v každé kategorii (obrázek 5.17). Tabulka je doplněna o sloupec s grafickým ukazatelem průběhu, jenž vizualizuje procentuální zastoupení jednotlivých věkových kategorií a usnadňuje tak orientaci v tabulce. Pro grafické zobrazení zastoupení pohlaví v jednotlivých věkových kategoriích slouží sloupcový graf (obrázek 5.18), kde osa x představuje definované věkové kategorie a osa y celkový počet uživatelů. Při pohybu myši nad grafem je zobrazována legenda[31] s číselnou reprezentací celkového počtu uživatelů zastoupených dle pohlaví v příslušné věkové kategorii.

Obr. 5.17: Rozbor zastoupení uživatelů dle jednotlivých věkových kategorií

[29] Ve verzi 1.0.1.

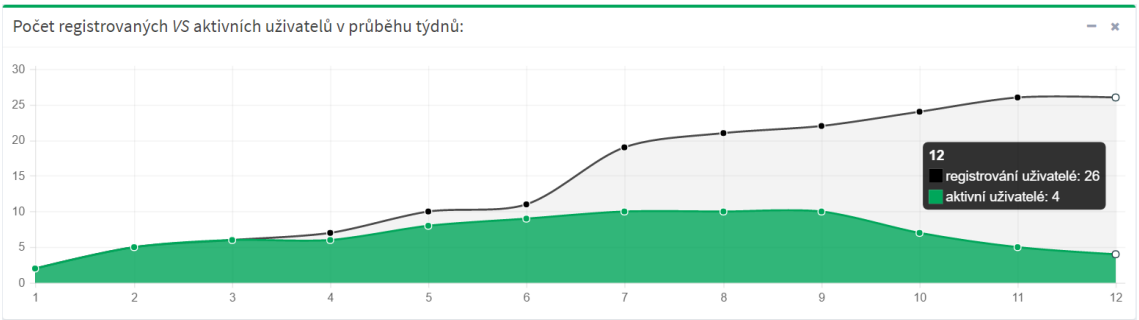
[30] Registrovaný není uživatel jako takový, ale instance vyvíjené Android aplikace, kterou využívá (zajištění anonymity) – více viz 5.3.1.

[31] Při přístupu k webové aplikaci pomocí mobilního zařízení je legenda zobrazena při kliknutí na příslušný sloupec grafu.



Obr. 5.18: Zastoupení pohlaví v definovaných věkových kategoriích

V rámci administrativního panelu se nachází také grafické zobrazení průběžného nárůstu počtu registrovaných uživatelů společně s nárůstem počtu aktivních uživatelů v průběhu jednotlivých týdnů (ukázka na obr. č. 5.19). Osa x představuje čísla týdnů v roce[32] a osa y představuje počet uživatelů. Interaktivní legenda je na obrázku zobrazena pro týden č. 12. Spojnicový graf obsahuje dvě datové řady. První reprezentuje celkový počet provedených registrací (počet záznamů v tabulce `users`) a druhá celkový počet aktivních uživatelů vždy k patřičnému týdnu (vypočteno na základě zpracování záznamů v tabulce `input`). Pod pojmem „aktivní“ je zde myšleno to, že z daného zařízení uživatele dorazil pro odpovídající týden jeho týdenní souhrn nejčastěji využívaných aplikací. Počet aktivních uživatelů tak představuje celkový počet přijatých týdenních souhrnů. Hlavní účelem uvedeného grafu je poskytnutí zpětné vazby o tom, zda počet registrací odpovídá počtu přijatých týdenních souhrnů. Za běžných okolností, tj. kdy vše funguje jak má, se obě datové



Obr. 5.19: Počet registrovaných a aktivních uživatelů v průběhu týdnů

[32]Počínaje číslem týdne, kdy došlo k první registraci Android aplikace na serveru.

řady vzájemně překrývají. Může také nastat menší pokles počtu přijatých týdenních souhrnů oproti počtu registrací např. v případě dlouhodobějšího vypnutí mobilního zařízení uživatelem. Rapidní pokles aktivních uživatelů oproti registrovaným může reprezentovat např. stav, kdy vnitřní logika vyvíjené Android aplikace přestane fungovat při aktualizaci OS mobilního zařízení na novější verzi. Díky včasné detekci uvedeného stavu je následně možné např. připravit potřebnou opravu aplikace.

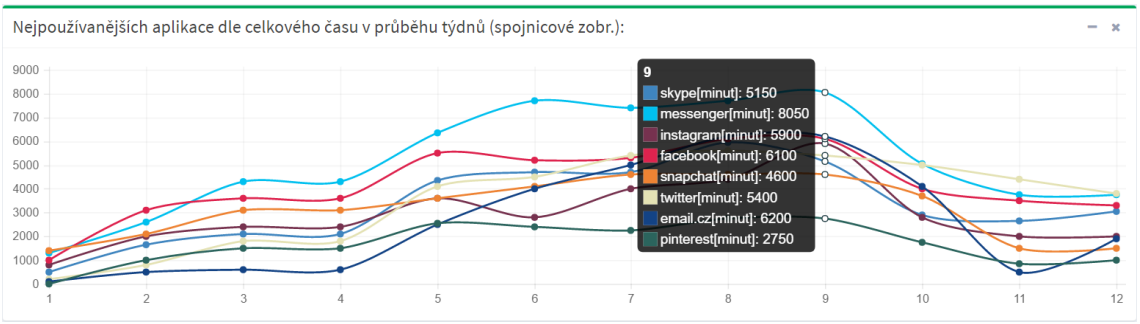
Panel vývojových trendů

Panel vývojových trendů slouží k zobrazení nejpoužívanějších aplikací dle celkového času v průběhu jednotlivých týdnů. Vzhled panelu se nachází v příloze E.2.

Pohled panelu je definován v souboru `trends_panel.blade.php` a příslušným řadičem je `TrendsController`. Definovaná routa pro tento panel je:

```
Route::get('trends', 'TrendsController@index').
```

Panel vývojových trendů je tvořen dvěma grafy – spojnicovým a sloupcovým, jež jsou vytvořeny za pomocí knihovny ChartJS verze 1.0.1. Ukázka spojnicového grafu je na obrázku 5.20. Vstupní data pro oba grafy jsou totožná a jsou vytvořena na základě zpracování záznamů z tabulky `input` (pro každou aplikaci je vypočtena celková doba jejího aktivního běhu v rámci příslušného týdne). Množství celkového času stráveného v jednotlivých aplikacích je závislé na celkovém počtu aktivních uživatelů v daném týdnů, jelikož zde není počítán průměr či medián hodnot, ale pouze celkový součet (dle zadání). Časová osa x reprezentuje čísla jednotlivých týdnů a osa y udává celkový počet minut strávených v příslušné aplikaci. Interaktivní legenda grafu zobrazuje názvy jednotlivých aplikací a jejich celkovou aktivní dobu běhu v minutách pro vybraný týden.



Obr. 5.20: Nejpoužívanější aplikace dle celkového času v průběhu týdnů

Panely top aplikací

Účelem panelů top aplikací je vyhodnocení nejpoužívanějších mobilních aplikací z hlediska celkového času stráveného v příslušné aplikaci a také z hlediska zastoupení

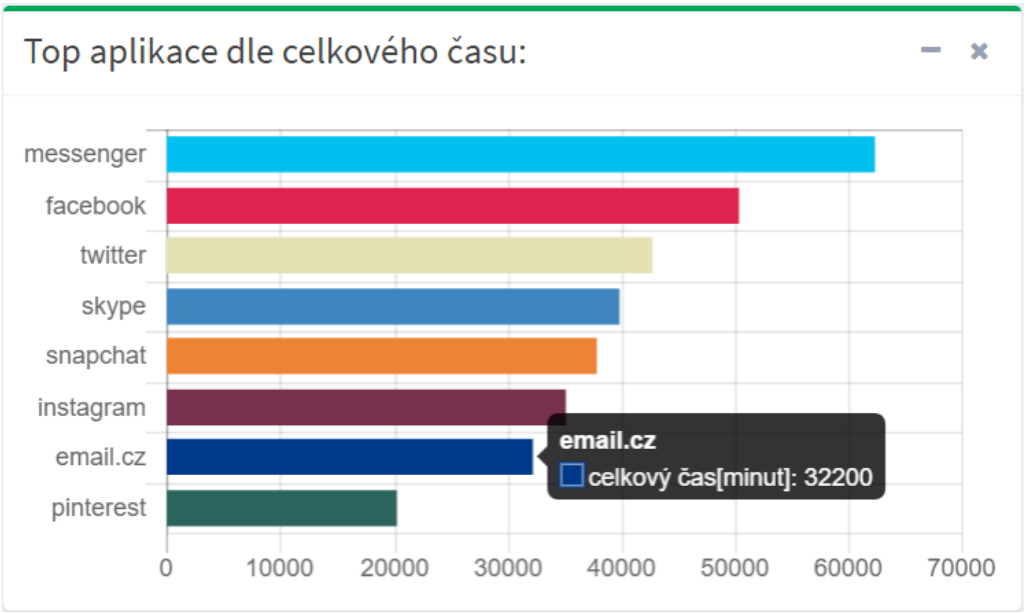
dané aplikace mezi uživateli. Celkem jsou vytvořeny tři dílčí panely, jejichž vzhled se nachází v příloze E.3:

- Top aplikace: celkový souhrn,
- Top aplikace: dle pohlaví,
- Top aplikace: dle věkové kategorie.

