VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Aplikace pro šifrovanou komunikaci

Application for Encrypted Communication

2019 Jakub Mrázek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **Jakub Mrázek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R059 Mobilní technologie

Téma: Aplikace pro šifrovanou komunikaci
Application for Encrypted Communication

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem bakalářské práce je vytvořit aplikaci pro operační systém Android se zaměřením na multimediální funkce a bezpečnost. Specifikace zadání bakalářské práce jsou následující:

1. Navrhněte šifrované VPN spojení se serverovou platformou prostřednictvím OpenVPN s využití certifikátů.
2. Navrhněte přenos audio hovorů s využitím SIP protokolu a napojením na SW PBX Asterisk.
3. Implementujte podporu pro přenos video hovorů.
4. Implementujte podporu pro textovou komunikaci s využitím XMPP protokolu.
5. Implementujte podporu pro přenos souborů oproti s využitím Owncloud serveru.

Seznam doporučené odborné literatury:

[1] Watkin L., Koelle D. Practical XMPP. Packt Publishing 2016
[2] Peicevic A. Introduction to Asterisk: Learn how to set up your own PBX telephone system. CreateSpace Independent Publishing Platform 2017

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Lukáš Kapičák**

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2019

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * 1D16 * OSTRAVA *

prof. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2019 .....

Tímto bych rád poděkoval vedoucímu mé bakalářské práce panu Ing. Lukáši Kapičákovi za odborné rady, trpělivost a čas strávený našimi konzultacemi.

Abstrakt

Cílem této bakalářské práce je vytvořit aplikaci pro operační systém Android, která umožňuje komunikaci přes šifrované OpenVPN spojení. Aplikace podporuje textovou komunikaci, hlasovou komunikaci, přenos videa a sdílení souborů mezi uživateli. První polovinu práce tvoří teorie týkající se využitých technologií, mezi které patří OS Android, OpenVPN, SIP protokol pro hlasovou a video komunikaci, XMPP protokol pro textovou komunikaci a Owncloud pro sdílení souborů. Druhá polovina práce se zabývá samotnou implementací aplikace. Jsou zde popsány API a knihovny, které byly při práci využity, struktura výsledné aplikace a její fungování včetně ukázek.

Klíčová slova: Android, OpenVPN, VPN, SIP, XMPP, Owncloud, Openfire, Asterisk, šifrování, komunikace

Abstract

The objective of this thesis is to develop an application for Android operating system that allows to communicate via encrypted OpenVPN connection. Application supports text messaging, voice communication, video transfer and sharing files between users. The first part of this thesis consists of theoretical description of used technologies including Android OS, OpenVPN, SIP protocol for voice and video communication, XMPP protocol for text messagging and Owncloud for file sharing. The second part deals with the implementation of the application itself. It describes the used API and libraries, structure of the final application and it's functioning including pictures.

Key Words: Android, OpenVPN, VPN, SIP, XMPP, Owncloud, Openfire, Asterisk, encryption, communication

Obsah

Seznam použitých zkratek a symbolů

OS	–	Operační systém
Inc	–	Incorporated
LLC	–	Limited Liability Company
JVM	–	Java Virtual Machine
API	–	Application Programming Interface
HAL	–	Hardware Abstraction Layer
VPN	–	Virtual Private Network
SSL	–	Secure Sockets Layer
TLS	–	Transport Layer Security
ABI	–	Application Binary Interface
ART	–	Android Runtime
DVM	–	Dalvik Virtual Machine
JIT	–	Just-In-Time
AOT	–	Ahead-Of-Time
SMS	–	Short Message Service
IDE	–	Integrated Development Environment
PPTP	–	Point-to-Point Tunneling Protocol
SSTP	–	Secure Socket Tunneling Protocol
L2TP	–	Layer Two Tunneling Protocol
IPSec	–	Internet Protocol Security
IKEv2	–	Internet Key Exchange version 2
AES	–	Advanced Encryption Standard
PKI	–	Public Key Infrastructure
CA	–	Certificate Authority
ISO	–	International Organization for Standardization
OSI	–	Open Systems Interconnection
IP	–	Internet Protocol
LAN	–	Local Area Network
UDP	–	User Datagram Protocol
TCP	–	Transmission Control Protocol
VoIP	–	Voice over Internet Protocol
SIP	–	Session Initiation Protocol
UA	–	User Agent
UAC	–	User Agent Client
UAS	–	User Agent Server
URI	–	Uniform Resource Identifier

DNS	–	Domain Name Servers
RTP	–	Real-Time Transport Protocol
RTCP	–	Real-time Transport Control Protocol
QoS	–	Quality of Service
XMPP	–	Extensible Messaging and Presence Protocol
XML	–	Extensible Markup Language
XEP	–	XMPP Extension Protocol
JID	–	Jabber Identifier
IQ stanza	–	Info/Query stanza
WebDAV	–	Web Distributed Authoring and Versioning
URL	–	Uniform Resource Locator
VšB-TUO	–	Vysoká škola báňská - Technická univerzita Ostrava
GPLv2	–	General Public License version 2
SWIG	–	Simplified Wrapper and Interface Generator
MIT	–	Massachusetts Institute of Technology
UI	–	User Interface

Seznam obrázků

1 Úvod

Vlastnit chytrý telefon je dnes samozřejmostí, přičemž primární funkcí těchto zařízení je poskytnout svým vlastníkům co nejširší možnosti komunikace. Se stále rostoucím počtem uživatelů chytrých zařízení však rostou i nároky na bezpečnost. Cílem této bakalářské práce je vyvinout aplikaci pro operační systém Android, která by uživatelům umožňovala bezpečně komunikovat všemi běžnými formami.

První kapitola pojednává o operačním systému Android, pro který byla tato aplikace vyvíjena. Android je operační systém, který má dnes celosvětově nejvíce uživatelů a to z něj dělá cíl mnoha útoků, proto je třeba dbát při jeho využívání na bezpečnost.

Následující kapitoly se věnují jednotlivým technologiím, na kterých je tato aplikace postavena. Veškeré šifrování zajišťuje OpenVPN modul, kterému se, spolu s dalšími VPN protokoly, věnuji ve druhé kapitole. Následuje popis audio a video komunikace prostřednictvím SIP protokolu, textové komunikace s využitím XMPP protokolu a teoretickou část uzavírá kapitola zabývající se systémem Owncloud, který slouží pro sdílení souborů mezi uživateli.

Další kapitola se zabývá implementací aplikace a samotným procesem jejího zhotovení. Jsou zde popsány nejdůležitější komponenty aplikace, konfigurace serverů a popisy knihoven třetích stran, které byly do této práce implementovány.

V závěru přiblížím strukturu výsledné aplikace a popíšu způsob jejího fungování v běžném provozu.

2 Android

Operační systém Android, původně vyvíjený společností Android, Inc. a v roce 2005 odkoupený americkým technologickým gigantem Google LLC. Tento operační systém, určený především pro chytré telefony a tablety, ale také hodinky nebo televize, je dnes nejrozšířenějším operačním systémem pro mobilní zařízení na světě. Mobilní zařízení s operačním systémem Android představují více než 75% globálního trhu [2].

Android je open-source projekt, jeho zdrojový kód je tak k dispozici komukoli, což přispívá k jeho rozšíření mezi různými výrobci chytrých zařízení, kteří tak mají možnost Android přizpůsobit potřebám svého hardwaru, ale také přidávat své vlastní nadstavby, což jim umožňuje odlišit se od konkurence a zároveň svým zákazníkům nabídnout široké spektrum aplikací, které jsou na Android k dispozici. Velice populární je totiž Android také mezi vývojáři mobilních aplikací. V oficiálním online obchodu Google Play bylo ke třetímu čtvrtletí k dispozici ke stažení přes 2.1 milionů aplikací [3].

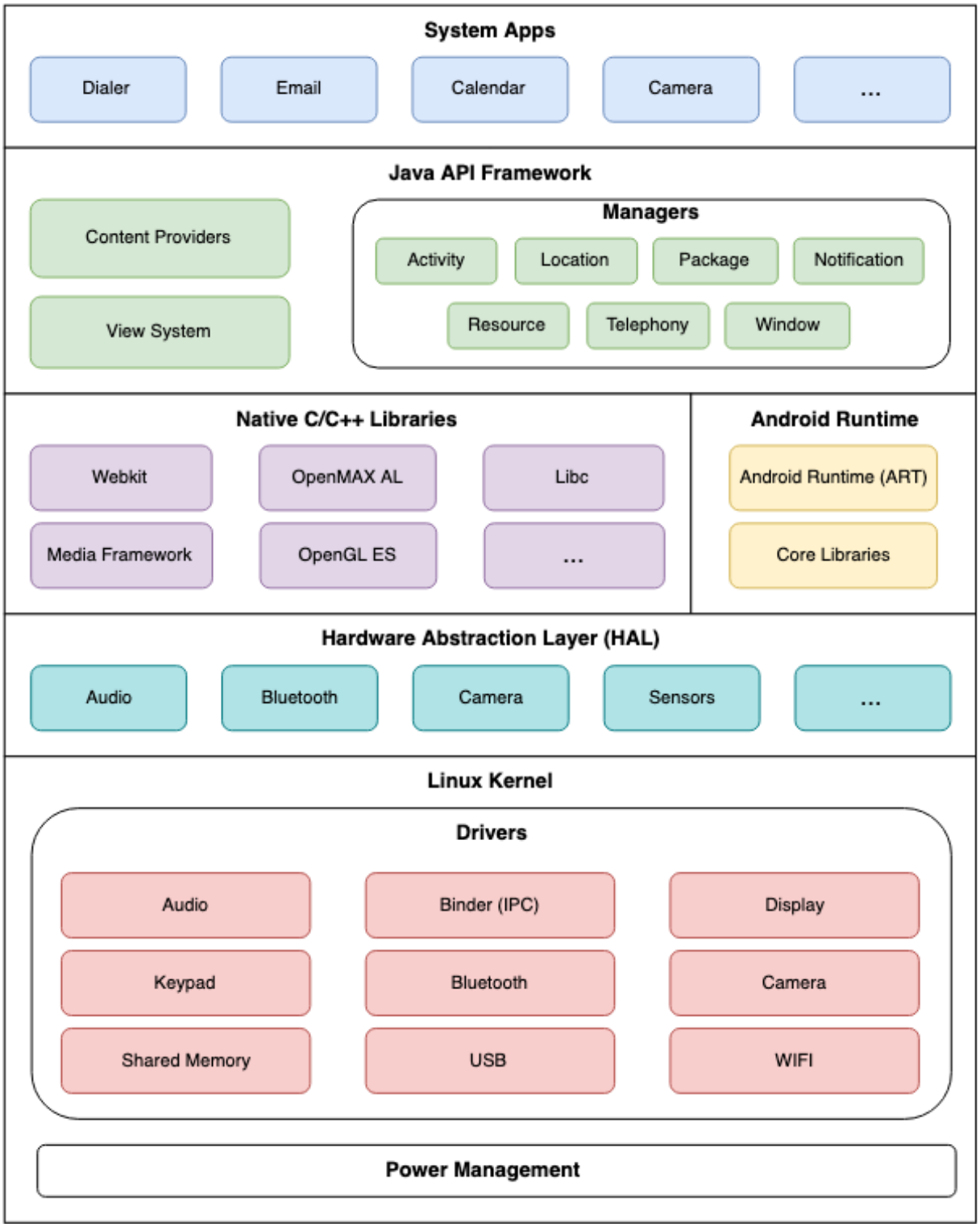
2 .1 Architektura Androidu

Z pohledu architektury je operační systém Android dělen do šesti vrstev, jak znázorňuje obrázek č. 1. Tyto vrstvy jsou:

- Linuxové jádro (Linux Kernel),
- Hardwarová abstrakční vrstva (Hardware Abstraction Layer - HAL),
- nativní C/C++ knihovny,
- Android Runtime,
- Java API Framework,
- systémové aplikace.

