VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE TYPU KLIENT-SERVER

MOBILE CLIENT SERVER APPLICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. ROMAN MANĎÁK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MICHAL KULA**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/18977/2016/xmanda03

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Manďák Roman, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Mobilní aplikace typu klient-server**
Mobile Client Server Application
Kategorie: Uživatelská rozhraní

Pokyny:

1. Prostudujte dostupné komunikačními protokoly umožňující zasílání zpráv mezi klienty v aplikacích typu klient-server.
2. Seznamte se s možnostmi vývoje aplikací na mobilní platformě iOS.
3. Navrhněte způsob implementace klientské a serverové části aplikace typu klient-server umožňující zasílání textových a multimediálních zpráv mezi klienty.
4. Implementujte navrženou aplikaci.
5. Diskutujte dosažené výsledky a možné pokračování práce.

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- body 1 až 3.

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

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

Vedoucí: **Kula Michal, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Diplomová práce se zabývá návrhem a implementací klientské i serverové části aplikace pro zasílání textových a multimediálních zpráv mezi uživateli. Návrh respektuje znovupoužití komponent pro jejich jednoduché zařazení do jiné aplikace. Součástí práce je detailní rozbor komunikačních protokolů využívaných pro tento typ aplikace, popis možností platformy iOS a vývoj na této platformě. Na závěr je shrnuta implementace a její možná rozšíření.

Abstract

This master thesis is focused on designing the client and the server part of an application for text and multimedia communication between its users. The design takes into account reusing of the components and making them easily embeddable into another application. The project consists of a detailed analysis of communication protocols used for this type of applications, a description of possibilities of the iOS platform and development on this platform. Finally, the implementation and its possible extensions are described.

Klíčová slova

Instant messaging, Klient-server, iOS, MQTT

Keywords

Instant messaging, Client-server, iOS, MQTT

Citace

MANĎÁK, Roman. *Mobilní aplikace typu klient-server*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Kula Michal.

Mobilní aplikace typu klient-server

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Michala Kuly. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Roman Manďák
24. května 2017

Poděkování

Tímto bych rád poděkoval mému vedoucímu diplomové práce, panu Ing. Michalu Kulovi za poskytnutou podporu a odborné vedení.

Obsah

Kapitola 1

Úvod

V posledních letech je velmi populární online komunikace. Služby nabízející tuto komunikaci jsou dnes již dostupné, jak na různých mobilních platformách, tak i přímo ve webovém prohlížeči. Uživatelé tak v reálném čase mohou posílat zprávy, obrázky a další multimediální obsah. Tento trend téměř nahradil v neformální komunikaci emaily a dopisy. Tyto moderní služby poskytují velké organizace jako nejznámější Facebook, Viber a Whatsapp. Jsou vyvíjeny společnostmi dlouhou dobu a vylepšovány atraktivními prvky pro uživatele. Na trhu je jen několik aplikací nabízejících tyto služby a pro jejich velkou popularitu nejsou nové aplikace žádoucí. To má za následek velmi malé množství kvalitních zdrojových kódů k začlenění takovéto služby do vlastní aplikace.

Cílem této práce je prozkoumat dostupné komunikačními protokoly umožňující zasílání zpráv v aplikacích typu klient-server a seznámit se s možnostmi vývoje aplikací na platformě iOS. Ze získaných znalostí navrhnout realizaci klientské a serverové části aplikace pro zasílání textových a multimediálních zpráv mezi uživateli. Výsledný návrh poté obecně realizovat pro další možné využití.

Práce je rozdělena do čtyř navazujících kapitol. Ve 2. kapitole je přiblížen význam instant messagingu a jsou teoreticky popsány komunikační protokoly využívající se pro online komunikaci. Kapitola 3 se zabývá platformou iOS, popisuje její architekturu, využití push notifikací, balíčkového systému CocoaPods a distribucí aplikace. Ve 4. kapitole jsou definovány uživatelské akce, struktura systému a detailněji vysvětlen návrh serverové a klientské části. V poslední kapitole 5 je popsána realizace serverových částí a iOS aplikace.

Kapitola 2

Protokoly pro instant messaging

Komunikace v reálném čase je dnes velmi populární. Nabízí uživateli možnost sledovat dostupnost ostatních účastníků komunikace, stav odeslaných zpráv nebo odesílat multimediální obsah.

Služba instant messaging a její význam se postupem času trochu mění. V původním znění „Instant messaging (dále jen IM) je typ komunikační služby, která umožňuje výměnu zpráv s jinými uživateli v reálném čase a poskytuje informaci o jejich dostupnosti“.

Některé definice obsahují spojení „dostupná na internetu“. Toto tvrzení je na zvážení, protože IM lze provozovat i na lokální síti nepřipojené k internetu. S rostoucími požadavky nyní IM není jen doménou textové komunikace, ale i multimédií jako jsou obrázky, videa, hlasové záznamy apod.

V následujících podkapitolách budou blíže definovány vhodné standardy a protokoly pro zasílání dat.

2.1 MQTT

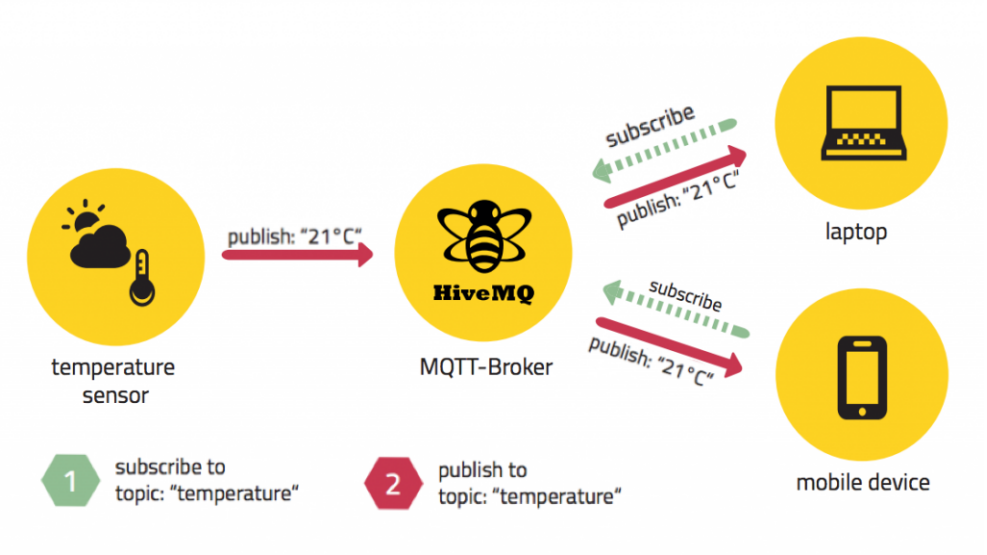
Jedná se o komunikační protokol Message Queuing Telemetry Transport [11] vyvinutý v roce 1999 společností IBM. Je určen pro méně výkonná zařízení nebo zařízení v omezených prostorech, kde je malá šířka pásma nebo vysoká odezva sítě. Hlavní výhodou je jednoduchost a snadná implementace. MQTT byl v roce 2014 oficiálně schválen jako OASIS[1] standard.

MQTT využívá protokol TCP/IP k připojení k internetu a je založen na bázi publish subscribe, kde publisher publikuje zprávu na určité téma (topic). Odběratel (subscriber) se přihlásí k odběru určitého tématu a následně všechny publikované zprávy na toto téma odebírá. Za přijímání a rozesílání zpráv je zodpovědný tzv. broker. Ten také zajišťuje autentizaci a autorizaci odběratelů, poskytuje zabezpečení dat pomocí SSL [1] a TLS [20] a při složitějších strukturách síťe komunikuje s ostatními brokery.

Základní princip MQTT můžeme vidět na obrázku 2.1 . Ve srovnání s klasickou point-to-point komunikací, zde publisher nemusí mít žádné informace o příjemcích zpráv.

Téma (topic) je hierarchická struktura levelů oddělených separátory. Specifikuje odběr zpráv k určitému tématu, nebo lze využít zástupné znaky $+$ a $\#$ pro odebírání více souvisejících témat. Příklad struktury lze vidět na obrázku 2.2. Jednotlivé úrovně jsou popsány ASCII řetězci rozlišujícími malá a velká písmena. Pro prevenci případných komplikací s kódováním je doporučeno využívat standardní UTF-8.

[1] OASIS - Advancing open standards for the information society viz. https://www.oasis-open.org



Obrázek 2.1: Ukázka principu publikování zpráv (Převzato z [10])

Obecně názvy témat nejsou nijak limitovány, existuje však výjimka. Témata začínající znakem $ například $SYS jsou rezervována pro interní statistiky brokeru. Nachází se zde například informace o počtu připojených zařízení, počet zaslaných zpráv apod.

Standard definuje tři úrovně spolehlivosti doručování zpráv (Qualitiy of Service - QoS). QoS je hlavním rysem MQTT, dělá komunikaci na nespolehlivé síti o mnoho jednodušší a klient může zvolit její úroveň na základě aplikační logiky nebo spolehlivosti vlastní sítě. Nastavení QoS lze rozdělit na dvě části, od publikujícího k brokeru a z brokeru k příjemci. Nastavení směrem k brokeru probíhá v každé publikované zprávě, zatímco pro přijímání zpráv, stanoví QoS klient při přihlašování k odběru na daný topic. Spolehlivost doručení na jednotlivých úrovních:

- **QoS 0 (nejvýše jednou)** – Při nejnižší úrovni QoS není přijatá zpráva potvrzována příjemcem a ani odesílatel zpráv ji nijak neukládá. Toto chování se často nazývá „odešli a zapomeň“. Zvolení této úrovně QoS se doporučuje při stabilním kabelovém spojení nebo nezáleží-li na ztrátě několika zpráv.

- **QoS 1 (alespoň jednou)** – Při použití první úrovně QoS je zaručeno doručení zprávy nejméně jednou. To také ale znamená, že může být doručena vícekrát. Příjemce pro potvrzení přijetí paketu odesílá potvrzovací paket PUBACK (publish acknowledgement) obsahující identifikátor přijatého paketu. Neobdrží-li odesílatel ve stanoveném čase potvrzovací paket, odesílá paket znovu. Ztratí-li se potvrzovací paket, dojde k situaci, kdy příjemce obdrží zprávu vícekrát. O této situaci je možné informovat nastavením duplikačního flagu (DUP). Nejčastější použití QoS je právě na této úrovni, protože zaručuje obdržení každé zprávy. Aplikace se však musí umět vypořádat s duplicitou zpráv.

- **QoS 2 (právě jednou)** – Nejvyšší úroveň zajišťuje obdržení zprávy právě jednou. To s sebou nese samozřejmě větší režii. Po odeslání zprávy odesílatel čeká na potvrzení stejně jako u QoS 1. Po přijetí potvrzovacího paketu PUBREC (publish receive), vyřadí stav znovupublikování zprávy a odešle paket PUBREL (publish release). Pří-

jemce po přijetí paketu PUBREL oznámí odesílateli kompletní přenos zprávy paketem PUBCOMP (publish complete). Vždy, když dojde ke ztrátě paketu, je odesílatel zodpovědný za znovuodeslání posledního paketu. Není-li schopen klient ošetřit situaci při duplikování zpráv, vybere tuto úroveň QoS.

Obrázek 2.2: Ukázka hierarchie témat (Převzato z [10])

Další možností, kterou MQTT protokol nabízí, je tzv. retain mód zprávy. Jedná se o normální MQTT zprávu, které je při publikování nastaven příznak retain módu. Tato zpráva je uchovávána brokerem v paměti a uživateli nově přihlášenému k odběru zpráv na dané téma je ihned po přihlášení odeslána. Typ takovéto zprávy může být pro každé téma uložen pouze jeden, opakovanou publikací je tak vždy poslední uložená zpráva nahrazena novější. Zrušit zasílání této uložené zprávy lze jednoduše, stačí publikovat tento typ zprávy s prázdným obsahem uživatelských dat. Takovéto chování může být vhodné v situacích, kdy nově přihlášený uživatel chce okamžitě přijímat zprávy a nečekat až na další publikované [10].

Protokol nachází využití například v lékařství pro monitorovácí zařízení nebo v sociálních sítích (konkrétně aplikaci Facebook messenger). V současné době patří mezi jeden z nejpoužívanějších protokolů v oblasti internetu věcí [21].

2.2 XMPP

Extensible Messaging and Presence Protocol [22] (XMPP - známý také jako Jabber) je otevřený standardizovaný protokol pro zasílaní zpráv, především pak pro textovou komunikaci více uživatelů, ale zároveň také pro přenos videa nebo záznamu zvuku. Byl vyvíjen komunitou Jabber jako open-source, na rozdíl od ostatních uzavřených protokolů instant messagingu. XMPP nabízí některé výhodné vlastnosti oproti ostatním službám:

- **Otevřenost** – XMPP protokol je otevřený a volně šířitelný, existuje mnoho implementací ve formě klientů, serverů, serverových komponent a knihoven.
- **Standardizovanost** – Organizace IETF [2] schválila XMPP jako standard a jeho specifikace byly publikovány v RFC 3920 a RFC 3921 v roce 2004. V roce 2011 byly

[2]Internet Engineering Task Force (IETF) - mezinárodní skupina zabývající se vývojem internetu

revidovány a aktualizovány v RFC 6120 (core [17]), RFC 6121 (rozšíření IM [18]) a RFC 7622 (address format [19]).

- **Decentralizace** – Architektura XMPP sítě je podobná emailu. Je tedy možné si nainstalovat vlastní XMPP server. Uživatel tak získá vlastní kontrolu nad systémem. Je vhodný k využití v rámci interní komunikace v organizaci.
- **Bezpečnost** – Do jádra protokolu je zaneseno zabezpečení SASL[3] a TLS[20], není tak možné číst zprávy třetí osobou a vytvořit útok „man in the middle“. Protokol také umožňuje izolaci XMPP serveru od internetu.
- **Rozšířitelnost** – Nedostačuje-li uživateli funkcionalita, lze ji díky XML charakteru protokolu obohatit o své vlastní funkce s nadále trvající podporou stávajících funkcí. Tato rozšíření jsou zveřejňována v tzv. XEP sériích a je na uživateli, zda své rozšíření publikuje nebo si jej nechá pro své soukromé využití.

Mezi další výhody patří rozšíření pro komunikaci více uživatelů najednou tzv. Multi User Conference (MUC), které je podobné protokolu Internet Relay Chat (IRC). Uživatelé mají k dispozici místnost, ve které probíhá jejich komunikace. Tato komunikace může být veřejně přístupná komukoli, nebo soukromá, kde pro vstup do místnosti je vyžadováno heslo. Ve výchozím nastavení jsou publikované zprávy zasílány všem uživatelům v místnosti. Pro soukromé zprávy dvou uživatelů je vytvořena zvláštní místnost.

Další vlastností protokolu je podpora přihlášení uživatele na více místech najednou. Server poté uživateli zasílá zprávy na připojení s vyšší prioritou. Protokol je založený na principu klient-server s využitím TCP spojení. Každý klient je jednoznačně určen pomocí identifikátoru *Jabber Identificator* (JID), který se skládá z uživatelského jména (přezdívka), názvu serveru a identifikátoru prostředku.

XMPP má v oblasti internetu věcí několik výhod. Je vyvíjen a testován více než 10 let v mnoha programovacích jazycích a za tuto dobu byla vytvořená velká infrastruktura, zejména v oblasti instant messagingu.

2.3 CoAP

Constrained Application Protocol (CoAP) je webový protokol pracující na aplikační vrstvě ISO/OSI definovaný v RFC 7252 [23]. Je primárně určený pro komunikaci machine-to-machine (M2M) a navržený pro jednoduchá nízkoenergetická zařízení. Využívá UDP protokolu. Není zaručena spolehlivost přenosu, protože samotný UDP protokol nezaručuje doručitelnost všech paketů a jejich doručení ve správném pořadí. Realizuje asynchronní výměnu zpráv.

CoAP na jednu stranu stejně jako Hypertext Transfer Protokol (HTTP) používá model dotaz/odpověď zasíláním různých typů žádostí (GET, POST, PUT, DELETE), na druhou stranu na základě lehkého UDP protokolu dovoluje IP multicast pro skupinovou komunikaci. Podporuje content-type hlaviček známých z HTTP, které umožňují výměnu informací pomocí používaných formátů JSON[4] a XML[5]. Tyto podobnosti mezi oběma protokoly umožňují jednoduché mapování na HTTP protokol. Ke snížení režie přispívá binární kódování hlaviček zpráv, metod i stavových kódů.