Příslušné pohledy pro uvedené panely jsou následující: `topapps_panel.blade.php`, `topapps_sex_panel.blade.php` a `topapps_ageGroups_panel.blade.php`. Všechny panely obsluhuje společný řadič `TopAppsController`. Odpovídající routy jsou tyto:

```
Route::get('topapps', 'TopAppsController@index'),
Route::get('topapps_sex', 'TopAppsController@sex'),
Route::get('topapps_age', 'TopAppsController@ageGroups').
```

Panel s celkovým souhrnem obsahuje dva horizontální sloupcové grafy, jež jsou vytvořeny za pomocí knihovny ChartJS verze 2.5.0.[33] První graf reprezentuje nejpoužívanější mobilní aplikace z hlediska celkového času stráveného v příslušné aplikaci (obrázek 5.21). Osa x udává celkový počet minut, jež uživatelé v dané aplikaci strávili a osa y představuje jednotlivé mobilní aplikace. Druhý graf reprezentuje nejpoužívanější mobilní aplikace z hlediska zastoupení dané aplikace mezi uživateli. Osa y je stejná jako u předchozího grafu a osa x uvádí počet výskytu dané aplikace mezi uživateli. Součástí grafů je také interaktivní legenda uvádějící název aplikace a odpovídající celkový počet minut či celkový počet výskytů (dle typu grafu). Důležité je podotknout, že současná verze webové aplikace zobrazuje výsledky zpracované za



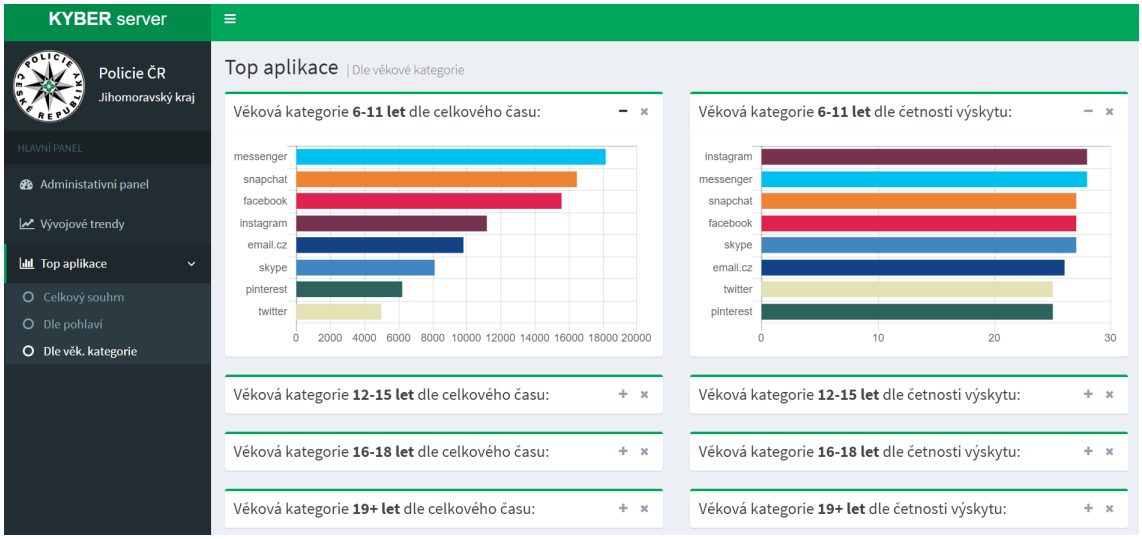
Obr. 5.21: Nejpoužívanější aplikace dle celkového času

[33] Disponuje zcela odlišným API než verze 1.x.x.

období všech týdnů, ve kterých byly přijaty týdenní souhrny. V další fázi projektu bude webová aplikace doplněna o možnost zobrazení výsledků zpracování dat pro konkrétní týden či období několika týdnů (viz závěrečná doporučení v části 6.2).

Panel top aplikací dle pohlaví obsahuje celkem čtyři grafy. Jedná se o stejné typy grafů jako u předchozího panelu a probíhá zde obdobné zpracování záznamů z tabulky `input`. Vyhodnocovány jsou nejpoužívanější mobilní aplikace z hlediska celkového času stráveného v příslušné aplikaci a z hlediska zastoupení dané aplikace mezi uživateli pro jednotlivá pohlaví.

U třetího panelu top aplikací se jedná o totožné vyhodnocení ovšem pro jednotlivé věkové kategorie (celkem jsou definovány čtyři kategorie). Panel top aplikací dle věkových kategorií tak obsahuje osm dílčích grafů, jejichž obsah lze v případě potřeby skrýt či je z panelu zcela odebrat pomocí tlačítek v pravém horním rohu boxu[34] (viz obrázek 5.22).



Obr. 5.22: Panel „Top aplikace“ pro definované věkové kategorie

[34]Tato možnost platí také pro všechny výše uvedené grafy.

5.3 Komunikace aplikace-server

Cílem této části je přiblížit fungování realizovaného systému z pohledu operací, jež jsou spojeny se vzájemnou komunikací mezi Android aplikací a vzdáleným serverem, resp. webovou aplikací. Vzájemná komunikace je navržena s ohledem na efektivní využití síťových prostředků a co nejnižší využití baterie mobilního zařízení. Existují pouze dva scénáře, kdy dochází k vzájemné komunikaci. Prvním scénářem je prvotní registrace aplikace na vzdáleném serveru – jedná se o jednorázovou záležitost, a druhým je pravidelné zasílání sumarizovaných týdenních statistik[35] na vzdálený server – komunikace se opakuje v týdenních intervalech.

5.3.1 Registrace aplikace

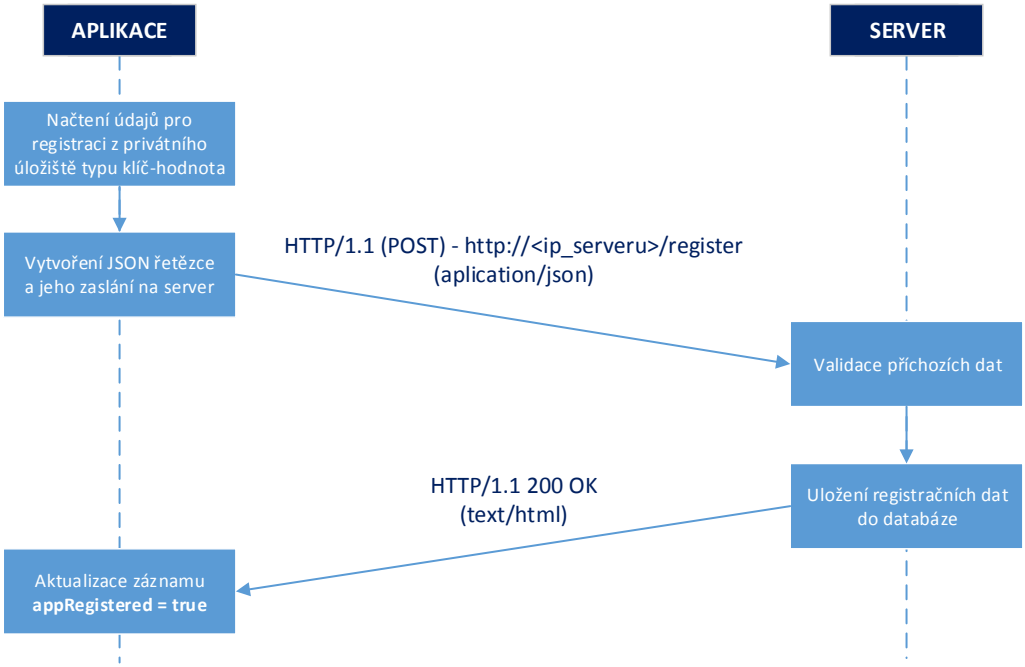
Dříve než bude popsán samotný proces registrace, je potřeba objasnit, co je pod tímto pojmem myšleno. Registrace aplikace na vzdáleném serveru slouží k identifikaci konkrétní instance vyvíjené Android aplikace, od které server následně přijímá zasílané týdenní sumarizované statistiky. Tato identifikace slouží primárně ke dvěma účelům. Prvním účelem je zajištění možnosti získávání určité zpětné vazby o tom, jak je realizovaný systém využíván, tzn. lze např. sledovat průběžný nárůst registrací vyvíjené aplikace a na základě znalosti celkového počtu registrací lze následně určit, kolik týdenních souhrnů má být zasláno na vzdálený server (viz administrativní panel webové aplikace 5.2.4). Druhým účelem je zajištění bezpečnosti systému před případnými útoky, které bude realizováno v další fázi vývoje systému a není předmětem této diplomové práce (viz závěrečná doporučení v části 6.2).

Důležitým aspektem realizovaného systému je zajištění anonymity uživatele. Z tohoto důvodu není z pohledu serveru identifikován ani samotný uživatel, ani jeho fyzické mobilní zařízení, ale pouze instance vyvíjené aplikace, jež je nainstalována v mobilním zařízení uživatele. Při odinstalaci aplikace a jejím opětovném nainstalování je vygenerováno nové UUID a je provedena nová registrace aplikace na vzdáleném serveru. Jak již bylo uvedeno dříve, jedná se o automatickou registraci bez vědomí uživatele.

Proces registrace aplikace na vzdáleném serveru je naznačen na obrázku 5.23. Po aktivaci naplánované úlohy a spuštění vlákna `AppRegistrationThread` jsou z privátního úložiště typu klíč-hodnota načtena data související s registrací aplikace (věk a pohlaví uživatele; jedinečné identifikační číslo aplikace UUID; číslo týdne, ve kterém byla aplikace poprvé spuštěna). Z načtených dat je vytvořen řetězec ve formátu JSON, který je zaslán do webové aplikace metodou POST protokolu HTTP[36] skrze

[35] Jedná se o statistiky využívání jednotlivých aplikací v mobilním zařízení uživatele.

[36] Hypertext Transfer Protocol.



Obr. 5.23: Ukázka procesu registrace aplikace na vzdáleném serveru

navázané TCP[37] spojení na adresu `http://<ip_serveru>/register`. Na straně serveru je aktivován řadič `InputController`, jenž je registrován pro cestu `/register`. Následně je spuštěna příslušná metoda řadiče, tj. metoda `registerApp()`, která zajišťuje základní validaci příchozích dat a jejich uložení do databáze. Z pohledu validace je kontrolováno především to, zda jsou přítomny všechny povinné údaje a zda splňují požadovanou délku znaků. U identifikačního čísla aplikace je ověřováno, zda je v rámci tabulky `users` jedinečné. Pokud validace proběhne v pořádku, jsou příchozí údaje uloženy do tabulky `users`. Identifikační číslo aplikace je na straně webové aplikace dále vnímáno jako identifikátor uživatele (v databázi uloženo pod atributem `user_id`). Po úspěšném uložení dat do databáze je aplikaci zaslána HTTP odpověď se stavovým kódem `200 OK`. Na základě přijetí této odpovědi je do privátního úložiště typu klíč-hodnota uložen záznam o úspěšné registraci (`appRegistered = true`). V případě, kdy je od vzdáleného serveru přijata jiná odpověď než `200 OK` (webová aplikace např. nedokázala zpracovat požadavek) nebo je vzdálený server nedostupný, pak je přenos dat označen za neúspěšný a k vytvoření záznamu v privátním úložišti nedochází. Úloha na registraci aplikace je následně pomocí API `JobScheduler` naplánována znovu.

[37] Transmission Control Protocol.