2 .1.1 Linuxové jádro

Operační systém Android je vystavěn na modifikovaném Linuxovém jádře neboli kernelu. Kernel zajišťuje nejdůležitější systémové funkce, mezi které patří správa napájení, správa paměti, správa procesů a obecně správa hardwaru zařízení prostřednictvím hardwarových ovladačů. Kernel Linuxu byl použit jak pro svou open-source licenci, tak pro svou prověřenost, která se pozitivně odráží na bezpečnosti celého systému, např. ve formě izolace jednotlivých procesů nebo modelu přístupových oprávnění založeného na uživatelských účtech [4].



Obrázek 1: Architektura platformy Android

2 .1.2 Hardwarová abstrakční vrstva

Aplikace přistupují k hardwaru zařízení prostřednictvím Java API Frameworku, ten ale nekomunikuje přímo s hardwarovými ovladači na úrovni jádra, jelikož hardware se může lišit u každého zařízení. Hardwarová abstrakční vrstva plní funkci prostředníka mezi těmito dvěma vrstvami. Zatímco výrobci zařízení HAL definuje rozhraní, které by měli pro svůj hardware implementovat [5], Java API Frameworku, potažmo aplikacím, poskytuje jednotné API, pomocí kterého mohou přistupovat k jednotlivým hardwarovým funkcím.

2 .1.3 Nativní C/C++ knihovny

Těmto knihovnám se říká nativní, protože obsahují tzv. nativní kód, nejsou tedy psány v jazyce Java, ale typicky v jazycích C a C++, ze kterých byly následně překompilovány do binárních souborů, tedy knihoven. Tyto knihovny jsou nezbytné pro chod některých systémových komponent, které jsou již součástí systému Android, jako například OpenGL ES pro práci s grafikou, Webkit, který poskytuje nástroje nezbytné pro prohlížení webu, nebo SQLite pro práci s databázemi. Jedná se však i o knihovny, které nejsou součástí systému Android, ale jsou součástí balíků externích aplikací, které tyto knihovny vyžadují pro svou funkčnost. To je případ i mé aplikace, pro kterou bylo třeba zkompilovat knihovny OpenVPN, OpenSSL, OpenH264 a PJSUA2.

Jelikož je systém Android provozován na různých zařízeních různých výrobců, kteří využívají odlišný hardware, musí se tomu přizpůsobit i nativní knihovny. Každý procesor podporuje nějakou instrukční sadu, které jsou v rámci Android sdružovány do tzv. ABI neboli Application Binary Interface. ABI definuje jakým způsobem má systém zpracovávat strojový kód aplikací za běhu [6]. Pokud chceme, aby aplikace, využívající nativní knihovny, běžela na všech android zařízeních, je třeba tyto knihovny zkompilovat pro každé ABI zvlášť. Knihovny, které využívá má aplikace, byly zkompilovány pro následující ABI:

- armeabi,
- armeabi-v7a,
- arm64-v8a,
- x86,
- x86_64,
- mips.

2 .1.4 Android Runtime

Android Runtime neboli ART je virtuální stroj, který slouží ke spuštění Dalvik Executable (.dex) souborů, což je již zkompilovaný kód aplikace [7], typicky psaný v jazyce Java nebo Kotlin. Jedním z největších výhod těchto multiplatformních jazyků je, že právě s využitím virtuálního

stroje je možné je spustit na jakémkoli zařízení, aniž by musel být kód nějak výrazně pozměněn nebo znovu zkompilován do spustitelného souboru.

Přestože virtuální stroj pro spuštění Java kódu již existoval v podobě Java Virtual Machine (JVM), pro systém Android byl z licenčních důvodů vyvinut stroj Dalvik Virtual Machine (DVM), který byl již od verze systému Android 4.4, neboli KitKat, nahrazen ART. Mezi hlavní výhody ART v porovnání s DVM patří menší paměťová náročnost a vyšší rychlost, které je dosaženo změnou v přístupu ke kompilaci souborů aplikace, kdy zatímco DVM tyto soubory kompiluje metodou JIT (Just-In-Time) až když jsou potřeba, ART používá metodu AOT (Ahead-Of-Time) a tudíž soubory kompiluje předem [8].

2 .1.5 Java API Framework

Tento framework je tvořen API (Application Programming Interface) psanými v Javě, prostřednictvím kterých aplikace přistupují ke všem funkcím systému Android. Mezi tato API patří například [9]:

- Resource Manager - umožňuje aplikaci přístup ke zdrojům jako jsou obrázky nebo textové řetězce.
- Notification Manager - umožňuje aplikacím zobrazovat zprávy v systémové liště.
- Activity Manager - řídí životní cyklus aplikace.
- Content Provider - zpřístupňuje aplikaci data jiných aplikací jako jsou například kontakty.

2 .1.6 Systémové aplikace

Aplikace, které jsou již součástí OS Android. Patří mezi ně aplikace pro správu e-mailu, SMS zpráv nebo kalendáře. Tyto aplikace poskytují uživateli základní funkce, které jsou u dnešního mobilního zařízení samozřejmost, jsou však na stejné úrovni jako kterékoli jiné aplikace třetích stran [9].

2 .2 Vývoj aplikace na Android

Obecně preferovaným vývojovým prostředím (IDE) pro Android je oficiální IDE Android Studio od Google, které bylo použito i k vývoji této aplikace. Toto IDE je přístupné zdarma a je založeno na velmi podobném prostředí IntelliJ IDEA. Další populární IDE pro vývoj na Android jsou například Eclipse nebo NetBeans. Android Studio jsem zvolil, jelikož jsem s ním měl dobré zkušenosti a nabízí vše co je potřeba k vývoji Android aplikací včetně Android Emulátoru, který byl použit k testování aplikace spolu s fyzickými zařízeními.

Oficiálním programovacím jazykem pro platformu Android je jazyk Java, ve kterém byla psána i tato aplikace. Od roku 2017 je oficiálně podporován také jazyk Kotlin, který je stejně jako Java jazykem multiplatformním a lze jej také kompilovat do Java byte kódu. Kotlin je modernější

jazyk, jehož hlavní výhodou je přehlednější a méně robustní syntaxe nebo také intuitivnější řešení tzv. „`NullPointerException`“, kdy lze již při deklaraci proměnné určit, zda je nebo není možné, aby obsahovala hodnotu „`null`“ [10], což znamená, že není třeba v kódu tak často kontrolovat hodnoty proměnných a v konečném důsledku to vede k větší stabilitě aplikace. Pro tuto aplikaci jsem zvolil jako primární jazyk Javu především kvůli svým zkušenostem s tímto jazykem a také množství dostupných zdrojů.

3 Virtuální privátní síť

Základem komunikace a jejího šifrování v rámci této aplikace je Virtuální privátní síť (VPN). VPN je technologie, která umožňuje klientům bezpečné spojení se vzdálenou privátní sítí prostřednictvím sítě veřejné. Toto má své využití například v korporátním prostředí, kdy lze klienty na vzdálených lokálních sítích, které nejsou fyzicky propojeny, propojit šifrovaným spojením přes veřejnou síť do jedné virtuální privátní sítě a tím docílit bezpečného sdílení privátních dat. VPN lze však využít k lepšímu zabezpečení síťové komunikace za všech okolností. Je-li VPN síť nakonfigurována tak, aby přesměrovávala veškerou síťovou komunikaci, lze identitu klientského zařízení v cílové síti, například síti Internet, úplně skrýt. V praxi tak klient komunikuje šifrovaným spojením (přes veřejnou síť) pouze s VPN serverem, který namísto klienta komunikuje s cílovým serverem a poté posílá výsledky komunikace zpět klientovi skrze toto privátní šifrované spojení. Tento princip se nazývá tunelování a je ilustrován na obrázku č. 2. Síťová aktivita klienta je tedy zabezpečena na dvou úrovních:

- Veškerá síťová komunikace klientského zařízení je směrována přes zvolený VPN server. Pokud jsou přenášená data vhodně šifrována, pak jsou i za předpokladu, že tuto komunikaci někdo odposlouchává, všechna data v bezpečí.
- Cílový server komunikuje pouze s VPN serverem, a ne přímo s klientem. Identita klienta je tedy známá pouze VPN serveru.

V rámci této aplikace byla VPN využita jednak pro navázání spojení se serverem, ze kterého jsou dostupné služby zajišťující komunikaci, a dále jako nástroj pro šifrování komunikace mezi jednotlivými klientskými zařízeními a serverem.

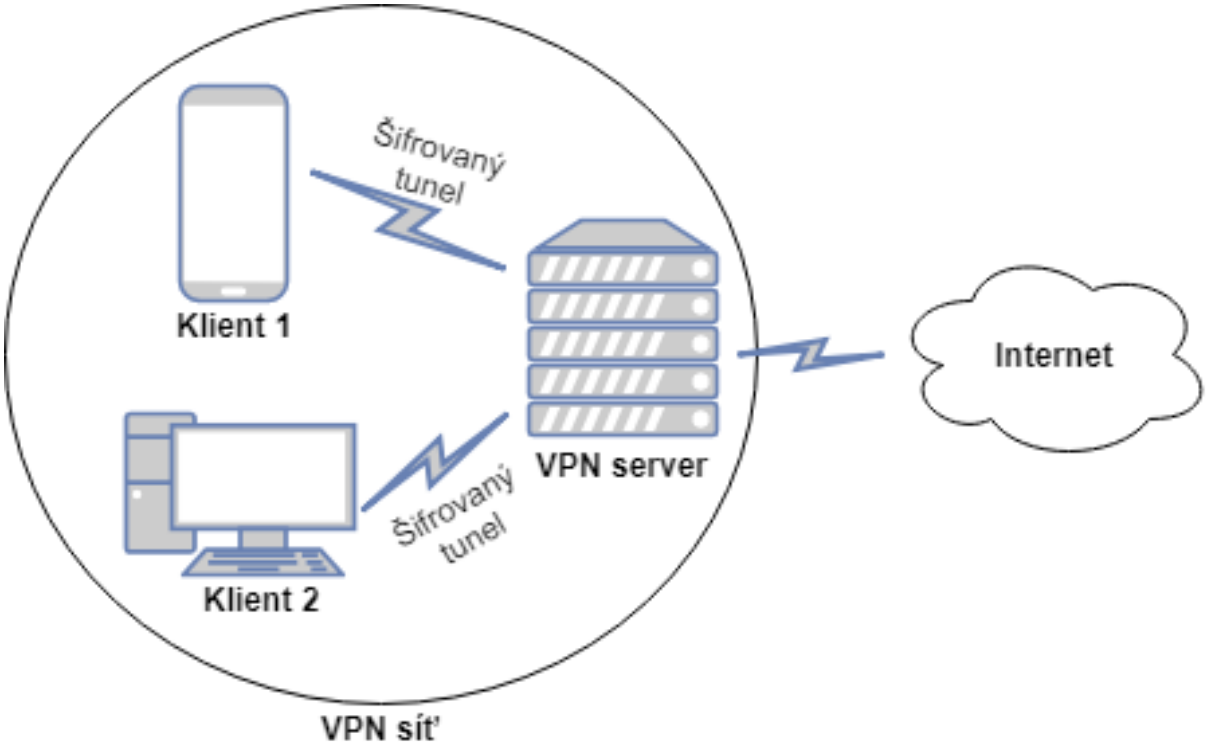
3 .1 VPN protokoly

VPN protokol určuje způsob, jakým jsou data přenášena mezi klientem a VPN serverem. Zde je výčet těch nejběžnějších:

- L2TP/IPSec
- SSTP
- IKEv2/IPSec
- PPTP
- OpenVPN

3 .1.1 PPTP

Point-to-point protocol (PPTP) je nejstarším z uvedených VPN protokolů původně vyvíjený společností Microsoft. Jeho podpora je implementována do mnoha běžných operačních systémů



Obrázek 2: Princip VPN

včetně systému Android, ale úroveň zabezpečení, kterou nabízí, není na dnešní standardy dostatečná. Právě kvůli nízké úrovni zabezpečení se však jedná o jeden z nejrychlejších z uvedených protokolů.