[3] Simple Authentication and Security Layer (SASL) - framework pro autentizaci a zabezpečení dat v internetových protokolech

[4] JavaScript Object Notation (JSON)

[5] Extensible Markup Language (XML)

Protože je protokol založený na UDP a ne TCP, nemůže využít v zabezpečení kryptografické protokoly jako SSL/TLS. Využívá tedy DTLS [7], který poskytuje stejné zabezpečení jako TLS, ale pro přenos dat skrze UDP. DTLS zamezuje odposlechům, padělání nebo falšování zasílaných zpráv. Odstraňuje případy ztráty paketu nebo jiné pořadí příchozích zpráv. Toho dosahuje vyžadováním odpovědi, přiřazením pořadového čísla paketu a opakovaným přenosem paketů.

CoAP je považován za stále se rozvíjející protokol na rozdíl od MQTT a XMPP, což může vést k problémům při implementaci. Má však potenciál dosáhnout stejné úrovně jako MQTT. Jeho výhodou je použití modelu dotaz/odpověď.

2.4 WebSocket

WebSocket je protokol umožňující obousměrnou komunikaci mezi jednotlivými TCP schránkami použitím jediného TCP spojení. Je standardizovaný ve W3C[6] a definovaný v RFC 6455[8]. WebSockets umožňují webovému prohlížeči posílat informace, aniž by neustále otvíral nové připojení. Protokol využívá porty 80 a 433 v závislosti na tom, zda je povolen TLS.

Tento protokol vznikl jako důsledek potřeby webových aplikací obousměrně komunikovat se serverem bez zbytečného vytváření nových spojení, nahradil tak například nevyhovující metodu polling[7]. Použití HTTP k zobrazování dynamického obsahu na stránkách bylo pouze dočasné řešení, protože HTTP bylo původně navrženo pro zobrazování statického obsahu. WebSockety využívají TCP spojení pro zajištění obousměrné komunikace, tím je možné odstranit HTTP hlavičky, což má za následek mnohem nižší režii a menší využití šířky pásma sítě.

Při navazování spojení tzv. handshake pošle klient HTTP požadavek v hlavičce s položkou *Upgrade: websocket*. Jestliže server podporuje protokol, odpoví status kódem 101 se stejnou položkou v hlavičce. Kód 101 označuje, že server akceptuje žádost a přepíná na websocketové spojení. Poté co klient obdrží tuto odpověď, je spojení navázáno a klient nebo server mohou odesílat data. Jakákoli jiná návratová hodnota než 101 značí, že inicializace spojení neproběhla v pořádku. Data přenášená mezi klientem a serverem mohou být jako UTF-8 text, binární rámce nebo speciální rámce pro obsluhu spojení.

2.5 HTTP/2

Protokol HTTP/2 byl vyvinut pracovní skupinou IETF a je specifikován v RFC 7540 [13]. Byl postaven na základech dnes již zastaralého síťového protokolu SPDY[8], podporovaného společností Google. Je vylepšenou verzí HTTP protokolu. Cílem bylo zrychlit vzájemnou komunikaci mezi klientem a serverem a minimalizovat zátěž obou stran. K tomu je využita komprese hlaviček, pro kterou byl vytvořen nový kompresní algoritmus HPACK, definovaný v RFC 7541 [16].

Komunikace probíhá přes jedno TCP spojení, kterým jsou posílány všechny požadavky klienta paralelně pomocí tzv. proudů (stream). Každý požadavek vytvoří nový proud s unikátním identifikátorem. Po přijetí požadavku server odesílá data, pro ukončení proudu

[6] Word Wide Web Consordium (W3C) viz. https://www.w3.org

[7] polling - metoda založená na pravidelných dotazech na server k získání změn

[8] Pronounced speedy (SPDY) - dnes již zastaralý síťový protokol

(odeslání všech dat) nastaví poslednímu paketu příznak END_STREAM. Jednotlivé identifikátory proudů zakládané serverem jsou sudé, klientem liché. Speciální proud s identifikátorem 0 nese informace k TCP spojení.

Formát zasílané zprávy je na rozdíl od předchozích verzí zvolen binární, který není tolik náchylný k chybám. Oproti textovému formátu není tolik tolerantní k zápisu například čísel, ale zpracování dat je jednodušší. Data jsou přenášena v podobě rámců (frames). Existují různé typy rámců jako rámec pro hlavičky, nastavení parametrů spojení, přenášená data apod.

Výhodou HTTP/2 může být zavedení principu tzv. server push. Server klientovi v rámcích PUSH_PROMISE zasílá data, která dle něj bude klient brzy žádat. Tato data jsou poslána v novém proudu (sudý identifikátor). Nechce-li klient tato data přijímat, ukončí proud paketem RST_STREAM. Tento mechanismus lze úplně vypnout v nastavení spojení.

Zabezpečení je zajišťováno pomocí šifrování komunikace přes TLS, které i když není povinné je většinou klientů(Safari, Chrome apod.) vyžadováno. To v podstatě dělá zabezpečení povinným.

Kapitola 3

iOS

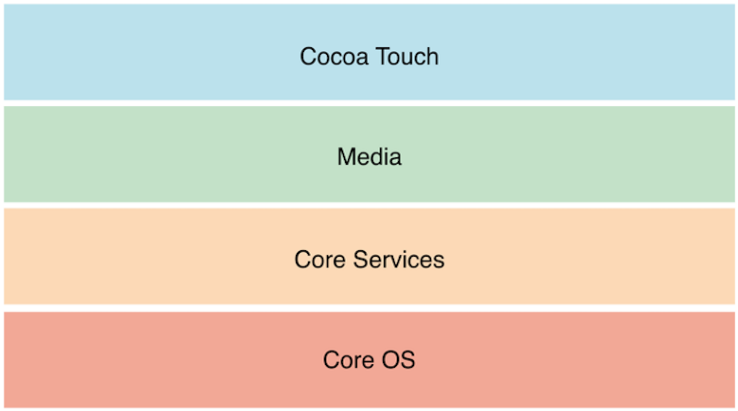
iOS je mobilní operační systém (dále jen OS) vyvinutý společností Apple. Tento název nese od čtvrté verze systému dříve nazývané iPhone OS. Poprvé byl představen v roce 2007 společně s prvním iPhonem, později pak rozšiřován i pro použití na novějších zařízeních společnosti (iPad, iTouch). Je postaven na základech OS pro stolní počítače Mac OS X (nyní MacOS) a sdílí s ním některé frameworky. Není s ním ale plně kompatibilní, protože pro uživatelské rozhraní využívá *Cocoa Touch* místo *Cocoa*. Obsahuje unixové jádro, ale oproti ostatním unix-like systémům není volně dostupný, společnost Apple jej používá výhradně pro své produkty.

iOS obsahuje při nové instalaci některé základní aplikace jako emailový klient, webový prohlížeč apod. Společnost tyto předinstalované aplikace nedovoluje odstranit, až od systému iOS 10 je umožňuje skrývat. Aplikace jsou pouze skryty z důvodu ověření systému.

3.1 Architektura systému

Na nejvyšší úrovni působí iOS jako prostředník mezi aplikací a hardwarem. Aplikace tak nekomunikuje přímo s hardwarem, ale pomocí definovaných rozhraní. Tato rozhraní dopomáhají k jednoduššímu vytvoření aplikace [2].

Na iOS architekturu můžeme pohlížet jako na několik vrstev, kde nižší úrovně slouží k základním funkcím a službám, naopak vyšší vrstvy nabízejí sofistikovanější metody. Schéma vrstev iOS můžeme vidět na obrázku 3.1.



Obrázek 3.1: Architektura systému iOS (Převzato z [2])

Pro vývoj aplikací je doporučeno upřednostňovat aplikační rámce (framework) využívající výše položených vrstev. Ty poskytují objektově-orientovanou abstrakci nižších vrstev a zapouzdřují komplexnější prvky jako sockety nebo vlákna. Lze tak psát přehlednější a kratší kód [2].

Cocoa Touch

Tato vrstva obsahuje klíčové frameworky pro stavbu aplikace, které definují vzhled aplikace. Poskytuje také základní aplikační infrastrukturu a klíčové technologie. Vybrané klíčové aplikační rámce:

- *UIKit* – Jeden z hlavních aplikačních rámců poskytuje důležitou infrastrukturu pro implementaci grafických, událostmi řízených aplikací iOS. Dále podporuje správu aplikace v hlavní běžící smyčce, animuje obsah uživatelského rozhraní, reaguje na uživatelská gesta, multitasking, push notifikace a další. Obsahuje také specifické funkce zařízení pro práci s fotoaparátem, knihovnou fotek, senzory apod.

- *MapKit* – Poskytuje mapy, které můžeme vložit do uživatelského rozhraní a různými gesty posouvat, přibližovat. Obsah map můžeme přizpůsobit svým potřebám a změnit styl zobrazení, vložit specifický bod a jiné. Kromě zobrazení map integruje spolupráci s nativní aplikací map a mapovými servery společnosti Apple pro získání informací o trasách, hromadné dopravě apod.

- *GameKit* – Implementuje podporu pro Game Center, které umožňuje uživatelům sdílet jejich herní informace jako dosažené skóre do online žebříčků nebo vyzvat ostatní k souboji ve hře a další.

Další aplikační rámce dovolují vývojáři interagovat s ostatními aplikacemi pro odeslání sms, uložení události do kalendáře, sdílení obsahu na sociálních sítí nebo vložení reklamy do aplikace.

Cocoa Touch poskytuje také důležité technologie pro sdílení souborů (AirDrops), vkládání dokumentů (DocumentPicker), vytváření widgetů a vlastní klávesnice, sdílení na sociálních sítích (App Extensions), multitasking nebo zahájení práce na jednom zařízení a její dokončení na druhém (Handoff).

Media

Media vrstva obsahuje grafické, audio a video technologie, které umožňují přehrávání multimédia přímo v aplikaci nebo sdílení obrazovky přes Apple TV pomocí technologie AirPlay. Pomáhá také k vytvořením graficky a zvukově propracovaných aplikací.

Grafické uživatelské rozhraní je nejjednodušší vytvářet použitím standardních komponent a nechat systém, aby vše realizoval. Někdy však toto nevyhovuje, nebo to nelze využít. V takové situaci můžeme využít knihovny jako *Core Graphics* a *OpenGl ES*, které dovolují větší kontrolu nad vykreslováním grafického obsahu, vytvoření vlastních vzhledů aplikace a práci s obrázky.

Knihovny zvukových technologií umožňují využívat vibrace zařízení, záznam a přehrávání audia v nejlepší kvalitě, přístup k iTunes knihovně. Nabízí jak vysokoúrovňové knihovny pro práci se zvukem (*The Media Player framework*), tak nízkoúrovňové (*Core Audio framework*) pro větší možnost ovlivnění operací.

Video technologie nabízí správu statického obsahu videa, přehrání obsahu z internetu nebo také nahrání videa a práci s ním v rámci aplikace. Opět zde nalezneme nízkoúrovňové knihovny pro vetší míru ovlivnění operací (*Core Media*) nebo pro snadnější práci vysokoúrovňové (*Media Player framework*).

Core Services

Tato vrstva obsahuje základní systémové služby pro aplikace. Nejdůležitější služby jsou dostupné pomocí frameworků *Core Foundation* a *Foundation*, které definují základní typy využité v aplikacích. Další služby poskytované touto vrstvou dávají možnost provádět platby uvnitř aplikace, sledovat aktuální polohu uživatele využívající všechny možnosti zjištění lokace (Wi-fi , telefonní síť, GPS), ukládat uživatelská data nebo podporovat zpracování XML dokumentů.

Core OS

Vrstva Core OS obsahuje nízkoúrovňové prvky, které nepřímo využívá většina výše postavených frameworků. Tuto vrstvu a její frameworky využijeme, chceme-li řešit bezpečnost nebo komunikaci s externími přídavnými zařízeními.

3.1.1 Push notifikace

Push notifikace je způsob jak oznámit uživateli, že jsou dostupná nová data pro jeho aplikaci. Je vhodná v situacích, kdy většina nebo všechna data jsou spravována serverem. Můžete se tak rozhodnou v jakých situacích uživatele upozorníte. Tyto notifikace jsou vhodné například u aplikací komunikačního typu. Dorazí-li nová zpráva když má uživatel aplikaci vypnutou, je o ní informován okamžitě [2].

Jakým způsobem bude uživatel informován můžete vybrat z možnosti:

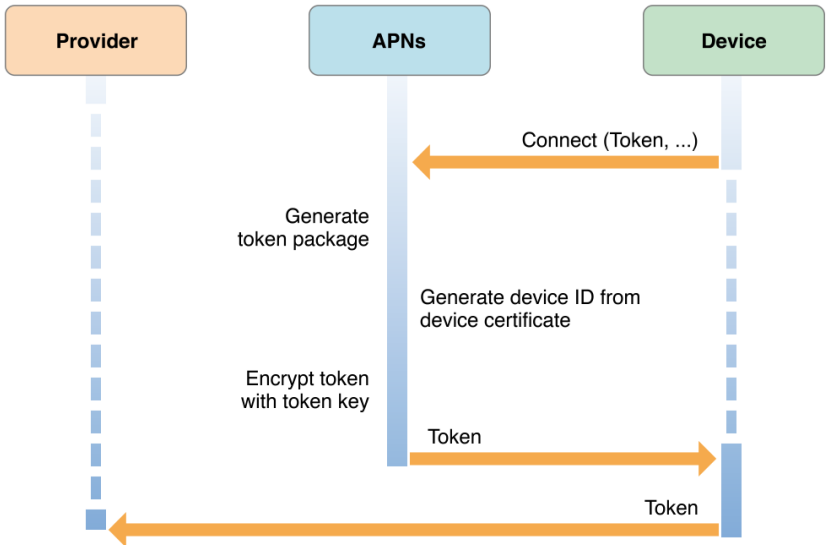
- Na obrazovce banner nebo alert
- Odznak na vaší ikoně aplikace
- Zvuk, který je přehrán při upozornění

Výše uvedené možnosti lze kombinovat nebo zvolit pouze jednu. Uživatel má samozřejmě možnost upozornění systémově pro konkrétní aplikaci vypnout. Poté APNs[1] toto upozornění neodešle.

Push notifikace jsou skvělým doplňkem aplikace, který může vylepšit kontakt s uživatelem, ale při nevhodném využití může být pro uživatele obtěžující.

Celkový průběh nastavení push notifikací v aplikaci je komplikovanější. V zjednodušeném principu je potřeba povolit notifikace v aplikaci a při jejím spuštění zařízení zaregistrovat k jejich odběru. Každé zařízení má vlastní token, který si při spuštění vyžádá aplikace z APNs. Ta jej vygeneruje a vrátí. Aplikace získá token a ten uloží na server. Server při odeslání notifikace pošle token zařízení a formát upozornění na APNs. Po úspěšné validaci tokenu, odešle APNs na zařízení notifikaci. Princip získání tokenu zařízení můžeme vidět na obrázku 3.2.

[1] Apple Push Notification service (APNs) - služba zajišťující rozesílání vzdálených upozornění pro iOS zařízení



Obrázek 3.2: Ukázka principu získání tokenu zařízen (Převzato z [2])

3.2 CocoaPods

CocoaPods [14] je manažer závislostí pro objective-C, swift a jiné jazyky. Zaměřuje se na distribuci zdrojových kódů třetích stran a jejich automatickou integraci do Xcode[2] projektů. Existují tisíce vytvořených balíčků, které lze importovat pomocí CocoaPods. Tento koncept podporuje znovupoužitelnost v různých projektech a je hojně využíván.

CocoaPods jsou ovládány z příkazové řádky. Obsahují příkazy pro inicializaci, instalaci, updatování v projektu, vytvoření nového balíčku apod. Každý CocoaPods je definován souborem s příponou *podspec*. V něm lze nalézt název, verzi, popis, autora, licenci, cestu ke zdrojovým kódům, závislosti na jiné balíčky. Syntaxe souboru *podspec* musí být validní, aby šel balíček použít.

Po inicializaci CocoaPods v projektu je vytvořen soubor Podfile[3]. Zde se definují závislosti projektu na jednotlivých CocoaPods balíčcích. Po úspěšné instalaci závislostí, otevřením projektu jako *workspace*, jsou viditelné i zdrojové kódy připojených balíčků. Při definici závislosti v *Podfile* můžeme specifikovat verzi, umístění balíčku nebo tag. Ten je vhodné použít, chceme-li pracovat s určitou verzí. Při vývoji CocoaPods je výhodné definovat lokální cestu k balíčku a ten je tak dostupný ve složce *Development Pods*.

CocoaPods jsou vytvořeny převážně pro veřejné využití, dovolují však také vytvářet soukromé pody. U těch je potřeba vždy v *Podfilu* specifikovat cestu.

