Mendelova univerzita v Brně
Provozně ekonomická fakulta

---

# Metody distribuce zpráv mezi mobilními zařízeními

**Bakalářská práce**

Vedoucí práce:
Ing. Jan Kolomazník, Ph.D.

Daniel Křeček

Brno, 2017

V první řadě bych chtěl poděkovat rodičům, kteří mi byli oporou, a pak panu Ing. Janu Kolomazníkovi, Ph.D., který mě při psaní práce vedl, vždy odpovídal na moje všetečné otázky a poskytoval rady.

**Čestné prohlášení**

Prohlašuji, že jsem tuto práci: **Metody distribuce zpráv mezi mobilními zařízeními**
vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*
Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.
Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

místo a datum prohlášení .....

## Abstract

Křeček, D. Methods of Message Distribution Between Mobile Devices. Bachelor thesis. Brno. 2017

This thesis deals with concept of processing messages from applications using MBaaS. The text deals with the necessary steps for communication between client and backend in Firebase and Kinvey environments. Furthermore, it compares specific parameters of these two services. The application was developed for Android OS using Java programming language.



## Key words

MBaaS, Firebase, Kinvey, Android OS, Java, synchronization



## Abstrakt

Křeček, D. Metody distribuce zpráv mezi mobilními zařízeními. Bakalářská práce. Brno. 2017

Práce se zabývá principem zpracování zpráv z aplikací využívajících MBaaS. V textu jsou popsány kroky nutné pro komunikaci mezi klientem a backendem v prostředí Firebase a Kinvey. Dále srovnává specifické parametry mezi oběma službami. Aplikace byla vytvářená pro OS Android v jazyce Java.



## Klíčová slova

MBaaS, Firebase, Kinvey, OS Android, Java, synchronizace

# Obsah

# 1 Úvod práce

Přenos zpráv z mobilních zařízení na vzdálené servery prostřednictvím internetu byl, je a nadále zůstane základem při komunikaci koncového zařízení s okolním světem. K tomu je zapotřebí určitých zavedených principů v oblasti informačních technologií. I po několika desítkách let stále využíváme stejných základů a s rozvojem IT pouze přidáváme nové vrstvy nejen ochrany, ale i komfortu při jejich zpracování.

Takovým celkem, který se stará o komunikaci v prostředí mobilních operačních systémů, je BaaS (Backend as a Service), někdy také přímo MBaaS (Mobile) okruh služeb. Ten si dává za cíl ulehčit vývojáři starosti o stranu serveru, kterou hostuje poskytovael přímo u sebe. Z toho vyplývá, že celou logiku (backend) přebírá poskytovatel na sebe. Stará se o data, hostuje databázi i autentizaci uživatelů a poskytuje pohodlný management.

Je vhodné se zmínit, že většina těchto služeb staví na cloud computingu. Z toho plynou i následné benefity. Zavedení MBaaS do aplikace šetří čas na development, který se využije jinde. Není potřeba zkoumat, jak věci fungují na pozadí, o to všechno se stará poskytovatel. Mezi přednosti patří správa uživatelů, push notifikace a integrace sociálních sítí, navíc poskytují také detailní analýzy provozu využívání služeb.

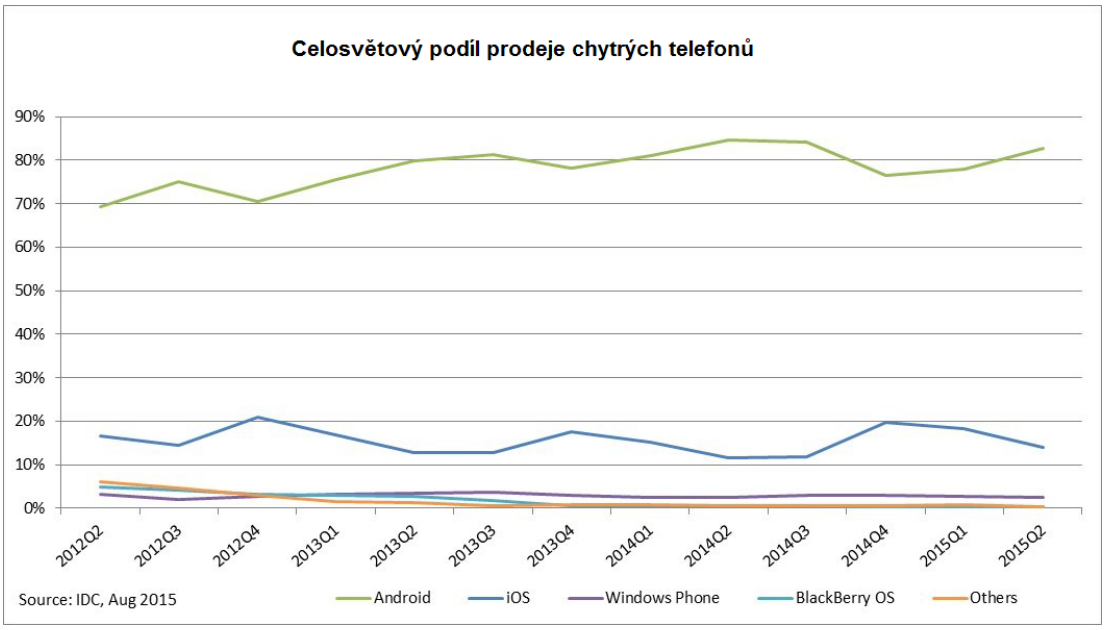
## 1.1 Cíl práce

Práce zkoumá současný trh na poli MBaaS. MBaaS hraje čím dál větší roli jakožto zprostředkovatel signálů mezi mobilními zařízeními. Ale jakého poskytovatele si vybrat? Až do nedávna nebyl na trhu nikdo dominantní.

Cílem práce je analyzování zvolených parametrů k MBaaS na konkrétních vybraných jedincích, respektive jak se při dané službě vysílají informace na server a zpět. Práce se zaměřuje na MBaaS využívající platformu Android (programy jsou psány v jazyce Java), případně multiplatformní.

# 2 Průzkum trhu

V době dnešních moderních technologií, kdy je téměř nutné udržovat krok s dobou, si málokdo představí člověka, který by nebyl nějakým způsobem propojen s okolím. To s sebou přineslo éru chytrých telefonů, bez nichž by se nemalé procento dnešní populace jednoduše neobešlo. Aby se tato zařízení udržovala aktuální a poskytovala nejposlednější informace, je třeba, aby mezi nimi a okolním světem (tedy internetem) probíhalo nepřetržité spojení. Než bude provedena podrobnější analýza principů MBaaS, bude proveden krátký přehled v současnosti používaných mobilních operačních systémů. Podle výzkumu IDC (2015) na obr. 1 je každoroční prodej zařízení běžících na jednotlivých operačních systémech následující:




Obr. 1: Každoroční podíl jednotlivých mobilních OS na trhu (IDC, 2015)

Trhu zcela zjevně dominuje OS Android. Druhou příčku si již několik let udržuje a udržovat bude iOS. Windows Phone nijak razantně svůj podíl nezlepšil a ostatní OS jsou spíše hodně okrajové a málokdo ví o jejich existenci.

## 2.1 Vybrané mobilní platformy

Jak vyplynulo z výzkumu IDC, a co zůstává v povědomí lidí díky masovým médiím, nemá cenu zkoumat operační systémy, které se celosvětově téměř neužívají, a zaměříme se tedy proto jen na ty nejvýznamnější. Jmenovitě Android, iOS, Windows Phone a BlackBerry.

### 2.1.1 Android

Hlavními programovacími jazyky pro Android jsou Java a C. Stejně jako tomu je i u dalších dále zmíněných operačních systémů, samotné zprávy se přenášejí ve formátech XML, JSON, nebo jim podobných strukturách. Nejčastějšími jsou právě zmíněné XML a JSON. V Androidu je možné pro přenos zpráv ve formátu XML užití rozhraní XmlPullParser, s jehož pomocí lze docílit odeslání zprávy na stranu serveru a zpět.

Chce-li si ale developer usnadnit práci a má-li zájem pro přenos raději využít formátu JSON, tato možnost se mu naskýtá. Řeč je o Google Cloud Messaging. Její implementace není tak pracná jako v případě ručního psaní rozhraní XML Parseru. Tato služba byla ovšem v roce 2016 prohlášena za zastaralou a na její místo nasadili novou, výkonnější a s více funkcemi – Firebase (Google, 2016).

### 2.1.2 iOS

V prostředí iOS je možné psát dvěma programovacími jazyky, Objective-C nebo Swift. Posílat data sám o sobě neumí, je třeba naincludovat některou z příslušných knihoven, které toto umožňují. Za zmínku jistě stojí RestKit stavící na technologii REST, tedy na jednoduchých HTTP voláních (Giddins, 2015). Staví na jazyce Objective-C a převádí všechny možné druhy SGML (XML, HTML, KML) i JSON.

Mezi další nástroje patří například Communication and Media Libraries for iOS, knihovna vyvinutá Midnight Coders. Stejně jako v případě RestKitu i ona staví na Objective-C. Data ze zařízení zasílá na RTMP server. Komunikace probíhá v reálném čase (Midnight Coders, 2012).

### 2.1.3 Windows Phone

Podobně jako tomu je u aplikací na stolních verzí Windowsu (Windows 8, Windows 10), i jejich mobilní verze se píše v jazyce Visual C#. Pro práci s daty je možné využít například framework Json.NET. K práci s JSON ani není nutné stahovat externí knihovny, obsahuje třídu JObject, kterou lze kód snadno vytvořit (Helmy, 2013).

### 2.1.4 BlackBerry OS

BlackBarry staví na jazyku C++. Podle developerské dokumentace pracují data v prostředí BlackBerry s XML, JSON a SQL (BlackBerry Limited, 2015). SQL je zde myšleno jako vytváření, vkládání, aktualizování a mazání dat z klasické SQL databáze.

Principy parsování dat shrnul Gontar (2010) na Stack Overflow, kde popsal, jakými prostředky a v jakém formátu tohoto lze dosáhnout. Jako standard pro JSON je zde uveden JSON ME. Pro XML pak navrhuje J2ME SAX.

## 2.2 MBaaS

Mobile Backend as a Service je typ služeb, který vývojářům webových i mobilních aplikací zprostředkovává spojení mezi klientským zařízením a backendovým serverem. Pro tento účel jsou vydávána API a SDK pro jednotlivé patřičné platformy. Princip BaaS staví na cloud computingu. Většina takto založených služeb začínala jako startupy (Dan Rowinski, 2012). Níže jsou uvedeny příklady některých aktivních MBaaS providerů, výčet featur byl ponechán v původním znění.

### 2.2.1 Firebase

Původně jako startup v roce 2011 založená společnost, která o rok později zahájila svoji BaaS činnost a poté v roce 2014 odkoupena společností Google za 7 miliónů amerických dolarů (Crunchbase, 2016), v letošním roce nahradila dosavadní googlovskou službu GCM.

SDK pro Android i iOS a REST API jsou volně distribuované, mimo zmíněná SDK je v beta verzi i nástavba pro jazyk C++, případně je možné si službu zavést na vlastní server. Firebase v současné době nabízí 3 tarify, viz obr. 2, přičemž základní Spark je bez poplatků.

| | | SPARK Free | FLAME $25 per month | BLAZE Pay as you go |
|---|---|---|---|---|
| Realtime Database | Simultaneous connections | 100 | Unlimited | Unlimited |
| | GB stored | 1 GB | 2.5 GB | $5/GB |
| | GB downloaded | 10 GB | 20 GB | $1/GB |
| | Automated backups | ✗ | ✗ | ✓ |
| Storage | GB stored | 5 GB | 50 GB | $0.026/GB[2] |
| | GB downloaded | 30 GB | 50 GB | $0.12/GB[2] |
| | Uploads & downloads | 50,000 of each | 100,000 of each | $0.01/K[2] |
| Hosting | GB stored | 1 GB | 10 GB | $0.026/GB |
| | GB downloaded | 10 GB | 50 GB | $0.15/GB |
| | Custom domain hosting & SSL | ✓ | ✓ | ✓ |
| Test Lab | Device hours | ✗ | ✗ | $5/physical, $1/virtual |
| Google Cloud Platform | Use BigQuery & other IaaS[3] | ✗ | ✗ | ✓ |

Obr. 2: Přehled platebních plánů Firebase (Firebase, 2016)

Mezi přednosti Firebase patří následující featury (Firebase, 2016):

- *Analytics* – možnost podrobného zkoumání využívání aplikace a správy uživatelů,
- *Cloud Messaging* (FCM) – prvek nahrazující bývalé GCM, možnost zasílání push notifikací na různé platformy,
- *Realtime Database* – NoSQL databáze hostovaná v cloudu, data uložená ve formátu JSON,
- *Authentication* – proces přihlašování ke službě, kromě tradičního e-mail/password způsobu podporuje i přihlašování přes sociální sítě,
- *Storage* – nahrávání a stahování souborů, hostováno na Google Cloud Storage,
- *Hosting* – možnost hostování statických stránek (CSS, HTML, JavaScript,…),
- *Test Lab* – prvek umožňující simulaci provozu služby na vzdálených zařízeních data cetnru Googlu, dostupné pouze s placenou variantou,
- *Crash Reporting* – debugovací nástroj pro detailní reporting při pádu aplikace, napojení na Analytics,
- *Notifications* – správa notifikačních kampaní, možnost zasílat notifikace jen vybrané skupině uživatelů, napojené na FCM,
- *Remote Config* – aktualizace aplikace bez potřeby vydání nové verze,
- *Dynamic Links* – Firebase nástavba deep linkingu, možnost sledování využívání přes Analytics,
- *Invites* – Out-of-the-box řešení pro sdílení aplikace, napojeno na Dynamic Links,
- *AdWords* – optimalizace a segmentace reklamních kampaní,
- *AdMob* – zasílání reklam od Google AdMob, pokročilá možnost konfigurace.

### 2.2.2 CloudMine

Dokumentace ke CloudMine je nedostatečná a těžko dohledatelná. Kromě hrubého výčtu vlastností není internetová stránka firmy příliš uživatelsky přívětivá. Podpora Javy, JavaScriptu, a do budoucna plánují zavést C#. SDK pro iOS, Android, Xamarin.