3 .1.2 L2TP/IPSec

Layer 2 Tunnel Protocol (L2TP) nezahrnuje sám o sobě žádné šifrování. Proto bývá implementován spolu s protokolem IPSec, který zajišťuje samotné šifrování dat. Tato kombinace protokolů je obecně považována za bezpečnou a je podporována řadou moderních operačních systémů, včetně OS Android, ve formě zabudovaného klienta. Nevýhodou tohoto protokolu je, že ke komunikaci využívá UDP (User Datagram Protocol) port 500, nedá se tedy snadno maskovat za jiný druh spojení, což usnadňuje jeho zablokování firewallem.

3 .1.3 SSTP

Secure Socket Tunneling Protocol (SSTP) je protokol vyvinutý společností Microsoft. K šifrování využívá protokol SSLv3/TLSv3 (Secure Sockets Layer/Transport Layer Security), lze jej tak nakonfigurovat na jakýkoli port a tím obejít pravidla firewallů. Podpora tohoto protokolu není zabudována v OS Android, je tedy třeba využít klientů třetích stran.

3 .1.4 IKEv2/IPSec

Internet Key Exchange version 2 je protokol, který stejně jako L2TP bývá implementován spolu s IPSec pro zajištění šifrování dat. Mezi jeho přednosti patří pokročilý systém opětovného navázání spojení, při ztrátě internetového připojení nebo změně sítě. Nevýhodou je opět nucené použití UDP portu 500.

3 .2 OpenVPN

Pro účely této aplikace byl použit protokol OpenVPN, který je na rozdíl od výše zmíněných protokolů, open-source, jeho kód může tedy kdokoli prohlížet a upravovat. Bezpečnost protokolu podporuje také to, že je možné jej nakonfigurovat k fungování na jakémkoli portu s využitím jak UDP, tak TCP (Transmission Control Protocol) protokolů, na rozdíl od L2TP/IPSEC nebo IKEv2/IPSec je tak možné takovéto VPN spojení skrýt za běžný síťový provoz a tím se přizpůsobit pravidlům firewallu. Tento protokol dále využívá OpenSSL šifrovací knihovnu, která podporuje řadu šifrovacích algoritmů, mezi které patří i doporučovaný Advanced Encryption Standard (AES). AES využívá k šifrování dat až 256-bitový klíč a pracuje na principu symetrického šifrování, to znamená, že k dešifrování dat používá stejný klíč. Při komunikaci dvou klientů je tudíž nezbytné tento klíč sdílet. Sdílení dat i klíče pro jejich dešifrování je v rámci této aplikace docíleno přes šifrované spojení na bázi SSLv3/TLSv1 protokolu, které je ustaveno pomocí certifikátů viz podkapitola 3 .2.1.

Alternativou autentizace s využitím certifikátů je využití tzv. statického klíče, který je vygenerován na straně serveru a poté distribuován na jednotlivé klienty ještě před ustavením tunelu [11]. Nevýhodou této metody je nižší bezpečnost v případě, že útočník je schopen kompromitovat klíče generované serverem. Na rozdíl od metody využívající certifikáty, totiž klient nemá jak ověřit autentičnost klíče, který obdržel od serveru.

3 .2.1 Autentizace s využitím certifikátů

Při implementaci autentizace s využitím certifikátů je nejprve třeba ustanovit tzv. Public Key Infrastructure (PKI), ta se skládá z:

- certifikátu a soukromého klíče pro server a každého klienta,
- certifikátu Certifikační Autority (CA) a klíče, který je použit k podpisu certifikátu serveru i klientů.

OpenVPN podporuje obousměrnou autentizaci, což znamená, že klient ověřuje certifikát serveru a naopak server ověřuje klientský certifikát. K dešifrování certifikátu protistrany je použit soukromý klíč. Nejprve je na obou stranách ověřeno, že certifikát protistrany je podepsán danou certifikační autoritou, poté probíhá kontrola informací obsažených v certifikátu, jako například jméno a typ certifikátu [12].

Proběhlo-li ověření certifikátu na obou stranách v pořádku, je mezi oběma stranami ustaveno SSLv3/TLSv1 spojení. Skrze toto spojení jsou odesílána šifrovaná data a privátní klíč nutný k jejich dešifrování na druhém konci [11].

Struktura a informace obsažené ve zmíněných certifikátech je dána standardem označovaným jako X.509. Kromě struktury certifikátu tento standard také definuje validační algoritmy certifikátů nebo podobu tzv. seznamu odvolaných certifikátů, které obsahují certifikáty, jež byly ze strany CA odvolány a ověřující instance by jim nadále neměla důvěřovat [13].

3 .2.2 Rozhraní TAP a TUN

OpenVPN podporuje virtuální síťová rozhraní TAP a TUN, rozdíl spočívá v tom, že rozhraní TAP pracuje s ethernetovými rámci na druhé (linkové) vrstvě modelu ISO/OSI, zatímco rozhraní TUN pracuje s IP pakety až na třetí (síťové) vrstvě modelu ISO/OSI [14]. V praxi se rozhraní TAP chová podobně jako fyzické rozhraní typu switch a lze jej využít například při vytváření mostu (angl. bridge), kdy je cílem spojit více vzdálených LAN sítí do jedné. Rozhraní TUN evokuje spíše chování fyzického rozhraní typu router, nelze jej využít k vytvoření mostu a podporuje pouze síťové protokoly IPv4 a IPv6.

Jelikož běžně dostupné mobilní zařízení s OS android nemá oprávnění tzv. super uživatele, nepodporuje ani rozhraní TAP [15], proto bylo OpenVPN v rámci tohoto projektu nakonfigurováno na rozhraní TUN, jehož možnosti jsou pro účely aplikace dostatečné.

4 Audio a video hovory

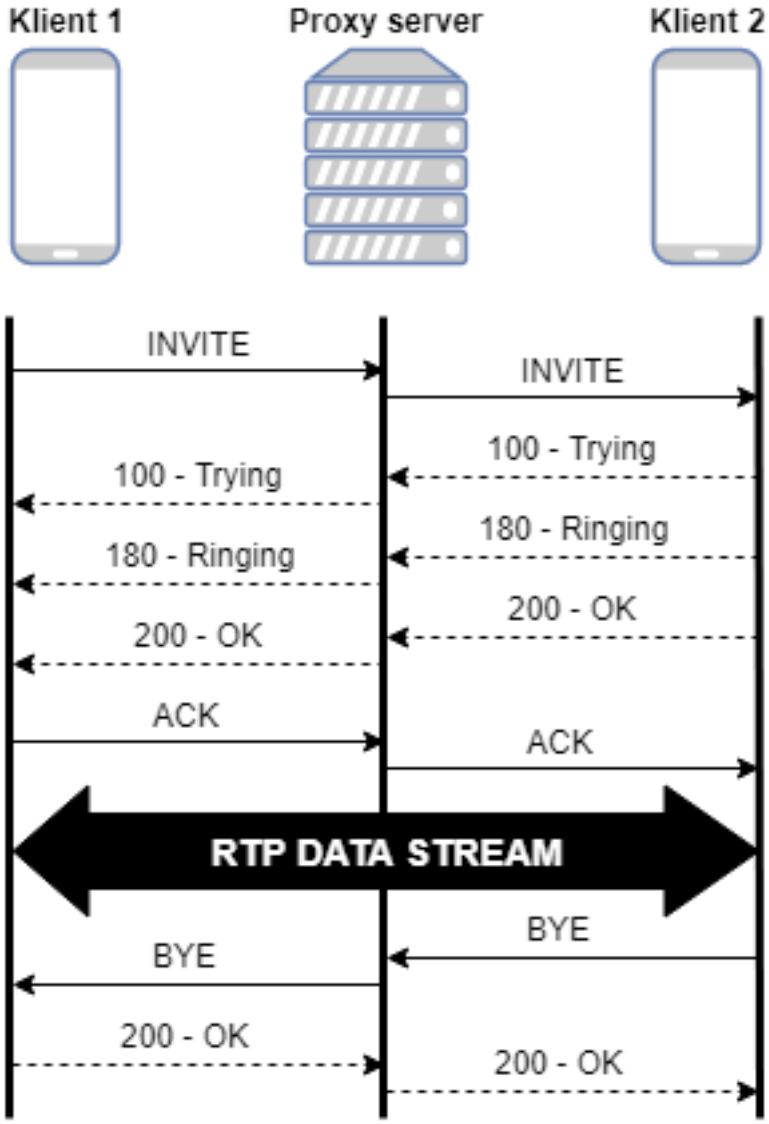
Mezi základní funkce zhotovené aplikace patří schopnost navázat a přenášet audio a video hovory metodou VoIP (Voice over Internet Protocol) s využitím SIP protokolu. Veškerá tato komunikace probíhá na čtvrté (transportní) vrstvě referenčního modelu ISO/OSI přes protokol UDP, který je vhodnější pro přenos dat v reálném čase, neboť na rozdíl od TCP nehlídá doručení každého datového paketu a přenos dat je tak rychlejší.

4 .1 SIP protokol

Session Initiation Protocol (SIP) je signalizační protokol používaný k zahajování, udržování a ukončování VoIP hovorů. Je možné jej využít jak pro audio a video hovory, tak i pro posílání textových zpráv. Komunikace přes SIP protokol zahrnuje následující prvky [16]:

- User Agent (UA) - jedná se o koncová zařízení v rámci SIP komunikace, na straně klienta (UAC) jej představuje tato aplikace, na straně serveru (UAS) jej představuje PBX server, v našem případě Asterisk.
- Registrar - server, na kterém probíhá registrace UA.
- Redirect server - zde probíhá případné přesměrování UA na jiný server.
- Proxy server - navazuje spojení s volaným serverem a informuje UAC o výsledku.
- Location server - ukládá informace o umístění a adrese UA pro Proxy a Redirect servery.

Každý uživatel je identifikován na základě URI (Uniform Resource Identifier). Obecně má SIP URI bez přidaných parametrů tvar „`sip:uživatel@host`“. Typický průběh SIP hovoru na příkladu Klienta 1 volajícího Klientovi 2 je znázorněn na obrázku 3. Ještě před samotným voláním se musí oba uživatelé registrovat na daném serveru (Registrar). Registrace probíhá vysláním zprávy typu REGISTER, obsahující přihlašovací údaje uživatele, na server. Pokud je uživatel v pořádku registrován, server zašle zpět odpověď s kódem 200 - „OK“. V tuto chvíli může začít pokus o navázání hovoru. Volající Klient 1 pošle zprávu typu INVITE, obsahující SIP URI volaného Klienta 2, na adresu proxy serveru, proxy server následně přepošle tuto zprávu na volaného Klienta 2, o čemž Klienta 1 informuje odpovědí s kódem 100 - „Trying“. Zařízení Klienta 2 začne vyzvánět a následně odešle proxy serveru zpět odpověď s kódem 180 - „Ringing“ o čemž je vzápětí informován i Klient 1. Jakmile Klient 2 zvedne hovor, je na Klienta 1 prostřednictvím proxy serveru odeslána odpověď s kódem 200 - „OK“ a po obdržení této odpovědi pošle Klient 1 na Klienta 2 zprávu typu ACK, na základě které dojde k zahájení samotného hovoru probíhajícího přes protokol RTP, viz podkapitola 4 .2. Ve chvíli kdy jeden z Klientů hovor zavěsí, odešle na protistranu zprávu typu BYE, na kterou protistrana odpoví kódem 200 - „OK“ a hovor je ukončen.



