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

Diplomová práce

Pokročilé metody vzdáleného přístupu k uživatelským datům mobilního zařízení

Plzeň 2018 Roman Zeleník

Místo této strany bude zadání práce

Prohlášení

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

V Plzni dne 28. června 2018

Roman Zeleník

Poděkování

Chtěl bych poděkovat panu Ing. Ladislavu Pešičkovi za odborné vedení, pomoc a cenné rady při vypracování této práce.

Abstract

This diploma thesis deals with the transfer of user-selected existing and new events (new SMS, call, notification) from the mobile device to the computer, which displays this events to the user. First, the actual existing solutions are explored, followed by a definition of my own solution and analysis of options for Android and iOS devices. It follows a selection of appropriate technologies, the design of my own solution (including the security of communication channels and user data) and the architecture of individual parts (mobile application, computer client, server). This thesis describes also the implementation according to the design, testing and evaluation of the functionality of the whole implemented system. It was preferred emphasising to the security of the whole solution.

Abstrakt

Tato diplomová práce se zabývá přenosem uživatelem vybraných stávajících i nových událostí (nová SMS, hovor, notifikace) z mobilního zařízení do počítače, ve kterém se zobrazí uživateli. Nejdříve je proveden průzkum aktuálně nabízených existujících řešení, pak následuje vymezení vlastního řešení a analýza možností pro Android a iOS zařízení. Následuje výběr vhodných technologií, návrh vlastního řešení (včetně zabezpečení komunikačních kanálů a uživatelských dat) a architektury jednotlivých částí (mobilní aplikace, počítačový klient, server). Dále je v práci popsána implementace dle návrhu, otestování a zhodnocení funkčnosti celého realizovaného systému. Byl kladen důraz na zabezpečení celého řešení.

Obsah

1 Úvod

Tato práce řeší problematiku spárování chytrého mobilního telefonu a osobního počítače tak, aby se v počítači zobrazovaly uživateli upozornění na události nastalé v telefonu včetně jejich obsahu.

Požadovanou funkcionalitu uvedeme na několika příkladech. Např. přijde nová SMS zpráva. Uživatel vidí od koho zpráva přišla a může si ji rovnou přečíst, aniž by musel vzít telefon do ruky. Uživatel může také čekat důležitý hovor, ale okolnosti vyžadují, aby telefon nerušil zvoněním. Telefon má v kapse či uložen v tašce a nemusí si všimnout, že telefon právě vibruje kvůli příchozímu hovoru. Ale když na notebooku před sebou vždy uvidí, jaké číslo či osoba mu volá, nemusí telefon pravidelně kontrolovat. Stejně tak jej nemusí brát do ruky pokaždé, když telefon pípne. Na monitoru vidí, zda je to důležitá upomínka nebo jen doporučené video ke shlédnutí.

Toto řešení je také užitečné v situacích, kdy je uživatel mimo dosah svého telefonu. Např. telefon zapomene doma připojený k nabíječce. Z počítače si zkontroluje nepřijaté hovory a nové SMS zprávy či zprávy z chatovacích aplikací (např. WhatsApp) aby věděl, kdo se ho snažil kontaktovat a mohl mu např. emailem objasnit situaci.

Cílem této práce je tedy navrhnout a implementovat vlastní prototyp mobilní a počítačové aplikace se základní funkcionalitou s přihlédnutím k existujícím řešením a také prototyp serveru, aby telefon a počítač nemusely být ve stejné podsíti. Za základní funkcionalitu se považuje upozornit uživatele na příchozí nebo zmeškané hovory (včetně identifikace volajícího), zobrazit nepřečtené nebo nové SMS zprávy a zobrazit notifikace uživatelem vybraných mobilních aplikací.

Při návrhu bude zohledněna bezpečnost přenosu a přístupu k odesílaným osobním datům uživatele. Dále také úroveň nezávislosti na platformě a spotřeba zdrojů (baterie, množství přenášených dat) mobilního zařízení.

2 Možnosti vzdáleného přístupu k mobilnímu zařízení

Práce se zabývá především přístupem k datům typu textové zprávy, hovory, notifikace apod. Hovory se míní informace o tom, kdy a s kým hovor probíhal, tedy výpis hovorů (např. příchozí, zmeškané apod.). Notifikace (oznámení) jsou informativní zprávy mobilních aplikací určené uživateli.

Pro sdílení a přístup k textovým nebo multimediálním souborům se dnes běžně používá cloudové řešení jako Dropbox, OneDrive nebo Google drive.

2.1 Existující řešení

Na trhu jsou k dispozici různě rozsáhlá řešení. Počínaje jednoduchými aplikacemi pro zobrazení přijatých SMS zpráv v počítači až po aplikace, které umožňují s využitím mobilního zařízení telefonovat přes počítač (s připojeným mikrofonem a reproduktory k počítači). Vždy je tu ale nějaké omezení funkčnosti závislé na tom, zda si uživatel nainstaluje plnou (placenou) verzi a nebo verzi zdarma (omezenou). V následujících podkapitolách je popis několika vybraných řešení.

2.1.1 Pushbullet

Pushbullet umožňuje propojit mobilní zařízení s počítačem prostřednictvím specializovaného počítačového klienta nebo rozšířením pro webový prohlížeč. Uživatelům nabízí omezenou bezplatnou verzi a placenou verzi Pro.

Aplikace umožňuje přijímat a odesílat textové zprávy z počítače skrze mobilní telefon. Funkce je podporována pouze na zařízeních se systémem

Android. Bezplatná verze je omezena na 100 zpráv měsíčně.

Funkce Notification mirroring (zrcadlení notifikací) uživateli na počítači zobrazuje notifikace vybraných aplikací v telefonu. Pokud na ni uživatel klikne, notifikace z obrazovky počítače zmizí a v klientovi už ji uživatel nenajde. Klikne-li na Dismiss, notifikace zmizí i z telefonu. U iPhonu lze zrcadlit pouze na Mac. Zařízení s Androidem může zrcadlit na osobní počítač s Windows, Linuxem nebo na Mac (prostřednictvím webového prohlížeče Safari). V placené verzi může uživatel s Androidem používat akce dané notifikace přímo z počítače.

Pushbullet dále nabízí vlastní chat pro přátele, funkci pro odesílání odkazů z počítače do telefonu a posílání souborů mezi zařízeními uživatele. Maximální povolená velikost jednoho odesílaného souboru je 25MB v bezplatné verzi a 1GB v placené verzi. Pro Notification mirroring a SMS si uživatel může zapnout end-to-end šifrování. Obsah se tedy před odesláním v telefonu zašifruje a počítač jej po příjetí opět rozšifruje.

Další informace lze získat na stránkách projektu [1].

Vlastní testování aplikace

Aplikaci jsem si nainstaloval na svůj telefon s Androidem a v počítači jsem použil jejich nativního Windows klienta. Nejprve je nutné přihlásit se prostřednictvím Google nebo Facebook účtu. Uživatelské prostředí je sice česky, avšak texty jsou přeloženy pravděpodobně strojově. Počítačová i mobilní aplikace obsahují malé rady a tipy ohledně svých funkcí. Nastavením vlastního hesla jsem si aktivoval end-to-end šifrování.

Po spuštění a přihlášení si mobilní aplikace vyžádá potřebná práva jako přístup k SMS zprávám, notifikacím a souborům v paměti telefonu. Ve výchozím nastavení jsou označeny všechny uživatelské aplikace pro zrcadlení notifikací a je zapnuta synchronizace SMS, která umožňuje jejich procházení z počítače. Aplikace tedy po získání práva na přístup k SMS zprávám

rovnou načte a odešle veškerou konverzaci. Po vypnutí synchronizace zpráv zmizí v počítačovém klientovi záložka, která k nim poskytuje přístup. Povolit aplikaci používání pouze Wi-Fi lze nastavit jen u zrcadlení notifikací. Aplikace o svém běhu uživatele neinformuje v notifikační liště a vypnout ji lze pouze odhlášením se nebo v nastavení systému Android.

2.1.2 Dell Mobile Connect

Další řešení je od společnosti Dell. Jejich aplikace je ale dostupná pouze pro počítače Dell XPS, Inspiron a Vostro s Bluetooth zakoupenými v lednu 2018 nebo později. Podpora Dell počítačů zakoupených před lednem 2018 se zatím zvažuje. Mobilní aplikace je kompatibilní s iOS verze 10 a vyšší a Androidem verze 5 a vyšší.

Dell Mobile Connect [2] umožňuje uživateli s iOS nebo Android telefonem přijímat hovory, odesílat a přijímat textové zprávy, přistupovat ke kontaktům a k notifikacím prostřednictvím jejich Dell počítače. U telefonu s Androidem může dokonce zrcadlit obrazovku telefonu do počítače a interagovat s každou aplikací v telefonu.

Při telefonování z počítače je využíván jeho mikrofon či připojená sluchátka, takže telefon může ležet někde dál. Zároveň ale musí být v dosahu Bluetooth a Wi-Fi neboť komunikuje právě přes tyto dvě sítě.

2.1.3 Flexispy

Flexispy [3] je čistě monitorovací a špehovací nástroj. Ve své podstatě nabízí obdobné funkce jako předchozí aplikace a ještě mnoho funkcí navíc. Z webového prohlížeče umožňuje rozsáhlý přístup k zařízení. Lze si zobrazit polohu zařízení dle GPS, procházet (SMS lze i mazat) veškerou textovou komunikaci včetně MMS zpráv a nejznámějších komunikátorů jako WhatsApp, FB Messenger, Skype atd. Dále umí zaznamenávat zvuk z okolí, z

telefoních hovorů a Skype hovorů. Umožňuje procházení multimediálních souborů, webové historie, adresáře a spoustu dalších podobných funkcí. Dle jejich webových stránek jich má být celkem přes 150.

Takto lze monitorovat mobilní zařízení s Androidem, staré telefony Nokia se systémem Symbian a některé staré modely Blackberry telefonů. Pro monitorování iOS zařízení (iPhone, iPad) musí mít nainstalovaná Flexispy aplikace „root“ přístup k danému zařízení (tzv. „iOS jailbreaking“). Monitorovat lze i počítač nebo notebook. Služba je nabízena ve dvou placených verzích. Platí se pravidelně každý měsíc, čtvrtrok nebo rok.

2.2 Vlastní řešení

Cílem je vytvořit bezplatného multiplatformního pomocníka (nástroj), jež bude co nejméně závislý na použitých operačních systémech (chytrý telefon a počítač) uživatele. Uživatel by díky němu měl mít lepší přehled o tom, co se v telefonu děje, aniž by jej pokaždé musel vyndavat z kapsy a odemykat. A nebo v horším případě, pokud uživatel u sebe vůbec telefon nemá.

Základem vlastního řešení budou dvě aplikace, které si uživatel stáhne do svých zařízení. Uživatel by měl mít nad aplikacemi plnou kontrolu a také přehled o tom, ke kterým jeho datům aplikace přistupují a jak s nimi nakládají.

Pro zvýšení důvěryhodnosti je důležité se zaměřit na bezpečnostní stránku aplikací, především v jejich komunikaci. Např. end-to-end šifrování bude samozřejmostí, aby si data nemohl nikdo jiný přečíst. Budou-li aplikace využívat ke komunikaci server, nebudou se na něm žádná uživatelská data ukládat. Technicky zdatnější uživatel by si mohl server provozovat sám, aby měl jistotu, jak je s jeho daty nakládáno.

Minimální funkčnost, kterou budou aplikace poskytovat, je upozornění uživatele na důležité události v telefonu. Např. přišla nová SMS, někdo volá

nebo volal, vyskočila upomínka v kalendáři či poznámkách atd. Uživatel si v počítači všechna tato obdržená upozornění může přehledně zobrazit a nezmizí z něj, dokud to uživatel sám nebude chtít.

Aplikace bude vytvářena s ohledem na další možná budoucí rozšíření umožňující uživateli např. interagovat s upozorněními (odepsat na přijatou zprávu apod.).

3 Přístup k datům mobilního zařízení

Jedním z cílů práce je informovat uživatele především o nově přijatých zprávách a příchozích či zmeškaných hovorech - souhrnně o nových událostech v jeho telefonu. Tento typ dat je považován za klíčový, protože je nelze (a někteří uživatelé ani nechtějí) běžně synchronizovat např. s Google účtem, na rozdíl od kontaktů nebo událostí v kalendáři. Tato data jsou přijímána prostřednictvím mobilního operátora a ukládána výhradně v paměti telefonu.

3.1 Podpora jednotlivých platforem

V rámci práce byl proveden rozsáhlý průzkum aplikačních rozhraní pro vývojáře. Cílem bylo zjistit, ke kterým uživatelským datům a jakým způsobem má vývojář aplikace pro danou platformu přístup. Při tvorbě práce bylo zvažováno i multiplatformní (pro Android i iOS) řešení. Proto byl do toho průzkumu zahrnut i framework Xamarin, jenž by multiplatformní vývoj zjednodušil. Výsledek průzkumu je shrnut v tabulce 3.1. Xamarin pluginy zahrnuté v tabulce a mnoho dalších je k dispozici na GitHubu komunity [4].

		Xamarin			Android (nativní)	iOS (nativní)
		Cross platform [1]	Android	iOS		
SMS	číst	ne	ano	ne	ano	ne
	odeslat	ano	ano	ano	ano	ano
Hovory	vytočit číslo	ano	ano	ano	ano	ano
	výpis [2]	ne	ano	ne	ano	ne
Notifikace	přidat	ano	ano	ano	ano	ano
	zachytávat	ne	ano	ne	ano[3]	ne
Kontakty		ano	ano	ano	ano	ano
Kalendář		ano	ano	ano	ano	ano
Pozice		ano	ano	ano	ano	ano
Baterie		ano	ano	ano	ano	ano
Info o zařízení (druh, OS + verze)		ano	ano	ano	ano	ano
Běh / zpracování na pozadí		ne	ano	ano	ano	ano

Tabulka 3.1: Podpora přístupu k vybraným typům dat dle platforem

Získané údaje v tabulce jsou aktuální k říjnu 2017. Hodnoty *ne* v některých buňkách znamenají, že v této době (během průzkumu) nebylo poskytováno žádné oficiální API pro požadovanou funkčnost.

[1] Xamarin plugin s jednotným rozhraním pro vývojáře (vnitřní implementace pro jednotlivé platformy)

[2] Přijaté, nepřijaté a odchozí hovory

[3] OS Android verze >= 4.3

3.2 Přístup k vybraným datům

Práce bude zaměřena hlavně na nové události typu SMS zpráva a hovor a na notifikace uživatelem vybraných nainstalovaných aplikací. Zde je přesný výčet v bodech:

- nepřečtené a nově příchozí SMS
- příchozí (případně odchozí) hovory
- zmeškané hovory
- nové notifikace jiných aplikací

Reagovat na nové notifikace ostatních aplikací v Androidu umožňuje NotificationListenerService. Systém prostřednictvím této komponenty informuje aplikaci, že jiná aplikace přidala notifikaci do stavového řádku a nebo, že notifikace byla odebrána (vlastněnou aplikací nebo uživatelem) [5]. V iOS obdobná služba zatím není k dispozici. Dle zdroje [6] iOS neumožňuje vývojářům ve svých aplikacích reagovat na nové notifikace ostatních aplikací. Každá aplikace běží kvůli bezpečnosti jako sandbox a tím je jejich vzájemná komunikace či interakce vyloučena.

Procházet SMS zprávy nebo výpis hovorů může jiná než-li primární aplikace opět pouze v Androidu. Práce s daty jako zprávy, hovory nebo kontakty je v Androidu umožněna prostřednictvím rozhraní `ContentProvider`. Aplikace bude mít povolen přístup až po jeho udělení uživatelem mobilního zařízení.

4 Návrh systému

Návrh jednotlivých částí celého řešení (systému) probíhal postupně a skládal se z několika fází. Přípravou základních scénářů a případů užití systému, definováním zásadních funkčních požadavků a technického návrhu jednotlivých částí. Všechny fáze jsou jednotlivě popsány v této kapitole.

4.1 Scénáře užití

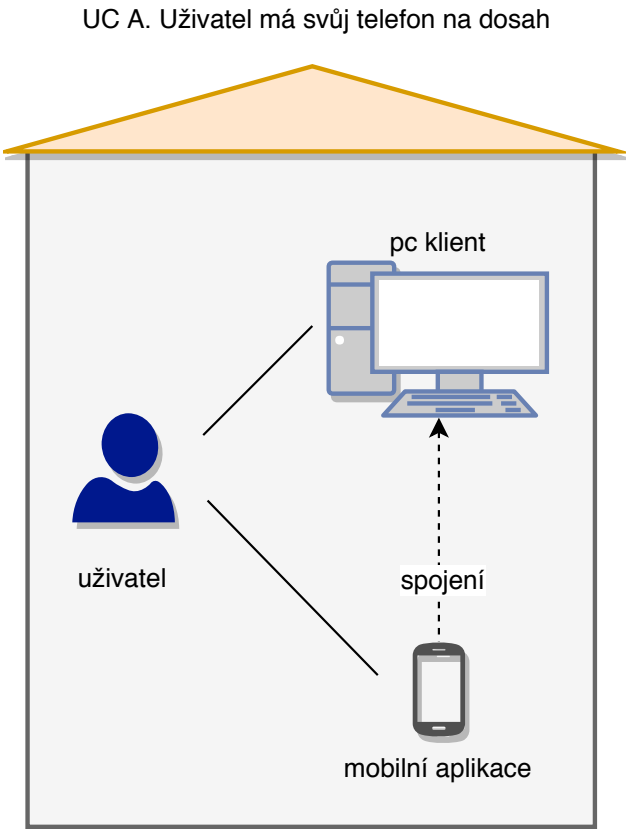
Uvažují se dva základní scénáře užití a sice:

UC A. Uživatel má telefon ve svém dosahu

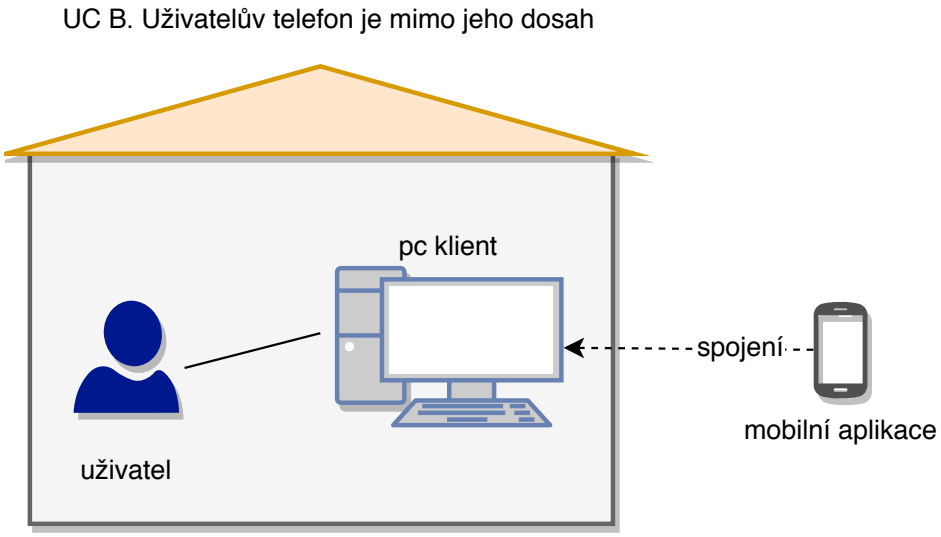
První scénář, jež znázorňuje diagram na obrázku 4.1, představuje případ, kdy uživatel pracuje na počítači a telefon má ve své blízkosti. Typicky v kapse, tašce nebo někde položený kvůli nabíjení. Při nové události se uživateli zobrazí na počítačovém klientovi oznámení o nastalé události. Může si tak zkontrolovat, jestli se musí telefonu věnovat. Nepřijde tedy o žádnou důležitou zprávu nebo očekávaný hovor. A to ani v případě, že potřebuje mít telefon v tichém režimu, aby např. nerušil ostatní.

UC B. Uživatel je mimo dosah svého telefonu

Druhý základní scénář (diagram na obrázku 4.2) je případ, kdy uživatel nemá telefon ve své blízkosti. Například po příchodu do práce zjistí, že ho zapomněl doma na nabíječce nebo v autě. Stačí se pohodlně z počítače podívat, zda ho mezitím někdo nesháněl nebo jestli se v telefonu nezobrazila např. nová upomínka, kterou si uživatel dříve do telefonu uložil.



