Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Mobilní aplikace využívající principy Klient-Server a Peer-to-Peer komunikace

Bakalářská práce

Autor: Tomáš Hovorka
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Malý Filip, Ph. D.

Hradec Králové duben 2018

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 23.4.2018 Tomáš Hovorka

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Ing. Filipu Malému, Ph.D. za metodické vedení práce a vstřícnou komunikaci.

Anotace

Bakalářská práce se zaměřuje na vytvoření aplikace, která umožňuje výměnu a přenos dat napříč různými klientskými aplikacemi pomocí architektury Klient-Server a Peer-to-Peer. V práci je implementována webová služba, která komunikuje s klientskými aplikacemi a databází. Dále pak klientská aplikace pro platformu Android, která umožňuje komunikaci s webovou službou, nebo připojení na jinou klientskou aplikaci pomocí technologie Bluetooth.

Annotation

Title: Mobile applications using principles Client-Server and Peer-to-peer communications

The bachelor thesis focuses on the creation of an application that allows the exchange and transfer of data across different client applications using Client-Server and Peer-to-Peer architecture. In bachelor thesis is implemented a web service, that comunicates with client applications and a database. Additionally, an Android based client application that allows communication with a web service or connect to another client application via Bluetooth technology.

Obsah

Seznam obrázků

Seznam zdrojových kódů

Klíčová slova

Program, aplikace, webová služba, WCF, Android, Klient-Server, Peer-to-Peer

1 Úvod

V dnešní době využívá většina lidí na světě chytrý mobilní telefon nebo tablet. Tato zařízení nám ulehčují nebo zpříjemňují každodenní práci. Jejich cena je stále příznivější, a proto je trh s touto elektronikou na velkém vzestupu. Pro jejich využívání je zapotřebí velké množství programů, které nám umožní je jednodušeji a efektivněji využívat. Velkým trendem je sdílení dat napříč různými zařízeními a aplikacemi. Cílem bakalářské práce je navrhnout a implementovat mobilní aplikaci, která bude komunikovat s jinou aplikací pomocí vytvořeného serveru, nebo přímo za pomoci připojení pomocí architektury Peer-to-Peer.

2 Mobilní aplikace

Mobilní aplikace je počítačový program, který umožnuje uživateli mobilního zařízení komunikaci s funkcemi daného elektronického přístroje. Některé mobilní aplikace jsou do zařízení nainstalovány defaultně výrobcem již před zakoupením telefonu, jako je například aplikace pro psaní SMS zpráv, vytáčení kontaktů nebo videokamera, fotoaparát a podobně. Pro využívání jiných je však potřeba dané aplikace stáhnout z jiného zdroje. Mezi takovéto zdroje patří například Google Play, App Store, Microsoft Store a jiné. Jedná se o distribuční služby, které umožní uživateli zakoupit instalační soubor, který se stáhne a nainstaluje mobilní aplikaci uživateli do telefonu. Ten ji následně může začít využívat. Zpravidla všechny mobilní aplikace v dnešní době komunikují se servery nebo jinými mobilními aplikacemi například pomocí internetu, Bluetooth nebo GPS a dalších způsobů, z důvodu výměny informací, zjištění polohy atd.

2.1 Mobilní operační systémy

Ne každá mobilní aplikace může fungovat na všech mobilních zařízeních. Každá je totiž napsaná pro skupinu nebo pro konkrétní platformu. Pod pojmem platforma si můžeme představit operační systém mobilního zařízení. Mezi základní operační systémy patří Android, iOS a Microsoft Windows Phone.

2.1.1 Android

Android je v dnešní době nejvyužívanější mobilní operační systém, který je založen na Linuxu. Původním majitelem byla firma Android Inc., která ho v roce 2005 prodala firmě Google a. s. Ta ho i nadále vyvíjí. Verze tohoto operačního systému nesou vždy název podle sladkostí. V dnešní době je nejaktuálnější verzí systému verze 7.1 Nougat, která je ovšem zatím spuštěna pouze na mobilních zařízeních od společnosti Google. Pro většinu zařízení je zatím dostupná verze nižší [1]. Výhodou Androidu je jeho otevřenost. Tento systém je založen na principu open source a není tedy nutné za jeho využívání platit. Je proto možné si ho nainstalovat a bezplatně využívat na jakémkoliv zařízení. Aplikace jsou k dispozici především přes internetový obchod Google Play. Tato distribuční služba obsahuje k dnešnímu dni, podle serveru Statista, více než tři miliony aplikací [2].

Spousta aplikací je šířená bezplatným způsobem, jiné jsou přístupné až po jejich zakoupení. Aplikace na Google Play může nahrávat jakýkoliv uživatel. Existuje proto velké množství aplikací, které jsou nefunkční, nebo nejsou odladěné pro značné množství mobilních a tabletových zařízení.

2.1.2 iOS

IOS neboli také iPhone OS je v dnešní době druhý nejpoužívanější operační systém pro mobilní zařízení, hned po systému Android. Jedná se o systém založený na systému macOS od společnosti Apple Inc., která jej vlastní i vyvíjí. Tento systém mohou využívat pouze zařízení od společnosti Apple. V dnešní době je nejaktuálnější verzí systému iOS 10, kterou využívá většina zařízení, a je připravována verze iOS 11. Jedná se o uzavřený systém, u kterého je zapotřebí pro konektivitu s ostatními zařízeními využívat jeho vlastní služby a aplikace, jako je například iTunes nebo iCloud. Uživatelé mohou přistupovat k novým aplikacím pomocí služby App Store, která je nativně připravená v zařízeních při jejich zakoupení. Nahrání nové aplikace do App Store je značně obtížnější oproti konkurenčnímu Google Play. Každá aplikace před nahráním prochází složitým schvalovacím procesem, při kterém je řádně otestována, aby se zamezilo distribuci špatných, nebo dokonce nefunkčních aplikací všem uživatelům. Tento proces má za následek nižší počet nabízených aplikací. Aplikace, které úspěšně prošly nahrávacím procesem, jsou k dispozici zdarma, nebo po jejich zakoupení.

2.1.3 Microsoft Windows Phone

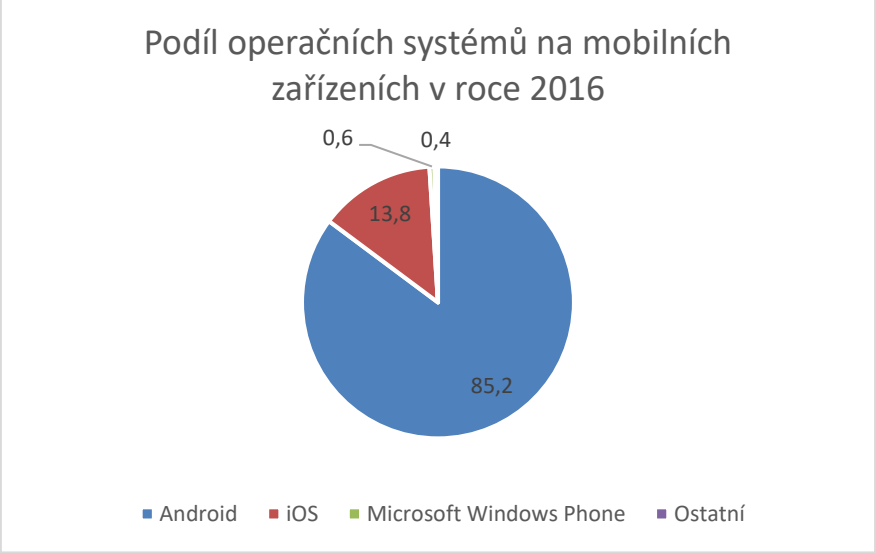
Microsoft Windows Phone je operační systém vlastněný a vyvíjený firmou Microsoft Corporation. Je to třetí nejrozšířenější systém na mobilních zařízeních. Dnes je nejaktuálnější verzí systému verze Windows 10 Mobile. Aplikace na tento typ operačního systému je možné získat pomocí služby Store, kterou obsahuje nativně každé zařízení. Počet aplikací vyvíjených pro tuto platformu je výrazně nižší než u systému Android nebo iOS. Tento operační systém je využíván především firmami jako služební telefony.

Dlouhodobým cílem je zpřístupnit uživatelům funkce desktopových zařízení pomocí mobilního zařízení. Dokazuje to například možnost vzdálené plochy

k počítačům bez instalace externích aplikací, dále také vývoj funkce Continuum, která umožní uživatelům po připojení většího zobrazovacího zařízení k mobilnímu telefonu využívat funkce desktopových aplikací. Jedná se tedy i o vývoj nových mobilních aplikací, které budou pracovat v mobilním i desktopovém režimu. Těchto aplikací zatím není mnoho, jde především o základní aplikace firmy Microsoft, jako je například balíček Office, internetový prohlížeč Internet Explorer a podobně.

2.1.4 Využití operačních systémů

Podle analýzy podílu operačních systémů na trhu od společnosti Business Insider v roce 2016 využívalo 85.2 % všech mobilních zařízeních na trhu operační systém Android. Na druhé místo v počtu smartphonů byl zařazen systém iOS, který využívalo 13.8 % uživatelů. Microsoft se svým systémem velmi zaostával. Využívalo ho přibližně 0.6 % uživatelů. Celkové pokrytí spotřebního trhu systémy Android a iOS tedy činí 99 %. Z jejich výsledků jasně vyplývá, že počet uživatelů využívajících systém Android se v letech 2009 až 2016 výrazně zvýšil. To může být způsobeno dostupností tohoto operačního systému pro velké množství zařízení. Apple si v tomto období udržuje přibližně stejné procento uživatelů. Procentuální využití operačních systémů na mobilních zařízeních v roce 2016 je znázorněno na Obrázku 1. [3]



Obrázek 1: Podíl operačních systémů na mobilních zařízeních v roce 2016 (zdrojem autor)

2.2 Vývoj mobilních aplikací

Vývoj mobilních aplikací je poměrně náročná záležitost. Je potřeba na začátku provést důkladnou analýzu systému, cílové skupiny uživatelů a také způsobu implementace řešení. V dnešní době se mobilní aplikace vyvíjejí třemi základními způsoby. Jedná se o nativní mobilní aplikace, hybridní aplikace a responzivní webové stránky.

Nativní mobilní aplikace je aplikace napsaná přímo pro konkrétní platformu. Ostatní systémy tuto aplikaci nebudou schopny využívat. Tento způsob aplikací přináší dosažení vysokého výkonu. Pro vývojáře je navíc velice jednouché využívat základní prvky zařízení a systému, jako je například GPS, videokamera, senzory snímající světlo nebo pozici zařízení a dále pak také základní prvky využívání zařízení, jako jsou například pozice doteku, gesta a podobně. Vývoj takových aplikací bývá ovšem finančně nejnáročnější, jelikož pro pokrytí co největšího počtu uživatelů je potřeba vyvíjet tuto aplikaci pro každou platformu zvlášť. V dnešní době je možné využívat nástroje pro tvorbu multiplatformních nativních aplikací. Jedná se o nástroje, ve kterých vyvineme danou aplikaci, a ta je následně kompilovaná do ostatních platforem. Zpravidla ale musí být zdrojový kód jednotlivých aplikací následně poupraven, neboť využití všech funkcí na jiném zařízení není zcela zaručeno. Mezi takovéto nástroje patří například Xamarin od společnosti Microsoft, Unity 3D pro vývoj her, také od společnosti Microsoft, nebo Cocos2d od společnosti Chunkong Technologies, který je také především využíván jako nástroj pro tvorbu her.