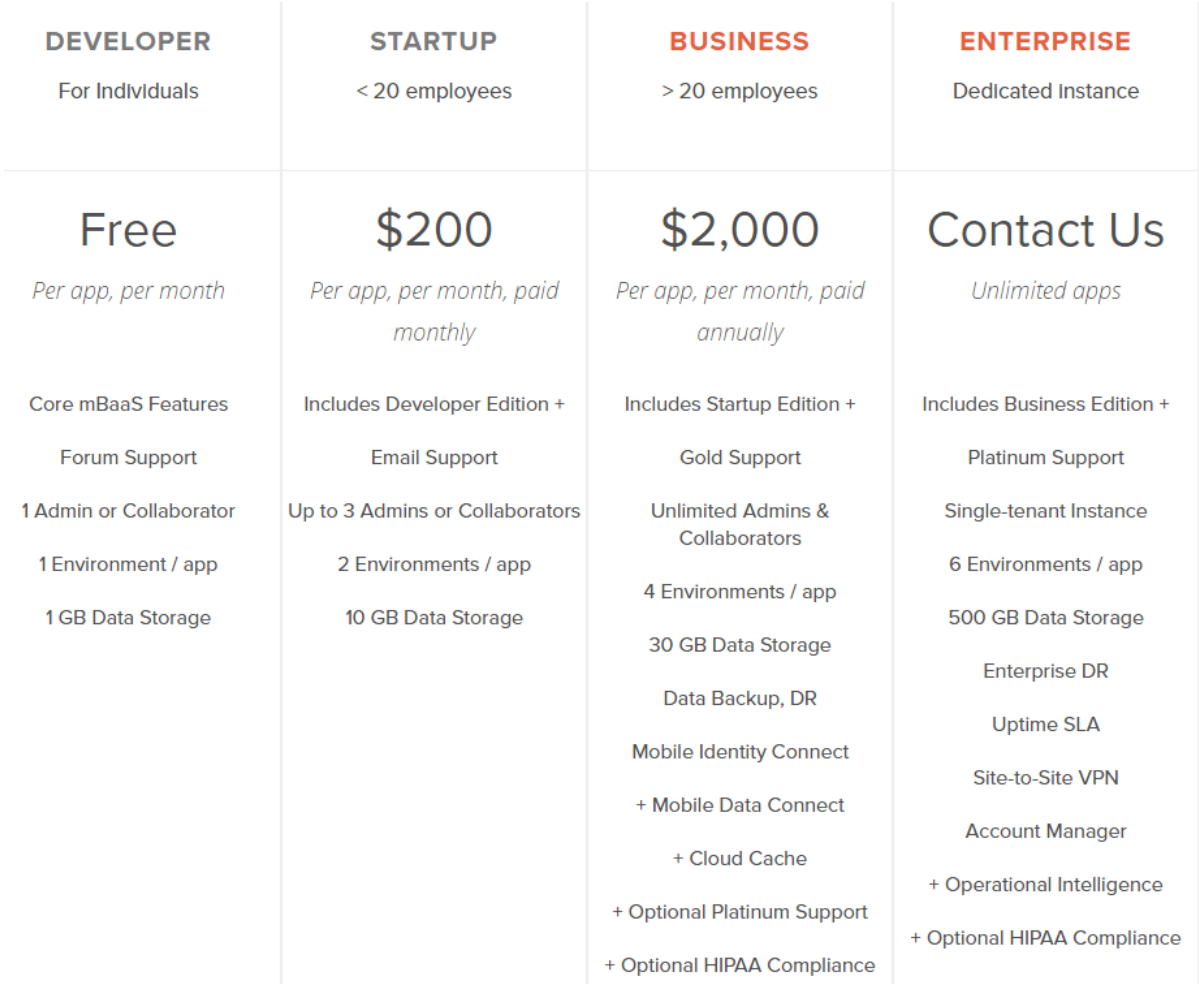
Konkrétní platební plány nejsou veřejně přístupné. Pro demo i trial verzi je nutné si dojednat schůzi.

Vlastnosti (CloudMine, 2016): Store data, Monitoring, Encryption, Research Kit (napojení na Apple's ResearchKit), Push Notifications, Geolocation, User Management.

### 2.2.3 Kinvey

Kinvey byl první Enterprise Backend as a Service na světě. Nabízí širokou škálu nejen prvků, ale i přídavků. Zde uvedeme výčet jen několika z nich (Kinvey, 2016): Database, Data Store, User Management, Push Notification, Authentication, Location Services, Cloud Caching, Business Logic (Google App Engine nebo externí), RAPID (napojení na SharePoint, Microsoft SQL Server, Salesforce, SAP, REST APIs), Microservice Runtime (NodeJS microservices).

Je k dispozici ve 4 variantách, viz obr. 3, bezplatná verze obsahuje pouze základní featury.




Obr. 3: Přehled platebních plánů Kinvey (Kinvey, 2016)

### 2.2.4 FatFractal

FatFractal podporuje kompatibilitu s Ruby on the Rail (JRuby modul) a Java/Servlets (kompatibilita s J2EE aplikacemi), a dále NoServer framewok – podpora pro iOS, Android, HTML5/JS.

Služba nabízí (FatFractal, 2014): Website hosting, Push Notifications, Send e-mails, Analytics, Monitoring.

Je k dispozici ve čtyřech variantách: varianta zdarma a 2 možné nástavby jsou na obr. 4. Poslední, čtvrtá je Enterprise, pro bližší informace je nutné si domluvit osobní schůzku.

| | SANDBOX<br>"NOSERVER"<br>*For non prod apps quick dev environment-up to 3 apps*<br>$0 | PRODUCTION<br>"NOSERVER"<br>*For prod apps, priced $30 per app, $10 per additional*<br>$30+ | RUBY/JAVA<br>*For prod apps, dedicated engine (PaaS)*<br>$35+ |
|---|---|---|---|
| APIs/Apps | 3 | 1 + ($10 per API up to 10) | 1 |
| API Calls | 1m per month | 5m + ($10 per million over 5m) | n/a |
| FFVS Hours | n/a | n/a | 720 + ($0.05 per hour over 720) |
| Data Storage | 100MB | 2.5GB + ($1 per GB over 2.5GB) | 10GB/ $0.20 per GB over 10GB |
| Blob Storage | 1GB | 10GB + ($0.2 per GB over 10GB) | 10GB + ($0.2 per GB over 10GB) |
| Bandwidth | 1GB | 10GB + ($0.2 per GB over 10GB) | 10GB + ($0.2 per GB over 10GB) |
| Relational DB | n/a | $10 | $10 |
| Private domain/SSL | n/a | $20 | $20 |

Obr. 4: Přehled platebních plánů FatFractal (FatFractal, 2014)

### 2.2.5 Kii Cloud

Cloudová MBaaS a IoT platforma od Kii Corporation. Verze zdarma neexistuje, je možné před koupí vyzkoušet trial verzi.

Featury (Kii, 2016): User Management, Data Management, Geolocation, Push Notifications, Device Management, Analytics, A/B Testing, Public i Private Cloud. Podpora SDK pro iOS, Android, Unity3D a JavaScript. Nabízejí také framework Thing-IF (Thing Interaction Framework), který poskytuje rozmanitou sadu IoT řešení.

### 2.2.6 Backendless

Jedna z možných náhrad za Parse. Podpora migrace dat z jiných systémů. Vysoce customizovatelná platforma. Integrace s OpenStack, Hadoop, Cassandra, AWS Stack, MQ messaging, J2EE application servers, Safeforce, SAP, ActiveDirectory-/LDAP. Podpora SDK pro Android, iOS, REST API, .NET, PHP, JavaScript, Actionscript.

Některé vybrané vlastnosti (Backendless, 2016): User Management, Data Persistence, Geolocation, Media Streaming, Publish/Subscribe Messaging, Push Notifications, Custom Business Logic, Analytics, Data Security.

Základní plán Backendless je zdarma, viz obr 5, při vyčerpání limitu je možné danou položku navýšit.

| Free Limit | Extension (Function Pack) | Extension Price |
|---|---|---|
| 50 API calls/second | additional 10 API calls/second | $50/month |
| 5 custom roles | unlimited custom roles | $20/month |
| 20 gb file storage | additional 20 gb | $1/month |
| 1,000,000 pub/sub messages | additional 100,000 messages | $10/month |
| 1,000,000 push notifications | additional 100,000 push notifications | $10/month |
| 1 developer on the team | unlimited team size | $10/month |
| 200 data tables | additional 100 tables | $5/month |
| 200,000 data objects per table | additional 1,000,000 data objects | $20/month |
| 100,000 geopoints | additional 100,000 geopoints | $1/month |
| Location monitoring for 1 geofence | additional 5 geofences | $5/month |
| 2 media streams | additional 10 media streams | $5/month |
| 5 custom business logic scripts | unlimited custom business logic scripts | $20/month |
| Script/code execution time (5 seconds) | Expanded script/code execution time (20 sec) | $5/month |
| Timer run frequency (60 seconds) | Enhanced timer run frequency (1 second) | $20/month |
| 1 external host | additional 10 external hosts | $5/month |
| 50 objects in cache | additional 100 objects in cache | $5/month |
| Management only via Console | Management REST API | $5/month |
| 2 mb script deployment limit | Expanded script deployment size to 10mb | $10/month |

Obr. 5: Přehled platebních plánů Backendless (Backendless, 2016)

### 2.2.7 Parse

Parse je dosluhující backendová služba od společnosti Facebook, která ukončí svoji činnost v září 2017. Toto prohlášení bylo vydáno na jejich blogu (Kevin Lacker, 2016). Přestože se ukončuje podpora od samotného vydavatele, Parse bude žít nadále – byla vydána open source verze s názvem Parse Server, která má zanikající službu nahradit a kterou si vývojář musí, pakliže s ní byl spokojen a hodlá ji nadále využívat, založit na vlastním serveru.

Nabízí se další alternativa, společnost Microsoft v březnu vydala prohlášení, že Parse začne hostovat na svém Azure Managed Services (Microsoft, 2016).

Parse podporuje multiplatformní development (iOS, Android, macOS), samozřejmostí je REST API. Zajímavostí je integrace SDK do Unity3D. Dále můžeme vyjmenovat .NET s Xamarinem, PHP, JavaScript, Arduino, Embedded C a Cloud Code (Parse, 2016).

### 2.2.8 AnyPresence

Enterprise MBaaS s nespočtem možností integrací, jmenovitě NoSQL Cloud Storage, Oracle Database, Microsoft SQL Server, REST Web Service, SOAP Web Service, MongoDB, Salesforce.com, SAP HANA, Amazon SimpleDB, oData.

Featury (AnyPresence, 2016): Cloud Storage, Unlimited Test Environments, API Versioning, Server Log Monitoring, SMS Messaging, E-mail Notification, Push Notifications (Google, Apple, Twitter), Scheduler (notifikace), XML Data Importer, Zigbee Device Gateway (napojení na služby Zigbee), Analytics, User Authentication,… SDK pro iOS, Android, Xamarin, Windows Phone, JavaScript Angular, JavaScript backbone.js, Java.

Možnost vyzkoušení jako 30denního trialu. Cena jako taková není pevně stanovená, účtují si každou jednotlivou položku, o kterou projeví zákazník zájem, a podle počtu aplikací.

### 2.2.9 Built.io

Enterprise MBaaS, tvorba webových, mobilních a IoT aplikací. Podpora AWS, Microsoft Azure, IBM SoftLayer, VMware vCloud Air, Rackspace a dalších. REST API, SDK pro iOS (jazyky Objective-C i Swift), Android a JavaScript je samozřejmostí, integrovaná podpora pro AppGyver a Xamarin.

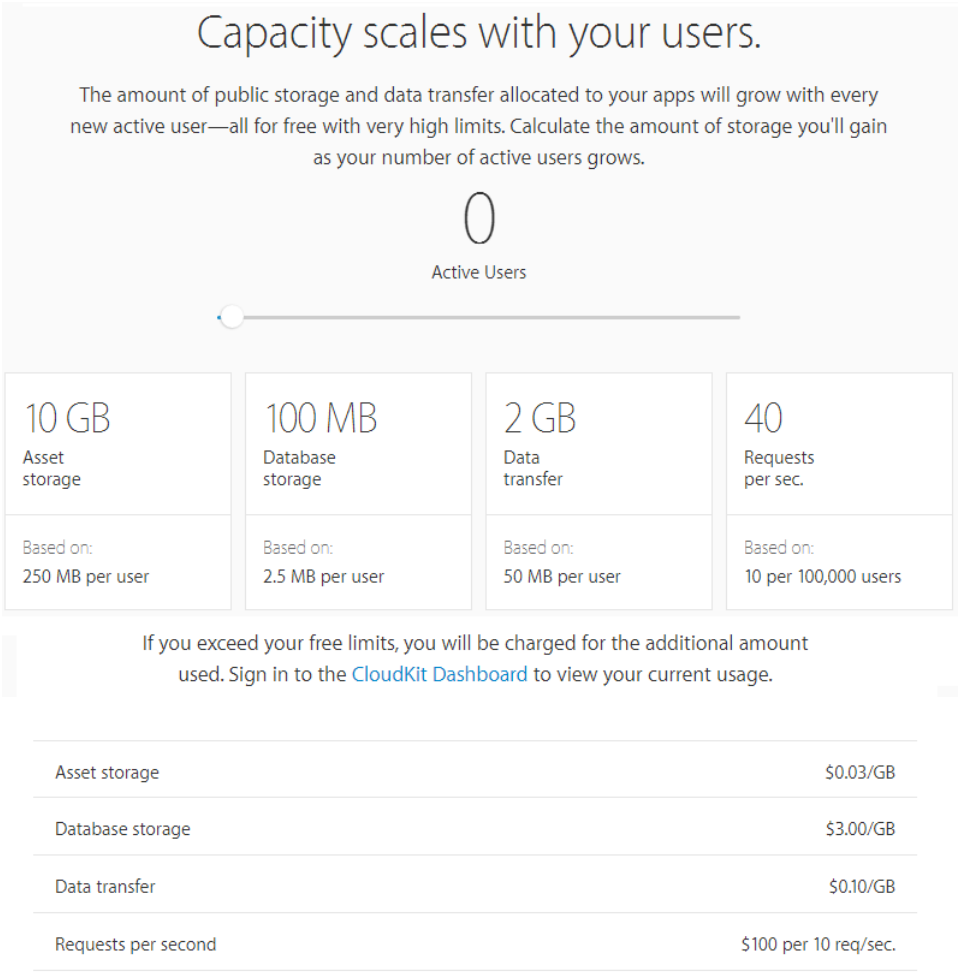
Featury (Built.io, 2016): Realtime, Security, Scale (až 10 miliónů uživatelů), Analytics, Social (integrace soc. služeb), Content Management, Geolocation, Notifications.

