VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

MONITOROVÁNÍ DAT NA SERVERU A NÁSLEDNÁ NOTIFIKACE PRO SYSTÉM ANDROID

SERVER DATA MONITORING WITH ANDROID NOTIFICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MICHAL BOHUŠ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MARTIN HOLKOVIČ**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20982/2017/xbohus01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Bohuš Michal**
Obor: Informační technologie
Téma: **Monitorování dat na serveru a následná notifikace pro systém Android**
Server Data Monitoring with Android Notification
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s platformou Android a zaměřte se na možnosti přijímaní notifikací oznamující výskyt nové události.
2. Identifikujte zdroje dat na serveru, které má smysl monitorovat a následně při splnění předem definovaných podmínek notifikovat uživatele.
3. Navrhněte a implementujte notifikační systém, který bude na serveru sledovat zdroje dat a na základě konfigurace generovat události.
4. Pro systém Android navrhněte a vytvořte aplikaci, která bude přijímat vygenerované události s cílem je zobrazit na mobilním zařízení (telefon, hodinky).
5. Otestujte všechny vytvořené aplikace z hlediska použitelnosti a navrhněte možná rozšíření.

Literatura:

- Gargenta, Marko. "Learning Android: Biulding Applications for the Android Market. Sebstopol." (2011).
- Kofler, Michael. "Mistrovství v MySQL 5: kompletní průvodce webového vývojáře." Petr Zetocha 99 (2007).
- Holubová, Irena, et al. Big Data a NoSQL databáze. Grada, 2015.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3 ze zadaní.

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

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

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

Vedoucí: **Holkovič Martin, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

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

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Práca sa zaoberá monitorovaním dát, generovaním správ a vytvorením mobilnej aplikácie pre odoberanie notifikácii zo servera. Monitorovanie užívateľom zvolených dát prebieha na serveri. Pre účely sledovania hodnôt boli ako zdroj vybrané NoSQL databázy, SQL databáza a terminál v rámci OS. Na základe užívateľom stanovených pravidiel nad monitorovanými dátami sa generujú správy. Tie môžu byť odoslané ako jednotlivcovi, tak skupine ľudí na e-mail alebo do mobilnej aplikácie. Prostredníctvom mobilnej aplikácie je možné sa prihlásiť alebo odhlásiť z odberu správ a zobrazovať prijatú správu.

Abstract

Thesis is concerned with server data monitoring, messages generation and mobile application development for receiving notifications from server. Monitoring of the user's selected data takes place on the server. For data monitoring purposes were as source chosen NoSQL databases, SQL database and terminal within OS. Messages are generated based on user's specified rules over the monitored data. They can be sent both to an individual or group of people to e-mail or mobile application. Through the mobile application it is possible to subscribe or unsubscribe to receive messages and view recieved message.

Kľúčové slová

Monitorovanie dát, Notifikácie, Android, Google Firebase, SQL, Big Data, Docker

Keywords

Data monitoring, Notifications, Android, Google Firebase, SQL, Big Data, Docker

Citácia

BOHUŠ, Michal. *Monitorování dat na serveru a následná notifikace pro systém Android.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Holkovič

Monitorování dat na serveru a následná notifikace pro systém Android

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing.Martina Holkoviča. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Michal Bohuš
17. mája 2018

Poďakovanie

Ďakujem pánovi inžinierovi Martinovi Holkovičovi za odborné usmernenia, cenné rady a vedenie práce.

Obsah

Kapitola 1

Úvod

Žijeme v rýchlej digitálnej dobe a je potrebné s ňou držať krok. Preto potrebujeme aktuálne informácie čo najrýchlejšie. Tie sa k nám musia nejakým spôsobom dostať. V poslednej dobe je týmto sprostredkovateľom smartfón. Ľudia používajú smartfóny po celom svete. Predpoveď do roku 2020 je, že bude 5 miliárd[1] používateľov smartfónov.

K prvému mesiacu roka 2018 je podiel mobilných operačných systémov na trhu v stave, že najväčšie zastúpenie má Android[2] s 68.69%. Druhú priečku drží iOS od firmy Apple s 29.26%. Zvyšok tvoria ostatné, nie tak úspešné mobilné operačné systémy, ktorých vývoj už prevažne skončil. Android má svoje prvenstvo vďaka dostupnosti, kde už za pár eur sme schopný si zaobstarať smartfón s dostatočným výkonom. Na informovanie o aktúalnej udalosti sa v smartfónoch využívajú notifikácie. Notifikácia je väčšinou krátka správa s podstatnou informáciou obdržaná alebo generovaná mobilnou aplikáciou.

Administrátori sietí, vedeckí pracovníci a mnoho ďalších sa stretávajú so situáciami, keď nevedia vopred o udalosti kedy nastane. Napríklad čakajú, kým sa pridá výsledok zdľhavého výpočtu do databázy, náhle sa ukončí služba na severi, prekročí sa limit používania serverovej služby. O týchto udalostiach chceme byť informovaný a včas na ne reagovať. Vhodnou formou upozornenia na daný stav sú už spomenuté notifikácie v smartfónoch.

Jedným z cieľov práce je vytvoriť program na monitorovanie dát na serveri podľa požiadaviek užívateľa. Ďalším cieľom je informovať užívateľa správou o monitorovaných dátach. Správa bude určená pre e-mail alebo mobilnú aplikáciu, ktorej vznik je účelom tejto práce.

Prvotne je potrebné sa zoznámiť s problematikou a preto sa v 2. kapitole rozoberá téma platformy Android a Google Firebase. Pre monitorovanie je potrebné identifikovať vhodný zdroj dát. Týmto informáciám je venovaná kapitola 3. Následne v kapitole 4 je rozobratý popis návrhu serverovej časti a mobilnej Android aplikácie. Ďalej sa práca zaoberá konkrétnou implementáciou podľa návrhu v kapitole 5. Nakoniec je potrebné hotovú implementáciu otestovať, čomu je venovaná 6. kapitola.

[1] Prognóza počtu zariadení https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] Podiel mobilných operačných systémov na trhu https://goo.gl/Ljy1s3/

Kapitola 2

Platforma Android a Google Firebase

Kapitola sa venuje platforme Android, pretože mobilná aplikácia bude určená práve pre ňu. Ako prvé je uvedený základný prehľad o Androide a určovanie jeho verzií v sekcii 2.1. Potom v sekcii 2.2 je popísaná architektúra systému Android po jednotlivých vrstvách. Hlavnými komponentami Android aplikácií ako je Activity, Services, atď. sa zaoberá sekcia 2.3. Následne sa spomína mobilná platforma Google Firebase 2.4, ktorá prináša množstvo produktov na analýzu a vývoj aplikácií pre mobilné zariadenia iOS, Android a internetové aplikácie. Pri Google Firebase sa zameriame na jej produkty cloud messaging a realtime databázu.

2.1 Prehľad o Androide

Android je open source platforma navrhnutá primárne pre mobilné zariadenia. Dnes je využívaná aj v zariadeniach ako sú televízory, hodinky alebo autá. Je presadzovaná spoločnosťou Google a vlastní ho Open Handset Alliance, čo je konzorcium 84 firiem, ktoré stoja za vývojom otvorených štandardov pre mobilné zariadenia. Táto platforma dovoľuje oddeliť hardvér od softvéru, ktorý na ňom beží. Vďaka tomu môže obrovský počet zariadení spúšťať tie isté aplikácie a vytvára bohatý ekosystém pre vývojárov a spotrebiteľov [3].

Version	Codename	API	Distribution	Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.4%	5.0	Lollipop	21	5.7%
				5.1		22	19.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.5%	6.0	Marshmallow	23	28.6%
4.1.x	Jelly Bean	16	1.9%	7.0	Nougat	24	21.1%
4.2.x		17	2.9%	7.1		25	5.2%
4.3		18	0.8%	8.0	Oreo	26	0.5%
4.4	KitKat	19	12.8%	8.1		27	0.2%

Obr. 2.1: Percentuálne zastúpenie používania verzií Android ku dňu 8.1.2018 [1]

[1] https://developer.android.com/about/dashboards/index.html

Verzia Androidu sa určuje číslom, názvom a API úrovňou. Zaujímavosťou je, že každá hlavná verzia je pomenovaná po niečom sladkom. Napríklad verzia 4.4 sa nazývala po známej čokoládovej tyčinke KitKat. Pri vydaní novej verzie je snaha o zachovanie spätnej kompatibility[8]. To znamená, že ak už máme aplikáciu a vyjde nová verzia systému, nie je potrebné v kóde urobiť žiadne zmeny alebo len minimálne, aby aplikácia správne fungovala na novej verzii. Pred vývojom aplikácie sa určí do akej verzie Android bude aplikácia kompatibilná. Na základe toho sa budú používať triedy z Application Framework vrstvy (spomenuté v sekcii 2.2). Podľa obrázku 2.1 vieme určiť, že 80% zariadení používa verziu systému Lollipop a vyššie. Systém Lollipop bol všeobecne dostupný koncom roka 2014.

2.2 Architektúra systému Android

Na obrázku 2.2 je vidieť architektúru systému Android. Tvorí ju päť vrstiev[2, 9]. Najspodnejšia, prvá vrstva je linuxové jadro. V závisloti od zariadenia jadro Androidu vychádza z verzie linuxu 3.18 alebo 4.4[2] (údaj z roku 2017). Jadro rieši napríklad správu napájania batérie, threading, správu pamäte na nízkej úrovni. Obsahuje ovládače od dodávateľov hardvérových komponént ako napríklad zvuk, Wi-fi, kamera.

Druhú úroveň tvorí hardvérová abstrakčná vrstva (HAL). Umožňuje implementovať vlastné ovládače a špecifikácie zariadenia. Je to štandartná metóda na vytvorenie spojenia medzi hardvérom a architektúrou Android. Taktiež nám vytvára rozhranie medzi hardvérovými komponentami zariadenia a vyššou vrstvou Java API Framework. HAL pozostáva z viacerých knižníc. Každá knižnica implementuje rozhranie pre hardvérovú súčiastku ako gyroskop alebo teplotný senzor.

Na ďalšej, tretej úrovni nad HAL sú natívne knižnice. Zabezpečujú prepojenie medzi operačným systémom a aplikáciami. Nachádza sa tu napríklad SQLite databáza pre ukladanie údajov z aplikácie, WebKit, ktorý je open-source engine pre prehliadače, SSL knižnice, OpenGL pre grafiku atď.

Na tretej úrovni je taktiež prítomný Android Runtime. Obsahuje virtuálny stroj pre preklad a optimalizáciu Java byte kódu a základné Java knižnice. Na mobilných zariadeniach do verzie Androidu 5.0 bol používaný virtuálny stroj DVM (Dalvik virtual machine). Zabezpečoval funkciu virtuálneho stroja Java (JVM) špeciálne určenú pre Android. Android 4.4 predstavil experimentálne vylepšenie ART (Android Runtime). Bolo možné si vybrať medzi ART a DVM manuálne. S verziou 5.0 sa stalo ART jedinou runtime možnosťou. Rozdiel je v kompilácii, ktorú využívajú. DVM používa JIT[3] (just in time) a ART AOT (ahead of time). JIT si preloží java triedy do strojového kódu, keď to potrebuje a AOT si ich po prvom preložení uloží. Tým sa ušetrí procesorový čas za cenu zabratia väčšieho miesta v pamäti.

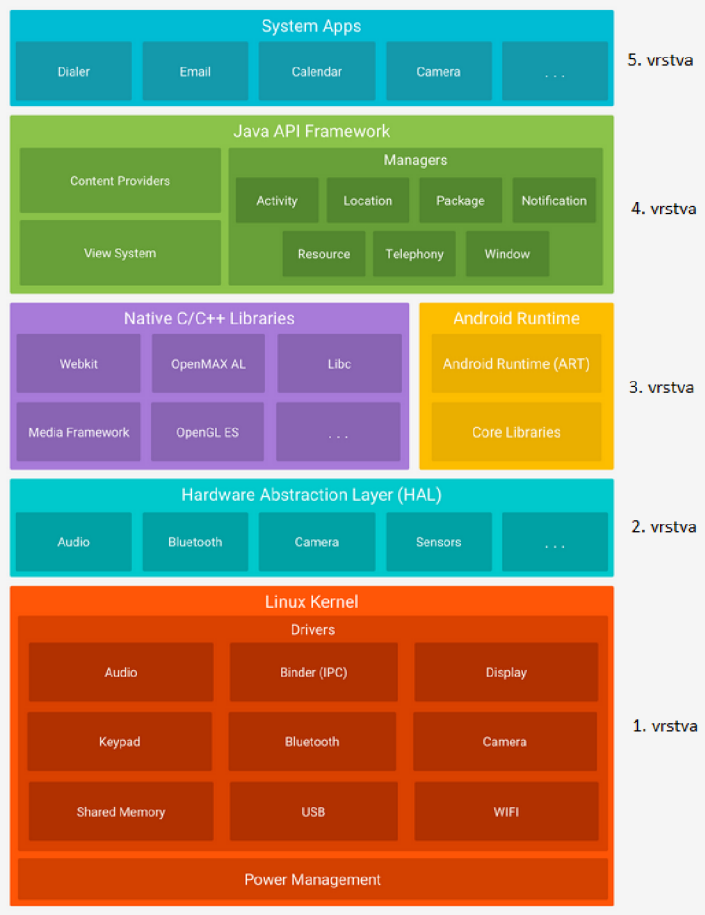
Štvrtú úroveň tvorí Java API Framework vrsta, ktorá ponúka vyššie služby aplikáciám v podobe Java tried. Sú to napríklad Activity Manager pre správu životného cyklu Activity (2.3.1), Content Provider, ktorý slúži na zdieľanie dát s inými aplikáciami, Notification Manager dovoľuje zobrazovať aplikáciám upozornenie a ďalšie.

Na najvyššej, piatej vrstve sú Android aplikácie. Je to najvyššia úroveň abstrakcie, ktorá využíva všetky nižšie vrsty. Koncový užívateľ nemusí vedieť čo všetko sa za tým skrýva, aby

[2]Verzia linuxu jadra https://arstechnica.com/gadgets/2017/05/ars-talks-android-googlers-chat-about-project-treble-os-updates-and-linux/

[3]Porovnanie JIT a ART https://android.jlelse.eu/closer-look-at-android-runtime-dvm-vs-art-1dc5240c3924

mohol aplikáciu používať. Vývojári by sa mali postarať, aby ovládanie ich aplikácie bolo intuitívne.



Obr. 2.2: Architektúra systému Android rozdelená do piatich vrstiev[4]

2.3 Hlavné komponenty Android aplikácie

Hlavné komponenty sú stavebné bloky, ktoré využíva vývojár na vytvorenie aplikácie [8]. Vymieňajú si medzi sebou informácie. Napríklad jednou z komponént je Activity (2.3.1). Tá nevie o dátach a metódach ďalšej Activity a ak jej dáta chce dostať, použije Intent[5](jedna z ďalších komponént). Niektoré komponenty sú na sebe závislé. Broadcast receiver (2.3.4) nedostane informácie pokiaľ Broadcast intent (2.3.4) na ktorý je prihlásený nevysiela.