Další možností je hybridní aplikace. Jedná se o aplikaci, která na první pohled vypadá jako nativní aplikace, avšak na pozadí využívá internetový prohlížeč daného operačního systému, který každý obsahuje. Data následně zobrazuje ve vlastní aplikaci. Tato možnost aplikace začíná být vývojáři stále více oblíbená. Hlavním důvodem je multiplaformnost. Vývoj takové mobilní aplikace není tolik finančně náročný, protože se aplikace vyvine pouze jednou, což ušetří čas při úpravě, nebo novém vytváření pro ostatní platformy. Problémem těchto aplikací je nižší výkon, možná nedostupnost základních funkcí telefonu nebo využití základních prvků telefonu.

Třetí alternativou je poté responzivní webová stránka, ke které se v zařízeních přistupuje pouze pomocí internetového prohlížeče. Výhodou těchto aplikací je jejich multiplatformnost a nabídka pro uživatele, kteří nemusí stahovat žádný instalační balíček od distribučních služeb. Oblíbenost těchto aplikací bývá u uživatelů zpravidla nižší než u nativních nebo hybridních aplikací.

2.2.1 Vývoj nativní mobilní aplikace

Aplikace pro Android se vyvíjí v jazyce Java. Je tedy zapotřebí nainstalovat balíček pro vývoj Java aplikací JDK (Java Development Kit) a sadu nástrojů SDK (Software Development Kit) od Androidu. Primárním vývojovým prostředím je Eclipse, pro který je možné doinstalovat ADT. Jedná se o sadu nástrojů, která usnadňuje vývoj. V dnešní době je možné vyvíjet také například v IDE NetBeans nebo Intellij od verze 10. Samotné SDK obsahuje pouze několik základních programů. Nejdůležitějším je Android SDK a AVD Manager. Pomocí něj může vývojář vytvářet a spouštět virtuální Android zařízení, na kterém je poté možné spouštět a testovat aplikace, nebo například stahovat další komponenty, jako jsou další verze systému Android pro testování a podobně. Dalším důležitým programem je program Dalvik Debug Monitor, pomocí kterého se dají aplikace ladit. Program umí pracovat jak s virtuálními instancemi emulátoru, tak také s připojeným telefonem. Samotné SDK obsahuje tyto balíčky.

- **SDK Platforms** – jedná se o sadu verzí Androidů. Některé jsou v základním balíčku, jiné je potřeba si ručně dostahovat.
- **Android SDK Tools** – jsou nástroje pro vývoj, testovaní, ladění a také emulátor pro spouštění.
- **Google API**

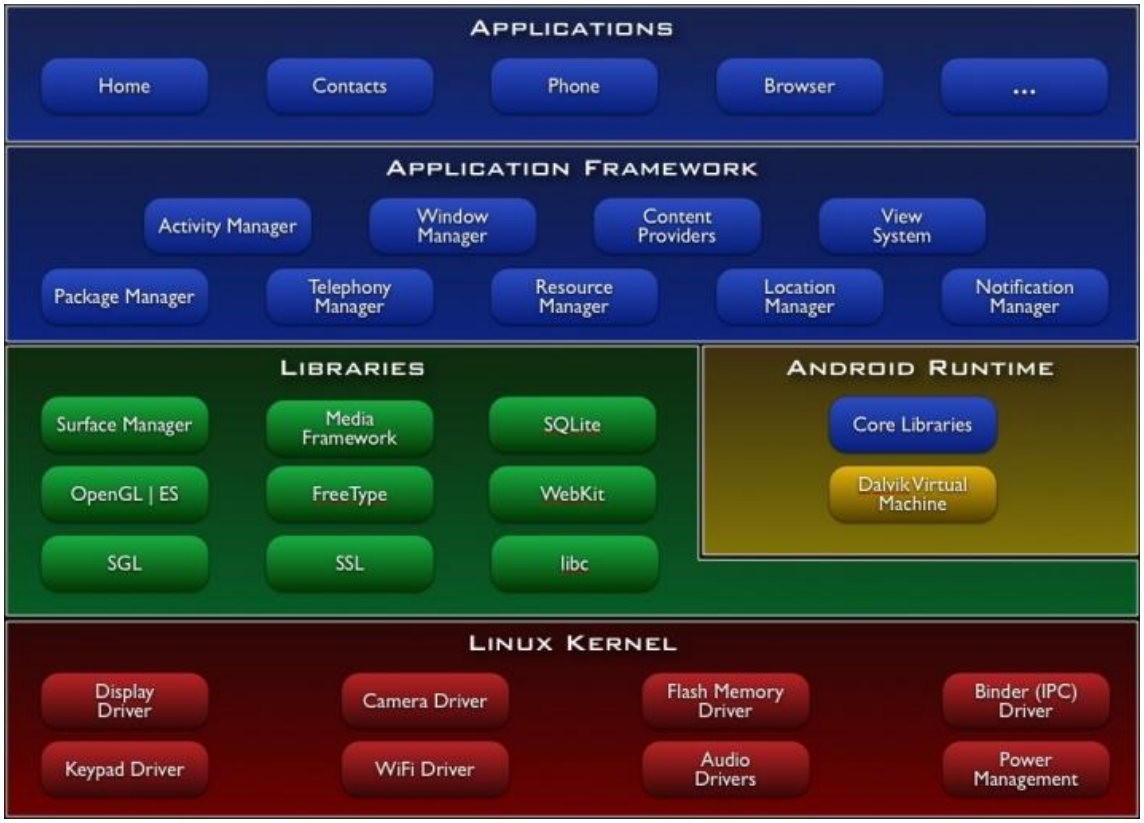
[4]

3 Základní prvky Androidu a jeho aplikací

Následující kapitola se zabývá detailněji pohledem na operační systém Android a běh aplikací na něm spuštěných.

3.1 Architektura operačního systému

Operační systém Android se skládá z pěti vrstev, které jsou znázorněny na Obrázku 5.



Obrázek 2: Architektura OS Android (převzato z http://simpledeveloper.com/android-architecture/)

- **Linux Kernel**

  Jedná se o nejnižší vrstvu systému. Slouží pro přímou komunikaci s hardwarem, čímž zajišťuje hardwarovou abstrakci pro vyšší vrstvy. Zajištuje komunikaci například s displejem, kamerou, procesy, pamětí, Bluetooth a podobně. Všechny aplikace a služby jsou v Androidu spuštěny odděleně. Proto je zapotřebí umožnit komunikaci mezi procesy. K tomuto účelu je v jádře modul **Binder**, který ji zajišťuje.

- **Knihovny**

  Nad jádrem jsou implementovány knihovny. Poskytují přímý přístup ke komponentám systému. Veškeré knihovny jsou napsané v jazyce C/C++.

- **Android Runtime**

  Součástí této knihovny je virtuální stroj Dalvik Virtual Machine, který slouží ke kompilaci zdrojových kódů, jejich spuštění a následnému uchování instancí spuštěných aplikací.

- **Aplikační framework**

  Jedná se o framework, který umožňuje vývojářům využívat základní ovládací prvky systému, jako je například ovládání displeje pomocí dotyku prstů, základní prvky systémů, tlačítka, scrollbary a podobně. Dále pak spravuje životní cykly aplikací.

- **Aplikace**

  [5]

3.2 Architektura aplikací pro Android

Architektura aplikací je založena na čtyřech základních částech. Všechny čtyři části jsou napsané jako třídy. [5]

- **Aktivita**

  Jde o hlavní třídu umožňující komunikaci mezi aplikací a uživatelem. Každá aktivita by měla obsluhovat jedno uživatelské rozhraní, jinak také nazývané jako GUI.

- **Služba**

  Jedná se o proces běžící na pozadí. Nereaguje s uživatelem pomocí rozhraní. Příkladem takové služby může být například běh stopek či časovače na pozadí.

- **Broadcast receiver**

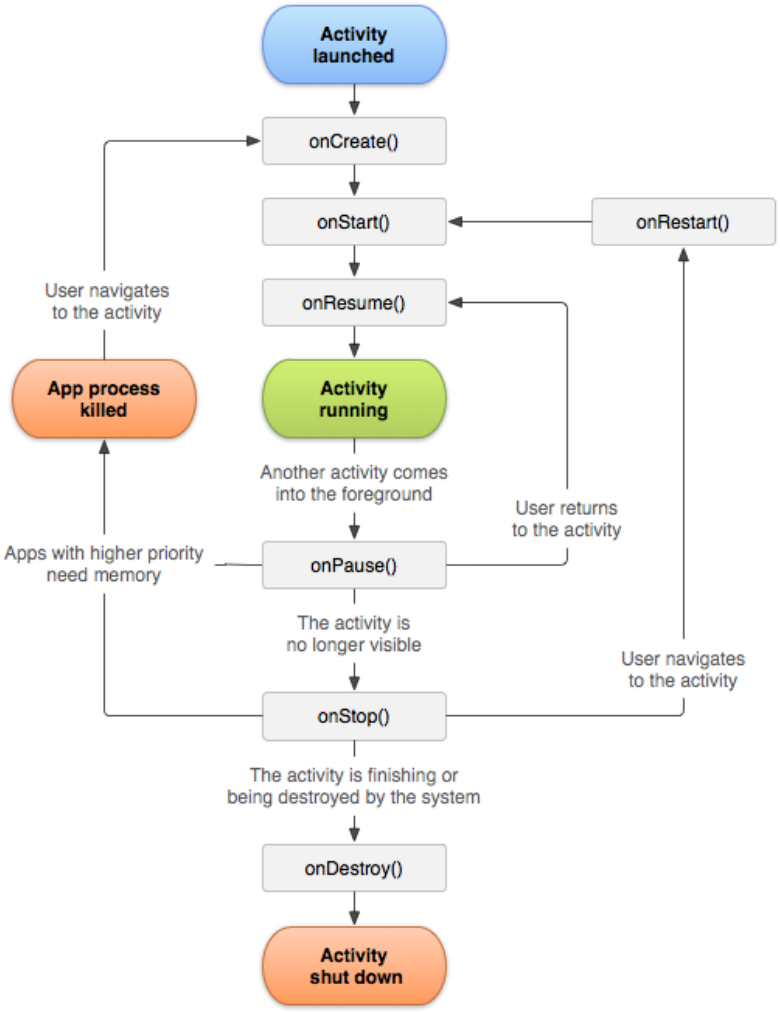
Jsou to objekty, které jsou určené pro vysílání a přijímaní událostí, které se odehrávají na pozadí v systému.

- **Content provider**

  Poslední komponentou je takzvaný poskytovatel obsahu. Umožňuje sdílení dat mezi aplikacemi či procesy. Ty k datům přistupují pomocí třídy ContentResolver.

3.2.1 Životní cyklus aktivit

Jak již bylo popsáno výše, třída Activity slouží pro obsluhu uživatelského rozhraní. Každá aktivita však prochází určitým cyklem stavů. Stavy jsou vyvolávány pomocí metod, které popisuje Obrázek 3. [6]



Obrázek 3: Životní cyklus aktivity (převzato z https://developer.android.com/reference/android/app/Activity.html)

Inicializační metody

1) onCreate()
2) onStart()
3) onResume()

Přerušovací metody

1) onPause()
2) onStop()

Zastavovací metoda

1) onDestroy

Všechny aktivity v systému jsou ukládány do zásobníku. Tento zásobník funguje na principu LIFO (Last in – First out).

Jak je patrné z Obrázku 3, každá aktivita začíná spuštěním metody onCreate(). V ní jsou implementovány všechny věci potřebné pro její běh. V případě, že je zapotřebí aktivitu vyvolat, je nutné ji spustit pomocí metody onStart(). V případě, že se aktivita má dostat do popředí, musí být zavolána metoda onResume(). V tu chvíli s ní může uživatel pracovat.