Obrázek 3: Průběh SIP hovoru

Předpokladem v příkladu výše je, že proxy server, respektive location server, zná umístění Klienta 2, pokud Klient 2 není registrován na stejném serveru a location server jeho umístění nezná, kontaktuje proxy server vzdálený server pomocí dostupných DNS (Domain Name System) na základě parametru „host“ ze SIP URI Klienta 2. Dostane-li proxy server ze vzdáleného serveru potvrzení, že Klient 2 je na něm registrován, je hovor přesměrován prostřednictvím redirect serveru na vzdálený server.

4 .2 Protokoly RTP a RTCP

Real-time Transport Protocol (RTP) je protokol určený k přenosu multimédií. Přenos probíhá prostřednictvím UDP protokolu a je řízen prostřednictvím řídícího protokolu RTP Control Pro-

tocol (RTCP). Prostřednictvím RTP jsou data přenášena zároveň s informací o jaký typ dat se jedná, jejich kódování a jelikož jsou data rozdělena, tak i pořadové číslo, které slouží k sestavení dat po přenosu. RTCP je zodpovědný za monitorování přenosu, synchronizaci datových proudů a Quality of Service (QoS), neboli snaží se dostupné zdroje rozdělit tak, aby pokud možno nedocházelo ke ztrátě datových paketů a byla co nejnižší latence a jitter.

4 .3 Kodeky

Kodek je v tomto kontextu označení pro algoritmus jehož úkolem je tzv. kódování a dekódování (nebo také komprese a dekomprese) audia a videa před a po přenosu. Tyto kodeky mohou být ztrátové nebo bezeztrátové, tedy buďto je cílem zachovat kvalitu přenášených dat nebo nižší objem dat. Pro přenos audia a videa v reálném čase se využívají druhé jmenované, jelikož spolehlivý přenos dat s pokud možno co nejmenším zpožděním je preferován i za cenu snížení kvality přenášených multimédií.

Kodeky podporovány touto aplikaci jsou:

- Audio - G.729, G711 (ulaw i alaw algoritmus), Speex, Opus, GSM, G.722, G.722.1
- Video - H.264/MPEG-4 AVC

5 XMPP protokol

Extensible Messaging and Presence Protocol, zkráceně XMPP, je protokol založený na značkovacím jazyce XML (Extensible Markup Language), který slouží především k přenosu textových zpráv v reálném čase a monitorování dostupnosti uživatelů. Protokol byl původně vyvíjen jako součást projektu pro IM komunikaci, Jabber. Komunikace funguje na bázi klient-server. Server může provozovat kdokoli, v rámci tohoto projektu byl využit open-source Openfire server.

Protokol je otevřený a kdokoli jej může rozšiřovat o další funkce. Běžná rozšíření jsou publikována pod hlavičkou XEP (XMPP Extension Protocol), mezi takové patří i „XEP-0055: Jabber Search“, který byl použit ve formě pluginu na testovacím Openfire serveru jako podpora pro vyhledávání existujících uživatelů [17].

5 .1 Adresa uživatele

Každý XMPP uživatel má svou adresu, obecný tvar takové adresy je „`uživatel@host`“ kde „uživatel“ je přihlašovací jméno, pod kterým je uživatel registrovaný na daném serveru a „host“ je doména nebo IP adresa serveru. Této adrese se také říká Jabber ID nebo JID. Na jeden JID je možné se přihlásit vícekrát a vytvořit tak více relací pro daného uživatele, zprávy však nejsou standardně doručovány na všechny relace zároveň. To, kam bude zpráva doručena, je určeno buďto na základě priority, kterou má daná relace nastavenou, nebo na základě parametru „resource“ (v adrese za lomítkem), který lze specifikovat jak při přihlášení uživatele, tak při zasílání zprávy na daného uživatele. Má aplikace definuje svůj vlastní „resource“ tak, aby zprávy z ní zaslané chodily primárně na relaci adresáta, která vznikla na základě přihlášení do této aplikace na jeho Android zařízení.

5 .2 Stanza

Při XMPP komunikaci jsou mezi klientem a serverem posílány XML zprávy, které se nazývají „stanza“. Na základě tagu v hlavičce se tyto zprávy dělí na [18]:

- Message stanza - obsahuje zprávu pro jiného uživatele a další informace jako JID adresáta, JID odesilatele a ID zprávy.
- IQ (Info/Query) stanza - slouží k získání informací ze serveru, na tuto stanzu server reaguje vlastní IQ stanzou.
- Presence stanza - slouží k odeslání zprávy o změně dostupnosti uživatele

6 Owncloud

Owncloud je open-source server, poskytující uživatelům online úložiště a možnost sdílet nahrané soubory a složky mezi sebou. Jedná se o plnohodnotné cloudové řešení, mezi pokročilejší funkce patří například správa kalendářů, správa kontaktů, je možné jej integrovat s LDAP serverem, lze vytvářet uživatelské skupiny a sdílet položky v rámci skupin nebo celého serveru. Webové rozhraní, které je součástí volně dostupné verze serveru, také implementuje například pokročilé grafické prostředí, galerii a jednoduchý textový editor. Owncloud servery je možno sdružovat a tím umožnit sdílení položek napříč servery. Prostřednictvím Owncloud serveru a oficiální Owncloud knihovny pro Android je v této aplikaci uživatelům umožněno [19]:

- Vytvářet a mazat složky,
- nahrávat, stahovat a mazat vlastní a sdílené soubory,
- sdílet soubory a složky s jinými uživateli (s právem čtení i zápisu),
- ukončit sdílení položky s jednotlivými uživateli.

6 .1 WebDAV

Má aplikace přistupuje k adresářům uloženým na Owncloud serveru přes tzv. WebDAV (Web Distributed Authoring and Versioning), což je rozšíření HTTP protokolu, které umožňuje přistupovat k serveru s cílem získat informace, vytvářet a měnit jeho obsah [20].

Zmíněná Owncloud knihovna přistupuje k datům jednotlivých uživatelů přes WebDAV rozhraní prostřednictvím URL adresy „`example.com/owncloud/remote.php/dav/files/USERNAME/`“, kde „example.com“ je doména nebo IP adresa serveru a „USERNAME“ je přihlašovací jméno uživatele k jehož adresáři se snažíme přistoupit. Samozřejmě je třeba nejprve v rámci aplikace uživatele přihlásit platným uživatelským jménem a heslem, v opačném případě server zamítne přístup.

7 Implementace aplikace

V této kapitole rozeberu klíčové komponenty Android aplikací, které byly v tomto projektu použity. Dále přiblížím konfiguraci serverové části projektu a popíšu nejdůležitější API a knihovny třetích stran, na kterých jsou vystavěny primární funkce samotné aplikace.

7 .1 Klíčové komponenty aplikace

Při vývoji Android aplikací jsou hojně využívány komponenty, které jsou součástí oficiálních Android knihoven. Patří mezi ně `Activity`, `Fragment`, `Broadcast Receiver`, `Content Provider`, `Intent` nebo `Service`. Níže popíšu komponenty, které považuji v této aplikaci za klíčové.

7 .1.1 Activity a Fragment

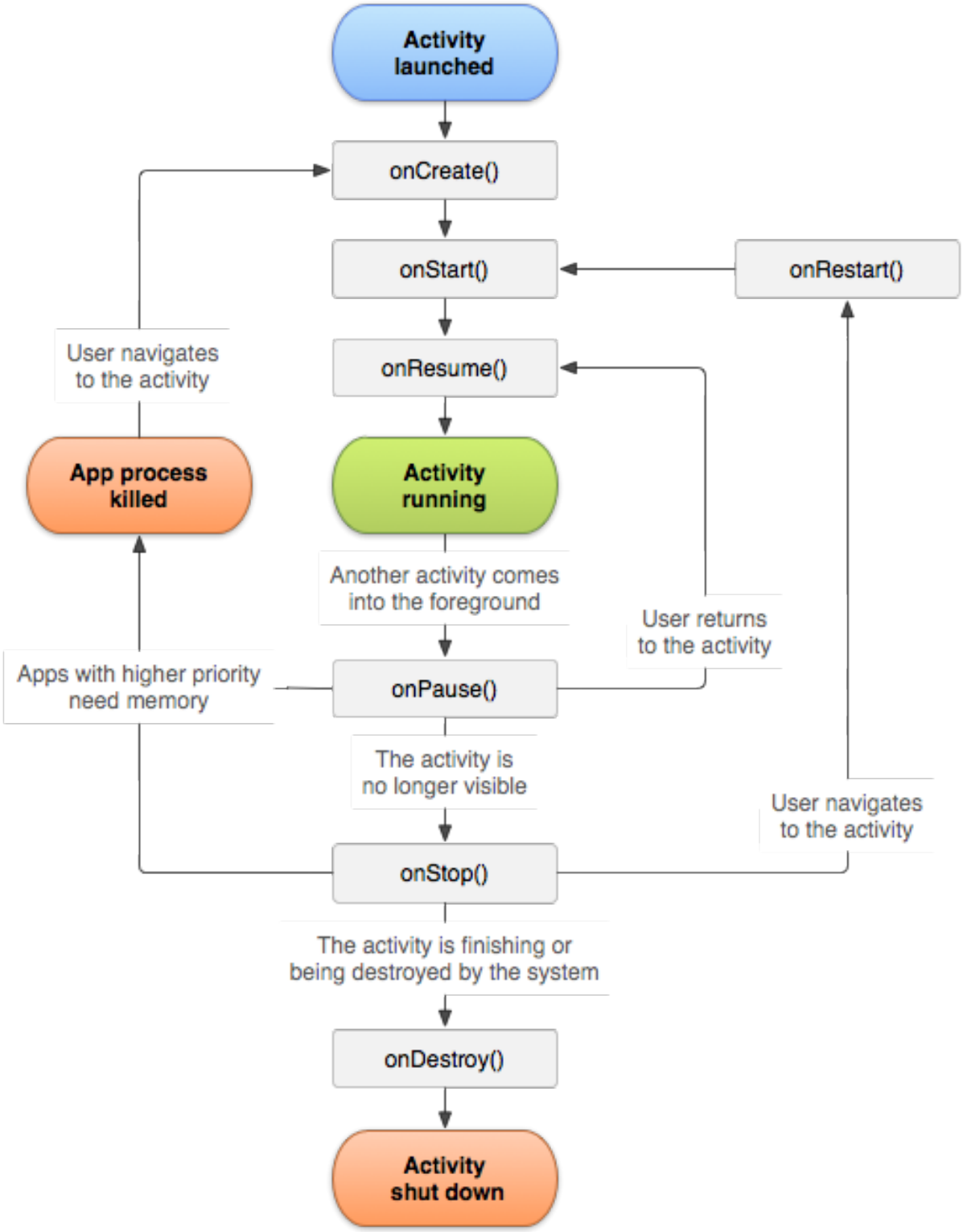
`Activity` je základní komponent aplikace, se kterým uživatel přijde do styku. `Activity` zajistí vytvoření „okna“, do kterého je možné umístit uživatelské prostředí (UI) pomocí metody `setContentView(View)`, prvky UI potom můžeme například naplnit daty z databáze a implementovat další funkce pomocí vlastních metod. Mezi další důležité metody, které jsou součástí `Activity`, patří [1]:

- `onCreate(Bundle)` - je volána při vytvoření `Activity` metodou `startActivity(Intent)`. V této metodě je typicky inicializováno UI voláním `setContentView(View)`.
- `onPause()` - je metoda, která je volána kdykoli uživatel opustí danou `Activity`. Více v podkapitole 7 .1.2.