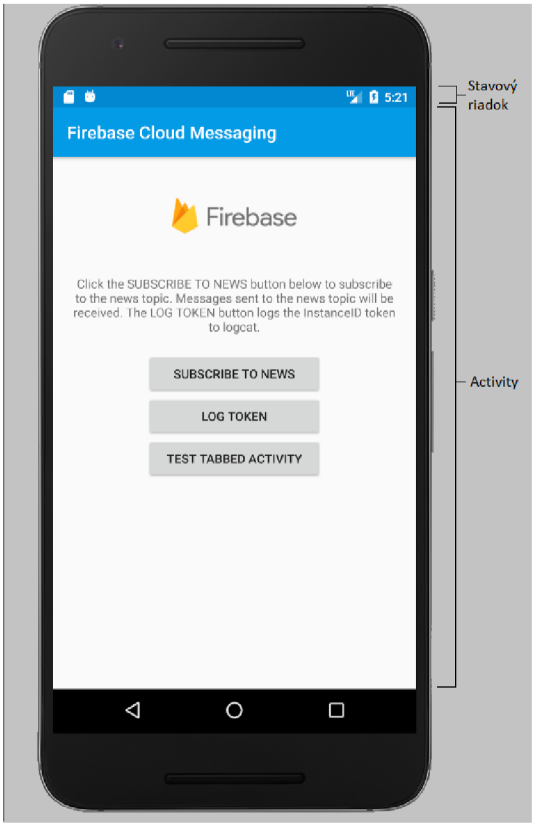
Každá aplikácia musí mať súbor s názvom *AndroidManifest.xml* v koreňovom adresári, ktorý poskytuje Android systému informácie o aplikácii. Predtým ako Android systém môže vykonať kód aplikácie musí mať informácie z Manifest súboru. V súbore Manifest je popísané, ktoré komponenty bude aplikácia používať, za akých podmienok majú byť spustené. Ďalej označuje Java package názov pre aplikáciu a ten slúži ako jedinečný identifikátor pre aplikáciu. Manifest obsahuje taktiež povolenia, ktoré aplikácia vyžaduje k činnosti a ďalšie

[4] https://developer.android.com/guide/platform/images/android-stack_2x.png
[5] https://developer.android.com/reference/android/content/Intent.html

informácie o nej. Konkrétne hlavné komponenty sú Activity, Service, Broadcast a Content Providers.

2.3.1 Activity

Activity (obrázok 2.3) je základom každej Android aplikácie. Užívateľ prostredníctvom nej ovláda aplikáciu. Je to okno, do ktorého je vložené užívateľské prostredie. Dá sa prirovnať k oknu v desktopových systémoch alebo jednej internetovej stránke. Väčšinou zaberá skoro celú plochu displeja. Nezvykne zakrývať stavový riadok, kde sú zobrazené upozornenia, čas, stav batérie. Pri zapnutí aplikácie sa spúšťa hlavná Activity pričom je vždy, len jedna hlavná.



Obr. 2.3: Activity aplikácie s tlačidlami a textom

Aplikácia sa väčšinou skladá z viacerých Activity, ktoré majú určitú náväznosť. Napríklad z Activity1 sa stlačením tlačidla 1 dostaneme do Activity2, stlačením tlačidla späť sa vrátime na Activity1. V Activity1 stlačením tlačidla 2 sa dostaneme do Activity3 a podobne. Z tohto dôvodu existuje zásobník nazývaný back stack[6], kam sa Activity ukladajú. Je potrebné si spravovať Activity pri prechodoch, aby sa v zásobníku neobjavili viackrát ako je vyžadované. Napríklad nechceme, aby pri prechode z Activity2 späť na Activity1, ktorá sa nachádza v zásobníku, sme vytvárali novú Activity1. Ak by sme ju vytvorili a užívateľ by aplikáciu zatvoril, znovu by uvidel predchádzajúcu Activity1, lebo tá bola ešte v zásobníku.

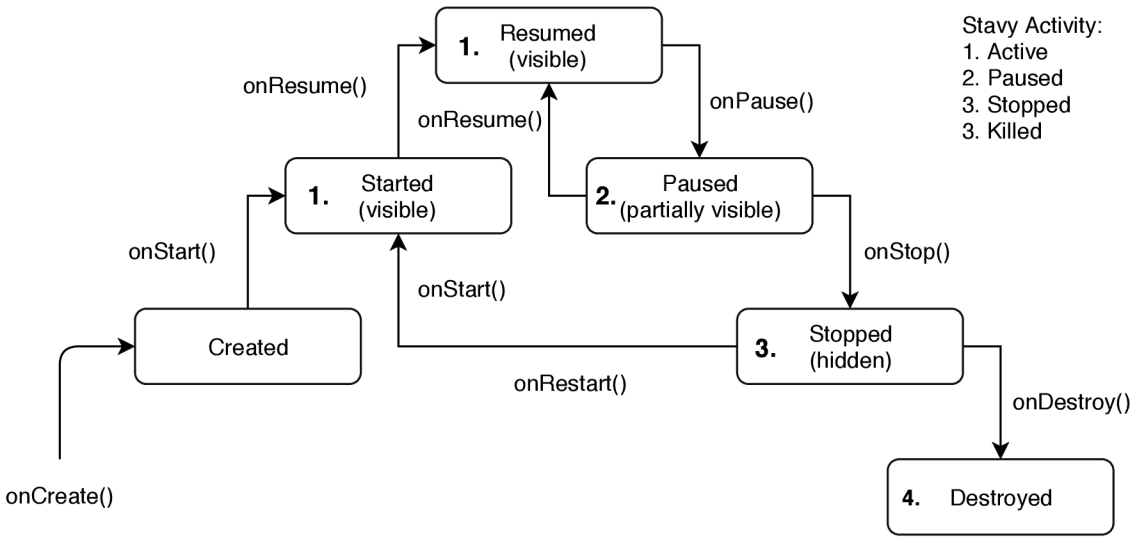
Activity sú znovupoužiteľné aj pri vývoji inej aplikácie. To znamená, že ak je potrebné v novej aplikácii implementovať správanie, ktoré už niekedy bolo implementované je možné

[6] https://developer.android.com/guide/components/activities/tasks-and-back-stack.html

ho znovu využiť. V tom prípade ostáva len zmeniť dizajn a zdroj dát Activity podľa potreby novej aplikácie. Activity sú tvorené ako podtrieda základnej Android Activity triedy. Activity nemôže volať metódy inej Activity alebo pristupovať k jej dátam. Je to možné dosiahnuť pomocou Intents[7] a Content Providers.

2.3.1.1 Životný cyklus Activity

Každá aplikácia má zásobník back stack, ktorý obsahuje jej Activity. Podľa pozície Activity v zásobníku vieme určiť jej stav [9]. Pri Activity rozoznávame niekoľko stavov a ich prechody sú znázornené na obrázku 2.4.



Obr. 2.4: Životný cyklus Activity

Stavy Activity sú nasledovné:

- **Active** - práve beží na popredí. Je viditeľná na displeji a užívateľ s ňou môže interagovať. Pri uvoľnovaní pamäte by to bola jedna z posledných možností, ktorá by sa ukončovala.
- **Paused** - je možné aby ju užívateľ videl, keď Active Activity nezaberá celú obrazovku alebo je priesvitná. Nemôže s ňou narábať, lebo je pozastavená a nie je Active. Je v back stack pripravená na rýchle prepnutie do stavu Active a presunutie na vrchol back stack. Tento stav napríklad nastane, ak vyskočí dialógové okno, ktoré prekrýva predchádzajúcu Activity. Je ju vidieť, ale nie je možné s ňou narábať, lebo je aktívne dialógové okno.
- **Stopped** - keď už Activity nie je viditeľná, čiže ju prekryje celá nová Activity. Pamätá si stavy a informácie. Je väčšia pravdepodobnosť, že bude zrušená v prípade nedostatku pamäte.
- **Killed** - Activity, ktorú buď užívateľ zničil spätným tlačítkom alebo z dôvodu nedostatku pamäte bola zrušená. Už nie je v zásobníku back stack. Opätovné vyžiadanie si zrušenej Activity znamená, že je potrebné ju opäť vytvoriť.

[7] https://developer.android.com/reference/android/content/Intent.html

2.3.2 Services

Services sú procesy bežiace na pozadí. Môžu byť spustené z aplikácie za účelom, keď akcia potrebuje dlhší čas na dokončenie. Sú určené aj na dlhodobé úlohy za účelom kontrolovania notifikácii. Napríklad pri počúvaní hudby a využívaní iných aplikácii ako je hudobný prehrávač, tak Service spustený z aplikácie hudobného prehrávača sa stará, aby hudba hrala. Service nemá grafické rozhranie. Upozorniť na svoj výsledok alebo problém môže pomocou Toast[8] správ alebo notifikácií. Existujú 3 typy Services:

- **Foreground** vykonáva operácie, ktoré užívateľ môže zaznamenať. Ako napríklad spomínané prehrávanie hudby. Počas celej doby musí zobrazovať ikonu na stavovom riadku a služba beží aj keď užívateľ nenarába s aplikáciou.
- **Background** vykonáva operácie, ktoré si užívateľ nevšimne. Prikladom môže byť odber RSS noviniek[9].
- **Bound** nastane vtedy, keď sa naň naviaže iný komponent aplikácie, vytvára rozhranie typu klient-server. Pomocou klient-server rozhrania môžu komponenty so službou komunikovať, prijímať výsledky, posielať požiadavky. Bound service beží pokiaľ je naň jeden alebo viac komponént naviazaných a v prípade, keď sa všetky odpoja ukončí sa.

2.3.3 Content providers

Content provider poskytuje abstrakciu nad dátami danej aplikácie. Hodí sa použiť hlavne, keď aplikácia bude svoje dáta zdieľať s inými aplikáciami. Ostatné aplikácie budú zdieľané dáta získavať pomocou Content provider aplikácie, ktorá dáta ponúka. Vďaka abstrakcii, ktorú vytvára sa pri zmene zdroja dát nemusí prerábať prepojenie medzi novým zdrojom vo všetkých aplikáciach, ktoré dáta čerpajú. Vytvorí sa prepojenie medzi novými dátami a Android triedou Content provider. Zjednodušená zmena zdroja dát, teda úprava prepojenia iba medzi Content provider a zdrojom dát, platí aj keď aplikácia nebude svoje dáta zdieľať.

2.3.4 Broadcasts

Android aplikácie môžu vysielať alebo prijímať broadcast správy z iných aplikácii alebo od Android systému. Broadcast správy sú poslané naraz všetkým odoberateľom. Broadcast je odoslaný vysielačom správ a prijatý prihlásenými odoberateľmi (aplikáciami). Tieto správy sú vyslané, keď nastane požadovaná udalosť. Napríklad systém Android vyšle broadcast správu hneď ako sa zariadeniu začne nabíjať batéria. Android aplikácia sa môže prihlásiť na konkrétny broadcast, ten je možné použiť na komunikáciu medzi aplikáciami.

Keď si vytvárame vlastný príjmač, tak rozširujeme Broadcast receiver triedu z Java API Framework definovaním vlastného správania. Broadcast sa dá odoslať v Androide tromi spôsobmi:

1. **Ordered broadcast**[10] je odosielaný v poradí. Poradie, v ktorom prijímače obdržia správy sa dá určiť prioritou. Prijímače s rovnakou prioritou obdržia správy v ľubovolnom poradí. Broadcast sa odošle prvému príjmaču. Ten správu spracuje, môže ju

[8] https://developer.android.com/guide/topics/ui/notifiers/toasts.html
[9] http://www.whatisrss.com/
[10] https://android-developers.googleblog.com/2011/01/processing-ordered-broadcasts.html

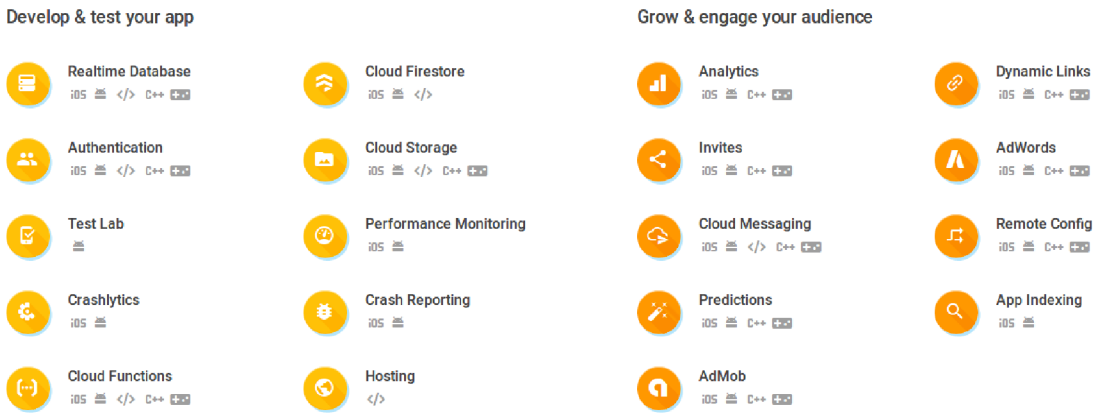
upraviť a upravenú alebo pôvodnú správu pošle ďalšiemu prijímaču alebo broadcast preruší a správa sa už ďalej nedostane.

2. **Normal broadcast** odosiela všetkým prijímačom naraz. Nevýhodou je, že prijímače nemôžu prečítať výsledok od iného prijímača, broadcast nemôže byť prerušený.

3. **Lokálny broadcast**[11] je odoslaný prijímačom v tej istej aplikácii ako je vysielač. To znamená, že dáta neopustia aplikáciu. Napríklad pridáme k zobrazeniu notifikácie tlačidlo, ktoré vytvorí lokálnu broadcast správu. Keď sa táto správa zachytí vykoná sa patričná akcia.

2.4 Google Firebase

Google Firebase je platforma uľahčujúca vývoj aplikácii pre Android, iOS a internetové aplikácie[1]. Ponúka množstvo funkcionalít uvedených na obrázku 2.5. Môžeme si vybrať, ktoré z nich budeme používať. Aktúalne platforma ponúka bezplatne pre realtime databázu 1GB dát , 100 simultánnych pripojení a 10GB stiahnutých dát za mesiac. Pri cloud funkcionalite je to 125 000 invokácii služby. Authentication vytvára prihlasovacie prostredie pre overenie užívateľa. Prihlásenie je možné cez Gmail, Facebook, Twitter alebo Github účet. V bezplatnom režime je to obmedzené na 10000 overení za mesiac. Android zariadenie musí mať verziu systému 4.0 a vyššiu, aby mohlo využívať produkty Firebase.



Obr. 2.5: Firebase produkty[12]

2.4.1 Firebase cloud messaging

Posielanie dátových správ alebo notifikácii do zariadenia zabezpečuje Firebase cloud messaging. Správa môže so sebou niesť až 4KB dát. Dátovú správu zachytí Service aplikácie a vykoná požadovanú akciu a to tak, že užívateľ ani nebude vedieť, že správa prišla alebo v prípade potreby aplikácia môže z prijatých informácii vytvoriť viditeľnú notifikáciu pre užívateľa. Ak sa nejedná o dátovú správu, rovno sa vytvorí notifikácia. Notifikácie alebo

[11] https://android.jlelse.eu/local-broadcast-less-overhead-and-secure-in-android-cfa343bb05be

[12] https://firebase.google.com/

dátové správy je možné posielať prihláseným zariadeniam na tému (téma je vytvorená v prostredí cloud messaging), jednému zariadeniu alebo skupine zariadení. Potrebné je implementovať dve hlavné komponenty na odosielanie a prijímanie. Odosielanie môže byť cez Admin SDK[13] alebo cez HTTP a XMPP API. O príjimanie sa postará mobilná aplikácia alebo javascript klient.