Bohužel nebylo možné zjistit jediný údaj o ceně služby backendu. Naskýtá se pouze možnost požádání o zprostředkování dema.

### 2.2.10 CloudKit

Na závěr okrajově zmíníme i CloudKit jakožto BaaS službu od společnosti Apple pro iOS, macOS a webové aplikace. API je také možno napojit do Xamarinu. Služba je stavěná na propojení s iCloud (iCloud Drive i iCloud Photo Library) a využívání tohoto hostingu pro soubory. Podporuje následující featury: Analytics, Authentication, Private and Public Database a Asset Storage Services.

Oproti jiným poskytovatelům BaaS je Apple velice štědrý, co se týká limitů využívání free varianty jeho služeb, viz obr. 6. Kapacitní možnosti se navyšují automaticky s růstem uživatelské základny. Horní hranice varianty zdarma (je důležité říct, že následující hodnoty platí pouze pro 10 miliónů aktivních uživatelů) je 1 petabyte pro ukládání souborů, 10 terabytů pro databázi, 200 terabytů pro přenos dat (denně) a 400 požadavků za sekundu. Následné překročení se započítává za jednotlivý gigabyte, případně 10 dalších požadavcích za sekundu (Apple, 2016).




Obr. 6: CloudKit, počáteční limity a sazebník (Apple, 2016)

## 2.3 Shrnutí

Komunikace se nijak zvlášť neliší od té, kterou až do éry chytrých telefonů provozovaly klient-server stolní aplikace. Začlenění některého MBaaS do vyvíjené aplikace je velice výhodné, vývojář má více času na development klientské strany aplikace, nemluvě o menších nákladech na provoz a údržbu serveru.

# 3 Metodika práce

Cílem práce je vybrat některé zástupce MBaaS, ty analyzovat a porovnat. Z 10 nalezených služeb budou vybrány 3: *Firebase*, *Kinvey* a *Parse Server*. *Kinvey* byl, podle slov autorů, první Enterprise MBaaS na trhu. Naproti tomu *Firebase* se po odkoupení společností Google v podobě, v jaké je zde popsán, nasadil teprve někdy ve druhé čtvrtině 2016. *Parse Server* nahrazuje dosavadní končící službu *Parse*.

Pro porovnání vybraných služeb byla v rámci praktické části práce vytvořena aplikace pro chatování uživatelů v reálném čase. Optimalizace aplikace probíhala na dvou mobilních zařízeních. Na této aplikaci postupně zhodnotíme vybrané parametry.
Parametry hodnocené na stupnici 1–3:

- délky programových kódů – srovnání kódů pro dosažení stejného cíle,
- množství informací včetně podpory – srovnání poskytovaných informací od vývojářů,
- zkušenost při práci s danými službami – osobitý pohled získaný při zpracování práce,
- chování při ztrátě spojení – jak si služba poradí a jaké jsou možnosti nastavení,
- zhodnocení potenciálních nedostatků – nalezené problémy během tvorby aplikace,
- náročnost služby na síťový provoz – test aplikace při déletrvajícím provozu,
- spotřeba baterie telefonu – test aplikace při déletrvajícím provozu.

Parametry hodnocené jako vhodné a nevhodné:

- přístup k synchronizaci dat – vhodnost služby při dané aplikaci,
- formát přenášených dat – vhodnost formátu dat při dané aplikaci.

Vytváření uživatelů a poskytování databáze pro ukládání a synchronizaci dat probíhá na serverech vybraných služeb.

# 4 Vlastní práce

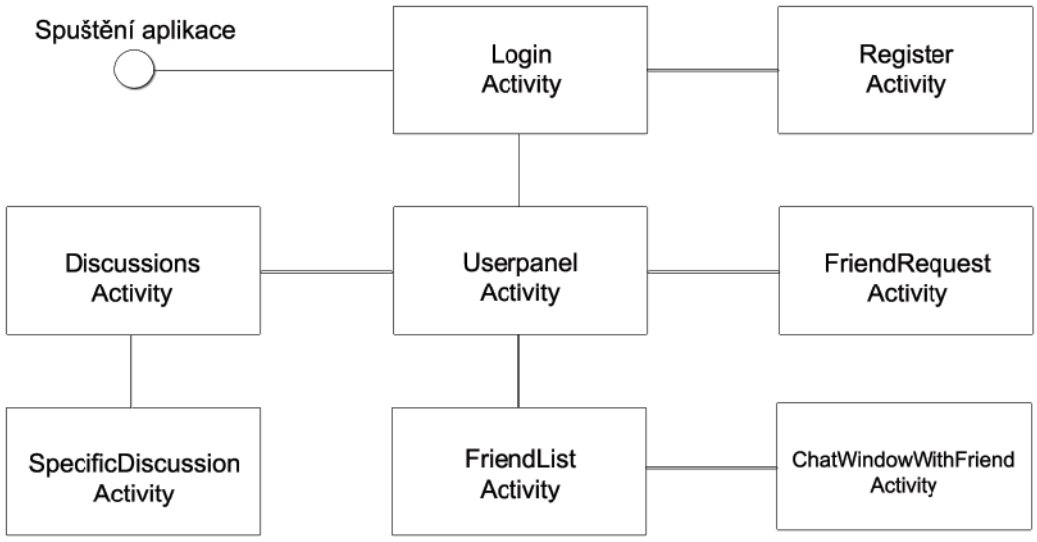
Hlavním kritériem výběru služeb byla jejich dostupnost zdarma bez nutnosti aktivace některého z platebních plánů. *Firebase* a *Kinvey* splňovaly toto kritérium, mimo jiné jejich stránky vzbuzovaly větší důvěru v nabízený produkt. *Parse Server* byl vybrán proto, že jako jediná MBaaS je open source.

Praktická část práce byla programována v jazyce Java a vývojovém prostředí Android Studio 2.2.3. Na vytvořenou aplikaci byly následně aplikovány obě vybrané služby. Výsledkem tedy jsou 2 funkcionálně shodné aplikace, běžících každá na jiné MBaaS. Testování bylo provedeno na dvou zařízeních: LG Nexus 4 a LG-D39n, obou s API 21.

## 4.1 Aplikace

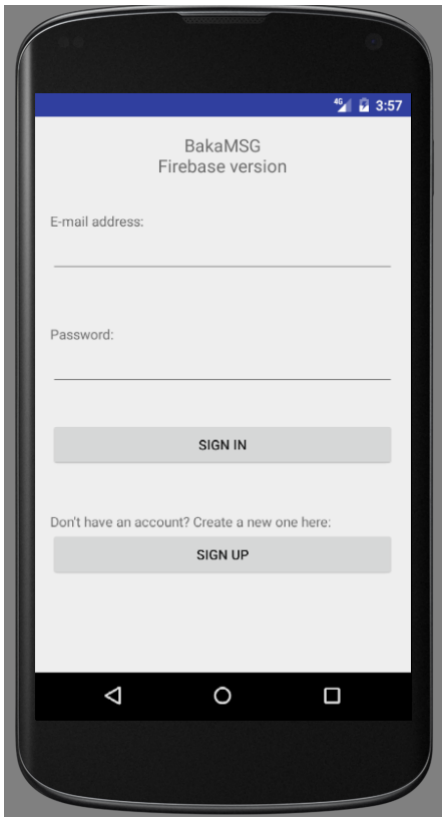
Vytvořená aplikace nesoucí název BakaMSG (MSG – zkratka pro messenger) slouží jako chatovací aplikace v reálném čase (dále jen aplikace). Mezi její funkce patří tvorba uživatelů, přidávání uživatelů do seznamu přátel, soukromý chat mezi dvěma uživateli a veřejné diskuse. Jako jazyk rozhraní aplikace byla vybrána angličtina. Debugovací logy pro testování jsou ponechány v češtině. Minimální potřebná verze SDK je 4.1 (API 16).

Aplikace se skládá z 8 aktivit. Pro potřeby ukázky jednotlivých aktivit byla vybrána verze aplikace pro *Firebase*. Vzhledem k tomu, že různé MBaaS používají jiné principy přenosu zpráv, bylo nutné přizpůsobit programovou část aplikace daným požadavkům. Na obr. 7 vidíme jednotlivé aktivity aplikace a jejich návaznosti.



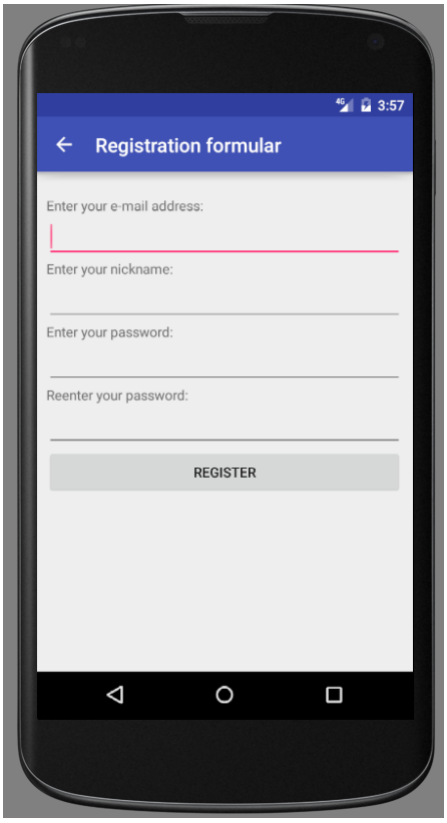

Obr. 7: Schéma aplikace

Nyní si popíšeme jednotlivé aktivity a jejich rozhraní. Poté, co uživatel spustí aplikaci, se dostává na aktivitu s přihlašovacími údaji, viz obr. 8. Tato aktivita, jak je patrné z jejího názvu, zastává roli vstupní brány. Odtud je možné přejít buď rovnou na aktivitu *Userpanel*, nebo na aktivitu *Register* a vytvořit zde nový účet. V případě obou služeb se již jednou přihlášený uživatel, který se při předchozí relaci neodhlásil a přesto aplikaci zavřel (ve smyslu uvolnění z paměti telefonu), automaticky přihlásí přes uložený *ID token* a přechází rovnou na aktivitu *Userpanel*.



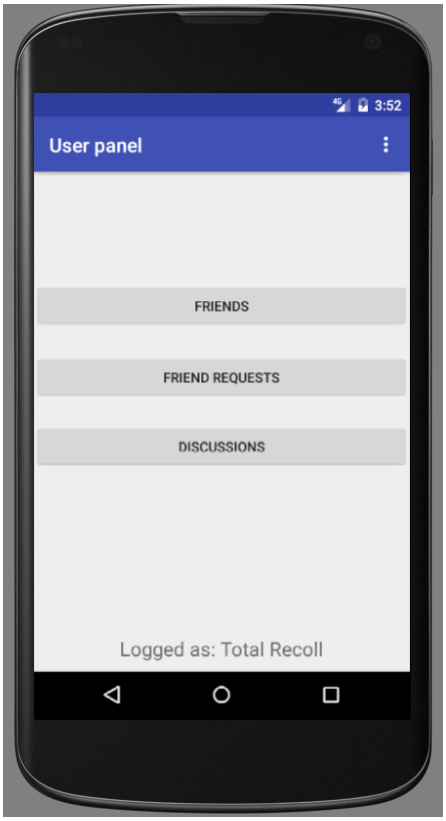

Obr. 8: Základní aktivita s přihlášením ke službě

Následuje aktivita *Register* (obr. 9). Vždy je nutné vyplnit 4 položky: e-mail, dále nick, pod kterým následně vystupujeme v celé aplikaci, a v poslední řadě heslo a jeho opakované zadání pro kontrolu. Tady se obě služby liší. *Firebase* jako autentizační údaj bere e-mail a nick (username, nebo alias) je pouze dobrovolný parametr. *Kinvey* to má přesně naopak, jako autentizační složku má nick a e-mail je dobrovolný. V obou případech se vyžaduje pro registrační účely vyplnit všechny zobrazované položky. Položka hesla má navíc striktně nastaveno, že délka hesla musí být alespoň 6 znaků a dále že délka nicku se pohybuje od 3 do 21 znaků. Jakmile uživatel odešle požadavek na službu ke zpracování a je mu vrácena kladná odpověď, je automaticky přenesen zpět na aktivitu *Loginu* a je mu tato skutečnost oznámena.



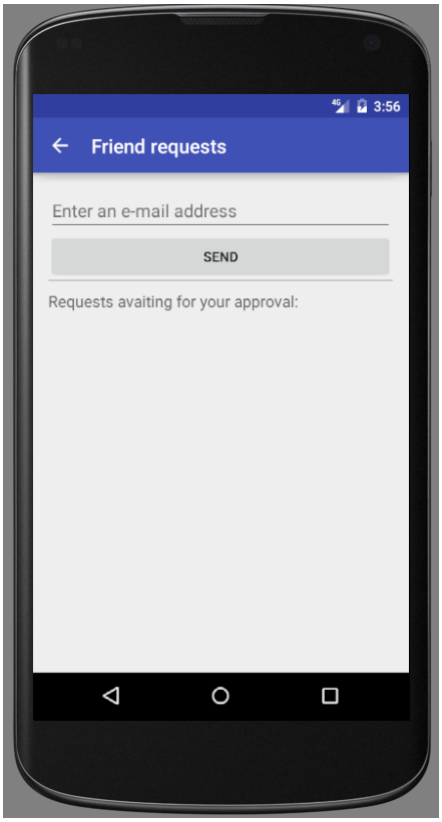

Obr. 9: Aktivita s registrací nového uživatele

Po zadání přihlašovacích údajů a potvrzení jejich pravosti ze strany serveru je uživatel přesunut z aktivity *Login* na aktivitu *Userpanel* (obr. 10). Tato aktivita je výchozím bodem pro celý zbytek aplikace. Dostaneme se jak k seznamu přátel na aktivitě *FriendList*, tak na veřejné diskuse v aktivitě *Discussions*. Tato aktivita je také důležitá proto, že si zde získáváme informace o aktuálně přihlášeném uživateli (jeho nick je následně zobrazen), které později přenášíme do ostatních aktivit. V pravém horním rohu se nachází volba pro odhlášení se ze služby. Po jejím klepnutí je odstraněn autentizační *token* a uživatel vrácen na aktivitu *Login*. Provedení této akce je nutné pro změnu uživatele, v opačném případě se pokaždé při novém spuštění aplikace uživatel ihned dostává na aktivitu *Userpanel* s údaji posledního přihlášeného klienta.