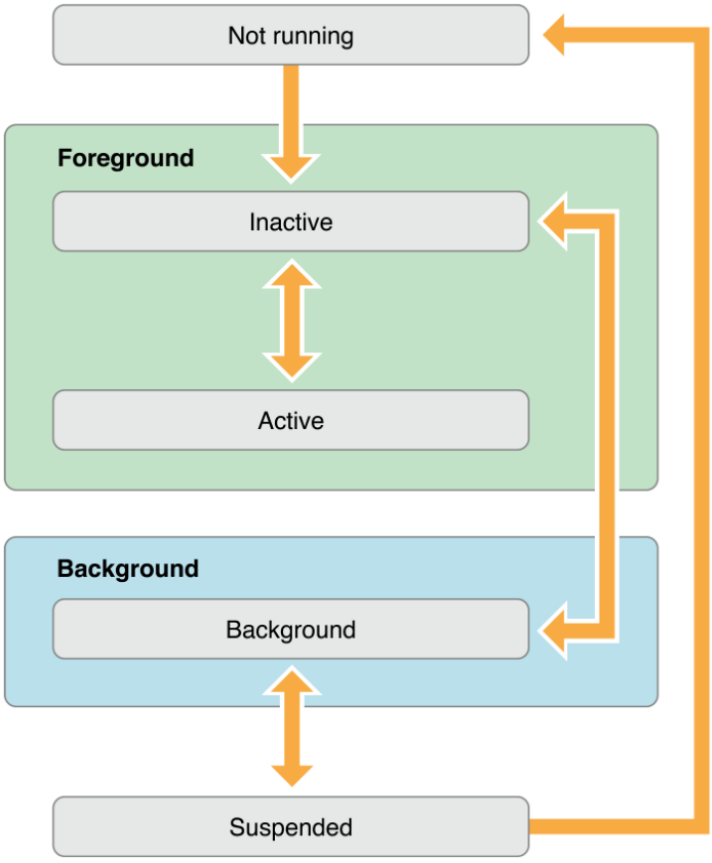
3.3 Stavy aplikace

Každá aplikace ve svém životním cyklu prochází řadou stavů. Přechody mezi stavy můžeme vidět na obrázku 3.3. Jednotlivé stavy mají definovaný přistup k prostředkům systému jako připojení k síti nebo ukazatele na soubory. Význam jednotlivých stavů [5]:

- **Neběžící** – Základní stav aplikace po stažení nebo restartu systému, kdy aplikace neběží, nevyužívá žádné zdroje. Aplikace byla vypnuta nebo ještě nebyla spuštěna.

[2]Xcode - Vývojové prostředí Applu pro vývoj aplikací na Apple zařízení
[3]Podfile - https://guides.cocoapods.org/syntax/podfile.html

- **Neaktivní** – Při spuštění, aplikace přejde krátce přes tento stav, přestože již běží, není připravena k přijetí uživatelských událostí.
- **Aktivní** – Normální režim aplikace běžící na popředí a přijímající události. Tento stav je běžný pro aplikace, které nevykonávají žádnou činnost na pozadí a neběží bez uživatelského rozhraní.
- **Na pozadí** – Aplikace je schopna vykonávat činnost, přestože není na popředí uživatelského rozhraní zařízení, většinou prochází krátce tímto stavem před pozastavením. V případě potřeby více času pro dokončení běžících úloh z popředí aplikace může zůstat v tomto stavu delší požadovaný čas. Při zavádění aplikace přímo do tohoto stavu, nahrazuje přechod přes neaktivní stav. Kromě dokončení úloh běžících na popředí může také vykonávat dlouhodobé činnosti na pozadí, které jsou popsány v podkapitole 3.4. Prováděná aktivita na pozadí by měla být pokud možno co nejkratší.
- **Pozastavený** – Aplikace běží na pozadí, ale nevykonává žádný kód. Systém automaticky přesune aplikaci do tohoto stavu, aniž by o tom informoval. V případě nedostatku paměti systém automaticky ukončí aplikace v pozastaveném stavu pro její uvolnění.



Obrázek 3.3: Změny stavů v iOS aplikaci (Převzato z [5])

Hlavním objektem pro vlastní nastavení chování aplikace při přechodech mezi stavy a reakce na další události je *AppDelegate*. Ten zachytává její inicializaci, změny stavů a další

vysokoúrovňové události. Protože je *AppDelegate* přítomen v každé aplikaci je navíc vhodný pro zaregistrování a inicializování globálních služeb. Příkladem vlastní reakce na přechody mezi stavy může být persistence dat, uvolňování zdrojů apod.

3.4 Dlouhodobé úlohy na pozadí

Některé aplikace vyžadují dlouhodobý běh úkolů na pozadí například aktualizace polohy zařízení nebo přehrávání audia. Systém se stará o efektivní využívaní zdrojů aplikacemi a proto ty, vyžadující provádění dlouhodobých aktivit na pozadí, potřebují zvláštní povolení, aby nebyly automaticky systémem pozastaveny nebo byly vůbec spuštěny. V iOS mohou běžet na pozadí pouze určité typy aktivit [4]:

- **Audio and AirPlay** – Aplikace přehraje audio obsah nebo zaznamená zvuk na pozadí. Tato aktivita zahrnuje streamování audio nebo video obsahu pomocí AirPlay. Při prvním použití uživatel musí povolit oprávnění aplikace využívat mikrofon.
- **Aktualizace lokalizace** – Aplikace udržuje aktuální lokaci zařízení při běhu na pozadí.
- **VoIP** – Aplikace umožňuje telefonní hovory přes internetové připojení namísto mobilních služeb. Tato aktivita vyžaduje trvalé připojení k internetu a další služby, aby aplikace mohla přijímat hovory na pozadí apod.
- **Stahování Kiosku** – Kiosek aplikace umožní stahování nových novin a magazínů na pozadí.
- **Komunikace s externím zařízením** – Aplikace, které pracují s externími příslušenstvími, mohou požádat o probuzení, když příslušenství poskytuje aktualizaci dat.
- **Komunikace s externím Bluetooth zařízením** – Aplikace, které pracují s periferními zařízeními připojenými přes Bluetooth, mohou požádat o probuzení vždy, když například měřič tepové frekvence na hrudi v pravidelných intervalech zašle data přes Bluetooth rozhraní.
- **Načtení na pozadí** – Aplikace pravidelně stahuje a zpracovává malé množství obsahu.
- **Vzdálené oznámení** – Aplikace při příchodu push notifikace viz 3.1.1, začne stahovat obsah. Minimalizuje tak dobu od oznámení o novém obsahu ke schopnosti zobrazení nových dat v aplikaci.

Systém při prvním spuštění aplikace požádá uživatele o povolení vykonávání těchto úloh, to lze v systémovém nastavení aplikace kdykoli změnit. K provedení výše popsaných aktivit systém přiděluje aplikaci zdroje (přístup k síti, čas na procesoru), v případě nízké spotřeby baterie zařízení je může omezit nebo dokonce neposkytnout vůbec. V takové situaci je běžící aktivita ukončena a další již neprovedena [6].

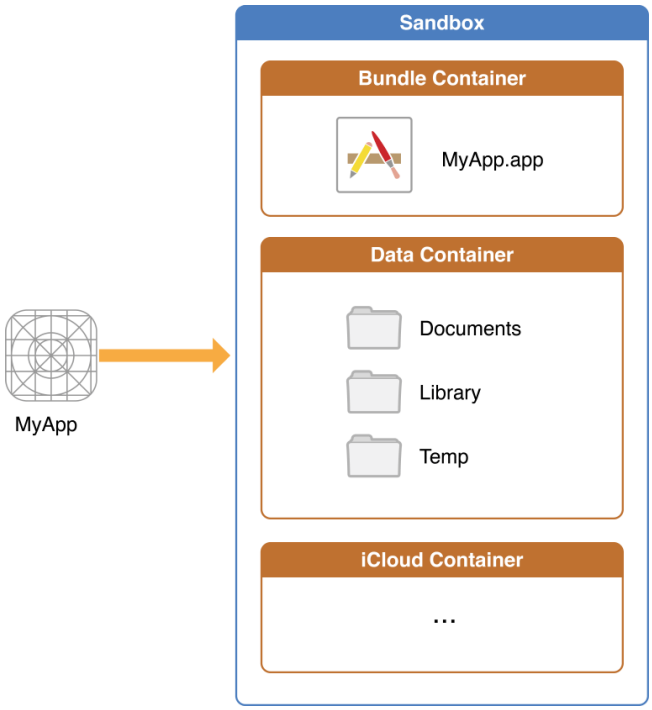
3.5 Práce s daty

Z důvodu bezpečnosti souborového systému má každá aplikace v iOS vytvořený svůj vlastní sandbox, kde může ukládat a dále pracovat s vlastními soubory. Ukázku jeho struktury můžeme vidět na obrázku 3.4. Sandbox je rozdělen do několika kontejnerů, které mohou být dále strukturovány. Jednotlivé kontejnery mají různá využití a povolení ke změně jejich obsahu.

Hlavním kontejnerem aplikace je *Bundle Container*, který obsahuje samotnou aplikaci a všechny zdroje (ikony, storyboad[4]). Tento kontejner je pouze ke čtení. Pro ověření jeho nezměněné podoby je při instalaci podepsán, jakákoli jeho změna pak vede k nespuštění aplikace.

Další kontejner *Data Container* je použit pro ukládání souborů a dat uživatele. Podle typu uložené informace a přístupu uživatele se dělí na složky. Více k jejích konkrétnímu využití v [3]. Některé složky tohoto kontejneru jsou zálohovány pomocí iTunes[5] nebo iCloudu[6].

Obecně tak není dovoleno vývojáři z aplikace pracovat se soubory mimo tento prostor, může však vyžádat přístup a připojit další kontejnery jako fotogalerie, kontakty nebo iCloud.



Obrázek 3.4: Ukázka struktury sandboxu aplikace (Převzato z [3])

Výše popsané informace ukazují, jak pracovat v rámci souborového systému aplikace, a její možnosti. V mnoha případech potřebujeme pracovat nejenom s daty jako soubory, ale také informacemi ukládanými do lokální databáze. Jedním ze souborů datového kontejneru pak právě může být ten, který drží tyto informace. Pro implementaci lokálního úložiště na

[4]Storyboard – visuální reprezentace uživatelského rozhraní
[5]iTunes – aplikace pro správu apple zařízeních, jejich souborů a další související aktivity na osobní počítač
[6]iCloud – cloudové úložiště a cloud computing služba od firmy Apple Inc.

mobilních zařízeních iOS existuje více frameworků. Ty nejpoužívanější jako Realm, Core Data a SQLite jsou více popsány a srovnány dále.

3.5.1 SQLite

SQLite je open source široce využívaný databázový nástroj dostupný pro Mac OS, iOS, Android, Linux a Windows platformy. Využívá konceptu databázových systémů jako MySQL nebo SQL. Data jsou uložena běžným způsobem jako v relačních databázích a lze na ně aplikovat standardní SQL dotazy. Mezi hlavní výhody SQLite patří uložení databáze do jednoho souboru na disku, který je možné přenášet mezi platformami. Dále nevyžaduje žádnou konfiguraci, je nezávislý na serveru a umožňuje jednoduchý a bezpečný přístup k datům z více vláken [12].

3.5.2 Core data

Core data jsou hlavním podporovaným frameworkem pro persistenci dat vyvíjené přímo společností Apple. Oproti standardním relačním databázím Core data umožňují vývojáři pracovat s větší mírou abstrakce nad daty.

Core data nejsou databáze, ale framework pro správu grafu objektů, který nabízí další funkce jako vstupní validace, verzování datového modelu nebo sledování změn. Pro uložení serializuje data do formátu XML, binárního nebo SQLite [15].

Ve srovnání s SQLite jsou Core data rychlejší, ale uložení dat zabírá více místa na disku. Pro ukládání komplexnějších dat je vhodné zvážit využití Core data i přes časově náročnější základní orientaci ve frameworku.

3.5.3 Realm

Realm mobilní databáze je alternativa k SQLite a Core data. Je vytvořena primárně pro mobilní zařízení. Oproti tradičním databázím jsou objekty v Realmu jako nativní objekty. Není potřeba vytvářet kopie objektů, ale lze přímo získaný objekt upravovat, kde změny se projeví ihned. Realm je multiplatformní, dostupný i pro Android, Xamarin a React Native. Databázové soubory lze snadno sdílet mezi těmito platformami.

Mezi hlavní výhody a rostoucí popularitu tohoto frameworku patří rychlost oproti SQLite a Core data, implementace Realm databáze do projektu je velmi rychlá a vyžaduje minimum kódu. Nejsou zde žádná omezení ukládání dat i při větším zaplnění zaručuje konzistentní rychlost a výkon. Oproti ostatním také poskytuje spolehlivou aplikaci pro prohlížení dat vyvíjenou přímo vývojáři frameworku.

V aplikaci je možné využít více Realm databází například jednu uloženou lokálně a další spravovat vzdáleně ze serveru. K tomu podporuje také využití Realm Object serveru, tzn. stejné úložiště, je také vytvořeno na serveru a po správném nakonfigurování připojení jsou automaticky obě databáze synchronizovány. Při úpravě dat na zařízení jsou nejdříve změny uloženy lokálně a při nejbližší příležitosti dosynchronizovány na server.

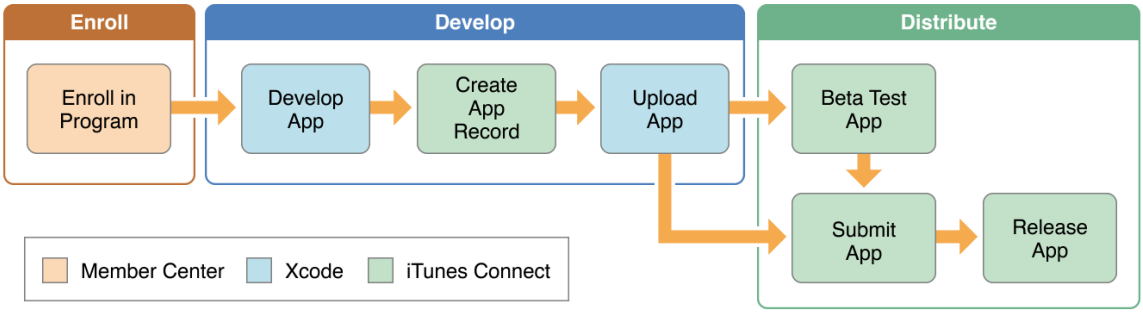
Mezi nevýhody patří podpora iOS pouze verze 8 a novější. Také některé operace jako kaskádové mazání navázaných objektů, ještě nejsou implementovány.

3.6 Distribuce aplikace

Distribuce aplikací pro zařízení Apple je složitější oproti platformě Android, zaručuje však bezpečnost a určitou kvalitu dostupných aplikací v AppStoru[7].

Možnost nainstalovat si vyvíjenou aplikaci na reálné zařízení, je dostupná poté, co má uživatel platný vývojářský účet se zaplaceným poplatkem nebo je součástí organizace, která má platnou licenci pro vyvíjení aplikací. Do této doby může testovat aplikaci pouze v simulátoru. Tím je zabráněno, aby si kdokoli s vývojovým prostředím vytvořil aplikaci pro vlastní využití bez vývojářské licence [9].

Jakmile vývojář dokončí aplikaci do fáze, kdy ji chce publikovat nebo ji dát testovat, vytvoří záznam o aplikaci v iTunes Connect[8], vyplní údaje o aplikaci, vloží náhledy apod. Poté nahraje aplikaci a může ji poskytnout pro testování vybraným uživatelům přes aplikaci TestFlight, nebo ji vložit do AppStoru ke stahování. Přehledné schéma jednotlivých kroků k distribuci aplikace je možné vidět na obrázku 3.5.



Obrázek 3.5: Schéma jednotlivých kroků k distribuci aplikace (Převzato z [2])

Aplikace před vložením do AppStoru nebo k testování prochází kontrolním mechanismem, kdy je ověřeno, zda jsou uvedeny všechny využívané služby zařízení (kamera, přístup k fotkám apod.) a je dodáno oprávnění k využití dat třetích stran. V další fázi je aplikace otestována reálným uživatelem. Pro testování aplikace je schvalovací proces aplikace pouze částečný a rychlejší oproti validaci do AppStoru. Průměrná doba schválení publikace do AppStoru je 7 dní.

[7] AppStore - obchod s aplikacemi pro zařízení se systémem iOS
[8] iTunes Connect - webová služba pro správu vyvíjených aplikací

Kapitola 4

Návrh

Hlavním úkolem této kapitoly je představit čtenáři cíl, který tato práce řeší, návrh řešení celé aplikace a jeho důležité součásti.