5.3.2 Zasílání týdenních statistik

Při zasílání týdenních souhrnů je způsob komunikace obdobný. V rámci vlákna `WeeklySyncThread` jsou vytvořeny sumarizované týdenní statistiky, jež jsou ve formátu JSON zaslány na vzdálený server spolu s informacemi o pohlaví a věku uživatele, UUID aplikace a číslem týdne, jemuž odpovídají dané statistiky. Zaslání dat probíhá stejným způsobem jako u registrace aplikace ovšem pro odlišné URI. Definovanou adresou je `http://<ip_serveru>/uploadStatistics`.

Na straně serveru je spuštěna metoda `saveData()` řadiče `InputController`, který je registrován i pro cestu `/uploadStatistics`. V rámci spuštěné metody následně probíhá validace příchozích dat (viz ukázka ve výpisu kódu 5.10). Kromě kontroly, zda jsou přítomny všechny povinné údaje a zda splňují požadovanou délku znaků (neplatí pro sumarizované týdenní statistiky), je ověřeno, zda se příchozí UUID aplikace nachází v tabulce `users`, tzn. zda již byla aplikace úspěšně zaregistrována. Pokud se UUID aplikace v tabulce nenachází, příchozí data jsou ignorována. V případě úspěšné validace jsou příchozí data uloženy do tabulky `input`. Při ukládání záznamů do tabulky je na straně serveru převeden celkový čas využití jednotlivých aplikací z jednotek milisekund na minuty[38].

Na základě přijetí odpovědi serveru o úspěšném uložení zaslaných dat (stavový kód `200 OK`) jsou v aplikaci z tabulky `DailyUsageStats` odstraněny záznamy pro číslo týdne, jehož sumarizované statistiky byly uloženy na serveru. Současně dochází k aktualizaci záznamu `weekNumber` v privátním úložišti typu klíč-hodnota. V případě neúspěšné komunikace se vzdáleným serverem k uvedeným krokům nedochází a úloha pro zpracování a zaslání sumarizovaných týdenních statistik je opětovně naplánována pomocí API JobScheduler.

Výpis 5.10: Ukázka validace příchozích dat v rámci metody `saveData()`

```
$this->validate($request, [
        'id' => 'required|exists:users,user_id|size:36',
        'age' => 'required|max:2',
        'sex' => 'required|max:6',
        'week' => 'required|max:2',
        'statistics' => 'required|array',
        'statistics.*.package' => 'required',
        'statistics.*.app' => 'required',
        'statistics.*.total_time' => 'required',
        ]);
```

[38] Dochází k zaokrouhlení na jednotky minut.

6 TESTOVÁNÍ SYSTÉMU A ZÁVĚREČNÁ DOPORUČENÍ

6.1 Testování systému

Testování realizovaného systému probíhalo průběžně během jeho vývoje a to na několika úrovních. Testována byla jak samotná Android aplikace a samotný vzdálený server (resp. webová aplikace), tak i vzájemná komunikace a fungování systému jako celku.

Účelem následujícího textu je stručně představit nástroje, jež byly použity v průběhu vývoje realizovaného systému pro účely testování, a uvést způsob jejich využití.

6.1.1 Testování Android aplikace

Uživatelská Android aplikace byla vyvíjena za pomocí vývojového prostředí Android Studio. Hlavním testovacím zařízením byl mobilní telefon HTC Sense 6.0 s verzí operačního systému Lillipop s úrovní Android API 21. Pro doplňkové testování bylo skrze vývojové prostředí vytvořeno virtuální zařízení Nexus 5X s úrovní Android API 22.

Vyvíjená Android aplikace byla testována jak z pohledu funkčnosti uživatelského prostředí, tak především z hlediska funkčnosti vnitřního mechanismu aplikace. Pro testování uživatelského prostředí nebylo využito žádných speciálních nástrojů. Testování probíhalo na základě manuálních testů, kdy např. pro úvodní aktivitu byla ověřována správnost validace zadaných vstupních informací o uživateli, znemožnění přechodu k dalším fragmentům při nesplnění požadovaných podmínek (např. povolení potřebného oprávnění), správnost přesměrování do patřičné aktivity na základě toho, zda byla aplikace spuštěná poprvé či opakovaně, správná odezva jednotlivých tlačítek a nabídek apod.

Pro účely základního testování vnitřního mechanismu aplikace byly využity standartní nástroje vývojového prostředí Android Studia (tj. Android Monitor, výpisy konzole, debug mode atp.). Pomocí těchto nástrojů byla v reálném čase ověřována funkčnost jednotlivých dílčích části vnitřního mechanismu. Vzhledem k tomu, že vnitřní mechanismus vyvíjené aplikace běží nezávisle na uživatelském rozhraní a k aktivaci klíčových komponent dochází v různých definovaných intervalech a pouze na krátkou dobu, bylo potřeba využít také několik externích nástrojů, díky kterým bylo možné testovat také vzájemnou kooperaci dílčích komponent a dlouhodobé fungování vnitřního mechanismu jako celku. Pro tyto účely bylo využito nástroje *AndroidDatabaseManager* [50], který umožňuje skrze uživatelské rozhraní aplikace nahlížet do jednotlivých tabulek privátní databáze SQLite. Druhým nástrojem je

Stetho [51] vyvinutý firmou Facebook, jenž poskytuje celou řadu užitečných funkcí pro testování a ladění připojeného mobilního zařízení skrze tzv. *Developer Tools* webového prohlížeče Google Chrome. Oba nástroje bylo potřeba integrovat do vyvíjené aplikace (více viz příslušné odkazy na dané nástroje).

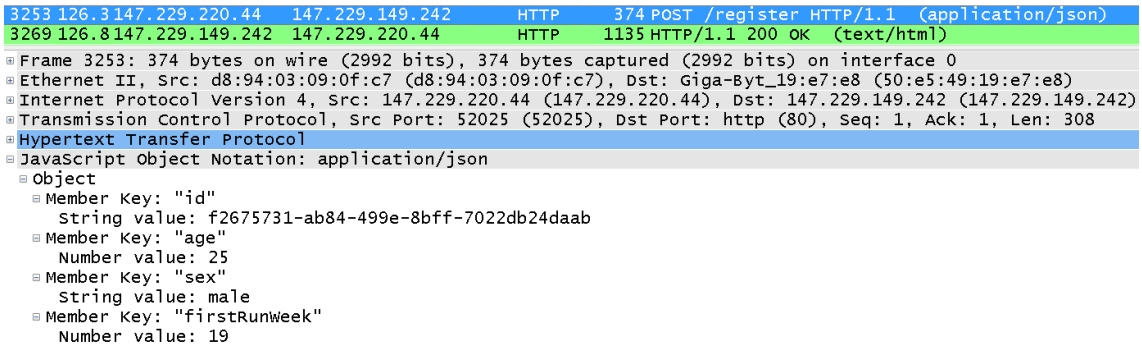
Testování správnosti fungování vnitřního mechanismu spočívalo na principu ukládání záznamu o chování jednotlivých komponent do privátní databáze SQLite a jejich následné analýzy. V rámci třídy `DatabaseController` byla vytvořena pomocná funkce `addForTesting()`, pomocí které byly do tabulky určené k ukládání denních statistik zaznamenávány také informace o aktivaci příslušných komponent spolu s názvy prováděných operací a informacemi o jejich úspěšném či neúspěšném vykonání. Veškeré ukládané informace byly opatřeny časovým razítkem, pomocí něhož bylo možné jednoznačně ověřit dobu aktivace dané komponenty a správnost pořadí aktivací dílčích komponent. Úspěšná aktualizace příslušných záznamů v privátním úložišti typu klíč-hodnota byla ověřována s pomocí nástroje Stetho. Na základě zpětné analýzy získaných informací byly identifikovány a následně odstraňovány vzniklé chyby. Ukázka vniklých testovacích záznamů pro příklad prvotního spuštění aplikace je vidět na obrázku 6.1. V červeném rámečku jsou vidět záznamy informující

ro...	m...	w...	d...	packageName	appName	ti...	timeStamp
1	9...	9...	9...	IntroActivity	AppRegistrationJobService SCHEDULED: OK!	9...	2017-05-10 13:56:59
2	9...	9...	9...	IntroActivity	MIDDAY ALARM was SET	9...	2017-05-10 13:56:59
3	9...	9...	9...	IntroActivity	MIDNIGHT ALARM was SET	9...	2017-05-10 13:56:59
4	9...	9...	9...	IntroActivity	DAYTIME ALARM was SET	9...	2017-05-10 13:56:59
5	9...	9...	9...	IntroActivity	firstRun set to FALSE	9...	2017-05-10 13:56:59
14	9...	9...	9...	Thread	AppRegistrationThread EXECUTED	9...	2017-05-10 13:59:03
15	9...	9...	9...	Thread	SUCCESS!!! App registered:-)	9...	2017-05-10 13:59:03
16	9...	9...	9...	Thread	AppRegistrationThread FINISHED	9...	2017-05-10 13:59:03

Obr. 6.1: Ukázka testovacích záznamů v tabulce `DailyUsageStats`

o úspěšném naplánování úlohy na registraci aplikace na vzdáleném serveru, nastavení patřičných alarmů a nastavení vnitřní proměnné `firstRun` na hodnotu `false`. Vzhledem k tomu, že v době po prvotním spuštění aplikace bylo dostupné internetové připojení, je možné v zeleném rámečku vidět aktivaci naplánované úlohy, resp. vlákna `AppRegistrationThread`, na registraci aplikace na vzdáleném serveru. Vidět lze záznam informující o spuštění vlákna, ke kterému došlo dvě minuty a čtyři sekundy po té, co byla úloha naplánována. Dále je zde záznam o úspěšné registraci aplikace a následně záznam o ukončení daného vlákna. Obdobným způsobem bylo postupováno při testování dalších části mechanismu či při testování fungování mechanismu jako celku. Pozornost byla zaměřena především na scénáře, jež by mohly narušit jeho standartní chod (restart mobilního zařízení, dlouhodobě nedostupné internetové připojení, neúspěšný přenos statistik apod.).

Pro účely testování vzájemné komunikace mezi aplikací a vzdáleným serverem byl využíván nástroj Wireshark. Ukázka zachycené komunikace pro výše zmíněný příklad, tj. registrace aplikace na vzdáleném serveru, je na obrázku 6.2. Je možné



Obr. 6.2: Ukázka zachycené komunikace pomocí nástroje Wireshark

si povšimnout např. JSON řetezce obsahující registrační údaje či odpovědi serveru `HTTP/1.1 200 OK`, jež označuje úspěšnou registraci aplikace. O úspěšném provedení registrace je možné se následně přesvědčit také na straně webové aplikace (vznik nového registračního záznamu).