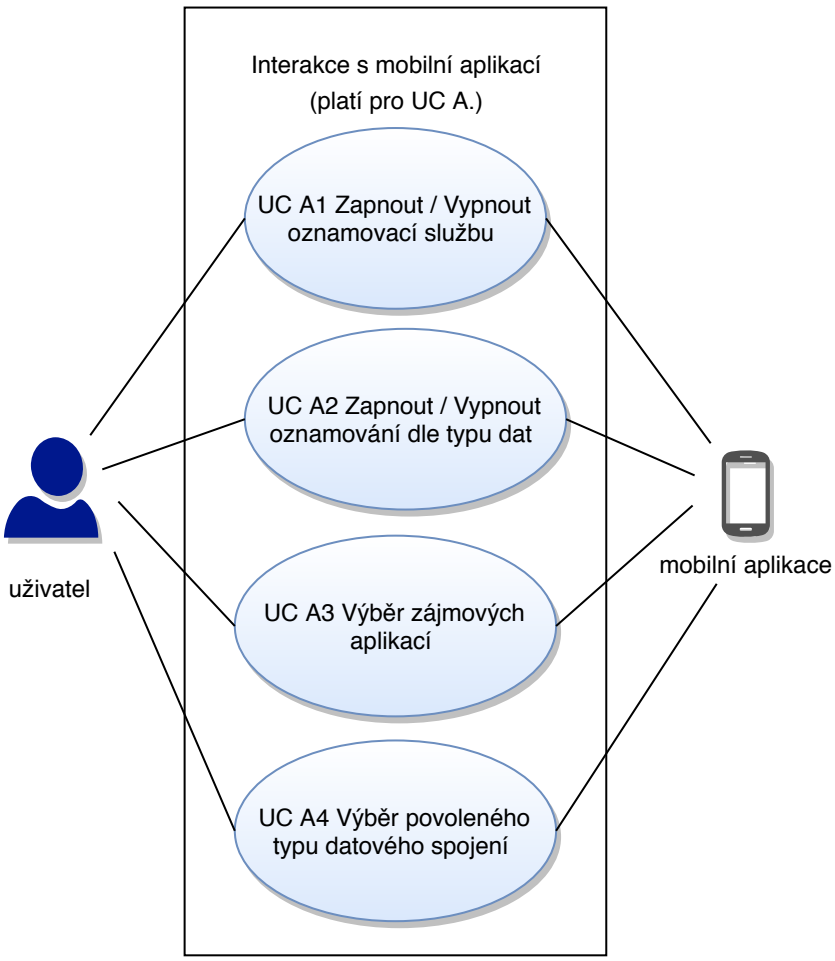
Obrázek 4.1: 1. základní scénář užití (UC A.)



Obrázek 4.2: 2. základní scénář užití (UC B.)

4.2 Případy užití

Následující diagramy (obr. 4.3 a 4.4) znázorňují základní funkce a interakce uživatele se systémem, respektive s mobilní aplikací a počítačovým klientem. Interakce s mobilní aplikací spočívá především v nastavení a přizpůsobení si chování aplikace dle uživatelových potřeb. Interakce s počítačovým klientem už je o využívání funkcí celého systému (spolupráce počítačového klienta a mobilní aplikace).



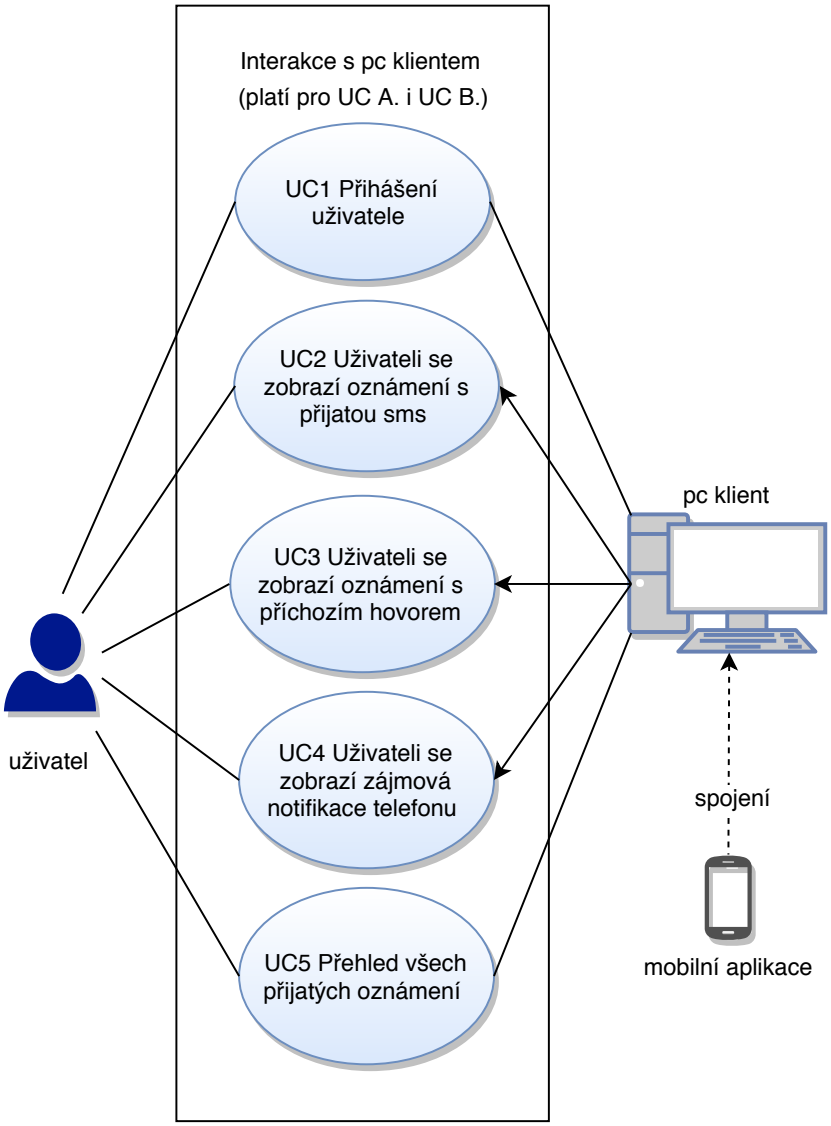
Obrázek 4.3: Základní interakce s mobilní aplikací

UC A1 Uživatel může aplikaci, resp. oznamovací službu, jednoduše zapnout nebo vypnout. Má tedy plnou kontrolu nad aplikací a nad tím, zda bude aplikace něco přeposílat či nikoliv.

UC A2 Pokud chce být uživatel upozorňován např. pouze na příchozí hovor může přeposílání ostatních typů událostí (SMS a notifikace) vypnout.

UC A3 Uživatel si pro přeposílání notifikací může vytvořit seznam tzv. zájmových aplikací. To znamená, že si uživatel zvolí (označí) aplikace, jejichž notifikace si přeje přeposílat na počítačového klienta.

UC A4 Aplikaci je možné vymezit typy datového spojení (např. mobilní data nebo Wi-Fi), které může pro přeposílání využívat.



Obrázek 4.4: Základní interakce s počítačovým klientem

UC1 Z bezpečnostních důvodů je nutné po zapnutí aplikace provést ověření uživatele.

UC2 Uživateli přijde nová SMS. Mobilní aplikace její obsah i odesílatele přepošle do počítačového klienta. Uživateli se na obrazovce počítače ukáže oznámení o nové textové zprávě.

UC3 Uživateli někdo volá. Mobilní aplikace zaznamená číslo volajícího a odešle ho opět do počítačového klienta. Pokud má uživatel číslo uloženo v kontaktech, odešle s číslem i jméno volajícího. Uživatel na monitoru v oznámení vidí jméno volajícího. V případě neznámého čísla, vidí samotné číslo.

UC4 Na mobilním telefonu se objeví nová notifikace jedné ze zájmových aplikací (viz předchozí případ UC A3). Mobilní aplikace přepošle jméno dané zájmové aplikace i s obsahem její nové notifikace.

UC5 Uživatel si v počítači může přehledně zobrazit všechny došlé události, jež aplikace od svého zapnutí přijala.

4.3 Požadavky na systém

Aby mohly být splněny oba scénáře užití (podkapitola 4.1), mobilní aplikace a počítačový klient (dále jen souhrnně zařízení) spolu nemohou komunikovat přímo, protože nemusí být vždy ve stejné (pod)síti. Je potřeba využít nějakého prostředníka - serveru. Server bude mít veřejnou IP adresu a veškerá komunikace s ním bude probíhat skrze internet. Komunikace serveru se zařízeními musí být tedy zabezpečená.

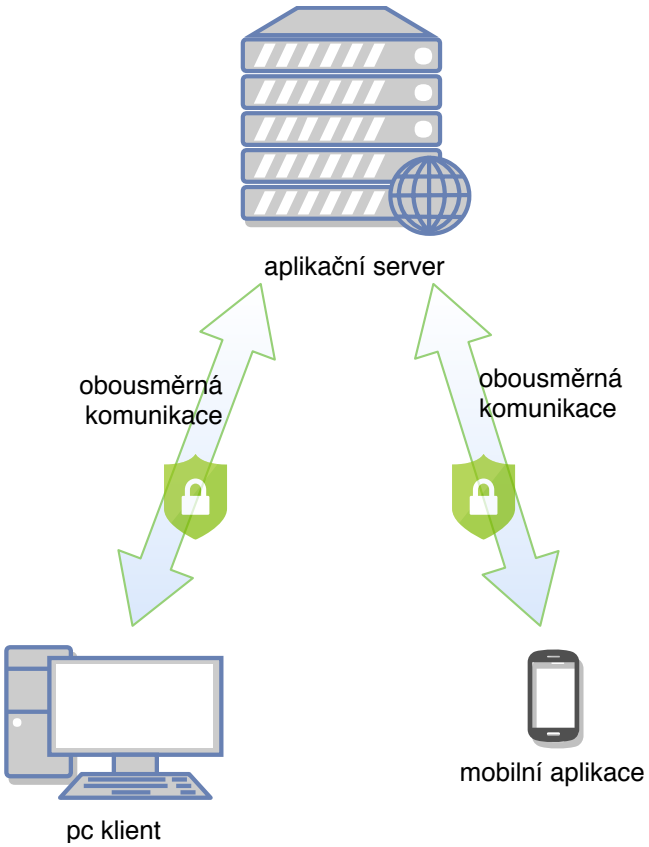
Server musí být schopen k sobě správně přiřadit zařízení, která spolu komunikují. Spojení s každým by mělo být šifrované, např. využitím SSL protokolu.

Uživatel by měl mít co největší kontrolu nad přístupem aplikace k datům v telefonu. Je kladen důraz na úsporné využívání zdrojů telefonu (paměť, čas procesoru, baterie, mobilní data), protože se předpokládá, že část aplikace poběží neustále jako služba na pozadí. Dále je kladen důraz na ochranu

proti přístupu a zneužitelnosti přeposílaných dat mezi zařízeními třetí stranou (provozovatelem spojení nebo útočníkem).

4.4 Návrh architektury systému

Podle požadavků v předchozí podkapitole (4.3) byl vytvořen základní návrh systému, který je schématicky znázorněn na obrázku 4.5.



Obrázek 4.5: Obecný návrh systému

4.4.1 Komunikace

Zařízení musí být schopna komunikovat se serverem oběma směry. Server musí být schopný kontaktovat počítačového klienta, aby mu předal zprávu o události přijatou z mobilní aplikace. Zároveň je potřeba informovat mobilní aplikaci, zda je na příjmu počítač, kterému by zprávu předal.

Komunikace může probíhat formou požadavek-odpověď bez trvalého spojení (např. REST služba) nebo navázáním a udržováním trvalého spojení využitím již existujícího protokolu (např. WebSocket) nebo implementací vlastního. Další možností je využít Firebase Cloud Messaging (dále jen FCM). Jedná se o multiplatformní řešení vyvíjené a provozované společností Google. Používání FCM je zdarma a poskytuje vývojáři možnost posílat textové zprávy ze svého serveru na klienta (Android, iOS, web, C++ nebo Unity) a z klienta na server. Odesílat zprávy ze serveru na klienta jde jednoduše použitím Admin FCM API (Node.js, Java, Python, Go). Pro přijímání zpráv od klientů je nutné v serveru implementovat XMPP protokol podle popisu na stránkách FCM. [7, 8]

Pro mobilní aplikaci je nejvhodnější první varianta (požadavek-odpověď), která je šetrná k baterii a datům telefonu. HTTP, respektive HTTPS protokol se použije pro posílání zpráv z telefonu na serveru formou POST požadavku. FCM se bude využívat jen pro přijímání zpráv ze serveru (např. připojil se počítačový klient, můžeš přeposílat nové události).

Počítačový klient potřebuje zprávy především přijímat. První možností je periodicky posílat dotazy na server, zda pro něj nemá novou zprávu. Tímto by však byl server zbytečně zaměstnáván. Vhodnější bude jen čekat, dokud sám server nepošle novou zprávu. V tomto případě však nelze opět použít FCM neboť Google nenabízí FirebaseMessaging API pro příjem zpráv na jiném než mobilním či webovém klientovi [7]. Ke komunikaci se tedy použijí WebSockety [9]. Není potřeba implementovat vlastní komunikační protokol skrze Sockety operačního systému.

4.4.2 Mobilní aplikace

V rámci práce bude mobilní aplikace vyvíjena pouze pro zařízení se systémem Android, protože poskytuje vývojáři veškerou potřebnou funkcionalitu a rozhraní pro přístup k vybraným datům (viz podkapitola 3.2). Systém iOS bohužel aktuálně neumožňuje k vybraným datům přístup (viz tabulka 3.1). Mobilní aplikace tedy nebude multiplatformní a v rámci této práce bude vyvíjena pouze pro Android. Základní podporovanou funkcionalitou bude tedy informování uživatele o nových událostech a o nové notifikaci přidané některou z uživatelem zvolených (zájmových) aplikací.

Aplikace bude vyvíjena nativně v jazyce Java prostřednictvím Android SDK a vývojového prostředí AndroidStudio. Minimální požadovanou verzí operačního systému bude Android Jelly Bean (4.3), protože je to minimální požadovaná verze pro použití NotificationListenerService (více v odstavci Přístup k notifikacím). Takto bude aplikace dostupná pro 95,6% (aktuální k 8.5.2018) aktivních zařízení [10].

Pokud je aplikace (oznamovací služba) aktivní, tzn. že reaguje na nové události, bude o své aktivitě informovat uživatele prostřednictvím persistentní notifikace.

Ovládání

Aplikaci bude možné otevřít klasicky přes spouštěcí ikonu aplikace a také klepnutím na její persistentní notifikaci (bude-li její služba aktivní). Zapínání a vypínání služby bude možné v hlavní Activity aplikace.

Activity je stěžejní Android komponenta. Je to vstupní bod aplikace, která se spouští klasicky klepnutím uživatele na její spouštěcí ikonu. Activity představuje okno - obrazovku, kterou uživatel po spuštění vidí a obsahuje prvky grafického uživatelského rozhraní pro interakci s aplikací. Android aplikace většinou obsahují několik obrazovek, což znamená, že jsou složeny z několika

Activity komponent. Vždy musí být jedna označena jako hlavní (spouštěcí) [11, 12].

Kromě zapínání a vypínání celé služby bude hlavní Activity umožňovat uživateli si vybrat, na jaké typy událostí (SMS, hovor či zájmová aplikace) chce být upozorňován. Další (specifičtější) nastavení budou rozvržena na dalších obrazovkách. Pro navigaci mezi obrazovkami bude použit Navigation Drawer, což je postranní navigační panel, který je skrytý, když není používán [13]. Jde o standardní designové řešení dnešních Android aplikací.

Jednotlivé obrazovky aplikace je možné implementovat i prostřednictvím doporučených (z hledika architektury aplikace) Fragment komponent namísto celých aktivit. Fragment komponenty nesou obsah, který se prezentuje uživateli. Aplikace potom může obsahovat pouze jednu Activity, která pouze spravuje fragmenty a určuje, který bude aktivní - tedy který z nich má svůj obsah aktuálně zobrazit uživateli [11, 14].

Přístup k notifikacím

V Androidu 4.3 byla přidána služba NotificationListenerService jež je jednou z klíčových služeb pro tuto práci. Umožňuje aplikaci být informována systémem o přidání či odebrání oznámení (notifikace) ze stavového řádku telefonu jinou aplikací [5].

Přístup k textovým zprávám

Aplikace bude reagovat na nové SMS prostřednictvím systémové komponenty BroadcastReceiver. BroadcastReceiver umožňuje aplikaci přijímat zprávy od systému nebo dalších aplikací. Jedná se o typ komunikace dle návrhové vzoru publish-subscribe. BroadcastReceiver se registruje společně s typem či typy zpráv, které chce přijímat. Obsah a typ zprávy je předáván v Intent objektu [11, 15].

Aplikace si musí BroadcastReceiver v systému zaregistrovat pro akci (Intent)

android.provider.Telephony.SMS_RECEIVED

a zažádat o

android.permission.RECEIVE_SMS

oprávnění [16]. Pro přístup k obsahu SMS je ještě nutné zažádat o

android.permission.READ_SMS

oprávnění.

Přístup k hovorům

Pro upozornění na hovory se opět použije BroadcastReceiver. Bude registrován pro dva typy zpráv (dvě akce). Reagovat na příchozí či zmeškaný hovor umožní

android.intent.action.PHONE_STATE [17].

Dále díky

android.intent.action.NEW_OUTGOING_CALL

může aplikace reagovat také na odchozí hovor [18]. Ještě je nutné zažádat o oprávnění

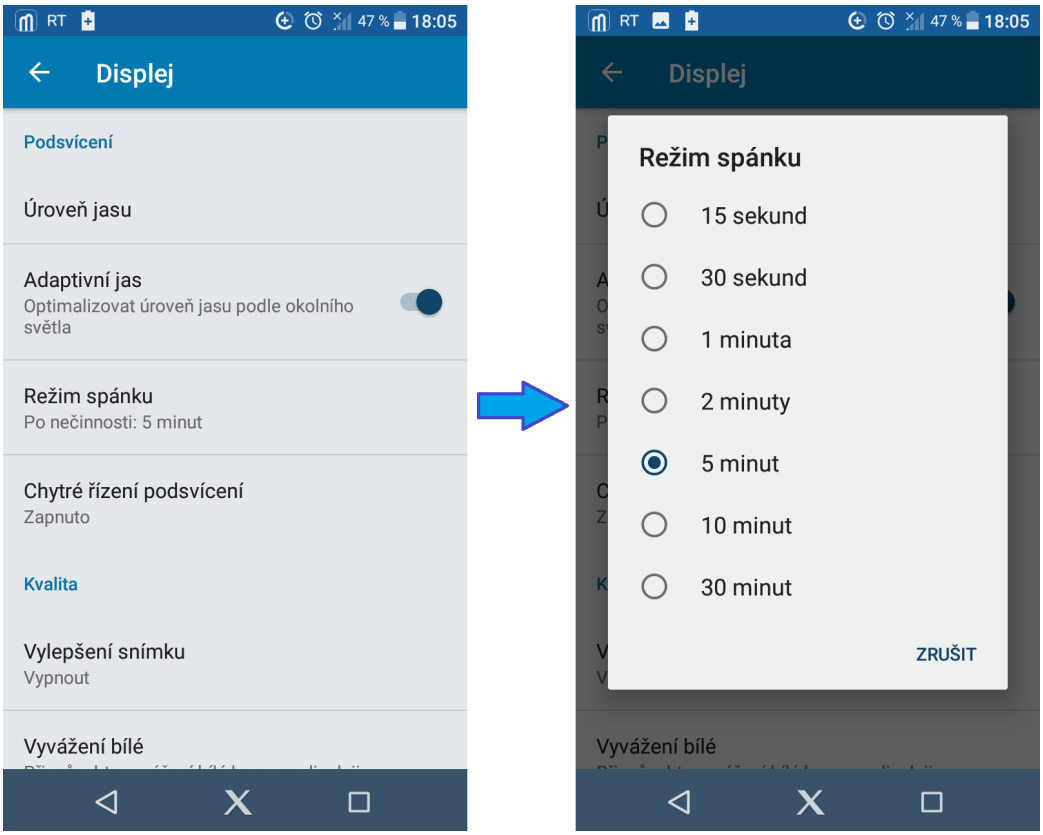
android.permission.READ_PHONE_STATE

a

android.permission.PROCESS_OUTGOING_CALLS.