`Fragment` je komponent velmi podobný `Activity`. Je volán z aktivity a jeho životní cyklus je na tuto aktivitu vázán. `Fragment` může reprezentovat jen část UI dané aktivity, je tedy možné, aby aktivita udržovala více fragmentů, naplněných jiným UI rozvržením, jinými daty a implementujících jiné metody. Tohoto je využito v hlavní obrazovce aplikace, která je tvořena třídou `MainActivity` (rozšířenou o třídu `Activity`) a ta implementuje `ContactListFragment` se seznamem kontaktů, `CallsFragment` s historií hovorů a `CloudFragment` s prohlížečem Owncloud úložiště.

7 .1.2 Životní cyklus Activity

Životní cyklus `Activity` je popsán na obrázku 4. Podstatné je si uvědomit, v jaké posloupnosti a kdy jsou jednotlivé metody volány. Pokud si přejeme provést akci pokaždé, když uživatel např. minimalizuje aplikaci, otevře jinou aktivitu, nebo celou aplikaci úplně ukončí, použijeme metodu `onPause()`, která je volána ve všech těchto případech, jelikož je volána i metodou `onStop()`, která je zase volána metodou `onDestroy()`. Naproti tomu metoda `onDestroy()` se nachází na konci tohoto pomyslného řetězce a je volána teprve ve chvíli, kdy dochází k úplnému ukončení aktivity, tedy uvolnění dané instance třídy `Activity` z paměti.



Obrázek 4: Životní cyklus Activity [1]

Třída `Fragment` má velmi podobný životní cyklus jako třída `Activity`. Rozšířen je především o metody `onAttach()` a `onDetach()`, které souvisí s „připnutím“ fragmentu na rodičovskou aktivitu. Ve chvíli kdy zanikne rodičovská instance třídy `Activity`, zanikají i připnuté instance třídy `Fragment`.

7 .1.3 Service

`Service` je komponent aplikace, který umožňuje udržovat v chodu nějakou operaci i v případě, že je aplikace minimalizovaná a uživatel s ní zrovna nepracuje. `Service` je spuštěna z jiného prvku aplikace, například `Activity`, metodou `startService()`. Je také možné `Service` spárovat s jiným prvkem aplikace pomocí metody `bindService()`, což tomuto prvku umožní se `Service` komunikovat nebo ji řídit [21]. Tato aplikace využívá tři instance tříd rozšířených o třídu `Service` a to jsou:

- `OpenVPNService` - třída, rozšířená o třídu `VpnService`, je inicializována z balíku ics-openvpn-core (viz podkapitola 7 .3.2) a slouží ke spuštění, ukončení, ale především udržování OpenVPN spojení.
- `SipService` - třída, inicializovaná z balíku Pjsip-android (viz podkapitola 7 .3.4), která zajišťuje registraci uživatele na serveru, načítání nativních knihoven PJSIP, ale hlavně aktivně naslouchá příchozím hovorům.
- `AppConnectionService` - mnou vytvořená třída, která primárně udržuje instanci `AppXMPPConnection` třídy, v rámci které jsou přijímány textové zprávy a také inicializuje `VoiceCallActivity` ve chvíli, kdy přijme broadcast zprávu ze `SipService` o příchozím hovoru.

Všechny zmíněné služby běží v této aplikaci na separátním vlákně (thread) a nezpomalují tak její chod.

7 .1.4 Broadcast

V Android aplikacích je možné odesílat a přijímat tzv. broadcast zprávy. Aplikace se registruje k přijímání konkrétních broadcast zpráv pomocí instance třídy `BroadcastReceiver`, jiná část aplikace (může být i úplně jiná aplikace nebo sám systém Android) může tyto zprávy odeslat pomocí metody `sendBroadcast(Intent)` [22].

Pomocí broadcast zpráv je v aplikaci řešeno především přijímání a posílání textových zpráv, kdy odchozí zprávy jsou posílány z `MessengerActivity` jako součást broadcastu do třídy `AppXMPPConnection` a odtud jsou následně odeslány na server pomocí metody `sendMessage(zpráva, adresát)`. Opačný postup platí pro přijaté textové zprávy. Dále jsou na základě přijatých broadcast zpráv řešeny operace související se SIP hovory.

7 .2 Serverová část

Po zadání přihlašovacích údajů uživatele a zvolení platného OpenVPN certifikátu se aplikace implicitně přihlašuje k testovacímu serveru v sítí VŠB-TUO. Server je postavený na linuxové distribuci Debian a jsou na něm nainstalovány a nakonfigurovány všechny služby, nezbytné ke správnému chodu jednotlivých funkcí aplikace, tedy:

- OpenVPN server,
- Openfire server,
- Asterisk PBX,
- Owncloud server.

7 .2.1 OpenVPN server

OpenVPN (viz podkapitola 3 .2) plní v rámci tohoto projektu dvě velmi důležité úlohy. Zaprvé je to vytváření VPN tunelu, skrze který má aplikace přístup ke všem ostatním službám na serveru, zajišťujícím komunikaci, a zadruhé je to samotné šifrování veškeré komunikace mezi jednotlivými klienty.

OpenVPN server byl pro účely projektu nakonfigurován k využití virtuálního rozhraní TUN, klientovi je tedy po úspěšné autentizaci přidělena IP adresa podsítě, kterou má k dispozici OpenVPN server. Veškerý síťový provoz je poté přesměrováván na server. Autentizace probíhá pomocí certifikátů viz podkapitola 3 .2.1.

7 .2.2 Openfire

Openfire je open-source server umožňující textovou komunikaci v reálném čase s využitím XMPP protokolu. Z pohledu konfigurace byla na serveru zakázána funkce odhlašování uživatelů, kteří delší dobu nekomunikovali, a bylo nastaveno ukládání všech offline zpráv tak, aby mohly být doručeny, jakmile se adresát přihlásí (platí i pro zprávy obsahující informaci o doručení zaslaných zpráv). Dále byl nainstalován plugin „XEP-0055: Jabber Search“, který je využit při vyhledávání existujících uživatelů.

7 .2.3 Asterisk

Asterisk je open-source IP PBX (Private branch exchange) telefonní systém, přes který probíhá v rámci aplikace veškerá SIP komunikace, tedy audio i video hovory. V rámci konfigurace bylo potřeba v konfiguračním souboru „sip.conf“ založit uživatele, povolit podporu videa, povolit preferované audio a video kodeky a povolit zasílání textových zpráv během hovoru, které byly využity k rozpoznání, zda je příchozí hovor s využitím videa, či pouze hlasový. Dále bylo třeba nastavit v konfiguračním souboru „extensions.conf“ tzv. dial plan, který říká Asterisk serveru

na jakou SIP adresu má, na základě volané klapky, hovor a zprávy směrovat. Každý uživatel má přidělenu klapku (extension). Ta může mít podobu čísla, ale i textu, jako je tomu v tomto případě, kdy jsou pro zjednodušení všechny klapky nastaveny na uživatelské jméno.

7 .2.4 Owncloud

Owncloud byl již popsán v kapitole 6 . Kromě uživatelských účtu nebylo třeba základní konfiguraci Owncloud serveru nijak měnit, ačkoli v ostrém provozu by bylo třeba zvýšit kapacitu úložiště, kterou mají uživatelé k dispozici (což je samozřejmě ovlivněno kapacitou samotného serveru). Bylo však třeba nainstalovat verzi Owncloud serveru 9.0 a vyšší, jelikož starší verze používali pro přístup k adresářové struktuře přes WebDAV jinou URL adresu, než se kterou počítá aktuální verze Owncloud knihovny pro android.

7 .3 Využitá API a knihovny

V aplikaci bylo implementováno několik API a knihoven třetích stran, zajišťujících klíčové funkce. Jsou to:

- Smack API,
- Ics-openvpn (balíček core),
- PJSIP (PJSUA2 API),
- Pjsip-android,
- Owncloud-android-library,
- SQLite,
- FileDirectoryPicker.

7 .3.1 Smack API

Smack je open-source XMPP knihovna, psaná v jazyce Java. Obsahuje Java třídy a metody, které aplikaci umožňují přihlášení k XMPP serveru, ustavení a ukončení spojení, přijímání a odesílání zpráv, odesílání potvrzení o přijetí zpráv, stažení offline zpráv ze serveru a ověření existence uživatele na serveru.

7 .3.2 Ics-openvpn

Ics-openvpn je pracovní název neoficiální OpenVPN aplikace známe jako OpenVPN for Android. Jedná se o Android aplikaci psanou v jazyce Java, která je postavená na oficiálním OpenVPN klientovi a je distribuovaná pod licencí GPLv2 [23]. Z celého zdrojového kódu byly použity pouze

Java třídy, jež jsou součástí balíku „core“. Tyto třídy obsahují vše nezbytné k načtení a parsování OpenVPN konfiguračního souboru a certifikátů, navázání a ukončení OpenVPN spojení, výpis OpenVPN logu, zobrazení notifikace s informacemi o připojení a obstarává načtení zkompilovaných nativních C/C++ OpenVPN knihoven (soubory libopenvpn.so a libopvpnutil.so), které již obsahují i OpenSSL knihovnu pro šifrování. Do kódu byl přidán tzv. callback, který slouží k předání informace o úspěšném přihlášení k VPN jiné části aplikace, byl změněn způsob získávání cesty k OpenVPN konfiguračnímu souboru a byl upraven `Intent` notifikace.

7 .3.3 PJSIP

PJSIP je open-source knihovna psaná v jazyce C, umožňující mimo jiné komunikaci přes protokoly SIP a RTP. V aplikaci bylo využito PJSUA2 API postavené na PJSIP, které je psáno v jazyce C++ a do jazyku Java bylo zkompilováno pomocí nástroje „SWIG“. Dále byla zkompilována knihovna OpenH264 (soubor libopenh264.so) pro kódování a dekódování videa.

7 .3.4 Pjsip-android

Tento projekt, distribuovaný pod licencí Apache v2, poskytuje souhrn Java tříd, jež umožňují vytvořit instanci `SipService` na základě PJSUA2 API a poté tuto službu ovládat přes třídu `SipServiceCommand` [24]. Tato třída obsahuje metody pro registraci uživatele na SIP serveru, vytvoření odchozího a přijetí příchozího hovoru nebo vložení snímaného a přijímaného videa do `Surface` objektu. Dále obsahuje třídy `BroadcastEventEmitter` a `BroadcastEventReceiver`, pomocí kterých je zbytek aplikace informován o stavu a aktivitách `SipService` a může na ně patřičně reagovat.

7 .3.5 Owncloud-android-library

Jedná se o oficiální Owncloud knihovnu psanou v jazyce Java, distribuovanou pod licencí MIT [25]. Jak bylo již popsáno v podkapitole 6 .1, knihovna přistupuje k adresáři uživatele, uloženém na serveru, přes rozhraní WebDAV a poskytuje aplikaci všechny třídy a metody nezbytné pro bezproblémové sdílení dat mezi uživateli.

7 .3.6 SQLite

K ukládání dat využívá aplikace SQLite databáze, přesněji „Room Persistance Library“. Room vytváří abstraktní vrstvu nad SQLite databází a umožňuje tak jednoduchou práci s daty v ní uloženými. Aplikace přistupuje k databázi přes tzv. „DAO“ (Data Access Object), což jsou třídy obsahující souhrn metod vytvořených nad SQL dotazy, pomocí kterých aplikace vytváří, čte a mění záznamy, jenž jsou reprezentovány tzv. „Entity“ objektem [26].

7 .3.7 FileDirectoryPicker

Tato knihovna, distribuovaná pod licencí MIT, poskytuje jednoduché uživatelské prostředí pro výběr složek a souborů. Cesta k těmto položkám je poté vrácena aktivitě, ze které byl modul spuštěn [27]. Knihovna je využita při výběru konfigurační OpenVPN souboru, ukládání OpenVPN logu, výběru souborů k nahrání na Owncloud nebo výběru fotky pro uložení ke kontaktům.