V současné době je populární přidávat komunikaci do různých systémů, jak s webovým, tak mobilním uživatelským rozhraním. Příkladem může být komunikace se zákazníkem k objednávce nebo interní konverzace k montážní zakázce apod. Cílem je tedy navrhnout aplikaci klient-server pro jednoduchou i skupinovou konverzaci uživatelů s možností zasílání multimédií, jejíž komponenty budou dále integrovatelné do jiných systémů.

Kapitola návrh se zabývá konkrétními detaily v oblasti instant messagingu, složitější správa uživatelských účtů není součástí návrhu. V první části kapitoly jsou definovány jednotlivé akce uživatele v rámci aplikace a poté podrobněji popsány rozdělení systému, komunikace jeho komponent a jejich detailnější rozbor.

4.1 Uživatelské akce

Uživatel v aplikaci vykonává řadu akcí. Na tyto akce lze pohlížet jako na akce vykonávané uvnitř místnosti (konverzace) nebo v rámci celého systému. Rozdělení těchto situací můžeme vidět na obrázku 4.1 a vysvětleny jsou v následující části.

4.1.1 Místnost

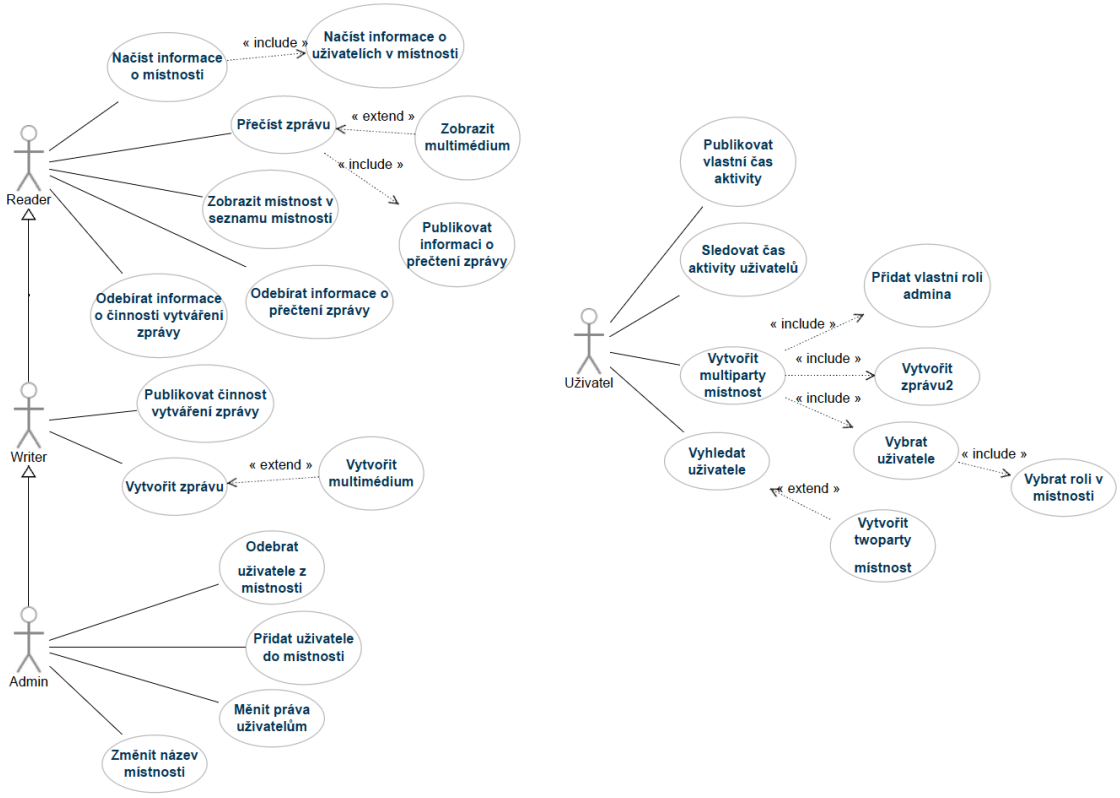
Uživatel přidaný do komunikace nebo po vytvoření vlastní získá uživatelskou roli v dané konverzaci. Ta ho poté opravňuje k uživatelským akcím. Uživatelské role jsou hierarchicky seřazeny na:

- **Reader** – Nejmenší oprávnění v místnosti opravňuje k načtení informací o chatu, čtení jednotlivých zpráv se zobrazením multimédií a publikaci informace o přečtení zprávy. Toto oprávnění určuje, že daný uživatel do místnosti patří, proto ji také může vyhledat.
- **Writer** – Standardní uživatelská role v místnosti, hierarchicky obsahuje oprávnění k akcím role Reader a přidává možnost přidat zprávu do místnosti. Při vytvoření zprávy může uživatel nahrát soubor a ten zaslat jako její přílohu. K oprávnění vytvořit zprávu přibývá také možnost publikovat svoji činnost při vytváření zprávy, tzn. při psaní zprávy dát ostatním uživatelům informaci o tom, že jsem začal/přestal psát.

- **Admin** – Nejvyšší oprávnění, k akcím rolí Reader a Writer, také dovoluje spravovat práva uživatelů v místnosti, přidat nebo odebrat uživatele a změnit název místnosti.

4.1.2 Systém

V celém systému, tzn. bez závislosti na konkrétní místnosti, je uživateli dovoleno sledovat čas poslední aktivity ostatních uživatelů a publikovat také svůj. Aktivita obsahuje časové razítko, kdy uživatel byl naposledy online. Bez ohledu na roli v jiných místnostech může uživatel vytvořit novou hromadnou (multiparty) konverzaci. Při jejím vytvoření musí přidat alespoň další dva uživatele a napsat první zprávu. Systém neřeší správu uživatelů z hlediska, kdo jakého uživatele může vyhledat, proto je možné přidat do místnosti nebo začít komunikovat s kýmkoli. Komunikace „uživatel uživatel“ (twoparty) se vytváří až v době, kdy je vyžadována, tzn. při vyhledávání označením daného uživatele. Nezasílá se tedy požadavek na vytvoření místnosti, ale jen na získání chatu s daným uživatelem. Ten pokud neexistuje, je automaticky vytvořen.



Obrázek 4.1: Use Case diagram uživatele v místnosti (levá část) a celkově v systému (pravá část)

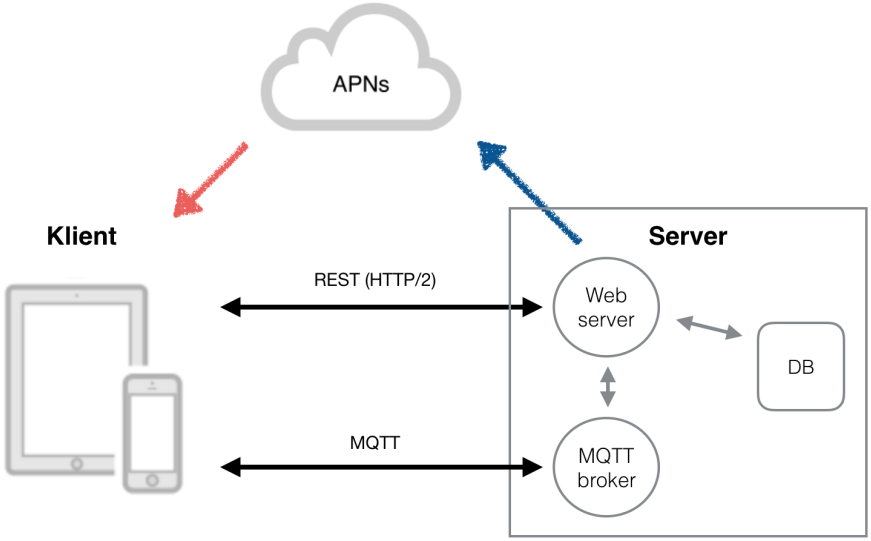
4.2 Struktura systému

Strukturu aplikace lze rozdělit na část serverovou a klientskou. Server 4.3 nabízí klientovi 4.4 dvě rozhraní pro obousměrnou komunikaci. Rozhraní MQTT brokeru je využíváno pro real-time komunikaci, kde klient zasílá novou zprávu, informaci o přečtení zprávy, svoji akti-

vitu při psaní zprávy a přijímá informace o nově vytvořené místnosti (uživatel má oprávnění alespoň pro čtení informací v místnosti), nové zprávě, informaci o přečtení zprávy, změně uživatelů v místnosti a čas poslední aktivity ostatních uživatelů. Pro načtení zbylých informací využívá druhé rozhraní REST. To je implementováno pomocí protokolu HTTP/2 a klient jej využívá k přihlášení, načtení konverzací, zpráv a informací o uživatelích, vytvoření místnosti nebo změně práv uživatelů. Pro stažení a uploadování multimédií je také využito REST rozhraní, avšak objekt nesoucí informace o souboru je přibalený v MQTT zprávě.

Třetím komunikačním kanálem, který je však pouze směrem ke klientovi, jsou push notifikace zprostředkovávané pomocí APNs. Ty mu přináší upozornění na novou událost, nepoužívá-li aktuálně aplikaci na popředí. Lze je však vypnout, proto neslouží k primárnímu účelu synchronizace dat.

Návrh komunikace aplikace instant messengeru a jejího rozdělení můžeme vidět na obrázku 4.2.



Obrázek 4.2: Princip komunikace systému

V rámci serveru MQTT broker komunikuje pouze s webovým serverem. Tento server poskytuje MQTT brokeru ověření autentizace uživatele, autorizace pro publikování a odebírání MQTT zpráv. V případě, kdy je potřeba data z MQTT zprávy uložit do databáze, webový server provede uložení a informuje o tom MQTT broker, který poté zdrojovou zprávu rozešle uživatelům. V ostatních situacích funguje MQTT broker standardním způsobem přeposílání zpráv.

Webový server jako jediný komunikuje s databází pro správnou synchronizaci dat. Dotazy přicházející přes REST rozhraní zpracuje a zašle odpověď, zároveň však může informovat ostatní uživatele přes MQTT rozhraní nebo zasláním push notifikace skrze APNs. Například při vytvoření místnosti nejenom odpoví na REST žádost, ale také informuje ostatní uživatele o vytvoření nové místnosti MQTT zprávou a push notifikací. Popis jednotlivých komponent je popsán detailněji v dalších podkapitolách.

4.3 Server

Serverovou část aplikace můžeme rozdělit na tři vzájemně spolupracující komponenty. Komponenty webový server a databáze jsou rozděleny na korespondující podaplikace (moduly). Jednotlivé komponenty, jejich podrobný návrh a synchronizace vzájemné komunikace jsou popsány v následujících podkapitolách.

4.3.1 Databáze

První komponenta představuje databázi, která slouží k ukládání veškerých potřebných informací jako historii všech komunikací, multimédia a uživatele. Pro respektování obecného návrhu je databáze rozdělena do více schémat. Jednotlivá schémata jsou na sobě do určité míry závislá pro omezení redundance dat. Pro respektování obecného návrhu jsou data jednotlivých modulů aplikace uložena v samostatných schématech databáze. Samotný návrh je rozdělen do tří schémat. Na obrázku 4.3 můžeme vidět schémata *Cloud* a *Notifications*. Tato dvě schémata jsou navržena jako základní, jejich funkcionalita se dá rozšiřovat, ale měla by vždy dodržet základní navrženou strukturu v návaznosti na schéma *Messengeru*.

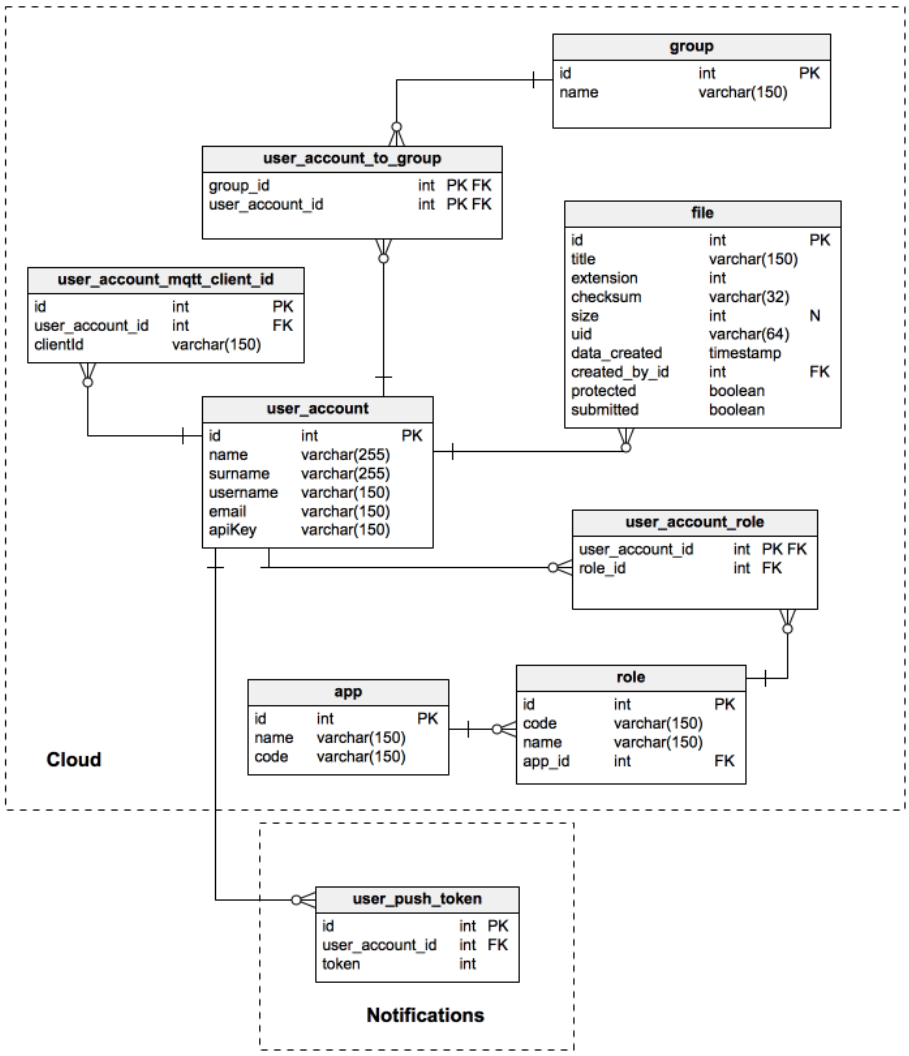
Schéma *Cloud* obsahuje tabulky pro základní správu uživatelů, skupin a rolí. Toto schéma obsahuje téměř vždy každá aplikace, tzn. správu uživatelských účtů, jejich rolí apod. Hlavní tabulkou je zde *user_account* dále jen uživatel, ten je definován unikátním emailem. Uživatel pak může být přiřazen do skupin a mohou mu být přiřazeny uživatelské role. Ty jsou zde chápány obecně pro využití ve více modulech aplikace. Například uživatel může mít roli admin v rámci modulu messengeru, ale také roli kontrolora v rámci modulu výroby. Tyto dvě role jsou pak vybírány správně v kontextu, díky cizímu klíči určující podaplikaci (modul). Tabulka *file* obsahuje název, příponu, velikost, datum vytvoření, autora, checksum pro ověření správnosti souboru a položku *submitted* označující soubor jako využitý (přiřazený ke zprávě, zakázce). Další informací, která je potřeba ukládat, je unikátní *clientId* pro každé přihlášení uživatele k mqtt brokeru z různých zařízení v tabulce *user_account_mqtt_client_id*. Celé schéma je možné rozšířit dalšími informacemi o uživateli jako jeho adresa a další potřebné věci k určitému kontextu využití celé aplikace.

Schéma *Notifications* obsahuje pouze jednu tabulku *user_push_token*. Ta reprezentuje podobné informace jako *user_account_mqtt_client_id* v rámci schématu *Cloud*, ale pro každé zařízení registrované uživatelem k odběru push notifikací. Tato tabulka je vyčleněna do samostaného schématu, které obsahuje obecné informace o notifikacích a může být rozšířeno o tabulky uchovávající historii všech notifikací, rozdělení různých typů notifikací apod.

Pro rychlejší vyhledávání jsou pak vytvořeny indexy na všechny primární klíče, cizí klíče, a na položky, podle kterých se vyhledávají data. Indexy se tedy vytvoří také na jméno, příjmení, email a apikey uživatele, kód aplikace a role, jméno skupiny a mqtt klientské id.

Část specifická pro oblast messengeru je vydělena do schématu *Messenger*, které můžeme vidět na obrázku 4.4. Návrh tohoto schématu určuje jeho chování, rozsah zprostředkovaných infomací a způsob jejich využití. Hlavní tabulkou je zde *chat_room*, ta nese informaci o tom, zda jde o komunikaci dvou uživatelů nebo skupiny. Dále také pomocnou informaci o času poslední přijaté zprávy, datum vytvoření a název topicu, přes který probíhá veškerá real-time komunikace a to včetně notifikace o přidání uživatele, změny práva uživatele v místnosti nebo potvrzení přečtení.