Ukládání předvoleb a nastavení

Android aplikace často obsahují nastavení, které umožňuje uživateli přizpůsobit si aplikaci v rámci nabízených možností. K tomu se standardně používá Preference API. Obsahuje komponenty a metody pro vytvoření jednotně vypadající a interagující obrazovky nastavení aplikace (ukázka na obr. 4.6). Pro vytvoření obrazovky je nutné použít speciální PreferenceActivity nebo PreferenceFragment. Struktura se definuje v XML souboru použitím komponent, které jsou rozšířením základní třídy Preference. Každá komponenta představuje jednu položku v nastavení (viz obr. 4.6). [11, 19]



Obrázek 4.6: Snímek ze systémového nastavení displeje. Klepnutím na položku se zobrazí rozhraní pro změnu hodnoty.

Pro ukládání a čtení hodnot nastavení slouží rozhraní SharedPreferences. To umožnuje ukládat a číst dvojice klíč-hodnota pro primitivní datové typy (boolean, int, long, float, String). Klíčem je vždy String. Jde o „app-private data“. To znamená, že data nejsou běžně přístupná ostatním aplikacím a při odinstalování aplikace se ze zařízení smažou. Nejsou přímo přístupná ani uživateli, pokud nemá „root“ přístup. [20]

4.4.3 Server

Na základě komunikačních požadavků (podkapitola 4.4.1) a implementačních zkušeností je pro vývoj serveru zvažováno použití Java EE webového serveru nebo Node.js serveru. Zároveň tím zde bude splněna multiplatformnost v tom smyslu, že není vyžadován specifický operační systém pod kterým server poběží.

Cílem je, aby server ideálně fungoval pouze jako zprostředkovatel komunikace mezi mobilním a počítačovým klientem. Pro svůj běh by měl potřebovat co nejméně informací o uživateli. K jeho osobním (přeposílaným) datům by neměl být potřeba přístup.

Část serveru bude implementována stylem RESTové služby. Ta bude přijímat různé typy zpráv na daných URL adresách z mobilních zařízení a předávat je patřičným počítačovým klientům. S počítačovými klienty bude komunikovat prostřednictvím WebSocketů [9]. Tento protokol by měl být plně dostačující a není tedy potřeba realizovat vlastní komunikační prostředek.

Pro předávání zpráv je nutné k sobě správně přiřadit zařízení uživatele. Jedním ze standardních řešení je, že se uživatel na obou zařízeních přihlásí ke svému účtu na serveru. Vytvořila by se databáze uživatelů a jejich registrovaných zařízení, kde by každé mělo svůj unikátní identifikátor. Uživatele je také možné ověřit skrze Goggle nebo Facebook účet, který u jedné ze společností používá. Cílem je najít vždy pro přijatou zprávu odpovídající navázané spojení s klientem. To však lze vyřešit i bez zakládání uživatelských účtů.

Každá dvojice (mobil-počítač) bude mít svůj „párovací token“, který přiloží k odeslané zprávě. Podle něj server pozná, komu přijatou zprávu přeposlat.

Server si tedy bude ukládat pouze dvojice identifikátorů - *párovací token* a *ID otevřeného soketu*. Proto bude potřeba ověřit, zda na vybrané platformě (Java nebo Node.js) lze identifikovat otevřený soket a poslat jím zprávu na základě přijatého HTTP požadavku. Odezva a výkon server se tedy mimo jiné bude odvíjet od počtu otevřených WebSocket spojení a zároveň počtu zpracovaných (obsloužených) zpráv (požadavků).

Java server

Pro implementování Java REST serveru lze použít některý ze známých frameworků (např. Spring nebo Jersey) nebo implementaci, která je součástí Java EE. Stejně tak WebSockety. Spolupráce těchto dvou služeb je možná, ale ne snadná. Inspirovat se lze ve webovém článku Lucase Jellema [21]. Služby lze propojit, tak že se na serveru vytvoří WebSocket klient (`@ClientEndpoint`), kterým se přijaté HTTP požadavky pošlou WebSocket serveru (`@ServerEndpoint`). Ten dále rozhodne, kterému počítačovému WebSocket klientovi zprávu pošle. To by mohla být potenciálně kritická sekce z hlediska výkonu, protože by veškeré zprávy z mobilních zařízení musely procházet jedním WebSocketem. Pokud by se jich na spolupráci služeb použilo více, snižuje se tím potenciální maximální počet současně připojených uživatelů.

Node.js server

V Node.js [22, 23] lze implementovat HTTP server se základním interním balíčkem a nebo jedním z mnoha vytvořených komunitou. Jedním z nejznámějších je minimalistický webový framework Express [22, 24]. Dalším často používaným balíčkem je modul Socket.IO, který umožňuje obousměrnou komunikaci v reálném čase [22]. Pro navázané spojení využívá i WebSockety,

jsou-li k dispozici. Tyto balíčky se běžně používají společně, protože je lze snadno použít v rámci jednoho serveru (stačí jeden společný port pro naslouchání). Předávání (HTTP) zpráv z Express některému ze Socket.IO klientů lze přímo na rozdíl od Java serveru (viz předchozí odstavec). Na druhé straně je nutný Socket.IO klient. Ten je k dispozici nejen pro webový prohlížeč (JavaScript), ale také jako knihovna pro jazyky Java, C++, Swift a Dart [25]. Tato varianta by byla implementačně jednoduší. Node.js server by také byl méně náročný na výpočetní zdroje než Java server, jehož běh je nutný v některém z kontejnerů (např. Tomcat, GlassFish, JBoss). Z těchto důvodů bude server vyvíjen na platformě Node.js.

4.4.4 Počítačový klient

Počítačový klient (dále jen klient) může být webový (např. single-page aplikace) nebo desktopový. Jedním z cílů práce je multiplatformní řešení. Tento požadavek by splnil webový klient i desktopový klient vyvíjený v jazyce Java nebo C++ (s využitím multiplatformního UI frameworku Qt). Dalším klíčovým požadavkem je upozornit uživatele, ideálně zobrazením notifikace na ploše, aniž by musel být klient aktivní (otevřené okno aplikace). Tedy mohl být např. minimalizován nebo překryt jinou používanou aplikací.

Webový klient

Na základě zvolené varianty serveru by webový klient nebyl Java webová aplikace, ale single-page aplikace ve skriptovacím jazyce JavaScript.

Pro JavaScript existuje Notification API, které umožňuje webovým stránkám informovat uživatele zobrazením notifikačního okna mimo okno prohlížeče. Podpora tohoto API v internetových prohlížečích nebyla v době provádění analýzy v rámci práce 100%. Je to spíše dominanta Google Chrome prohlížeče [26].

Implementace webového klienta by vyžadovala větší zainteresování serveru do celkového řešení, což je částečně v protikladu s cílem používat server pouze jako prostředníka komunikace. Výhodou oproti desktopové verzi je, že by uživatel vždy používal aktuálního klienta.

Desktop klient

Na základě implementačních zkušeností a znalostí bude počítačový klient vyvíjen v jazyce Java. To umožní sdílení fragmentů kódu s mobilní aplikací (především datový model) a nebude nutné kompilovat klienta extra pro každou platformu jako v případě C++ aplikace.

Uživatelské prostředí bude vyvíjeno na JavaFX platformě [27], jejíž UI komponenty lze rozšířit dalšími knihovnami. Např. ControlsFX umožňuje mimo jiné snadné zobrazení notifikačních oken na ploše.

Uchování přijatých dat, ukládání nastavení či citlivých dat, jako např. šifrovacího klíče (více v podkapitole 5.2), zde bude více pod kontrolou samotné aplikace, než-li v paměti prohlížeče webového klienta. Uživatelské preference lze ukládat do šifrovaného souboru nebo využít Java Preferences.

V rámci této práce bude tedy vyvíjen desktop Java klient, který bude distribuován jako přenositelný spustitelný `jar` soubor. Hodnoty nastavení aplikace se budou ukládat šifrovaně prostřednictvím Java Preferences, takže aplikace nemusí vytvářet další pomocné soubory. O webovém klientu lze uvažovat do budoucna jako o možném rozšíření.

5 Bezpečnost systému

Bezpečnost je zde velice důležitá, protože se pracuje s osobními a citlivými daty (např. SMS z banky) uživatele. Je řešena na několika úrovních napříč celým systémem.

5.1 Komunikační vrstva

Základem je bezpečný komunikační kanál. Zprávy z mobilní aplikace na server budou posílány přes HTTPS protokol. Stejně tak komunikace serveru a počítačového klienta (v obou směrech). Socket.IO totiž umožňuje navázat spojení přes HTTP i HTTPS protokol. Z tohoto důvodu bude nutné vygenerovat po nasazení serveru SSL certifikát pro jeho veřejnou adresu.

5.2 Datová vrstva

Systém se nebude spoléhat pouze na šifrované spojení. Pro zvýšení bezpečnosti a důvěryhodnosti uživatelů bude obsah zpráv šifrován tzv. end-to-end šifrováním. To znamená, že obsah zprávy bude zašifrován klíčem, který budou znát pouze uživatelova zařízení (mobil a počítač). Toto opatření bude sloužit např. jako obrana proti útoku typu „Man in the middle“ nebo obrana proti pokusům číst přeposílaná data na serveru.

Šifrovací klíče tedy nesmí být přidělovány serverem. Klíč bude generován přímo na jednom ze zařízení. Nejbezpečnější a nejjednodušší způsob přenosu klíče bude opsat ho před prvním použitím z jednoho zařízení do druhého. Klíč tak bude znát pouze vlastník obou zařízení, což ztíží zneužití počítačového klienta pro odposlouchávání cizího telefonu. Pokud bude klíč vyzrazen, stačí vygenerovat na zařízení nový.

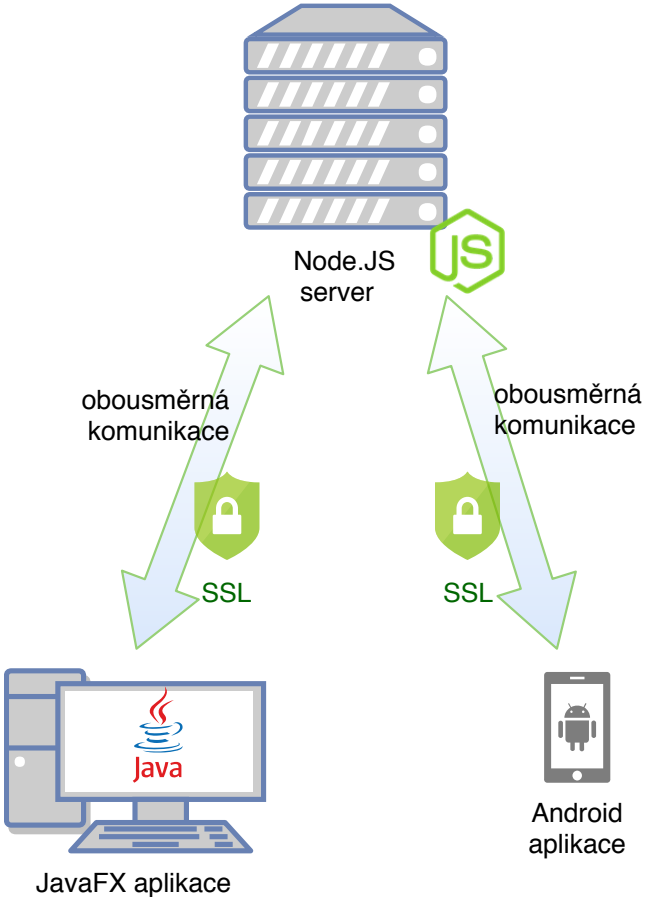
Dále bude třeba zajistit bezpečné uložení klíče na obou zařízeních. V Androidu lze využít SharedPreferences (více v předešlé podkapitole 4.4.2, odstavec Ukládání předvoleb a nastavení). Počítačový Java klient ho bude uchovávat šifrovaně v Java Preferences společně s ostatními preferencemi aplikace. Dalšími možnostmi jsou použití KeyStore nebo zašifrování do souboru na disku.

5.3 Aplikační vrstva

Přístup k počítačovému klientu bude chráněn přihlášením. Klienta tak může spustit pouze vlastník, což by mělo také zabránit útočníkovi v opsání citlivých údajů (jako šifrovacího klíče) pro odposlouchávání vlastníkova telefonu. Veškeré tyto citlivé údaje musí být v počítači uloženy v šifrované podobě, jinak by šlo ochranu počítačového klienta jednoduše obejít.

6 Architektura systému

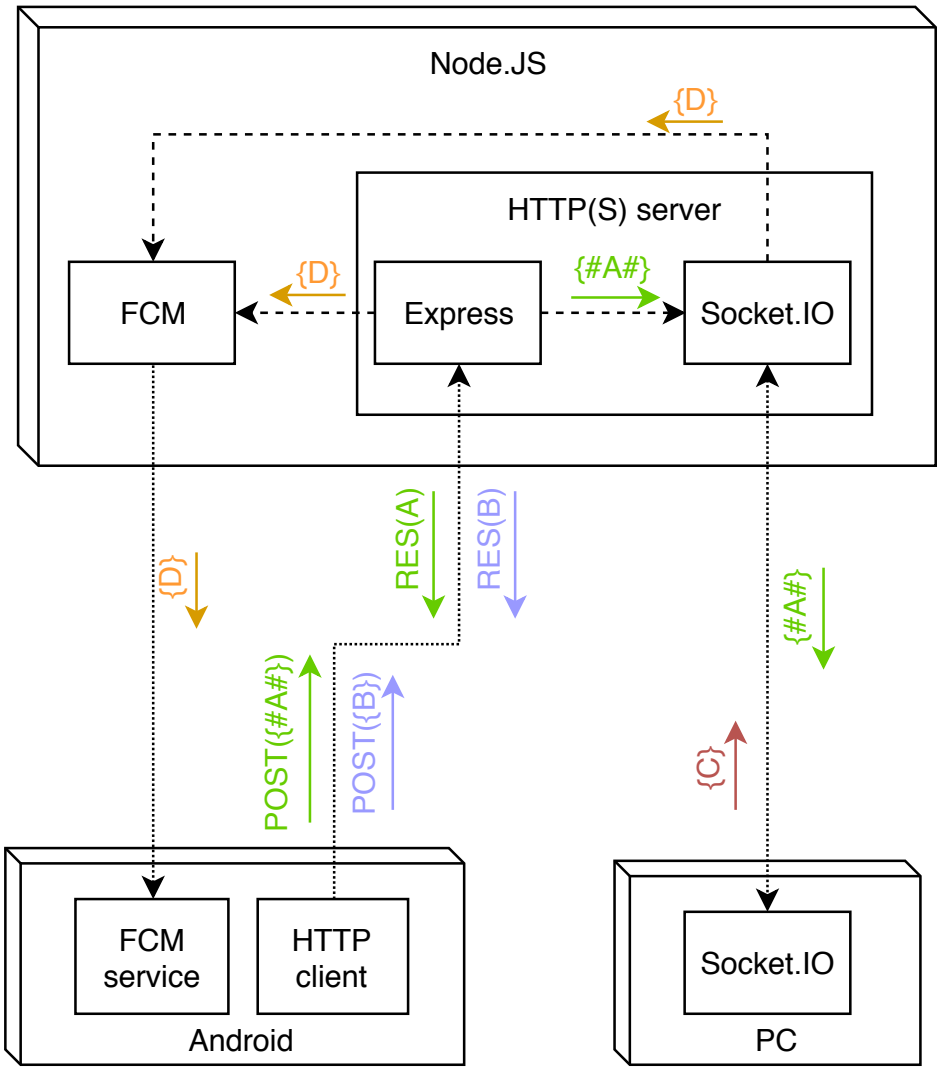
Implementace navrženého systému je rozdělena a popsána v následujících podkapitolách. Systém se skládá ze tří hlavních částí, jež jsou schématicky znázorněny na následujícím obrázku (6.1). Systém byl implementován dle výše uvedených případů užití, požadavků a návrhu (kapitoly 4 a 5).



Obrázek 6.1: Architektura celého systému (hlavní části)

6.1 Komunikace

Jednotlivá zařízení spolu komunikují prostřednictvím textových zpráv ve formátu JSON. Následující schéma (obr. 6.2) zobrazuje tok (ve směru šipky) zpráv rozlišených podle zařízení, které je odeslalo. U každého zařízení je také uvedena použitá technologie, modul či protokol transportní vrstvy.



Obrázek 6.2: Komunikace - tok zpráv systémem (včetně použitých modulů)

Typy zpráv znázorněných ve schématu 6.2 :

POST(#A#[1]) HTTP POST požadavek odeslaný z mobilní aplikace na server. Obsahem HTTP požadavku je JSON objekt, který obsahuje zašifrovanou zprávu *A*. Je-li k serveru připojen počítačový klient, jemuž je zpráva adresována, přepošle přijatý JSON skrze otevřené Socket.IO spojení klientovi. Klient přijatou zprávu rozšifruje a zpracuje obsah. Zprávy se na serveru neukládají. A to ani tehdy, když adresát není zrovna k serveru připojen. V takovém případě se zpráva zahodí a mobilní aplikace obdrží odpověď s chybou (více v podkapitole 6.1.2). Obsahem JSON objektu jsou uživatelské zprávy, např. nová SMS.

RES(A) HTTP odpověď na přijatý POST požadavek se zprávou *A*.

POST({B}[2]) HTTP POST požadavek odeslaný z mobilní aplikace na server. Obsahem je JSON objekt se zprávou (nešifrovanou) *B*. Obsahem může být příkaz či odpověď na příkaz serveru.

RES(B) HTTP odpověď na přijatý POST požadavek se zprávou *B*.

{C} JSON objekt se zprávou *C* odeslaný skrze otevřený Socket.IO z počítačového klienta na server. Obsahem je příkaz serveru.

{D} Datová zpráva *D* ve formátu JSON poslaná ze serveru mobilní aplikaci skrze Firebase Cloud Messaging. Obsahem je příkaz. Může být i informační, např. připojil se počítačový klient.

6.1.1 Mobil - Počítač

Mobilní aplikace a počítačový klient spolu nekomunikují přímo, ale pouze skrze prostředníka - server. Mobilní aplikace je vždy informována o změně stavu počítačového klienta, tedy zda je dostupný (online) či nikoliv. Díky

[1] #...# značí zašifrovaná data
[2] {...} značí JSON formát

tomu mobilní aplikace přeposílá nové události pouze tehdy, když na ně někdo čeká, čímž šetří zdroje telefonu (data, baterii). Nastane-li nová událost když není počítačový klient dostupný, žádná zpráva se neposílá. Po obdržení informace, že je vzdálený (počítačový) klient dostupný, přepošle mobilní aplikace veškeré nepřečtené SMS, zmeškané hovory a aktuálně zobrazené notifikace. Samozřejmě v závislosti na nastavení uživatele.

Dílčí komunikace, tedy komunikace mobilní aplikace se serverem a komunikace počítačového klienta se serverem, jsou popsány v následujících podkapitolách.

6.1.2 Mobil - Server

Pro komunikaci ve směru mobilní aplikace → server se používají HTTP POST požadavky (viz schéma 6.2). Podle typu zprávy se určí sub-doména, na kterou se požadavek pošle. Formát zpráv je popsán v podkapitole 6.1.4. Server přijímá POST požadavky na sub-doménách:

/event Zde se zpracovávají uživatelské zprávy s událostmi (zprávy typu *A*). Server odpoví jedním z následujících kódů :

- **200** požadavek přijat, zpráva přeposlána
- **400** neplatný formát JSON zprávy (např. chybí povinný atribut)
- **440** požadavek odmítnut, protože není připojen adresát (počítačový klient) zprávy

/firebase/registration Zde se přijímají registrační zprávy, resp. příkazy s FCM ID mobilní aplikace (zpráva typu *B*). Jejich účel je popsán níže. Server odpoví jedním z následujících kódů :

- **200** požadavek přijat a zpracován
- **400** neplatný formát JSON zprávy (např. chybí povinný atribut)

/ping Slouží mobilní aplikaci k ověření adresy a dostupnosti serveru. Server vždy vrátí **200**.

A GET požadavky na sub-doméně:

/ping pro jednoduché ověření, že server běží a je dostupný. Server vždy vrátí **200**.