8 Struktura výsledné aplikace

Výslednou aplikaci tvoří importované Java třídy, které jsou součástí knihoven popsaných v podkapitole 7 .3, XML dokumenty tvořící jednotlivé UI komponenty aplikace a mé vlastní Java třídy. Mnou vytvořené Java třídy jsem rozdělil do celkem osmi balíku. V této kapitole popíšu třídy obsažené v jednotlivých balících.

8 .1 Balík activity

Do tohoto balíku byly zařazeny všechny třídy rozšiřující třídu `Activity`. Jsou to:

- `LoginActivity` - tvoří hlavní obrazovku aplikace do doby, než dojde k úspěšnému přihlášení uživatele. Tato aktivita vyzve uživatele k zadání uživatelského jména a hesla v platném formátu a k založení VPN profilu. Pro založení VPN profilu aktivita spouští `CreateVpnProfileFragment` a profily ukládá do `ProfileDatabase`. Po zadání platných údajů se spustí `VpnLogFragment` a začne přihlašování k OpenVPN. Je-li přihlášení úspěšně, `LoginActivity` je ukončena a až do odhlášení je nahrazena `MainActivity`.
- `MainActivity` - hlavní aktivita aplikace po přihlášení. Implementuje základní menu a slouží jako kontejner pro `ContactListFragment`, `CallsFragment` a `CloudFragment`.
- `MessengerActivity` - slouží k textové komunikaci. Postupně načítá z databáze a poté zobrazuje jak přijaté, tak odeslané zprávy, které již byly zpracovány, a předává si nové zprávy s třídou `AppXMPPConnection`.
- `VoiceCallActivity` - tato aktivita je spuštěna při odchozích, příchozích a probíhajících audio nebo video hovorech. Pro každý typ hovoru načítá jiný „layout“. V případě video hovorů implementuje Android třídu `Surface` pro vykreslení přenášeného obrazu.

8 .2 Balík db

Aplikace obsahuje dvě SQLite databáze a k tomu potřebné DAO rozhraní a Entity, viz podkapitola 7 .3.6. Zde je jejich přehled:

- `ProfileDatabase` - tuto databázi tvoří jedna tabulka, ke které mají přístup všichni uživatelé prostřednictvím `LoginActivity`. Ta data načítá s využitím metod deklarovaných v rozhraní `ProfileDao`. Záznamy v této databázi tvoří instance třídy `ProfileEntity`.
- `ProfileEntity` - třída definuje objekt představující jeden záznam v `ProfileDatabase`. Atributy tohoto objektu jsou název VPN profilu a cesta k OpenVPN konfiguračnímu souboru.
- `UserDatabase` - tato databáze obsahuje tabulku pro každého uživatele, ve které je uložen jeho seznam kontaktů (načítáno třídou `ContactModel`), historie volání (načítáno třídou

CallsFragment) a historie zpráv (načítáno třídami MessengerActivity a AppXMPPConnection). K datům je přistupováno prostřednictvím metod popsaných v rozhraních ContactDao, CallDao a MessageDao.

- ContactEntity - třída definuje objekt představující záznam v UserDatabase. Atributy tohoto objektu jsou JID a SIP adresa kontaktu, jeho skutečné a uživatelské jméno a cesta k fotografii.
- CallEntity - třída definuje objekt představující záznam v UserDatabase. Atributy tohoto objektu jsou typ hovoru (ve smyslu příchozí, odchozí a nepřijatý), datum a čas volání, délka volání, jméno volaného a identita přihlášeného uživatele.
- MessageEntity - třída definuje objekt představující záznam v UserDatabase. Atributy tohoto objektu jsou odesílatel, adresát, samotná zpráva, typ zprávy (ve smyslu přijatá nebo odeslaná), čas přijetí, ID zprávy přidělené Smack API a status doručení odchozí zprávy.

8 .3 Balík enums

V tomto balíku jsou umístěny tyto dva výčtové typy:

- LoginState - obsahuje možné stavy přihlášení uživatele.
- XMPPMessageType - obsahuje možné typy textových zpráv.

8 .4 Balík fragment

Zde se nachází všechny třídy rozšiřující třídu Fragment. Tyto třídy jsou:

- ContactListFragment - obsahuje Android komponent RecyclerView, do kterého načítá seznam kontaktů, jenž má klient uloženy v UserDatabase. Kontakty jsou z databáze načítány přes instanci třídy ContactModel a jsou vraceny fragmentu prostřednictvím callbacku.
- CallsFragment - načítá historii hovorů z UserDatabase a vypisuje je prostřednictvím inicializovaného RecyclerView.
- CloudFragment - využívá RecyclerView pro vypsání adresářové struktury uživatele z Owncloud serveru. Dále je zde komponent ExpandableListView, do kterého jsou vypisovány uživatelem sdílené položky.
- VpnLogFragment - je volán z LoginActivity a vypisuje stav přihlašování k OpenVPN serveru do vlastního RecyclerView. Pokud je přihlášení úspěšné vrací tuto informaci LoginActivity prostřednictvím callbacku.

Dále je zde zařazeno několik tříd rozšiřujících třídu `DialogFragment`, které jsou si svým UI velmi podobné a slouží převážně k získání vstupu od uživatele, jsou to:

- `CreateVpnProfileFragment` - dialog vyzývá uživatele k zadání názvu nového VPN profilu a zadání cesty k OpenVPN konfiguračnímu souboru.
- `AddContactFragment` - dialog umožňuje uživateli přidat do `UserDatabase` nový kontakt, přičemž volá metodu `AppXMPPConnection#userExists(uživatel)` pro ověření zda takový uživatel existuje na Openfire serveru.
- `CloudCreateNewFolder` - jednoduchý dialog, do kterého uživatel zadá název složky, která má být vytvořena v jeho adresáři na Owncloud serveru. Potřebná metoda je deklarovaná ve třídě `CloudFragment`.
- `CloudShareInfoFragment` - dialog s informacemi o sdílené položce a nabídkou ukončit sdílení. Takový požadavek je předán zpět do `CloudFragment` přes callback.
- `CloudShareWithFragment` - jednoduchý dialog, který vyzývá uživatele k zadání uživatelského jména jiného uživatele, se kterým si přeje danou položku sdílet. Potřebná metoda je opět deklarovaná ve třídě `CloudFragment`.

8 .5 Balík service

Obsahuje pouze třídu `AppConnectionService`. Kromě vytvoření a udržování instance `AppXMPPConnection` pro přijímání textových zpráv a inicializace `VoiceCallActivity` při příchozím hovoru, také využívá importovanou třídu `SipServiceCommand` k registrování uživatele na Asterisk serveru. Dále je přes tuto třídu řešeno korektní odhlášení uživatele z aplikace pomocí metody `stop()`.

8 .6 Balík utils

Zde jsem umístil všechny jinam nezařaditelné třídy, které plní spíše pomocnou funkci. Zde je jejich přehled:

- `ConnectionManager` - smyslem třídy je zajistit, že OpenVPN spojení a všechna ostatní spojení budou provázána a docílit tak toho, že pro aktivní OpenVPN spojení je nutné, aby byl do aplikace přihlášen platný uživatel.
- `ConnectionState` - obsahuje konstanty definující stav připojení a metodu `isConnected(VPNstatus)`, která upřesňuje, zda je skutečně připojeno jak OpenVPN, tak i jednotlivé služby.
- `ContactModel` - provádí operace s kontakty v rámci `UserDatabase` a vrací seznam kontaktů do `ContacListFragment` prostřednictvím callbacku.

- `ImageLoader` - obsahuje převzaté metody, které komprimují fotky uživatelů v `ContactListFragment` tak, aby bylo možné seznamem plynule listovat.
- `PagerAdapter` - třídu využívá `MainActivity` pro inicializaci zobrazovaných fragmentů.
- `SaveVpnLogFile` - třída obsahuje jedinou metodu `save(Context, cesta, vpn log)`, která uloží OpenVPN log jako textový soubor do zvoleného adresáře.

8 .7 Balík xmpp

V tomto balíku se nachází pouze třída `AppXMPPConnection`. Třída obsahuje metodu `connect()`, ve které probíhá samotné přihlášení k Openfire serveru, je zde deklarován `ReconnectionManager` pro případ změny sítě, dále `DeliveryReceiptManager` pro odeslání potvrzení o doručení zprávy zpět na odesílatele a také je zde deklarován listener pro příjem samotných zpráv. Další důležité metody jsou `sendMesssage(zpráva, adresát)` pro zasílání zpráv, `pushNotification()` pro vytváření notifikací o přijetí nové zprávy a `disconnect()` pro ukončení XMPP spojení. Ve třídě jsou také deklarovány metody pro přístup k `UserDatabase` kvůli ukládání zpráv a jsou z ní také odesílány broadcasty s informací o stavu připojení.

9 Podoba a fungování výsledné aplikace

9 .1 Přihlášení uživatele

`LoginActivity` tvoří první obrazovku, se kterou se uživatel setká. Zde si uživatel vytváří své OpenVPN profily a zadává své přihlašovací údaje. Aplikace si při zapnutí okamžitě vyžádá chybějící povolení k potřebným hardwarovým komponentům zařízení, například kameře nebo mikrofónu. Probíhá zde také jednoduchá kontrola správného zadání všech požadovaných přihlašovacích údajů. Přihlašovací obrazovka je k náhledu na obrázku č. 5.

Jakmile jsou zadány přihlašovací údaje a zvolen OpenVPN profil, `LoginActivity` zavolá metodu `startVpn()`, čímž začne na pozadí přihlašování k OpenVPN serveru prostřednictvím importované knihovny ics-openvpn-core (viz 7 .3.2), a také inicializuje `VpnLogFragment` (obrázek č. 6) do jehož `RecyclerView` je vypisován záznam o průběhu přihlašování. Pokud se přihlašování k OpenVPN nezdaří, zůstane fragment otevřen a nabídne uživateli uložit detailnější záznam do textového souboru pro pozdější debugování. V opačném případě se fragment uzavře a prostřednictvím callbacku potvrdí `LoginActivity`, že může začít s přihlašováním k serverovým službám. To začíná startem `AppConnectionService`.

Co se týče přihlašování k jednotlivým službám, primární je zde Openfire server. Pouze v případě, že přihlášení k Openfire serveru proběhlo v pořádku, je prostřednictvím `ConnectionManager` odeslán broadcast vyzývající `LoginActivity` k inicializaci `MainActivity`. S úspěšným přihlášením k Asterisk serveru a později k Owncloud serveru, se počítá. Nastane-li zde nějaký problém, nebude daná služba dostupná. Textová komunikace však bude dostupná vždy, když dojde k zobrazení `MainActivity`.

9 .2 Hlavní obrazovka

Hlavní obrazovku aplikace, po přihlášení uživatele, tvoří `ContactListFragment`, `CallsFragment` a `CloudFragment`, které jsou zakotvené v `MainActivity`. Nejdůležitější je zde záložka „Kontakty“ tvořená `ContactListFragment` (obrázek č. 7). Zde je uživateli umožněno přidat (nebo upravit již přidané) kontakty prostřednictvím dialogu `AddContactFragment` a následně započít textovou komunikaci a audio nebo video hovor pomocí příslušných tlačítek. V `MainActivity` je také nabídka umístěná v pravém horním rohu, prostřednictvím které lze uložit záznam o přihlášení OpenVPN a odhlásit uživatele ze všech služeb včetně OpenVPN.

9 .3 Textová komunikace

Textová komunikace se z pohledu uživatele odehrává v `MessengerActivity` (obrázek č. 8). Tuto aktivitu lze otevřít buďto pomocí k tomu určeného tlačítka v `ContactListFragment` nebo stisknutím notifikace o příchozí zprávě. UI této aktivity tvoří mé vlastní rozvržení a vzhledem kopíruje obvyklé IM komunikátory. Na levé straně jsou zprávy vzdáleného a na pravé straně lokálně přihlášeného uživatele, kde je navíc ještě informace o tom, zda byla zpráva úspěšně

odeslána a doručena. V pravém horním rohu je potom nabídka s možností smazat historii zpráv s daným uživatelem.