Při testování vlivu realizovaného mechanismu na spotřebu baterie mobilního zařízení nebyla vyvíjená aplikace v historií využití baterie (funkce poskytovaná skrze nastavení systému) vůbec zaznamenána. Tento fakt odpovídá teoretickým předpokladům, jelikož k aktivaci příslušných komponent zajišťující fungování vnitřního mechanismu dochází pouze na velmi krátké okamžiky a prováděné operace nejsou pro dnešní mobilní zařízení nijak náročné. Efektivnímu využití baterie také přispívá nízká frekvence komunikace aplikace se vzdáleným serverem, která probíhá v týdenních intervalech, přičemž zasílaná data jsou přenesena pomocí jediného IP paketu.

6.1.2 Testování webové aplikace

Webová aplikace byla testována z pohledu správného fungování jednotlivých prvků uživatelského rozhraní a jejich zobrazování v různých webových prohlížečích, dále fungování vytvořených rout, ověření funkčnosti vstupní validace dat a také správnost zobrazovaných výsledků v jednotlivých panelech.

Správnost fungování uživatelského rozhraní aplikace byla otestována v prohlížečích Google Chrome[1], Mozilla Firefox[2] a Internet Explorer[3].

[1] Ve verzi 57.
[2] Ve verzi 53.
[3] Ve verzi 11.

Správnost zobrazovaných výsledků v jednotlivých panelech byla ověřena na základě testovací databáze. Vytvořena databáze byla naplněna záznamy se strukturou, jež odpovídá reálným statistikám zasílaných od jednotlivých mobilních zařízení. Na základě těchto záznamů bylo následně možné manuálně ověřit správnost jejich zpracování a grafické interpretace v příslušných panelech webové aplikace. Rozšířená verze testovací databáze byla použita pro demonstraci uživatelského rozhraní webové aplikace a je součástí obsahu přiloženého CD.

Pro úplnost je nutné podotknout, že současná verze webové aplikace disponuje několika omezeními, jež vycházejí z absence funkcionality, jejíž vývoj nebyl součástí zadání této diplomové práce, ale je obsahem další fáze plánovaného vývoje. Jedná se o funkcionalitu jako je např. možnost určení, které aplikace budou součástí grafických výstupů zpracování dat a které nikoliv, možnost určení konkrétního časového období, pro které mají být data zpracována, či možnost určení počtu zobrazených statistik. Absence této funcionality se může projevit například v nepřehlédnosti grafů v případě velkého množství zobrazených aplikací, či nesprávnými výsledky zpracování dat v případě, kdy by databáze obsahovala záznamy z několika let. Vzhledem k tomu, že nasazení realizovaného systému je plánováno až po dokončení výše uvedených funkcionalit, jedná se pouze o dočasná omezení.

6.2 Závěrečná doporučení

Implementace jednotlivých požadavků Policie ČR byla úspěšně dokončena a vyvinutý systém je funkční. Následující text stručně představuje další plánované kroky ve vývoji a nasazení realizovaného systému v rámci projektu „Bezpečně v kyberprostoru“ Jihomoravského kraje. Jedná se zejména o tyto záležitosti:

1. Rozhodnutí o tom, do jaké mobilní aplikace bude vnitřní mechanismus nasazen. Jedná se buďto o nasazení daného mechanismu do jiné existující mobilní aplikace či jeho ponechání v aplikaci vytvořené v rámci této diplomové práce. V případě druhé varianty bude uživatelské rozhraní aplikace doplněno o další obsah a s ním související funkcionalitu.
2. Otestování fungování vnitřního mechanismu na nejnovějších verzích OS Android a zařízeních různých výrobců. Pozornost je potřeba zaměřit především na novou funkcionalitu tzv. *run-time permissions* (práva přidělována za běhu aplikace), jež byla představena ve verzi OS Android 6.0 (verze API 23).
3. Vylepšení webové aplikace z pohledu efektivnějšího vyhodnocování získaných dat. Jedná se např. o tuto funkcionalitu:
   (a) možnost určení konkrétního časového období (v intervalech jednotlivých týdnů), pro které mají být data zpracována a následně zobrazena;

(b) možnost filtrování aplikací, jež nespadají do kategorie aplikací určených k virtuální komunikaci uživatelů;
(c) možnost určení počtu zobrazených aplikací v jednotlivých grafech;
(d) možnost procházení záznamů jednotlivých tabulek skrze uživatelské rozhraní webové aplikace;
(e) efektivnější uspořádání jednotlivých panelů na základě podnětů z praktického používání webové aplikace, tvorba nových panelů apod.

4. Nasazení webové aplikace na dedikovaný server včetně zajištění zabezpečené komunikace mezi Android aplikací a vzdáleným serverem pomocí HTTPS. Vytvoření přihlašovací stránky pro přístup k webové aplikaci.
5. Umístění Android aplikace do online distribuční služby Google Play v rámci programu *Google Play Beta Testing*, kdy je přístup k aplikaci umožněn na základě získané pozvánky.
6. Vylepšení procesu registrace aplikace na vzdáleném serveru za účelem ochrany realizovaného systému před potencionálními útoky. Hlavním cílem je umožnit příjem statistik pocházejících pouze z oficiální verze vytvořené Android aplikace. Díky předchozímu kroku, tj. umístění aplikace do online distribuční služby Google Play, bude možné k ověření autentičnosti aplikace využít například API *InstanceID*. V kombinaci s využitím API *SafetyNet Attestation* bude možné při registraci aplikace na serveru např. určit, zda je aplikace nainstalovaná na virtuálním či opravdovém fyzickém zařízení. Bližší informace lze nalézt ve zdrojích [52] a [53].

7 ZÁVĚR

Diplomová práce byla řešena ve spolupráci s Policií České republiky a měla za úkol vývoj systému, který poskytuje kontinuální sběr a vyhodnocování informací o tom, kolik času uživatelé tráví v jednotlivých aplikacích určených pro mobilní zařízení s operačním systémem Android. Na základě získávání anonymních statistik je možné sledovat, jaké jsou nejčastěji využívané mobilní aplikace dle pohlaví a věku uživatelů. Díky těmto informacím bude možné lépe reagovat na nově vznikající trendy ve virtuálním prostředí a v rámci přípravy preventivních programů se tak efektivně zaměřit na rizikové oblasti. Realizovaný systém bude sloužit pro účely projektu Jihomoravského kraje „Bezpečně v kyberprostoru“ jako podpůrný nástroj pro zkvalitnění prevence kyberšikany.

V textu byly nejdříve stručně popsány nebezpečné jevy internetu z pohledu problematiky kyberšikany a jevů s ní bezprostředně souvisejících. Následně byly uvedeny požadavky na realizovaný systém a dále byl představen OS Android, jeho základní komponenty a princip fungování životního cyklu aktivity.

Další část práce se věnovala analýze problematiky měření aktivní doby běhu mobilních aplikací v OS Android. Na základě diskuze nalezených úskalí bylo s pracovníky Policie ČR rozhodnuto o zaměření vývoje aplikace na verze OS Android 5.0 a vyšší. Hlavním výstupem následné analýzy byla volba API `android.app.usage`, které poskytuje přístup k historii využívání mobilního zařízení a umožňuje získat informace o aktivní době běhu jednotlivých aplikací, jež se v mobilním zařízení nacházejí.

Stěžejní část práce byla věnována návrhu a realizaci systému, který se skládá z uživatelské Android aplikace a vzdáleného serveru s webovou aplikací. Uživatelská Android aplikace obsahuje vnitřní mechanismus, pomocí něhož jsou průběžně získávány a ukládány denní statistiky využití zařízení. Získané statistiky jsou v týdenních intervalech dále zasílány na vzdálený server ve formě sumarizovaných týdenních souhrnů. Vnitřní mechanismus pracuje na pozadí operačního systému a funguje bez ohledu na to, zda je aplikace uživatelem aktivně využívána či nikoliv. Velkou výhodou je možnost jeho nasazení do jakékoliv jiné aplikace určené pro OS Android. Webová aplikace je vybudována na PHP frameworku Laravel a je zodpovědná za příjem dat od jednotlivých mobilních zařízení a za jejich následné zpracování. V rámci uživatelského rozhraní webové aplikace bylo vytvořeno pět dílčích panelů, jež obsahují vizualizaci výsledků zpracování dat dle požadavků Policie ČR.

Funkčnost realizovaného systému byla ověřována průběžně během jeho vývoje, a to jak z pohledu fungování systému jako celku, tak z pohledu fungování jeho dílčích částí. V rámci zmíněného projektu „Bezpečně v kyberprostoru“ je plánováno oficiální nasazení vyvinutého nástroje a jeho další rozvoj.

LITERATURA

[1] JIRÁSEK, P., NOVÁK, L., POŽÁR, J. *Výkladový slovník kybernetické bezpečnosti: Cyber security glossary.* 3. aktualiz. vyd. Praha: Policejní akademie ČR v Praze, 2015, xxviii. ISBN 978-80-7251-436-6.

[2] KOPECKÝ, K., KREJČÍ, V. Co je kyberšikana? *Portál E-Bezpečí* [online]. 22. Května 2009, Univerzita Palackého v Olomouci, Centrum PRVoK PdF, © 2008 - 2015 [cit. 2016-11-17]. Dostupné z URL: <http://www.e-bezpeci.cz/index.php/temata/kyberikana/17-cojekybersikana>.

[3] BURÝŠKOVÁ, L., Víte co je KYBERŠIKANA? *Policie České republiky – KŘP Královéhradeckého kraje* [online]. 11. 12. 2009, Policie ČR, © 2016 [cit. 2016-11-17]. Dostupné z URL: <http://www.policie.cz/clanek/vite-co-je-kybersikana.aspx>.

[4] KOPECKÝ, K., KREJČÍ, V. Nebezpečí internetové komunikace. *Portál E-Bezpečí* [online]. Olomouc, 2009 - 2010 [cit. 2016-11-18]. Dostupné z URL: <http://www.e-bezpeci.cz/index.php/ke-stazeni/doc_download/1-nebezpei-elektronicke-komunikace-1-2009-2010>.

[5] Co je to kyberšikana a jak se projevuje? *Bezpečně-online.cz* [online]. Národní centrum bezpečnějšího internetu [cit. 2016-11-18]. Dostupné z URL: <http://www.bezpecne-online.cz/pro-rodice-a-ucitele/teenageri-a-komunikace-na-internetu/co-je-to-kybersikana-a-jak-se-projevuje.html>.