Pokud je aktivita překryta jinou aktivitou, zastaví se pomocí metody onPause() a přestává být v zásobníku na prvním místě. V případě potřeby uvolnění místa, může být aktivita z paměti odstraněna. Poté je potřeba aktivitu opět zahájit metodou onCreate().

V případě, že aktivita přestává být viditelná, je zavolána metoda onStop() a dostává se do pozadí. Může být opět spuštěna a nebo, pokud již není potřeba, se ukončí její životní cyklus pomocí metody onDestroy().

4 Principy komunikace Klient-Server a Peer-to-Peer

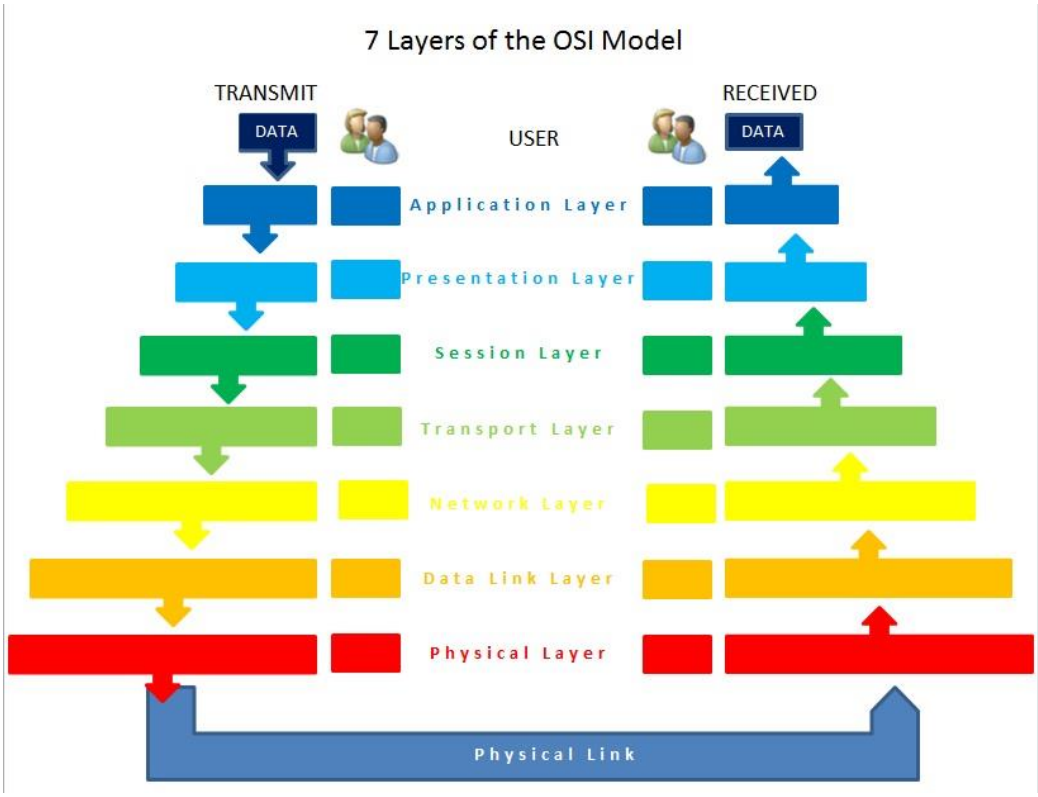
Pro síťovou komunikaci je nutné vytvořit všem zařízením dostupnou síť, aby spolu mohly komunikovat. V dnešní době je nejrozšířenější komunikace zařízení založena na technologii ethernet, která využívá protokoly TCP/IP. Tento protokol vychází z modelu ISO/OSI. Jednotlivé sítě je možné řadit podle několika kritérií. Například podle velikosti sítě, topologie, technologie a dalších.

4.1 Komunikace v síti

Následující kapitola popisuje základní komunikační prvky v síti, jejich účel a pozici v síti.

4.1.1 ISO/OSI

Jedná se o referenční model, sestavený mezinárodní organizací pro normalizaci, který rozděluje komunikaci zařízení do sedmi souvisejících vrstev. Každá vrstva komunikuje s vrstvou, která je přímo pod ní. Při odesílání dat postupuje proces od aplikační vrstvy k fyzické a při jejich přijímání zase opačně. [7]



Obrázek 4: Model ISO/OSI (převzato z https://www.cnews.cz/technologie-pocitacove-site-jak-pracuje-tcpip-a-isoosi/)

- **Aplikační vrstva**

  Jedná se o nejvyšší vrstvu, která definuje způsob, jakým komunikují aplikace se sítí. Může se jednat například o odesílání a přijímaní zpráv pošty pomocí protokolů POP3 a SMTP, nebo komunikaci dokumentů pomocí protokolu HTTP nebo HTTPS a mnoho dalších.

- **Prezentační vrstva**

  Tato vrstva přijímá data od aplikační vrstvy, která převede do binární podoby. Dále se pak stará o šifrování a kompresy dat.

- **Relační vrstva**

  Navazuje a udržuje spojení mezi zařízeními, dokud je potřeba. Zajišťuje správné odeslání a přijetí dat pomocí synchronizačních značek.

- **Transportní vrstva**

  Rozděluje data na segmenty, které jsou odesílány příjemci zpráv.

- **Síťová vrstva**

  Zabalí segmenty do paketů, které jsou pak směrovány v síti. Přidává jim adresu odesílatele a příjemce. Příkladem síťové vrstvy je protokol IP.

- **Spojová vrstva**

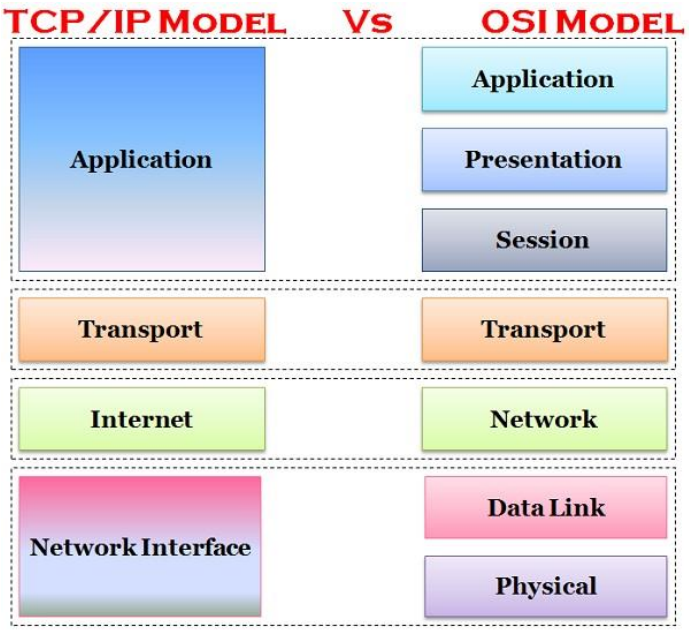
  Přetváří pakety na rámce a zajišťuje jejich tok v síti. Je zde prováděna synchronizace dat pomocí CRC, čímž se zajistí správný přenos dat mezi stanicemi. Jedná se o hardwarovou vrstvu.

- **Fyzická vrstva**

  Odesílá a přijímá signály přes fyzické médium.

4.1.2 TCP/IP a ISO/OSI

V dnešní době je využíván model TCP/IP, který vychází z modelu ISO/OSI. Došlo ke zjednodušení modelu ze sedmi vrstev na čtyři. [8]



Obrázek 5: Porovnání TCP/IP a ISO/OSI (převzato z https://techdifferences.com/difference-between-tcp-ip-and-osi-model.html)

- **Aplikační vrstva**

  Zajišťuje komunikaci s aplikacemi pomocí protokolů HTTP, HTTPS, POP3 a mnoho dalších. Předává data transportní vrstvě.

- **Transportní vrstva**

  Navazuje spojení a tok dat pomocí protokolů TCP, který zajistí spolehlivý přenos všech dat. Druhou možností je protokol UDP, který se nezabývá kontrolou, zda jsou všechna data doručena příjemci.

- **Síťová vrstva**

  Funguje na stejném principu jako v modelu ISO/OSI: Zajišťuje směrování dat v síti.

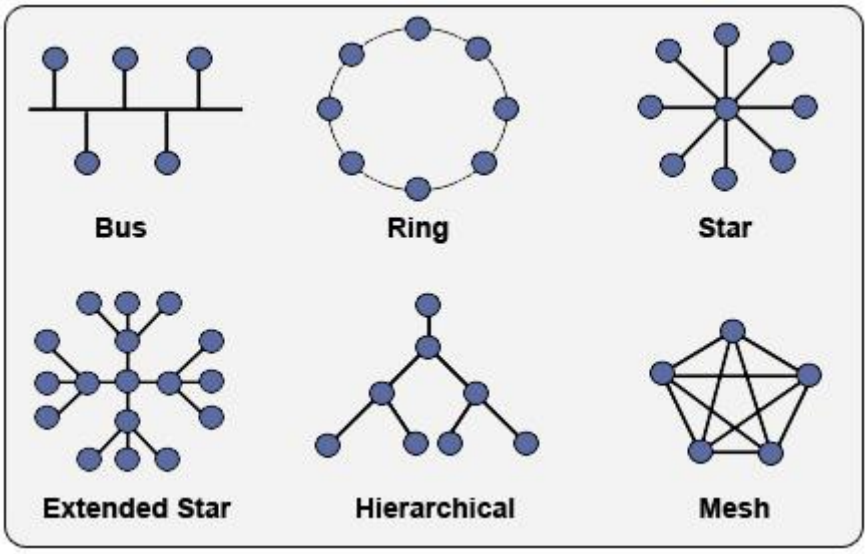
- **Vrstva síťového rozhraní**

  Zajišťuje přenos dat mezi médii. Zpravidla pomocí technologie Ethernet.

4.2 Topologie sítě

Topologie sítě udává základní seskupení či tvar jednotlivých zařízení. Jedná se o fyzické zapojení jednotlivých prvků do sítě. Mezi jednotlivé prvky patří hvězda,

rozšířená hvězda, kruh, sběrnice nebo mřížka. Všechny tyto topologie jsou znázorněny na Obrázku 6.



Obrázek 6: Základní prvky síťové topologie (převzato z https://www.host.co.in/blog/network-topology/)

V dnešní době mezi nejvyužívanější topologie patří hvězda nebo rozšířená hvězda, kde se pomocí switche, hubu nebo routeru vytvoří bod, který šíří komunikaci do více zařízení. Všechny tyto topologie slouží pro přenos drátové komunikace jednotlivých zařízení.

4.3 Postavení prvků v síti

Při vytvoření sítě nemusí být nutně všechny prvky na stejné úrovni. Rozlišujeme možnosti typu Klient-Server a Peer-to-Peer.

4.3.1 Klient-Server

Jedná se o postavení, kde jeden nebo více prvků jsou v roli serveru a připojují se na něj ostatní prvky v roli klienta. Klienti vznáší požadavky na servery a následně čekají na jejich odpověď. Naproti tomu servery pouze naslouchají a vyčkávají na případný dotaz od klientů, aby na něj mohly odpovědět. Takovéto uspořádání nalezneme typicky u webových aplikací. Výhodou tohoto uspořádání je případná údržba programu, který servery nabízí. Daný program stačí vyměnit pouze na jedné nebo několika stanicích a následně je všem uživatelům přístupná aktuální verze programu. Nevýhodou takového uspořádání je vytíženost serverů. Při velkém

množství dotazujících se klientů budou servery vytížené a doba odpovědí bude následně prodloužena.