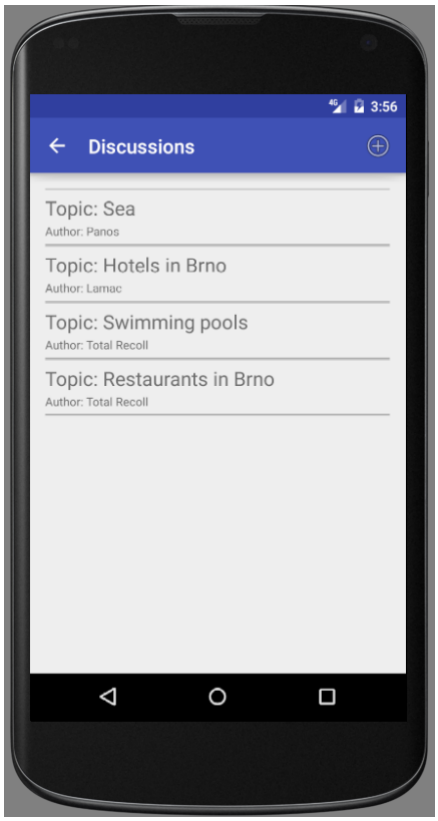

Obr. 10: Aktivita s možnostmi aplikace

Abychom mohli spojit komunikaci mezi dvěma uživateli, je nutné mezi nimi vytvořit pouto. K tomuto účelu slouží aktivita *FriendList*. Jak je ukázáno na obr. 11, ve *Firebase* po zadání příslušného e-mailu (v Kinveym po zadání unikátního nicku), aplikace ověří existenci údajů v databázi. V případě, že cílový uživatel v databázi existuje, odešle aplikace na server potřebná data pro navázání spojení. Ta se poté fetchnou do listview na téže aktivitě u cílového uživatele. Ten buď požadavek přijme, a vytvoří tak záznam v databázi o dvou kamarádech, nebo jej odmítne. Kam přesně se tyto informace odešlou a jak vypadají, bude ukázáno později.




Obr. 11: Aktivita se zpracováním požadavků o přidání do přátel

Jednou z možností komunikace skrze aplikaci jsou tzv. veřejné diskuse na aktivitě *Discussions*. Tedy témata, u kterých není potřeba dodatečných záznamů o kamarádech, ale jsou přístupná všem uživatelům. Ukázkový seznam diskusí je vidět na obr. 12. Tvorba nového vlákna probíhá přes ikonku v pravém horním rohu. Mazání celého vlákna (včetně příslušných zpráv) je pak přístupné přes přidržení klepnutí nad danou položkou v listview, samozřejmě pouze za předpokladu, že daný přihlášený uživatel je autorem dané diskuse.




Obr. 12: Aktivita s veřejnými diskusemi

Aktivita *FriendList* zprostředkovává validní záznamy o přátelích přiřazených k danému uživateli. Stejně jako tomu bylo v ostatních aktivitách, záznamy se fetchují do listview. Po klepnutí na konkrétní položku se přechází na aktivitu *ChatWindowWithFriend*.


Obr. 13: Aktivita se seznamem přátel

A na závěr aktivity *ChatWindowWithFriend* a *SpecificDiscussion*. Ve své podstatě se liší akorát v zasílaných požadavcích na server. Jinak jsou funkcionálně jedna ku jedné. Rozhraní na obr. 14 ukazuje výpis ze soukromých zpráv mezi uživateli Pascal a Total Recoll. Fetchnutou zprávu může její vlastník upravit po klepnutí na danou položkou v listview. Mazání není umožněno. Principy zasílání, mazání a aktualizace dat budou vysvětleny v kapitolách 4.2.3 a 4.3.3.


Obr. 14: Aktivita s komunikací mezi dvěma uživateli

## 4.2 Firebase

Verze *Firebase* v době tvorby aplikace byla 9.6.1. Pro napojení *Firebase* do projektu je potřeba nadeklarovat následující řádky na patřičná místa (Firebase, 2016): do gradle na úrovni rootu `classpath'com.google.gms:google-services:3.0.0'`. A do gradle na aplikační úrovni:

- `compile 'com.google.firebase:firebase-database:9.6.1'`,
- `compile 'com.google.firebase:firebase-auth:9.6.1'`,
- `apply plugin: 'com.google.gms.google-services'`.

Dále `compile 'com.firebaseui:firebase-ui:0.6.0'`. Knihovna *FirebaseUI* nám umožní pracovat s `FirebaseAdapterem`; ten je nutnou prerekvizitou pro všechny naše listview.

Od verze Android Studia 2.1.2 je možné *Firebase* napojit přímo přes rozhraní vývojového prostředí. To se nachází v Tools -> Firebase. Napojením služby do vytvářené aplikace provede vývojáře rychlý průvodce.

### 4.2.1 Základní informace

Projekt se nachází na adrese https://bakamsg-85f98.firebaseio.com/. Databáze u *Firebase* je typu NoSQL a má stromovou *JSON* strukturu. Název databáze *bakamsg-85f98* se bere jako kořen stromu. Následné hlavní větve jsou: discussionMessages, discussions, friends, friendRequests, privateMessages, users. Náhled databáze je na obr. 15. Všechny položky začínající na -K***** jsou unikátní klíče generované přes metodu `.push()`, kterou vlastní proměnná typu `DatabaseReference`.

Zavoláním `FirebaseDatabase.getInstance().getReference()` v kódu 1 dostáváme odkaz do naší databáze a přes `database.child("cesta")` nastavíme cestu ke konkrétním datům. `.child()` nám říká, že si přejeme zanořit se o úroveň níže v hierarchii stromu. Tohoto postupu využijeme při každé manipulaci s daty.

Kód 1: Získání reference do databáze

```
DatabaseReference database = FirebaseDatabase.getInstance().
    getReference();
database.child("discussionMessages");
```

V tomto konkrétním případě jsme se dostali na 2 objekty ve větvi discussionMessages. `.child()` se dá skládat za sebou tak dlouho, dokud se nedostaneme do potřebné podvětve. Druhou možností, jak lze poskládat cestu, je nadefinovat ji celou v jediném volání, paříklad `database.child("privateMessages/"+mergedID+"/"+msg.getMsgKey()+"/text")`.

Položky ve větvi friends odkazují na ID uživatelů, kteří mají nějakého přiřazeného přítele, a položky pod privateMessages jsou sloučená ID vždy dvou uživatelů, mezi kterými již proběhla nějaká komunikace, podle abecedy vzestupně.


Obr. 15: Struktura databáze aplikace ve Firebase

### 4.2.2 Registrace, autentizace, uživatelé a práva

V aplikaci bylo použito principu registrace pojmenované jako *Password Authentication* (e-mail a heslo). Tento způsob přihlášení není závislý na žádné třetí straně jako jiné způsoby autentizace, jmenovitě: Facebook, Twitter, Google nebo GitHub (Firebase, 2016).

Registrace nového uživatele provedeme kódem 2. Proměnná `mAuth` je vstupní bod pro autentizační službu *Firebase SDK*. Poté do ní přiřadíme instanci spuštěné služby a do metody `createUserWithEmailAndPassword()` odešleme zadanou e-mailovou adresu a platné heslo.

Kód 2: Vytvoření nového uživatele v prostředí Firebase

```
private FirebaseAuth mAuth;
mAuth = FirebaseAuth.getInstance();
mAuth.createUserWithEmailAndPassword(email,password)
    .addOnCompleteListener(this, new OnCompleteListener() {
        @Override
        public void onComplete(@NonNull Task task) {
            if(task.isSuccessful()){
                createNew(task.getResult().getUser());
            }
            if (!task.isSuccessful()){
                Toast.makeText(RegisterActivity.this,R.string.
                    auth_failed,Toast.LENGTH_LONG).show();
                return;
            }
        }
    });
```

Po úspěšném odeslání na server dostáváme potvrzující zprávu. Zároveň s tím se automaticky uživatel přihlašuje autentizačním *tokenem*, který jsme nazpět taktéž obrželi. Jelikož se při registraci zadává pouze e-mail a heslo, dodatečný údaj o nicku zpracujeme ihned poté zavoláním metody `createNew()`, ukázka v kódu 3, do které si předáme informace o nově založeném účtu z `task.getResult()`. Následně na předaném uživateli voláme patřičné metody a skrz `updateProfile()` odesíláme na server.

Abychom mohli pracovat s údaji různých uživatelů, je nutné si při registraci jejich informace dále zaznamenávat i přímo do databáze (*JSON* stromová struktura). *Firebase* neobsahuje žádnou metodu, která by podle určitého parametru mohla zkoumat jiné uživatele. Této informace využijeme u zasílání požadavku o přidání do přátel.

Kód 3: Aktualizace nově vytvořeného uživatele

```
private void createNew(FirebaseUser u){
    final FirebaseUser user = u;
    UserProfileChangeRequest profileUpdates = new
        UserProfileChangeRequest.Builder()
```

```
                .setDisplayName(alias.getText().toString())
                .build();
        user.updateProfile(profileUpdates)
                .addOnCompleteListener(new OnCompleteListener() {
                    @Override
                    public void onComplete(@NonNull Task task) {
                        creatUserInDatabase(user.getUid(),alias.getText
                            ().toString(),user.getEmail());
                    }
                });
    }
```

Stejné proměnné `mAuth` se užije i pro přihlášení. Po zavolání metody `signInWithEmailAndPassword()` v kódu 4 se na server zasílá požadavek na prokázání totožnosti. V případě úspěchu se vrací autentizační *token* a uživatel je odeslán na aktivitu *Userpanel*.

Kód 4: Proces přihlašování v prostředí Firebase

```
mAuth.signInWithEmailAndPassword(email,password)
    .addOnCompleteListener(this, new OnCompleteListener() {
        @Override
        public void onComplete(@NonNull Task task) {
            if(!task.isSuccessful()){
                Toast.makeText(LoginActivity.this,R.string.
                    login_failed,Toast.LENGTH_LONG).show();
                return;
            }
            if(task.isSuccessful()){
                goToUserPanel();
            }
        }
    });
```

Instanci aktuálně přihlášeného uživatele získáme zavoláním: `FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser()`. Tato proměnná posléze obsahuje jeho username, e-mailovou adresu, či ID. Pokud žádný uživatel není přihlášen, obdržíme `null`.

*Firebase* umožňuje definovat čtecí a zapisovací práva i zvlášť pro každou větev ve stromové struktuře databáze. V případě naší aplikace jsou povolené všechny operace, protože je nutné pracovat i s daty, jejichž nejsme vlastníky. Dále je možné specifikovat, zdali je nutné, aby k prohlížení konkrétních dat musel uživatel být přihlášen. Služba totiž nabízí i možnost anonymního užívání, které ale v námi vytvořené aplikaci není povoleno.

Odhlášení ze služby provedeme přes zavolání `FirebaseAuth.getInstance().signOut()`. Po zavolání tohoto kódu dojde k odstranění autentizačního *tokenu* a ke smazání údajů z cache i na disku.

### 4.2.3 Zasílání, přijímání a aktualizace dat

Jak bylo již nastíněno v kapitole 4.2.1, data mají podobu *JSON* objektů. Tvorba objektů v aplikaci se dá realizovat mnoha způsoby. Doporučuje se vytvářet datové modely. Ty se využijí i při získávání dat z databáze. Mimo zmíněné modely se objekt posílající na server může poskládat do `HashMapy`. Jako klíče se použijí identifikátory proměnných z výše řečených modelů.

V datových modelech je nutné uvádět i prázdný konstruktor, přestože by nikdy nemělo dojít k jeho volání, protože vždy budeme chtít nastavit alespoň jednu hodnotu. *Firebase* ovšem toto vyžaduje. Při vložení klíčového slova `@IgnoreExtraProperties` v části deklarace datového modelu je možné při vkládání hodnot do konstruktoru některou vynechat. V kódu 5 následuje tvorba modelu pro přidání nového přítele.

Kód 5: Tvorba datového modelu

```
import com.google.firebase.database.IgnoreExtraProperties;
@IgnoreExtraProperties
public class NewFriend {
    private String friendKey;
    private String nick;
    private String email;

    public NewFriend(){}

    public NewFriend(String friendKey, String nick, String email){
        this.friendKey = friendKey;
        this.nick = nick;
        this.email = email;
    }
    public String getFriendKey(){
        return friendKey;
    }
    ...
}
```

Tvorba zprávy v kódu 6 probíhá s využitím `HashMapy`. Proměnná `msgRef` drží adresu v databázi, kam data uložíme. Zavoláním `msgRef.push().getKey()` získáme unikátní identifikátor pro nově vkládanou zprávou. ID generuje `.push()`, `.getKey()` nám tuto hodnotu vrátí do nachystané proměnné jako textový řetězec.

Důvod, proč prvně voláme samotné `.push()` bez uložení objektu, je, že hodnotu tohoto klíče použijeme při práci s danou zprávu v případě aktualizace nebo mazání. `FirebaseAdapter` by nám jinak při získání dat nedal žádnou informaci o tom, pod jakým klíčem je objekt uložen.