[6] Kyberšikana a šikana. *Minimalizace šikany* [online]. [cit. 2016-11-18]. Dostupné z URL: <http://www.minimalizacesikany.cz/en/chci-se-dozvedet/fenomen-kybersikana/179-kybersikana-sikana>.

[7] Kyberšikana. *Nebuď oběť! Rizika internetu a komunikačních technologií* [online]. © 2010 - 2016 [cit. 2016-11-18]. Dostupné z URL: <http://www.nebudobet.cz/?cat=kybersikana>.

[8] KOPECKÝ, K., KREJČÍ, V. *Rizika virtuální komunikace: příručka pro učitele a rodiče.* Olomouc: NET UNIVERSITY, 2010. Registrační číslo: CZ.1.07/1.3.13/02.0040. ISBN 978-80-254-7866-O.

[9] WACHS, S., WOLF D., K., PAN, C. Cybergrooming: Risk factors, coping strategies and associations with cyberbullying. *In: Psicothema 2012* [online]. 2012, vol. 24, no. 4, pp. 628-633 [cit. 2016-11-19]. ISSN 0214-9915. Dostupné z URL: <http://www.psicothema.com/pdf/4064.pdF>.

[10] Kybergrooming. *Portál E-Bezpečí* [online]. 13. Září 2008, Univerzita Palackého v Olomouci, Centrum PRVoK PdF, ©2008 - 2015 [cit. 2016-11-19]. Dostupné z URL: <http://www.e-bezpeci.cz/index.php/temata/kybergrooming/125-42>.

[11] ČÍRTKOVÁ, L., Nebezpečné pronásledování. *Společnosti pro kriminalistiku* [online]. 8. 12. 2008, Společnost pro kriminalistiku, ©2015 [cit. 2016-11-19]. Dostupné z URL: <http://www.spkcz.cz/index.php?option=com_content&view=article&id=65:stalking&catid=34:lanky&Itemid=57>.

[12] PATHÉ, M., MULLEN E., P., PURCELL, R. Management of victims of stalking. *In: Advances in Psychiatric Treatment 2001* [online]. 2001, vol. 7, pp. 399-406 [cit. 2016-11-19]. Dostupné z URL: <http://apt.rcpsych.org/content/aptrcpsych/7/6/399.full.pdf>.

[13] JANOUŠKOVÁ, V., Rizika internetové komunikace. *Inflow: information journal* [online]. 6. 6. 2012, Inflow, ©2007 – 2013 [cit. 2016-11-19]. Dostupné z URL: <http://www.inflow.cz/rizika-internetove-komunikace>.

[14] POLIÁN, T., *Trestněprávní úprava nebezpečného pronásledování podle § 354 trestního zákoníku* [online]. Olomouc, 2013. Bakalářská práce. Masarykova univerzita, Právnická fakulta, Katedra trestního práva [cit. 2016-11-19]. Dostupné z URL: <https://is.muni.cz/th/388625/pravf_b/BC_PRACE-_Tomas_POLIAN.txt>.

[15] Android. *Open Handset Alliance* [online]. [cit. 2016-11-24]. Dostupné z URL: <http://www.openhandsetalliance.com/android_overview.html>.

[16] Smartphone OS Market Share, 2016 Q3. *IDC: International Data Corporation* [online]. IDC Research, Inc. ©2017 [cit. 2017-04-27]. Dostupné z URL: <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>.

[17] Mobile Operating System Market Share in Czech Republic 2016. *StatCounter Global Stats* [online]. StatCounter. ©1997-2017 [cit. 2017-04-27]. Dostupné z URL: <http://gs.statcounter.com/os-market-share/mobile/czech-republic/2016>.

[18] Market share of mobile operating systems in Czech Republic from 2010 to 2016. *Statista: The Statistics Portal* [online]. Statista Inc. [cit. 2017-04-27]. Dostupné z URL: <https://www.statista.com/statistics/669587/market-share-mobile-operating-systems-czech-republic/>.

[19] Dashboards. *Android Developers* [online]. Android Open Source Project [cit. 2016-11-24]. Dostupné z URL: <http://developer.android.com/about/dashboards/index.html>.

[20] Android Architecture. *Tutorialspoint.com* [online]. Tutorials Point India Private Limited, ©2016 [cit. 2016-11-28]. Dostupné z URL: <http://www.tutorialspoint.com/android/android_architecture.htm>.

[21] Android (SDK). *Open Websites: Collaborative Bluetooth EduMANET* [online]. The University of Texas at Austin, College of Education [cit. 2016-11-28]. Dostupné z URL: <https://ows.edb.utexas.edu/site/collaborative-bluetooth-edumanet/android-sdk-2>.

[22] Application Fundamentals. *Android Developers* [online]. Android Open Source Project [cit. 2016-11-29]. Dostupné z URL: <http://developer.android.com/guide/components/fundamentals.html>.

[23] Android Application Components. *Tutorialspoint.com* [online]. Tutorials Point India Private Limited, ©2016 [cit. 2016-11-28]. Dostupné z URL: <http://www.tutorialspoint.com/android/android_application_components.htm>.

[24] Starting an Activity. *Android Developers* [online]. Android Open Source Project [cit. 2016-11-30]. Dostupné z URL: <http://developer.android.com/training/basics/activity-lifecycle/starting.html>.

[25] Android Activities. *Tutorialspoint.com* [online]. Tutorials Point India Private Limited, ©2016 [cit. 2016-11-30]. Dostupné z URL: <http://www.tutorialspoint.com/android/android_acitivities.htm>.

[26] ActivityManager. *Android Developers* [online]. Android Open Source Project [cit. 2016-12-08]. Dostupné z URL: <http://developer.android.com/reference/android/app/ActivityManager.html>.

[27] What is API Level? *Android Developers* [online]. Android Open Source Project [cit. 2016-12-08]. Dostupné z URL: <http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels>.

[28] getRunningTasks. *Android Developers* [online]. Android Open Source Project [cit. 2016-12-08]. Dostupné z URL: <http://developer.android.com/reference/android/app/ActivityManager.html#getRunningTasks%28int%29>.

[29] android.app.usage. *Android Developers* [online]. Android Open Source Project [cit. 2016-12-08]. Dostupné z URL: <http://developer.android.com/reference/android/app/usage/package-summary.html>.

[30] Android 5.0 APIs. *Android Developers* [online]. Android Open Source Project [cit. 2016-12-08]. Dostupné z URL: <https://developer.android.com/about/versions/android-5.0.html#System>.

[31] UsageStatsManager. *Android Developers* [online]. Android Open Source Project [cit. 2016-12-09]. Dostupné z URL: <https://developer.android.com/reference/android/app/usage/UsageStatsManager.html>.

[32] Docs : missing information about UsageStatsManager class. *Google Issue Tracker* [online]. [cit. 2016-12-09]. Dostupné z URL: <https://issuetracker.google.com/issues/37007357>.

[33] UsageStatsManager returning data from just last day/week/month/year? *Stack Overflow* [online]. Stack Exchange [cit. 2016-12-09]. Dostupné z URL: <http://stackoverflow.com/questions/26882956/usagestatsmanager-returning-data-from-just-last-day-week-month-year>.

[34] Package Index. *Android Developers* [online]. Android Open Source Project [cit. 2017-04-04]. Dostupné z URL: <https://developer.android.com/reference/packages.html>.

[35] Getting Started. *Android Developers* [online]. Android Open Source Project [cit. 2017-04-04]. Dostupné z URL: <https://developer.android.com/training/index.html>.

[36] Introduction to Android. *Android Developers* [online]. Android Open Source Project [cit. 2017-04-04]. Dostupné z URL: <https://developer.android.com/guide/index.html>.

[37] AlarmManager. *Android Developers* [online]. Android Open Source Project [cit. 2017-04-04]. Dostupné z URL: <https://developer.android.com/reference/android/app/AlarmManager.html>.

[38] Scheduling Repeating Alarms. *Android Developers* [online]. Android Open Source Project [cit. 2017-04-04]. Dostupné z URL: <https://developer.android.com/training/scheduling/alarms.html>.

[39] JobScheduler. *Android Developers* [online]. Android Open Source Project [cit. 2017-04-04]. Dostupné z URL: <https://developer.android.com/reference/android/app/job/JobScheduler.html>.

[40] Intelligent Job-Scheduling. *Android Developers* [online]. Android Open Source Project [cit. 2017-04-04]. Dostupné z URL: <https://developer.android.com/topic/performance/scheduling.html>.

[41] android.database.sqlite. *Android Developers* [online]. Android Open Source Project [cit. 2017-04-05]. Dostupné z URL: <https://developer.android.com/reference/android/database/sqlite/package-summary.html>.

[42] SharedPreferences. *Android Developers* [online]. Android Open Source Project [cit. 2017-04-04]. Dostupné z URL: <https://developer.android.com/reference/android/content/SharedPreferences.html>.

[43] Android Material Intro Screen. *GitHub* [online]. [cit. 2017-04-20]. Dostupné z URL: <https://github.com/TangoAgency/material-intro-screen>.

[44] *Pixabay* [online]. 2017 [cit. 2017-04-20]. Dostupné z URL: <https://pixabay.com>.

[45] Material Design. *Material Design Guidelines* [online]. [cit. 2017-04-20]. Dostupné z URL: <https://material.io/guidelines/>.

[46] OTWELL, T. *Laravel - The PHP Framework For Web Artisans* [online]. [cit. 2017-04-28]. Dostupné z URL: <https://laravel.com/>.

[47] Architecture of Laravel Applications. *Laravel book* [online]. [cit. 2017-04-28]. Dostupné z URL: <http://laravelbook.com/laravel-architecture/>.

[48] ALMSAEED, A. AdminLTE. *GitHub* [online]. [cit. 2017-05-05]. Dostupné z URL: <https://github.com/almasaeed2010/AdminLTE>.

[49] OTWELL, T. Directory structure. *Laravel - The PHP Framework For Web Artisans* [online]. [cit. 2017-04-28]. Dostupné z URL: <https://laravel.com/docs/5.4/structure>.

[50] KUMAR, S. Database Manager For Android. *GitHub* [online]. [cit. 2017-05-05]. Dostupné z URL: <https://github.com/sanathp/DatabaseManager_For_Android>.

[51] Stetho. *Facebook GitHub* [online]. [cit. 2017-05-05]. Dostupné z URL: <http://facebook.github.io/stetho/>.

[52] What is Instance ID? *Google Developers* [online]. [cit. 2017-05-10]. Dostupné z URL: <https://developers.google.com/instance-id/>.