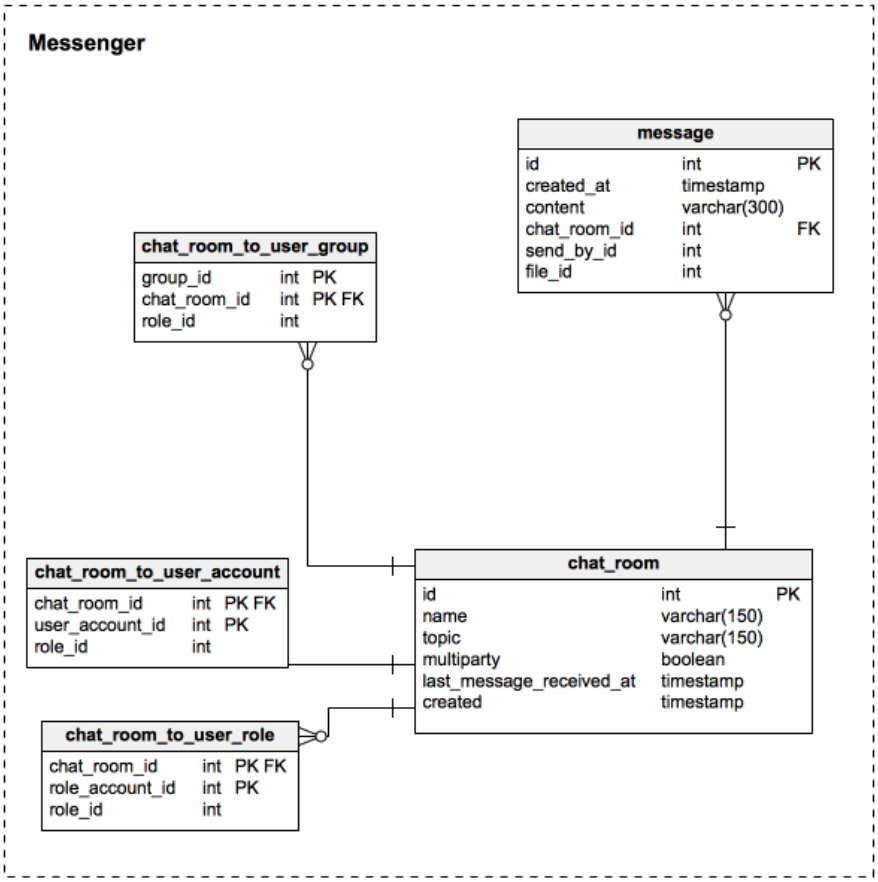
Na tabulku *chat_room* jsou pak navázány ostatní tabulky ve schématu. Zasílané zprávy vytvořené uživatelem, jejich obsah, případně odkaz na přílohu, jsou uchovávány v tabulce



Obrázek 4.3: Návrh databázového schématu pro cloud a notifications

message. Každá zpráva může obsahovat jeden přiložený soubor. Přidání uživatele do konverzace v místnosti je možné třemi způsoby. Standardním způsobem je uživatel přidán do místnosti tabulkou *chat_room_to_user_account* s rolí v dané místnosti. Role v rámci messengeru je případ popsaný výše a odkazuje cizím klíčem do tabulky rolí ve schématu *Cloud*, které jsou odlišeny *app_id* pro *Messenger*. Dále je možné přidat do místnosti skupinu, do níž uživatel patří a má tak stejnou roli jako ostatní členové skupiny. Příkladem může být skupina pracovníků určitého střediska, kterou lze jedním záznamem v databázi přidat do komunikace. Poslední možností, která najde využití více v jiných typech aplikací než čistě komunikačních, je přidání uživatele přes roli. Tento případ je užitečný tehdy, kdy chcete přidat specifickou roli pro přístup k dané komunikaci, například všechny uživatele s rolí mistra do všech místností k montážním zakázkám.

Také zde jsou vytvořeny indexy pro rychlejší vyhledávání na primární, cizí klíče a dále na všechny role představující roli v rámci messengeru, datum vytvoření zprávy a u tabulky pro místnost na jméno, datum poslední přijaté zprávy, datum vytvoření a typ místnosti (multiparty).



Obrázek 4.4: Návrh databázového schématu pro messenger

4.3.2 Webový server

Druhá komponenta webový server komunikuje skrze REST[1] rozhraní. To poskytuje klientské aplikaci všechny služby, které pracují s vetším tokem dat nebo nejsou vhodné pro zasílání informací přes MQTT rozhraní. Pro uchovávání aktuálních informací v klientské aplikaci může některé změny v datech po REST dotazu webový server rozeslat mqtt zprávou.

Webový server je jediná komponenta, která přímo komunikuje s databází a to pro zajištění synchronizace dat. Při potřebě ukládat nebo číst informace jsou ostatní komponenty nuceny komunikovat přes webový server. Pro akce přihlášení uživatele, načítání historie zpráv, vytvoření místnosti, zobrazení seznamu místností uživatele nebo editace uživatelů skupinové místnosti, klientská aplikace využívá REST rozhraní webového serveru. Pro zasílání zpráv a informace, které je potřeba v reálném čase, klient odesílá a přijímá MQTT zprávy.

Webový server zajišťuje rozesílání push notifikací pomocí APNs. Uživatel při přihlášení zasílá spolu s uživatelským jménem a heslem také jednoznačný token zařízení. Ten je v případě úspěšného přihlášení uložen a následně jsou mu zasílána na dané zařízení i oznámení v podobě push notifikací. Má-li uživatel v zařízení k aplikaci povolené oznámení, systém

[1]REST (Representational State Transfer) - architektura rozhraní

je zobrazí, i když není aplikace zapnutá. Webový server nemá informaci, zda-li je uživatel online, protože tuto informaci není žádoucí ukládat do databáze, proto se push notifikace zasílají i v případě, kdy je uživatel online a má aplikaci na popředí. Toto představuje do jisté míry výhodu v dalších verzích klientské aplikace, kdy se změní princip reakce na notifikace aplikace na popředí, ale serverová část zůstane stejná.

4.3.3 MQTT Broker

Třetí komponenta MQTT broker zajišťuje real-time komunikace. Pro zachování synchronizace dat se MQTT broker dotazuje na informace v databázi nepřímo skrze webový server. Oprávnění uživatele pro připojení k MQTT brokeru, publikovaní zprávy nebo odebíraní zprávy k určitému topicu jsou ověřována přes rozhraní webového serveru. Real-time komunikace probíhá přes definovaný protokol specifikovaný níže v tabulkách 4.1, 4.2, 4.4, 4.3 a 4.6. Tento protokol obsahuje formulaci jednotlivých typů zpráv pro využití v aplikaci messengeru. Samozřejmostí komunikace je šifrování pomocí SSL nebo TLS.

MQTT broker využívá první úroveň spolehlivosti doručení zpráv (QoS 1). Nemůže nastat situace, kdy MQTT zpráva nedorazí. Klientská aplikace i server se však musí vypořádat s případným doručením stejné zprávy vícekrát. To nelze ověřit podle položky *packetId* v MQTT zprávě, protože ta nemusí být unikátní. Pro tuto nejednoznačnost je do protokolu v první vrstvě zprávy přidána položka *hash*. Ověření probíhá dotazem do databáze s časovým rozsahem (jednotky minut). Pravděpodobnost výskytu dvou stejných hashů v nedávné době je minimální. Spolehlivost doručení se netýká jenom publikování zpráv směrem ke klientům, ale také při publikování od klienta. Klient při publikování zprávy na určitý topic dostává od serveru potvrzující paket, to samé klient odesílá při přijetí paketu.

MQTT broker využívá v režimu QOS 1 pro ukládání paketů, které nebyly odeslány, rychlé lokální úložiště jako například Redis[2]. Protože využíváme MQTT broker pro real-time komunikaci, nepotřebujeme držet a posílat informace, které již nejsou aktuální. Proto je nastavena jednotlivým paketům expirace. Stejným způsobem si MQTT broker ukládá i informace o klientech a jejich odebíraných topicích. Odebírání zpráv na určitý topic také po určeném čase vyprší. Klient jej může automaticky prodloužit posláním subscribe paketu ještě před uplynutím expirace nebo se k odběru znovu přihlásit .

Pro zasílání stavu, zdali je uživatel online, je využito retain módu zprávy, který je popsán v kapitole 2.1. Uživatel tak publikuje svoji aktivitu s časovým razítkem a jakmile se někdo přihlásí k odběru této informace, dostane okamžitě zprávu o jeho poslední aktivitě.

4.3.4 Hierarchie MQTT topiců

Pro rozlišení oprávnění uživatelů k informacím zasílaným přes MQTT rozhraní a důležitosti informace pro uživatele se využívá hierarchické struktury topiců viz 2.1.

Pro informace relevantní pouze k dané místnosti existuje unikátní topic. Ten se skládá z obecného názvu místnosti a unikátního vygenerovaného identifikátoru při vytváření místnosti v další úrovni */chat-room/{chatRoomUid}* . Ten se využívá namísto primárního klíče z databáze pro jeho podstatně horší odhalení k případné nežádoucí činnosti. Na tento topic jsou zasílané uživatelem vytvořené nové zprávy (tabulka 4.2), informace o jeho činnosti vytváření zprávy (tabulka 4.5). Každý uživatel s právem číst zprávy v místnosti, publikuje informaci o poslední přečtené zprávě (tabulka 4.3). O změně uživatelů nebo jejich rolí v místnosti informuje server (tabulka 4.4).

[2]Redis - viz.https://redis.io

Dalším definovaným topicem v messengeru je obecný uživatelský topic */messenger-notifications/{userId}*. Ten slouží k zasílání informací pouze uvnitř messengeru. Lze jej využít v situacích k lepší synchronizaci dat po úspěšném přihlášení uživatele do aplikace nebo pro obdržení nových zpráv v místnostech se starší konverzací, kde uživatel automaticky neodebírá zprávy pro místnost. Přístup k odebírání zpráv má pouze konkrétní uživatel a je zde serverem zasílán pouze typ nové zprávy (tabulka 4.2).

Poslední využitý topic v aplikaci je obecný pro uživatele. Ten zde zasílá informace o své poslední aktivitě (tabulka 4.6), které ostatní uživatelé mohou odebírat. Jeho struktura vypadá následovně */user/{userId}/online*.

4.3.5 Protokol MQTT zpráv

V rámci aplikace bylo nutné navrhnout a definovat vlastní protokol pro komunikaci přes MQTT rozhraní tak, aby byla jasná komunikace pro všechny její účastníky. Díky tomu tak nedochází k nejasnostem nebo v nejhorším případě zahazování zpráv vlivem špatného formátu.

MQTT message

Obálku nad všemi typy zpráv tvoří vlastní MQTT message, kterou můžete vidět v tabulce 4.1. Ta definuje typ zprávy obecného objektu obsahující informace ke konkrétnímu typu zprávy a hash. Podle typu zprávy je možné potom z dat vytvořit správný typ objektu. Položka hash slouží k identifikaci MQTT zprávy, jak pro server, tak i pro klientskou aplikaci.

Název	Typ
type	string
data	object
hash	string

Tabulka 4.1: Zpráva MQTT message

Dále budou popsány jednotlivé typy zpráv a obsah jejich položky data.

New message

Pro odeslání nové zprávy slouží typ zprávy *new-message-notification*, která v datech obsahuje položky z tabulky 4.2. Protože uživatel má již načtené informace o místnosti a uživatelích, stačí poslat pouze id místnosti a uživatele, tím docílíme menšího datového toku. Položka *uid* je jednoznačný identifikátor v krátkém časovém rozsahu (jednotky minut) a slouží pro identifikaci duplicity zpráv na serveru a jednoznačného určení v rámci lokálního úložiště aplikace, není-li ještě zpráva synchronizována se serverem. Volitelná položka zprávy *file* je pro okamžitou možnost načítání dat zasílána jako objekt nikoli pouze *id*. Objekt *file* obsahuje název, url , datum vytvoření, velikost, formát souboru a náhledy ve dvou velikostech.

Last read message

Při zobrazení zpráv v komunikaci uživatel informuje ostatní o přečtení poslední zprávy MQTT zprávou typu *read-message-notification*. Tato informace je posílána pouze na topic

Název	Typ
sentById	int
chatRoomId	int
content	string
uid	string
file	object?

Tabulka 4.2: Zpráva MQTT new message

místnosti, takže pokud ji uživatel dostane, má všechny potřebné informace a stačí posílat pouze id místnosti, zprávy a uživatele, jak můžeme vidět v tabulce 4.3.

Název	Typ
messageId	int
userId	int
chatRoomId	int

Tabulka 4.3: Zpráva MQTT last read message

ChatRoom change notification message

Editace práv uživatele v místnosti, tzn. i jeho přidání do místnosti, není zásadní pro klientskou aplikaci, protože server v případě odebrání práv uživateli mu nedovolí provést akci, na kterou nevlastní práva. Vzhledem k zobrazování informací o uživatelích například v detailu místnosti je vhodné mít aktuální data. K notifikaci o změně uživatelů nebo jejich rolí v rámci místnosti slouží typ zprávy *chatroom-change-notification*. Vzhledem ke komplexnější možnosti přidávání uživatelů do místností a definování jejich rolí, tento typ zprávy slouží pouze jako upozornění, kdy po jejím obdržení je potřeba obnovit informace dotazem na webový server. Zpráva obsahuje pouze id místnosti, ve které byla provedena změna a id uživatele, který změnu provedl viz tabulka 4.4.

Název	Typ
editById	int
chatRoomId	int

Tabulka 4.4: Zpráva MQTT chatroom change notification

Creating message info

Uživatel při vytváření zprávy může oznámit tuto činnost ostatním uživatelům v místnosti v podobě nové MQTT zprávy *creating-message-info*. Tento typ zprávy není žádným způsobem zpracováván serverem a je pouze přeposílán uživatelům v místnosti přes MQTT rozhraní. Uživatel by měl při korektním chování zasílat i zprávy o ukončení této akce. U této zprávy je potřeba ošetřit situaci, kdy nedorazí zpráva o ukončení činnosti. To lze například expirací této infomace v klientské aplikaci. V tabulce 4.5 pak můžeme vidět strukturu zprávy s informací o stavu činnosti a jejím časovém razítku. Informace o vytváření nové zprávy může být svázána logikou v klientské aplikaci s typem zprávy oznamující poslední aktivitu uživatele, která je definována dále.

Název	Typ
creating	Bool
time	Timestamp

Tabulka 4.5: Zpráva MQTT creating message info

Online user message

Volitelnou zprávou, kterou uživatel informuje ostatní o svém stavu je *online-user-message*. Tato zpráva chodí na topic popsaný v podkapitole 4.3.4. Tento typ zprávy také není nijak zpracováván serverem a ostatní uživatelé dostanou vždy poslední zprávu o aktivitě. V tabulce 4.6 můžeme vidět jednoduchou strukturu zprávy s časovým razítkem poslední aktivity.

Název	Typ
lastActivity	Timestamp

Tabulka 4.6: Zpráva MQTT online user message

4.4 Klient

Klientská část představuje aplikaci běžící na zařízení s platformou iOS. Uživatel může komunikovat online s ostatními uživateli a v případě, kdy aplikace běží na pozadí nebo je vypnuta, je informován o nových zprávách přes push notifikace. Klientská aplikace je schopna lokálně ukládat data pro možnost prohlížení offline dat, efektivní načítání multimédií a ošetření případné duplicity zpráv.

Po přihlášení klienta načte historii komunikací a ke každé synchronizuje s lokální databází definovaný počet nejnovějších zpráv. Při přístupu do historie konverzace nenacachované v lokální databázi, jsou data synchronizována přes rozhraní REST webového serveru. Toto rozhraní je dále použito i pro synchronizaci změn práv uživatelů nebo vytvoření nové hromadné komunikace. Pro zobrazení událostí v reálném čase využívá rozhraní MQTT brokeru.

4.4.1 Rozdělení aplikace

Vzhledem k poskytnutí možnosti využití jednotlivých prvků aplikace i v dalších aplikacích, je při implementaci využito rozdělení do balíčků (CocoaPods viz. kapitola 3.2) na:

- **Extensions** – Balíček rozšiřující chování jednotlivých základních datových typů i objektů. Rozšiřující metody je možné vyčlenit úplně zvlášť, protože znovupoužití tohoto balíčku je možné i u aplikací úplně jiného typu. Jako příklad můžeme uvést rozšíření datového typu *Date* o konstruktor vytvoření instance z textového řetězce s definovaným formátem datumu.
- **Components** – Komponenty k možnému využití i v dalších aplikacích. Například view pro zobrazení obrázku, nebo přehrání videa má využití nejenom v aplikaci messengeru.
- **Messenger** – Balíček specifických prvků pro messenger, všechna logika messengeru, modely apod.