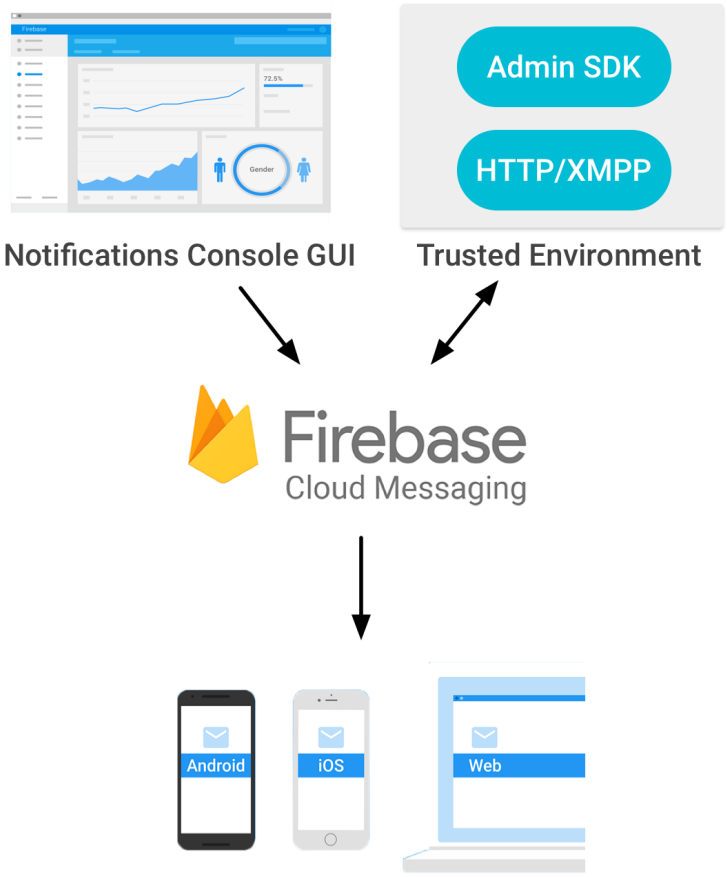
Přijaté události se na serveru nikdy neukládají.

Firebase Cloud Messaging

Pro komunikaci ve směru server → mobilní aplikace se používá FCM.

FCM implementace obsahuje dvě základní komponenty (obr. 6.3) pro odesílání a příjímání zpráv:

1. Důvěryhodné prostředí nebo aplikační server, na kterém je možné vytvářet a odesílat zprávy konkrétnímu zařízení nebo skupině.
2. Android, iOS nebo web (JavaScript) klientskou aplikaci, která přijímá zprávy.



Obrázek 6.3: Základní části FCM. Zdroj [7].

Pro účely testování nebo marketingu poskytuje Google online nástroj Notifications Console. Server implementovaný v této práci používá pro odesílání zpráv Admin FCM API pro Node.js. Ten umožňuje posílat notifikační (tzv. „Push notifikace“) a datové zprávy. Datové zprávy umožňují posílání vlastních JSON zpráv. Velikost je omezena na 4KB na zprávu [7]. Zprávu lze poslat skupině zařízení nebo jednomu konkrétnímu zařízení. Skupinové zprávy jsou „tématické“ a posílají se zájemcům o dané téma. Pro poslání zprávy konkrétnímu zařízení je nutné v FCM zprávě vyplnit *FCM ID* zařízení [28].

Prostřednictvím datových zpráv se doručují následujících příkazy:

FCM_SIGN_UP Posílá se prostřednictvím skupinové zprávy. Slouží k vyžádání *FCM ID* od všech mobilních zařízení, aby pak mohl posílat individuální zprávy konkrétním zařízením.

CLIENT_CONNECTED Zpráva pro konkrétní zařízení, že je aktivní párový počítačový klient. Zpráva je pouze informační a neočekává se na ni odpověď.

CLIENT_DISCONNECTED Zpráva pro konkrétní zařízení, že párový počítačový klient už není aktivní. Zpráva je opět pouze informační a neočekává se na ni odpověď.

Na serveru se *FCM ID* přiřadí k párovacímu tokenu zařízení uživatele (mobil i počítač). Obě hodnoty jsou poslány v jedné zprávě. Tyto dvojice identifikátorů umožňují serveru později určit mobilní zařízení (resp. aplikaci), na které má poslat informaci o změně stavu počítačového klienta (dostupný / nedostupný).

6.1.3 Počítač - Server

Obousměrnou komunikaci počítačový klient $\leftrightarrows$ server zajišťuje komunikační framework Socket.IO pro Node.js server. Klientská část je dostupná pro několik programovacích jazyků včetně Javy. Z pohledu serveru je každý připojený klient prezentován vlastním soketovým spojení, které má svůj identifikátor. Socket.IO dokáže detekovat ztrátu spojení a automaticky se jej snaží obnovit dokud se mu to nepodaří nebo dokud jedna strana nevynutí ukončení spojení. Komunikace skrze Socket.IO je založená na událostech. Každá změna stavu spojení nebo přijatá zpráva je událost, na kterou lze reagovat callback funkcí. Typ události se určuje podle jejího názvu. To umožňuje posílat zprávy různých typů a na každý typ mít připravenou extra obslužnou rutinu. Obdobně, jako když REST zpracovává zprávy na různých sub-doménách.

Počítačový klient naváže spojení se serverem přes HTTPS protokol na sub-doméně /socket. Po navázání spojení se serverem odešle klient registrační zprávu se svým aktuálním *ID* - párovací token. Server jej přiřadí k identifikátoru daného soketového spojení, aby později dokázal určit, kam přeposlat zprávu z mobilní aplikace. Dál už jen klient čeká na zprávy s událostmi.

6.1.4 Formát zprávy

HTTP a Socket.IO zprávy

Pro zprávy typu *A*, *B* a *C* (viz obr. 6.2) se používá jednotná struktura JSON zprávy (viz ukázka 6.1).

```
{
  "id": string,
  "content": array [strings|objects]
}
```

Ukázka kódu 6.1: Obecný formát vlastních zpráv

Atribut *id* je identifikátor uživatele a používá se k párování jeho zařízení na serveru a adresování zpráv. Vznikne zahešováním řetězce, který je složen ze dvou částí. Názvu mobilního zařízení, jež si volí sám uživatel a tzv. párovacího klíče, který se generuje v mobilní aplikaci. Tyto dvě hodnoty jsou po uživateli vyžadovány při prvním spuštění počítačového klienta.

V atributu *content* je obsah zprávy. Obsahem je JSON strukturovaný dle typu posílaných dat (událost nebo příkaz) nebo textový řetězec (zašifrovaná data). Je možné posílat více datových JSONů (např. událostí) v jedné zprávě, protože se v *content* vkládají do pole. Příklad uživatelské zprávy s událostmi před zašifrováním obsahu je v ukázce 6.2.

```
{
  "id":"8DNU0B69R3bqsuVZr...",
  "content": [
    {
      "content": {
        "name":"Android Studio",
        "number":"6505551212",
        "type":"MISSED",
        "when":1527960653971
      },
      "type":"CALL"
    },{
      "content": {
        "content":"Oreo\u0027s a slam dunk!",
        "name":"Android Studio",
        "number":"6505551212",
        "when":1527950271293
      },
      "type":"SMS"
    },{
      "content": {
        "app":"cz.zelenikr.remotetouch",
        "label":"RemoteTouch",
        "text":"Běží...",
        "title":"RemoteTouch",
        "when":1527959241750
      },
      "type":"NOTIFICATION"
    }
  ]
}
```

Ukázka kódu 6.2: Příklad uživatelské zprávy před šifrováním obsahu.

Po zašifrování obsahu s událostmi (ukázka 6.3) se zpráva odešle serveru.

```
{
  "id":"8DNU0B69R3bqsuVZr...",
  "content":
     ["dY2khmpezIOltAKfoABGzhb9388UuYYeNVsOaNZ6ZbXSBWxSzZaqe..."]
}
```

Ukázka kódu 6.3: Zpráva z ukázky 6.2 po zašifrování obsahu.

Na poslední ukázce (6.4) je zpráva typu *B*, konkrétně odpověď na příkaz pro FCM registraci zařízení:

```
{
  "id":"8DNU0B69R3bqsuVZr...",
  "content": [
    {
      "cmd":"FCM_SIGN_UP",
      "output":"fyai54bHBNI:APA91bEN5k..."
    }
  ]
}
```

Ukázka kódu 6.4: Zpráva s FCM ID

FCM zprávy

Zprávy typu *D* (viz obr. 6.2) jsou datové FCM zprávy (viz předchozí podkapitola 6.1.2, odstavec Firebase Cloud Messaging). Datová zpráva má dva povinné atributy - *token* a *data*. Atribut *token* obsahuje *FCM ID* cílového zařízení. V atributu *data* je obsah datové zprávy (data k odeslání). Obsahem datové zprávy je opět vlastní strukturovaný JSON.

FCM zprávy se používají pro posílaní příkazů. Seznam příkazů byl uveden v předchozí podkapitole 6.1.2, odstavec Firebase Cloud Messaging. Následující ukázka (6.5) je příkladem zprávy s „informativním příkazem“, že je aktivní počítačový klient:

```
{
  "token":"fyai54bHBNI:APA91bEN5k...",
  "data": {
    "cmd":"CLIENT_CONNECTED"
  }
}
```

Ukázka kódu 6.5: FCM zpráva, že je počítačový klient aktivní.

Tzv. tématické zprávy (viz předchozí podkapitola 6.1.2, odstavec Firebase Cloud Messaging) mají atribut *topic* namísto *token*. Ten obsahuje „téma“, kterého se zpráva týká. Tématickou zprávu používá server jako prostředek pro broadcast zprávy všem mobilním zařízením. Slouží k doručení příkazu `FCM_SIGN_UP` (poslání *FCM ID*). Zpráva (ukázka 6.6) má stejný formát jako předchozí ukázka:

```
{
  "topic":"cz.zelenikr.remotetouch",
  "data": {
    "cmd":"FCM_SIGN_UP"
  }
}
```

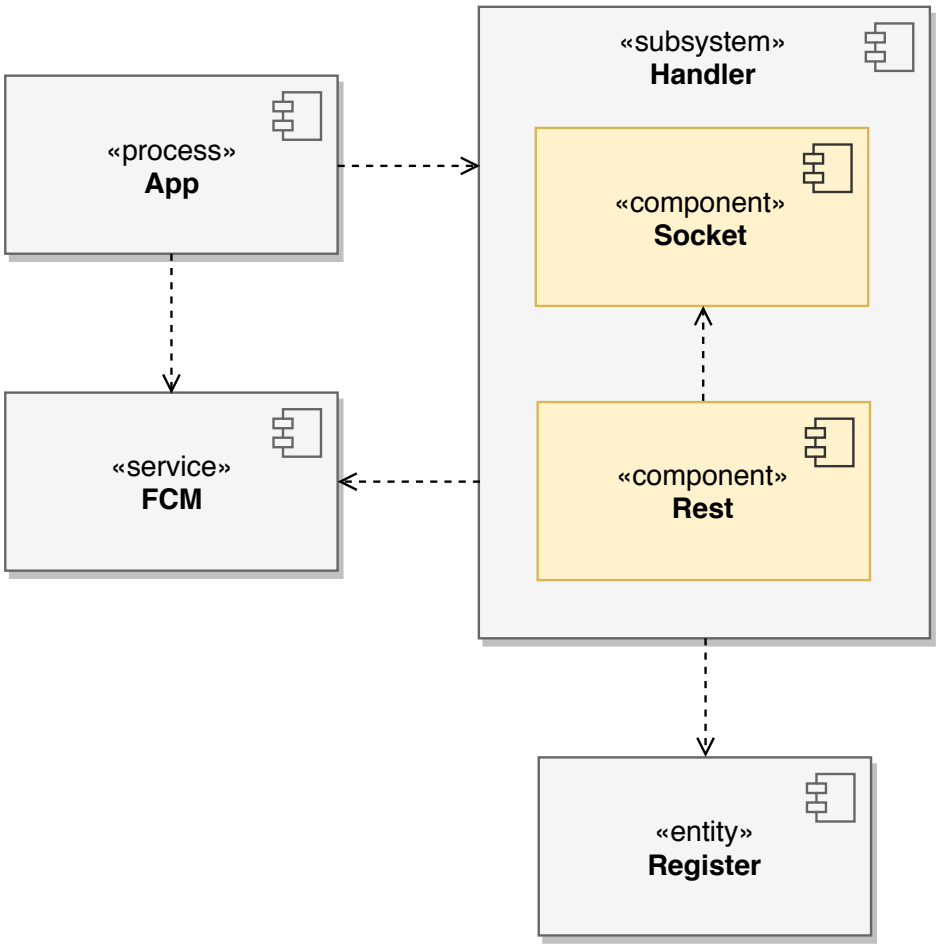
Ukázka kódu 6.6: FCM broadcast s žádostí o FCM ID.

6.2 Server

Server je postaven na platformě Node.js 8.9.4 LTS. Může běžet pod Windows, macOS, Linuxem nebo v Dockeru. Node.js je JavaScriptové běhové prostředí postavené na Chrome V8 JavaScript engine. Node.js používá neblokující I/O model řízený událostmi, který jej činí nenáročným a efektivním. Slouží k vytváření škálovatelných síťových aplikací. Používá vlastní balíčkovací ekosystém open-source knihoven - npm. [23]

6.2.1 Architektura

Server zprostředkovává komunikaci pro dvojice zařízení (mobil-počítač). Každé mobilní zařízení může komunikovat (být spárováno) právě s jedním počítačovým klientem. Následující schéma na obrázku 6.4 znázorňuje spolupráci hlavních komponent.



Obrázek 6.4: Server - diagram komponent

App komponenta

`App` je vstupním bodem programu - aplikace. Představuje samotný server. Importuje a inicializuje moduly Socket.IO a Express, resp. „soketovou“ a „restovou“ část serveru. Pro Socket.IO nadefinuje typy zpráv a přiřadí obsluhy z `Handler` komponenty. Stejně tak obsluhy sub-domén restové části (Express). Celý kód je v hlavním skript souboru `app.js`, který je spouštěn Node.js prostředím.

Zde se také nastavuje protokol transportní vrstvy (HTTP nebo HTTPS) a případně SSL certifikát. Pro účely testování byl server nasazen v cloudu a bylo nutné použít Nginx (softwarový webový server s load managementem a reverzní proxy), za kterým je Node.js server „schovaný“. Z tohoto důvodu je SSL certifikát načítán Nginxem a v `app.js` je nastaven HTTP.

Handler komponenta

`Handler` je opět jeden soubor - `api/handler.js`. Inicializuje registery a obsahuje sadu obslužných rutin rozdělených do dvou skupin:

Socket Zpracovává registrační zprávy od počítačového klienta a reaguje na jeho změny stavu (připojen / odpojen). Podle typu zprávy buď přidává nebo odebírá položky z `Register` komponenty a informuje mobilní aplikaci o změně stavu počítačového klienta prostřednictvím `FCM`.

Rest Zpracovává GET i POST požadavky. Aktualizuje *FCM ID* v registru nebo s jeho pomocí předává zprávy od mobilní aplikace počítačovému klientovi prostřednictvím `FCM`.

FCM komponenta

`FCM` komponenta importuje modul (knihovnu) Firebase Admin a inicializuje jej pro využívání Admin FCM API. Poskytuje obalovací funkce pro snadné posílání zpráv skrze FCM (Firebase Cloud Messaging).

Register komponenta

Mobilní aplikace (zařízení) a počítačové klienty je potřeba správně párovat. K tomu slouží registery. Register je datová struktura typu mapa, která umožňuje hledat podle klíče i podle hodnoty. V registeru jsou tedy unikátní klíče i hodnoty. Nemůže obsahovat např. dvojice *A:10* a *B:10* současně. Obsah registrů je uložen pouze v RAM. Používají se dva registry:

api/register/fcmClients.js Zde se ukládají dvojice *FCM ID* a *Client ID* (párovací token).

api/register/socketClients.js Zde se ukládají dvojice *Socket ID* a *Client ID*.

6.2.2 Použité knihovny

Express 4.16.3

Minimalistický webový framework pro Node.js. Je vhodným řešením pro single-page aplikace, webové stránky, hybridní nebo veřejná HTTP API [24].

Socket.IO 2.1.0

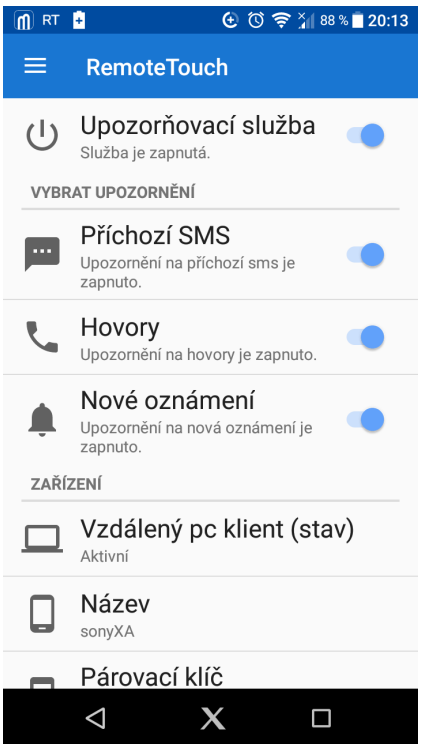
Framework pro real-time aplikace. Umožňuje obousměrnou real-time komunikaci (klient-server) založenou na událostech. Dokáže detekovat odpojení, umožňuje automaticky obnovit spojení, přenášet serializovatelné (*Serializable*) datové struktury a multiplex (dokáže vytvořit několik oddělených komunikačních kanálů pro skupiny klientů) [25].

Firebase Admin 5.12.0

Umožňuje Node.js serverům jednoduše využívat služby Google Firebase, jako je Firebase Cloud Messaging pro posílání notifikací nebo datových zpráv koncovým zařízením nebo skupinám zařízení [7].

6.3 Mobilní aplikace

Mobilní aplikace byla vyvinuta pro zařízení s operačním systémem Android verze Jelly Bean (4.3) a vyšší. Uživatelské rozhraní je kompletně v češtině. Náhled spuštěné aplikace je na obrázku 6.5.



Obrázek 6.5: Náhled hlavní obrazovky mobilní aplikace.

6.3.1 Architektura

Aplikaci lze rozdělit na dvě části. Jedna část běží na popředí a druhá na pozadí. Po otevření nainstalované aplikace se spustí hlavní aktivita s ovládáním a nastavením aplikace (viz předchozí obr. 6.5). To je první část. Za druhou část aplikace běžící na pozadí lze považovat službu/služby (dle nastavení uživatele), které běží na pozadí a to i po zavření aplikace. Tyto služby reagují na události v mobilním zařízení, zpracují je a odešlou počítačovému klientovi (skrze server).

6.3.2 UI

Obrazovky slouží uživateli hlavně k nastavení a přizpůsobení aplikace. Hlavní obrazovka je zároveň hlavní a jedinou aktivitou (Activity komponentou). Další obrazovky jsou implementovány prostřednictvím Fragment komponent. Rozložení ovládacích prvků každé obrazovky je definováno v XML souborech. Veškeré texty a popisky se načítají z textového souboru (`res/values/strings.xml`) prostřednictvím resources. To umožňuje snadnou budoucí lokalizaci pro další jazyky.

6.3.3 Nastavení

Veškerá změny v nastavení aplikace (včetně názvu zařízení, či párovacího klíče) uživatel provádí prostřednictvím PreferencesFragment komponent jež hodnoty automaticky ukládají do SharedPreferences (více v předešlé podkapitole 4.4.2, odstavec Ukládání předvoleb a nastavení).

Ostatní komponenty mohou k nastavení aplikace přistupovat prostřednictvím metod třídy `SettingsHelper`.