[53] Protecting against Security Threats with SafetyNet. *Android Developers* [online]. Android Open Source Project [cit. 2017-05-10]. Dostupné z URL: <`https://developer.android.com/training/safetynet/index.html`>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

API	Application Programming Interface
CCO	Creative Commons
CPU	Central Processing Unit
DVD	Digital Versatile Disc
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secured
IP	Internet Protocol
JSON	JavaScript Object Notation
MPEG	Moving Picture Experts Group
MVC	Model View Controller
OS	Operating System
PHP	Hypertext Preprocessor
SMS	Short Message Service
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TOR	The Onion Router
UI	User Interface
URI	Uniform Resource Identifier
UTC	Coordinated Universal Time
UUID	Universally Unique IDentifier

SEZNAM PŘÍLOH

A Testování API `android.app.usage` **97**

B Manifest Android aplikace **98**

C Diagramy tříd Android aplikace **100**

D Konfigurační soubor .env **103**

E Uživatelské rozhraní web. aplikace **104**

E.1 Administrativní panel 104

E.2 Panel vývojových trendů 105

E.3 Panely top aplikací 106

F Obsah přiloženého CD **108**

A TESTOVÁNÍ API `android.app.usage`

Výpis A.1: Zjednodušená ukázka kódu stěžejní části testovací aplikace pro ověření funkčnosti balíčku `android.app.usage`

```
// Možnost flexibilního určování časového rozsahu
Calendar beginCalendar = Calendar.getInstance();
beginCalendar.add(Calendar.WEEK, -3);
Calendar endCalendar = Calendar.getInstance();
endCalendar.add(Calendar.WEEK, -1);

// Převedení počátečního a koncového čas. rozsahu na milisekundy
long beginTime = beginCalendar.getTimeInMillis();
long endTime = endCalendar.getTimeInMillis();

// Volba časového intervalu (denní/týdenní/měsíční/roční)
int selectedInterval = UsageStatsManager.INTERVAL_WEEKLY

// Získání přístupu k balíčku, zavolání metody a uložení kolekce se
    statistikami
UsageStatsManager usageManager = (UsageStatsManager) context.
   getSystemService(Context.USAGE_STATS_SERVICE);
List<UsageStats> list = usageManager.queryUsageStats(
   selectedInterval, beginTime, endTime);

// Výpis jednotlivých položek kolekce do konzole (název aplikace +
   aktivní doba běhu)
for (int i = 0; i < list.size(); i++){
Log.i("Item n." + i, list.get(i).getPackageName() + ":" + list.get(
   i).getTotalTimeInForeground());
}
```

B MANIFEST ANDROID APLIKACE

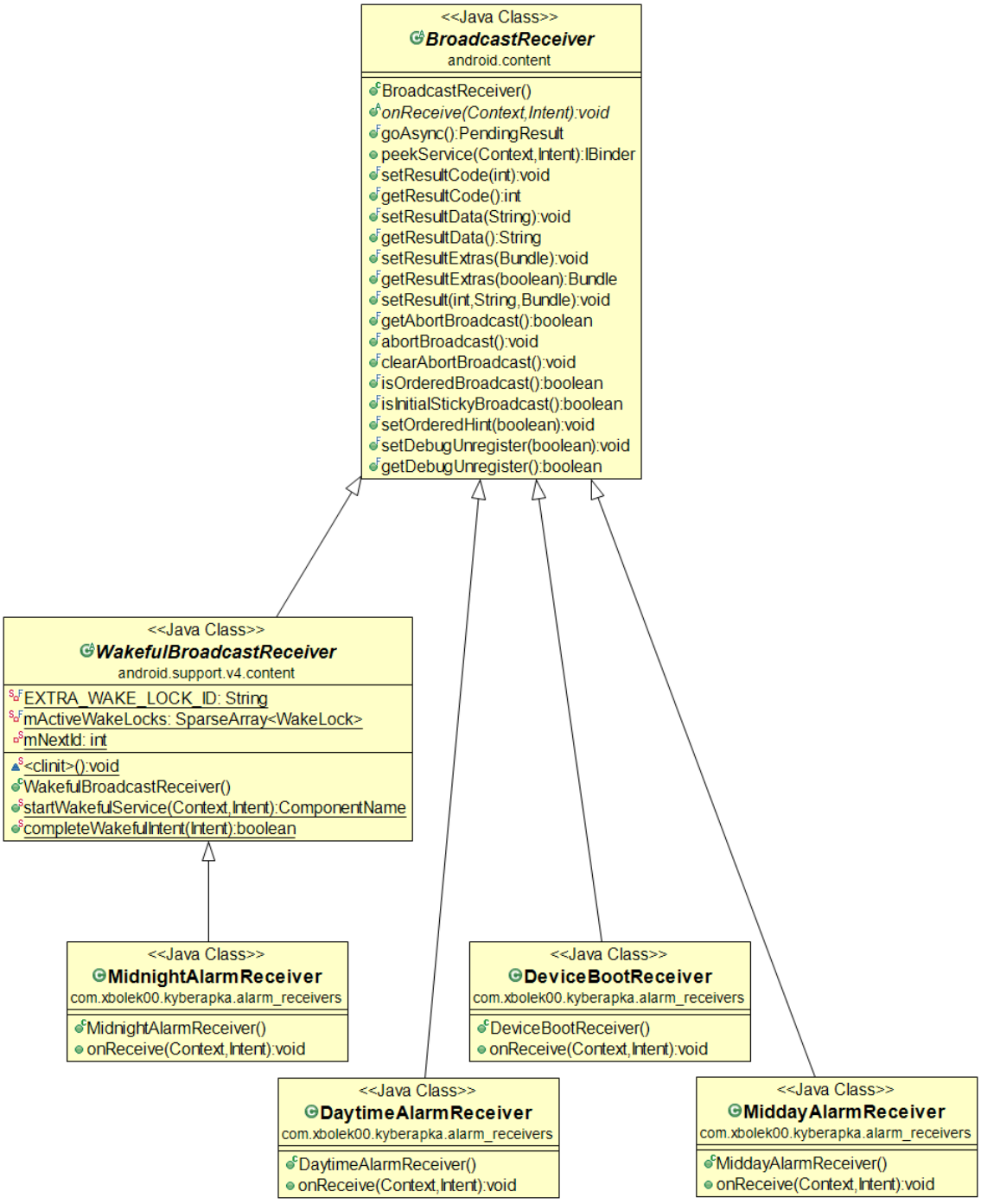
Výpis B.1: Soubor `AndroidManifest.xml` projektu Android aplikace

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.xbolek00.kyberapka">

    <uses-permission android:name="android.permission.PACKAGE_USAGE_STATS"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.permission.WAKE_LOCK" />

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity
            android:name=".SplashActivity"
            android:theme="@style/SplashTheme">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".ui_intro.IntroActivity"
            android:theme="@style/Theme.Intro">
        </activity>

        <activity
            android:name=".ui_main.MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">
        </activity>

        <receiver android:name=".alarm_receivers.DeviceBootReceiver"
            android:enabled="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"></
                    action>
            </intent-filter>
        </receiver>

        <receiver android:name=".alarm_receivers.MidnightAlarmReceiver">
        </receiver>

        <receiver android:name=".alarm_receivers.MiddayAlarmReceiver">
        </receiver>

        <receiver android:name=".alarm_receivers.DaytimeAlarmReceiver">
        </receiver>

```

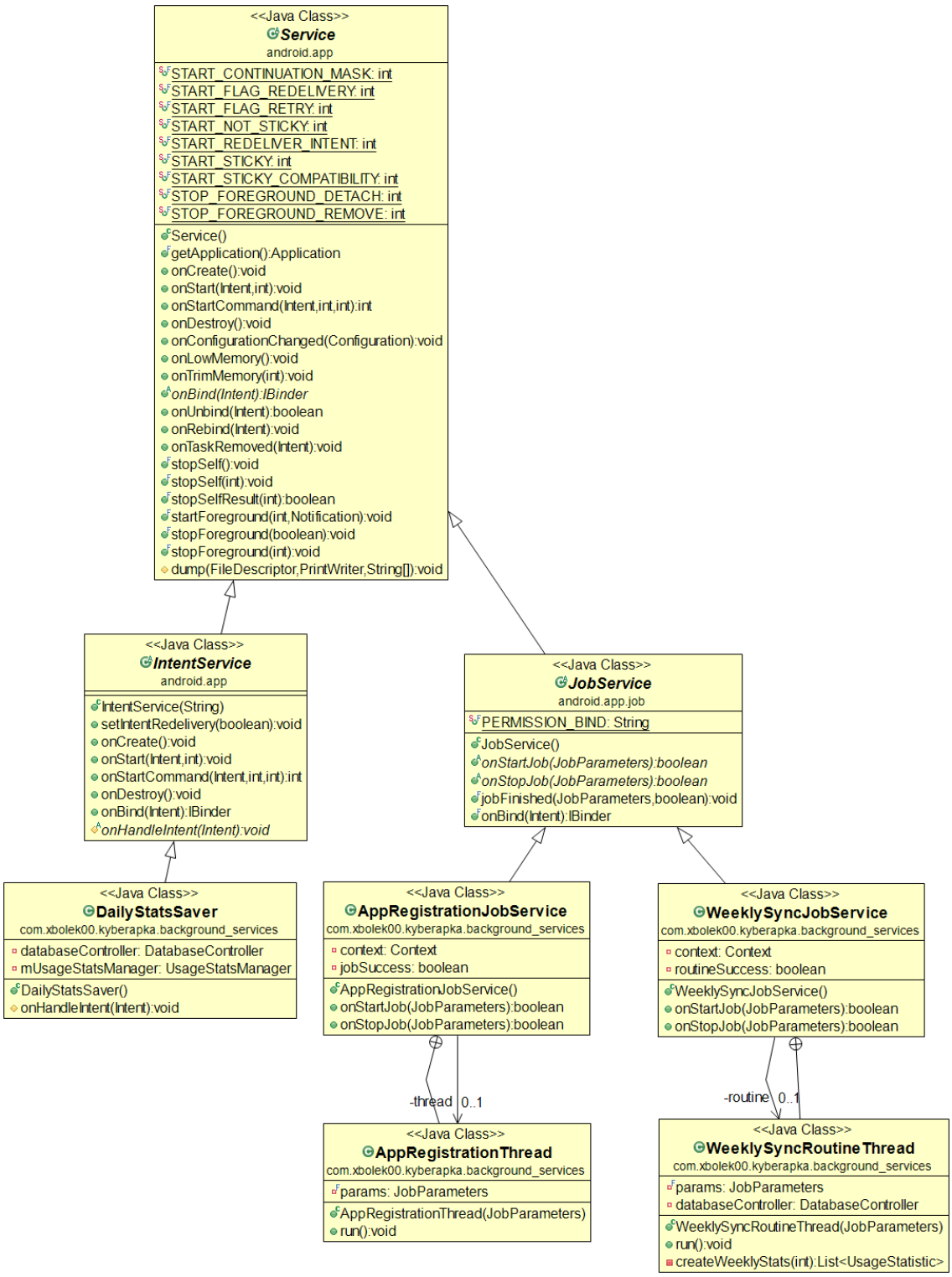
```
        <service android:name=".background_services.DailyStatsSaver">
        </service>