Při otevření `MessengerActivity` probíhá načítání historie zpráv z `UserDatabase` do seznamu typu `LinkedList`. Veškeré operace s Room databázemi jsou řešeny pomocí tzv. `AsyncTask`, tedy probíhají na jiném vlákně než ostatní operace. Zprávy jsou načítány po dvaceti, uživatel si může vyžádat načtení dalších zpráv „scrollováním“ nahoru, čímž se aktivuje implementovaný `SwipeRefreshLayout` prvek a aktivita zobrazí starší zprávy. V rámci `MessengerActivity` probíhá pouze načítání již uložených zpráv, všechny nové zprávy jsou ukládány ve třídě `AppXMPPConnection`.

V okamžiku kdy listener deklarovaný v `AppXMPPConnection` obdrží novou zprávu, třída vytvoří objekt typu `MessageEntity` a uloží jej do databáze. Poté jej rovnou odešle broadcastem do `MessengerActivity`. Aby bylo možné tento objekt odeslat broadcastem, musí třída `MessageEntity` implementovat rozhraní `Parcelable`. Rozhraní vyžaduje deklaraci metody `writeToParcel(Parcel, flags)`, pro převod původního objektu do `Parcel` kontejneru, a metody `createFromParcel(Parcel)`, pro opětovné vybudování původního objektu.

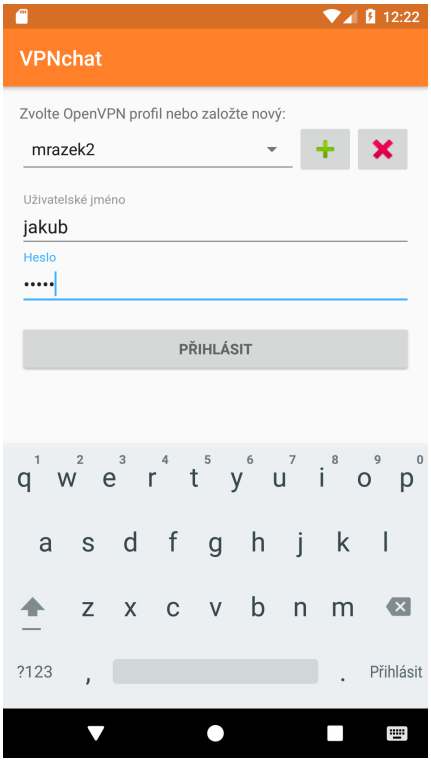
Odesílání zpráv je také řešeno přes broadcast. Ten obsahuje samotnou zprávu a JID adresáta a je určený pro `AppXMPPConnection`, kde je odeslán metodou `sendMessage(zpráva, adresát)`. Pokud se odeslání zprávy podaří, je opět broadcastem o této skutečnosti informována `MessengerActivity`, která přepíše stav zprávy na „Odesláno“. Jakmile z druhé strany dorazí potvrzení o doručení zprávy (zasláno prostřednictvím třídy `DeliveryReceiptManager` pocházející ze Smack API), situace se opakuje a `MessengerActivity` přepíše stav dané zprávy na „Doručeno“.

9 .4 Audio a video hovory

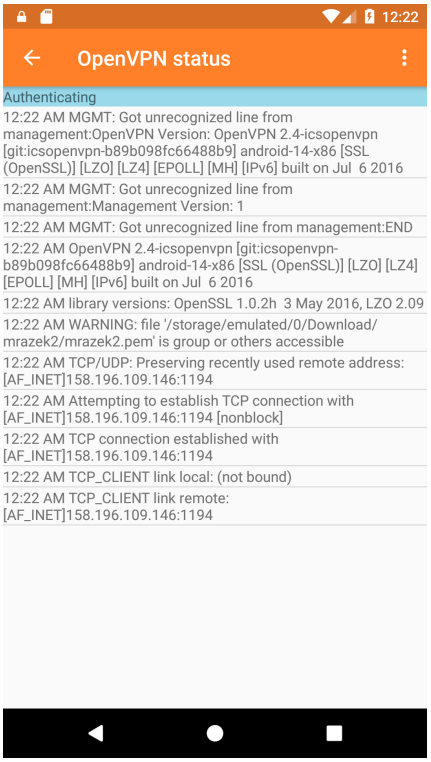
Audio i video hovory probíhají ve `VoiceCallActivity`. Tato aktivita využívá čtyři různá UI rozvržení pro odchozí hovor, příchozí hovor, probíhající hlasový hovor a probíhající videohovor. Tato UI aktivita střídá na základě stavu hovoru.

Chce-li uživatel zavolat jinému uživateli, učiní tak přes tlačítko v `ContactListFragment`. Po stisknutí tlačítka je inicializována `VoiceCallActivity` s UI rozvržením pro odchozí hovor (obrázek č. 9), které je naplněno pomocí metody `inflateOutgoingLayout()`. Na druhém telefonu zaregistruje `SipService` příchozí hovor, o čemž informuje broadcastem `AppConnectionService` a ta inicializuje `VoiceCallActivity` s UI rozvržením pro příchozí hovor, které je naplněno metodou `inflateIncomingLayout()`.

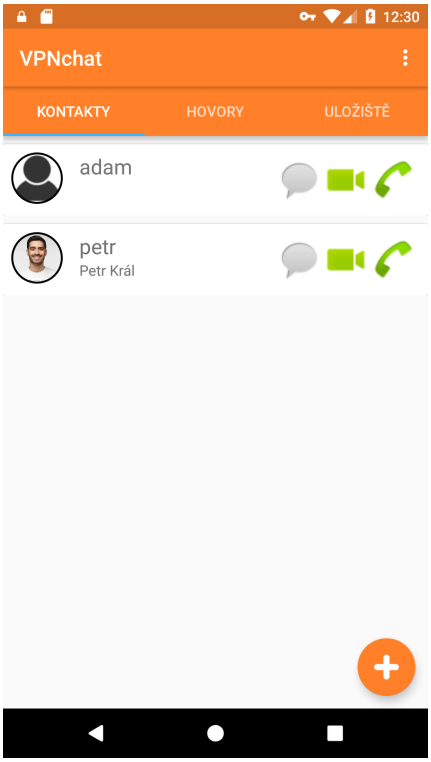
Pokud jeden z účastníků hovoru stiskne červené tlačítko, hovor je ukončen spolu s `VoiceCallActivity` na obou zařízeních. To samé nastane pokud volaný uživatel nestihne hovor zvednout do časového limitu definovaného na straně Asterisk serveru (zde je to dvacet vteřin). V případě, že volaný uživatel hovor zvedne zeleným tlačítkem, `SipService` o tom informuje `VoiceCallActivity` prostřednictvím broadcastu. `VoiceCallActivity` na základě toho opět změní své UI rozvržení, buďto na probíhající audio hovor (obrázek č. 10) metodou



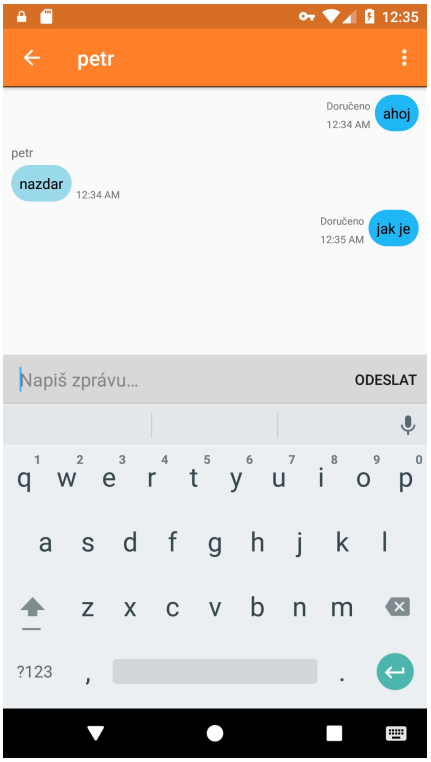
Obrázek 5: Přihlašovací obrazovka



Obrázek 6: Průběh přihlašování k OpenVPN



Obrázek 7: Seznam kontaktů



Obrázek 8: Textové zprávy

`inflateOngoingLayout()` nebo na probíhající videohovor (obrázek č. 11) metodou `inflateVideoLayout()`.

Rozvržení probíhajícího audio hovoru obsahuje stopky, jméno uživatele na druhé straně, tlačítko pro zavěšení, tlačítko pro ztlumení mikrofonu a tlačítko pro zapnutí reproduktorů. Co se týče video hovoru, je zde pouze tlačítko na ukončení hovoru a dva `Surface` komponenty, z nichž ten menší snímá obraz z přední kamery lokálního zařízení a ten větší vykresluje obraz přenášený z přední kamery vzdáleného zařízení. Metody pro přenos obrazu jsou spouštěny na separátním vláknu a je zde implementované mírné zpoždění, aby nedošlo k pádu aplikace na méně výkonných mobilních zařízeních. To zda je příchozí hovor audio nebo video zařízení pozná na základě přijetí tzv. „InstantMessage“ zprávy z volajícího zařízení, která je vždy odeslána přes SIP zároveň s odchozím hovorem a na volaném zařízení je přijímána listenerem deklarovaným ve třídě `SipAccount`, jež je součástí balíku pjsip-android, viz 7 .3.4.

Vždy když dojde k ukončení hovoru a tím i ukončení `VoiceCallActivity`, je záznam o hovoru uložen do `UserDatabase` ve formě `CallEntity`. Tento záznam obsahuje jméno uživatele, se kterým hovor proběhl, délku hovoru, datum a čas hovoru a typ hovoru. Záznam je poté okamžitě k náhledu v `CallsFragment`.

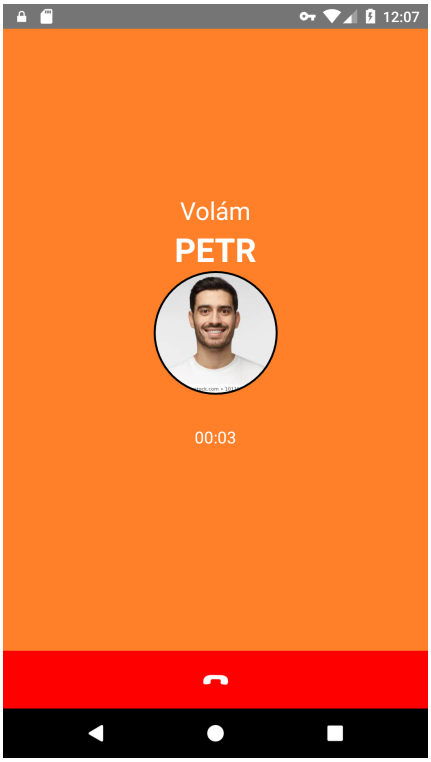
Dále je zde implementováno vyzvánění, potažmo vibrace v případě, že je zařízení ztlumené, komponent `WakeLock`, díky kterému je zařízení probuzeno při příchozím hovoru, a je využito tzv. „proximity“ senzoru pro vypnutí obrazovky při přiblížení k hlavě.