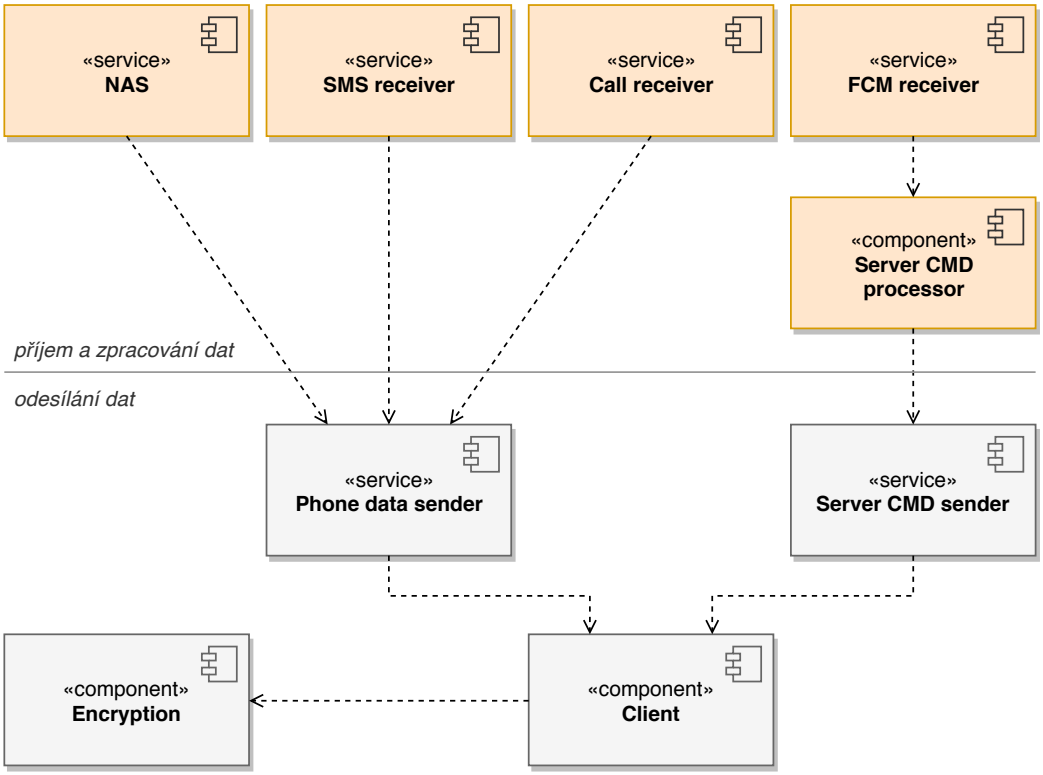
Možnosti nastavení aplikace jsou popsány v uživatelské dokumentaci.

6.3.4 Události

Na hlavní obrazovce je vypínač upozorňovací služby. Tato služba běží v systému na pozadí. Je-li vypnutá, počítačovému klientovi se žádné události nepřeposílají. Při zapnuté službě si uživatel může zvolit (zapnout/vypnout) typy upozornění (událostí), které se mají přeposílat do počítačového klienta. Není-li však počítačový klient aktivní (tedy není připojen k serveru), žádně události nejsou odesílány.

Diagram na obrázku 6.6 znázorňuje hlavní komponenty, které reagují

na nové události v systému (např. přijatá zpráva, příchozí hovor nebo nová notifikace) s ohledem na preference uživatele. Tyto události zpracují a odešlou skrze server počítačovému klientovi.



Obrázek 6.6: Mobilní aplikace - digram komponent pro obsluhu událostí

NAS (NotificationAccessService)

`NotificationAccessService` je třída, která rozšiřuje speciální Android službu NotificationListenerService. Ta umožňuje přijímat od operačního systému objekt StatusBarNotification (dále SBN). SBN představuje novou nebo právě změněnou notifikaci jiné aplikace.

Pokud je `NAS` uživatelem zapnutá (přesněji povolená), zkontroluje, zda notifikace patří aplikaci, která je na seznamu zájmových aplikací (aplikací vy-

braných uživatelem). Na rozdíl od `SMSReceiver` a `CallReceiver` komponent běží tato Android služba neustále. Aby mohla přijímat zprávy od operačního systému, musí jím být sama zapnuta. Přepínač v nastavení tedy pouze určuje, zda má `NAS` přijatý SBN zpracovat či ignorovat.

Data z SBN se předají jako DTO objekt komponentě `MessageSender`. DTO obsahuje název aplikace, titulek a text notifikace a časovou značku, kdy byla notifikace vytvořena (a nebo změněna).

SMSReceiver

`SMSReceiver` je třída, která rozšiřuje BroadcastReceiver. Když přijde nová SMS, `SMSReceiver` je inicializován operačním systémem a obdrží Intent s číslem odesílatele a obsahem zprávy (viz také předchozí podkapitola 4.4.2, odstavec Přístup k textovým zprávám).

`SMSReceiver` není Android služba jako `NAS`, takže neběží (resp. nečeká) stále na pozadí. Životní cyklus BroadcastReceiver končí po zpracování přijatého Intent objektu. Je-li uživatelem vypnuto upozornění na SMS, `SMSReceiver` nebude při přijetí nové SMS Androidem vůbec inicializován.

Data obsažená v Intent předá jako DTO objekt komponentě `MessageSender`. DTO obsahuje číslo a jméno (je-li v kontaktech) odesilatele, text zprávy a časovou značku, kdy byla zpráva odeslána.

CallReceiver

`CallReceiver` je další třída, která rozšiřuje BroadcastReceiver. Přijímá dva typy Intent akcí (PHONE_STATE a NEW_OUTGOING_CALL), viz předchozí kapitola 4.4.2, odstavec Přístup k hovorům. Z jejich dat dokáže rozlišit pět stavů:

- Příchozí hovor (zvoní telefon)
- Probíhá hovor (uživatel přijal hovor)
- Hovor ukončen (uživatel ukončil hovor)
- Zmeškaný hovor (telefon přestal zvonit)
- Odchozí hovor (telefon vytáčí číslo)

Je-li uživatelem vypnuto upozornění na hovory, `CallReceiver` nebude inicializován ani pro jeden registrovaný Intent.

Získaná data jsou opět předána jako DTO objekt komponentě `MessageSender`. DTO obsahuje číslo a jméno (je-li v kontaktech) volajícího, typ hovoru a časovou značku začátku události.

FCMReceiver komponenta

`FCMReceiver` komponenta jsou dvě (FCM) Android služby (třídy), které je nutné oddědit (FirebaseInstanceIdService a FirebaseMessagingService). Jedna slouží k získání *FCM ID* od Google Firebase a druhá k přijímání a odesílání FCM zpráv se serverem. Přijaté FCM zprávy (příkazy, viz podkapitola 6.1.2, odstavec Firebase Cloud Messaging) předává ke zpracování komponentě `ServerCMDReceiver`.

ServerCMDReceiver

`ServerCMDReceiver` je třída, která rozšiřuje BroadcastReceiver. Slouží k obsluze a zpracování příkazů přijatých od serveru. Příkazy mohou být typu:

FCM_SIGN_UP Načte *FCM ID* a odešle ho serveru prostřednictvím `ServerCMDSender` komponenty.

CLIENT_CONNECTED Změní stav vzdáleného (počítačového) klienta na *aktivní* prostřednictvím `SettingsHelper`. Potom počítačovému klientovi odešle aktuální stav zařízení. To znamená, že načte prostřednictvím `SmsHelper`, `CallHelper` a `NotificationHelper` komponent nepřečtené SMS zprávy, seznam nepřijatých hovorů a existující notifikace zájmových aplikací. Tato data ve formě DTO objektů předá komponentě `MessageSender`.

CLIENT_DISCONNECTED Změní stav vzdáleného (počítačového) klienta na *neaktivní*. To znamená, že nebude odesílat žádné události, protože je žádný počítačový klient nebude přijímat.

ServerCMDSender

`ServerCMDSender` je Android služba, konkrétně JobIntentService. Funguje obdobně jako BroadcastReceiver. Zpracuje přijatý `Intent` a systém ji ukončí. Na rozdíl od BroadcastReceiver komponenty je vhodná na časově náročnější operace (např. síťové), protože je vykonává v dalším vlákně.

V aplikaci slouží k odesílání příkazů serveru. Příkaz je reprezentován speciálním DTO objektem, který obsahuje typ příkazu a případně odpověď (např. na `FCM_SIGN_UP`). Příkazy odesílá serveru prostřednictvím části `Client` komponenty, která nešifruje obsah zprávy end-to-end šifrováním.

MessageSender

`MessageSenderService` je třída, která rozšiřuje obecnou Android službu, Service. Uživateli je prezentována jako Upozorňovací služba. Běží neustále na pozadí systému, dokud ji uživatel sám nevypne. V aplikaci je spouštěna metodou startForeground. Díky tomu systém ví, že ji nemá ukončovat, pokud to není nutné (např. kvůli paměti). I když ji ukončí, služba se opět spustí. Také to zajišťuje zobrazení persistentní notifikace po celou dobu běhu služby.

Služba se stará o odesílání uživatelských dat (tzn. zpracované události nebo souhrný stav zařízení), viz `NotificationAccessService`, `SMSReceiver`, `CallReceiver` a `ServerCMDReceiver`. Je-li uživatelem vypnuta, systém ji nespustí ani když jí jiná komponenta posílá data.

Přijaté DTO objekty odesílá serveru prostřednictvím části `Client` komponenty, která šifruje obsah zprávy end-to-end šifrováním.

Client komponenta

`Client` slouží k odesílání zpráv serveru ve formě HTTP POST požadavků. Komponenty `MessageSenderService` a `ServerCMDSender` mu předají DTO objekty s daty k odeslání. `Client` objekty serializuje do textové podoby (JSON) a vloží do POST požadavku v požadovaném formátu (viz podkapitola 6.1.4).

Také je zodpovědný za šifrování obsahu zprávy (serializovaného objektu). K tomu využívá `Encryption` komponentu. Šifrovacím klíčem je tzv. párovací klíč, který si uživatel může zobrazit na hlavní obrazovce a v případě potřeby si vygenerovat nový. `Client` vždy používá aktuální klíč.

Encryption komponenta

Umožňuje šifrovat a dešifrovat textová data algoritmem AES se 128 bitovým klíčem. Používá existující implementaci algoritmu v javax.crypto.Cipher. `Client` ji používá k šifrování obsahu odesílaných zpráv.

Komponenta také slouží ke generování nového párovacího (a šifrovacího) klíče. Ten se generuje prostřednictvím javax.crypto.KeyGenerator jako SecretKey objekt.

6.3.5 Použité knihovny

Google HTTP Client 1.23.0

Jde o Google knihovnu pro jazyk Java, která umožňuje efektivně přistupovat k webovému obsahu prostřednictvím HTTP. Obsahuje upravenou implementaci přímo pro Android. Mimo jiné také dokáže snadno konvertovat (serializací) objekty do JSON formátu.

Firebase Messaging 15.0.2

`FirebaseMessaging` API slouží k implementaci FCM Android klienta, respektive `FCMReceiver` komponenty. Umožňuje přijímat notifikační a datové zprávy nebo se registrovat pro skupinové zprávy (viz podkapitola 6.1.2, odstavec Firebase Cloud Messaging).

WaveLoadingView 0.3.5

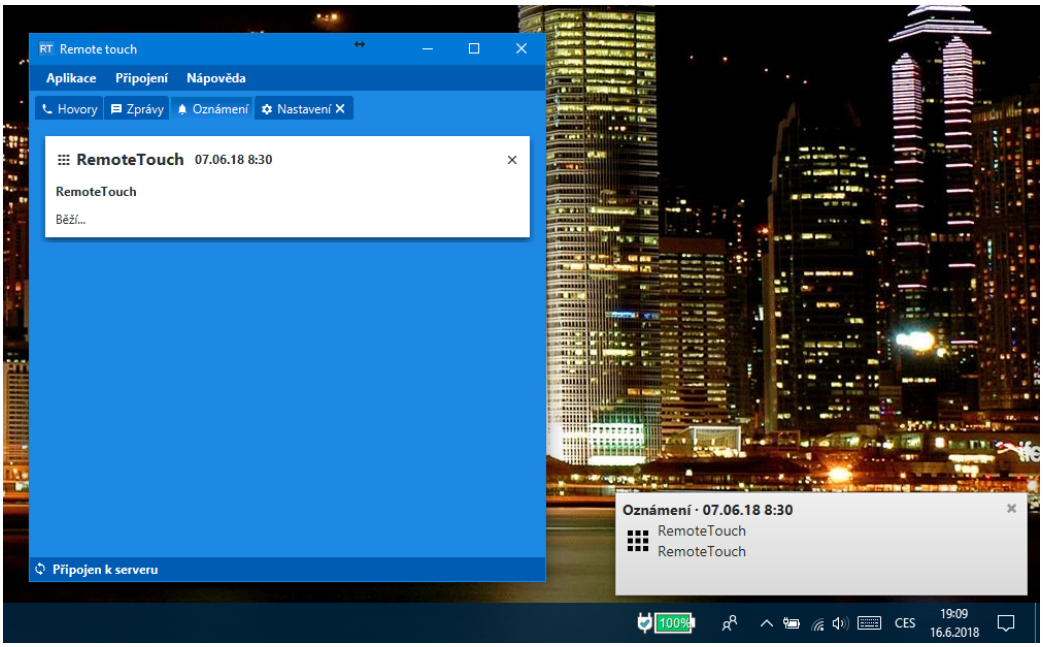
Android knihovna pro snadné zobrazení animace během načítání či zpracování dat. Animace má podobu vlny na vodní hladině a lze si ji snadno přizpůsobit (barva, výška hladiny, atd.). V aplikaci se používá při načítání seznamu nainstalovaných aplikací v zařízení.

Android About Page 1.2.4

Umožňuje jednoduše vytvořit „About“ obrazovku aplikace. Je strukturovaná, přizpůsobitelná a skládá se z předefinovaných elementů (např. ikona, popis, kontakt).

6.4 Počítačový klient

Počítačový klient je JavaFX aplikace. Uživatelské rozhraní je kompletně v češtině. Klient byl vyvíjen v Java SE 10 a minimální požadovaná verze je Java SE 9. Náhled aplikace je na obrázku 6.7.



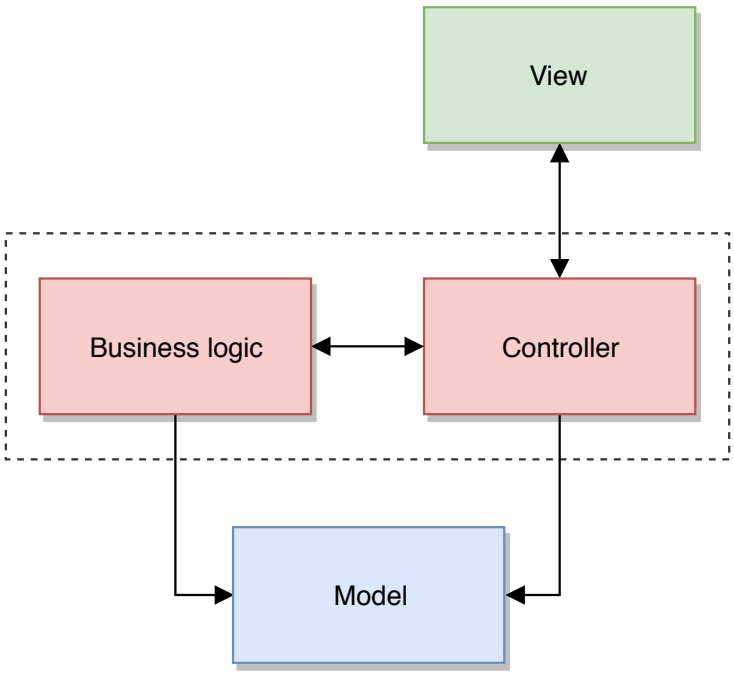
Obrázek 6.7: Náhled počítačového klienta s novým oznámením (v pravém spodním rohu obrazovky).

6.4.1 Architektura

Vstupním bodem programu je třída `MainFX`, která rozšiřuje javafx.application.Application a obsahuje metodu main. Po spuštění jar souboru řídí načtení a zobrazení okna pro prvotní nastavení či odemknutí aplikace a zobrazení hlavního UI okna.

Aplikace má tří, resp. čtyř vrstvou architekturu. Prezentační vrstvu, lo-

gickou vrstvu + kontrolery a datový model. Viz následující diagram (obr. 6.8):



Obrázek 6.8: Počítačový klient - architektura

6.4.2 View - prezentační vrstva

Uživatelské prostředí je složené z několika částí. Všechny části (např. záložky s přijatými událostmi, nastavením nebo položky v seznamech) mají svou strukturu UI komponent definovanou v .fxml souborech v `resources/view/`. FXML je značkovací jazyk založený na XML, který byl vyvinut v rámci JavaFX. Slouží k definování struktury UI komponent, podobně jako např. HTML pro webové stránky.

Vzhled aplikace (jejích komponent) je definován prostřednictvím CSS stylů. Těmi lze měnit výchozí vzhled komponent. Veškeré popisky a texty v aplikaci jsou čteny z `resources/values/strings_cs.properties` souboru. To umožňuje jejich rychlou a jednoduchou úpravu. Přidáním dalšího

`strings_` souboru pro jiný jazyk lze jednoduše rozšířit lokalizaci aplikace.

Každému FXML soubor lze přiřadit kontroler. Buď se určí přímo v FXML souboru a nebo programově před načtením samotného FXML souboru.

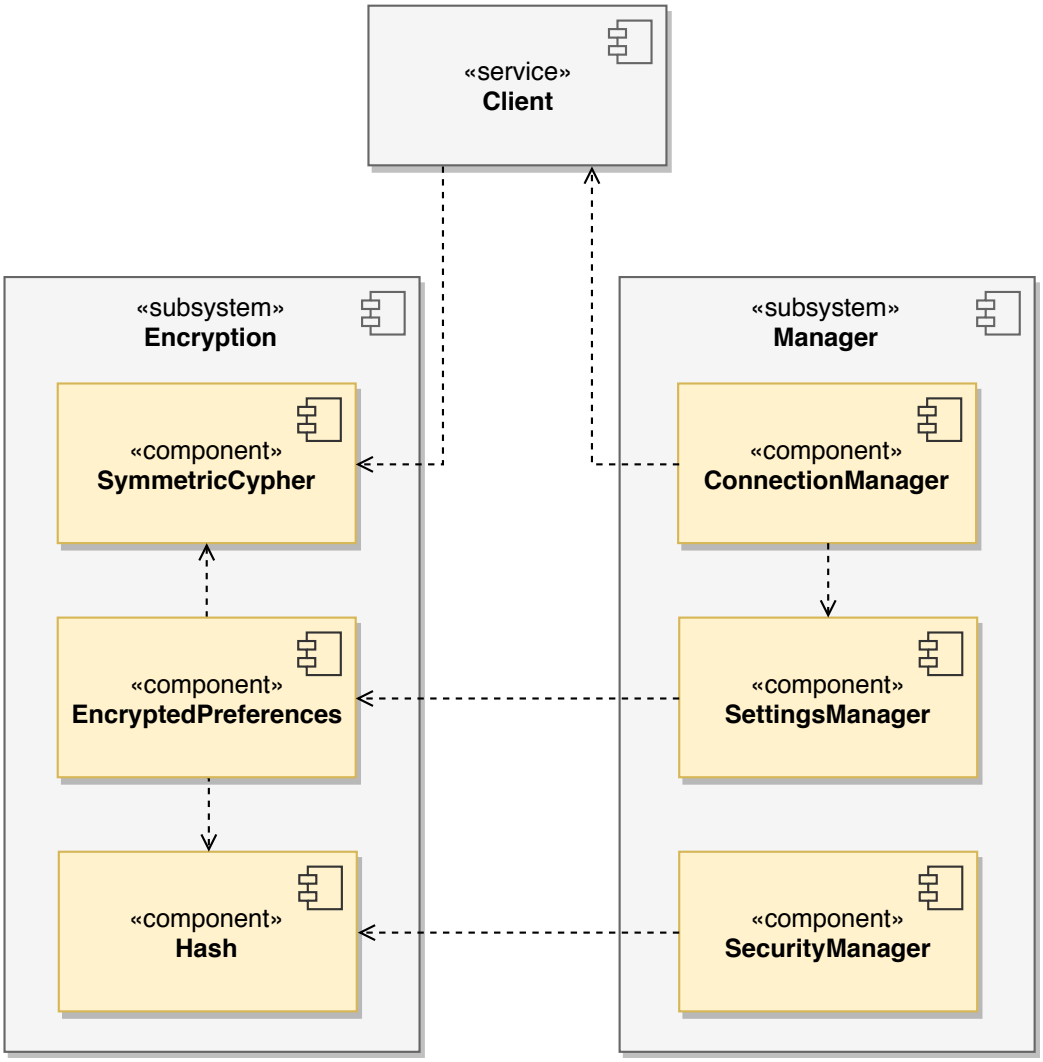
6.4.3 Controller (kontroler)

Kontroler je vždy pojmenován podle FXML souboru, ke kterému je přiřazen. Kontrolery slouží hlavně k obsluze událostí vyvolaných interakcí uživatele (kliknutí v menu, změna hodnoty v nastavení, atd.) nebo zpracování hodnot formulářů (odemknutí aplikace či změna nastavení). Také iniciují aktualizaci vzhledu aplikace přidáváním nebo odebíráním položek ze seznamů přijatých zpráv (událostí). Obsahy seznamů (jejich data) jsou drženy ve struktuře ObservableList v příslušných kontrolerech. Přijaté události nejsou totiž ukládány v permanentním úložišti. Kontrolery také úzce spolupracují s komponentami logické vrstvy.