4.3.2 Peer-to-Peer

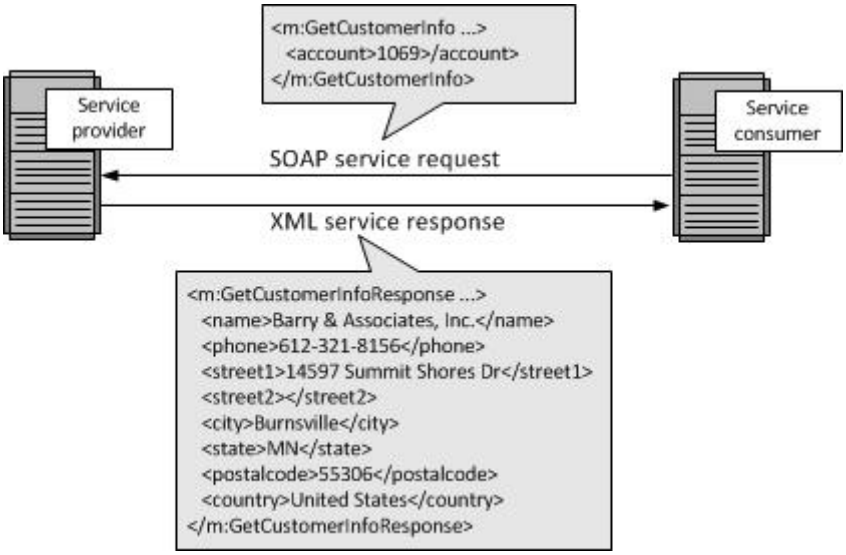
Jde o postavení prvků, ve kterém jsou si všichni vzájemně rovni. Jde tedy o přesný opak architektury Klient-Server. Všechny prvky plní zároveň funkci jak serveru, tak klienta. Výhodou tohoto organizačního seskupení je, že s rostoucím počtem zařízení se zvyšuje rychlost sítě. Na tomto principu stojí základní infrastruktura internetu nebo například přenos souborů pomocí FTP protokolu.

5 Webová služba

Webová služba je program, který poskytuje ostatním programům rozhraní, na které se oni mohou dotázat. Poté, co webová služba obdrží požadavek, pokusí se ho zpracovat a vrátit klientovi výslednou odpověď. Webové služby pracují na straně serveru. Jejich prvním tvůrcem byl Microsoft. Výhodou služeb je, že díky rozhraní, které poskytují, mohou pracovat na jiné platformě než program, který je vyvolává. V dnešní době jsou využívány především webové služby pracující na technologii SOAP nebo REST. Obě možnosti jsou popisovány v kapitolách níže.

5.1 SOAP

Simple Object Access Protocol (SOAP) je protokol založený na výměně zpráv ve formátu XML jak ze strany serveru, tak i klienta. Pracuje především na protokolech HTTP nebo SMTP. Tento protokol definuje standardy, které musí být pro přenos dodržovány. Samotný koncept je velmi obecný a rozšiřitelný. [9]



Obrázek 7: Komunikace s webovou službou pomocí SOAP (převzato z https://www.service-architecture.com/articles/web-services/web_services_explained.html)

5.1.1 WSDL

Web Service Description Language (WSDL) je soubor, který popisuje danou webovou službu. Je vždy napsaný ve formátu XML a je zodpovědný za správné přenesení informací o webové službě klientovi, který se na ní bude připojovat. Poskytuje jim informace o vstupních a výstupních parametrech, způsobu navázání

spojení, poskytované operace a další. Na jeho základě se poté mohou klienti dané služby dotazovat. [10]

5.1.2 UDDI

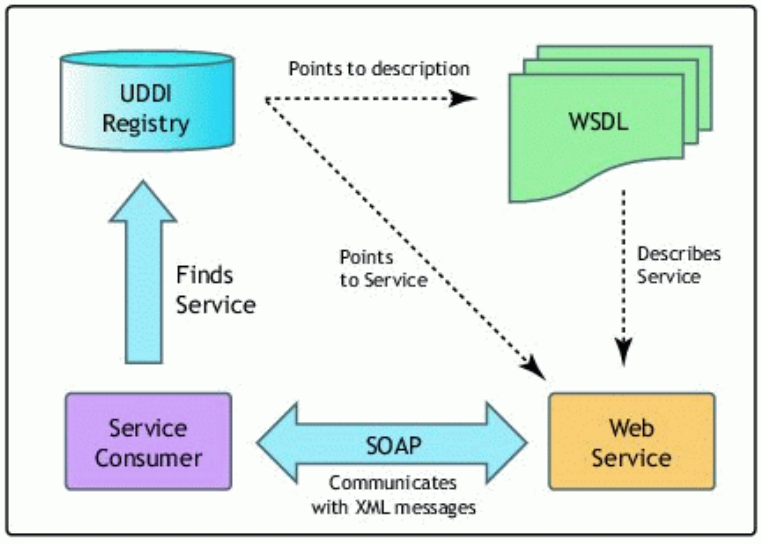
Celým názvem Universal Discovery Description and Integration. Ve své podstatě se jedná o registr webových služeb, který by pomohl zprostředkovat připojujícím se klientům WSDL dané webové služby. V dnešní době se UDDI pro většinu služeb nevyužívá.

5.1.3 Původní koncept

Jako původní koncept SOAP protokolu byla navržena architektura, která se skládala ze čtyř prvků. Jsou jimi:

- **Klient**
- **Server**
- **UDDI**
- **WSDL**

Nejprve byla vytvořena webová služba, která se zaregistrovala do registru. Poté, co klient potřeboval využívat některou webovou službu, dotázal se registru na seznam možných služeb. Ten mu vrátil daný předpis webové služby a její adresu. Poté k ní již mohl klient přistupovat. Vizualizace je zobrazena na Obrázku 8. [17]

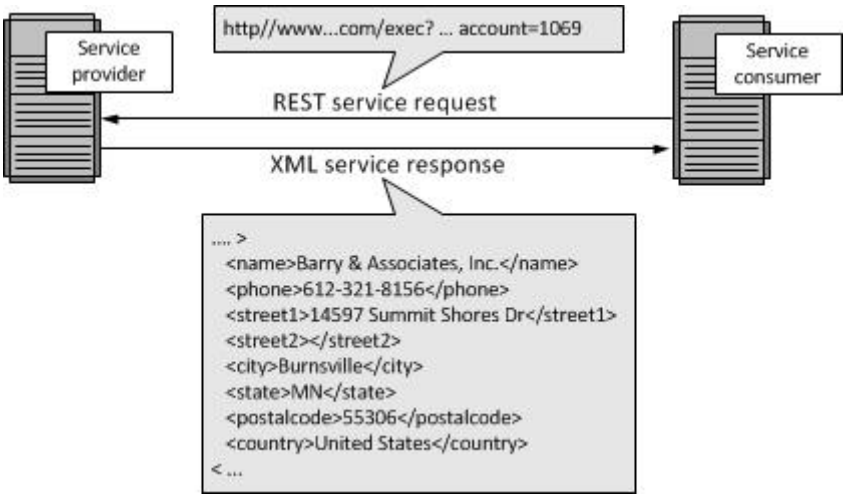


Obrázek 8: Původní architektura SOAP (převzato z https://www.systemonline.cz/sprava-it/webove-sluzby-a-xml.htm)

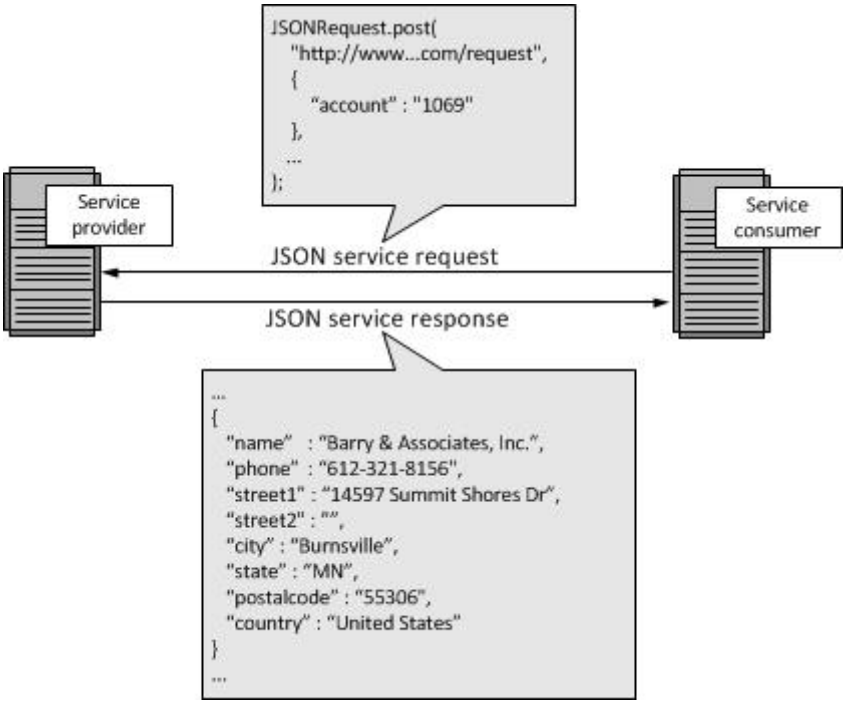
5.2 REST

Representation State Transfer je architektonický styl sloužící pro výměnu dat na protokolu HTTP. Je náhradou protokolu SOAP. Jeho komunikace dat nemusí být nutně pomocí XML. Může být dále také nadefinován například pomocí JSON, CSV, nebo dalších. [18]

Jedná se o lehčí alternativu komunikace oproti SOAP. Klientovi stačí vznést požadavek na konkrétní URL adresu, a ta mu vrátí výsledek. Komunikace mezi službou a klientem může být zprostředkována i pomocí webového prohlížeče. Důležité je dodržení správných vstupních parametrů a metody odeslání dat (GET nebo POST). Možné způsoby komunikace pomocí REST jsou znázorněny na Obrázku 9 a 10. [10]



Obrázek 9: Komunikace s webovou službou pomocí REST metodou GET (převzato z https://www.service-architecture.com/articles/web-services/web_services_explained.html)



Obrázek 10: Komunikace s webovou službou pomocí REST metodou POST (převzato z https://www.service-architecture.com/articles/web-services/web_services_explained.html)

6 Analýza a implementace

Celý systém, jehož vytvoření je cílem této práce, by měl umožňovat komunikaci pomocí architektury Klient-Server a Peer-to-Peer. Bude tedy zapotřebí implementovat dvě oddělené části. Klientskou a serverovou část.

6.1 Server

Serverová část je program, který poskytuje ostatním programům rozhraní (API), na které umí reagovat, a programy se ho tedy na něj mohou dotázat.

6.1.1 Funkční požadavky

- **Zjištění dostupnosti webové služby**
  Je žádoucí, aby bylo možné jednoduchým způsobem ověřit dostupnost webové služby.

- **Uložení souboru na server**
  Systém by měl umět uložit soubory do lokálního úložiště.

- **Poskytnutí souboru**
  Systém by měl umožňovat stažení souboru ostatními aplikacemi.

- **Poskytnutí seznamu dostupných souborů**
  Aby bylo možné soubor stáhnout, musí mít klienti také dostupný seznam souborů, ze kterých mohou vybírat.

- **Uložení dat do databáze**
  Systém by měl umět ukládat počet uživatelů registrovaných na serveru. Měla by být implementována kontrola jedinečnosti uložení dat.