- **Services** – Globální služby pro práci s lokální databází, MQTT klientem, internetovým připojením apod. Tyto služby je potřeba samozřejmě nakonfigurovat pro konkrétní aplikaci, jsou-li však implementovány obecně bez konkrétní závislosti lze je použít i v jiných typech aplikací.

Návrh rozdělení aplikace není specifický pro messenger, ale reflektuje obecné dělení prvků aplikace pro další využití. Při implementaci může dojít k potřebě dalšího dělení nebo naopak sjednocení některého balíčku, které vyplyne ze zvoleného návrhového vzoru a dalších zvolených metod implementace.

4.4.2 Uživatelské rozhraní

Návrh uživatelského rozhraní není jednoduchá a zanedbatelná věc, protože i vynikající aplikace se spoustou funkcí, která pro uživatele není dostatečně intuitivní a vzhledově atraktivní, je hodnocena špatně. Vytvořené uživatelské rozhraní se drží stylu aktuálních nejznámějších messengerů, které jsou prověřeny miliony uživateli. Návrh reflektuje zažité standardy platformy, jejich chování a zpětnou vazbu uživateli.

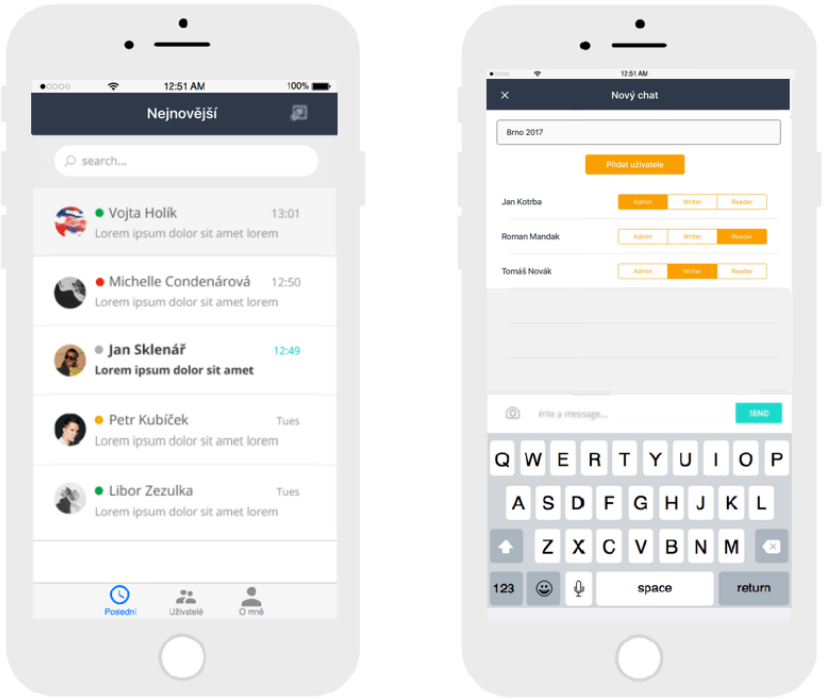
Navržené uživatelské rozhraní klientské aplikace využívá základní komponenty platformy iOS. Na obrázcích 4.5 a 4.6 můžeme vidět jeho základní návrh. Obrázky ukazují možnosti zobrazení na různých zařízeních.

Návrh na obrázku 4.5 vlevo nám zobrazuje seznam konverzací na menším zařízení iPhone. Každá položka seznamu zobrazuje uživatele v dané konverzaci, poslední zprávu a zvýrazňuje ji, nebyla-li přečtena nejnovější zpráva konverzace. Po kliknutí na konverzaci je zobrazen detail komunikace. Jako v každém seznamu i zde je možné vyhledávaní konverzací, případně zahájit novou. Ve spodní části přepnutím záložky se zobrazí seznam online dostupných uživatelů nebo informace o přihlášeném uživateli.

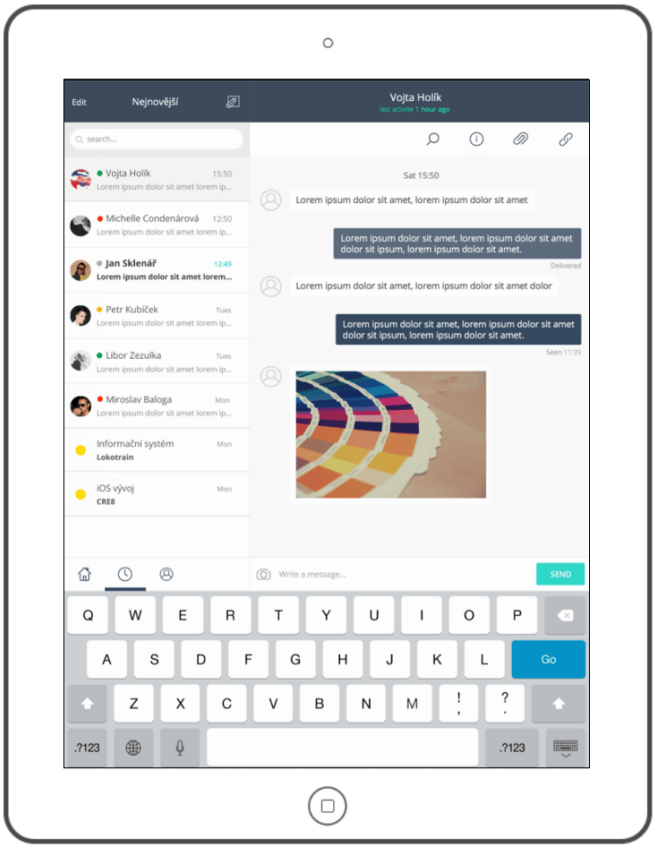
V pravé části obrázku je pak zobrazen návrh uživatelského rozhraní pro vytvoření místnosti. Pro dostupné tlačítko k vytvoření komunikace je potřeba zadat název místnosti, ze seznamu uživatelů vybrat alespoň dva. Těm lze zvolit, jak můžeme vidět v seznamu uživatelů jednu z rolí. Poté po vyplnění obsahu zprávy můžeme vytvořit novou místnost.

Další obrázek 4.6 ukazuje zobrazení komunikace na větším zařízení (iPad), kde jsou zároveň zobrazeny list komunikací i detail komunikace. Po kliknutí na určitou místnost se její detail zobrazí v pravé části. Na menších zařízeních je zobrazen list komunikací a detail zvlášť. Detail konverzace je složen ze záložek. Hlavní záložka, která je zobrazena jako první, je seznam zpráv. Každá zpráva může obsahovat i multimediální obsah, jeho detail je možné zobrazit po kliknutí na náhled. Zpráva může zobrazovat kromě obsahu také čas vytvoření. Další záložka zobrazuje informace o uživatelích v místnosti a jejich rolích. Ta je dostupná pouze u hromadné konverzace. Má-li uživatel dostatečné oprávnění, může ostatní uživatele nebo jejich role editovat a změny uložit. U komunikace „uživatel uživatel“ je zobrazena poslední aktivita uživatele přímo pod jménem uživatele. Na obrázku jsou naznačeny pro ukázku obě varianty (dostupný detail místnosti a poslední aktivita uživatele). Poslední záložka, která je dostupná vždy, je seznam souborů v komunikaci. Ta zobrazuje jednoduchý seznam s možností zobrazení detailu souboru stejně jako v seznamu zpráv.

Navrhnuté uživatelské rozhraní bylo diskutováno s potenciálními uživateli různých věkových skupin a zkušeností s mobilní platformou, kde některé nedostatky byly reflektovány. Tyto nedostatky byly však vizuálního vzhledu a designu. Reálná uživatelská zkušenost a hodnocení návrhu je až po implementaci prototypu, kde uživatel vyzkouší reálnou práci s aplikací.



Obrázek 4.5: Základní návrh uživatelského rozhraní zobrazení seznamu místností na zařízení iPhone 5S (vlevo) a vytvoření nové místnosti na zařízení iPhone 7 (vpravo)



Obrázek 4.6: Základní návrh uživatelského rozhraní na zařízení iPad

Kapitola 5

Implementace

Tato kapitola popisuje implementaci jednotlivých navrhnutých komponent a jejich chování. Snaží se dbát zásady možnosti znovupoužití jednotlivých komponent. Kapitola implementace se více zaměřuje na klientskou aplikaci a popis důležitých části. V úvodu kapitoly jsou popsány části serveru a dále pak detailněji implementace iOS aplikace.

5.1 Webový server

Webový server je rozdělen na moduly aplikace *Cloud*, *Messenger* a *Notifications*. Server je implementován v jazyce *PHP* a pro jednoduché sestavení projektu, je každý modul implementován jako balíček, který lze pomocí *composeru*[1] nainstalovat.

Jednotlivé moduly jsou implementovány dle návrhového vzoru MVC a využívají externí knihovny pro snadnější práci s databází, zpracování požadavků a různé abstraktní třídy zpřehledňující kód a odstraňující jeho duplicitu.

Webový server poskytuje klientské aplikaci rozhraní pro potřeby vykonání všech uživatelských akcí popsaných v kapitole 4.1. Navíc implementuje v podobě služeb i přístup ke komplexnějšímu přidávání uživatelů do konverzací, které pak mohou být využity při integraci do již existujících systémů. Například přidání uživatelů dle rolí z jiných aplikací získaných z tabulky *app* z obrázku 4.3. Tyto akce bývají většinou navázány na jiné aktivity uvnitř systému, proto jsou implementovány jako služby bez dostupnosti přes API.

Dále jsou popsány jednotlivé problematické části implementace serveru jako práce s multimediálními soubory a princip činnosti serveru pro efektivní vykonávání operací.

5.1.1 Multimédia

Pro práci s multimédii je využíváno standardní REST rozhraní, které soubor po úspěšném uploadování uloží a vytvoří k němu náhledy. Uložení podle parametru *protected* proběhne do obecně dostupné složky bez vyžadování autorizace nebo skryté složky s autorizací. V rámci odpovědi na detail dokumentu je pak poslán přímo odkaz na soubor nebo url, která uživatele přesměruje na ověření oprávnění přístupu k souboru.

Se zasílání obrázku, jak s ověřením přístupu, tak bez něj, nemají klientské aplikace problém, protože soubor je stahován celý najednou. Při stahování videa jde o stejný případ jako s obrázky, ale při streamování obsahu videa dochází v některých prohlížečích k problému s přehráváním.

[1] Composer – manažer závislostí pro jazyk PHP viz. https://getcomposer.org

Jedná se o problém s přehráváním videa bez přímého odkazu na soubor, tzn. s ověřením autorizace. Některé prohlížeče dokáží streamovat video bez serverové podpory částečného přenosu. Společnost Apple však používá svůj vlastní *QuickTime* přehrávač i v rámci prohlížeče nebo aplikací iOS, který částečný přenos od serveru vyžaduje. Zvláštní situace nastává u stejného přehrávače na jiné platformě (Windows), který toto pro přehrávání nepotřebuje.

Pro možnost streamování videa ze serveru s ověřením přístupu bylo potřeba implementovat částečné zasílání souboru serverem, kde pomocí hlaviček je možné si vyžádat pouze část souboru například v situaci, kdy při přehrávání videa vypadne internetové připojení a přehrávač poté žádá jen o dosud nestažená data.

Pro ověřování autorizace k souboru lze zaslat potřebné informace v hlavičce nebo také jako parametr. Soubory vyžadující oprávnění uživatele tak nelze získat jinou cestou, než s oprávněním tento soubor číst, protože cesta k souboru je dána logikou serveru, který tento soubor načte a začne uživateli posílat.

5.1.2 Vykonávání dotazů

Vykonávání jednotlivých dotazů přicházejících na REST rozhraní webového serveru by při provádění všech souvisejících operací prodlužovalo odezvu serveru a logicky by nesouhlasily zaslané informace. Například uživatel v aplikaci, který poslal novou zprávu, by dostal její potvrzení doručení později, než jiný uživatel push notifikaci o nové zprávě.

Pro rychlejší odezvu serveru a také logickou návaznost akcí byla využita fronta událostí. Vznikne li požadavek na vykonání dalších potřebných akcí v návaznosti na přijatý požadavek, je přesunut do fronty událostí. Tyto události jsou pak vykonávány jinými procesy. Pro rychlejší zpracování těchto navazujících aktivit jsou události ještě dále děleny a každý typ má svoji frontu. Může pak být spuštěno více procesů vykonávajících jednotlivé typy událostí pro rozdělení zátěže a minimalizaci zpoždění za hlavní událostí.

V situaci, kdy je vytvořena nová zpráva, je potřeba tuto zprávu uložit a rozeslat co nejrychleji dál. Zároveň však je vhodné publikovat MQTT zprávy na obecný notifikační topic všem uživatelům z konverzace a také rozeslat push notifikace. Zde je pak využito právě front, kde jedna slouží pro rozesílání MQTT zpráv a druhá pro zasílání push notifikací.

5.2 MQTT broker

Pro implementaci MQTT brokeru je využita knihovna *Mosca*[2], kterou lze jednoduše nainstalovat jako balíček. K výběru knihovny přispěla její podpora zasílání MQTT zpráv přes websockety viz. podkapitola 2.4, a tak umožňuje i vytvoření webového klienta pro tento messenger a jeho real-time komunikaci. Knihovna také podporuje šifrování komunikace a nabízí možnost přepsat výchozí metody, např. pro publikování zprávy, autentizaci uživatele apod.

Pro potřeby aplikace byly implementovány vlastní metody pro ověření uživatele, jeho autorizaci k publikování a odebírání zpráv k určitému topicu a také hlavní metoda rozesílání zpráv. Tyto metody v první fázi prováděly příkaz, který při každém spuštění opětovně inicializoval některé komponenty webového serveru a ty vyžadovaly další systémové zdroje. To se ukázalo při zaslání 30 požadavků za sekundu jako velmi neefektivní a velmi zatěžující server. Z tohoto důvodu bylo nahrazeno vykonávání command příkazu zasláním REST dotazu. Ten je vyřízen velmi rychle, protože má webový server většinu informací v cache, nedochází tak ani k zabírání dalších systémových zdrojů. Odpověď pro ověření uživatele je

[2]Mosca – MQTT broker in node.js viz https://www.npmjs.com/package/mosca

získána tedy standardním REST dotazem na server a pokud nedošlo k chybě, je uživateli povolen přístup nebo MQTT zpráva rozeslána dál. V některých případech, kdy jsou přidány další informace (např. čas vytvoření zprávy), dochází k nahrazení dat MQTT zprávy daty z odpovědi webového serveru.

Publikování každé nové zprávy v konverzaci předchází uložení informací do databáze, když jsou data v pořádku uložena, pokračuje broker rozesláním. Toto odstraňuje problém, kdy se například nepodaří uložit novou zprávu do databáze, uživatelé by ji však dostali přes MQTT rozhraní, ale v databázi by tato zpráva neexistovala. Nově připojení uživatelé do konverzace by danou zprávu nikdy neviděli. Navíc při ukládání dat jsou často přidány další informace do těla MQTT zprávy.

Protože využíváme QOS 1, MQTT broker vyžaduje vlastní rychlé úložiště pro ukládání informací k posílání zpráv uživatelům, kteří nejsou aktuálně připojeni, nebo čas expirace registrace odebírání zpráv uživatele k topicu. Z nabízených možností knihovny je použito lokální úložiště typu key-value *Redis*. Protože však není potřeba informace držet dlouho v úložišti brokeru, je nastavena expirace neodeslaných zpráv na 1 hodinu. V případě nastavení expirace doby k registraci odběru zpráv k topicu tak vyloučíme jeho stále odebírání zpráv i po odebrání oprávnění.

Pro další využití MQTT brokeru je snadné nakonfigurovat doménu webového serveru, port běžícího Redis úložiště a případně cestu k certifikátům pro šifrování komunikace.

5.3 iOS aplikace

V této podkapitole je detailněji popsána implementace iOS aplikace. Jsou zde vysvětleny koncept aplikace, způsob synchronizace dat s popsaným lokálním úložištěm a další důležité součásti vývoje.

5.3.1 Koncept

Při implementaci aplikace je potřeba brát v úvahu složitost její logiky, využití různých rozhraní pro získání informací, lokální ukládání dat a efektivní práci s multimédii. Návrh je inspirovaný návrhovým vzorem MVVM[3], který je vhodný vzhledem ke strukuře aplikace. Není to však čistá implementace dle MVVM, ale ViewModel zde obsahuje více logiky, tak aby model byl čistě statický objekt nesoucí pouze data.