        <service android:name=".background_services.WeeklySyncJobService"
            android:permission="android.permission.BIND_JOB_SERVICE">
        </service>

        <service android:name=".background_services.AppRegistrationJobService"
            android:permission="android.permission.BIND_JOB_SERVICE">
        </service>

    </application>
</manifest>
}
```

C DIAGRAMY TŘÍD ANDROID APLIKACE

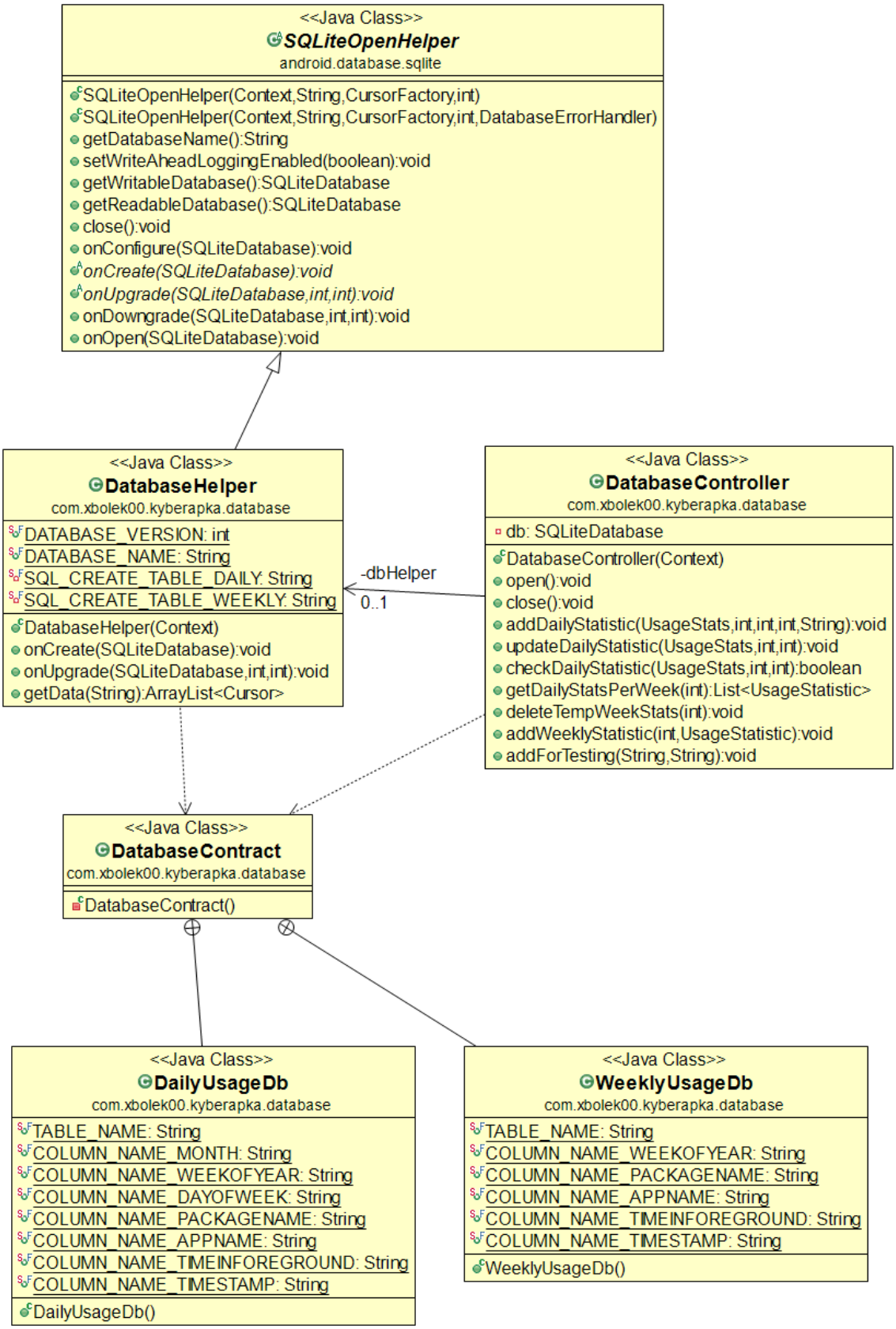
Vybrané diagramy tříd vnitřního mechanismu Android aplikace.



Obr. C.1: Diagram dědičnosti tříd příjemců vysílání – balíček `alarm_receivers`



Obr. C.2: Diagram dědičnosti tříd služeb – balíček `background_services`



Obr. C.3: Diagram tříd – balíček `database`

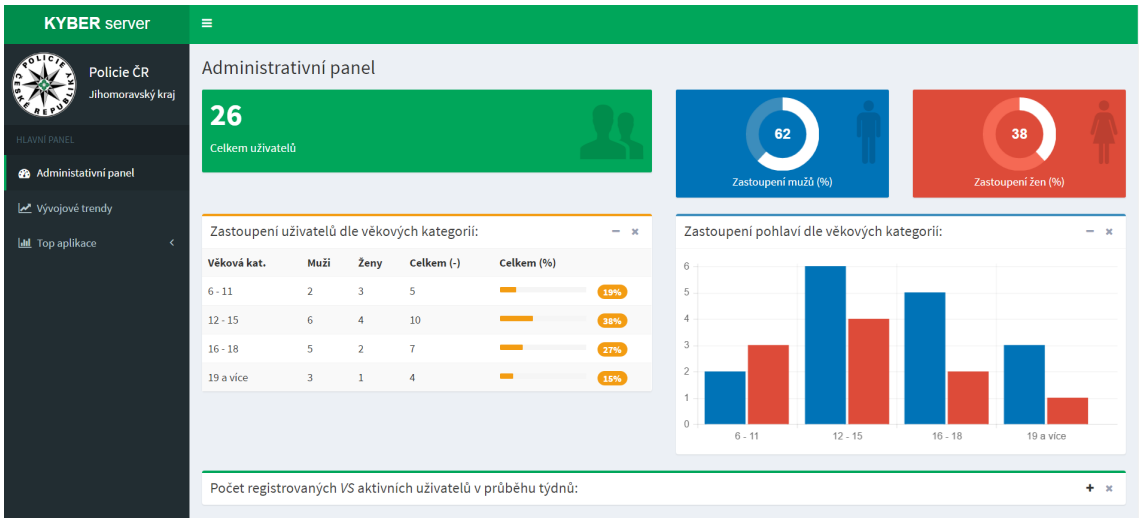
D KONFIGURAČNÍ SOUBOR .ENV

Výpis D.1: Ukázka nastavení konfiguračního souboru `.env` projektu webové aplikace

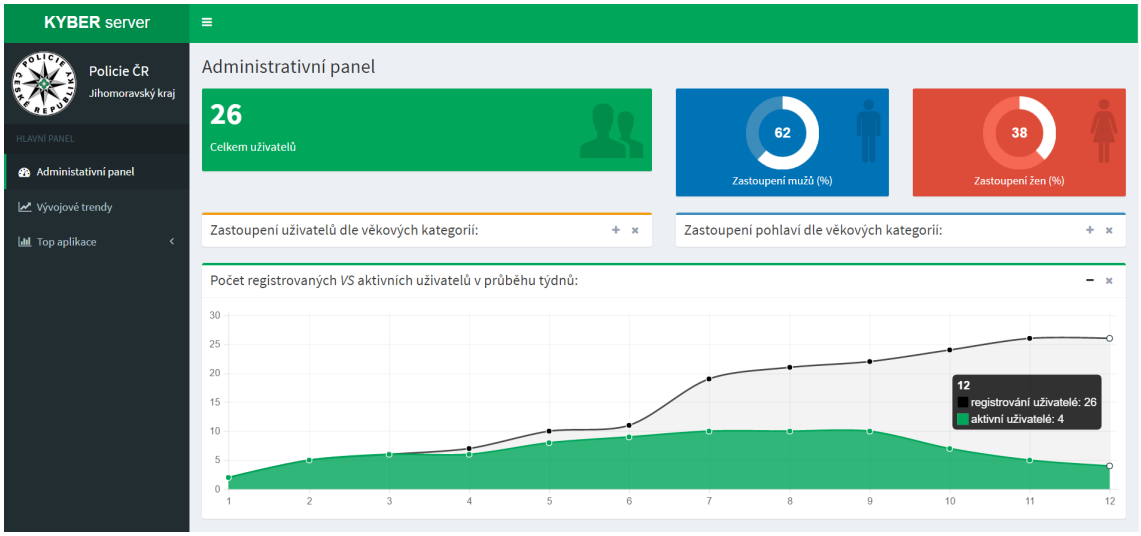
```
APP_ENV=local
APP_KEY=base64:bFdcF7Sd1dgkDxH2H26GcFpYJUBJM3Vhtw1LJUGc/4M=
APP_DEBUG=false
APP_LOG_LEVEL=debug
APP_URL=http://localhost

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=kyberdb
DB_USERNAME=root
DB_PASSWORD=kyber_server2017

BROADCAST_DRIVER=log
CACHE_DRIVER=file
SESSION_DRIVER=file
QUEUE_DRIVER=sync

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_DRIVER=smtp
MAIL_HOST=mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null