2.4.2 Firebase realtime databáza

Firebase realtime databáza je typu NoSQL, ktorý bude popísaný v sekcii 3.2. Dáta sa v databáze ukladajú vo formáte JSON[14]. Aktuálne dáta sú prístupné všetkým pripojeným zariadeniam v rámci milisekúnd. Namiesto klasických HTTP požiadaviek využíva synchronizáciu dát zakaždým ako sa dáta zmenia. To znamená, že máme aktuálne dáta bez vyžiadania. Ak je zariadenie offline, ukladá si dáta lokálne a pri následnom pripojení sa synchronizuje s aktuálnym stavom servera. Realtime databáza ponúka jazyk pravidiel nazývaný Firebase Realtime Database Security Rules. Tieto pravidlá určujú, kto môže k dátam pristupovať, ako majú byť štrukturované a kedy môžu byť čítané alebo zapisované.

[13] https://firebase.google.com/docs/cloud-messaging/admin/
[14] https://www.json.org/

Kapitola 3

Identifikovanie zdrojov dát

Nastanú situácie, keď používatelia počítačov čakajú na výsledok, ale nevedia kedy sa vypočíta, pridá medzi ostatné dáta a podobne. V takom prípade by bolo najlepšie nejakým spôsobom informovať o výsledku namiesto pravidelnej kontroly (zmena prístupu z pull na push). Keď chceme informovať o zmenách dát na serveri, tak na začiatok si nejaké zdroje dát potrebujeme vybrať. Obsahom tejto kapitoly je identifikácia a popis zdrojov dát pre monitorovanie.

Vhodným adeptom na zdroj dát určených pre monitorovanie sú databázy. Do databáz sa ukladá veľké množstvo dát či sa jedná o používateľov nejakej služby, záznamy zo senzorov alebo čokoľvek iné. Medzi najpoužívanejšie databázové technológie stále patria relačné databázy. V obľube sú čoraz viac Big Data a NoSQL databázy. Ak máme databázu internetového obchodu, môže nás zaujímať, keď počet objednávok prekročí určitý počet.

Operačný systém poskytuje poskytuje tiež veľké množstvo dát na monitorovanie. Táto práca identifikuje zdroje na desktopových operačných systémoch Microsoft Windows a Linux. V rámci operačného systému prostredníctvom príkazového riadku je možné pristupovať ku všetkým informáciám. Napríklad v príkazovom riadku pomocou konzolových príkazov sa dá zistiť stav procesov a služieb. Pre beh systému správy databáz tiež potrebujeme operačný systém, na ktorom bude fungovať.

3.1 Relačná databáza

Z klasických databáz ako objektové, relačné, objektovo-relačné sú relačné najpopulárnejšie. Väčšinou využívajú jeden server, kde sú dáta uložené a ak potrebujeme uskladniť väčšie množstvo dát zväčšíme úložný priestor na serveri. Podľa DB-ENGINES[1] je v top desať najpoužívanejších databázových systémov šesť relačných. Najpoužívanejšie relačné databázové systémy podľa poradia sú Oracle, MySQL, Microsoft SQL Server, Postgre SQL, DB2 a Microsoft Acces.

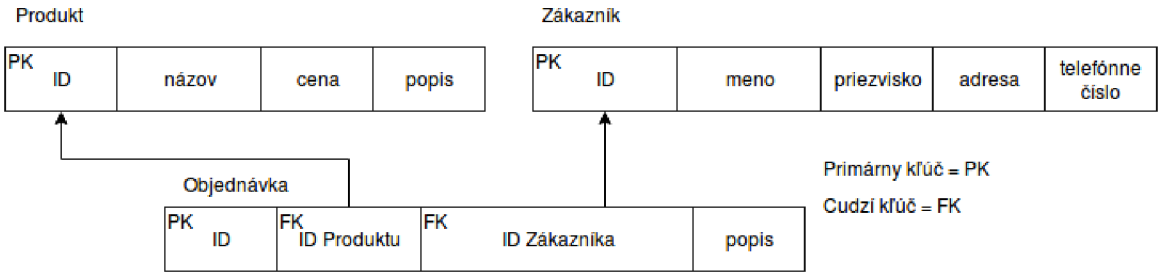
Princíp relačnej databázy je, že dáta sú uložené v tabuľkách a tabuľky sú medzi sebou prepojené referenciou [5]. Tento typ databázy sa nazýva relačná, vďaka matematickej definícii pojmu relácia.

Tabuľke vopred definujeme formát, v ktorom bude informácie ukladať. Pre jednoznačné identifikovanie záznamu (riadku) tabuľky je potrebný primárny kľúč. Primárny kľúč je jedinečný identifikátor záznamu v rámci tabuľky. Primárnym kľúčom môže byť napríklad poradie záznamu v tabuľke alebo informácia, ktorá v tabuľke je jednoznačne unikátna. Pri-

[1] Navštívené dňa 18.1.2018 https://db-engines.com/en/ranking

márny kľúč sa dá vytvoriť aj kombináciou hodnôt, viacerých stĺpcov tabuľky. Keď chceme tabuľky prepojiť využívame cudzí kľúč, ktorý jednoznačne odkazuje na riadok odkazovanej tabuľky.

Na zjednodušenom príklade si ukážeme ako to vyzerá pre internetový obchod (obrázok 3.1). Nachádza sa v ňom tabuľka s produktami. Produkt má svoje ID, čo je primárny kľúč, názov, cenovku a popis. Ďalej existuje tabuľka zákazník. Tá by mala obsahovať základné údaje o zákazníkovi ako je napríklad meno, priezvisko, adresa (adresa je vhodným adeptom na samostatnú tabuľku), telefónne číslo. Ako primárny kľúč bude použité vygenerované poradové číslo.



Obr. 3.1: Príklad tabuliek relačnej databázy

Tretia tabuľka sú objednávky. Obsahuje cudzí kľuč odkazujúci na zákazníka, teda zákazníkov primárny kľúč a to isté pre produkt. Tiež obsahuje svoj vlastný primárny kľúč. Týmto spôsobom máme v našej databáze informácie o produktoch, zákazníkovi a jeho objednávkach.

K dátam v relačnej databáze pristupujeme pomocou *SELECT* dotazu. Jeho základná syntax je:

SELECT stĺpec1, stĺpec2, ... FROM názov_tabuľky

Tento výraz vieme doplniť o klauzulu *WHERE*, za ktorou nasleduje podmienka. Nemusíme ostať pri jednej podmienke a pomocou výrazov *AND*, *OR* vieme pridávať ďalšie. Vyberať dáta sa dajú aj z viacerých tabuliek naraz. Na to sa používa výraz *JOIN*, pomocou ktorého pospájame riadky patriace ku sebe. Je možné použiť aj určité funkcie, napríklad funkcia *Count()* spočíta počet výsledných prvkov, funkcia *Avg()* vypočíta priemer alebo funkcia *Sum()* sčíta prvky dokopy.

3.1.1 MySQL

MySQL je open-source systém pre správu databáz vyvíjaný spoločnosťou Oracle[2]. MySQL je rýchly a spoľahlivý, funguje na rôznych operačných systémoch a je možné ho integrovať do programu pomocou známych programovacích jazykov. Zdrojový kód MySQL je napísaný v C a C++. Je navrhnutý, aby fungoval paralelne. To sa dá dosiahnuť pomocou viacerých vláken a s týmto spracovaním sa využije potenciál viacjadrových procesorov.

Obrázok 3.2 ilustruje tabuľku z MySQL databázy s názvom *passGuess*. Tabuľka je zobrazená vo vývojovom prostredí PyCharm. Dáta, ktoré sú na nej uvedené sa viažu na hádanie hesiel. Prvý stĺpec je typu INTEGER, predstavuje identifikačné číslo procesu, ktorý je možné riešiť. Druhý a tretí stĺpec sú typu BIT, môžu nadobúdať hodnoty 0 alebo 1. Druhý

[2] https://www.oracle.com

stĺpec informuje či proces beží a tretí, či už je vyriešený. Štvrtý stĺpec je typu VARCHAR s počtom znakov sto a je určený pre uloženie výsledku. Posledný piaty stĺpec je primárny kľúč.

	PROCESS_ID	RUNNING	SOLVED	SOLUTION	pk
1	101	☑	☐	<null>	1
2	102	☐	☑	a2A3s1d5Sf4e	2
3	103	☑	☐	<null>	3
4	104	☑	☐	<null>	4
5	105	☐	☑	12A2s3d5e8rR9t	5

Obr. 3.2: Tabuľka MySQL databázy v PyCharm IDE

Napríklad správca takýchto údajov čaká na to, aby boli všetky procesy dokončené. Dopredu nemá ako vedieť kedy táto udalosť nastane a za normálnych okolností by musel zisťovať či už prebehli všetky procesy v poriadku. Nasledujúcim dotazom by zistil počet dokončených procesov, no nebude informovaný keď skončia všetky automaticky:

SELECT COUNT(PROCESS_ID) FROM passwordGuessing
WHERE RUNNING = 0

3.2 Big data a NoSQL databázy

Termín Big Data zahrňuje problematiku týkajúcu sa veľkého objemu dát [4]. Pojem Big Data na začiatku roka 2001 definoval priemyselný analytik Dough Lane[3] ako tri V:

- **Velocity** (veľkosť);
- **Volume** (nárast);
- **Variety** (rôznorodosť).

Ak máme na mysli Big Data, tak ich veľkosť je taká, že nám nestačí jeden server pre uloženie a spracovanie, ale využívame ich niekoľko desiatok. Využívajú sa v priemysle pri: zbere dát zo senzorov, sociálnych sieťach, mobilných technológiách, obchodovaní na burze. Najväčšia sociálna sieť Facebook[4] má mesačne 2 miliardy aktívnych užívateľov. Facebook k tomu využíva dátový sklad s 300PB dát, ktoré spracováva. Pri Big Data je väčšinou veľký nárast dát.

Big Data sú pološtrukturované dáta napríklad vo formáte XML, JSON alebo nie sú štrukturované vôbec. Súčastne sa na analýzu Big Data používajú rôzne spôsoby strojového učenia. Keďže jednotná a formálne presná definícia nie je, k pôvodnym trom V pribúdajú ďalšie. Napríklad Value (hodnota pre firmu), Validity (obmedzená doba platnosti), Veracity (neistá vierohodnosť).

NoSQL vzniklo pre potreby Big Data. Potrebujeme použiť iný spôsob ukladania a prístupu k dátam ako pri relačných databázach a riešením sú NoSQL databázy. NoSQL zahrňuje technológie pre efektívne spracovanie Big Data. Výhodami NoSQL databáz je ich

[3] pôvodný článok https://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf

[4] Štatisitka z novembra roku 2017 https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/

flexibilná škálovatelnosť a flexibilný dátový model. Vertikálna škálovateľnosť znamená, že za cieľom zlepšenia výkonu vynovujeme hardvér na jednom stroji. NoSQL databázy fungujú v uzloch ako distribuovaná sieť medzi viacerými servermi a to sa nazýva horizontálna škálovateľnost. Pri potrebe zvýšiť výkon sa zapojí ďalší server. Dátový model NoSQL databáz buď nie je alebo je voľný. Dodržovanie nie je striktné, často je to v réžii samotnej aplikácie. Vytvárať dátové štruktúry pre NoSQL je vhodné tak, aby odpovedali najčastajším dotazom.

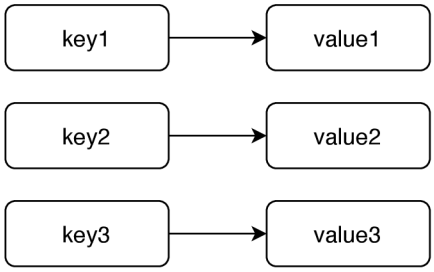
Keďže sa jedná o horizontálnu škálovateľnosť a systém je distribuovaný vo viacerých uzloch, problémom bude počiatočná inštalácia, správa a údržba všetkých uzlov. Ďalším problémom je, že neexistuje štandartný dotazovací jazyk ako u SQL. To má za následok, že každá NoSQL databáza má svoj vlastný dotazovací jazyk.

Základné typy NoSQL databáz vychádzajúce z ich dátového modelu:

- databázy typu kľúč-hodnota;
- dokumentové databázy;
- stĺpcové databázy;
- grafové databázy.

3.2.1 Databázy typu kľúč-hodnota

Databázy typu kľúč-hodnota umožňujú ukladanie dát na základe jedinečných kľúčov. S dátami sa dá pracovať iba na základe určeného kľúča a operácie nad vzniknutými dátami sú jednoduché. Majú slobodný dátový model. Tieto databázy fungujú podobne ako hašovacia tabuľka alebo asociatívne pole. Podľa jedinečného kľúča ukladajú hodnoty. Z obrázku 3.3 vidíme, že pod kľúčom *key1* je uložená hodnota *value1*.



Obr. 3.3: Spôsob uloženia dát databázy typu kľúč hodnota

3.2.2 Dokumentové databázy

Dokumentové databázy ukladajú a spravujú štrukturované dokumenty, ktoré okrem dát obsahujú aj metadáta popisujúce význam jednotlivých častí dokumentov. Príkladom sú JSON, BSON, XML a ďalšie hierarchické štruktúry. Tieto formáty tvoria stromovú štruktúru. Obsahujú asociatívne pole, zoznamy a základné dátové typy. Dokumenty v dokumentových databázach nie sú využívané len na ukladanie dát, ale aj na komunikáciu. To znamená, že dáta nemusia meniť svoj formát. Klient požaduje dáta v rovnakom formáte ako ich má databáza uložené. Preto stačí požadované dáta vybrať a bez zmeny formátu odoslať.

Dáta sú uložené v kolekciách dokumentov. V kolekcii nie je daný striktný model, ktorý by sa mal dodržovať. V relačnej databáze je vopred definovaný formát tabuľky. V dokumentových databázach môže jeden dokument obsahovať kľúče meno, priezvisko, vek a adresa

a ďalší dokument bude obsahovať rovnaké kľúče v inom poradí alebo úplne iné kľúče. Je možné dáta ukladať aj neštruktúrovane, pokiaľ je aplikácia na to prispôsobená, ale zvykne sa nejaká schéma používať.

Jedným z predstaviteľov dokumentových databáz je MongoDB, patrí medzi najpoužívanejšie NoSQL databázy. V nej má každý dokument atribút „*_id*“ slúžiaci ako primárny kľúč, jeho hodnota musí byť unikátna.

Pri rozdelení dát do kolekcií dokumentov je viac možností. Buď dokument bude využívať vnorené dokumenty alebo bude odkazovať na iný dokument. Vnorený dokument znamená, že pod kľúčom dokumentu nenájdeme jednu hodnotu, ale celý dokument.

Vnorený dokument má výhodu, že vieme manipulovať s dátami v rámci jednej operácie. Pri odkazovaní je to podobné ako s cudzími kľúčmi v relačných databázach. Ostatné dokumenty potrebujú vedieť k čomu patria a musia mať uložený primárny kľúč hlavného dokumentu. S odkazovanými dokumentami je problém, že ak chceme dostať všetky dáta, jedná sa o viacero dotazov do databázy. Dokumentové databázy málokedy umožňujú v jednom dotaze vybrať viac dokumentov.

Ako vyzerá typ dokumentu pre dokumentové databázy ilustruje obrázok 3.4. Môže sa jednať o kolekciu zamestnancov. K danému dokumentu by sme sa dostali pomocou jedinečného atribútu „*_id*“. Ak by sme chceli nájsť dokument podľa mena (*firstName*), tak by sme ako výsledok s najväčšou pravdepodobnosťou dostali viac dokumentov, lebo očakávame, že mená sa opakujú. Pod kľúčom *address* sa nachádza vnorený dokument obsahujúci dodatočné informácie k adrese.

```
{
        "_id":462154,
        "firstName": "Roman",
        "lastName": "Golian",
        "age": 23,
        "phoneNumber": "+421911123456",
        "address":
                        {
                        "state": "Slovakia",
                        "city": "Detva",
                        "street": "Ulica 8"

                        }
}
```

Obr. 3.4: Dáta dokumentovej databázy

3.2.3 Ostatné NoSQL databázy

Stĺpcové databázy[5], ako z názvu vyplýva, ukladajú dáta do stĺpcov. Riadok klasickej relačnej databázy, by bol rozdelený do viacerých stĺpcov na tom istom indexe. Dáta jedného stĺpca sú na disku uložené za sebou. To nám zabezpečuje, že je možné hádať na akej adrese budú potrebné dáta z ďalšieho stĺpca. Použitím dátových typov s fixnou dĺžkou by sme to vedeli určiť presne.