- **Načtení dat z databáze**
- **Poskytnutí seznamu dostupných výsledků operací**

6.1.2 Nefunkční požadavky

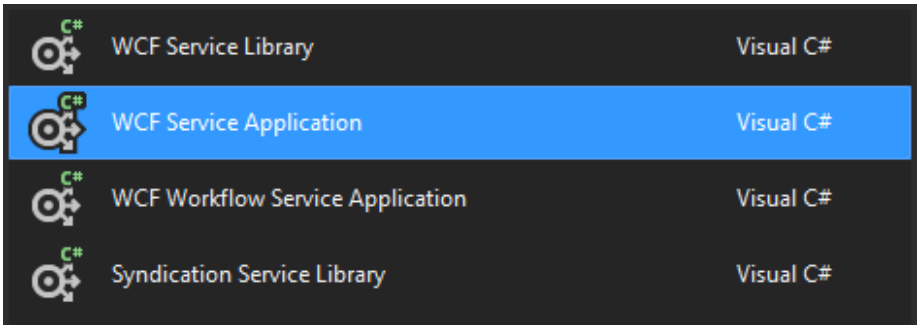
- **Implementace proběhne pomocí WCF**
- **Systém bude nasazen na veřejném hostingu**
- **Komunikace se serverem nebude pro klienta nijak vytěžující**
- **Bude implementována kontrola nepovolených znaků při uložení souborů**
- **Bude implementována kontrola maximální možné velikosti souboru**
- **Systém bude konfigurován tak, aby byl dostupný i z webového prohlížeče**

6.1.3 Vytvoření služby

Pro prvotní implementaci webové služby je zapotřebí stáhnout a nainstalovat vhodné vývojové prostředí. Takovým prostředím je Visual Studio od společnosti Microsoft Corporation. V tomto případě bylo využito prostředí ve verzi 17.

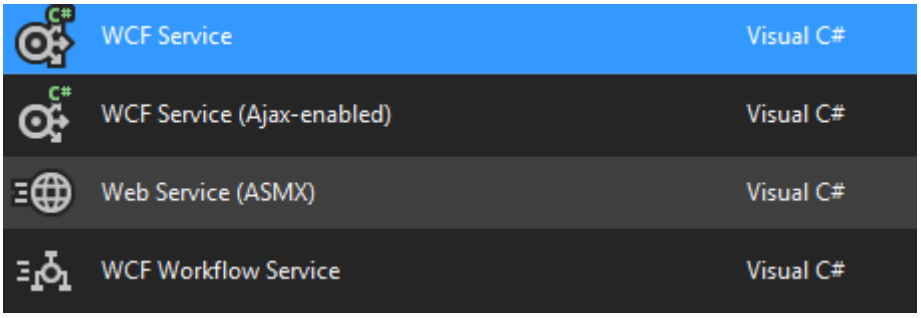
Obrázek 11: Logo vývojového prostředí Visual Studio 2017 (převzato z https://commons.wikimedia.org/wiki/File:Visual_Studio_2017_logo_and_wordmark.svg)

První krokem je vytvoření nového projektu WCF, což je framework sloužící pro tvorbu serverově orientovaných aplikací. Úplný název frameworku je Windows Communication Foundation. V tomto konkrétním případě je vhodné zvolit „WCF Service Application“. Visual Studio automaticky vygeneruje základní nastavení a rozvržení aplikace s první službou, která implementuje API pro zavolání funkce „Hello world“.



Obrázek 12: Vytvoření WCF aplikace (zdrojem autor)

Nyní je zapotřebí vytvořit a implementovat novou službu, která bude naprogramována podle výše popsaných požadavků. Pro účely této práce je vhodné zvolit službu, která bude mít příponu „.svc“, namísto služby s příponou „.asmx“. Obě možnosti jsou zvýrazněné na Obrázku 13.



Obrázek 13: Vytvoření webové služby (zdrojem autor)

Nyní vývojové prostředí připravilo rozhraní (interface) pro webovou službu a její implementační třídu.

Nejdříve začneme implementovat rozhraní. V něm nadefinujeme veškeré nutné metody, které by měla třída implementovat. Důležitý je zde anotační atribut „ServiceContract“. Ten definuje, že se jedná o kontrakt webové služby. Píše se nad název rozhraní. Každá metoda, která má být dostupná z ostatních aplikací musí obsahovat anotaci „OperationContact“. Ta definuje, že se jedná o metody podporované danou službou. Další anotací nad metodami je „WebInvoke“. Ta obsahuje několik dalších parametrů.

- **Method**

  Jedná se o způsob volání dané metody. Je možné vybrat mezi voláním typu GET nebo POST.

- **ResponseFormat**

  Upřesňuje způsob komunikace mezi službou a klientem. Na výběr je z možností pomocí JSON nebo XML formátu.

- **BodyStyle**

  Definuje styl dotazu a odpovědi, tedy zda bude komunikace více nebo jednořádková.

- **UriTemplate**

  Zde je možné definovat URL volání dané metody. V případě, že se jedná o metodu typu GET s určitými parametry, je potřeba je také definovat v předpisu. Každé URL musí být v rámci jedné webové služby unikátní. Webová služba jinak není validní a nepůjde spustit.

Poté je nutná následná implementace rozhraní do třídy webové služby. Tam se již využívá základní konvence jazyka C#. Na ukázkách Zdrojového kódu 1 a 2 je zobrazena kompletní implementace metody, díky které je možné ověřit, zda je webová služba momentálně dostupná.

```
[ServiceContract]
public interface IService
     {
         [OperationContract]
         [WebInvoke(Method = "GET", ResponseFormat = WebMessageFormat.Json,
             BodyStyle = WebMessageBodyStyle.Wrapped, UriTemplate = "ping/")]
          bool Ping();
     }
```

Zdrojový kód 1: Rozhraní webové služby (zdrojem autor)

```
public class Service : IService
{
        public bool Ping()
        {
            return true;
        }
}
```

Zdrojový kód 2: Implementace webové služby (zdrojem autor)

6.1.4 Konfigurace služby

Pro správné využívání aplikace je zapotřebí správně nastavit soubor „Web.config“. Jde o soubor, který obsahuje základní nastavení celé aplikace. Je

napsaný pomocí předpisu XML. Je implicitně umístěn v kořenovém adresáři celé aplikace. Zároveň určuje, jakým způsobem má celá aplikace fungovat. [11]

Tři základní nastavení v souboru „web.config“:

- Chování
- Spojení
- Definice služby

6.1.4.1 Konfigurace chování

Konfigurace chování je rozdělena na dvě části. První částí je definice endpointů, druhou poté definice chování. Jelikož naše aplikace nepotřebuje využívat více možností chování, stačí nám nakonfigurovat pouze jeden endpoint, využívající předpis webHttp, a jedno chování. Důležité je oba XML tagy pojmenovat pomocí atributu „name“. Ten je následně využíván při konfiguraci služby.

```
<behaviors>
  <endpointBehaviors>
    <behavior name="web">
      <webHttp />
    </behavior>
  </endpointBehaviors>
  <serviceBehaviors>
    <behavior name="serviceBehavior">
      <serviceMetadata httpGetEnabled="true" />
      <serviceDebug includeExceptionDetailInFaults="false" />
    </behavior>
  </serviceBehaviors>
</behaviors>
```

Zdrojový kód 3: Konfigurace chování webové služby (zdrojem autor)

6.1.4.2 Konfigurace spojení

Pro konfiguraci spojení postačí definovat, že se jedná o spojení typu „webHttpBinding“, a přidat atributy povolující dostatečnou velikost dat.

```
<bindings>
  <webHttpBinding>
    <binding name="webBindingWithMaxSize"
             maxBufferPoolSize="2147483647"
             maxReceivedMessageSize="2147483647"
             maxBufferSize="2147483647">
      <readerQuotas
        maxArrayLength="2147483647"
        maxBytesPerRead="2147483647"
        maxDepth="2147483647"
        maxNameTableCharCount="2147483647"
        maxStringContentLength="2147483647" />
    </binding>
  </webHttpBinding>
</bindings>
```

Zdrojový kód 4: Konfigurace spojení webové služby (zdrojem autor)

6.1.4.3 Konfigurace definice služby

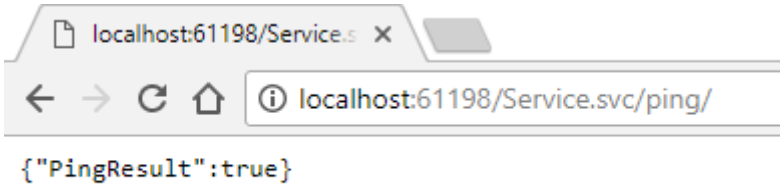
Posledním prvkem konfigurace konfiguračního souboru je nastavení služby. V tomto případě je zapotřebí odkazovat na vytvořenou servisu „Service“, které se nastaví konfigurace připraveného chování a vazeb. V tomto případě budeme využívat již vytvořené konfigurace znázorněné výše. Ty využijeme zapsáním jejich názvů do příslušných atributů. V poslední řadě je nadefinován kontrakt. Ten definuje rozhraní, které daná služba bude podporovat.

```
<services>
  <service name="BachelorWcfService.Service"
           behaviorConfiguration="serviceBehavior">
    <endpoint address="" binding="webHttpBinding"
              contract="BachelorWcfService.IService" behaviorConfiguration="web"
              bindingConfiguration=" webBindingWithMaxSize " />
  </service>
</services>
```

Zdrojový kód 5: Konfigurace webové služby (zdrojem autor)

6.1.5 Ověření dostupnosti služby

V případě, že je služba správným způsobem nakonfigurována, je možné ověřit její dostupnost zavoláním vybrané metody, například v internetovém prohlížeči pomocí jejího URL. Její dostupnost je zobrazena na Obrázku 14.



Obrázek 14: Ukázka ověření dostupnosti webové služby pomocí prohlížeče (zdrojem autor)

[12]

6.1.6 Implementace služby

Následující kapitola se zabývá zobrazením hlavních částí implementace webové služby. Z důvodu přehlednosti byly z kódu odstraněny komentáře.

6.1.6.1 Poskytnutí seznamu možných výsledků operací

Z důvodu jednotnosti jsou výsledky operací implementovány na serverovou část. Je proto vytvořená třída obsahující seznam enumů, které je možné poskytovat. V případě budoucí implementace nebo jazykové mutace je možné tento seznam rozšířit, nebo upravit o další možnosti, které se poté automaticky přenesou všem klientům.

```
public enum Results
{
    [Description("Vše proběhlo v pořádku.")]
    Success = 0,

    [Description("Překročena maximální možná velikost souboru.")]
    MaxSizeExceed = 1,

    [Description("Soubor s tímto názvem již existuje.")]
    FileNameDuplicated = 2,

    [Description("Neočekávaná chyba, opakujte prosím znovu.")]
    UnexpectedError = 3,

    [Description("Ve jméně souboru se nacházejí nepodporavetlené znaky.")]
    NotSupportedCharsInFileName = 4,

    [Description("Vybraný soubor neexistuje.")]
    FileNotExist = 5,

    [Description("Toto zařízení bylo již uloženo do databáze.")]
    ImeiLogged = 6,

}
```

Zdrojový kód 6: Seznam možných výsledků operací webové služby (zdrojem autor)

Enum je datový typ, který po jeho předání vrátí buď jeho hodnotu, v našem případě číslo, nebo svůj název. Pro naše účely je zapotřebí odesílat jeho popis, který se nachází v anotaci „Description“. Je proto nutné vytvořit metodu, která očekává enum, a vrátí jeho popis. Ten již obsahuje informace vhodné pro předání uživateli. Metoda je zobrazena na implementaci Zdrojového kódu číslo 7.

```
public static string GetDescriptionFromEnumValue(Enum value)
{

    FieldInfo fi = value.GetType().GetField(value.ToString());
    DescriptionAttribute[] attributes =
       (DescriptionAttribute[])fi.GetCustomAttributes(
       typeof(DescriptionAttribute), false);
    if (attributes.Length > 0)
    {
        return attributes[0].Description;
    }

    return value.ToString();
}
```