PUSHER_APP_ID=
PUSHER_APP_KEY=
PUSHER_APP_SECRET=
}
```

E UŽIVATELSKÉ ROZHRANÍ WEB. APLIKACE

E.1 Administrativní panel

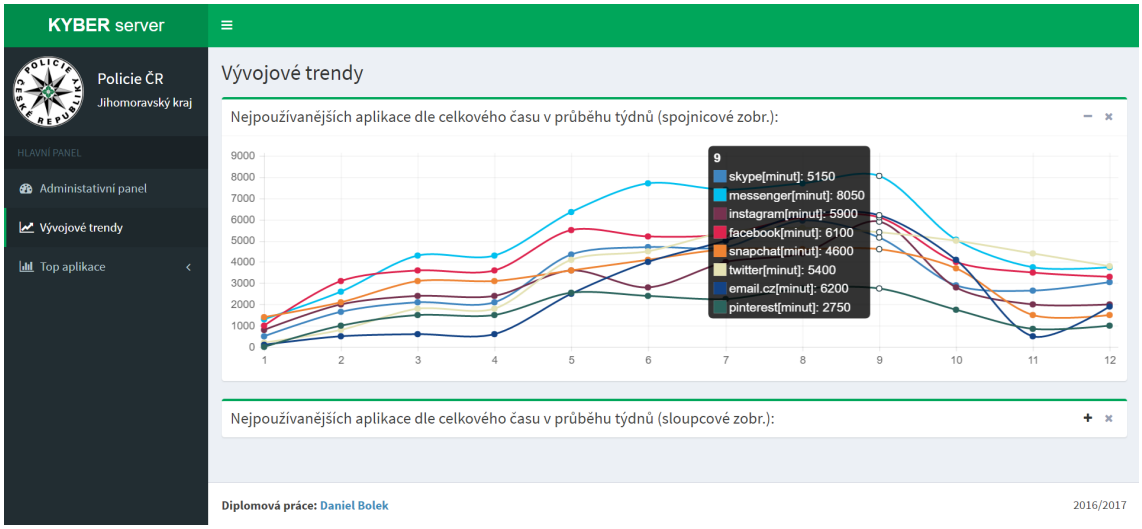


Obr. E.1: Administrativní panel – pohled č. 1

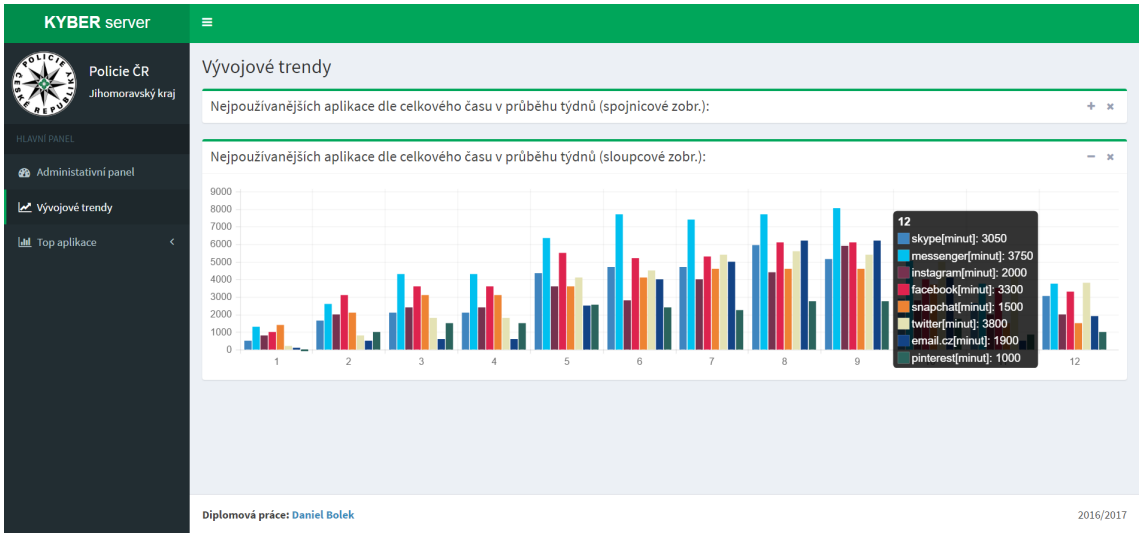


Obr. E.2: Administrativní panel – pohled č. 2

E.2 Panel vývojových trendů

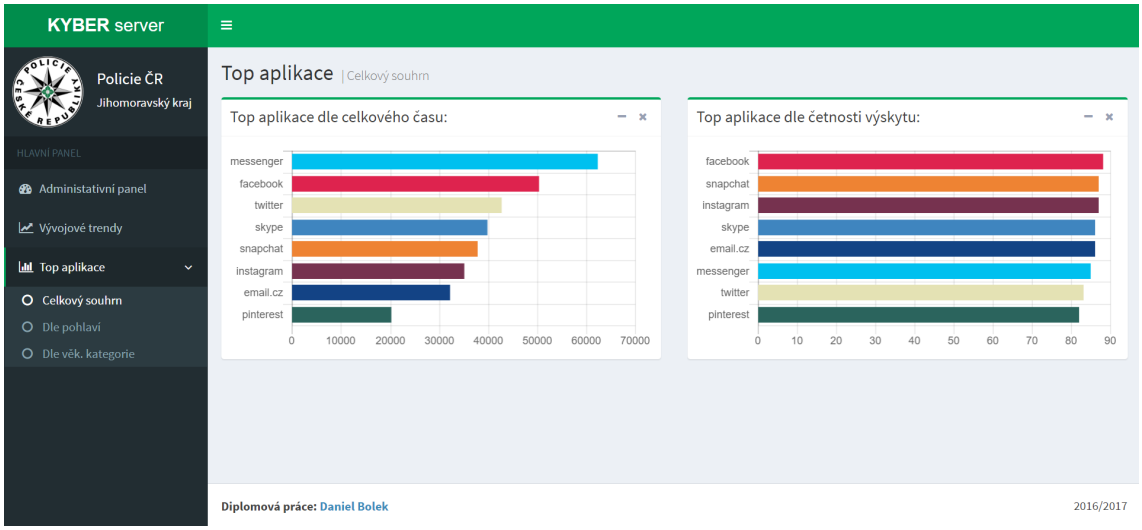


Obr. E.3: Vývojové trendy – pohled č. 1

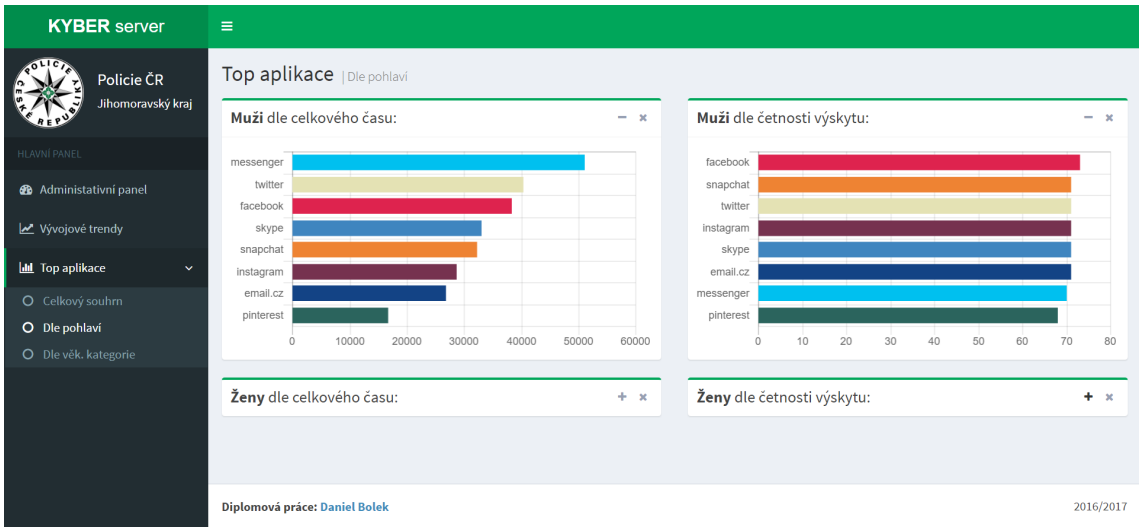


Obr. E.4: Vývojové trendy – pohled č. 2

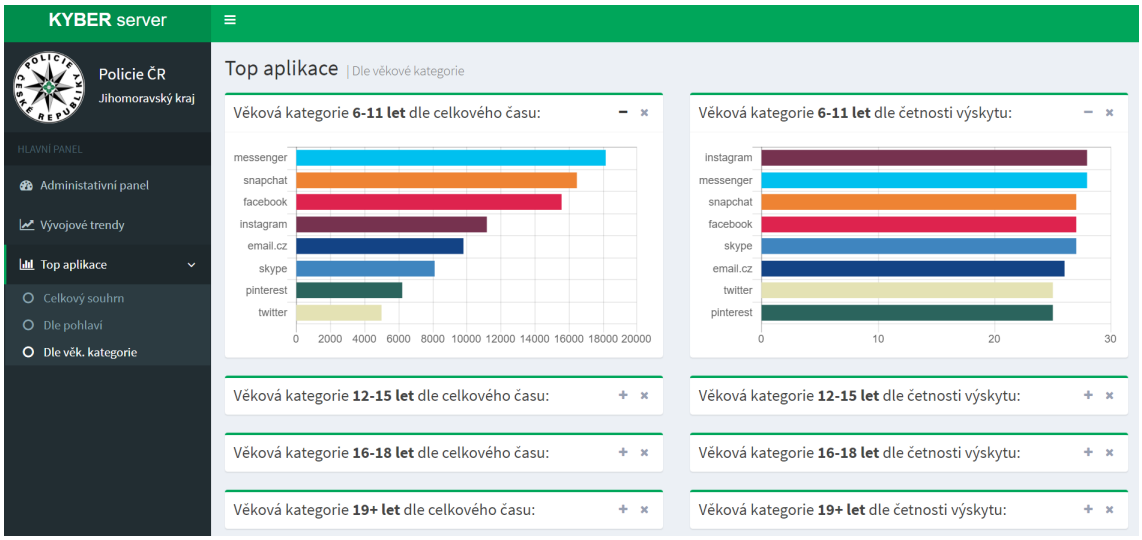
E.3 Panely top aplikací



Obr. E.5: Top aplikace – celkový souhrn



Obr. E.6: Top aplikace – dle pohlaví



Obr. E.7: Top aplikace – dle věkové kategorie

F OBSAH PŘILOŽENÉHO CD

Na přiloženém médiu se nachází:

- projekt Android aplikace vytvořený v rámci Android Studia,
- projekt webové aplikace se standartní strukturou Laravel projektu (včetně již vytvořeného adresáře `vendor` a konfiguračního souboru `.env`),
- rozšířená verze testovací MySQL databáze určena pro demonstraci uživatelského rozhraní webové aplikace,
- elektronická verze diplomové práce ve formátu PDF.

Výpis adresářové struktury přiloženého média:

```
/ ..... kořenový adresář přiloženého CD
├── KYBER_apka ..... projekt Android aplikace
├── KYBER_server ..... projekt webové aplikace
├── kyberdb.zip ..... demonstrační MySQL databáze
└── Bezpecnostni_hrozby_internetu.pdf ..... diplomová práce ve formátu PDF
```