[5] https://medium.com/@hellomichibye/column-oriented-database-introduction-part-1-572e5780aebb

Grafové databázy[6] využívajú grafovú štruktúru pre získavanie a ukladanie dát. Základom je pochopiť termíny uzol a hrana. Uzol nesie podstatnú informáciu o entite, ktorú reprezentuje. Môžeme ho prirovnať k riadku v relačných databázach. Hrana prepája uzly medzi sebou. Pomocou hrán je možné nachádzať určité vzory v dátach a pochopiť vzťahy v nich.

Táto práca sa nebude venovať ostatným typom NoSQL databáz podrobne, pretože nie sú pre výsledné riešenie dôležité.

3.3 Príkazový riadok

Príkazový riadok je používateľské rozhranie, v ktorom s užívateľom komunikuje s programom pomocou textu. Užívateľ zadáva príkazy, ktoré spracuje interpret príkazového riadku nazývaný shell. Spúšťanie programov a výstup v príkazovom riadku je na rôznych operačných systémoch rovnaký. Spúšťanie programov v príkazovom riadku je v tvare: *názov_príkazu prvý_parameter druhý_parameter atď.*

```
~$ df
Súborový systém  1K-bloky     Použ Dostupné Pou% Pripojený na
udev               8141320        0  8141320   0% /dev
tmpfs              1633556     9684  1623872   1% /run
/dev/sdb6        113401504 43345760 64272244  41% /
tmpfs              8167772    18036  8149736   1% /dev/shm
tmpfs                 5120        4     5116   1% /run/lock
tmpfs              8167772        0  8167772   0% /sys/fs/cgroup
/dev/sdb2           262144    34432   227712  14% /boot/efi
cgmfs                  100        0      100   0% /run/cgmanager/fs
tmpfs              1633556       48  1633508   1% /run/user/1000
/dev/sdc1          1955520  1374784   580736  71% /media/michal/E0FD-1813
~$
```

Obr. 3.5: Príklad výsledku po zadaní príkazu v termináli

3.3.1 Linux

Linux je rodina open-source a voľných operačných systémov [10], ktoré majú spoločné to, že bežia na Linuxovom jadre. V roku 1991 Linus Torvald vydal linuxové jadro. Vďaka Androidu, ktoré má linuxové jadro je nainštalovaný na najviac zariadeniach. Operačné systémy z rodiny linux sú často nainštalované na serveroch.

Typickými interprétmi pre príkazový riadok v operačných systémoch rodiny Linux sú Korn Shell (ksh), Bourne shell (sh), C Shell (csh), Bourne Again Shell (bash). Napríklad v systéme Ubuntu (OS z Linux rodiny) sa k príkazovému riadku dostaneme vďaka gnome-terminal. Príkazový riadok v systéme Ubuntu využíva bash implicitne, ale je to možné zmeniť.

Príklad príkazu vykonaného v Bourne shell a je zobrazený v obrázku 3.6. Na prvom riadku je viac príkazov prepojených pomocou *pipeline*. Výsledok prvého príkazu sa posunie ďalšiemu a podobne. Prvý príkaz zistí všetky práve bežiace procesy, výsledok dostane program *grep*, ktorý vyselektuje riadky s obsahom reťazca *python3* a druhý výsledok spracuje program *wc*. Tomu parameter *-l* určuje, že má spočítať riadky. Výsledok napísaný na dru-

[6] https://medium.com/@wjaynes/graph-databases-living-on-the-edge-f6307a6c5088

hom riadku uvádza počet riadkov zo všetkých procesov s reťazcom *python3*. Inak povedané, počet bežiacich skriptov interpretovaných pomocou Python3.

```
/ # ps -A | grep "python3" | wc -l
2
/ #
```

Obr. 3.6: Príklad výsledku po zadaní príkazu v linux termináli s Bourne shell

3.3.2 Microsoft Windows

Microsoft Windows je najpopulárnejším operačným systémom pre stolné a prenosné počítače. Je na trhu už 32 rokov od firmy Microsoft. Má prívetivé grafické užívateľské prostredie a nie je ťažké ho obsluhovať. Je dostupný v 137 jazykoch. Aktuálne najnovšia verzia je Windows 10.

V operačnom systéme Windows je príkazový riadok prístupný cez *cmd.exe* [7]. Jeho oficiálny názov je Windows Command Processor. Je zastaralý a nedovoľuje prístup k viacerým administrátorským nástrojom. Dokáže interpretovať skripty z Batch súborov. Tie majú príponu BAT a obsahujú príkazy, ktoré môžu byť vykonané aj samostatne, keď sú zadané do príkazového riadku.

Keďže spomínaný príkazový riadok je zastaralý a neposkytuje všetky funkcie v roku 2006 vo verzii Windows 7 sa prvýkrát objavil PowerShell. O desať rokov na to Microsoft urobil PowerShell open-source a dokáže fungovať aj na iných platformách (napr. Ubuntu, CentOS, macOS). Dokáže spúšťať 4 typy príkazov a to cmdlets, PowerShell skripty, PowerShell funkcie a samostatne spustiteľné programy. Cmdlets sú programy z .NET Framework navrhnuté pre PowerShell. Na vstupe cmdlets môže byť objekt a výsledky sú vždy podávané ako objekty.

```
C:\Users\Lenovo Z510>ipconfig | find "2001:67c:1220:c1a3"
   IPv6 Address. . . . . . . . . . . : 2001:67c:1220:c1a3:9c20:7f0f:53ff:ca9c
   Temporary IPv6 Address. . . . . . : 2001:67c:1220:c1a3:a595:62b9:5845:46a0

C:\Users\Lenovo Z510>
```

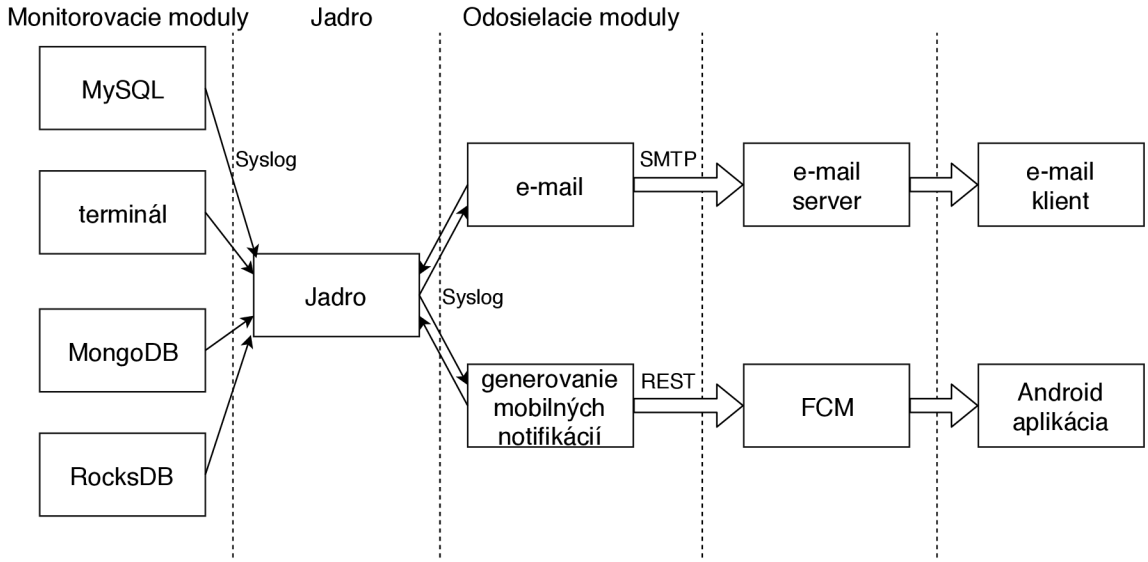
Obr. 3.7: Príklad výsledku po zadaní príkazu vo Windows cmd.exe termináli

Kapitola 4

Návrh serverovej časti a Android aplikácie

Na serveri bude prebiehať monitorovanie dát a generovanie notifikácii. Monitorovať bude možné relačnú MySQL databázu, NoSQL databázu a výsledky príkazov v linux a Windows terminály. Za NoSQL zástupcov som zvolil MongoDB a RocksDB. Akonáhle nastane udalosť, ktorú očakávame potrebujeme odoslať správu. Správa bude doručené e-mailom alebo push notifikáciou na zariadenie Android prostredníctvom aplikácie.

Problematika serverovej časti, teda monitorovanie a generovania notifikácii je rozdelené do piatich častí ako môžeme vidieť na obrázku 4.1 . Prvú časť tvoria monitorovacie moduly, ktoré sledujú požadovanú udalosť. Udalosťou sa myslí zmena hodnoty v databáze, výsledná hodnota skriptu nadobudla očakávanú hodnotu a podobne. Druhou časťou je jadro, ktoré tvorí prostredníka medzi prvou a treťou časťou. Monitorovacie moduly jadru odošlú správu, keď nastane udalosť. Jadro dostane informáciu o tom čo sa stalo a odošle správu modulu v tretej časti. Tretiu časť tvoria dva odosielacie moduly. Jeden má za úlohu odosielať maily a druhý generuje notifikácie pre mobilné zariadenia.



Obr. 4.1: Systém monitorovania dát a generovania notifikácie rozdelený do troch častí

Štvrtú časť tvoria prostredníci medzi odosielacími modulmi a klientskou aplikáciou zabezpečujúci doručenie na mailový server alebo push notifikáciou do mobilného zariadenia. Piatu časť tvorí mailový klient a Android aplikácia, ktorá slúži na zobrazovanie správ užívateľovi. Push notifikácia je spôsob ako správu čo najrýchlejšie dostať k užívateľovi, keďže smartfóny väčšinou máme so sebou, lebo nám uľahčujú komunikáciu a získavanie informácii.

4.1 Komunikácia modulov

Moduly komunikujú cez TCP[1] protokol ale aj UDP[2] protokol. Záleží, pre ktorý protokol sa užívateľ rozhodne. TCP protokol zabezpečuje spoľahlivý prenos dát. Vytvára stále spojenie medzi serverom a klientom. UDP protokol nezaručuje spoľahlivý prenos dát a akceptuje straty. Jadro beží ako server, na ktorý sa monitorovacie a odosielacie moduly pripoja.

4.1.1 Syslog

Syslog je protokol považovaný za štandard pre logovanie správ. Funguje ako klient-server čo dovoľuje ukladanie správ na server z rôznych zdrojov. Syslog správy majú informačný charakter, napríklad o chybe programu. Aby aplikácia mohla reagovať aj na od neznámych zdrojov, bude pre výmenu správ použitý formát syslog. Je možné vytvoriť vlastné logovacie zariadenie, ktoré odošle správu pri nejakej udalosti. V tejto práci logovacie zariadenia predstavujú monitorovacie moduly.

Podľa RFC 3164 [6] je maximálna dĺžka paketu obsahujúceho syslog správu 1024 bajtov. Minimálna dĺžka nie je stanovená, ale nemalo by sa jednať o prázdny syslog paket. Plný formát syslog správy má tri časti. Prvou je PRI známa ako hodnota priority, druhou je hlavička HEADER a treťou časťou je správa MSG.

Časť PRI musí mať 3 až 5 znakov. Tie pozostávajú z čísla a porovnávacích znamienok <>, ktoré ho ohraničujú. Uvedené číslo reprezentuje hodnotu pre zariadenie z ktorého bola správa vygenerovaná a závažnosť správy. Číslo zariadenia je pridelené niektorým systémovým démonom a procesom. Tie procesy a démoni, ktorí nemajú hodnotu priradenú môžu použiť akúkoľvek z „local use“ zariadení alebo „user-level“. Zariadenia majú priradené hodnoty od 0 až 23 a závažnosť správ môže nadobúdať hodnôt od 0 po 7. Niektoré hodnoty sú znázornené v tabuľke 4.1.

Hodnota priority je počítaná vynásobením čísla zariadenia číslom 8 a pripočítaním čísla závažnosti. Napríklad užívateľská správa s numerickým kódom 1 a kritickou závažnosťou s číslom 2 bude mať PRI <10> (8*1+2).

Časť HEADER obsahuje časovú značku a IP adresu alebo názov zariadenia. Pokiaľ odosielacie zariadenie nemá názov použije sa IP adresa. Ak má zariadenie viacej IP adries, môže byť nakonfigurované tak, aby odosielalo správy s jednou IP adresou nezávisle z ktorého rozhrania správa bude odoslaná, tým sa zabezpečí jednotný názov zariadenia pre všetky správy. Časová značka je lokálny čas vo formáte „Mesiac deň hodina:minúta:sekunda“.

MSG časť tvorí zvyšok syslog paketu. Obsahuje dodatočné informácie o procese, ktorý vytvoril správu a text správy. Táto časť nesmie obsahovať netlačiteľné znaky.

[1] https://tools.ietf.org/html/rfc793
[2] https://www.ietf.org/rfc/rfc768.txt

Facility numerical code	Facility	Severity numerical code	Severity
0	kernel messages	0	Emergency: system is unusable
1	user-level messages	1	Alert: action must be taken immediately
2	mail system	2	Critical: critical conditions
3	system daemons	3	Error: error conditions
4	security/authorization messages (note 1)	4	Warning: warning conditions
5	messages generated internally by syslogd	5	Notice: normal but significant condition
...	...	6	Informational: informational messages
23	local use 7 (local7)	7	Debug: debug-level messages

Tabuľka 4.1: Hodnoty zariadení a závažnosti syslog správ

4.1.2 Formát správ

Pre potreby aplikácie je potrebné doručiť z monitorovacích modulov informáciu odosielacím modulom o tom pre koho je správa určená, akým spôsobom sa má správa odoslať a samotný obsah správy popisujúci detekovanú udalosť. Na tieto účely využijeme časť MSG syslog správy. V nej použijeme vlastný formát správy, ktorý bude jednoduchý na spracovanie.

Správa je rozdelená do 3 hlavných častí plus začiatočná a koncová časť. Znak doláru bude použitý ako oddeľovač medzi jednotlivými časťami. Správa začína jednoduchými úvodzovkami a nasleduje kľúčové slovo **start**. Po ňom idú názvy odosielacích modulov pre ktoré je správa určená, za nimi je oddeľovací znak. Pokračuje to názvom cieľovej skupiny, ktorá má správu dostať. Ďalší znak doláru odddeľuje cieľovú skupinu a správu, určenú pre túto skupinu. Po správe je oddeľovač a ukončovacie kľúčové slovo **end**.

'start$<spôsob_doručenia>$<cieľová_skupina>$<správa>$end'

Príklad takejto správy:

'start$mobile, mail$admini$počet užívateľov presiahol 2000$end'

4.2 Monitorovacie moduly

Monitorovací modul má za úlohu sledovať konkrétny typ dát a v prípade, že nastane očakávaná udalosť odošle správu Jadru. Potrebujeme monitorovaciemu modulu nejakým spôsobom definovať aký typ dát bude sledovať. Taktiež je dôležité určiť podmienku, ktorá vyvolá odoslanie správy a cieľovú skupinu, ktorej je správa určená. Na definovanie podmienok, typu monitorovaných dát a ostatných parametrov využijem konfiguračný súbor, s ktorým sa monitorovací modul bude spúšťať.