Model zde představují objekty reprezentující data, které je možno jak uložit, tak načíst ze serveru nebo lokálně. ViewModel obstarává veškerou logiku pro zobrazení dat, aktualizuje model a připravuje informace pro zobrazení. Část View v koncepci iOS aplikace obsahuje view a view controllery pro vykreslování obsahu. Mimo koncept aplikace jsou globální služby zajišťující informace o internetovém připojení, dostupnosti MQTT, práci s lokálním úložištěm nebo obsluhu přihlašování a uchování dat o přihlášeném uživateli.

Model

Model v aplikaci reprezentuje všechny informace získávané ze serveru. Kopíruje strukturu odpovědí serveru a přidává položky potřebné k ukládání informací offline. V rámci ukládání

[3]Model View ViewModel viz https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52

přijatých dat ze serveru po zpracování JSON formátu hierarchicky vytvoří strukturu jednotlivých zanořených objektů. Tyto objekty jsou zároveň typem objektu, který lze lokálně uložit do databáze.

Lokální databáze reprezentující objekty vyžaduje úpravu logiky ukládání. Například primární klíč u některých položek, jako je zpráva, která nebyla ještě odeslána, je potřeba zvolit jiný. Zde je využito jednoznačného lokálního identifikátoru *uid*. Ten je vytvořen složením id místnosti a vygenerováním unikátního identifikátoru. Dalšími položkami, které je potřeba přidat ke struktuře přijaté ze serveru, jsou data pro uložení multimédií. Pokud je jednou obrázek stažen, uloží se lokálně a ušetří tak při dalším zobrazení opětovné stahování.

Model je tedy statická datová struktura, která nese pouze data. Detailnější popis a schéma lokální databáze najdeme v kapitole 5.3.3. Veškerou logiku obstarává ViewModel, který pracuje s načtenými daty a multimédii.

View

View reprezentují v aplikaci jednotlivá view a další typy controllerů pro různý způsob zobrazení. Aplikace ideálně zobrazuje informace pro všechny typy zařízení a v obvyklé velikosti, tzn. pro iPhony je obvyklý menší font, řádky tabulky jsou také menší apod.

Celá aplikace respektuje guidelines uživatelského rozhraní pro iOS. Základem je navigační controller zajišťující zobrazování login view nebo list komunikace po přihlášení. V případě messengeru se nabízí využití zobrazení komunikace na větších zařízeních v levé části list komunikací a vpravo detail komunikace. Při práci s více aplikacemi automaticky reaguje na šířku a zobrazuje stejně jako na menších zařízeních (iPhone) pouze jedno view (list komunikace nebo detail). V levé části view zobrazující list komunikací je možné přepnout do záložky s informací o aktuálně přihlášeném uživateli nebo záložky listu uživatelů.

Všechna tato zobrazovaná view jsou obsahem view controllerů nebo navigačních controllerů, kteří řídí flow aplikace, tzn. zobrazování nebo skrývání dalších view. View controllery i jednotlivé view tabulky je možné využít odděleně při nastavení potřebných dat pro zobrazení (ViewModelů).

Pro automatické překreslování view při změně dat je využito reaktivního programování To sice není při programování iOS aplikací běžné, ale tento návrhový model jej vyžaduje. Reaktivní programování poskytuje více možností navázání informací. V principu jsou data obalena do wrapperu, který při jejich změně vyšle signál a dané view reaguje překreslením nebo jinými akcemi.

ViewModel

Hlavní částí aplikace jsou ViewModely. Ty v aplikaci obstarávají tzv. bussiness logiku práce s daty, připravují data v požadovaném formátu pro zobrazení ve view. ViewModely jsou inicializované ve většině případů modely, až na ViewModely reprezentující například pole místností nebo uživatelů. Instance těchto ViewModelů může být vytvořena bez naplnění dat, protože jsou schopny si je získat z lokální databáze nebo serveru. Po získání dat vytvoří hierarchii ViewModelů podobnou hierarchii modelů.

Aby mohl být model generovaný a reprezentovat pouze statická data bez logiky, je logika přesunuta na ViewModel. ViewModel zahrnuje práci s lokálním úložištěm pro daný model. V případě real-time změny dat, zachytává MQTT zprávy a upravuje data modelu. U dat, kde je potřeba reagovat na změnu hodnoty, vytváří wrapper emitující událost o změně nebo přímo možnost bindování hodnoty na některou komponentu view. To představuje tzv. two-way data binding, kde změna hodnoty modelu se promítne do view a opačně.

Globální služby

Mimo koncept aplikace a jednotlivých Modelů a ViewModelů je potřeba služeb, které jsou mimo koncept MVVM tzv. globální služby. Ty jsou dostupné odkudkoliv v aplikaci a ve většině případů jsou implementovány jako singletony. U přístupu k lokální databázi to není úplně vyžadováno, ale u služby zajišťující MQTT připojení je to nezbytné. Dále si popíšeme jednotlivé služby využité v aplikaci, které mohou být převzaty bez komponent messengeru.

S dále uvedenými službami pracují ViewModely, které podle hodnot pak například volí zdroj získání dat. View pak přímo pracují s property bindovanými na dostupnost tlačítek. Například dostupnost internetového připojení ovlivňuje tlačítka vyvolávající akci s potřebným internetovým připojením.

UserService Informace o přihlášeném uživateli jsou potřeba získávat s globálním přístupem, aby programátor odkudkoli v aplikaci mohl získat například uživatelské jméno nebo jen základní informaci, zdali je uživatel přihlášen a dle toho zobrazovat informace. K tomu slouží služba *UserSevice*, která je do jisté míry podobná ViewModelu, protože obsahuje některé stejné property, ty při změně hodnoty emitují událost o této informaci. Například hlavní View po odhlášení automaticky zobrazí login view. Služba také zajišťuje persistenci přihlášeného uživatele i po vypnutí aplikace a jejím zapnutí.

RealmService *RealmSevice* je další služba obsluhující přístup k lokální databázi. U této služby se jedná hlavně o ukládání dat, jejich načítání nebo mazání celé databáze. Načtené objekty z databáze potřebují každou úpravu provádět v transakci, proto je zvolen přístup kopie objektu, která se může jakkoli upravit mimo transakci a poté vložit do databáze jako update. Nevyužijeme tak výhodu automatického aktualizace dat po úpravě, to ale při konceptu podobnému návrhovému vzoru MVVM není potřeba, protože zde upravuje a poskytuje data pro zobrazení ViewModel.

Jak je již z názvu zřejmé, pro lokální úložiště je zvolena Realm mobilní databáze, více popsána v kapitole 3.5. Ta nabízí jednoduchý a rychlý návrh databáze s rychlým vyhledáváním dat. Realm dokáže automaticky synchronizovat data s databází na serveru, ten tuto službu ale musí podporovat. To ale v našem případě není žádoucí, protože bychom se dostali k jednoúčelovosti aplikace.

ConnectionService Kontrola aktuálního stavu připojení k internetu, kde lze identifikovat i typ připojení, je žádanou hlavně u messengeru. V situaci, kdy je nedostupné připojení, není možné synchronizovat data se serverem, dochází k omezení funkcionality některých komponent a procházení dat offline. Služba nabízí property stavu připojení, na které je možné přímo bindovat změnu například dostupnosti tlačítka. Dále využívá notifikačního centra uvnitř aplikace, kde může podobným způsobem jako MQTT rozesílat na určitý topic informace nebo objekt o dostupnosti připojení.

Služba dokáže detekovat typ připojení, proto je možné stahovat větší soubory pouze při wifi připojení. Implementace služby využívá knihovnu *Reachability*[4] a vytváří nad ní jakýsi wrapper s vlastní přidanou funkcionalitou.

MqttService Hlavní služba pro real-time komunikaci obsluhuje připojení a všechen přenos přes MQTT rozhraní. Připojení k MQTT je možné různými způsoby nakonfigurovat

[4]Reachability – cocoapod viz https://cocoapods.org/pods/Reachability

například šifrování spojení, udržování keepAlive nebo opakované odesílání paketu apod. Tato služba využívá externí framework klienta MQTT *Moscapsule*[5], kde dle požadavků nastaví připojení ke klientovi a zachytává veškerou komunikaci s MQTT brokerem.

Na příchozí zprávy informující o úspěšném přihlášení k MQTT brokeru reaguje nastavením proměnných o připojení, stejně jako služba *ConnectionService* u dostupnosti internetového připojeni. Pro ostatní zprávy využívá stejného principu jako MQTT tedy publish/subscribe. Nově příchozí zpráva je přetypována na její očekávaný typ objektu. Poté je zaslána na stejný topic jako v rámci MQTT pomocí notifikačního centra uvnitř aplikace. Informaci o nové zprávě tak dostanou jenom komponenty, které tato informace zajímá.

Tato služba se musí vypořádat s neočekávanými stavy například se ztrátou internetového připojení a po jeho obnovení se znovu dostat do stavu před jeho výpadkem. Proto si pamatuje aktuálně všechny subscribnutí topiců a může tak jednoduše obnovit předchozí stav. Další chování služby při přechodech mezi jednotlivými stavy aplikace jsou popsáno v následující kapitole viz 5.3.2.

5.3.2 Životní cyklus aplikace

Jednoduché aplikace zobrazující pouze online data, nevyužívající žádné aktivity na pozadí nemusí reflektovat různé přechody mezi stavy aplikace a jejich životní cyklus reaguje pouze na aktivní stav aplikace. Tato aplikace může využívat nejenom úlohy spojené s aktivitou na pozadí, ale především využívá služby vyžadující změnu chování při přechodech mezi aktivním a neaktivním stavem. Nejprve si popíšeme chování aplikace při běžných situacích a poté v rámci možných režimů běžících na pozadí.

Při přechodu aplikace do aktivního stavu načte nejnovější data a služba obsluhující MQTT se přihlásí k odběru dat. Tyto akce probíhají i při změně výpadku stavu internetového připojení na dostupné. V situaci, kdy internetové připojení vypadne MQTT broker už nemá možnost odhlásit se korektním způsobem. Po připojení ale musí obnovit veškeré subscribnutí a všechny ViewModely musí synchronizovat aktuální data.

Akcemi uživatele, kdy aplikace přechází do neaktivního stavu, není prováděna persistence dat, protože ta se ukládají ihned po jejich obdržení, ale MQTT služba odhlásí odběr od všech topiců a odhlásí se od MQTT. Při neaktivním stavu je odebrán aplikaci přistup k internetu a proto by nastalo neočekávané odhlášení, kterému se vyhneme korektním způsobem.

Při přechodu aplikace zpět do aktivního stavu proběhne situace popsaná výše, ale při povolení načtení dat na pozadí pro aplikaci uživatelem, může dojít nejdříve k vykonání vlastní definované akce a poté k přechodu do aktivního stavu. Toho lze využít pro přednačtení dat a lepší uživatelské interakci s aplikací, kdy po jejím přechodu na popředí jsou již data aktualizována bez zpoždění.

Další redukcí obsahu načítání dat můžeme přispět stažením dat po příchodu push notifikace. Po jejím přijetí můžeme aktualizovat data v konverzaci příchozí zprávy a pak při synchronizaci dat již tato místnost bude mít nejaktuálnější informace. Pro provedení akce po příjetí push notifikace je také potřeba povolení uživatele.

5.3.3 Lokální databáze

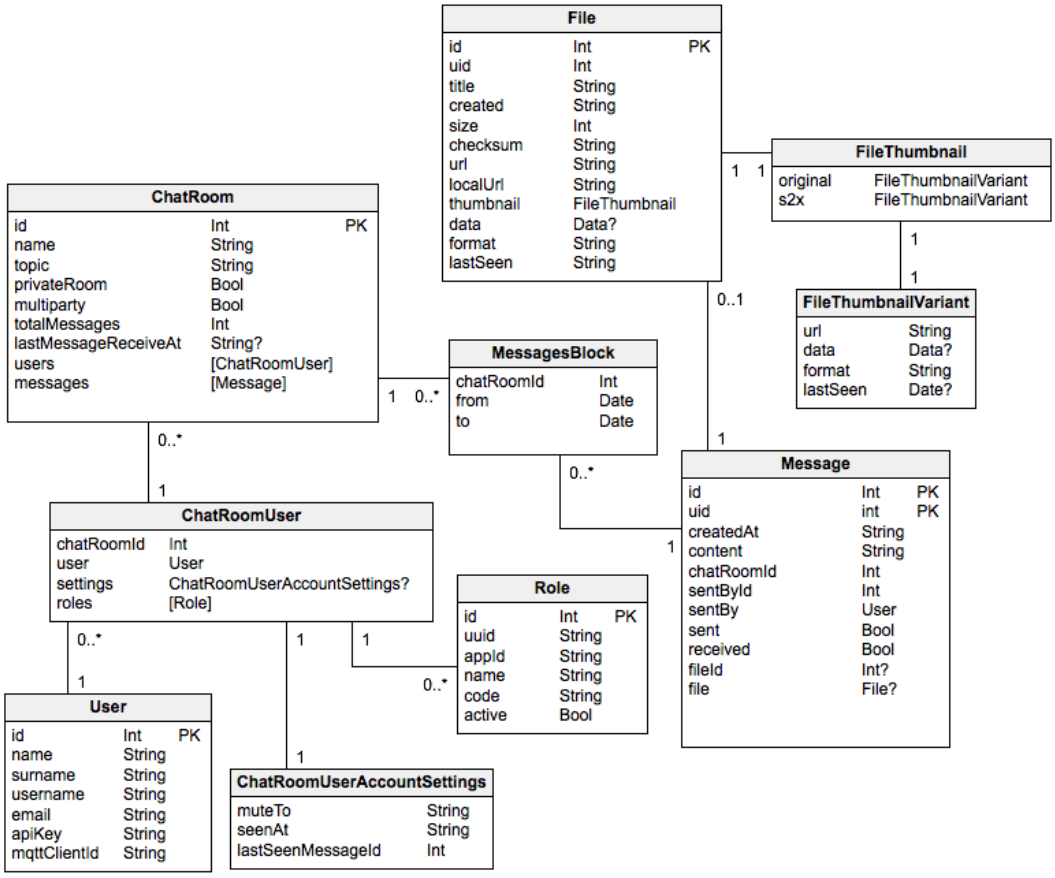
Schéma lokální databáze můžeme vidět na obrázku 5.1. Lokální databáze je odlišná od databáze na serveru, ukládá data ve formátu, v jakém byla obdržena ze serveru. K těmto

[5]Moscapsule – viz https://github.com/flightonary/Moscapsule

datům přidává další položky, které jsou nutné pro logiku nad daty uloženými lokálně na zařízení nebo odstraňují redundantní stahování dat.

Ve schématu můžeme vidět, že některé informace oproti serveru jako skupiny uživatelů nejsou zde vůbec potřeba ukládat. Naopak je přidána tabulka *MessagesBlock* pro synchronizaci zpráv na zařízení, také jsou přidány položky pro data stažených souborů. Ty, pokud nejsou dostupná lokálně, jsou stažena, ale nemusí se stahovat při každém zobrazení.

Při implementaci aplikace se pak pracuje s každým záznamem v tabulce lokálního úložiště jako s objektem. Rozšiřuje základní objekt Realm a definuje vlastní proměnné dle základních typů nebo jiných stejně definovaných objektů jako například *File* objekt v *Message*. Vyhledávat v databázi lze velmi snadno, u některých objektů není žádná položka primárního klíče nutná, protože lze vyhledávat i přes položky obsaženého objektu, například objekt *ChatRoomUser* lze vyhledat podle položky *user* a jejího *id*.