Hlavním kontrolerem je `AppController`, který po zobrazení hlavního okna přebírá řízení aplikace od `MainFX`. Obsluhuje všechny položky menu včetně ukončení aplikace, aktualizuje stav připojení a iniciuje zobrazení informační notifikace uživateli, že přišla nová zpráva s událostí.

6.4.4 Business logic - logická vrstva

Komponenty logické vrstvy (především `*Manager` komponenty) jsou využívány kontrolery a společně mění obsah datového modelu, resp. seznamy přijatých událostí. Diagram 6.9 zobrazuje kooperaci hlavních komponent logické vrstvy.



Obrázek 6.9: Počítačový klient - diagram komponent logické vrstvy

Client komponenta

Komponenta `Client` zajišťuje obousměrnou komunikaci se serverem prostřednictvím knihovny Socket.IO-client. Hlavní úlohou klienta je čekání na uživatelské zprávy - nové události. Obsah přijaté zprávy dešifruje prostřednictvím `SymmetricCypher` komponenty a párovacího klíče. Z dešifrovaného

(textového) obsahu zprávy vytvoří DTO objekt, který může být typu SMS, hovor nebo notifikace. Informuje aplikaci o nově přijaté události a předá DTO objekt k dalšímu zpracování (viz `ConnectionManager`).

Encryption komponenta

SymmetricCypher Umožňuje jednoduše šifrovat a dešifrovat textová data algoritmem AES se 128 bitovým klíčem. Implementace je shodná s tou v mobilní aplikaci.

Hash Umožňuje jednoduše „zahešovat“ text prostřednictvím funkce SHA-256.

EncryptedPreferences Slouží k ukládání a čtení předvoleb (nastavení) aplikace ve formátu klíč-hodnota. Využívá výchozí implementace Java Preferences, ale klíč i hodnota jsou uloženy v šifrované podobě. Klíče i hodnoty jsou šifrovány heslem uživatele, kterým se přihlásil k aplikaci. Samotné šifrování a dešifrování zajišťuje `SymmetricCypher` komponenta.

Manager komponenta

ConnectionManager Obecně zajišťuje komunikaci aplikace se vzdáleným zařízením, tedy mobilním zařízením uživatele. K tomu používá výše zmíněnou komponentu `Client`. Další části aplikace (jako `AppController`) si u něj mohou zaregistrovat posluchače dle typu přijaté události (SMS, hovor, notifikace) a být tak o události informovány.

SettingsManager Slouží k ukládání a čtení nastavení aplikace (název mobilního zařízení, párovací klíč, adresa serveru). K tomuto účelu používá `EncryptedPreferences` komponentu, která ukládá název i hodnotu ukládané položky šifrovaně (na rozdíl od výchozí Preferences implementace poskytované Javou). Položky se šifrují

heslem, kterým se uživatel přihlašuje při každém spuštění aplikace.

SecurityManager Spravuje a ověřuje tzv. „vlastníka“ aplikace. To znamená, že ověřuje uživatele při spuštění aplikace. Při prvním spuštění aplikace na konkrétním počítači si uživatel zvolí heslo, kterým bude počítačová aplikace chráněna. Při dalším spuštění je po něm toto heslo vyžadováno. `SecurityManager` se stará o vytvoření, uložení a ověření tohoto hesla. Heslo „zahešuje“ `Hash` komponentou a uloží prostřednictvím Java Preferences. Pokud ho uživatel zapomene, může jej resetovat a zvolit si heslo nové. Po nastavení nového hesla je nutné opět spárovat zařízení (tzn. vyplnit název zařízení a párovací klíč).

6.4.5 Model - datový model

Přijatá data (nepřečtené zprávy, nepřijaté hovory a nové události) se v počítači permanentně neukládají. Jsou uchovávána pouze v RAM po dobu běhu počítačového klienta. Jednotlivé typy dat (SMS, hovory, notifikace) zobrazené v seznamech jsou uloženy v kolekcích v příslušných kontrolerech.

6.4.6 Použité knihovny

Gson 2.8.5

Google knihovna pro práci s JSON objekty v Javě. Kromě jiného umožňuje převod JSON objektu do/z textového formátu serializací a deserializací Java DTO objektu.

Socket.IO-client 1.0.0

Implementace klientské části frameworku Socket.IO pro jazyk Java.

ControlsFX 9.0.0

Rozšiřuje JavaFX o další vlastní UI komponenty. V aplikaci se používají především její validátory formulářových komponent a API pro vytvoření a zobrazení notifikačních oken.

FontAwesomeFX 9.1.2

Balík několika sad glyph (grafický symbol, který definuje vzhled nebo podobu znaku) ikon, které lze snadno přidat do UI komponent. V aplikace se používají sady FontAwesome 4.7 a Material Icons 2.2.

7 Testování

Všechny části realizovaného systému (mobilní aplikace, počítačový klient, server) byly testovány v průběhu celého vývoje. Jednotlivé komponenty každé části systému byly během jejich implementace testovány a laděny.

V rámci testování funkčnosti systému jako celku bylo připraveno několik testovacích scénářů, které simulovaly využití systému v reálných situacích.

7.1 Běhové prostředí

7.1.1 Počítačový klient

Počítačový klient byl testován na školním i osobním počítači s Windows 10 a Javou SE 10.

7.1.2 Mobilní aplikace

Mobilní aplikace byla průběžně testována na několika virtuálních a reálných mobilních zařízení. Přehled těchto zařízení je uveden v tabulce 7.1

Zařízení	verze Androidu	Rozlišení displeje [pix]
Android Emulator	4.3 (Jelly Bean)	480 x 800
Android Emulator	8.0 (Oreo)	1440 x 2560
Samsung Galaxy J5 (2016)	6.0.1 (Marshmallow)	1280 x 720
Sony Xperia XA	7.0 (Nougat)	1280 x 720

Tabulka 7.1: Přehled použitých mobilních zařízení v průběhu vývoje a testování mobilní aplikace.

7.1.3 Server

Funkčnost serveru byla v počátku jeho vývoje testována na osobním počítači s Node.js 8.9.4 LTS. V rámci testování spolupráce mobilní aplikace a počítačového klienta byl server později nasazen v cloudu s veřejnou webovou adresou. Běh serveru byl testován také na školním počítači (pouze s HTTP).

Microsoft Azure

Nejprve byl server nasazen jeden měsíc v Microsoft Azure v rámci bezplatné zkušební licence jako App Service (měsíc je limit pro bezplatný provoz). App Service je služba typu PaaS, tedy platforma jako služba. Vytvořená Node.js aplikace (server) má přidělenou veřejnou webovou adresu a automaticky používá HTTPS protokol. Nebylo potřeba řešit vlastní SSL certifikát.

Google App Engine

Narozdíl od Microsoft Azure nabízí Google Cloud bezplatnou zkušební licenci, která vyprší za rok nebo po spotřebování přiděleného kreditu ($ 300).

Google App Engine (také PaaS) je jednou z nabízených služeb v Google Cloud. Node.js server opět dostal přidělenou veřejnou webovou adresu se zabezpečeným spojením, jako v předchozím případě. Jelikož ale docházelo k neustálému odpojování a připojování počítačového klienta (každé 2 vteřiny), byl server nakonec přesunut na Google Compute Engine. Google App Engine totiž zatím nepodporuje WebSocket protokol, který využívá použitý komunikační framework Socket.IO.

Google Compute Engine

Jde o další službu nabízenou v Google Cloud, avšak toto je služba typu IaaS, tedy infrastruktura jako služba. Byla vytvořena VM (Virtual Machine) instance typu *f1-micro* s operačním systémem Linux a přidělenými prostředky: 1 CPU (Intel Haswell), 0.6 GB RAM a 10 GB úložného prostoru na disku. Server má přidělenou pouze veřejnou IP adresu. Bylo tedy nutné zaregistrovat pro ni u třetí strany (Freenom) doménu a následně pro tuto doménu vygenerovat SSL certifikát (SSL For Free). Aby mohl Node.js server veřejně naslouchat na portech 80 a 443, bylo potřeba ještě použít Nginx jako proxy mezi Node.js a internetem.

Server zde běžel bezproblémově následující 3 měsíce (než byl ukončen vývoj systému v rámci této práce). Pro účely předvedení funkčnosti systému oponentovi této práce a pro osobní využívání zde server zůstal běžet i nadále.

7.2 Testovací scénáře

V průběhu obou variant (A, B) byl již server nasazen v cloudu (Google Compute Engine).

A. Testování jednotlivých dvojic zařízení

Dvojice zařízení znamená mobilní aplikace (zařízení) spárovaná s počítačovým klientem. Testování probíhalo ručně. Touto variantou bylo postupně testováno několik dvojic, respektive mobilních zařízení:

- počítač. klient a emulátor Androidu 8 (Oreo)
- počítač. klient a Sony Xperia XA s Androidem 7 (Nougat)
- počítač. klient a Samsung Galaxy J5 s Androidem 6 (Marshmallow)

Testovaná funkcionalita

V případě emulátoru byly hovory a SMS simulovány vývojovým prostředím Android Studio.

V následujících scénářích (1-4) byl počítačový klient vždy spárován a aktivní a mobilní aplikace využívala Wi-Fi připojení.

1. V mobilní aplikaci jsou zapnuta jednotlivá upozornění, ale samotná upozorňovací služba je vypnutá.

   Očekávané chování: Aplikace ignoruje veškeré události (např. příchozí hovor) a nic neodesílá.

2. Upozorňovací služba je zapnuta, ale jednotlivá upozornění jsou vypnuta.

   Očekávané chování: Aplikace ignoruje veškeré události (např. příchozí hovor) a nic neodesílá.

3. Upozorňovací služba a upozornění na SMS je zapnuto.

   Očekávané chování: Aplikace přepošle pouze nově přijatou SMS zprávu a příchozí hovor ignoruje. Počítačový klient upozorňuje na novou zprávu.

4. V mobilní aplikaci je vygenerován nový klíč, ale není přepsán do počítačového klienta, čímž je porušeno párování zařízení.

   Očekávané chování: Aplikace aktualizuje stav vzdáleného klienta na *neaktivní* a neodesílá žádné nové události.

V dalších scénářích (5-10) byla v mobilní aplikaci vždy zapnuta upozorňovací služba včetně jednotlivých upozornění.

5. Počítačový klient je po spárování odpojen od serveru.

Očekávané chování: Mobilní aplikace aktualizuje stav vzdáleného klienta na *neaktivní* a neodesílá žádné nové události.

6. Dvojice byla již spárována. V mobilním zařízení jsou 2 zmeškané hovory (staré několik minut) a nepřečtená SMS zpráva. Následně je spuštěn počítačový klient.

   Očekávané chování: Po připojení počítačového klienta k serveru se do pár vteřin v mobilní aplikaci změní stav vzdáleného klienta na *aktivní* a aplikace následně odešle aktuální stav počítačovému klientovi (tzn. odešle oba zmeškané hovory a nepřečtenou SMS zprávu). Počítačový klient upozorní uživatele na všechny 3 události. Z času a data je u každé události vidět, kdy nastala.

7. Počítačový klient je aktivní (připojen k serveru). Uživateli přijde SMS zpráva s diakritikou a emotikony.

   Očekávané chování: Počítačový klient upozorní na novou SMS. Celý obsah zprávy je korektně zobrazen (diakritika i emotikony jsou shodné s původním obsahem v telefonu).

8. V mobilní aplikaci je zvoleno několik zájmových aplikací (např. WhatsApp a Gmail). Přijde WhatsApp zpráva. Pak je WhatsApp odebrán ze zájmových aplikací. Přijde další WhatsApp zpráva a email do Gmail schránky.

   Očekávané chování: První přijatá WhatsApp zpráva se odešle počítačovému klientovi, druhá už ne. Email se přepošle (resp. obsah Gmail notifikace).

9. V mobilní aplikaci jsou povolena pouze mobilní data. Mobilní zařízení je připojeno k internetu prostřednictvím Wi-Fi, mobilní data jsou vypnuta. Přijde nová SMS. Následně uživatel zapne v mobilním zařízení mobilní data a vzdálí se z dosahu Wi-Fi sítě (zařízení je donuceno přejít na mobilní připojení k internetu). Po té přijde další SMS.

   Očekávané chování: První SMS zpráva se počítačovému klientovi nepřepošle. Druhá SMS už se přepošle, protože zařízení používá typ

povoleného připojení (mobilní data).

10. Zařízení jsou spárována. V počítačovém klientovi je změněn název zařízení a je restartováno spojení se serverem kvůli uplatnění provedených změn. Uživatel zkusí z mobilní aplikace odeslat zkušební oznámení. Po té je stejný název zařízení zadán i do mobilní aplikace.

    Očekávané chování: Po restartování klientovo spojení se serverem se do pár vteřin v mobilní aplikaci změní stav vzdáleného klienta na *neaktivní*. Zkušební oznámení se vytvoří, ale nezobrazí se v počítačovém klientovi. Po změně názvu zařízení i v mobilní aplikaci se stav klienta změní na *aktivní* a následně je odeslán aktuální stav mobilního zařízení (tzn. odešle se již existující zkušební oznámení), který se zobrazí v počítačovém klientovi.

Vyhodnocení

Mobilní aplikace se na všech testovaných zařízeních chovala konzistentně a dle očekávání. Počítačový klient také.

Pozn.: scénář č. 7, emotikony byly dekódovány správně, ale nezobrazují se barevně, jako v mobilním zařízení.

B. Testování několika dvojic zařízení

V této variantě se testovaly dvě dvojice zařízení současně, aby se ověřilo především správné směrování zpráv na serveru. Byla použita reálná mobilní zařízení z předchozí varianty A. Každé bylo spárováno s vlastním počítačovým klientem. Každý počítačový klient byl spuštěn na jiném počítači.

Mobilní telefony si navzájem poslaly několik SMS zpráv, WhatsApp zpráv (pro testování notifikací) a prozvonily se.

Vyhodnocení

Server správně přesměroval všechny přijaté události od mobilních zařízení patřičným (spárovaným) počítačovým klientům.

7.3 Zhodnocení funkčnosti

Systém se chová konzistentně a stabilně. Všechny připravené testovací scénáře byly úspěšné. Systém byl testován na několika mobilních zařízeních. Občasná (časová) prodleva (řádově vteřiny) v reakci mobilní aplikace na změnu stavu (připojen/odpojen od serveru) počítačového klienta je dána zpožděním při doručení FCM zpráv mobilnímu zařízení.

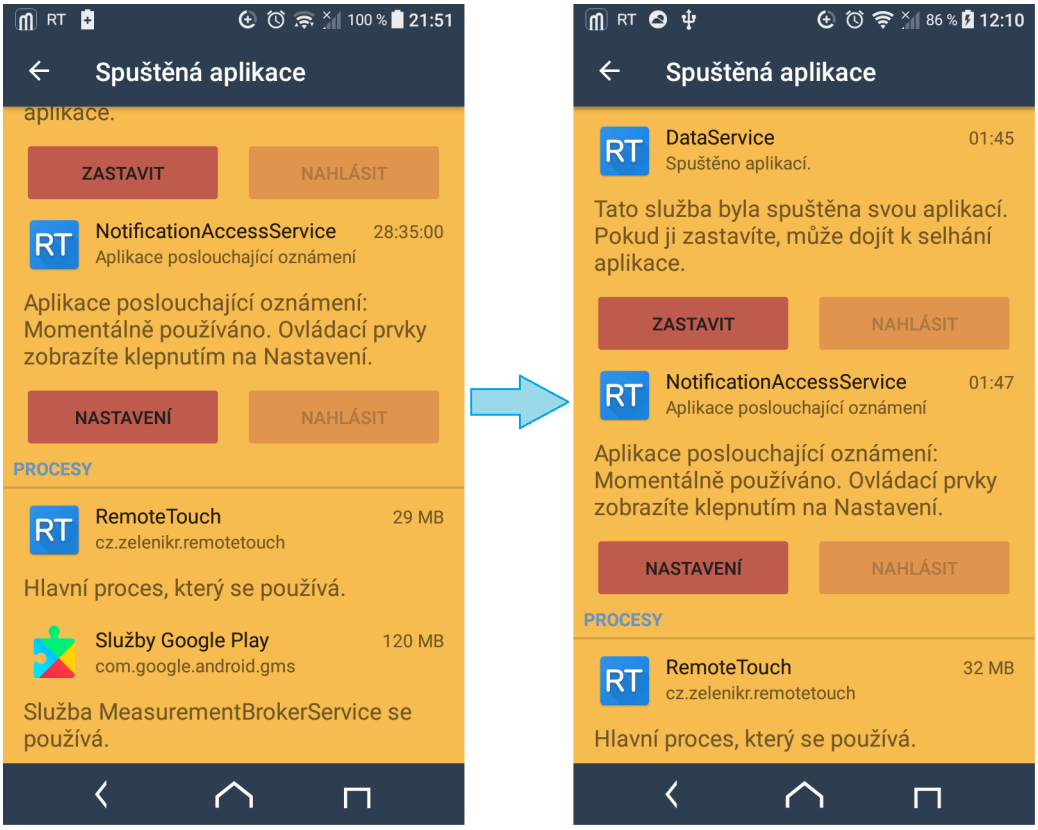
Server byl nasazen v cloudu, kde běžel nepřetržitě a stabilně několik měsíců.

Mobilní aplikace byla také používána několik týdnů v reálném mobilním zařízení. Aplikace (přesněji upozorňovací služba) byla aktivní a běžela nepřetržitě. Během používání aplikace nebyl zaznamenán vliv na výdrž baterie či větší spotřebu dat (desítky až stovky KB).

Analýza paměťové náročnosti

V rámci testování byla také zjišťována spotřeba RAM paměti mobilní aplikací. Tuto hodnotu lze získat v nastavení mobilního zařízení prostřednictvím režimu „Pro vývojáře“. Kromě vlastních očekávaných dvou služeb zde běžela ještě jedna, MeasurementBrokerService, která vyžadovala navíc 120 MB RAM paměti. Po malém průzkumu bylo zjištěno, že je tato služba využívána nástrojem Firebase Analytics, který sbírá informace o chodu a chybách v FCM klientovi a odesílá je k analýze. Řešením bylo zakázat tento modul v dependencies Android projektu. Na obrázku 7.1 je srovnání stavu před (vlevo)

a po (vpravo). Vlevo běží navíc jako proces MeasurementBrokerService pod Službami Google Play, kdežto vpravo již běží jen hlavní proces RemoteTouch.



Obrázek 7.1: Porovnání spotřeby RAM paměti před a po odebrání Firebase Analytics modulu z projektu.

7.4 Možná rozšíření

7.4.1 Rozšíření systému

Systém by mohl být v budoucnu rozšířen na další platformy. Počítačový klient by mohl být kromě desktopového také webový. To by zahrnovalo roz-

šířit server o část poskytující front-end webové aplikace. Vzhledem ke komunikaci prostřednictvím Socket.IO by bylo možné využít stávající back-end. Zásadním faktorem bude podpora Notification API ve webových prohlížečích.

Dalším rozšířením by mohla být implementace mobilní aplikace pro zařízení se systémem iOS. Ta je ovšem podmíněna poskytnutím API ze strany iOS vývojářů, které by umožnilo přístup k SMS zprávám, hovorům či notifikacím ostatních aplikací.

7.4.2 Další funkce

- Možnost přímého spojení mobilní aplikace a počítačového klienta prostřednictvím Bluetooth. V takových situacích by nebyl server třeba a mohl by být vynecháván z komunikace (nebyl by potřebný jako zprostředkovatel komunikace).
- GPS souřadnice mobilního zařízení na vyžádání. Počítačový klient je automaticky zadá přes WebView do mapy a uživatel vidí, kde telefon nechal (zapomněl).
- Interakce s událostmi v počítačovém klientovi. Např. možnost odpovědět na SMS zprávu, označit ji jako přečtenou či označit oznámení (notifikaci) jako vyřízené (= odebrat ze stavového řádku mobilního zařízení).
- Na vyžádání uživatele mobilní aplikace odešle stav baterie (úroveň nabití) a jestli se právě mobilní zařízení nabíjí.

8 Závěr

Cílem této diplomové práce bylo navrhnout a vytvořit systém, který upozorní uživatele na obrazovce jeho počítače na příchozí nebo zmeškané hovory (včetně identifikace volajícího), zobrazí nepřečtené nebo nové SMS zprávy a zobrazí notifikace uživatelem vybraných mobilních aplikací. V rámci práce byla zkoumána možnost multiplatformního vývoje (pro Android i iOS). Avšak kvůli API omezení na straně iOS byla mobilní aplikace, jež je součástí daného systému, vyvíjena pouze pro Android. Nicméně architektura systému byla navržena tak, aby bylo možné v budoucnu začlenit verzi mobilní aplikace i pro iOS, případně další platformy.