Konfiguračné súbory sú písané v YAML[3] formáte, určený pre serializáciu dát, ktorý je ľahko čitateľný pre ľudí a preto je vhodný a využívaný v konfiguračných súboroch. Pre

[3] http://yaml.org/

konfiguračné súbory sa zvykne používať ešte napríklad XML alebo JSON formát. V XML máme otváracía uzatváracía element, čo pri väčšom konfiguračnom súbore znamená minimálne dvojnásobok riadkov. JSON je efektívnejší ako formát XML v serializácii dát. Kvôli spomínanej čitateľnosti som uprednostnil YAML.

Konfiguračný súbor monitorovacích modulov musí obsahovať štyri základné časti. Niektoré časti sú podobné, niektoré obsahujú rovnaké kľúče. Prvou časťou je **core** zobrazené na obr. 4.2, kde vidíme formát, teda aké hodnoty prislúchajú daným kľúčom a príklad použitia. Táto časť konfiguračného súboru určuje adresu (*address*), port (*port*) a typ transportného protokolu (*proto*, tcp alebo udp) pre komunikáciu s jadrom aplikácie, ktoré prijíma a preposiela syslog správy. Zdrojom syslog správy bude samotný monitorovací modul. Táto časť bude rovnaká pre všetky monitorovacie moduly.

Formát	**Príklad**
core:	core:
address: IPv4 adresa	address: 127.0.0.1
port: číslo portu	port: 601
proto: tcp alebo udp	proto: tcp

Obr. 4.2: Ukážka spoločnej core časti v konfiguračnom súbore

Ďalšia spoločná časť pre všetky moduly je **notification**. V nej je potrebné uviesť spôsob odoslania správy (*module*, môže nadobúdať hodnotu mail, mobile alebo oboje oddelené čiarkou) o udalosti, teda splnenej podmienke z pravidla. Ešte potrebujeme definovať globálnu správu (*message*) a globálnu skupinu (group). Ich hodnoty sa využijú, ak pravidlo nemá uvedenú vlastnú správu alebo skupinu. Z obrázku 4.3 vidíme použitie na príklade a formát.

Formát	**Príklad**
notification:	notification:
module: mobile|mail|mobile, mail	module: mobile, mail
message: globálna správa	message: skontroluj web page
group: globálna skupina	group: admins

Obr. 4.3: Ukážka spoločnej notification časti v konfiguračnom súbore

Časť **source** už nie je rovnaká pre všetky moduly, avšak použité kľúče môžu byť podobné, hlavne u databázových modulov. Definuje zdroj dát, ktoré sa budú monitorovať. To znamená o aký typ dát sa jedná, ako k nim môžeme pristúpiť poprípade definuje prihlasovacie údaje, pomocou ktorých je možné sa k dátam dostať.

Poslednou základnou a spoločnou časťou monitorovacích modulov je **rules**, určená na definovanie pravidiel. Keď nastane podmienka definovaná v pravidle, tak sa odošle správa. Pravidiel môže byť definovaných viac, no nesmú mať rovnaký názov. V každom pravidle musí byť uvedená frekvencia (*frequency*) s číselnou hodnotou. Tá udáva v sekundách ako často sa má vykonať a porovnať akcia definovaná v pravidle. Pri všetkých monitorovacích moduloch môžeme v tejto časti uviesť konkrétnejšiu správu (*message*) a skupinu (*group*). V takom prípade sa uprednostní tento popis správy pred popisom v časti **notifications**.

Ešte je možné uviesť v pravidle kľúčové slovo *matchOnce*. To udáva, že keď udalosť nastane a odošle sa správa, už sa nebude pravidlo ďalej kontrolovať. Ak toto kľúčové slovo nie je uvedené a nastane očakávaná udalosť odošle sa správa. Aj keď je udalosť stále platná ďalšie správy už nebudú odoslané. Keď sa udalosť vráti do stavu nevyhovujúcemu podmienke znovu je možné odoslať upozornenie. Kľúčové slovo *matchOnce* by sme použili, ak by nás napríklad zaujímalo, kedy sa prekročí stanovená hranica používateľov. Hranica sa

raz prekročí a už to nie je potrebné kontrolovať. V inom prípade, keď chceme byť informovaný o tom, že nám prestane fungovať vybraná služba sa nám *matchOnce* nehodí. To nie je jednorázová udalosť, ale môže sa znova zopakovať

V nasledujúcich podsekciách budú popísané jednotlivé monitorovacie moduly a ich špecifické kľúče v rámci konfiguračného súboru častí **source** a **rules**.

4.2.1 Terminál

Pomocou terminálového monitorovacieho modulu vieme sledovať výsledky z terminálu po vykonaní požadovaného príkazu alebo zreťazených príkazov cez pipeline. Výhodou je možnosť napísania vlastného skriptu a tým pádom je možne monitorovať čokoľvek.

V konfiguračnom súbore pre časť **source** musí byť uvedený typ zdroja dát (*type*, vždy *cli*), príklad na obr. 4.4. To je skratka od command line interface čo v preklade znamená rozhranie príkazového riadku.

Formát	**Príklad**
source: type: cli	source: type: cli

Obr. 4.4: Ukážka source časti v konfiguračnom súbore pre terminál monitorovací modul

V **rules** časti konfiguračného súboru je povinné uviesť príkaz (*command*), ktorý sa bude vykonávať. Potrebujeme mu určiť frekvenciu (*frequency*), ako často sa má príkaz definovaný v pravidle vykonať. Ďalej je potrebné definovať čomu sa má rovnať (*equal*) výsledok po vykonaní príkazu alebo nemá (*notEqual*).

Pre pravidlo je možné uviesť voliteľné kľúče, ktorými určíme špecifickú hodnotu správy (*message*), vyberieme inú skupinu ako je určená globálne (*group*) alebo chceme, aby sa správa odoslala iba raz a ďalej sa stav neoveroval (*matchOnce*).

V obrázku 4.5 vidíme označené voliteľné kľúče znakom „*“ a príklad. Výsledkom príkazu z príkladu je počet všetkých procesov bežiacich na zariadení. Pri kľúči *equal* je uvedená hodnota 15. To znamená, že správa sa odošle, keď výsledná hodnota po vykonaní príkazu sa jej bude rovnať. Príkaz sa vykoná a overí každých 180 sekúnd (*frequency: 180*). Kvôli kľúču *matchOnce* po zhode výsledku z príkazu a hodnote pri kľúči *equal* sa už ďalej príkaz nebude vykonávať a upozornenie na udalosť nastane iba raz. Vygenerovaná správa je určená pre skupinu *testeri*.

Formát	**Príklad**
rules: názov_pravidla: command: shell príkaz notEqual|equal: porovnávaná hodnota frequency: čas v sekundách *matchOnce: *message: správa *group: názov cieľovej skupiny	rules: test: command: ps -A | wc -l equal: 15 frequency: 180 matchOnce: group: testeri

Obr. 4.5: Ukážka rules časti v konfiguračnom súbore pre terminál monitorovací modul

4.2.2 MySQL

MySQL monitorovací modul umožňuje vykonávanie príkazov nad týmto typom databázy. Najjednoduchšie je porovnávať ak návratová hodnota je iba jedna. Keď je očakávaná jedna hodnota z viacerých riadkov, tak je možné očakávané hodnoty oddeliť čiarkou. Napríklad majme tabuľku so stĺpcami majiteľ a zviera. Urobíme výber z tabuľky, kde budeme chcieť všetkých majiteľov čo majú mačku. Očakávané mená môžeme zapísať tak, že ich oddelíme čiarkou a medzerou. Príkazy, ktoré vracajú hodnoty z viacero stĺpcov vrátia výsledok jednotlivých hodnôt oddelených čiarkou a medzerou. Výsledky sú zoradené podľa toho ako sú zoradené jednotlivé stĺpce v databáze.

V konfiguračnom súbore pre MySQL monitorovací modul v časti **source** (pozri príklad na obrázku 4.6) musí byť uvedený typ zdroja (*type*, hodnota vždy database), typ databázy (*dbType* s hodnotou mysql), názov databázy (*dbName*), s ktorou sa bude pracovať. Pre prístup na server s databázou potrebujeme autorizačné údaje ako prihlasovacie meno (*userName*) a heslo *password*. Nakoniec potrebujeme adresu (*address*) a port (*port*) servera s databázou.

Formát

```
source:
 type: database
 dbType: mysql
 dbName: názov databázy
 userName: prihlasovacie meno
 password: heslo
 address: hostname alebo IPv4
 port: číslo portu
```

Príklad

```
source:
 type: database
 dbType: mysql
 dbName: tutDB
 userName: user
 password: pass123
 address: localhost
 port: 3306
```

Obr. 4.6: Ukážka source časti v konfiguračnom súbore pre MySQL monitorovací modul

Do pravidiel **rules** zadávame SQL dotaz (*query*), ktorý sa vykoná nad databázou a potrebujeme mu definovať ako často sa má vykonať a porovnať (*frequency*). Je potrebné určiť porovnávaciu hodnotu a či sa má rovnať (*equal*) výsledku alebo nemá (*notEqual*). Záleží čo nám viac vyhovuje. Je možné dopísať voliteľné kľúče pre správu (*message*), skupinu (*group*) a či má byť zhoda iba raz (*matchOnce*).

Formát

```
rules:
 názov_pravidla:
  query: sql dotaz
  notEqual|equal: porovnávané hodnoty
  frequency: čas v sekundách
  *matchOnce:
  *message: správa
  *group: názov cieľovej skupiny
```

Príklad

```
rules:
 test2:
  query: SELECT COUNT(nick) FROM users
  equal: 12
  frequency: 90
  message: správa z pravidla test2
  matchOnce:
```

Obr. 4.7: Ukážka rules časti v konfiguračnom súbore pre MySQL monitorovací modul

Na obrázku 4.7 vidíme príklad pravidla pre MySQL monitorovací modul. Zobrazený SQL dotaz počíta z tabuľky *users* počet prvkov v stĺpci *nick*. Výsledná hodnota dotazu je porovnávaná s číslom 12. To sa urobí raz za 90 sekúnd. Po zhode sa to už ďalej nebude kontrolovať a odoslaná správa bude „správa z pravidla test2“.

4.2.3 RocksDB

Pomocou RocksDB monitorovaciemu modulu je možné monitorovať hodnoty v RocksDB NoSQL databáze typu kľúč-hodnota. Monitorovací modul je určený pre prípady, keď je databáza uložená lokálne.

V konfiguračnom súbore pre časť **source** (príklad je možné vidieť na obr. 4.8) je potrebné uviesť typ zdroja dát (*type* s hodnotou database), typ databázy (*dbType*, vždy rocksdb) a cestu k databáze (*dbPath*).

Formát	**Príklad**
source: type: database dbType: rocksdb dbPath: cesta k databáze	source: type: database dbType: rocksdb dbPath: /home/new/test.db

Obr. 4.8: Ukážka source časti v konfiguračnom súbore pre RocksDB monitorovací modul

Časť **rules** pre RocksDB potrebuje informáciu o kľúči (*key*) z databázy, s ktorým sa bude pracovať. Následne musíme určiť s čím sa bude hodnota pri kľúči porovnávať a to podľa toho či nás zaujíma rovnosť (*equal*) hodnoty pri kľúči a zadanej hodnoty alebo nerovnosť (*notEqual*). Nakoniec potrebujeme nastaviť frekvenciu (*frequency*) v sekundách ako často sa má hodnota v *key* skontrolovať. Voliteľné kľúče sú v obrázku 4.9 uvedené so znakom „*“.

Formát	**Príklad**
rules: názov_pravidla: key: kľúč notEqual|equal: porovnávané hodnoty frequency: čas v sekundách *matchOnce: *message: správa *group: názov cieľovej skupiny	rules: test3: key: name notEqual: Honza frequency: 290

Obr. 4.9: Ukážka rules časti v konfiguračnom súbore pre RocksDB monitorovací modul

Obrázok ilustruje aj príklad použitia pravidla pre rocksdb monitorovací modul. V tomto konkrétnom prípade sa každých 290 sekún kontroluje kľúč *name* a porovná sa s hodnotou *Honza*. Ak sa to nerovná, bude odoslaná správa. Správa a skupina sa použije z **notification** časti. Po odoslaní správy sa bude pokračovať v kontrole. Bude potrebné, aby sa stav z vyhovujúceho podmienke pre odoslanie správy zmenil na nevyhovujúci a ak znovu bude podmienka pre odoslanie platná opäť sa odošle správa. Tento dej sa bude opakovať pokiaľ sa nepreruší monitorovací modul.

4.2.4 MongoDB

Aktuálne posledným monitorovacím modulom je MongoDB monitorovací modul. Slúži na sledovanie dát v MongoDB NoSQL databáze dokumentového typu. Konfiguračný súbor MongoDB monitorovacieho modulu v **source** časti musí mať nasledujúce údaje: typ zdroja dát (*type* s hodnotou database), typ databázy (*dbType* s hodnotou mongodb), názov databázy (*dbName*), adresu (*address*) a port (*port*) servera, kde je požadovaná databáza uložená. V prípade ak sa jedná o lokálnu nezabezpečenú databázu nie je potrebné uvádzať prihla-

sovacie meno (*userName*) a heslo (*password*), inak sú tieto autorizačné údaje umožňujúce prístup k databáze povinné.

Formát

```
source:
 type: database
 dbType: mongodb
 dbName: názov databázy
 userName: prihlasovacie meno
 password: heslo
 address: hostname alebo IPv4
 port: číslo portu
```

Príklad

```
source:
 type: database
 dbType: mongodb
 dbName: bp
 userName: user2
 password: 123pass
 address: localhost
 port: 27017
```

Obr. 4.10: Ukážka source časti v konfiguračnom súbore pre MongoDB monitorovací modul

Pre časť **rules** konfiguračného súboru je potrebné uviesť kolekciu (*collection*), s ktorou sa bude v rámci databázy pracovať. Potrebujeme definovať podľa akých kritérii sa má hľadať, keďže sa jedná o NoSQL databázu dokumentového typu hodnôt pre porovnanie môže byť viacej a sú štruktúrované ako kľúč-hodnota. Môžeme vyhľadávať jeden prvok (*findOne*) alebo ich vyhľadávať viacero (*find*). Keď vyhľadávame jeden prvok neudáva sa hodnota na porovnanie. V opačnom prípade, ak vyhľadávame viac prvkov, tak výsledkom je ich súčet a potrebujeme zadať akej hodnote sa to má rovnať (*count*) alebo nemá (*count!*). Ako pri každom pravidle je potrebné uviesť frekvenciu (*frequency*) a je možné použiť aj zvyšné voliteľné kľúče.

Na obrázku 4.11 vidíme použitie pravidla s kľúčom *find*. Vyhľadávať sa bude v kolekcii *users* podľa kľúča *name* s hodnotou *Michal* a kľúča *surname* s hodnotou *Danko*. Po vykonaní kontroly či sa v kolekcii *users* nachádzajú dokumenty s týmito hodnotami sa výsledné dokumenty spočítajú. Správa sa v tomto prípade vygeneruje, keď výsledok nie je rovný dvom. Kontrola sa vykoná každých 133 sekúnd až po ukončenie monitorovacieho modulu.