To by neplatilo v případě `ChildEventListeneru` nebo `ValueEventListeneru`, které tuto informaci drží v `DataSnapshotech`, o těch dále v textu.

Po získání ID změníme cestu v databázi a jako konečnou větev zvolíme nastavený získaný klíč. Struktura databáze se vytváří za běhu aplikace, není třeba, aby

na dané pozici existoval nějaký záznam. A vytvořený objekt pošleme přes metodu `setValue()`.

Kód 6: Tvorba a odeslání zprávy přes HashMapu

```
public void sendMessage(String message, String fNick, String fUID,
    String merUIDs){
     DatabaseReference msgRef = database.getRef().child("
        privateMessages").child(merUIDs);
     Map value = new HashMap<>();
     String key = msgRef.push().getKey();
     value.put("text",message);
     value.put("myNick",user.getDisplayName());
     value.put("user1ID",user.getUid());
     value.put("friendNick",fNick);
     value.put("user2ID",fUID);
     value.put("date", ServerValue.TIMESTAMP);
     value.put("msgKey",key);
     msgRef = database.getRef().child("privateMessages").child(
        merUIDs).child(key);
     msgRef.setValue(value);
     chatMessage.setText("");
}
```

V kódu 7 následuje ukázka tvorby diskuse s využitím konstruktoru připraveného datového modelu, totožný princip jako u `HashMapy`.

Kód 7: Tvorba a odeslání zprávy přes datový model

```
DatabaseReference msgRef = database.child("discussions");
String key = msgRef.push().getKey();
Discussion disc = new Discussion(edittext.getText().toString(),user
    .getDisplayName(),key);
msgRef = database.child("discussions").child(key);
msgRef.setValue(disc);
```

Aktualizace konkrétního údaje v databázi je nejrychlejší (v našem případě aktualizace zprávy při chatu mezi dvěma uživateli). Do proměnné typu `DatabaseReference` v kódu 8 přiřadíme absolutní cestu zakončenou identifikátorem hodnoty, kterou se chystáme změnit. Pak už jen stačí zavolat metodu `setValue()` stejně jako při vložení nového záznamu a do parametru vložit požadovanou novou hodnotu.

Kód 8: Aktualizace dat

```
DatabaseReference msgRef = database.child("privateMessages/"+
    mergedID+"/"+msg.getMsgKey()+"/text");
msgRef.setValue(edittext.getText().toString());
```

Data z databáze v aplikaci fetchujeme do speciálního `FirebaseListAdapteru`, který napojíme na zcela obyčejné listview. `FirebaseListAdapter` pochází z přídavné knihovny *FirebaseUI*. Při vytváření tohoto adaptéru je nutné do něho vložit

následující parametry: *this* (odkazující na aktivitu držící listview), předpřipravenou třídu datového modelu, do kterého se nahrají data z databáze, připravený layout pro listview a cestu k datům. Získaná data lze seřadit třemi způsoby (Firebase, 2016):

- `orderByChild("specifický klíč")` – setřídění podle hodnoty zadaného identifikátoru položky potomků (například podle data),
- `orderByKey()` – setřídění podle identifikátorů potomků (-K****),
- `orderByValue()` – setřídění podle hodnot potomků.

Vždy je možné použití pouze jedné metody třídění získaných dat.

`FirebaseListAdapter` v kódu 9 obsahuje metodu `populateView()`, přes kterou se získaná data z databáze posléze nasetují na připravené `TextView`.

Kód 9: Fetchnutí dat do listview přes FirebaseAdapter

```
final FirebaseListAdapter adapter = new FirebaseListAdapter(this,
    Message.class, R.layout.list_chat_messages, database.child("
    privateMessages").child(mergedID)).orderByChild("date"){
     @Override
     protected void populateView(View v, Message model, int position
         ) {
          ...
          ((TextView)v.findViewById(R.id.textViewListMessage)).
              setText(model.getText());
          ((TextView)v.findViewById(R.id.textViewListNick)).setText("
              From: "+model.getMyNick());
          ((TextView)v.findViewById(R.id.textViewListDate)).setText("
              ["+dateFormat.format(new Date (model.getDate()))+"]");
          ...
     }
};
listOfMessages.setAdapter(adapter);
```

Alternativní způsob získání dat z databáze je skrze 2 jiné listenery: `ChildEventListener` nebo `ValueEventListener`. Ty je nutné napojit na konkrétní referenci přiřazenou v položce typu `DatabaseReference`. Prvně si popíšeme `ChildEventListener`. Ten se skládá ze 4 callbacků (Firebase, 2016):

- `onChildAdded()` – získání záznamů z databáze a naslouchání na nově přidané potomky,
- `onChildChanged()` – naslouchání na změny v položkách potomků,
- `onChildRemoved()` – naslouchání na odstranění položek potomků,
- `onChildMoved()` – naslouchání na změny pozic potomků.

Všechny 4 callbacky poskytují jako předávaný parametr hodnotu z `DataSnapshot`, která v sobě nese informace o změněném potomkovi. Jakmile listener není nadále potřeba, zavoláním `removeEventListener(this)` zrušíme jeho naslouchání.

Naproti tomu `ValueEventListener` vrací celou strukturu jako jeden jediný `DataSnapshot`. Ke konkrétním potomkům se dostává přes průchod cyklem. S `ValueEventListenerem` se tedy můžeme i zeptat, kolik se na daném místě nalézá záznamů.

Mazání hodnot z databáze v kódu 10 probíhá obdobně jako v případě aktualizace konkrétní hodnoty. Nejprve do proměnné typu `DatabaseReference` uložíme cestu a pak zavoláme metodu, kterou je možné danou položku smazat. `RemoveValue()` odstraní celou větev a všechny její potomky.

Kód 10: Mazání dat podle předaného klíče

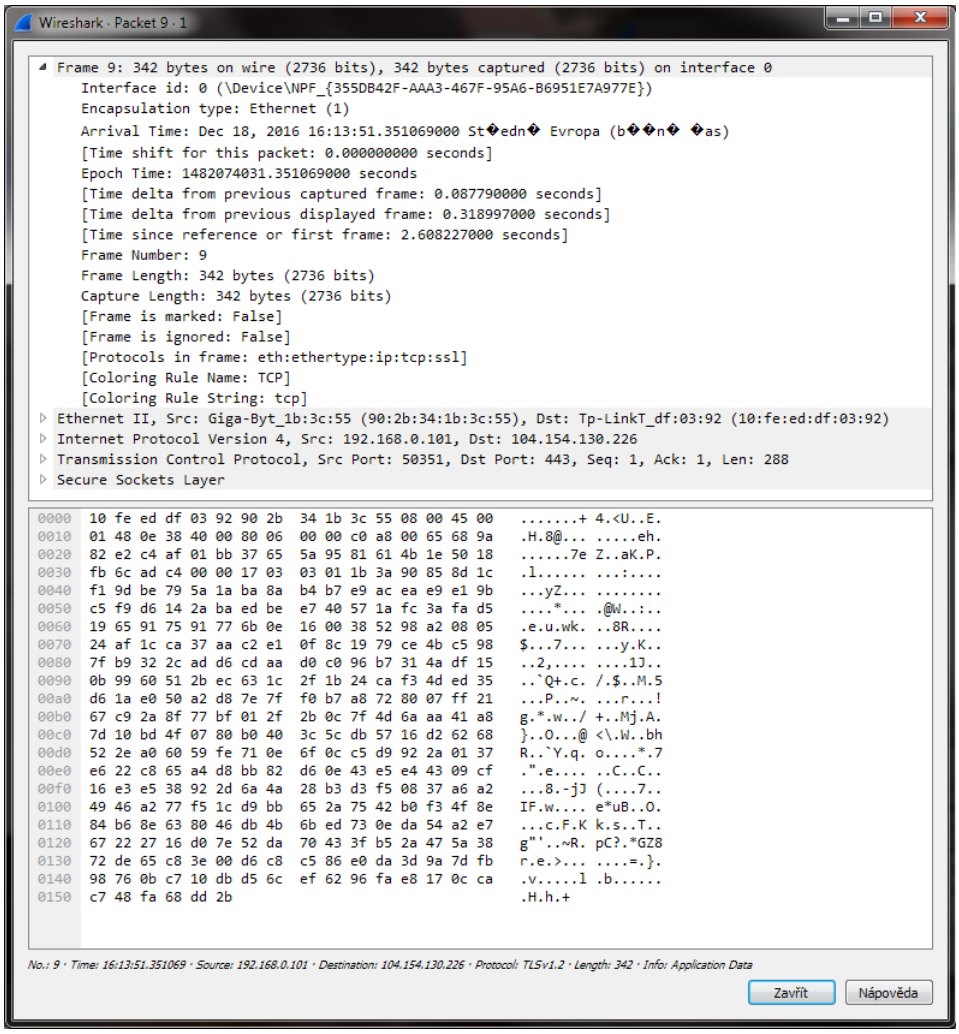
```
DatabaseReference discRef = database.child("discussions/"+
    longClickDisc.getKey());
discRef.removeValue();
discRef = database.child("discussionMessages/"+longClickDisc.getKey
    ());
discRef.removeValue();
```

Dále existují další dvě metody, kde je potřeba jako parametr předhodit `null`. Jedná se o `setValue()` a `updateChildren()`. S `updateChildren()` je možné během jediného volání smazat vícero různých potomků současně. Vytváří se jako `HashMapa` (Firebase, 2016).

V poslední řadě se krátce podíváme na samotný přenos. Na server jsme odeslali zprávu z veřejné diskuse *Sea*: „Toto je zkusebni zprava pro zjisteni prenosu.“ U veřejných diskusí se nepoužívá `.push()` pro natažení unikátního klíče, takže přenášíme jen vytvořený objekt za pomoci `HashMapy`.

```
    9 16:13:51.351069 192.168.0.101   104.154.130.226   TLSv1.2   342 Application Data
   10 16:13:51.478906 104.154.130.226 192.168.0.101     TCP        60 443→50351 [ACK] Seq=1 Ack=289 Win=22022 Len=0
```

Obr. 16: Pakety přenosu zprávy z Wiresharku (Firebase)




Obr. 17: Bližší pohled na paket z Wiresharku (Firebase)

Samotný přenos pobíhá přes protokol TLSv1.2 (obr. 16). Zpráva je zašifrovaná a odesílaný řetězec ani ostatní připojené údaje nelze přečíst (obr. 17). Odpovědí nám je potvrzení o doručení zprávy na server.

### 4.2.4 Offline režim

*Firebase* automaticky cachuje natažená data ze serveru. Ukládá je v paměti až do doby, než aplikaci uvolníme z paměti telefonu. Aby si data ukládal na stálo, je potřeba deklarovat jeden řádek kódu při získání instance služby: `FirebaseDatabase.getInstance().setPersistenceEnabled(true)`.

Pakliže během ztráty internetového spojení jako přihlášený uživatel napíšeme zprávy, po opětovném připojení k internetu *Firebase* tyto zprávy odešle sám na server. To samozřejmě není vše, čím *Firebase* disponuje. Okrajově se ještě zmíníme o možnosti synchronizování dat z určité větve databáze i bez aktivního spojení přes listener. Defaultně si *Firebase* cachuje až 10 MB dat z poslední synchronizace (Firebase, 2016).

Následující kód 11 umožňuje fetchovat data o diskuzích do lokálního úložiště, aniž bychom museli přepnout do dané aktivity.

Kód 11: Zavedení lokálního úložiště pro data z diskusí

```
DatabaseReference scoresRef = FirebaseDatabase.getInstance().
    getReference("discussions");
scoresRef.keepSynced(true);
```

### 4.2.5 Dokumentace a podpora

Dokumentaci u *Firebase* není co vytknout. Na svých stránkách mají vše krásně přehledné, jsou zde kusy ukázkových kódů a dokonce linkují videa z Youtube, ve kterých vysvětlují principy všech svých součástí. K videím na Youtube se váže i další podpora, a to videa věnující se ukázkám kódů. U těch lze vytknout jen fakt, že většina jich v době psaní této práce odkazovala na starší verzi syntaxe, než jaká se používá v současnosti. Mnohdy není ani nutné zacházet na stránku s popisky API, kde mají taktéž detailně rozepsané vysvětlivky. Nechybí ani demo aplikace na GitHubu.

### 4.2.6 Potenciální nedostatky

Všechny akce s daty probíhají asynchronně a listener neustále naslouchá na změny. Není tedy žádná možnost, jak zničehonic během jednoho volání učinit další. Jsme tedy nuceni si do jednotlivých větví například opakovaně ukládat data o uživateli (nick, ID, případně e-mail). Z tohoto důvodu také zanášíme tyto údaje speciálně i do stromové struktury; navíc již v momentě, kdy daného uživatele vytváříme při registraci. Navíc nám tím vznikají nadbytečné záznamy v samotné databázi.

K chování listeneru se váže ještě jeden drobný nedostatek, na který se narazilo. Bylo zamýšleno k procesu prvotního natažení dat do listview užít `ProgressDialogu`, které by signalizovalo, že se provádí daný úkon. Hledání způsobu dosažení kýženého výsledku po několika nezdarech skončilo úspěchem, ale za cenu dalšího volání při navazování prvotní komunikace se serverem při přechodu na novou aktivitu.

Podle dokumentace k Androidu (Google, 2016) je k dispozici metoda adaptéru `registerDataSetObserver()`, která obsahuje `onChanged()` a `onIvalidated()`. S `onChanged()` se dá zeptat na moment, kdy byla získána první data, jenže se žádná nemusejí nalézt.

Celý tento problém se dal obejít přes jednorázové užití `ValueEventListeneru` a jeho vnitřní metodu `onDataChange()`. Z námi určeného místa v databázi předaného z proměnné typu `DatabaseReference` získáme počet potomků. Je-li roven nule, žádná data neexistují. Pak už jen zbývá v samotné metodě zavolat `removeEventListener(this)` a již nepotřebný listener odstranit.

## 4.3 Kinvey

Verze *Kinvey* v době tvorby aplikace byla 2.10.9. Knihovny služby je nutné stáhnout v .zip archívu a vyextrahovat do adresáře libs. Tento archív obsahuje 8 knihoven v .jar formátu a 1 v .aar. Dále je nutné do gradle na aplikační úrovni v sekci *dependencies* dopsat `compile(name:'kinvey-android-2.10.9', ext:'aar')`.