Výsledkem této práce je tedy funkční systém, který umožňuje přenos stávajících i nových událostí (nová SMS, hovor, notifikace) z mobilního zařízení se systémem Android do desktopového Java klienta, který události zobrazuje uživateli. Systém je složen ze tří částí - mobilní aplikace pro Android, Node.js serveru a desktopového Java klienta.

Mobilní aplikace přeposílá jen uživatelem zvolené typy událostí a notifikace aplikací. Dále umožňuje uživateli vymezit typy sítí (Wi-Fi, mobilní data, mobilní data v zahraničí), které smí používat pro odesílání dat. Data jsou odesílána pouze pokud je spuštěný patřičný Java klient, který je bude přijímat. Odesílaní dat je v mobilní aplikaci možné vypnout úplně bez nutnosti odinstalování aplikace.

Node.js server slouží jako prostředník pro komunikaci různých „dvojic zařízení“: mobilní aplikace - Java klient. Spojení je šifrované a samotná data jsou chráněna end-to-end šifrováním (AES algoritmem). Na serveru se neregistrují žádné uživatelské účty a neukládají se žádná přeposílaná uživatelská data. Dvojice zařízení se identifikují generovaným klíčem a volitelným názvem mobilního zařízení.

Mobilní aplikace byla v průběhu celého vývoje testována na dvou virtu-

álních a dvou reálných zařízeních. Server byl nasazen a provozován několik měsíců v Google Cloud (Google Compute Engine).

Podařilo se vytvořit funkční a bezpečný systém, který byl otestován na celé řadě testovacích scénářů a následně byl bez problémů používán několik týdnů. Součástí práce je také několik návrhů možných rozšíření.

Seznam zkratek

AES	Advanced Encryption Standard
API	Application Programming Interface
CSS	Cascading Style Sheets
DTO	Data Transfer Object
GPS	Global Positioning System
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
IP	Internet Protocol
NPM	Node package manager
PaaS	Platform as a Service
REST	Representational State Transfer
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SMS	Short Message Service
SSL	Secure Sockets Layer
UC	Use Case
UI	User Interface
URL	Uniform Resource Locator
XML	eXtensible Markup Language

Seznam obrázků

Diagramy na obrázcích 4.1 až 4.5, 6.1, 6.2, 6.4, 6.6, 6.8 a 6.9 byly vytvořeny prostřednictvím online nástroje `https://draw.io`.

Seznam tabulek

Seznam ukázek kódu

Literatura

[1] *Pushbullet - Your devices working better together*, [online]. 2018 [cit. 18.5.2018]. Dostupné z: https://www.pushbullet.com.

[2] *Získat Dell Mobile Connect – Microsoft Store v: cs-CZ*, [online]. 31.12.2017 [cit. 18.5.2018]. Dostupné z: https://www.microsoft.com/cs-cz/store/p/dell-mobile-connect/9nx51w9gbs5t?rtc=1.

[3] *FlexiSPY^{TM} Unique Monitoring Software For Mobiles & Computers*, [online]. 2018 [cit. 18.5.2018]. Dostupné z: https://www.flexispy.com.

[4] *GitHub - xamarin/XamarinComponents: Plugins for Xamarin*, [online]. září 2017 [cit. 5.10.2017]. Dostupné z: https://github.com/xamarin/XamarinComponents.

[5] *NotificationListenerService | Android Developers*, [online]. 24.5.2018 [cit. 25.5.2018]. Dostupné z: https://developer.android.com/reference/android/service/notification/NotificationListenerService.

[6] *Does iOS have a 'Notification Listener'? - Stack Overflow*, [online]. 27.2.2014 [cit. 11.10.2017]. Dostupné z: https://stackoverflow.com/a/22073980.

[7] *Firebase Cloud Messaging | Firebase*, [online]. 2017 [cit. 23.10.2017]. Dostupné z: https://firebase.google.com/docs/cloud-messaging/.

[8] *Firebase Cloud Messaging XMPP Protocol | Firebase*, [online]. 2017 [cit. 23.10.2017]. Dostupné z: `https://firebase.google.com/docs/cloud-messaging/xmpp-server-ref`.

[9] *WebSockets API - Web APIs | MDN*, [online]. 29.5.2018 [cit. 30.5.2018]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API`.

[10] *Distribution dashboard | Android Developers*, [online]. 8.5.2018 [cit. 25.5.2018]. Dostupné z: `https://developer.android.com/about/dashboards/`.

[11] A. Grant, *Android 4 - Průvodce programováním mobilních aplikací*. Brno: Computer Press, 1. ed., 2013. ISBN: 978-80-251-3782-6.

[12] *Introduction to Activities | Android Developers*, [online]. 17.4.2018 [cit. 29.5.2018]. Dostupné z: `https://developer.android.com/guide/components/activities/intro-activities`.

[13] *Create a navigation drawer | Android Developers*, [online]. 23.5.2018 [cit. 29.5.2018]. Dostupné z: `https://developer.android.com/training/implementing-navigation/nav-drawer`.

[14] *Fragments | Android Developers*, [online]. 8.5.2018 [cit. 29.5.2018]. Dostupné z: `https://developer.android.com/guide/components/fragments`.

[15] *Broadcasts overview | Android Developers*, [online]. 8.5.2018 [cit. 28.5.2018]. Dostupné z: `https://developer.android.com/guide/components/broadcasts`.

[16] J. Ley, *Detecting & sending SMS on Android - AndroidPub*, [online]. 1.7.2017 [cit. 28.5.2018]. Dostupné z: `https://android.jlelse.eu/detecting-sending-sms-on-android-8a154562597f`.

[17] P. Gotecha, *Detecting Incoming Phone Calls In Android*, [online]. 7.5.2017 [cit. 28.5.2018]. Dostupné z: `http://www.theappguruz.com/blog/detecting-incoming-phone-calls-in-android`.

[18] *Intent | Android Developers*, [online]. 8.5.2018 [cit. 28.5.2018]. Dostupné z: https://developer.android.com/reference/android/content/Intent.html#ACTION_NEW_OUTGOING_CALL.

[19] *Settings | Android Developers*, [online]. 23.5.2018 [cit. 29.5.2018]. Dostupné z: https://developer.android.com/guide/topics/ui/settings.

[20] *Data and file storage overview | Android Developers*, [online]. 23.5.2018 [cit. 29.5.2018]. Dostupné z: https://developer.android.com/guide/topics/data/data-storage.

[21] L. Jellema, *Java Web Application sending JSON messages through WebSocket to HTML5 browser application for real time push - AMIS Oracle and Java Blog*, [online]. 14.5.2015 [cit. 30.5.2018]. Dostupné z: https://technology.amis.nl/2015/05/14/java-web-application-sending-json-messages-through-websocket-to-html5-browser-application-for-real-time-push/.

[22] M. Cantelon, M. Harter, T. J. Holowaychuk, and N. Rajlich, *Node.js IN ACTION*. 20 Baldwin Road, PO Box 261, Shleter Island, NY 11964: Manning Publications Co, 2014. ISBN: 9781617290572.

[23] *Node.js*, [online]. 2.1.2018 [cit. 30.5.2018]. Dostupné z: https://nodejs.org/en/.

[24] *GitHub - expressjs/express: Fast, unopinionated, minimalist web framework for node.*, [online]. 12.5.2018 [cit. 30.5.2018]. Dostupné z: https://github.com/expressjs/express.

[25] *GitHub - socketio/socket.io: Realtime application framework (Node.JS server)*, [online]. 17.5.2018 [cit. 30.5.2018]. Dostupné z: https://github.com/socketio/socket.io.

[26] *Using the Notifications API - Web APIs | MDN*, [online]. 17.5.2018 [cit. 30.5.2018]. Dostupné z: https://developer.mozilla.org/en-

US/docs/Web/API/Notifications_API/Using_the_Notifications_API#Browser_compatibility.

[27] J. Vos, S. Chin, W. Gao, J. Weaver, and D. Iverson, *Pro JavaFX 9 - A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients*. Apress, 4. ed., 2018. ISBN: 978-1-4842-3041-1.

[28] *Send Messages | Firebase*, [online]. 22.5.2018 [cit. 4.6.2018]. Dostupné z: https://firebase.google.com/docs/cloud-messaging/admin/send-messages.

Přílohy

A Instalace

A.1 Mobilní aplikace

Minimální požadavky
Mobilní aplikace je určena pro mobilní zařízení s Androidem verze 4.3 (Jelly Bean) a vyšší.

Požadovaná oprávnění
Na Androidu 6 (Marshmallow) a novějším bude uživatel požádán o některé z následujících oprávnění, až když bude potřeba (při prvním zapnutí některého z upozornění). Na starších Androidech budou tato oprávnění vyžadována již při zahájení instalace. Požadovaná oprávnění jsou tato:

- Povolit přístup k SMS zpráv, aby mohla aplikace přeposílat nepřečtené nebo nově přijaté SMS zprávy včetně jejich obsahu.
- Povolit přístup k telefoním hovorům, aby mohla aplikace přeposílat zmeškané hovory nebo upozorňovat na příchozí atd..
- Povolit přístup k telefoním kontaktům pro získání jména volajícího nebo odesilatele zprávy.
- Povolit přístup k notifikacím, aby mohla aplikace přeposílat notifikace ostatních aplikací.

Instalace

1. Z přiloženého cd zkopírujte do mobilního zařízení soubor `cz.zelenikr.remotetouch.apk` nebo jej stáhněte z webové stránky `http://home.zcu.cz/~zelenikr/android/remote-touch/`.

2. Zahajte v telefonu instalaci otevřením souboru. Ujistěte se, že máte v nastavení telefonu povolenou instalaci aplikací z neznámých zdrojů.

A.2 Počítačový klient

Minimální požadavky

Pro běh počítačového klienta je nutné mít nainstalovanou Javu SE 9 nebo vyšší.

Instalace

Počítačového klienta není potřeba instalovat. Stačí rozbalit přiložený archiv `RemoteTouch.zip`. Archiv můžete také stáhnout z webové stránky `http://home.zcu.cz/~zelenikr/android/remote-touch/`, stejně jako mobilní aplikaci.

A.3 Server

Minimální požadavky

Pro běh serveru je nutné mít nainstalovaný Node.js, který lze stáhnout z `https://nodejs.org/`. Doporučená verze je 8.9.4 LTS nebo vyšší.

Instalace

1. Zkopírujte přiložený archiv `server.zip` do počítače a rozbalte jej.
2. V adresáři se souborem `package.json` spusťte příkazovou konzoli a zadejte příkaz `npm install -production`. Tím se nainstalují potřebné závislosti (moduly) serveru.
3. Nastavení FCM

(a) Nejprve si vytvořte Firebase project na adrese `https://console.firebase.google.com`, pokud jej ještě nemáte, kliknutím na Add project.

(b) Otevřete Settings/Service accounts.

(c) Klikněte na tlačítko Generate New Private Key na záložce Firebase Admin SDK. Tím si vygenerujete a stáhnete JSON soubor obsahující pověření k Vašemu účtu.

(d) V adresáři s rozbalenými soubory serveru vytvořte adresář `firebase_admin` a přesuňte do něj stažený JSON soubor. Zajistěte, aby se soubor jmenoval `serviceAccountKey.json`.

(e) V souboru `api/fcm.js` změňte hodnotu proměnné *GCLOUD_PROJECT* na ID svého Firebase projektu:

```
// The ID of the Google Cloud project associated with the
    App.
exports.GCLOUD_PROJECT = 'remotetouch-75ea3';
```

a také aktualizujte hodnotu *databaseURL*:

```
// Set Firebase admin
firebaseAdmin.initializeApp({
    credential:
        firebaseAdmin.credential.cert(firebaseServiceAccount),
    databaseURL:
        "https://remotetouch-75ea3.firebaseio.com"
});
```

(f) Oficiální návod najdete na adrese `https://firebase.google.com/docs/admin/setup`.

4. Pro použití vlastního SSL certifikátu přímo v serveru je nutné učinit následující úpravy v souboru `app.js`:

(a) Zakomentujte řádek s kódem:

```
const server = require('http').createServer(restServer);
```

(b) Odkomentujte následující řádky:

```
const fs = require('fs');
const options = {
  key: fs.readFileSync('./cert/file.pem'),
  cert: fs.readFileSync('./cert/file.crt')
};
const server = require('https').createServer(options,
    restServer);
```

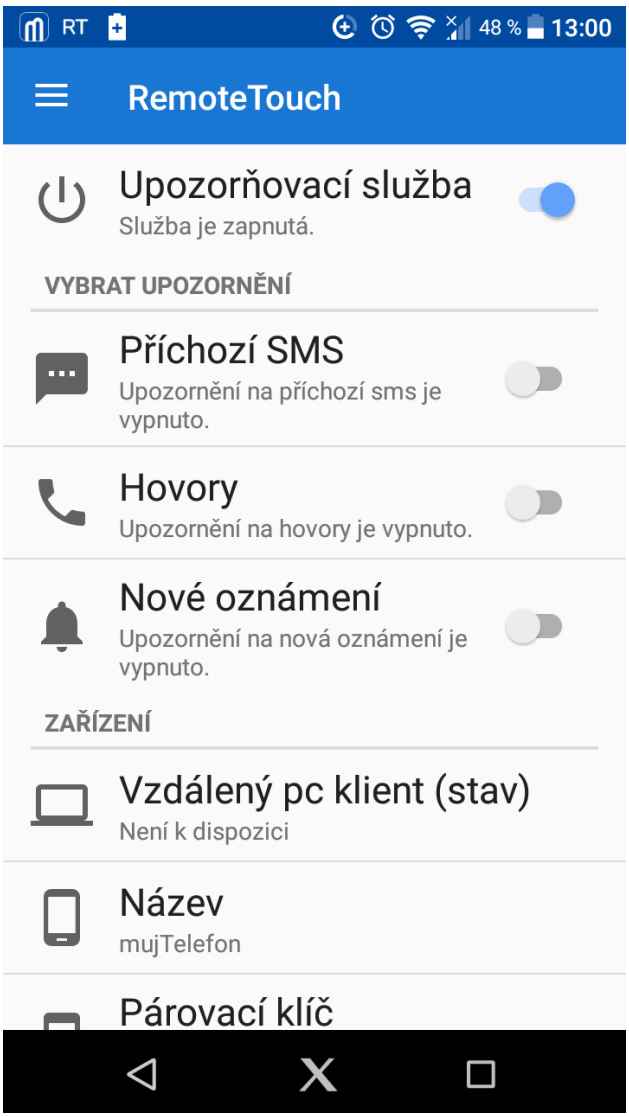
(c) Na závěr vytvořte adresář `cert` a vložte do něj soubory s klíčem a certifikátem pojmenované `file.pem` a `file.crt`.

B Uživatelská příručka

B.1 Mobilní aplikace

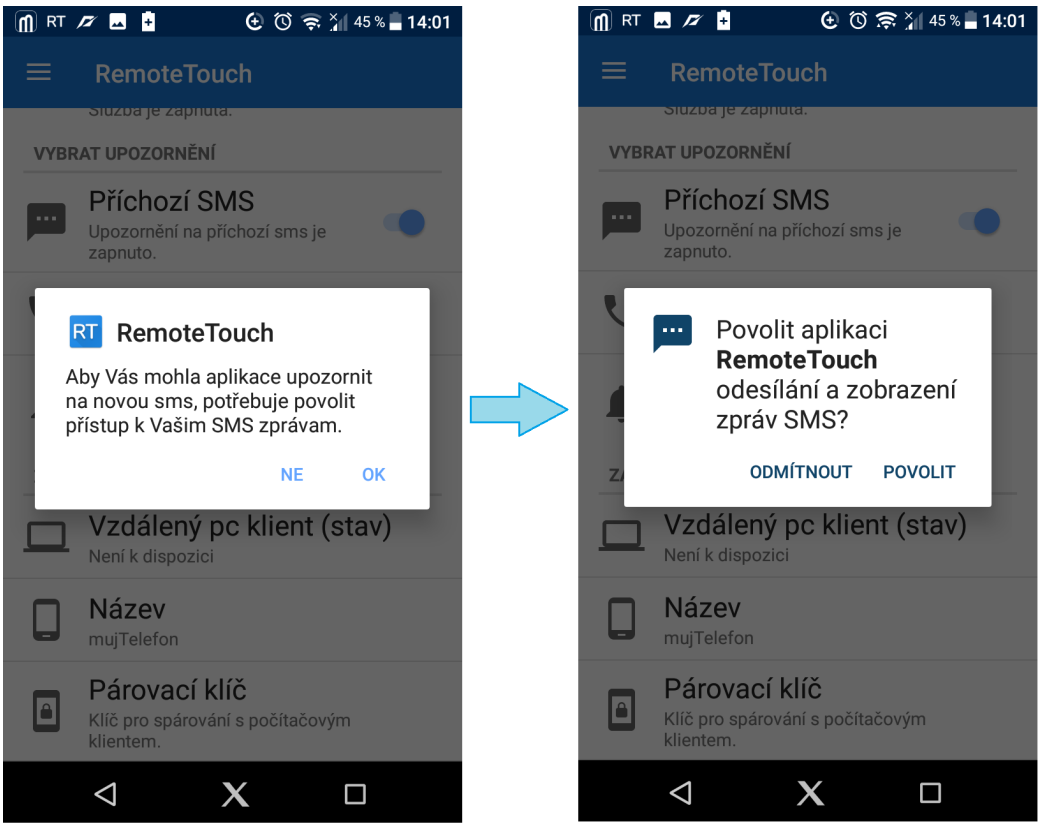
B.1.1 Základní nastavení a ovládání

Po nainstalování aplikace je nutné provést spárování s počítačovým klientem opsáním názvu zařízení (uživatelem pojmenované mobilní zařízení) a párovacího klíče (vygenerován automaticky). Tyto údaje si lze zobrazit a také měnit na hlavní obrazovce otevřené mobilní aplikace. Předvyplněný název zařízení si může uživatel kdykoli změnit. Musí ho poté ale upravit i v počítačovém klientovi. Název zařízení a párovací klíč se v obou aplikacích musí shodovat, jinak nebude uživatel na žádné události upozorňován.



Obrázek B.1: Hlavní obrazovka (Ovládání) aplikace po nainstalování.

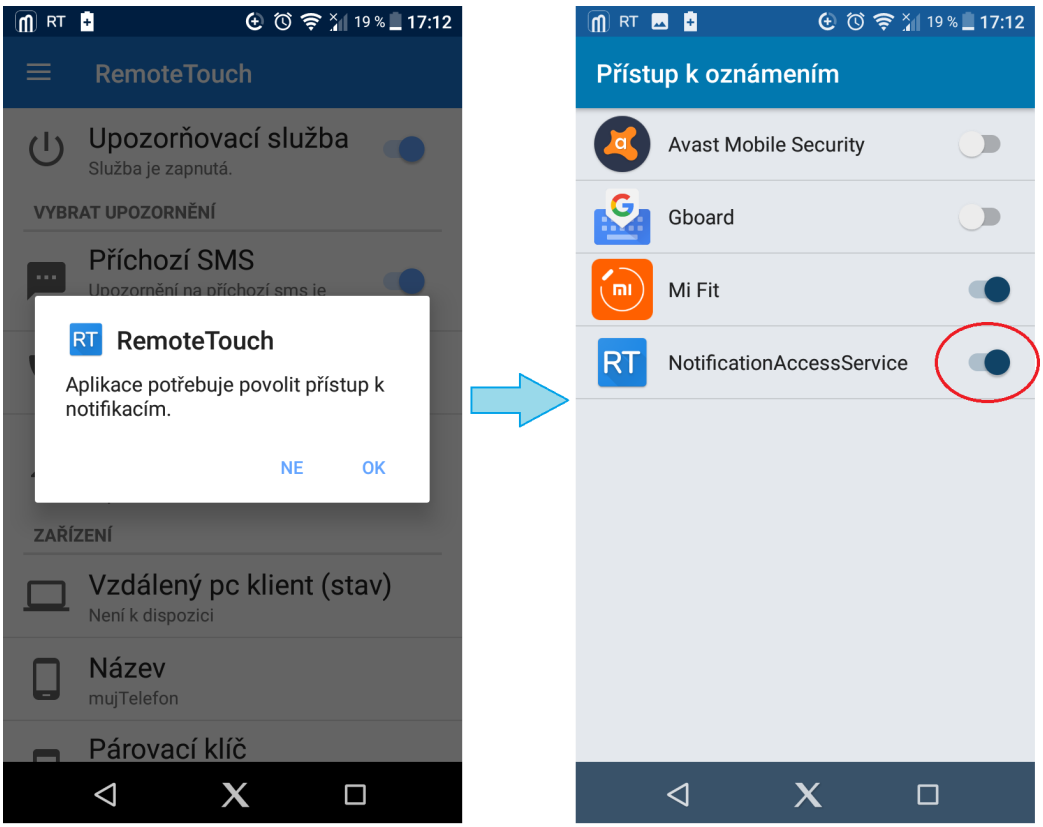
Po spárování si může uživatel jednoduše zvolit, na jaké události chce být v počítačovém klientovi upozorňován (tedy co se má přeposílat) zapnutím nebo vypnutím jednotlivých typů upozornění na hlavní obrazovce. Může také celou službu (všechna upozornění) vypnout. Pokud je služba vypnutá, nebude nic přeposílat i když budou zapnuté jednotlivé typy upozornění.