Formát A

```
rules:
 názov_pravidla:
  collection: názov kolekcie databázy
  find: vyhľadávacie kľúče
  count|count!: porovnávaná hodnota
  frequency: čas v sekundách
  *matchOnce:
  *message  správa
  *group: názov skupiny
```

Príklad A

```
rules:
 test4:
  collection: users
  find:
   name: Michal
   surname: Danko
  count!: 2
  frequency: 133
```

Obr. 4.11: Ukážka rules časti v konfiguračnom súbore pre MongoDB monitorovací modul

Použitie kľúča *findOne* ilustruje obr. 4.12. V uvedenom príklade sa pracuje s kolekciou *admins*, kde sa bude vyhľadávať jeden dokument čo nám určuje *findOne*. Prvé vyhľadávacie kritérium je kľúč *name* s hodnotou *Jan* a druhé *surname* s hodnotou *Kan*. Kontrola sa vykoná každých 290 sekúnd či sa aspoň jeden taký dokument v databáze nachádza. Ak áno odošle sa správa s textom „mongo message“. Kontrola pokračuje naďalej.

Formát B

```
rules:
 názov_pravidla:
  collection: názov kolekcie databázy
  findOne: vyhľadávacie kľúče
  frequency: čas v sekundách
  *matchOnce:
  *message: správa
  *group: názov skupiny
```

Príklad B

```
rules:
 test5:
  collection: admins
  findOne:
   name: Jan
   surname: Kan
  frequency: 290
  message: mongo message
```

Obr. 4.12: Ukážka rules časti v konfiguračnom súbore pre MongoDB monitorovací modul

4.3 Jadro

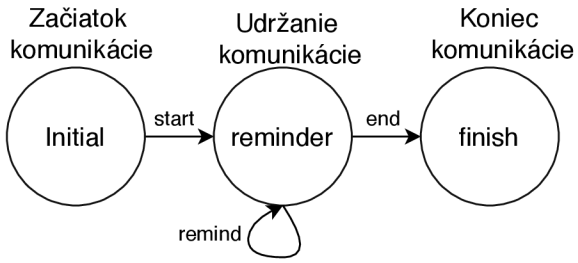
Úlohou jadra bude prijímať správy od monitorovacích modulov a preposielať ich odosielacím modulom, ktoré sa na jadro pripoja. Teda jadro sa bude po pripojení odosielacieho modulu správať ako syslog relay. Môže byť pripojený jeden alebo viac odosielacích modulov naraz. Takto bude možné v budúcnosti pridávať rôzne ako odosielacie, tak monitorovacie moduly. Jadro je schopné prijímať syslog správy, to znamená, že odosielateľom nemusí byť len jeden z monitorovacích modulov.

Jadro bude využívať existujúci program na preposielanie syslog správ. Tomuto programu bude potrebné meniť konfiguračný súbor. Pre jednoduché použitie využijem kontajerovú službu. Jadro bude možné spustiť na ľubovoľnom porte so zvolenou IP adresou.

Správy z odosielacieho modulu pre program prepisujúci konfiguračný súbor budú mať určitý formát. Na začiatku je spôsob komunikácie medzi preposielacím programom a odosielacím modulom TCP alebo UDP. Za tým je oddeľovací znak dolár a po ňom ide IP adresa odosielacieho modulu. Po IP adrese oddeľovací znak a následne číslo portu odosielacieho modulu. Po poslednom oddeľovacom znaku ide aktuálny stav.

<spôsob_doručenia>$<ip_adresa>$<port>$<stav>

Stav komunikácie je pri prvej správe *start*. Po odoslaní správy *start* sa každé tri minúty musí odosielací modul pripomenúť, že ešte chce dostáva správy so stavom *remind*. Každých päť minút sa skontroluje, kedy sa naposledy obdržala *remind* správa od odosielacích modulov a tie čo neposlali *remind* viac ako päť minút už nebudú dostávať správy od Jadra. Keď sa odosielací modul ukončí odošle sa správa so stavom *end* a ďalej už správy posielať nebude. Priebeh stavov je zobrazený na obrázku 4.13.



Obr. 4.13: Priebeh komunikácie odosielacieho modulu s rekonfigurákom

4.4 Odosielacie moduly

Odosielacie moduly sa starajú o odoslanie správy užívateľovi. V tejto práci sú navrhnuté dva odosielacie moduly. Jeden doručuje správy pomocou e-mailov a druhý je zameraný na notifikácie pre Android mobilné zariadenia. Odosielací modul sa po spustení pripojí na jadro a od neho bude dostávať správy.

Modulom potrebujeme dať vedieť kam sa majú pripájať, takže tie budú mať svoj konfiguračný súbor, kde budú všetky potrebné údaje. Moduly majú v konfiguračnom súbore spoločné 4 hlavné časti z toho 3 sú rovnaké. Prvou spoločnou a rovnakou časťou je **address**, kde je uvedená IP adresa (*ip*), port (*port*) na ktorom bude odosielací modul prijímať správy a protokol na transportnej vrstve (*proto, môže byť buď tcp alebo udp*).

Formát	**Príklad**
address:	address:
ip: IPv4 adresa	ip: 192.168.122.1
port: číslo portu	port: 8008
proto: tcp alebo udp	proto: udp

Obr. 4.14: Príklad časti address konfiguračného súboru pre odosielacie moduly

V druhej časti, **reconfig**, ktorá je spoločná a rovnaká pre oba moduly je uvedená adresa (*address*) a číslo portu (*port*) časti jadra, kam sa pripája odosielací modul. Príklad vidíme na obrázku 4.15, kde je uvedený aj formát.

Formát	**Príklad**
reconfig:	reconfig:
address: IPv4 adresa	address: 127.0.0.1
port: číslo portu	port: 8000

Obr. 4.15: Príklad časti reconfig konfiguračného súboru pre odosielacie moduly

Treťou časťou konfiguračného súboru je **groups**, ktorú majú oba odosielacie moduly, ale každý ju definuje inak. Bližšie informácie budú popísané pri jednotlivých moduloch.

V poslednej spoločnej časti **default** je názov skupiny z *groups*, ktorej sa odošle informácia o udalosti, keď príde syslog správa z iného zdroja ako sú monitorovacie moduly. Syslog správa z iného zdroja nebude mať formátovanú časť MSG podľa požiadaviek. Ak by tá časť bola naformátovaná odoslalo by sa to požadovanej skupine. Časť *groups* bude viac popísaná pri jednotlivých odosielacích moduloch a je to štvrtá spoločná časť.

Formát	**Príklad**
default: názov skupiny	default: admini

Obr. 4.16: Príklad časti default konfiguračného súboru pre odosielacie moduly

4.4.1 E-mail

E-mailový odosielací modul odosiela správy na e-mail. Na odoslanie e-mailu využije už existujúce riešenia. Prostredníctvom uvedeného účtu v konfiguračnom súbore sa odosielací modul pokúsi prihlásiť na SMTP server a odoslať elektronickú poštu.

Formát	**Príklad**
mail: username: e-mailová adresa password: heslo address: hostname alebo IPv4 port: číslo portu	mail: username: user@gmail.com password: nbu123 address: smtp.gmail.com port: 587

Obr. 4.17: Príklad časti mail konfiguračného súboru pre e-mail odosielací modul

Obsahom konfiguračného súboru je časť **mail**. V nej je uvedené prihlasovacie meno (*username*) a heslo (*password*) e-mailového účtu. Prihlasovacie meno (*username*) je zároveň adresa odosielateľa. Pre úspešné prihlásenie sa na účet potrebujeme vedieť aj na aký server sa máme prihlasovať. Preto je nevyhnutné zadať adresu (*address*) a číslo portu (*port*) servera prostredníctvom, ktorého sa pošta bude odosielať.

Formát	**Príklad**
individuals: - jednotlivé e-mailové adresy	individuals: - admini - testeri - zakaznici

Obr. 4.18: Príklad časti individuals konfiguračného súboru pre e-mail odosielací modul

V nepovinnej časti **individuals**, špecifickej iba pre e-mail odosielací modul, je uvedený zoznam e-mailových adries jednotlivcov. Je to možné využiť, keď nechceme odoslať upozornenie viacerým ľuďom. Namiesto názvu skupiny v správe bude uvedená konkrétna e-mailová adresa.

Formát	**Príklad**
groups: názov_skupiny: - jednotlivé e-mailové adresy	groups: admini: - mail1@gmail.com testeri: - tester1@domena.sk - tester2@domena.cz

Obr. 4.19: Príklad časti groups konfiguračného súboru pre e-mail odosielací modul

Poslednou časťou konfiguračného súboru je **groups**. Pre mailový modul sa uvedie názov skupiny a zoznam e-mailových adries (pozri obrázok 4.19). Na obrázku sú uvedené dve skupiny admini a testeri. Skupina admini obsahujú jednu e-mailovú adresu a testeri obsahujú dve.

4.4.2 Android notifikácie

Upozornenia na Android mobilné zariadenia sa vytvoria pomocou mobilného odosielacieho modulu. Android zariadenia dostanú správu pomocou služby firebase cloud messaging a push notifikácie. Bude možné odosielať správu skupinám zariadení, čiže môže byť doručená na viac zariadení naraz.

Mobilný odosielací modul potrebuje mať v konfiguračnom súbore informácie pre firebase službu. Budú uvedené pod kľúčom **firebase**. Potrebujeme definovať API kľúč (*apiKey*), potrebný na autentizáciu pri odosielaní požiadaviek na firebase cloud messaging. Pre prácu

s firebase realtime databázou potrebujeme autorizačný súbor (*file*). Ešte je potrebné stanoviť URL adresu firebase NoSQL databázy (*databaseUrl*). Príklad definovanej časti **firebase** je zobrazený na obr. 4.20. API kľúč v príklad nie je uvedený celý.

Formát	**Príklad**
firebase:	firebase:
apiKey: firebase api kľúč	apiKey: AIzaSyDaGkDRmUF0...
file: cesta k autorizačnému súboru	file: /home/michal/creds.json
databaseUrl: url do firebase DB	databaseUrl: https://bp.firebaseio.com/

Obr. 4.20: Príklad časti firebase konfiguračného súboru pre mobilný odosielací modul

Skupiny pre mobilný odosielací modul sú uvedené v časti **groups** ako zoznam pozri obrázok 4.21. Na ne sa budú môcť prihlasovať užívatelia mobilnou aplikáciou.

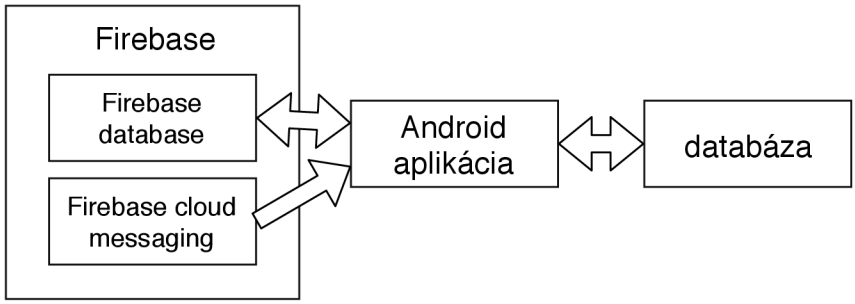
Formát	**Príklad**
groups:	groups:
- jednotlivé skupiny	- admins
	- testers
	- workers

Obr. 4.21: Príklad časti groups konfiguračného súboru pre mobilný odosielací modul

4.5 Návrh Android aplikácie

Úlohou Android aplikácie bude možnosť prihlasovať sa a odhlasovať zo skupín, ktorým sú určené správy vygenerované z monitorovacích modulov. Ďalej samotné prijímanie správ a zobrazovanie notifikácii. Aplikácia bude ukladať staré správy a umožní ich zobrazovať a mazať. Nové správy by mali byť označené tak, aby sa dali rozlíšiť od tých starších. Vhodné by bolo zobrazovať počet nových správ.

Pre získanie zoznamu skupín, na ktoré sa bude užívateľ prihlasovať potrebujeme prístup k firebase realtime database. Z nej si aplikácia stiahne všetky skupiny. Ak sa pridá nová skupina počas behu aplikácie, zariadenie si ju stiahne. Správy sa do smartfónu dostanú prostredníctvom push notifikácie z firebase cloud messaging. Tieto správy po prijatí budú uložené v lokálnej databáze. Z toho nám vychádza základná architektúra aplikácie zobrazená na obrázku 4.22.



Obr. 4.22: Návrh Android aplikácie

4.5.1 Návrh zobrazenia notifikácie

Notifikácia o prijatí správy by sa mala zobraziť ako vysúvateľné okno, aby výrazne upovedomila užívateľa. Toto okno prekryje aktívnu Activity z vrchnej časti a neovplyvní jej stav, takže je možné v Activity ďalej pracovať. Ak na to užívateľ nezareaguje mala by sa zobraziť medzi ostatnými notifikáciami. Na obrázku 4.23 vidíme návrh zobrazenia notifikácie. Je tam obsah správy a možnosť ju priamo odstrániť.

Obr. 4.23: Návrh zobrazenia notifikácie

4.5.2 Návrh užívateľského rozhrania

Podľa toho na čo má aplikácia slúžiť, teda zobrazovanie skupín, možnosť prihlásenia a odhlásenia sa zo skupiny, zobrazovanie notifikácii, prijímanie správ, ukladanie správ, bolo potrebné navrhnúť prijateľné užívateľské rozhranie. Z daných požiadaviek mi vyšlo najvhodnejšie, aby aplikácia mala dve záložky (karty). Jednu určenú pre správu prihlasovania a odhlasovania sa zo skupín a druhú pre odznačovanie, mazanie a zobrazovanie prijatých správ. Návrh oboch záložiek môžeme vidieť na obrázku 4.24.

Obr. 4.24: Návrh užívateľského rozhrania

V záložke pre správu skupín bude zoznam jednotlivých skupín s možnosťou označenia skupiny, keď máte záujem prijímať správy určené pre ňu. V spodnej časti je umiestnené tlačidlo s nápisom **Save**. To bude slúžiť pre potvrdenie, že chceme odoberať správy označených skupín.

V druhej záložke bude zoznam správ zobrazený chronologicky. Teda najnovšie budú na začiatku a najstaršie na konci. V hornej lište, pri názve druhého tabu **Notifications** bude, v prípade ak sú nové správy, zobrazený zelený kruh s počtom nových správ. Nové, neodznačené správy budú zvýraznené zeleným kruhom a pri každej bude možnosť jej trvalého odstránenia.

Kapitola 5

Implementácia

Začiatok kapitoly sa zaoberá implementáciou serverovej časti, ktorá sa venuje monitorovaniu a zabezpečeniu odoslania správ. Pod serverovú časť spadajú odosielacie, monitorovacie moduly a jadro. Následne kapitola popisuje vývoj mobilnej Android aplikácie pre obdržiavanie správ.

5.1 Serverová časť

Ako implementačný jazyk bol zvolený Python 3.5.2[1] a serverová časť bola vyvíjaná vo vývojovom prostredí PyCharm IDE[2] od spoločnosti JetBrains. Pri vývoji som používal program **virtualenv**[3]. Pomocou neho som si vytvoril virtuálne Python prostredie, kde som si postupne inštaloval použité knižnice v rámci projektu. Výhodou toho je, že po implementácii som si jedným príkazom vypísal všetky potrebné knižnice. Pomocou virtuálneho prostredia máme potrebné knižnice a vhodnú verziu Python interprétu zvlášť pre jednotlivé projekty. Serverová časť je podľa návrhu v kapitole 4 rozdelená do troch častí.

5.1.1 Jadro

Najprv si rozoberieme **jadro**, lebo to budú využívať zvyšné monitorovacie a odosielacie moduly, ktorých implementáciu si popíšeme následne. Jadro má mať správanie ako syslog relay. Na to využijeme už existujúci open-source program syslog-ng [4]. Syslog-ng vie zachytávať syslog správy. V jeho konfiguračnom súbore sa nastaví odkiaľ chceme zachytávať syslog správy a kam sa uložia respektíve prepošlú.