Obrázek 5.1: Schéma lokální databáze klienta

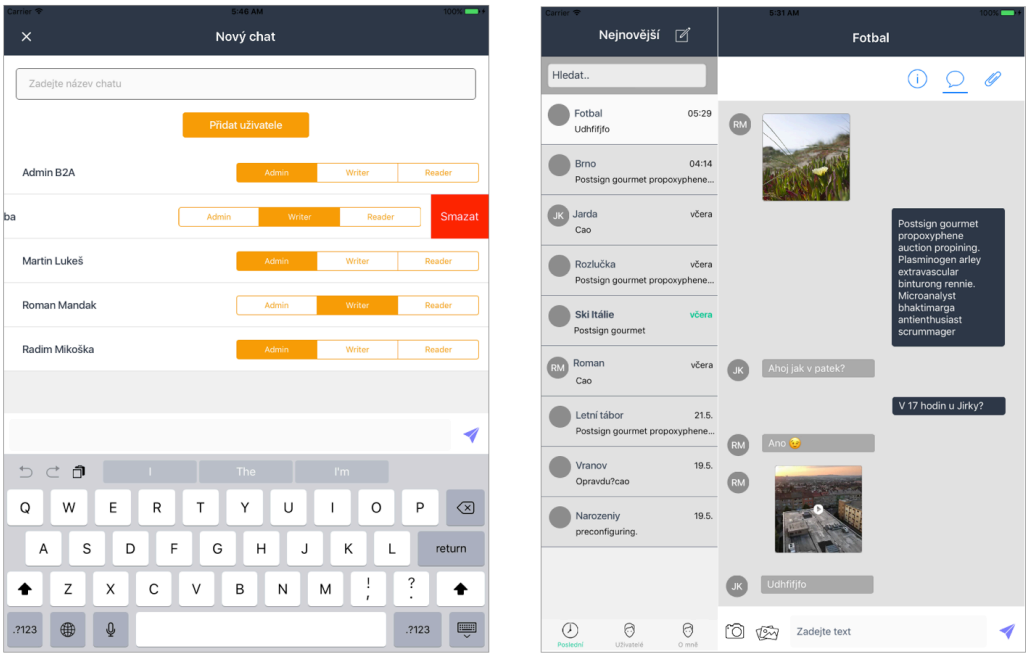
I když zařízení mají v dnešní době velké kapacity úložiště, je zbytečné držet historická data, proto jsou v messengeru průběžně promazávána stará. Speciální případ uložených dat jsou soubory, které zabírají mnohonásobně více místa. Ty jsou v aplikaci zobrazovány v detailu zprávy, ale i samostatně v seznamu k dané místnosti. Při zobrazení detailu a načtení dat je uložena informace o posledním zobrazení a mazání probíhá na základě této informace. Pokud tato informace není k dispozici dochází k mazání spolu se zprávou.

5.3.4 Rozdělení View

Aplikace iOS a její jednotlivé View byly při implementaci rozděleny na komponenty, které lze opakovaně použít ať už v rámci messenger (komponenty závislé na konkrétních typech messengeru) nebo obecně i v jiných typech aplikací například input zprávy. Na obrázku 5.2 můžeme vidět realizaci uživatelského rozhraní.

Můžeme zde vidět komponentu pro zadání nové zprávy (obrázek vpravo v pravé dolní části), která je použita i při vytváření nové místnosti. Ta může obsahovat další komponentu pro zobrazení náhledu obrázku nebo videa, posílaného jako přílohu ke zprávě. Komponenta pro náhled souborů je také použita v detailu zprávy a také v seznamu příloh.

Znovupoužité komponenty v jiných situacích mohou vyžadovat jiný typ zobrazení nebo omezení některých akcí. Například komponenta pro vytvoření zprávy nemusí zobrazit tlačítka pro výběr přílohy (obrázek 5.2 vlevo). Konkrétní chování a zobrazení je možné definovat nastavením odpovídajících konfiguračních proměnných, které jsou pak reflektovány při vykreslení.



Obrázek 5.2: Implementace uživatelského rozhraní vytváření nové hromadné konverzace (vlevo), zobrazení seznamu místností a jejich detailu na zařízení iPad (vpravo)

Příklad komponenty, která může být využita v úplně jiném typu aplikace, je tabBar. Naopak jednotlivé View zobrazované tabBarem (detail místnosti, seznam souborů, konverzace) lze použít jen v kontextu messengeru. Jednotlivá okna lze však použít zvlášť, tedy pokud uživatel nechce zobrazit seznam souborů v další záložce, může zakomponovat do vlastní aplikace jenom seznam zpráv.

Jednotlivé komponenty závislé na datových typech messengeru jsou uloženy do jiného CocoaPodu než ty, co jsou nezávislé na messengeru.

5.3.5 Synchronizace dat

Synchronizace dat uložených lokálně na zařízení s aktuálními daty na serveru může být implementována několika způsoby. Od těch více datově náročných bez jakékoli logiky až po ty propracovanější s minimálním stahováním redundantních dat. U této aplikace bylo snahou dosáhnout více sofistikovanější synchronizace a způsobu ukládání dat lokálně. Princip můžeme rozdělit na synchronizaci a správné zobrazení seznamu místností a zpráv.

Místnosti

Synchronizace konverzací je výrazně jednodušší. Po přihlášení načteme seznam nejnovějších konverzací s určitým počtem nejnovějších zpráv, stejně tak nemáme-li lokálně k dispozici žádné záznamy o místnostech k načtení. V ostatních případech načítáme pouze informace o místnostech obsahující čas poslední přijaté zprávy. Tato položka je pro nás důležitá z hlediska řazení místností. Nemusíme tak načítat i informace o zprávách, které by mohly být redundantní. Může nastat situace, kdy obdržíme data o místnosti, kterou nemáme lokálně uloženou, poté místnost uložíme lokálně a načtení zpráv proběhne způsobem popsaným dále.

Zprávy

Pro efektivní ukládání zpráv v místnosti jsou navazující zprávy seskupeny do bloků. Příchozí zprávy ze serveru můžeme považovat jako správně seřazený souvislý blok, který lokálně uložíme. V případech, kdy požádáme o data od určitého bloku nebo se nám bloky zpráv překryjí, je můžeme sloučit. Při dotazování na nové zprávy nebo načítání starých, pracujeme s lokální databází a žádáme jen zprávy, které nemáme uloženy. Situace, které mohou nastat vysvětlíme na konkrétních příkladech.

V prvním příkladu máme zobrazeny zprávy z aktuálního bloku a uživatel chce zobrazit starší zprávy. Pokusíme se je načíst z lokálního úložiště aktuálního bloku, pokud však blok uložených zpráv skončil, pošleme žádost o načtení starších zpráv mezi časem nejstarší zprávy aktuálního bloku a nejnovější zprávy následujícího bloku, pokud existuje. Těchto dat může být samozřejmě mnoho, proto využijeme stránkování, kdy dostaneme nový blok zpráv. Dorazí-li nám menší počet zpráv než limit jaký jsme žádali, můžeme tyto tři bloky sloučit, jinak sloučíme pouze aktuální blok s nově načteným. Dojdeme-li na poslední zprávu sloučeného bloku, opět tuto akci opakujeme.

V další situaci zobrazíme list konverzací a chceme mít pro každou zobrazenou místnost synchronizovány poslední zprávy. Zdali máme načtena nejaktuálnější data, zjistíme tak, že se pokusíme najít blok zpráv s časem poslední zprávy shodný s časem nejnovější zprávy v místnosti. Pokud jej nenalezneme, vyhledáme nejnovější blok zpráv a načteme pouze zprávy novější než poslední zpráva tohoto bloku. Nejsou-li žádné zprávy v místnosti, tzn. datum nejnovější zprávy je prázdný, tuto akci neprovádíme.

Jakmile se uživatel přihlásí, načte nejnovější konverzace ze serveru a k nim nejnovější zprávy. Mezi načtením konverzací a přihlášením k odběru MQTT zpráv může nastat velmi krátká časová mezera, kdy mohou být vytvořeny nové zprávy, které uživatel nezíská REST dotazem a ani MQTT zprávou. Pro tento případ existuje pro každého uživatele obecný

notifikační topic v rámci messengeru, ke kterému se ihned po úspěšnému přihlášení zaregistruje k odebírání zpráv. Na tento topic jsou zasílány notifikace o všech nových zprávách v místnostech, ve kterých má uživatel právo alespoň číst. K uvedené chybě tak nedojde, protože informace o nové zrpávě získá uživatel z jeho obecného topicu v rámci messengeru.

Výše popsané komplexnější ukládání a synchronizace zpráv jsou i přes některou logiku v rámci tříd ViewModelu vyčleněny do samostatné synchronizační služby. Ta tento ViewModel obsahuje, stará se o logiku zobrazování a ukládání jednotlivých zpráv a poskytuje mu rozhraní, přes které požadované akce volá.

Kapitola 6

Závěr

Cílem práce bylo prozkoumat dostupné komunikační protokoly umožňující zasílání zpráv mezi klienty v aplikacích typu klient-server. V kapitole 2 jsou analyzovány vybrané protokoly. Dalším úkolem bylo seznámit se s možnostmi vývoje aplikací na platformě iOS. Důležité části vývoje jsou pak shrnuty v kapitole 3. Ze získaných informací o komunikačních protokolech a platformě iOS byla navrhnuta ve 4. kapitole implementace jednotlivých částí aplikace. Kapitola na úvod představuje možné uživatelské akce v aplikaci a celkový návrh struktury systému. Dále jsou pak detailněji popsány principy komunikace a synchronizace dat, jednotlivé komponenty serverové části, návrh uživatelského rozhraní klientské aplikace a její rozdělení. Realizace navržené aplikace klient-server je popsána v kapitole 5, kde je blíže vysvětlen koncept iOS aplikace, její důležité části a vlastnosti. K serverové části aplikace je obecně popsána podstata realizace, princip zpracování požadavků, způsob ověřování a publikování zpráv MQTT brokerem.

Aplikace umožňuje skupinovou komunikaci uživatelů a posílání multimédií jako obrázky a videa. Interakce s uživatelským rozhraním zachovává doporučené standardy platformy. Způsob, jakým je aplikace navrhnuta a implementována, dovoluje jednoduše dále rozvíjet jednotlivé moduly serveru i konkrétní komponenty uživatelského rozhraní.

Při integraci do existujících systémů je možné jednotlivým částem přizpůsobit chování nebo přidat vlastní funkcionality. Další možností rozvoje aplikace může být přidání další konfigurace chování existujících částí nebo vytvoření jiné potřebné funkčnosti. Z hlediska uživatelského rozhraní je možné upravit vzhled nebo realizovat nové atraktivní prvky (animace) a otestovat jejich přínos pro uživatele při práci s aplikací.

Literatura

[1] A. Freier, P. Karlton, P. Kocher: The Secure Sockets Layer (SSL) Protocol Version 3.0. 2011, [Online; navštíveno 20.12.2016].
URL https://tools.ietf.org/html/rfc7622

[2] Apple Inc.: *About the iOS Technologies*. 2014, [Online; navštíveno 3.1.2017].
URL
https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html

[3] Apple Inc.: *File System Programming Guide*. 2016, [Online; navštíveno 3.1.2017].
URL https://developer.apple.com/library/content/documentation/FileManagement/Conceptual/FileSystemProgrammingGuide/FileSystemOverview/FileSystemOverview.html

[4] Apple Inc.: *Background Execution*. 2017, [Online; navštíveno 3.1.2017].
URL https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BackgroundExecution/BackgroundExecution.html

[5] Apple Inc.: *The App Life Cycle*. 2017, [Online; navštíveno 3.1.2017].
URL https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html

[6] David Mark, Kim Topley, Jack Nutting, Fredrik Olsson, JEFF LAMARCHE: *Beginning iPhone Development with Swift 2: Exploring the iOS SDK*. Apress, 2015, ISBN 1484217543.

[7] E. Rescorla, N. Modadugu: Datagram Transport Layer Security Version 1.2. 2012, [Online; navštíveno 20.12.2016].
URL https://tools.ietf.org/html/rfc6347

[8] Fette, I.; Melnikov, A.: The WebSocket Protocol. Září 2011, [Online; navštíveno 20.12.2016].
URL https://tools.ietf.org/html/rfc6455

[9] Hagop Panosian: *Learn iOS Application Distribution: Successfully Distribute Apps*. Apress, 2017, ISBN 1484226828.

[10] HiveMQ: *MQTT essentials*. [Online; navštíveno 18.12.2016].
URL http://www.hivemq.com/blog/mqtt-essentials/

[11] IBM: *MQTT V3.1 Protocol Specification*. 2010, [Online; navštíveno 3.1.2017].
URL http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/MQTT_V3.1_Protocol_Specific.pdf

[12] Languedoc, K.: *Build iOS Database Apps with Swift and SQLite*. Apress, 2016, ISBN 978-1-4842-2232-4.

[13] M. Belshe, R. P.; M. Thomson, E.: Hypertext Transfer Protocol Version 2 (HTTP/2). Květen 2015, [Online; navštíveno 20.12.2016].
URL https://tools.ietf.org/html/rfc7540

[14] Marius Rackwitz, S. G., Orta Therox: *CocoaPods*. [Online; navštíveno 18.12.2016].
URL https://cocoapods.org

[15] Michael Privat, Robert Warner: *Pro iOS Persistence*. Apress, 2014, ISBN 978-1-4302-6029-5.

[16] R. Peon, H. Ruellan: HPACK: Header Compression for HTTP/2. Květen 2015, [Online; navštíveno 20.12.2016].
URL https://tools.ietf.org/html/rfc7541

[17] Saint-Andre, P.: Extensible Messaging and Presence Protocol (XMPP): Core. 2011, [Online; navštíveno 20.12.2016].
URL https://tools.ietf.org/html/rfc6120

[18] Saint-Andre, P.: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. 2011, [Online; navštíveno 20.12.2016].
URL https://tools.ietf.org/html/rfc6121

[19] Saint-Andre, P.: Extensible Messaging and Presence Protocol (XMPP): Address Format. 2015, [Online; navštíveno 20.12.2016].
URL https://tools.ietf.org/html/rfc7622

[20] T. Dierks, E. Rescorla: The Transport Layer Security (TLS) Protocol Version 1.2. 2008, [Online; navštíveno 20.12.2016].
URL https://tools.ietf.org/html/rfc5246

[21] Valerie Lampkin, L. O., Weng Tat Leong; aj.: *Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry*. IBM, Září 2012, ISBN 0738437085.

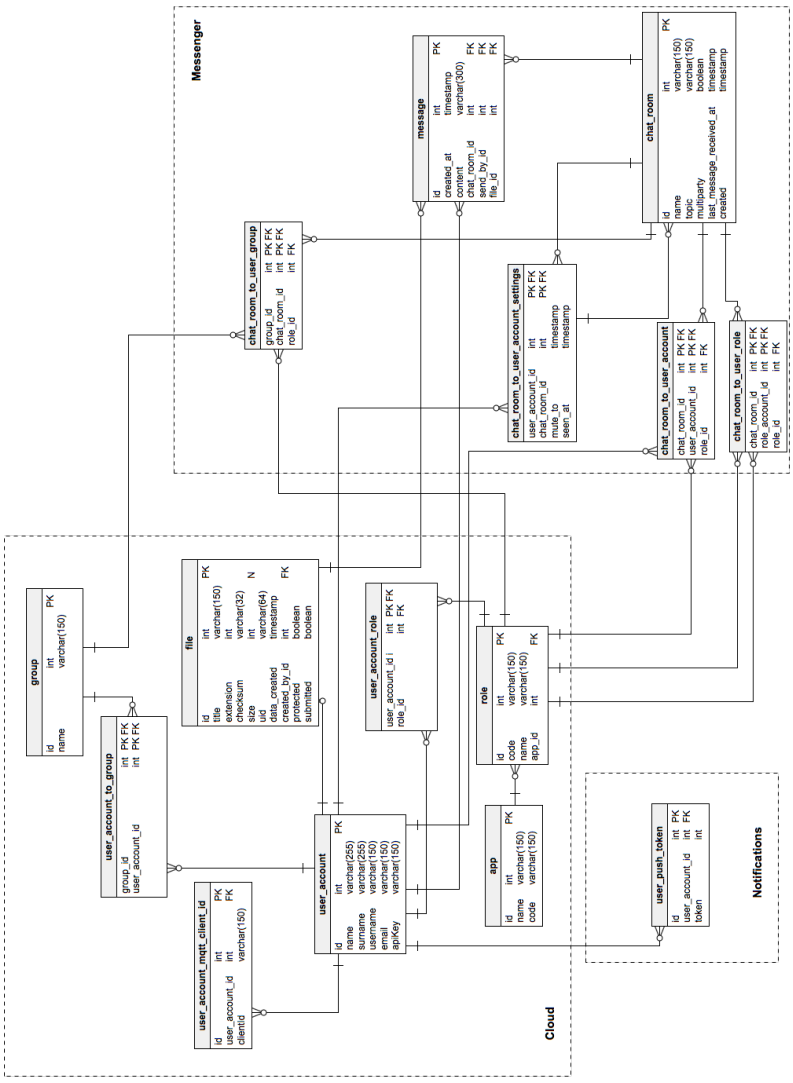
[22] XMPP: Oficiální stránky. [Online; navštíveno 20.12.2016].
URL http://xmpp.org/

[23] Z. Shelby, K. H.; Bormann, C.: The Constrained Application Protocol (CoAP). [Online; navštíveno 20.12.2016].
URL https://tools.ietf.org/html/rfc7252

Přílohy

Příloha A

Databázové schéma



Obrázek A.1: Návrh databázového schématu pro messenger

Příloha B

Obsah CD

- Manuál k zprovoznění **readme.txt**
- Tato práce ve formátu pdf
- Zdrojový kód serverové části v adresáři **/src/server**
- Zdrojový kód iOS aplikace v adresáři **/src/app**
- Zdrojové kódy tohoto dokumentu v adresáři **/tz_src**