Zdrojový kód 7: Vrácení popisu z enumu (zdrojem autor)

6.1.6.2 Nahrání souboru na server

Nahrání souboru na server je jednou z klíčových funkčností systému. Metoda očekává dva parametry, a to data souboru a jeho úplný název. Její tělo předává logiku statické třídě, která se postará o kontrolu souboru, tedy zda neobsahuje nepovolené znaky, nebo jestli již není ve složce soubor s tímto názvem uložen. Pokud proběhne vše v pořádku, soubor se uloží. Výsledkem metody je vždy číslo odpovídající hodnotě enumu z kapitoly 6.1.6.1, které se předá klientovi, a informuje ho o výsledku průběhu akce.

```
public int UploadFile(string byteArray, string filename)
{
    return (int)FileHelper.SaveFile(
                      Convert.FromBase64String(byteArray), filename);
}
```

Zdrojový kód 8: Implementace nahrání souboru webovou službou (zdrojem autor)

6.1.6.3 Stažení souboru ze serveru

Pro stažení souboru ze serveru je zvolena technika, která navrací URL adresu ke stažení souboru, nebo chybovou hlášku z výčtu enumů. Pro konkrétní výběr možnosti byl využit ternární operátor.

```
public string DownloadFile(string fileName)
{
    return FileHelper.ExistFile(fileName) ==
                           Enums.Results.FileNotExist
                           ? FileHelper.GetUrlOfFile(fileName)
                           : Enums.GetDescriptionFromEnumValue(
                             Enums.Results.FileNotExist);
}
```

Zdrojový kód 9: Předání výsledku dotazu o stažení souboru (zdrojem autor)

6.1.6.4 Čtění dat z databáze

Během využívání databáze je ve WCF možné vybírat z několika možných způsobů. Vzhledem k jednoduchosti databázové logiky byly zvoleny přímé dotazy na databázi. Takovéto dotazy nepodléhají žádnému převodu z LINQ expressionu na SQL dotazy. Mezi výhody takových dotazů patří primárně jejich rychlost, která se projeví u složitějších a komplexnějších dotazů. Dotazy je možné je upravovat přímo, dle potřeby. Nevýhodou je ovšem jejich unikátnost. V případě změny názvu sloupce nebo tabulky je nutné ručně přepsat všechny dotazy v aplikaci, zatímco jiné nástroje by nám tuto práci značným způsobem ulehčily.

Pro čtení z databáze byla v systému implementována funkčnost, která vrací počet přihlášených uživatelů k dané verzi systému. Metoda navrací datový typ „slovník“, který obsahuje textový řetězec a číslo. Pro takový datový typ je nutné zajistit, aby vždy klíčová hodnota, v našem případě textový řetězec, byly unikátní. To je zajištěno pomocí nastavení unikátnosti příslušného sloupce v databázi.

```
public Dictionary<string, int> GetLoggedMobiles()
{
    Dictionary<string, int> toReturn = new Dictionary<string, int>();
    using (var conn = new SqlConnection(ConnectionString))
    {
        conn.Open();
        using (SqlCommand cmd = conn.CreateCommand())
        {
            cmd.CommandText = "Select Version, Count From
                               [dbo].[ConnectedDevice]";

            cmd.CommandType = CommandType.Text;

            SqlDataReader reader = cmd.ExecuteReader();

            while (reader.Read())
            {
                toReturn.Add(Convert.ToString(reader[0]),
                             Convert.ToInt32(reader[1]));
            }
        }
    }

    return toReturn;
}
```

Zdrojový kód 10: Obsluha databáze z webové služby (zdrojem autor)

6.1.6.5 Zápis dat do databáze

Zápis dat do databáze je vyřešen obdobným způsobem jako jeho čtení, tedy pomocí přímých SQL dotazů.

Výsledný případ využití přijímá od klientských aplikací data o verzi systému, na které jsou spuštěny. Byl kladen důraz na jedinečnost zápisu. Ta je zajištěna pomocí unikátního čísla IMEI, které obsahuje každé mobilní zařízení. Metoda tedy přijme číslo IMEI od klienta a verzi systému. Následně z databáze přečte, zda již nebylo toto číslo uloženo do databáze. Pokud se tento uživatel ještě nepřipojoval na službu, uloží jeho verzi systému do databáze. Vždy poté vrátí enum o výsledku operace klientovi.

6.2 Klient

Klientská část je program pro mobilní zařízení, který dokáže komunikovat s webovou službou a ostatními mobilními zařízeními. Vyzkouší se na něm tedy komunikace pomocí Klient-Server i Peer-to-Peer.

6.2.1 Funkční požadavky

- **Přehlednost**
- **Jednoduché a příjemné ovládání aplikace pro všechny uživatele**
- **Odeslání souboru na server**
- **Stažení vybraného souboru ze serveru**
- **Výpis možných dostupných souborů na serveru**
- **Navázání spojení s jiným mobilním zařízením pomocí P2P[1]**
- **Odeslání dat pomocí P2P[1]**

6.2.2 Nefunkční požadavky

- **Aplikace bude funkční na systémech Android 4.4 a vyšší**
- **Komunikace pro P2P bude využívat technologii Bluetooth**

6.2.3 Vytvoření klienta

Současným vhodným vývojovým prostředím pro takovou aplikaci je Android Studio od Google Inc., které umožňuje také ladění aplikace buď ve zvoleném emulátoru, nebo na reálném zařízení, které se připojí k počítači.

Obrázek 15: Logo vývojového prostředí Android Studio (převzato z http://www.steptoinstall.com/how-to-install-android-studio-on-windows-step-by-step.html)

Na začátku je zapotřebí vytvořit pomocí wizzardu nový projekt. Vývojové prostředí nám založí čistý projekt obsahující pouze základní prvky a rozložení. Následně je program připraven k implementaci.

[1]Peer-to-Peer. Způsob navázání spojení.

6.2.4 Konfigurace systému

Jedním z požadavků na systém je, aby byl program spustitelný na zařízeních s verzí operačního systému Android 4.4 a vyšší. To lze zajistit úpravou konfiguračního souboru „build.gradle“, ve kterém je nastavena minimální verze systému. Ta se zobrazuje pomocí pořadového čísla systému. V našem případě se jedná o verzi 19. [13]

```
compileSdkVersion 26
defaultConfig {
    applicationId "com.bachelor.tomas.client"
    minSdkVersion 19
    targetSdkVersion 26
    versionCode 1
    versionName "1.0"
    testInstrumentationRunner
"android.support.test.runner.AndroidJUnitRunner"
}
```

Zdrojový kód 11: Konfigurace klientské aplikace (zdrojem autor)

V tomto konfiguračním souboru jsou nadále naimportovány potřebné knihovny, které se využívají.

V klientské aplikaci jsou naimportovány následující tři knihovny:

- **com.mcxiaoke.volley:library:1.0.19**
  Slouží pro odesílání požadavků na server a příjem jeho odpovědí.

- **com.android.support:recyclerview-v7:26.1.0**
  Tato knihovna slouží pro úpravu výpisu seznamů.

- **com.github.PhilJay:MPAndroidChart:v3.0.3**
  Poslední potřebnou knihovnou je MPAndroiChart, která slouží pro jednoduché využívání zobrazení grafů.

Všechny externí knihovny je možné nahrát z úložiště (CDN), nebo přímo vložením daného balíčku do aplikace a referencí na něj. V případě této aplikace byl u všech tří referencí využit způsob reference na CDN.

Dalším důležitým konfiguračním souborem je soubor „AndroidManifest.xml. V něm je zapotřebí nastavit potřebné oprávnění programu na zařízení. Jedná se o

přístup na internet, čtení dat z externího úložiště, zápis na externí úložiště, čtení údajů o telefonu a přístup k Bluetooth.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.BLUETOOTH" />
```

Zdrojový kód 12: Potřebná oprávnění pro využívání aplikace (zdrojem autor)

6.2.5 Implementace klienta

Následující kapitola se zabývá zobrazením hlavních částí implementace klientské aplikace. Z důvodu přehlednosti a zkrácení byly z následujících kódů odstraněny nezbytné bloky, které například informují uživatele o průběhu programu či odchytávají výjimky.

6.2.5.1 Architektura klientské aplikace

Klientská aplikace obsahuje dvě základní kategorie, které se poté dále dělí na subkategorie. Jedná se o připojení na server a připojení k jinému zařízení. Z toho důvodu úvodní aplikační obrazovka obsahuje pouze dva prvky, které odkazují na příslušnou kategorii. Výřez úvodní aktivity je zobrazen na Obrázku 16.

Obrázek 16: Ukázka rozložení úvodní obrazovky aplikace (zdrojem autor)

6.2.5.2 Získání a uložení možných výsledků událostí z webové služby

Jak bylo popsáno v kapitole 6.1.6.1, je potřeba, aby si klient uložil výpis výsledků událostí možných na webové službě. Z důvodu redundance dotazů na databázi je vytvořena statická třída, která obsahuje datový typ HashMap (jedná se o ekvivalent datového typu Dictionary, který se využívá v jazyce C#) udržující informace o čísle výsledné chyby a jejího popisu.

```
private static HashMap<Integer, String> dictionary;
```

Zdrojový kód 13: Datový typ slovníku udržující výsledky služby (zdrojem autor)

Pro naplnění dat je využívána metoda, která se dotáže serveru na příslušný seznam poskytovaných výsledků. Pro dotázání se webové služby je ve všech případech v aplikaci využita knihovna Volley [14], fungující na technologii HTTP. Volání serveru funguje asynchronním způsobem. Veškerá komunikace s webovou službou, popsanou v kapitole 6.1, funguje na principu odpovědí ve tvaru JSONu. Je tedy zapotřebí vytvořit požadavek na danou URL adresu, který vrací pole objektů, a ty jsou následně uloženy do HashMapy, na kterou se poté aplikace dotazuje. Metoda dotazující se na data webové služby je zobrazena na ukázce Zdrojového kódu číslo 14.

```
public static void loadData(final Context context){

    final RequestQueue requestQueue = Volley.newRequestQueue(context);

    dictionary = new HashMap<>();

    JsonObjectRequest request = new
                 JsonObjectRequest(Request.Method.GET, url,
                (String) null, new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {

                JSONArray jArray =
                     response.getJSONArray("GetUploadResultsResult");
                for (int i = 0; i < jArray.length(); i++) {
                        JSONObject oneObject =
                                              jArray.getJSONObject(i);

                        int key = oneObject.getInt("Key");
                        String value = oneObject.getString("Value");
                        dictionary.put(key, value);
                }

            requestQueue.stop();
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {

            requestQueue.stop();
        }
    });

    requestQueue.add(request);
}
```

Zdrojový kód 14: Načtení dat z webové služby pomocí metody GET (zdrojem autor)

Posledním krokem při implementaci třídy poskytující seznam výsledků bylo zajištění předání obsahu v rámci aplikace. K tomu slouží metoda „getMessage“, která očekává číslo chyby vrácené webovou službou při dotazování. Následně vybere příslušný text výsledku dle daného čísla, nebo v případě, že zatím nejsou data načtena, zajistí jejich načtení do paměti zavoláním metody „loadData“.

```
public static String getMessage(final Context context, int code) {

    if (dictionary == null || dictionary.size() == 0) {
        loadData(context);
    }

    return dictionary.get(code);
}
```

Zdrojový kód 15: Poskytnutí výsledku operace z webové služby (zdrojem autor)

6.2.5.3 Nahrání souboru na serveru

Pro nahrání souboru na server je nejdříve zapotřebí, aby uživatel vybral daný soubor. K tomu v systému Android slouží aktivita, která je již v systému vytvořená, takže stačí přidat záměr, kterým se daná aktivita zobrazí.