Keďže chceme, aby sa na jadro mohli dynamicky pripájať a odpájať odosielacie moduly, potrebujeme program, ktorý bude tieto moduly pridávať a odoberať z konfiguračného súboru syslog-ng. Tento program si nazvem **rekonfigurák**. Odosielacie moduly sa naň pripoja, keď budú chcieť dostávať správy. Po pripojení modulu ho rekonfigurák pridá do konfiguračného súboru syslog-ng a vykoná príkaz, aby si syslog-ng znovu načítal novú konfiguráciu. Rekonfigurák sa bude správať ako konkurentný TCP server, vo svojom konfiguračnom súbore bude mať uvedený port, na ktorý sa naviaže a cestu ku konfiguračnému súboru syslog-ng.

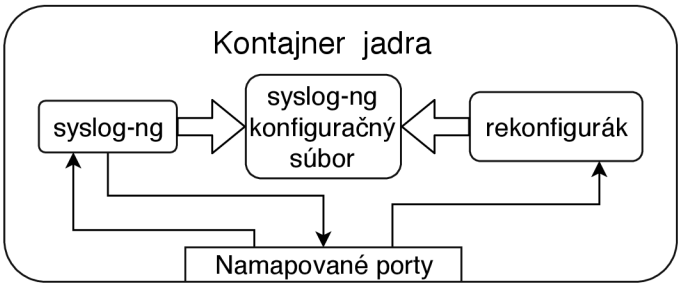
[1] https://www.python.org/downloads/release/python-352/
[2] https://www.jetbrains.com/pycharm/
[3] https://virtualenv.pypa.io/en/stable/
[4] https://syslog-ng.com/open-source-log-management

Pre jednoduché použitie jadra využijem Docker kontajner[5]. Kontajnerové služby sú v dnešnej dobe veľmi populárne. Vývojári vytvoria obraz s nastaveným programom a všetkými potrebnými knižnicami, poprípade verziou jazyka, ktorú používajú. Potom stačí, aby na serveri, kde chcú službu prevádzkovať, nainštalovali Docker a spustili tam z obrazu kontajner.

Kontajner je samostatný spustiteľný balík so softvérom, ktorý má všetko potrebné na spustenie. Kontajner izoluje softvér, ktorý v ňom je od prostredia, kde kontajner beží. Docker funguje na systéme Linux, Windows aj macOS, čiže zaobalením programu do docker obrazu sa mu zabezpečí multiplatformovosť. Kontajner je primárne určený na jednu službu. Pri spustení kontajneru nastavíme, aby nasadil naše súbory a už len stačí správne namapovať porty. Obrazy je možné vytvoriť aj vlastné. V jednom kontajneri môže bežať webová služba a v ďalšom databáza. Je možné, aby viacero kontajnerov medzi sebou komunikovalo.

V prípade implementácie jadra som sa snažil, aby v kontajneri bežali dve služby a vytvorili ako celok jadro serverovej časti. Jednou službou je rekonfigurák a druhou syslog-ng. Keďže syslog-ng závisí od rekonfiguráku bolo vhodné ich nechať v jednom kontajneri spolu. Pretože kontajner je určený primárne na jednu službu bolo potrebné vyriešiť ako ich v rámci kontajneru spustiť viac. Problém vyriešil správca procesov supervisor[6]. Prostredníctvom neho, som nastavil, ktoré procesy sa majú po zapnutí kontajneru spustiť a pri ukončení kontajneru sa postará aj o ukončenie procesov.

Správanie vytvoreného kontajneru jadra je znázornené na obrázku 5.1. Rekonfigurák prijíma spojenia na porte 8000. Po vytvorení spojenia a prijatí správy od odosielacieho modulu prepisuje konfiguračný súbor a vykoná príkaz, aby si syslog-ng znovu načítal konfiguračný súbor. Syslog-ng prijíma syslog správy na porte 514 pre UDP a 601 pre TCP. Porty je potrebné pri spustení namapovať na porty počítača, aby moholi služby komunikovať s okolitým svetom. To znamená, že môžeme využiť ľubovoľné porty.



Obr. 5.1: Docker kontajner jadra

5.1.2 Monitorovacie moduly

Úlohou monitorovacích modulov je odoslať syslog správu, ak nastane predom definovaná udalosť. Každý modul je samostatný Python súbor. Monitorovacie moduly majú spoločný súbor s funkciami, ktoré využívajú.

Po spustení je potrebné skontrolovať či sú v konfiguračnom súbore informácie pre pripojenie sa k jadru, pravidlá, definovaný typ zdroja, všeobecná správa s cieľovou skupinou a typom odoslania správy. Jednotlivé pravidlá sa spúšťajú v samostatných vláknach. Niektoré

[5] https://www.docker.com/what-docker
[6] http://supervisord.org/running.html#running-supervisord

pravidlá majú definovanú podmienku, že sa má kontrolovať iba po prvú zhodu pomocou kľúču **matchOnce**. V takom prípade sa vlákno po odoslaní správy ukončí. Ak podmienka **matchOnce** nie je nastavená, pri prvej zhode s očakávanou hodnotou (po odoslaní správy) sa nastaví odosielacia značka na *false*. Keď sa hodnota dostane do stavu, že opäť nevyhovuje očakávanej hodnote odosielacia odosielacia značka sa nastaví späť na *true* a môže znovu odoslať správu o zmene stavu. Teda v tom istom monitore môže bežať ďalšie vlákno až po ukončenie programu.

Správy sú odosielané pomocou Python knižnice logging[7]. Z logging knižnice som použil syslog handler, ktorý sa dá nastaviť, aby odosielal správy na určitú adresu a port. Syslog handler sa priradí inštancii loggeru z knižnice logging a generovanie syslog správ je už úplne jednoduché. Pre odoslanie správy stačí zavolať metódu, napríklad debug, inštancie logger so správou ako parametrom.

Jednotlivé moduly využívajú rôzne knižnice pre prácu so svojim zdrojom dát a taktiež iný spôsob získavania dát:

- **Monitorovací modul terminál** využíva knižnicu subprocess[8] na vykonanie príkazu z konfiguračného súboru. Je to nastavené tak, aby monitorovací modul vedel o návratovej hodnote po vykonaní príkazu. Z návratovej hodnoty sa odstráni posledný znak nového riadku.

- **MySQL monitorovací modul** pre prácu s databázou používa knižnicu MySQLdb[9]. Pomocou knižnice je jednoduché sa pripojiť k databáze podľa informácii z konfiguračného súboru. Modul sa pripojí k databáze a vykoná príkaz nad tabuľkou daný v pravidle, ukončí spojenie s databázou a vráti výsledok operácie. Ten sa upraví do formy hodnôt oddelených čiarkou a medzerou v poradí podľa riadkov. Ak je vybratých viac stĺpcov, tak je to podľa poradia stĺpcov a riadkov.

- **RocksDB monitorovací modul** pracuje s lokálnou databázou. Pre prácu s ňou využíva knižnicu rocksdb[10]. Vytvorí sa inštancia s databázou uvedenou v konfiguračnom súbore a po zvyšok programu sa s ňou pracuje. Výsledok z databázy dostaneme podľa kľúča v pravidle z konfiguračného súboru.

- **MongoDB monitorovací modul** pracuje ako s lokálnou, tak aj so vzdialenou databázou. Má oficiálnu knižnicu pymongo[11] určenú pre Python. Ako pri rocksDB monitorovacom module, tak aj tu stačí vytvoriť na začiatku inštanciu s databázou a môžeme ju využívať po zvyšok programu. Dáta z databázy získavame pomocou kľúčov a ich hodnôt definovaných v konfiguračnom súbore.

5.1.3 Odosielacie moduly

Odosielacie moduly zabezpečujú odosielanie správ od monitorovacích modulov zadaným skupinám. Každý modul je samostatný Python súbor. V spoločnom súbore majú funkcie, ktoré oba moduly využívajú.

Odosielací modul funguje ako server a klient. Klient odošle rekonfiguráku informácie o svojom serveri. Teda IP adresu, typ spojenia a číslo portu. Táto komunikácia je popísaná

[7] https://doc.bccnsoft.com/docs/python-3.5.2-docs-html/library/logging.html
[8] https://doc.bccnsoft.com/docs/python-3.5.2-docs-html/library/subprocess.html
[9] http://mysql-python.sourceforge.net/MySQLdb.html
[10] http://python-rocksdb.readthedocs.io/en/latest/
[11] https://api.mongodb.com/python/current/

v sekcii 4.3. Keď syslog-ng dostane správu prepošle ju odosielaciemu modulu. Odosielací modul dostane tú istú správu čo syslog-ng. Po jej prijatí si musí skontrolovať či je určená pre neho. Túto informáciu zistí v druhej časti správy za hodnotou „**start$**“, kde je uvedené akému odosielaciemu modulu je určená.

E-mailový odosielací modul pošle pomocou knižnice **smtplip**[12] správu cieľovej skupine. Po príchode syslog správy sa jej časť MSG spracuje. Vytvorí sa zoznam s troma prvkami (napr. ["mobile, mail", "názov_skupiny", "správa"]). Prvý prvok je akým odosielacím modulom sa má správa odoslať. Ak sa tam nachádza mail, tak sa kontroluje či cieľová skupina, ktorá je na prvom indexe zoznamu, je v konfiguračnom súbore medzi groups. Keď sa tam skupina nenachádza kontroluje sa ešte časť individuals, kde nie sú skupiny mailových adries, ale jednotlivé adresy. Potom sa pomocou údajov zadaných v konfiguračnom súbore odosielací modul prihlási na mailový účet, z ktorého sa maily odošlú. Iteruje sa cez celý zoznam mailových adries a každej sa odošle správa, ktorá je na druhom indexe zoznamu. Po odoslaní všetkých správ sa spojenie so SMTP serverom ukončí.

Android odosielací modul využíva oficiálnu knižnicu firebase_admin[13] pre prístup k firebase realtime databáze a knižnicu **pyFCM**[14] pre zjednodušenie komunikácie s firebase cloud messaging. Pri spustení tohto modulu sa všetky skupiny uvedené v konfiguračnom súbore skontrolujú či nie sú uložené vo firebase realtime databáte (FCM). Tie čo tam sú sa ignorujú a pridajú sa chýbajúce skupiny do databázy. To z dôvodu, aby sa v databázi nevyskytovali duplikáty a boli v nej všetky skupiny.

Po obdržaní syslog správy si Android odosielací modul, podobne ako e-mail odosielací modul, najprv skontroluje komu je správa určená. Ak obdržaná správa je určená pre neho, spracuje sa do zoznamu o troch prvkoch. Jedným z prvkov zoznamu je názov skupiny. Mobilný odosielací modul sa pokúsi dostať do časti databázy, kde by mal byť uvedený zoznam unikátnych tokenov prihlásených zariadení. Obdrží zoznam a pomocou pyFCM kontaktuje FCM, ktorý vygeneruje push notifikáciu pre všetky zariadenia uvedené v zozname. Dotaz na skupiny sa vykonáva vždy pred odoslaním požiadavku na FCM, ak by sa pripojilo nové zariadenie.

5.2 Android aplikácia

Pri vývoji aplikácie som používal Android studio IDE[15], ktorého súčasťou je správca virtuálnych prostredí. Pomocou neho si jednoducho môžeme vytvoriť virtuálne zariadenia, na ktorých sa dá aplikácia spúšťať a testovať. Android aplikáciu som implementoval v jazyku Java.

5.2.1 Activities Android aplikácie

Aplikácia má dve Activity. Jednu na načítanie dát z firebase, ktorá zobrazuje logo aplikácie a animáciu načitávania. Môžeme ju vidieť na obrázku 5.2 ako prvú zľava. Druhá activity zobrazuje dva fragmenty. Tieto fragmenty sa správajú ako záložky vďaka nástroju *view pager*, ktorý je špeciálny typ zobrazenia určený na to.

Prvý fragment je na obrázku 5.2 ilustrovaný v strede. Zobrazuje názvy skupín stiahnuté z firebase realtime databázy. Tieto skupiny sa načítajú počas zobrazovania prvej Activity

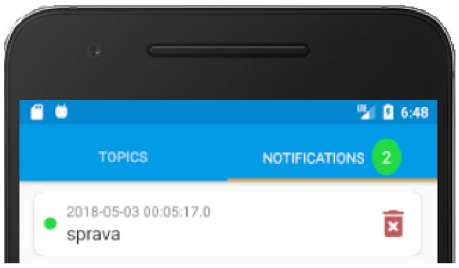
[12] https://doc.bccnsoft.com/docs/python-3.5.2-docs-html/library/smtplib.html
[13] https://firebase.google.com/docs/admin/setup
[14] https://pypi.org/project/pyfcm/1.2.4/
[15] https://developer.android.com/studio/

a vytvorí sa listener na referenciu databázy. Pomocou nástroja *recycler view*, ktorý slúži na zobrazovanie zoznamu dát sa vykresľujú jednotlivé skupiny v užívateľskom rozhraní. Na jednotlivé skupiny je možné sa prihlásiť. Prihlásenie znamená uloženie pod danú skupinu do firebase realtime databázy jedinečný token zariadenia.

Obr. 5.2: Zobrazenie notifikácie v Android smartfóne

Na začiatku, po spustení aplikácie sa načítajú všetky dáta, a keď sa pri behu aplikácie vo firebase databáze zmenia skupiny rovno sa načítajú dáta aj do aplikácie a zobrazia sa aktuálne skupiny. Aplikácia používa lokálnu SQLite databázu pre ukladanie správ a skupín. Skupiny sa ukladajú z dôvodu, aby aplikácia vedela na ktorú skupinu je prihlásená a zobrazila ju označenú v užívateľskom rozhraní.

V druhej záložke sú v *recycler view* zobrazené správy z lokálnej databázy. Správa má v databáze uložené či je nová. Toto označenie má iba na začiatku a keď na správu užívateľ stlačí, odznačí sa a už sa nebude zobrazovať so zeleným kruhom. Kliknutím na kôš sa správa odstráni z databázy.



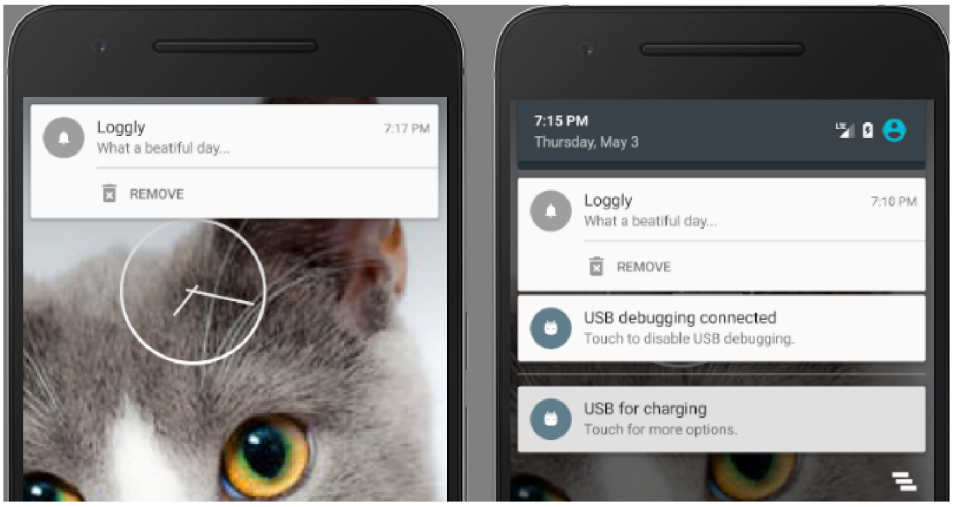
Obr. 5.3: Zobrazenie správy v Android aplikácii

Keď je aplikácia spustená, zobrazí sa označená nová správa v užívateľskom rozhraní. Ak aplikácia nie je spustená na popredí, vytvorí sa notifikácia. Na obrázku 5.3 vidíme pri druhej záložke číslo dva v zelenom kruhu. To značí, že sú dve nové, neodznačené správy. V spodnej časti obrázku je zobrazený obsah jednej neodznačenej správy.