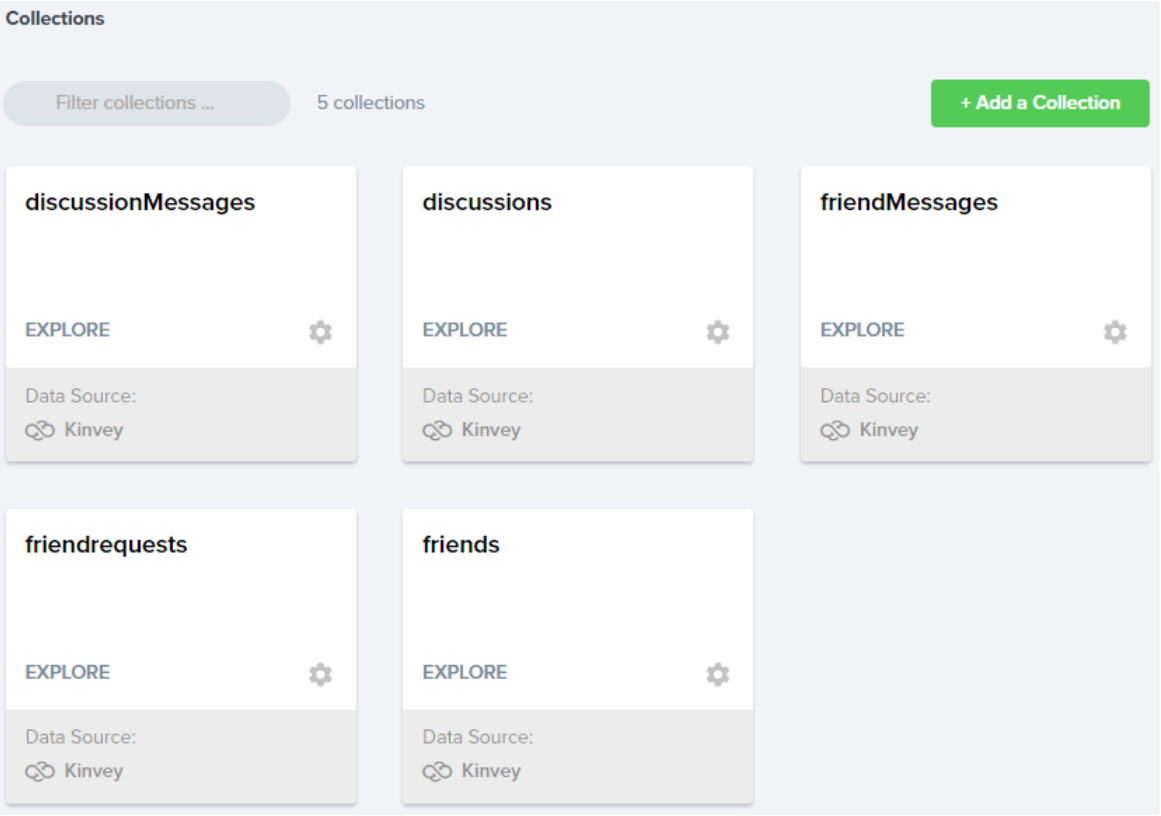
Následně si vytvořit ve složce assets soubor *kinvey.properties* a dopsat do něj 2 řádky: app.key=aplikační_klíč, app.secret=tajný_klíč. Klíče jsou vygenerované v konzoli projektu v backendu.

### 4.3.1 Základní informace

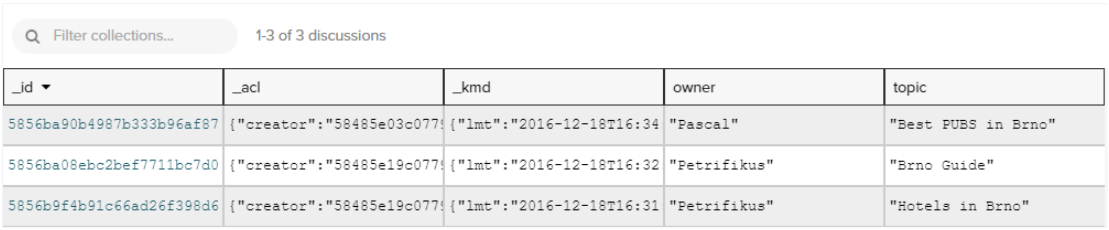
Projekt se nachází na adrese https://us1.kinvey.com/apps/BakaMSG-78bfd/. Stejně jako v případě *Firebase*, i Kinvey má databázi NoSQL, konkrétně se jedná o *MongoDB*. Místo stromové struktury s větvemi jsou zde tzv. kolekce. Kolekce je ve své podstatě zcela obyčejná tabulka se sloupci jako identifikátory a řádky jako entitami. Jednotlivé kolekce jsou zobrazeny na obr. 18. Každá buňka tabulky se chová jako *JSON* objekt, takže můžeme bez problému vkládat i složitější struktury, viz obr. 19.

Každá kolekce si automaticky vytváří 3 sloupce: _id, _acl a _kmd. _id je unikátní identifikátor entity, v _acl je ID uživatele, který danou entitu vytvořil, to se vkládá automaticky. _kmd drží 2 hodnoty – čas vytvoření entity a čas poslední provedené změny v entitě.

Odkaz do databáze získáme zavoláním následujícího příkazu, kdy si přiřadíme instanci služby do proměnné `mKinveyClient`: `Client mKinveyClient = new Client.Builder(this.getApplicationContext()).build()`. Nad tou pak provádíme všechny operace.




Obr. 18: Struktura databáze aplikace v Kinvey




Obr. 19: Struktura kolekce diskusí

### 4.3.2 Registrace, autentizace, uživatelé a práva

Uživateli se v prostředí Kinvey říká *active user*. Existují zde 2 druhy uživatelů, explicitní a implicitní. Explicitní se vytváří ručně, máme na výběr buď kombinaci uživatelské jméno + heslo, nebo použití některé ze sociálních sítí (Facebook, Google, Twitter či Linkedln). Implicitního pak vytváří knihovna sama, a to tak, že vygeneruje náhodné přihlašovací jméno a náhodné heslo. Stejně jako u *Firebase*, i zde bylo využito možnosti registrace bez použití třetí strany.

Následující kód 12 při volání `.user().create()` zakládá nového uživatele s předaným nickem a heslem. Pokud server vrátí pozitivní odezvu, v `onSuccess()` ihned tohoto nového uživatele využijeme a pošleme obratem na server i žádost o vložení e-mailové adresy. K tomu slouží metoda `.user().update()`. E-mailová adresa v prostředí *Kinvey* nezastává roli identifikátoru a její zadání je volitelné.

Kód 12: Vytvoření nového uživatele v prostředí Kinvey

```
mKinveyClient.user().create(alias, password, new KinveyUserCallback
    () {
    @Override
    public void onSuccess(User user) {
        mKinveyClient.user().put("email",e_mail.getText().toString
            ());
        mKinveyClient.user().update(new KinveyUserCallback() {
            @Override
            public void onSuccess(User u) {
                Intent result = new Intent();
                result.putExtra("Registered",true);
                setResult(2,result);
                finish();
            }
            @Override
            public void onFailure(Throwable e) {}
        });
    }
    @Override
    public void onFailure(Throwable throwable) {
        CharSequence text = "Could not sign up.";
        Toast.makeText(getApplicationContext(), text, Toast.
            LENGTH_SHORT).show();
    }
});
```

Proces přihlašování v kódu 13 se zahajuje zavoláním `.user().login()`. Předáváme username a heslo. Pokud dostaneme pozitivní odezvu a autentizace prošla, přecházíme na aktivitu *Userpanel*.

Kód 13: Proces přihlašování v prostředí Kinvey

```
mKinveyClient.user().login(alias, password, new KinveyUserCallback
    () {
    @Override
    public void onSuccess(User user) {
        CharSequence text = "Welcome back, " + user.getUsername() +
            ".";
        Toast.makeText(getApplicationContext(), text, Toast.
            LENGTH_SHORT).show();
        goToUserPanel();
    }
    @Override
    public void onFailure(Throwable t) {
        CharSequence text = "Wrong username or password.";
        Toast.makeText(getApplicationContext(), text, Toast.
            LENGTH_SHORT).show();
    }
});
```

Přes metodu `.user().isUserLoggedIn()` zavolanou na proměnné `mKinveyClient` se ptáme, jestli existuje v paměti uložená instance nějakého uživatele, a vrací true/false.

Získání údajů o přihlášeném uživateli v kódu 14 provádíme voláním `.user().retrieve()`. V metodě `onSuccess()` si je následně můžeme vytáhnout.

Kód 14: Získání údajů o uživateli

```
mKinveyClient.user().retrieve(new KinveyUserCallback() {
    @Override
    public void onSuccess(User user) {
        textViewShowUser.setText(getString(R.string.logged)+" "+
            user.get("username"));
        username = user.get("username").toString();
        email = user.get("email").toString();
    }
    @Override
    public void onFailure(Throwable e) {}
});
```

Přístupová práva v prostředí *Kinvey* jsou řešena jinak než tomu bylo u *Firebase*. V základu jsou 4 možnosti nastavení práv ke každé jednotlivé kolekci (Kinvey, 2016):

- Pouze čtení – data je možné pouze vidět, ale všechny operace jsou zakázány,
- Sdílené – modifikovat danou entitu smí pouze její vlastník, cizí entity lze jen číst,
- Soukromé – každý uživatel vidí jen své vytvořené entity,
- Veřejné – každý uživatel vidí vše a může modifikovat vše.

V naší aplikaci se vyskytuje kombinace sdílených a veřejných. V některých případech je vyloženě nutné mazat i cizí vytvořené záznamy (například záznam o přidání do seznamu přátel, kde je vlastník odesilatel, ale zpracoval je příjemce), _acl by nás jinak zablokoval.

Pokud bychom přece jenom neradi vystavovali data potenciálnímu zneužití, dají se tato práva ručně přepsat v parametru _acl (například je možné specifikovat, kdo z uživatelů získá zapisovací práva k dané entitě). Tyto změny se provádějí přes `KinveyMetaData` – konkrétně zavoláním `.getMeta().getWrite()` nebo `.getMeta().getRead()` nad datovým modelem dané entity (Kinvey, 2016). O datových modelech v prostředí *Kinvey* v následující kapitole.

Jednou přihlášený uživatel zůstává v paměti aplikace tak dlouho, než je zavoláno `mKinveyClient.user().logout().execute()`.

Pakliže by nastalo, že se nějakým nedopatřením bez provedení `.logout().execute()` dostaneme opět na aktivitu *Login* a chtěli bychom vyplnit přihlašovací údaje znovu, služba nás nepustí dál. Nepomůže ani uvolnění aplikace z paměti, ani smazání dat. Je nutné celou aplikaci nainstalovat znovu. Toto se nedopatřením několikrát stalo při tvorbě aplikace.

### 4.3.3 Zasílání, přijímání a aktualizace dat

Strukturu dat jsme si popsali již v kapitole 4.3.1. Zaměříme se tedy jen na tvorbu datového modelu, který je zde nutný. Konkrétní dat. model pro přidání záznamu požadavku o přidání do přátel může vypadat jako v kódu 15.

Kód 15: Tvorba datového modelu

```
import com.google.api.client.json.GenericJson;
import com.google.api.client.util.Key;
import com.kinvey.java.model.KinveyMetaData;

public class NewFriend extends GenericJson {

    @Key("_id")
    private String id;
    @Key("_acl")
    private KinveyMetaData.AccessControlList acl;
    @Key
    private String friend1;
    @Key
    private String friend2;
    @Key
    private String friend1Email;
    @Key
    private String friend2Email;

    public NewFriend(){}
```

```
    public NewFriend(String friend1, String friend2, String email1,
        String email2){
        this.friend1 = friend1;
        this.friend2 = friend2;
        this.friend1Email = email1;
        this.friend2Email = email2;
    }

    public String getId(){
        return id;
    }
    ...
}
```

Samotný datový model rozšiřuje třídu `GenericJson`. To nám dovolí pracovat s *Kinvey* backendem. Klíčové slovo `@Key` u každé proměnné nám říká, že se pod tímto názvem v dané kolekci nachází sloupec téhož označení. Pokud bychom u některé proměnné zapomněli toto nadeklarovat, po odeslání vytvořeného objektu se tento parametr vynechá. Stejně jako u *Firebase*, i zde je nutné nechávat prázdný veřejný konstruktor. `KinveyMetaData.AccessControlList` je volitelná deklarace, využije se v případě, že se chystáme měnit přístupová práva pro jednotlivé entity v kolekci, jak jsme si popsali v kapitole 4.3.2.

Objekt pro poslání na server tvoříme přes konstruktor (kód 16), do kterého předáme potřebné parametry. Následně vytváříme `AppData` rozhraní, které realizuje přenosy. Prvním parametrem je název kolekce v backendu a druhým struktura dat. modelu entity. Rozhraní `AppData` obsahuje 4 druhy volání (Kinvey, 2016):

- `.save(objekt, new KinveyClientCallback))` – vložení nové entity nebo modifikace stávající,
- `.getEntity(identifikátor, new KinveyClientCallback())` – získání konkrétní entity podle identifikátoru,
- `.get(query, new KinveyListCallback())` – získání skupiny entit, dotaz se skládá do query, bez specifikace query je možné fetchnout celý obsah dané kolekce,
- `.delete(identifikátor, new KinveyDeleteCallback())` – smazání entity podle identifikátoru.

Nutno v tomto bodě podotknout, že žádné ze zmíněných volání nijak neovlivní data v zařízení. O jejich aktualizaci se musíme postarat sami. Po získání kladné odezvy ze serveru dostáváme námi vytvořený objekt nazpět a vložíme jej do nachystaného adaptéru.