```
public void chooseFile(View view) {

    Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);

    intent.addCategory(Intent.CATEGORY_OPENABLE);

    intent.setType("*/*");

    startActivityForResult(intent, READ_REQUEST_CODE);
}
```

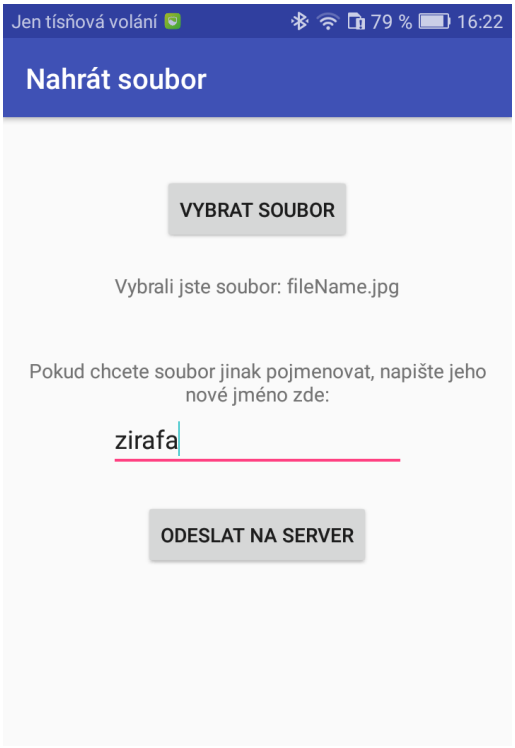
Zdrojový kód 16: Vyvolání jiné aktivity (zdrojem autor)

Zavoláním výběrové aktivity se v systému Android přeruší původní aktivita, která je nahrazena novou aktivitou. Poté, co je nová aktivita ukončena, je původní opět spuštěna. Jelikož je zapotřebí uložit URL adresu vybraného souboru, je třeba na takovou akci reagovat. Důležité je ovšem reagovat na správné přerušení aktivity. K tomu slouží identifikační čísla, která si programátor může volit dle svého uvážení.

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent
resultData) {

    if (requestCode == READ_REQUEST_CODE
              && resultCode == Activity.RESULT_OK) {

        if (resultData != null) {
            uri = resultData.getData();
        }
    }

}
```

Zdrojový kód 17: Zachycení znovuspuštění aktivity (zdrojem autor)

Poté, co je soubor vybrán, má uživatel možnost ho pojmenovat jiným názvem. Ukázka takovéto situace je zobrazena na Obrázku 17.



Obrázek 17: Ukázka nahrání souboru na server (zdrojem autor)

Následně je soubor pomocí knihovny Volley nahrán na server zavoláním příslušné URL adresy. Toto volání je implementované pomocí metody POST. Musí se u něj předat potřebné parametry, které webová služba očekává.

```
final RequestQueue requestQueue = Volley.newRequestQueue(this);

byteArrays = android.util.Base64.encodeToString(byteArray, 0);

Map<String, String> jsonParams = new HashMap<String, String>();
jsonParams.put("byteArray", byteArrays);
jsonParams.put("filename", getNewFileName());

JsonObjectRequest myRequest = new JsonObjectRequest(
        Request.Method.POST,url,new JSONObject(jsonParams),

        new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {
                setSaveResult(response.getInt("UploadFileResult"));
            }
        },
        new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
            }
        });
requestQueue.add(myRequest);}
```

Zdrojový kód 18: Načtení dat z webové služby pomocí metody POST (zdrojem autor)

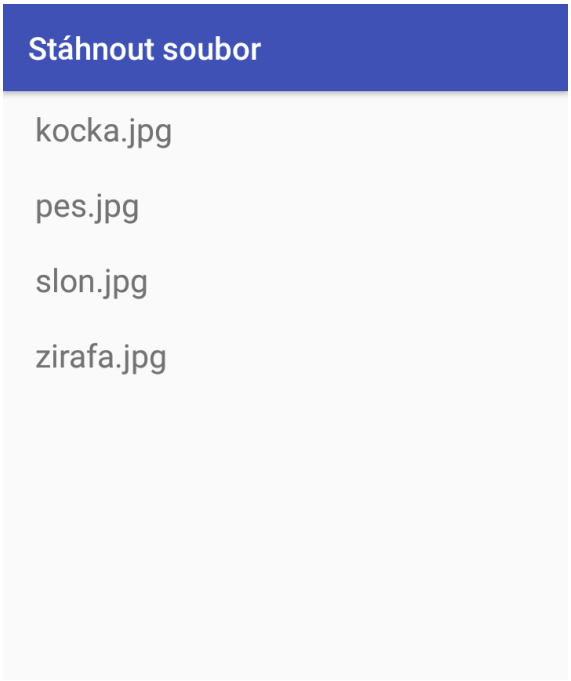
Poté, co je soubor odeslán, je od webové služby obdržen výsledek nahrávání souboru. Ten je uživateli zobrazen. Možné výsledky jsou zobrazeny na Obrázku 18 a 19.

Obrázek 18: Ukázka zobrazení úspěšného nahrání souboru (zdrojem autor)

Obrázek 19 Ukázka zobrazení neúspěšného nahrání souboru (zdrojem autor)

6.2.5.4 Stažení souboru ze serveru

Pro stažení souboru ze serveru je nejprve vyžádán seznam dostupných souborů na webové službě. Ten se uživateli zobrazí pomocí seznamu názvů souborů. Náhled aplikace je zobrazen na Obrázku 20



Obrázek 20: Ukázka výběru dostupných souborů ke stažení (zdrojem autor)

Když uživatel klikne na daný soubor, je zahájeno stahování pomocí stahovacího manageru, který soubor uloží do složky „Downloads“ mezi ostatní stahované soubory. V první fázi je obdržen ze strany serveru seznam dostupných souborů, ke kterým má klient přístup. Následně je zapotřebí dotázat se služby na příslušnou URL adresu, ze které je možné soubor stáhnout.

```
    private void download(final String filename) {

        final RequestQueue requestQueue =
                  Volley.newRequestQueue(getApplicationContext());

        JsonObjectRequest request =
                 new JsonObjectRequest(Request.Method.GET,
                 url + filename, new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {

                haveStoragePermission();

                DownloadManager.Request request =
                     new DownloadManager.Request(Uri.parse(
                     response.getString("DownloadFileResult")));

                 request.setDestinationInExternalPublicDir(
                     Environment.DIRECTORY_DOWNLOADS, filename);

                request.setNotificationVisibility(
                     DownloadManager.Request.
                     VISIBILITY_VISIBLE_NOTIFY_COMPLETED);

                manager.enqueue(request);

                requestQueue.stop();
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                requestQueue.stop();
            }
        });

        requestQueue.add(request);
    }
```

Zdrojový kód 19: Získání souboru z webové služby (zdrojem autor)

Důležitá je funkce „haveStorragePermission“, která zajišťuje zjištění a případné dotázání uživatele na přístup k ukládání souborů.

```
private boolean haveStoragePermission() {
        if(checkSelfPermission(android.Manifest.permission.
              WRITE_EXTERNAL_STORAGE) ==
              PackageManager.PERMISSION_GRANTED) {
            return true;
        } else {
            ActivityCompat.requestPermissions(this,
             new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
             1);
            return false;
        }
}
```

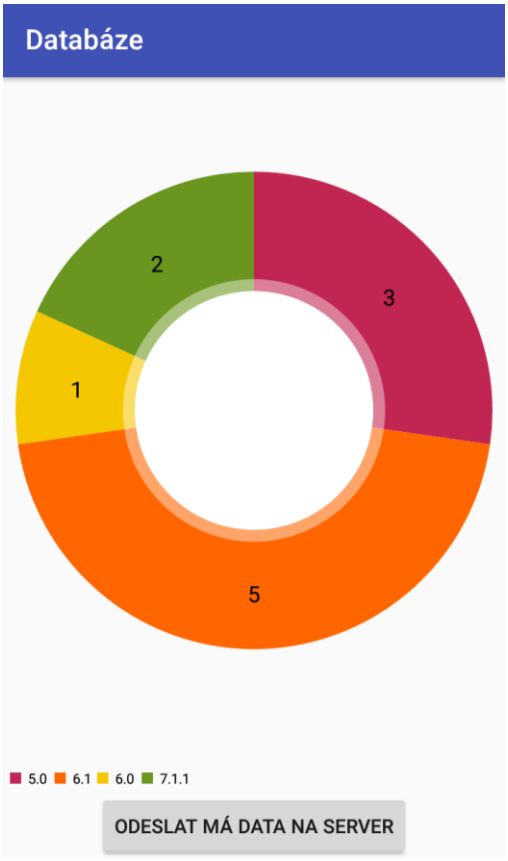
Zdrojový kód 20: Ověření o přístupu ke stahování (zdrojem autor)

6.2.5.5 Vykreslení dat v grafu

Pro vytváření grafu je v projektu využívána knihovna „MPAndroidChart“ [15]. Nejprve se systém dotáže webové služby na příslušná data, která mají být v grafu zobrazena. Ta vrátí seznam hodnot uložených ve formátu JSON. Poté je zapotřebí data uložit do objektu „PieEntry“, se kterým dokáže graf pracovat. Následně už se připravená data uloží do grafu, a ten se vykreslí pomocí funkce „invalidate“. Ta zajistí zrušení platnosti minulých dat a zavolá nově metodu pro vykreslení. Zobrazení dat pomocí grafu se nachází na ukázce Zdrojového kódu 21, náhled na Obrázku 21.

```
JSONArray data = response.getJSONArray("GetLoggedMobilesResult");

List<PieEntry> pieEntries = new ArrayList<>();

int count = 0;
while (count < data.length()) {

   JSONObject jsonObject = data.getJSONObject(count);

   pieEntries.add(
              new PieEntry(jsonObject.getInt("Value"),
                           jsonObject.getString("Key")));

        count++;
    }

PieDataSet dataSet = new PieDataSet(pieEntries, "");
dataSet.setValueFormatter(new PieChartFormatter());
PieData pieData = new PieData(dataSet);

PieChart chart = (PieChart) findViewById(R.id.graph);

chart.setData(pieData);

chart.invalidate();
```

Zdrojový kód 21: Zobrazení grafu (zdrojem autor)



Obrázek 21: Ukázka vykreslení grafu (zdrojem autor)

6.2.5.6 Navázání spojení pomocí Bluetooth mezi dvěma aplikacemi

Pro navázání spojení mezi dvěma zařízeními pomocí Bluetooth je zapotřebí, aby jedno zařízení očekávalo připojení a druhé se na naslouchající připojilo. Z toho důvodu bylo třeba vytvořit dvě rozdílné třídy pro danou obsluhu. Ty využívají třídu „BluetoothSocket“, která mezi nimi udržuje spojení shodné se spojením pomocí TCP soketu. [16]

Dále jsou v systému vytvořeny třídy, které takovéto spojení mezi sebou budou udržovat, a data buď odesílají, nebo přijímají. Z takovéto architektury se nyní stává obdoba architektury Klient-Server. Serverová aplikace naslouchá klienta a reaguje na jeho komunikaci. Rozdílnou částí ovšem je, že obě dvě aplikace mohou vykonávat stejnou funkčnost.

Za účelem navázání jednoznačného spojení si obě aplikace navzájem vyměňují jednoznačný identifikační klíč (UUID), který se musí shodovat.