9 .5 Úložiště a sdílení souborů

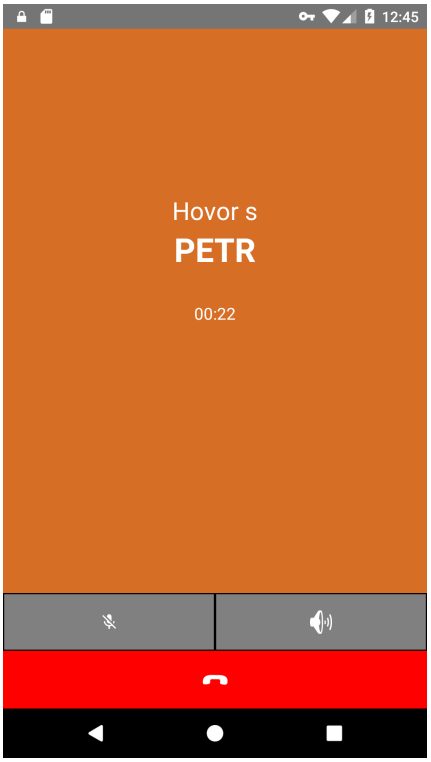
Prostřednictvím `CloudFragment` (obrázek č. 12) má uživatel přístup ke svému Owncloud adresáři. Horní část UI se skládá z indikátoru aktuálně vnořené složky, tlačítka pro návrat do předchozí složky, tlačítka na vytvoření nové složky skrze dialog `CloudCreateNewFolder`, tlačítka pro nahrání souboru vybraného prostřednictvím nástroje FileDirectoryPicker (viz 7 .3.7) a komponentu `ProgressBar` zobrazujícím průběh nahrávání nebo stahování vybrané položky.

Pod horním panelem se nachází `RecyclerView`, do kterého jsou vypisovány jednotlivé položky získané z Owncloud serveru metodou `readFolder(cesta)`. Pokud je daná položka složkou, po krátkém stisknutí se uživatel do této složky vnoří a po dlouhém stisknutí získá kontextovou nabídku umožňující složku smazat nebo sdílet. V případě souboru se rovnou zobrazí kontextová nabídka umožňující soubor stáhnout, odstranit nebo sdílet. Sdílení souborů probíhá nejprve inicializací dialogu `CloudShareWithFragment`, kam uživatel zadá jméno jiného uživatele, se kterým chce položku sdílet, toto uživatelské jméno je následně předáno (za předpokladu, že uživatel existuje) callbackem zpět na `CloudFragment`, kde je položka sdílena voláním metody `createShareLink(cesta, uživatel)`.

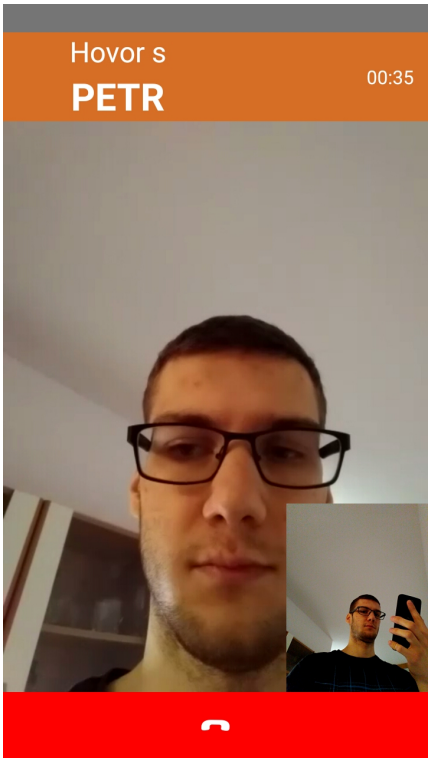
Přehled sdílených položek najde uživatel v sekci „Mé sdílené položky“, což je komponent typu `ExpandableListView`. Po kliknutí na některou z položek se zobrazí dialog `CloudShareInfoFragment`, který vypíše detail sdílené položky, včetně uživatelů, se kterými je položka sdílena, a umožní uživateli přestat tuto položku sdílet.



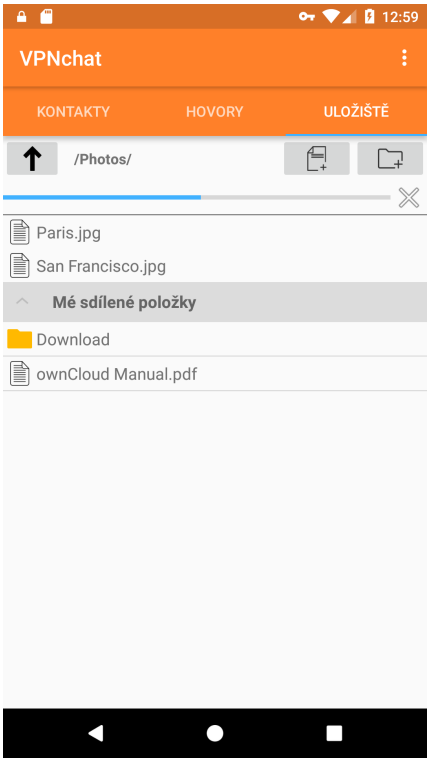
Obrázek 9: Odchozí volání



Obrázek 10: Průběh audio hovoru



Obrázek 11: Průběh video hovoru



Obrázek 12: Úložiště a sdílení souborů

10 Závěr

Cílem této bakalářské práce bylo vytvoření aplikace pro operační systém Android, která umožní šifrovanou komunikaci formou textových zpráv, audio a video hovorů a sdílení souborů mezi uživateli. Při vývoji aplikace jsem použil řadu knihoven třetích stran, které jsem popsal v podkapitole 7 .3. Základním stavebním kamenem celé aplikace bylo OpenVPN spojení. To je vystavěno na jádře projektu OpenVPN for Android. Jakmile byl VPN tunel ustaven a aplikace získala přístup k serverové části, mohla začít práce na samotné komunikaci.

Jako hlavní službu, oproti které ověřuji úspěšnost přihlášení nebo třeba existenci uživatelů, jsem zvolil XMPP spojení s Openfire serverem. Následoval přenos audio a video hovorů. Součástí systému Android je sice třída SipService, neumožňuje však přenos videa, proto byl použit SIP stack známý pod názvem PJSIP. Posledním článkem bylo zakomponování Owncloud knihovny pro Android, která poskytla vše potřebné ke splnění poslední části zadání.

Podařilo se tedy splnit všechny body zadání a výslednou aplikaci lze tudíž použít k bezpečné komunikaci napříč sítí Internet. Z hlediska budoucího vývoje aplikace bude žádoucí vytvořit systém zakládání uživatelských účtů. Takový systém by vyžadoval existenci nějakého veřejného, například webového, rozhraní na straně serveru, aby nebylo vyžadováno připojení k OpenVPN, prostřednictvím kterého by bylo možno založit nový uživatelský účet a požádat o vygenerování certifikátů. S tím by souvisela i implementace LDAP serveru a jeho napojení na ostatní serverové služby. Takto by bylo možné založit jednoduše nový účet na všech službách zároveň. Samotná aplikace by poté musela ověřovat uživatele oproti LDAP serveru a ne Openfire jako je tomu nyní. Aplikaci by dále prospělo přidání možnosti přizpůsobení UI a rozšíření výčtu funkcí, jako například přidání indikátoru psaní a přečtení do textové komunikace nebo přidání konferenčních hovorů a přesměrování hovorů do SIP komunikace.

Literatura

[1] Android Developers: Activity. [online]. [cit. 2019-04-27]. Dostupné z: https://developer.android.com/reference/android/app/Activity.

[2] StatCounter: Mobile Operating System Market Share Worldwide. [online]. [cit. 2019-04-19]. Dostupné z: http://gs.statcounter.com/os-market-share/mobile/worldwide.

[3] Statista: Number of apps available in leading app stores as of 3rd quarter 2018. [online]. [cit. 2019-04-19]. Dostupné z: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

[4] Android Source: System and kernel security. [online]. [cit. 2019-04-19]. Dostupné z: https://source.android.com/security/overview/kernel-security.html.

[5] Android Source: Legacy HALs. [online]. [cit. 2019-04-19]. Dostupné z: https://source.android.com/devices/architecture/hal.html.

[6] Android Developers: ABI Management. [online]. [cit. 2019-04-19]. Dostupné z: https://developer.android.com/ndk/guides/abis.html.

[7] KRUMNIKL, Michal. Android. [online]. [cit. 2019-04-19]. Dostupné z: http://tamz2.mrl.cz/download/TAMZ2-2018-1.pdf.

[8] Android Source: ART and Dalvik. [online]. [cit. 2019-04-19]. Dostupné z: https://source.android.com/devices/tech/dalvik/index.html.

[9] Android Developers: Platform Architecture. [online]. [cit. 2019-04-19]. Dostupné z: https://developer.android.com/guide/platform.

[10] Kotlinlang.org: Null Safety. [online]. [cit. 2019-04-19]. Dostupné z: https://kotlinlang.org/docs/reference/null-safety.html.

[11] Openvpn.net: OpenVPN cryptographic layer. [online]. [cit. 2019-04-20]. Dostupné z: https://openvpn.net/community-resources/openvpn-cryptographic-layer/.

[12] Openvpn.net: SETTING UP YOUR OWN CERTIFICATE AUTHORITY (CA) AND GENERATING CERTIFICATES AND KEYS FOR AN OPENVPN SERVER AND MULTIPLE CLIENTS. [online]. [cit. 2019-04-20]. Dostupné z: https://openvpn.net/community-resources/how-to/.

[13] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk. RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, IETF, 2008. Dostupné z: https://tools.ietf.org/html/rfc5280.

[14] Openvpn.net - wiki: BridgingAndRouting. [online]. [cit. 2019-04-20]. Dostupné z: https://community.openvpn.net/openvpn/wiki/BridgingAndRouting.

[15] Openvpn.net: FAQ regarding OpenVPN Connect Android. [online]. [cit. 2019-04-20]. Dostupné z: https://openvpn.net/vpn-server-resources/faq-regarding-openvpn-connect-android/.

[16] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler. RFC 3261 - SIP: Session Initiation Protocol, IETF, 2002. Dostupné z: https://tools.ietf.org/html/rfc3261.

[17] Xmpp.org: An Overview of XMPP. [online]. [cit. 2019-04-23]. Dostupné z: https://xmpp.org/about/technology-overview.html.

[18] Jabber Software Foundation. RFC 3920 - Extensible Messaging and Presence Protocol (XMPP): Core, IETF, 2004. Dostupné z: https://tools.ietf.org/html/rfc3920.

[19] Owncloud.org: Frequently Asked Questions. [online]. [cit. 2019-04-24]. Dostupné z: https://owncloud.org/faq/.

[20] CommerceNet. RFC 4918 - HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV), IETF, 2007. Dostupné z: https://tools.ietf.org/html/rfc4918.

[21] Android Developers: Services overview. [online]. [cit. 2019-04-27]. Dostupné z: https://developer.android.com/guide/components/services.

[22] Android Developers: Broadcasts overview. [online]. [cit. 2019-04-27]. Dostupné z: https://developer.android.com/guide/components/broadcasts.

[23] Arne Schwabe: OpenVPN for Android. [online]. [cit. 2019-04-26]. Dostupné z: https://github.com/schwabe/ics-openvpn.

[24] VoiSmart: SIP Service for Android based on PJSIP. [online]. [cit. 2019-04-26]. Dostupné z: https://github.com/VoiSmart/pjsip-android.

[25] Owncloud: The ownCloud Android Library. [online]. [cit. 2019-04-26]. Dostupné z: https://github.com/owncloud/android-library.

[26] Android Developers: Save data in a local database using Room. [online]. [cit. 2019-04-26]. Dostupné z: https://developer.android.com/training/data-storage/room.

[27] BoardiesITSolutions: Simple directory/file picker for Android. [online]. [cit. 2019-04-26]. Dostupné z: https://github.com/BoardiesITSolutions/FileDirectoryPicker.

A Přílohy v IS EDISON

Archiv vložený do IS EDISON obsahuje:

- kompletní zdrojový kód aplikace,
- dokumentaci mé části aplikace, generovanou nástrojem JavaDoc,
- ukázky z práce.