Obrázek B.2: Zapnutí upozornění na SMS. Na Androidu Marshmallow a novějším je uživatel vždy informován o tom, proč aplikace požaduje dané oprávnění.

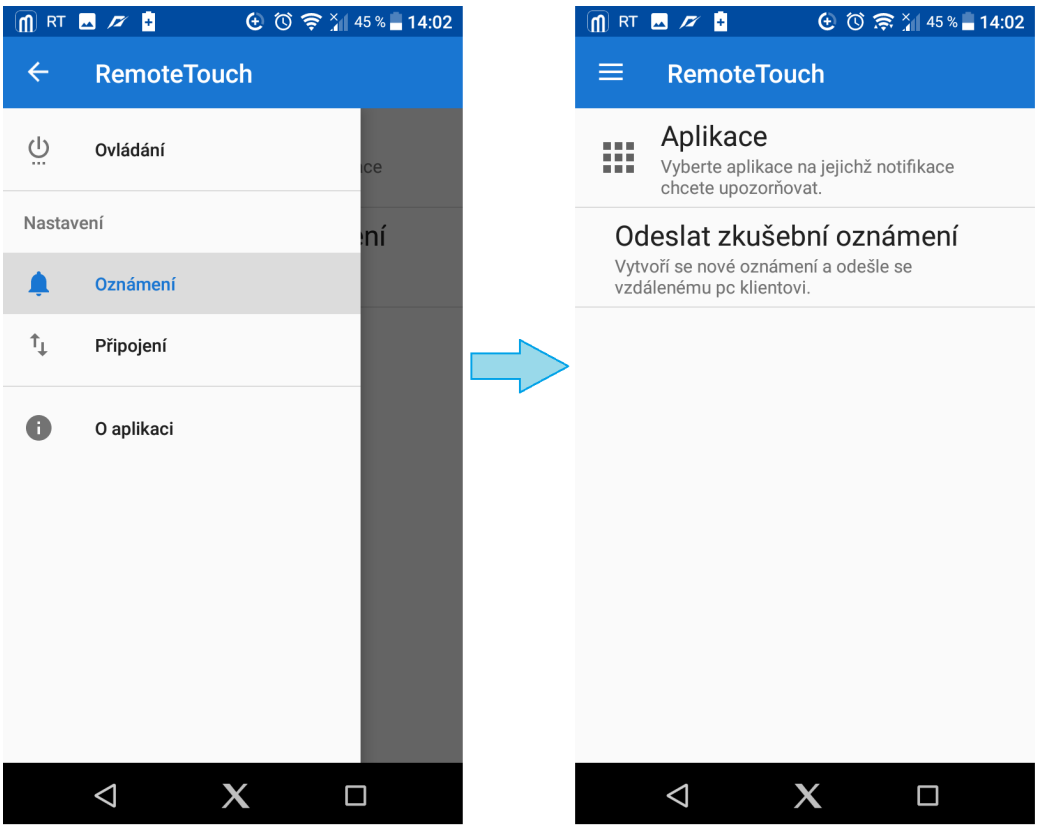
B.1.2 Nastavení oznámení

Aby se oznámení (notifikace) přeposílala, musí být toto upozornění zapnuté. Po zapnutí je uživatel požádán o umožnění přístupu k notifikacím, pokud tak ještě neučinil nebo pokud tento přístup aplikaci odebral. Bez něj nebude upozornění na oznámení fungovat.



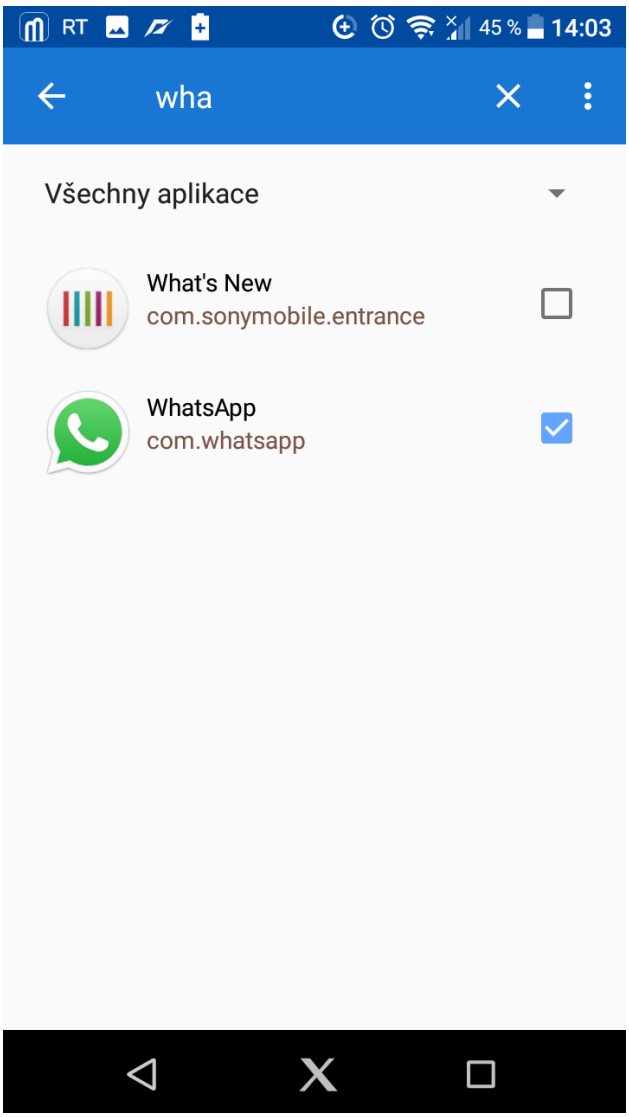
Obrázek B.3: Povolení přístupu aplikace k notifikacím ostatních aplikací.

Přepnutím na obrazovku Oznámení si uživatel může vybrat aplikace, jejichž notifikace (oznámení) má aplikace přeposílat či odeslat zkušební oznámení, kterým ověří spojení s počítačovým klientem.



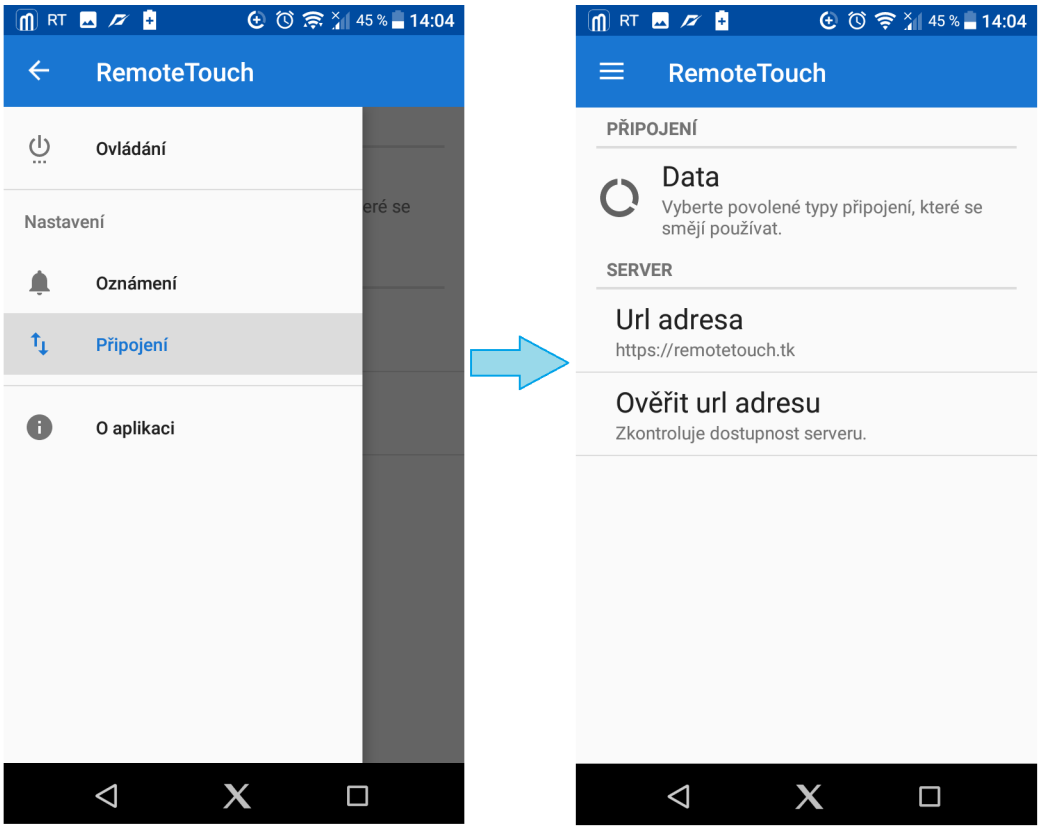
Obrázek B.4: Otevření nastavení Oznámení.

Klepnutím na Aplikace se otevře zaškrtávací seznam s nainstalovanými aplikacemi. V seznamu lze vyhledávat podle názvu aplikace či názvu balíčku (např. cz.zelenikr.remotetouch). Uživatel si také může odfiltrovat a zobrazit pouze vybrané aplikace.



Obrázek B.5: Výběr aplikací, jejichž notifikace se budou přeposílat.

B.1.3 Nastavení připojení



Obrázek B.6: Otevření nastavení připojení.

Ve výchozím nastavení smí aplikace používat pro připojení k internetu Wi-Fi i mobilní data. Klepnutím na Data si může uživatel zvolit, typy sítí, jež může aplikace používat:

- mobilní data (domácí síť)
- mobilní data v zahraničí (roaming)
- Wi-Fi

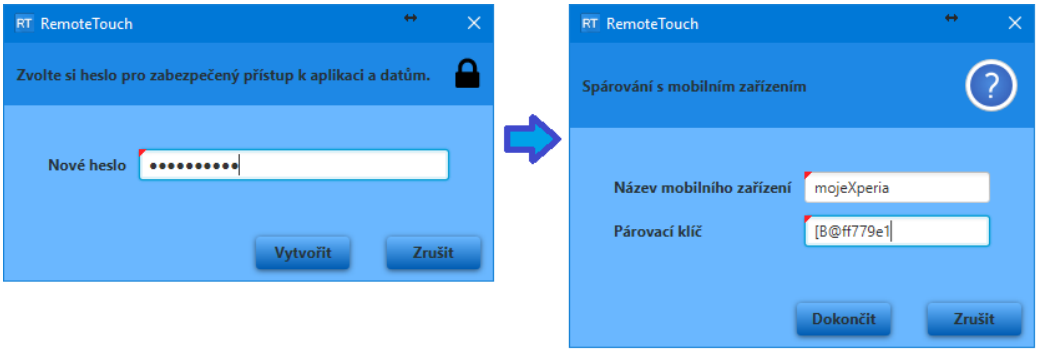
Položka Url adresa slouží ke změně internetové adresy komunikačního serveru. Tato možnost je určena především technicky zkušenějším uživatelům, kteří chtějí komunikovat přes vlastní server. Korektnost a dostupnost si může uživatel vyzkoušet klepnutím na Ověřit url adresu.

B.2 Počítačový klient

B.2.1 Spuštění

Klient se spouští dvojklikem na .jar soubor.

Při prvním spuštění je uživatel vyzván k vyplnění vlastního hesla, kterým bude chráněn přístup ke počítačovému klientovi při jeho dalším spuštění. Heslo musí být dlouhé alespoň 5 znaků. Druhým krokem při prvním spuštění je spárování s mobilním zařízením. To znamená opsat z mobilní aplikace název zařízení a párovací klíč.



Obrázek B.7: Prvotní nastavení počítačového klienta

Správnost nastavení si může uživatel ověřit odesláním zkušebního oznámení z mobilní aplikace.

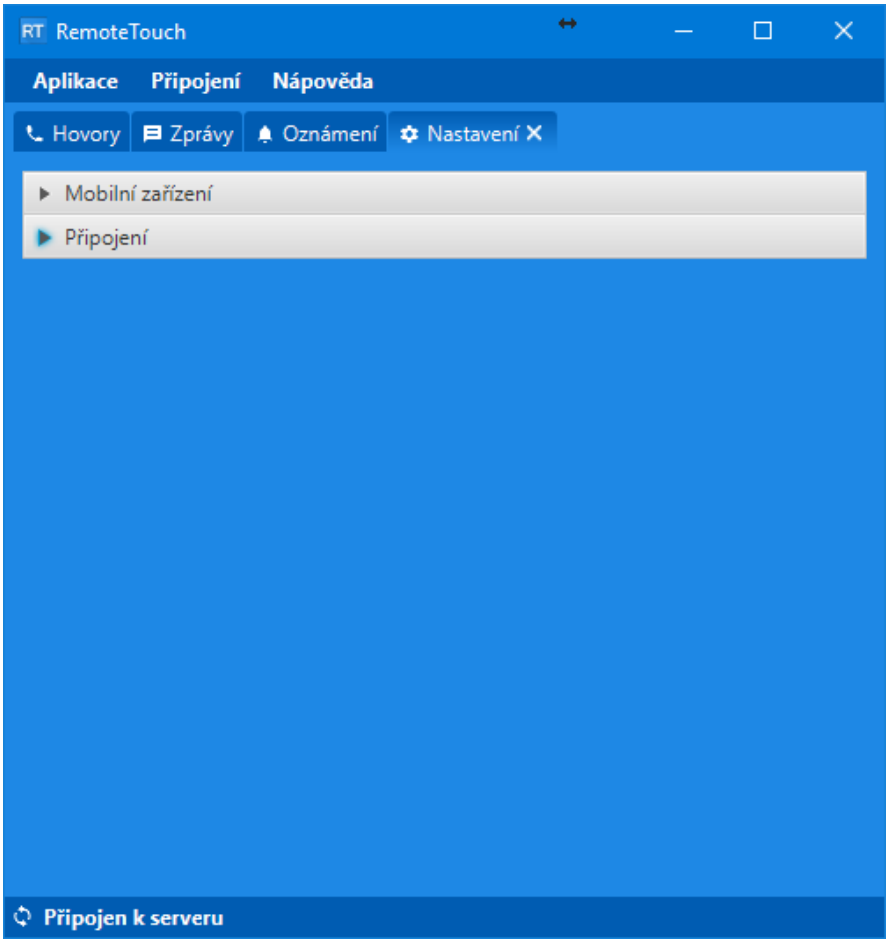
Při dalším spuštění klienta jej musí uživatel tzv. odemknout. To znamená, že

musí zadat heslo, které si zvolil při prvním spuštění. Uživatel má neomezený počet pokusů. Pokud si uživatel nemůže na své heslo vzpomenout, může jej resetovat. Pak se klient chová jako při prvním spuštění.

Pokud je upozorňovací služba v telefonu zapnutá a má přístup k internetu, přijde klientovi po připojení k serveru upozornění, že „RemoteTouch Běží...“. Zároveň na základě nastavení mobilní aplikace obdrží klient všechny dosud nepřečtené SMS zprávy, zmeškané hovory a existující notifikace všech uživatelem vybraných aplikací.

B.2.2 Změna nastavení

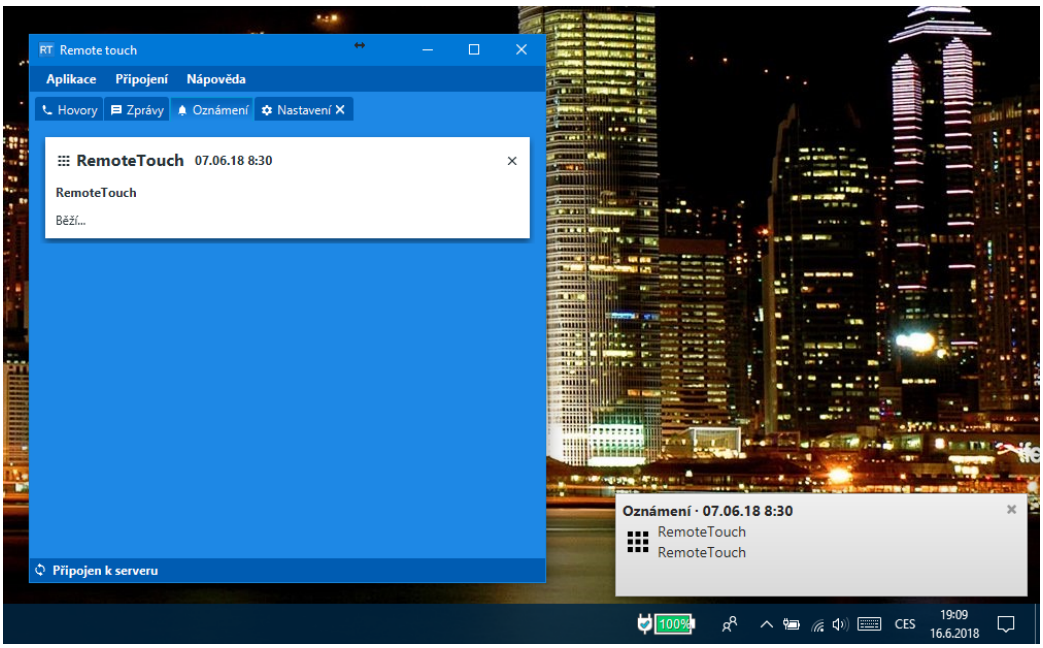
V záložce nastavení může uživatel měnit název mobilního zařízení, párovací klíč nebo adresu serveru (používá-li jiný než je přednastaven). V rámci testování a zkoušení celého řešení byl server nasazen v cloudu a jeho veřejná url adresa je v klientovi předvyplněna. Po změně jedné nebo více hodnot je nutné restartovat spojení se serverem, čímž se použije nové nastavení.



Obrázek B.8: Nastavení klienta je rozděleno podle kategorií.

B.2.3 Práce s událostmi

Uživatel je o nové události (SMS, hovor, notifikace) informován zobrazením notifikace v pravém dolním rohu obrazovky. Notifikace se zobrazí i když je klient minimalizován. Kliknutím na notifikaci se klient otevře a zobrazí obsah přijaté události v patřičné záložce (Hovory, Zprávy nebo Oznámení). Notifikace nikdy nezobrazuje přímo svůj obsah. Lze z ní vyčíst pouze o jakou událost jde a kdy nastala.



Obrázek B.9: Náhled klienta a notifikace s oznámením aplikace RemoteTouch

Události je možné z každé záložky postupně odebrat kliknutím na křížek v pravém horním rohu dané události. Odebrání události z tohoto seznamu nijak neovlivní obsah mobilního telefonu.

B.3 Server

Server nemá uživatelské rozhraní. Poskytuje pouze informativní výpisy do konzole. Pro spuštění serveru, zadejte do příkazové konzole v adresáři se souborem `app.js` příkaz `npm run start`.

C Obsah přiloženého CD

V kořenové složce dodaného disku je 5 adresářů:

android/ Obsahuje Android Studio projekt vytvořené mobilní aplikace, vygenerovaný JavaDoc a instalační .apk soubor.

pc/ Obsahuje Maven projekt vytvořeného Java desktop klienta, vygenerovaný JavaDoc a archiv se spustitelným .jar souborem a potřebnými knihovnami.

server/ Obsahuje návod, jak nasadit server v Google Cloud Compute Engine (+ zdarma registrace domény a získání SSL certifikátu). Dále .zip archiv se zdrojovými kódy vytvořeného Node.js serveru a package.json souborem s deklarovanými závislostmi.

text/ Obsahuje LaTeX zdrojový kód této diplomové práce a její vysázenou .pdf verzi.

poster/ Obsahuje .pub poster a jeho .pdf verzi.