Kód 16: Tvorba nové diskuse

```
Discussion dsc = new Discussion(edittext.getText().toString(),
    username);
AsyncAppData disc = mKinveyClient.appData("discussions",Discussion.
    class);
disc.save(dsc, new KinveyClientCallback() {
    @Override
    public void onSuccess(Discussion newDiscussion) {
        adapter.add(newDiscussion);
    }
    @Override
    public void onFailure(Throwable e) {}
});
```

Modifikace (aktualizace) dat probíhá totožně jako v případě jejich prvotní tvorby, viz kód 17. Pokud máme objekt již k dispozici, například přes námi užívaný adaptér, stačí provést změny a zavolat kód výše. Pokud objekt k dispozici ale nemáme, a přesto známe jeho ID, zavoláme druhou metodu ze seznamu, `.getEntity()`. Tím si daný objekt z kolekce získáme. Poté provedeme potřebné změny a opět přes `.save()` odesíláme zpět.

Kód 17: Aktualizace dat podle předaného ID

```
AsyncAppData newMsg = mKinveyClient.appData("friendMessages",
    Message.class);
newMsg.getEntity(msg.getId(), new KinveyClientCallback() {
    @Override
    public void onSuccess(Message result) {
        // v získaném resultu zmenime data pres setry a zavolame
            dalsi callback, tentokrat .save() a postupujeme stejne
        }

    @Override
    public void onFailure(Throwable error) {}
});
```

Naše aplikace využívá převážně listview k zobrazení dat. Na listview jsou třeba adaptéry. V případě *Firebase* se všechno obešlo přes `FirebaseAdapter`. Kinvey bohužel nic takového nemá. Pro každý list musíme vytvářet custom adaptéry. Přestože jejich tvorba není nijak náročná, třídy přibývají a přibývají. Čím více máme listview, tím více máme adaptérů. Dále jsme si již řekli, že Kinvey nikterak nemění jednou natažená data. Jeho asynchronní metody pouze zprostředkovávají přenosy.

Z tohoto důvodu jsme na aktivitách, kde bylo potřeba zprávy udržovat neustále aktuální, zavedli vlákno (kód 18). Jediné, co toto vlákno dělá, je, že v pravidelném intervalu 3 vteřin volá metodu `callForData()`, přes kterou následně opakovaně fetchuje data do adaptéru. Je nastaveno počáteční zpoždění 5 vteřin – při otevření aktivity se tato metoda ihned zavolá pro načtení prvotních dat, není tedy nutné další brzké volání.

Kód 18: Zavedení asynchronního vlákna

```
public void startNewThread(){
    new Thread(new Runnable() {
        public void run() {
            new Timer().scheduleAtFixedRate(new TimerTask() {
                @Override
                public void run() {
                    callForData();
                }
            },5000,3000);
        }
    }).start(); }
```

Řečená metoda `callForData()` v kódu 19 ze všeho nejdřív uvolní obsah listu, kdybychom toto neprovedli, data se budou do nekonečna na výstupu duplikovat s každým dalším voláním. Následně sestavíme query. Query podporuje logické operace, řazení a porovnávání. Jedná se o mocný nástroj pro specifikaci, jaká data požadujeme stáhnout. Objekty získáváme v poli, které pak překlopíme do nachystaného adaptéru. Na závěr upozorníme adaptér na změnu dat v listview.

Kód 19: Fetchnutí dat do listview přes custom adaptér

```
public void callForData(){
    Query q;
    Query query1 = new Query().equals("friendNick",friendNick);
    Query query2 = new Query().equals("myNick",myNick);
    Query query3 = new Query().equals("friendNick",myNick);
    Query query4 = new Query().equals("myNick",friendNick);
    q = (query1.and(query2)).or(query3.and(query4));
    q.addSort("date", AbstractQuery.SortOrder.ASC);
    AsyncAppData data = mKinveyClient.appData("friendMessages",
       Message.class);
    data.get(q, new KinveyListCallback() {
        @Override
        public void onSuccess(Message[] msgs) {
            msgArray.clear();
            adapter.addAll(msgs);
            adapter.notifyDataSetChanged();
        }
        @Override
        public void onFailure(Throwable t) {}
});
```

Pro účely mazání entit z kolekce využijeme poslední metody z rozhraní `AppData`, `.delete()`. Postup si ukážeme při smazání diskuse v kódu 20, stejně jako tomu bylo u *Firebase*. Nejprve si získáme ID dané diskuse a to si uložíme do dvou query. Pokaždé potřebujeme identifikovat jiný sloupec v entitě.

První volání `.delete()` smaže entity zpráv z kolekce discussionMessages, jejichž ID v kolonce `discId` je totožné jako ID diskuse. Pokud operace prvního volání skončí úspěšně, provedeme druhé volání a smažeme samotnou entitu dané diskuse.

Kód 20: Mazání záznamů z databáze

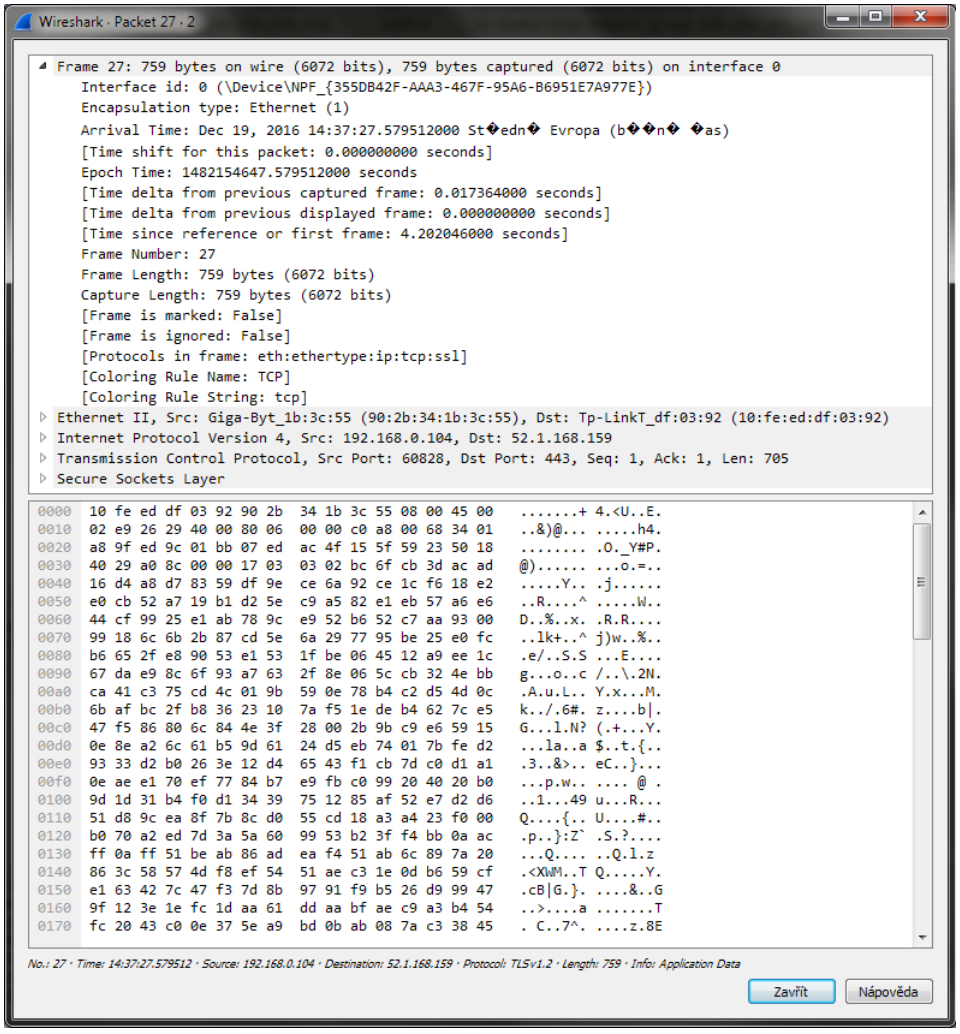
```
final Discussion longClickDisc = adapter.getItem(pos);
final Query query1 = new Query().equals("discId",longClickDisc.
    getId());
final Query query2 = new Query().equals("_id",longClickDisc.getId()
    );
AsyncAppData delDiscussionMessages = mKinveyClient.appData("
    discussionMessages", DiscussionMessage.class);
delDiscussionMessages.delete(query1, new KinveyDeleteCallback() {
    @Override
    public void onSuccess(KinveyDeleteResponse response) {
        AsyncAppData delDiscussion = mKinveyClient.appData("
            discussions", Discussion.class);
        delDiscussion.delete(query2, new KinveyDeleteCallback() {
            @Override
            public void onSuccess(KinveyDeleteResponse response) {
                Log.w(TAG, "Diskuse byla smazána.");
                adapter.remove(longClickDisc);
                adapter.notifyDataSetChanged();
            }
            public void onFailure(Throwable e) {
                Log.e(TAG, "Nepodarilo se diskusi smazat. ", e);
            }
        });
    }

    public void onFailure(Throwable e) {
        Log.e(TAG, "Nepodarilo se zprávy smazat. ", e);
    }
});
```

V poslední řadě se opět podíváme na samotný přenos. Opět stejný scénář, z diskuse *Sea* odesíláme zprávu: „Toto je zkusebni zprava pro zjisteni prenosu.“

```
27 14:37:27.579512 192.168.0.104    52.1.168.159      TLSv1.2   759 Application Data
29 14:37:27.703008 52.1.168.159     192.168.0.104     TCP        60 443→60828 [ACK] Seq=1 Ack=706 Win=370 Len=0
```

Obr. 20: Pakety přenosu zprávy z Wiresharku (Kinvey)

I *Kinvey* přenáší zprávy přes TLSv1.2 protokol (obr. 20), přenos je šifrovaný. Odesílaný paket (obr. 21) je velikostně jednou takový než u *Firebase*.




Obr. 21: Bližší pohled na paket z Wiresharku (Kinvey)

### 4.3.4 Offline režim

Nativně není cache ani offline provoz povolen. Obojího se dosahuje přes práci s `AppData`. Cachování má 6 režimů (Kinvey, 2016):

- `CachePolicy.NOCACHE` – žádné cachování, je nastaveno defaultně,
- `CachePolicy.CACHEONLY` – použijí se data pouze z cache paměti, nikoli z backendu,
- `CachePolicy.CACHEFIRST` – prvně se nahrají data z cache, pokud neexistují, zavolá se na backend, následně se cache aktualizuje,
- `CachePolicy.CACHEFIRST_NOREFRESH` – prvně se nahrají data z cache, pokud neexistují, zavolá se na backend, pakliže dojde ke stažení dat z backendu, nejsou v cache aktualizovány,
- `CachePolicy.NETWORKFIRST` – prvně se data stáhnou z backendu a následně uloží do cache, pokud není přístup k internetu, data se natáhnou z cache,
- `CachePolicy.BOTH` – prvně se provede natažení z cache a ihned poté z backendu, vždy dochází ke dvěma voláním.

Nastavení cache pro ukládání stažených zpráv mezi dvěma uživateli by vypadalo jako v kódu 21.

Kód 21: Zavedení cache

```
AsyncAppData data = mKinveyClient.appData("friendMessages", Message
    .class);
data.setCache(new InMemoryLRUCache(), CachePolicy.TYP REZIMU);
```

Offline režim, má-li být v aplikaci napojen, je nutné v první řadě nadeklarovat do `Manifestu` jako service. Má 3 možnosti nastavení (Kinvey, 2016):

- `OfflinePolicy.ONLINE_FIRST` – prvně se pokusí navázat spojení s backendem, pokud je tato akce úspěšná, aktualizuje lokální úložiště, v opačném případě se pokusí data vyhledat v lokálním úložišti
- `OfflinePolicy.LOCAL_FIRST` – prvně prohledá lokální úložiště, pokud data najde, načte si je, pokud ne, zavolá na server request
- `OfflinePolicy.ALWAYS_ONLINE` – lokální úložiště se nepoužije, všechny transakce se provádí přímo na backend.

Offline režim v kódu 22 se napojuje obdobně jako `CachePolicy`.

Kód 22: Zavedení lokálního úložiště

```
AsyncAppData data = mKinveyClient.appData("friendMessages", Message
    .class);
data.setOffline(OfflinePolicy.TYP REZIMU, new SqlLiteOfflineStore(
    getContext()));
```

### 4.3.5 Dokumentace a podpora

Dokumentace ve formě návodů je popsána dopodrobna. Nechybí ani tutoriály nebo ukázkové aplikace. Dokonce i sami vývojáři odpovídají na otázky pokládané na Stack Overflow. Jediná výtka je k referencím API dokumentace. Ta vytváří dojem, že je pouze, aby se neřeklo. Je suchá a krom názvů metod tam pomalu nic není.

### 4.3.6 Potenciální nedostatky

Ono ani tak nejde o nedostatek, jako spíše o skutečnost, že *Kinvey* bych nedoporučil pro data v reálném čase, která je třeba neustále aktualizovat. Což byl případ naší chatovací aplikace. Listener nezatěžuje provoz tolik jako volání v pravidelném intervalu, kde daná data pokaždé taháme všechna.

## 4.4 Parse Server

V práci uvedeme pouze zapojení *Parse Serveru* a všech jeho součástí, jelikož nastaly jisté komplikace zabraňující komunikaci mezi vnějškem a databází. K jejich hlubší analýze nebyl vzhledem k blížícímu se datu odevzdání práce čas.

Pro instalaci *Parse Serveru* byl použit VMware Workstation a linuxová distribuce Ubuntu 16.04 64-bit. Ubuntu bylo vybráno z důvodu dřívějších zkušeností. Podle návodu vývojářů Parse (2016) na GitHubu stačí před samotnou instalací následující komponenty:

- Node 4.5,
- MongoDB verze 2.6.X, 3.0.X nebo 3.2.6,
- Python 2.x.

Ve skutečnosti výše uvedené součásti nejsou dostačující. Alespoň ne v případě čisté instalace Ubuntu, kde během instalace došlo hned k několika komplikacím. Proces instalace *Parse Serveru* s MongoDB na localhost zahájíme přes terminál příkazem `npm install -g parse-server mongodb-runner`.

- Problém číslo 1, *npm* není nainstalován (spustíme `apt install npm`),
- Problém číslo 2, nainstalovaná verze *Node* je starší než minimální požadovaná, musíme spustit update,
- Problém číslo 3, nemáme podporu *Gitu* (`apt-get install git-all`).