5.2.2 Services Android aplikácie

Aplikácia pre svoje fungovanie potrebuje, aby na pozadí bežal vlastný service, ktorý prijíma správy vygenerované monitorovacími modulmi čo sa prijme vo forme push notifikácie. Service beží aj keď aplikácia nie je zapnutá. Zapne sa pri naštartovaní zariadenia. Informáciu o tom, že sa má zapnúť dostane od broadcast reciever, ktorý dáva pozor na to či sa zariadenie zaplo. Keď service obdrží správu, tak ju uloží do lokálnej databázy. Ak aplikácia nebeží na popredí service vytvorí notifikáciu, ktorá sa užívateľovi zobrazí.

Príklad vygenerovanej notifikácie vidíme na obrázku 5.4. Zľava v prvej časti obrázku je zobrazená vyskakovacia notifikácia, ktorá prekryje aj aktuálne používanú aplikáciu. Po určitej dobe, ak na vyskakovacie okno užívateľ nezareaguje notifikácia sa odloží do panelu s notifikáciami (notification panel). V ňom je možné notifikáciu odstrániť alebo na ňu reagovať príslušnými možnosťami. V našom prípade sú možnosti dve a to buď zmazať správu z databázy pomocou tlačidla *REMOVE* alebo stlačiť notifikáciu a vtedy sa otvorí aplikácia na druhom fragmente.



Obr. 5.4: Zobrazenie notifikácie v Android smartfóne

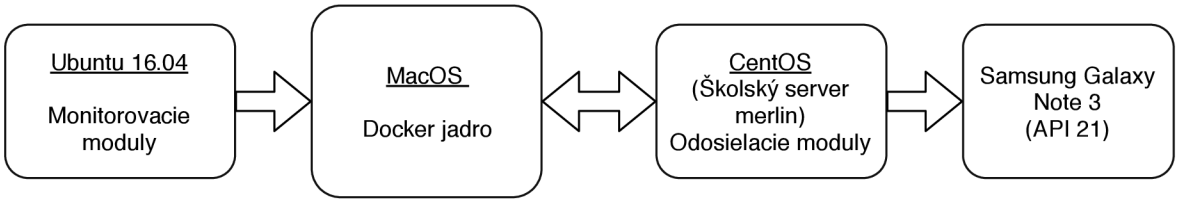
Aplikácia sa otvorí vďaka tomu, že pri vytváraní notifikácie v service sa jej pridá *Intent*, ktorý otvorí konkrétnu *Activity*. Správanie tlačidla *REMOVE* je tiež definované pri vytváraní notifikácie. Nastaví sa príslušná ikona a text, ktoré sa na tlačidle zobrazia. Po jeho stlačení sa odošle lokálny broadcast, ktorý obdrží broadcast reciever a spustí service na odstránenie správy z databázy.

Kapitola 6

Testovanie

6.1 Testovacie prostredie

Pri implementácii boli všetky časti aplikácie spúšťané na jednom počítači, ale aplikácia je navrhnutá tak, že jednotlivé časti môžu bežať na rôznych počítačoch. To je dôvod, prečo som sa rozhodol pri testovaní pustiť jednotlivé časti na rôznych strojoch. Z obrázku 6.1 vidíme, ako bude aplikácia rozdelená pre testovacie účely. Monitorovacie moduly pobežia na Ubuntu 16.04 (môj počítač), docker image jadra sa spustí na MacOS (notebook) a odosielacie moduly budú na školskom serveri *merlin*.



Obr. 6.1: Testovacie prostredie

V mojom zariadení som už mal nainštalované a vytvorené databázy, preto som ho zvolil pre spúšťanie monitorovacích modulov. Na počítači s MacOS bol *docker* a *docker-compose* už nainštalovaný. Stačilo vykonať príkaz **docker-compose up** v priečinku s docker súbormi pre jadro aplikácie a vytvoril sa image, spustil sa aj kontajner. Na školskom serveri bol dostupný program **virtualenv**. Pomocou neho som vytvoril nové virtuálne Python prostredie, kde som nainštaloval potrebné knižnice pre chod odosielacích modulov.

Android aplikácia bola spustená na smartfóne Samsung Galaxy Note 3 s Lollipop verziou Androidu API 21. Zobrazovanie notifikácia som chcel odskúšať aj na smart hodinky a na to som použil virtuálne zariadenie s verziou Androidu Marshmallow API 23. Virtuálne smart hodinky som prepojil so smartfónom, na ktorom bola aplikácia nainštalovaná. Pre sťahovanie elektronickej pošty som použil Mozzila Thunderbird.

6.2 Funkčné testovanie

Pre testovanie je vhodné zvoliť určitý postup. Preto som si pripravil body, ktoré by testovanie malo pokryť:

- zapojenie oboch odosielacích modulov s rovnakým transportným protokolom (TCP alebo UDP);
- zapojenie oboch odosielacích modulov s rôznym transportným protokolom;
- vypnutie jedného odosielacieho modulu a jeden ostane aktívny;
- pripojenie vypnutého odosielacieho modulu;
- odpojenie aktívneho odosielacieho modulu;
- odoslanie správy z každého monitorovacieho modulu v priebehu vykonávania uvedených bodov.

Najprv bolo potrebné zapnúť jadro, aby sa mali na čo pripájať odosielacie moduly. Následne som na školskom serveri *merlin* zapol oba odosielacie moduly s nastavením v konfiguračnom súbore na TCP komunikáciu.

Prvý spustený modul bol MySQL monitorovací modul. Bol nastavený na sledovanie lokálnej databázy a s odoslaním správy pre oba odosielacie moduly cez TCP spojenie na jadro. Konfiguračný súbor obsahoval dve pravidlá. Prvé pravidlo kontrolovalo počet mien či sa rovná šiestim a odoslalo správu zakaždým, keď nastala udalosť. Druhé pravidlo odoslalo správu iba raz a to keď sa primárny kľúč nerovnal hodnote jedenásť.

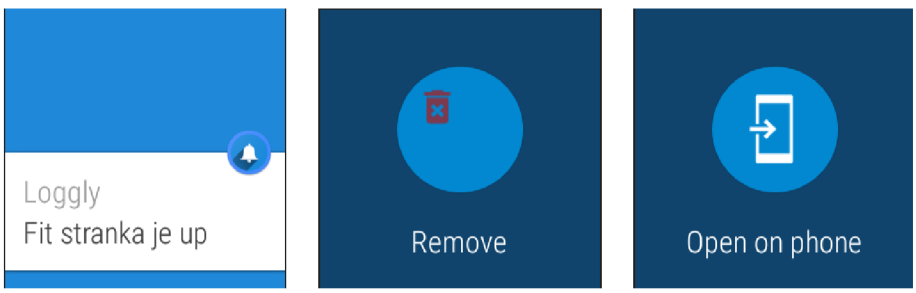
Po ukončení MySQL monitorovacieho modulu som vypol e-mail odosielací modul a spustil RocksDB monitorovanie. To bolo nastavené na lokálnu RocksDB databázu s TCP spojením na jadro. Konfiguračný súbor obsahoval dve pravidlá, ktoré odoslali správu raz a to vtedy, keď nastala udalosť.

Opäť som zapojil e-mail odosielací modul a spúšťal terminál monitorovací modul. Ten mal v konfiguračnom súbore nastavené TCP spojenie s jadrom a definované dve pravidlá. Prvé pravidlo kontrolovalo či je stránka *www.fit.vutb.cz* dostupná. Druhé pravidlo kontrolovalo či je spustený mailový klient.

Následne som vypol mobilný monitorovací modul a zapol ho s nastavením na UDP komunikáciu. Ako posledný som púšťal mongoDB monitorovací modul nastavený na UDP komunikáciu s jadrom. Mal definované dve pravidlá, ktoré odoslali správu raz.

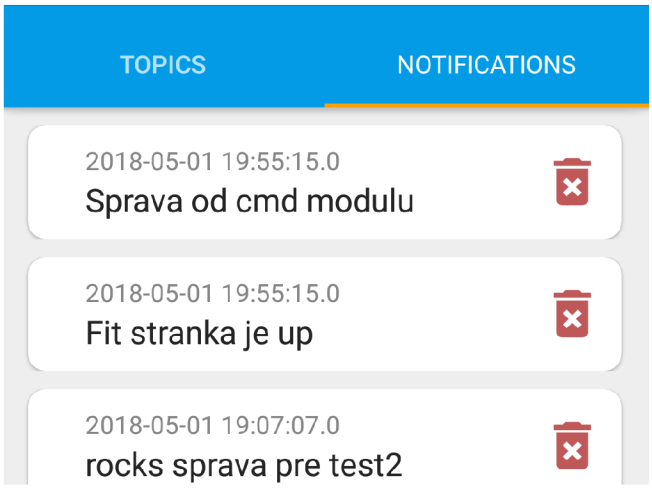
6.3 Zobrazenie výstupov

Po každom spustení monitorovacieho modulu sa doručili správy k odosielacím modulom. Odosielacie moduly následne zabezpečili doručenie pomocou e-mailu alebo firebase push notifikácie do Android zariadenia alebo elektronickej pošty.



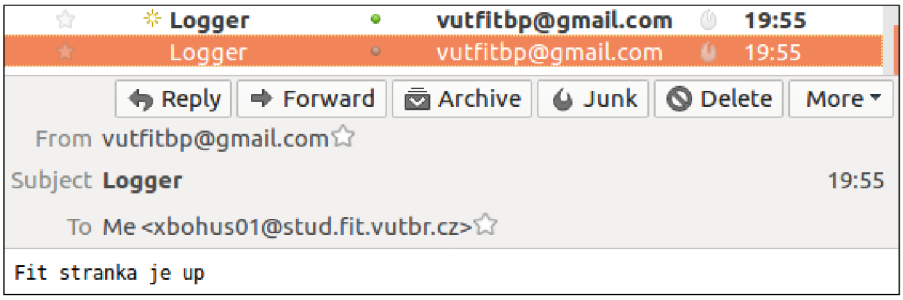
Obr. 6.2: Prijatá notifikácia zobrazená na smart hodinkách

Na obrázku 6.2 vidíme prijatú notifikáciu na smart hodinkách pripojených k mobilnému telefónu. V prvej časti je zobrazený názov aplikácie a správa. V druhej časti je možnosť zmazania notifikácie, bez potreby otvárania telefónu. V poslednej, tretej časti je možnosť otvoriť aplikáciu v telefóne. Prijaté notifikácie zobrazené v aplikácii na smartfóne sú v obrázku 6.3. Môžeme vidieť dve správy z monitorovacích modulov (terminál, rocksDB).



Obr. 6.3: Prijaté notifikácie na Android zariadenie

Výpis prijatých správ z e-mailového klienta a konkrétnu správu ukazuje obrázok 6.4. Správy sú doručené z mailu zadaného v konfiguráku a predmetom správy je stále Logger. Posledné dve správy boli vygenerované pre oba odosielacie moduly, tak na obrázku 6.4 a 6.3 je zobrazený rovnaký čas prijatia.



Obr. 6.4: Prijaté e-maily

Kapitola 7

Záver

V práci som sa venoval informáciám o platforme Android. Bolo to dôležité kvôli naplánovanej mobilnej aplikácii na odoberanie notifikácii pre tento systém. S touto platformou je úzko spätý Google Firebase, ktorý svojimi možnosťami uľahčuje vývoj mobilných aplikácii. Vývojára odbremení od implementácie vlastného databázového servera, dokáže generovať notifikácie a správy do mobilných zariadení.

Pre potreby monitorovania dát bolo nevyhnutné identifikovať vhodné zdroje. Z relačných databáz som zvolil MySQL ako zástupcu. Z nerelačných databáz, ktoré sú v dnešnej dobe v obľube som použil na monitorovanie MongoDB. Tá je dokumentového typu a z NoSQL databáz je najpoužívanejšia. Ďalší zástupca nerelačných databáz zvolený na monitorovanie je typu kľúč hodnota RocksDB. Spúšťanie príkazov v termináli otvára množstvo možností na monitorovanie. Napríklad sledovanie bežiacich procesov alebo napísaný skript na overovanie dostupnosti nejakej služby.

V návrhu serverovej časti som rozobral na aké časti bude systém rozdelený, teda monitorovacie moduly, jadro a odosielacie moduly. Monitorovacích modulov môže byť viac. Aktuálne sú štyri a navyše je možnosť jadru odosielať správu aj z akéhokoľvek zdroja generujúceho syslog správy. Tá bude preposlaná prednastavenej skupine. Návrh zahrňuje dva odosielacie moduly. Jadro je prispôsobené pre komunikácia s viacerými odosielacími modulmi. Každý modul, vrátane jadra, má svoj konfiguračný súbor a má v ňom podstatné informácie pre svoje fungovanie.

Mobilná Android aplikácia dokáže zobrazovať skupiny pridané do firebase realtime databázy. Na tieto skupiny je možné sa pomocou aplikácie prihlasovať. Na ktoré skupiny sa užívateľ prihlási, z tých mu budú prichádzať správy. Správy sú ukladané v zariadení, vďaka čomu sa k ním dá spätne vrátiť.

Výsledok práce som otestoval. Monitorovací systém bežal na troch zariadeniach a v smartfóne som mal nainštalovanú Android aplikáciu. K Android aplikácii som pripojil virtuálne hodinky, na ktorých sa notifikácie tiež zobrazovali.

V budúcnosti je možnosť pridáva ďalšie monitorovacie moduly, ktoré budú sledovať ďalší typ zdroja dát. Taktiež je možné rozšíriť a pridať odosielacie moduly. Napríklad by bolo možné pridať GSM modul, ktorý by odosielal SMS správy. Android aplikácii by bolo vhodné pridať možnosť zmeny zdrojovej databázy. Tým pádom by nebola určená iba pre úzku skupinu ľudí, ale všeobecne použiteľná.

Literatúra

[1] Documentation | Firebase. [Online; navštívené 17.01.2018].
URL https://firebase.google.com/docs

[2] Platform Architecture | Android Developers. [Online; navštívené 13.01.2018].
URL https://developer.android.com/guide/platform/index.html

[3] Gargenta, M.: *Learning Android*. O'Reilly, c2011, ISBN 978-14-493-9050-1.

[4] Irena Holubová, K. M. D. N., Jiří Kosek: *Big Data a NoSQL databáze*. Grada, 2015, ISBN 978-80-247-5466-6.

[5] Kofler, M.: *The Definitive Guide to MySQL5 Third Edition*. eBook Frenzy, 2005, ISBN 978-15-905-9535-0.

[6] Lonvick, C.: The BSD Syslog Protocol. RFC 3164, RFC Editor, August 2001.

[7] Meyer, T.; LLC, A.: Command shell overview. [Online; navštívené 12.01.2018].
URL https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb490954(v=technet.10)

[8] Murphy, M. L.: *The busy coder's guide to Android development*. CommonsWare, 2017, ISBN 978-09-816-7800-9.

[9] Smyth, N.: *Android 4.4 App Development Essentials - First Edition*. eBook Frenzy, 2014, ISBN 978-14-953-5806-7.

[10] Vugt, S. v.: *Beginning the Linux command line*. Apress, 2015, ISBN 978-14-302-6830-7.