Následují 2 hlášky o zastaralosti používaných graceful-fs a MongoDB. Doporučuje se udělat update na novější, ale nic z toho nebrání dokončení instalace. Pokračujeme příkazem `mongodb-runner start` (tento příkaz nelze spustit, pokud nedojde k update *Node*).

Spuštění již nyní nainstalovaného *Parse Serveru* zahájíme následujícím příkazem: `parse-server --appId APPLICATION_ID --masterKey MASTER_KEY`. APPLICATION_ID i MASTER_KEY volíme dle vlastního uvážení.

## 4.5 Test provozu

Byl proveden malý zátěžový test na otestování spotřeby baterie a množství zpracovaných dat na námi zhotovené aplikaci. Tento test trval 3 hodiny. Zprávy se na server zasílaly v pravidelném intervalu 10 minut. Obsah zpráv byl pokaždé jiný.

V případě *Firebase* se za vyměřenou dobu přeneslo 0,10 MB dat a výdrž baterie neklesla o více než 1 %.

*Kinvey* na tom byl dle očekávání hůře. Přeneslo se asi 15 MB a výdrž baterie klesla o zhruba 5 %. Hlavní příčinou zvýšeného provozu je pravidelná komunikace způsobená námi nastavenou obnovovací frekvencí pro natažení dat na listview přes vlákno. Velikost přenášených dat rostla konstantně se zvyšujícím se počtem zpráv v databázi.

## 4.6 Zhodnocení zkoumaných parametrů

V následující tab. 1 uvedeme výsledky srovnání mezi *Firebase* a *Kinvey* nad námi použitou aplikací. *Parse Server* není uveden, protože nebylo možné testy provádět.

Tab. 1: Zhodnocení parametrů

| **Parametr** | **Firebase** | **Kinvey** |
|---|---|---|
| náročnost služby na síťový provoz | 1 | 3 |
| délky programových kódů | 1 | 2 |
| množství informací včetně podpory | 1 | 2 |
| zkušenost při práci s danými službami | 1 | 1 |
| chování při ztrátě spojení | 1 | 2 |
| zhodnocení potenciálních nedostatků | 2 | 1 |
| přístup k synchronizaci dat | vhodný | nevhodný |
| formát přenášených dat | vhodný | vhodný |

Význam jednotlivých hodnot:

- 1 – dobré, 2 – průměrné, 3 – špatné
- vhodné – vhodné pro aplikaci s daným charakterem
- nevhodné – nevhodné pro aplikaci s daným charakterem

# 5 Výsledky práce

MBaaS přináší řešení mnoha nesnází, pokud vývojář postrádá vlastního zázemí. Přenechání části břemene na poskytovateli služeb šetří čas, energii a část nákladů spojených s provozem vlastních serverů.

Největší rozdíl mezi *Firebase* a *Kinvey* je bezesporu princip samotného zpracování zpráv. Na straně *Firebase* vidíme listenery – ty nepřetržitě sledují danou pozici ve stromové struktuře databáze backendu a naslouchají na provedené změny. Jakmile se nalezne odchylka od toho, co si lokálně pamatují, provedou nutné úkony pro svou synchronizaci. Až na počáteční načtení dat (za předpokladu prázdné cache nebo nepoužívání offline režimu) nikdy nedostaneme celou větev dat opakovaně. Při prováděných změnách pracujeme vždy jen s daným objektem (nebo jeho potomky).

Naproti tomu *Kinvey* ničím na způsob listeneru nedisponuje, volání musíme zahajovat my. Nedokážeme ani určit, která data již máme a která vznikla nově, vždy pracujeme se vším, co ve vytvořené query nalezneme. Všechno zlé je i k něčemu dobré, větší objem přenášených dat kompenzuje lepší kontrola nad samotnými daty. V MongoDB není nutné uchovávat duplicitní záznamy, jako tomu bylo u *Firebase*.

Přenášená data šifrují obě zkoumané služby totožně přes protokol TLSv1.2. Přenášené packety z kapitol 4.2.3 a 4.3.3 potvrzují naše tvrzení výše, že při práci s *Kinvey* můžeme očekávat větší síťový provoz.

Ani jedna ze zkoumaných služeb sama od sebe do lokálního úložiště data neukládá. Vždy je nutné tuto informaci předávat. Liší se i fungováním cache. *Firebase* ji využívá automaticky, je dostupná ihned po napojení služby. Nemá žádný režim pro nastavení. *Kinvey* ji sice nemá automatickou, ale nabízí hned několik režimů jejího nastavení v závislosti na tom, na jaké bázi má vytvářená aplikace pracovat, a podle toho ji přizpůsobit.

Podle naměřených hodnot se dá s jistotou říct, že *Firebase* je méně náročný než *Kinvey*. Spotřeba baterie byla zcela nepatrná a počet přenášených dat se lišil mnohonásobně. Samozřejmě musíme brát v úvahu i charakter vytvořené aplikace, na kterou se *Kinvey* ve výsledku příliš nehodil.

O dokumentaci a podpoře byla již řeč v kapitolách 4.2.5 a 4.3.5. Dodáme jen, že se vývojáři Firebase o svůj výtvor náležitě starají. Nový obsah vydávají pravidelně. Převážně jde o videa na jejich Youtube kanále. Snad jim tento zápal vydrží i nadále a z *Firebase* se stane ještě lepší služba, než jaká je dnes. Pokrok ostatně přinesl i přechod pod hlavičku Google.

Z pohledu délek kódů byl *Kinvey* roztahanější, na druhou stranu dovoloval na základě jednoho volání uskutečňovat další. Měli jsme mnohem větší kontrolu nad prováděnými akcemi. Nepříjemností byla i skutečnost, že pro každý jednotlivý listview bylo nutné vytvoření třídy custom adaptéru.

A máme-li srovnat zkušenosti získané při práci s těmito dvěma službami, jako člověk, který se potýkal s touto problematikou poprvé, můžu jednoznačně říct, že jsem si odnesl mnohé. Ani z jedné služby nemám špatný pocit, každá má své pro a proti a určitě bych jmenované poskytovatele doporučil.

# 6 Závěr

Cílem této práce bylo srovnání MBaaS na současném trhu. Bohužel nebylo možné do práce zahrnout větší množství poskytovatelů, a to z důvodu, že varianty zdarma mají jen někteří. Původně se v práci měl objevit i *Parse Server*, open source duchovní nástupce končícího *Parse*. Vzhledem k určitým komplikacím při jeho zprovoznění v prostředí VMware Workstation na adrese localhostu bylo od tohoto záměru upuštěno.

Služby se mohou lišit principy při přenosu zpráv, formou přenášených zpráv, strukturou databáze, cenovou přístupností a dalšími aspekty. Všechny ale mají jeden společný jmenovatel – odklonit alespoň část práce při vývoji aplikací na nejrůznějších platformách.

Z výsledků shrnutých v kapitole 5 jednoznačně vyplývá, že *Firebase* zavedeme do aplikací, u nichž očekáváme nepřetržitou komunikaci mezi klienty a backendem. V opačném případě můžeme zvolit zkoumaný *Kinvey*, s nímž máme větší kontrolu nad přenášenými daty.

Výběr správné MBaaS bude záležet na možnostech její integrace. Služby s označením Enterprise (například *Kinvey* nebo *Backendless*) přinášejí začlenění do podnikové logiky, a tedy se primárně soustředí na propojení s podnikovým software. *Firebase* je stále ve fázi vývoje a nelze odhadnout, kam až ji vývojáři posunou. Můžeme jen doufat, že jim dosavadní zápal vydrží a v nepříliš vzdálené budoucnosti o této službě budeme slýchat častěji.

# 7 Slovníček pojmů

API – Application Programming Interface
AWS – Amazon Web Services
BaaS – Backend as a Service
CSS – Cascading Style Sheets
FCM – Firebase Cloud Messaging
GCM – Google Cloud Messaging
HTML – HyperText Markup Language
HTTP – Hypertext Transfer Protocol
IoT – Internet of Things
J2EE – Java Platform, Enterprise Edition
J2ME SAX – Java Platform, Micro Edition Simple API for XML
JSON – JavaScript Object Notation
JSON ME – JSON for Java Platform, Micro Edition
KML – Keyhole Markup Language
LDAP – Lightweight Directory Access Protocol
MBaaS – Mobile Backend as a Service
MQ messaging – Message Queue mesaging
OS – operation system
REST – Representational state transfer
RTMP – Real Time Messaging Protocol
SDK – Software development kit
SGML – Standard Generalized Markup Language
SOAP – Simple Object Access Protocol
SQL – Structured Query Language
TLS – Transport Layer Security
XML – Extensible Markup Language

# 8 Seznam zdrojů

AnyPresence, Inc. *AnyPresence / Features* [online]. [cit. 2016-10-25]. Dostupné z: http://www.anypresence.com/features/.

Apple, Inc. *iCloud / Apple Developer* [online]. [cit. 2016-10-25]. Dostupné z: https://developer.apple.com/icloud/.

Backendless Corp. *Backendless Cloud Functional Limits / Backend as a Service Platform* [online]. [cit. 2016-10-23]. Dostupné z: https://backendless.com/pricing/backendless-cloud-functional-limits/.

Backendless Corp. *Products / Backend as a Service Platform* [online]. [cit. 2016-10-23]. Dostupné z: https://backendless.com/products/.

BlackBerry Limited. *Data sources – BlackBerry Native* [online]. [cit. 2015-11-20]. Dostupné z: http://developer.blackberry.com/native/reference/cascades/data_management_data_sources.html.

Built.io *Built.io Backend – Mobile Backend-as-a Service (mBaaS) for Enterprise* [online]. [cit. 2016-10-23]. Dostupné z: https://www.built.io/products/backend/overview.

CloudMine, Inc. *CloudMine* [online]. [cit. 2016-10-23]. Dostupné z: https://cloudmineinc.com/platform/developer-tools/.

Crunchbase, Inc. *Firebase / crunchbase* [online]. [cit. 2016-10-23]. Dostupné z: https://www.crunchbase.com/organization/firebase.

FatFractal *Documentation / FatFractal* [online]. [cit. 2016-10-25]. Dostupné z: http://fatfractal.com/v2/documentation/.

FatFractal *Pricing Details / FatFractal* [online]. [cit. 2016-10-25]. Dostupné z: http://fatfractal.com/v2/pricing-details/.

Firebase, Inc. *Add Firebase to Your Project / Firebase* [online]. [cit. 2016-12-11]. Dostupné z: https://firebase.google.com/docs/android/.

Firebase, Inc. *Features / Firebase* [online]. [cit. 2016-10-23]. Dostupné z: https://firebase.google.com/features/.

Firebase, Inc. *Pricing / Firebase* [online]. [cit. 2016-10-23]. Dostupné z: https://firebase.google.com/pricing/.

Giddins, E. Samuel. *RestKit/RestKit – Github* [online]. [cit. 2015-11-20]. Dostupné z: https://github.com/RestKit/RestKit.

Gontar, M. *Parse XML file on BlackBerry – Stack Overflow* [online]. [cit. 2015-11-20]. Dostupné z: http://stackoverflow.com/questions/2757990/parse-xml-file-on-blackberry.

Google, Inc. *Android | Android Developers* [online]. [cit. 2016-12-15]. Dostupné z: https://developer.android.com/reference/packages.html.

Google, Inc. *Cloud Messaging | Google Developers* [online]. [cit. 2015-11-20]. Dostupné z: https://developers.google.com/cloud-messaging/.

Helmy, D. *Parsing JSON in Windows Phone Apps – AdricaAps* [online]. [cit. 2016-12-23]. Dostupné z: https://blogs.msdn.microsoft.com/africaapps/2013/02/25/parsing-json-in-windows-phone-apps/.

IDC Research, Inc. *IDC: Smartphone OS Market Share 2015, 2014, 2013, and 2012* [online]. [cit. 2015-11-20]. Dostupné z: http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

Kii *Kii Documentation* [online]. [cit. 2016-10-25]. Dostupné z: http://docs.kii.com/en/.

Kinvey *About – Enterprise Mobile Backend as a Service | Kinvey* [online]. [cit. 2016-10-23]. Dostupné z: https://www.kinvey.com/about/.

Kinvey *Android Guides | Kinvey* [online]. [cit. 2016-11-12]. Dostupné z: http://devcenter.kinvey.com/android/guides/.

Kinvey *Pricing – Enterprise Mobile Backend as a Service | Kinvey* [online]. [cit. 2016-10-23]. Dostupné z: https://www.kinvey.com/pricing/.

Microsoft Corporation *Announcing the publication of Parse Server with Azure Managed Services* [online]. [cit. 2016-10-23]. Dostupné z: https://azure.microsoft.com/en-us/blog/announcing-the-publication-of-parse-server-with-azure-managed-services/.

Midnight Coders, Inc. *Communication and Media Libraries for iOS User Guide* [online]. [cit. 2015-11-20]. Dostupné z: http://www.themidnightcoders.com/fileadmin/docs/ios/.

Parse, Inc. *Moving On* [online]. [cit. 2016-10-23]. Dostupné z: http://blog.parse.com/announcements/moving-on/.

Parse, Inc. *Parse* [online]. [cit. 2016-12-23]. Dostupné z: https://parseplatform.github.io/sdks.

Parse, Inc. *Parse Server Guide* [online]. [cit. 2016-12-26]. Dostupné z: https://github.com/ParsePlatform/parse-server/wiki/Quick-Start.

READWRITE *The Rise of Mobile Cloud Services: BaaS Startups Grow Up - ReadWrite* [online]. [cit. 2016-12-23]. Dostupné z: http://readwrite.com/2012/04/17/mobile-backend-as-a-service-ec/.

# Přílohy

# A Zdrojové kódy

Zdrojové kódy vytvářených aplikací spolu s přihlašovacími údaji jsou zabaleny v archívu .zip v elektronické odevzdávárně a na přiloženém CD. Archív obsahuje:

- kompletní kódy k aplikaci se službou Firebase,
- kompletní kódy k aplikaci se službou Kinvey,
- txt soubor s přihlašovacími údaji.