Ukázka výběru navázaní spojení je zobrazena na Obrázku 22.

Obrázek 22: Výběr typu akce Peer-to-Peer (zdrojem autor)

- **Implementace očekávání připojení**

  Pokud uživatel vybere možnost „přijmout zprávu“, je spuštěna aktivita očekávající připojení jiného zařízení. Pro neustálé očekávání připojení je v systému vytvořena asynchronní třída, běžící ve vlastním vlákně. Ta je neustále připravena na následné připojení. Třída při jejím vytvoření vytvoří instanci třídy „BluetoothServerSocket“. Po jejím spuštění třída očekává připojení jiné aplikace pomocí stejného identifikačního klíče. Když dojde k připojení jiné aplikace, je následně vytvořeno a uchováno spojení mezi nimi, které je opět spuštěno ve vlastním vlákně. Naslouchání připojení dalších aplikací je zastaveno.

```
private class ServerClass extends Thread{

    private BluetoothServerSocket serverSocket;

    public ServerClass(){
        serverSocket =
            bluetoothAdapter.listenUsingRfcommWithServiceRecord(
                                    APP_NAME, MY_UUID);
    }


    public void run(){
        BluetoothSocket socket = null;

        while (socket == null){

            socket = serverSocket.accept();

            if(socket != null){

                receive = new Receive(socket);
                receive.start();

                break;
            }
        }

    }

}
```

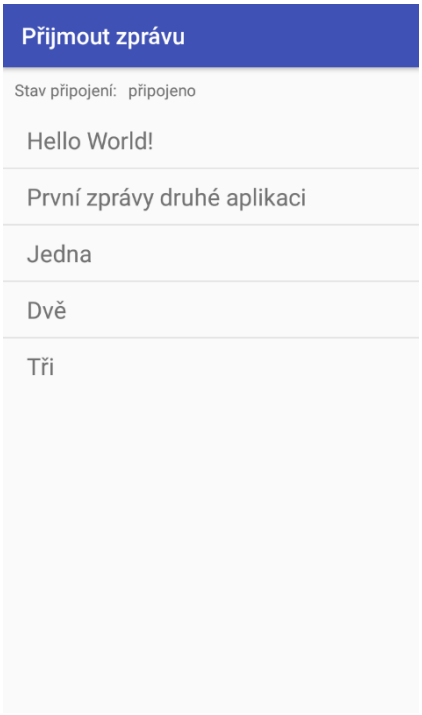
Zdrojový kód 22: Třída očekávající připojení (zdrojem autor)

Pro obsluhu příjmu zpráv je v systému vytvořena asynchronní třída „Receive“, která si uchovává spojení s druhou aplikací a neustále naslouchá příjmu dat a předává je handleru, který obstarává zobrazení dat uživateli.

```
private class Receive extends Thread{

    private final BluetoothSocket socket;
    private final InputStream inputStream;

    public Receive(BluetoothSocket socket){

        this.socket = socket;
        inputStream = this.socket.getInputStream();
    }

    public void run(){
        byte[] buffer = new byte[1024];
        int bytes;

        while(true){

            bytes = inputStream.read(buffer);
            handler.obtainMessage(STATE_MESSAGE_RECEIVED, bytes, -1,
                                  buffer).sendToTarget();

        }
    }
}
```

Zdrojový kód 23: Příjem dat z připojeného zařízení (zdrojem autor)

Ukázka zobrazení přijatých dat od jiné aplikace je zobrazena na Obrázku 23.



Obrázek 23: Zobrazení přijatých zpráv z druhé aplikace (zdrojem autor)

- **Implementace připojení**

  Pro připojení na naslouchající třídu je zapotřebí využít opačný přístup. Byla z toho důvodu vytvořena třída „ClientClass“, která se připojuje ke konkrétnímu zařízení, které vybere uživatel. Výběr zařízení je zobrazen na Obrázku 24. V případě úspěšného připojení je druhé aplikaci předán identifikační klíč, díky kterému se navzájem aplikace spárují. Po úspěšném připojení je vytvořena třída „Send“.

```
private class ClientClass extends Thread{

    private BluetoothDevice device;
    private BluetoothSocket socket;

    public ClientClass(BluetoothDevice device){
        this.device = device;

        socket = device.createRfcommSocketToServiceRecord(MY_UUID);
    }

    public void run(){

        socket.connect();

        sendReceive = new Send (socket);

    }

}
```

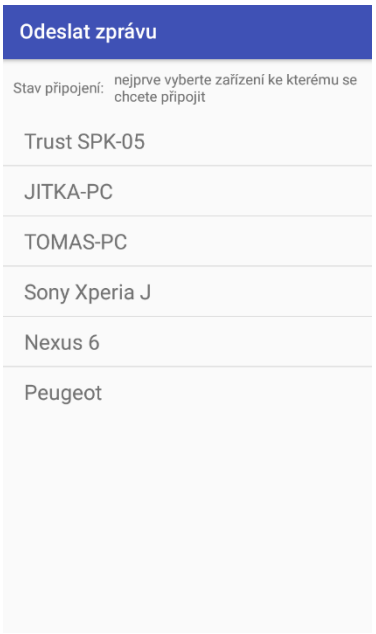
Zdrojový kód 24: Třída, která se připojuje na jiné zařízení (zdrojem autor)

Třída „Send“ udržuje navázané spojení s druhou aplikací. Pomocí metody „write“ umožňuje přeposlat druhé aplikaci data.

```
private class Send{

    private final BluetoothSocket socket;
    private final OutputStream outputStream;

    public Send (BluetoothSocket socket){
        this.socket = socket;
        OutputStream tmpOut = null;

        outputStream = this.socket.getOutputStream();
    }

    public void write(byte[] bytes){
        outputStream.write(bytes);
    }
}
```

Zdrojový kód 25: Odeslání dat k připojenému zařízení (zdrojem autor)

Z pohledu uživatele je nejprve potřeba vybrat zařízení, ke kterému se chce připojit. To je znázorněno na Obrázku 24. Poté, co vybere dostupného uživatele, je zahájen přenos zpráv. Ten je zobrazen na Obrázku 25.



Obrázek 24: Výběr zařízení pro připojení (zdrojem autor)

Obrázek 25: Odesílání zpráv jinému uživateli (zdrojem autor)

7 Shrnutí výsledků

Cílem bakalářské práce bylo ověřit a implementovat komunikaci mezi mobilními aplikacemi. Nejprve byla vytvořena webová služba, která přijímá data od klientů a komunikuje s databází. Ta je nasazená na veřejném serverovém úložišti. Druhou fází byla implementace aplikace, která dokáže komunikovat nejen s webovou službou, ale také s ostatními aplikacemi, bez připojení k serveru. K tomuto účelu byla v aplikaci využita technologie Bluetooth, která podporuje komunikaci mezi dvěma uživateli.

Webová služba podporuje přenos souborů, které si ukládá na vlastní serverové úložiště. Pomocí připojení Bluetooth k jinému zařízení je možné posílat textové zprávy druhému uživateli, které se mu následně zobrazí v aplikaci.

8 Závěry a doporučení

Závěrem práce doporučuji všem vývojářům pracovat s mobilními i serverovými aplikacemi.

Serverové aplikace jsou příjemným nástrojem na propojení více klientů, užívajících různá zařízení a operační systémy. Je ovšem potřeba, aby byl klient vždy připojený k internetu, pokud je serverová část nasazená na externím úložišti. V případě, že je uložená na lokální síti, stačí mít zajištěné připojení do dané lokální sítě.

Mobilní aplikace mají vzhledem k rostoucí míře poptávky velký potenciál do budoucího rozvoje. Velkým problémem klientské aplikace navrhované v bakalářské práci je multiplatformní unikátnost. Navrhujeme vyzkoušet vývojářské nástroje, které umožňují kompilaci výsledné aplikace do více mobilních platforem.

9 Seznam použité literatury

[1] PECKHAM, James, SWIDER Matt. *Android 7 Nougat: release date: when you'll get the update and every new features* [online]. Dostupné z http://www.techradar.com/news/phone-and-communications/mobile-phones/android-7-what-we-want-to-see-1311290, 31.07.2017 [cit. 2017-07-25]

[2] STATISTA. *Number of available applications in the Google Play Store from December 2009 to June 2017* [online]. Dostupné z https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/, 2017 [cit. 2017-07-25]

[3] DUNN, Jeff. *There´s is no hope of anyone catching up to Android and iOS* [online]. Dostupné z http://www.businessinsider.com/smartphone-market-share-android-ios-windows-blackberry-2016-8, 22.8.2016 [cit. 2017-07-28]

[4] KYPTA, Tomáš. *Vyvíjíme pro Android – úvod* [online]. Dostupné z https://www.svetandroida.cz/vyvijime-pro-android-1-uvod-201103/, 23.03.2011 [cit. 2017-08-01]

[5] LACKO, Luboslav. *Vývoj aplikací pro Android*. 1. vydání. Brno : Computer Press, 2015. 472s. 978-80-251-4347-6 [cit. 2018-01-17]

[6] Andoid Developers. *Activity* [online]. Dostupné z https://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle, [cit. 2018-01-17]

[7] ZEZULKA, František, HYNČICA, Ondřej. *Průmyslový Ethernet II: Referenční model ISO/OSI*. Automa, 2007, roč. 13, č. 3, s. 86-90 [cit. 2018-01-18]

[8] KEJDUŠ, Radek. *Technologie počítačové sítě: jak pracuje TCP/IP a ISO/OSI* [online]. Dostupné z https://www.cnews.cz/technologie-pocitacove-site-jak-pracuje-tcpip-a-isoosi/, 2012 [cit. 2018-01-21]

[9] DOUGLAS, K. Barry. *Web service explained* [online]. Dostupné z https://www.service-architecture.com/articles/web-services/web_services_explained.html, [cit. 2018-01-21]

[10] BOOTH, David, HUGO, Haas, MCCABE, Francis, NEWCOMER, Eric, CHAMPION, Michael, FERRIS, Chris, ORCHARD, David. *Web service architecture* [online].

Dostupné z https://www.w3.org/TR/ws-arch/#WSDL, 2004 [cit. 2018-01-21]

[11] QUORA, N. Singh, *What is web.config file? What is the use of the config file in web appliactions?* [online]. Dostupné z https://www.quora.com/What-is-web-config-file-What-is-the-use-of-the-config-file-in-web-applications, [cit. 2018-04-10]

[12] CODEPROJECT, K. Shukla, *Create RESTful WCF Service API: Step By Step Guide* [online]. Dostupné z https://www.codeproject.com/Articles/105273/Create-RESTful-WCF-Service-API-Step-By-Step-Guide, 30. 08. 2010, [cit. 2018-04-10]

[13] ANDROID DEVELOPER, *<uses-sdk>* [online]. Dostupné z https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels, [cit. 2018-04-10]

[14] ANDROID DEVELOPER, *Transmitting Network Data Using Volley* [online]. Dostupné z https://developer.android.com/training/volley/index.html , [cit. 2018-04-10]

[15] GITGUB, *MPAndroiChart* [online]. Dostupné z https://github.com/PhilJay/MPAndroidChart/wiki, 27.12.2017 [cit. 2018-04-10]

[16] ANDROID DEVELOPER, *BluetoothSocket* [online]. Dostupné z https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html , [cit. 2018-04-10]

[17] SYSTEM ONLINE, J. ŠTUMPF. *Webové služby a XML* [online]. Dostupné z https://www.systemonline.cz/sprava-it/webove-sluzby-a-xml.htm, 10.2006 [cit. 2018-04-17]

[18] SMARTBEAR, J. Mueller. *Understanding SOAP and REST Basics and differences* [online]. Dostupné z https://blog.smartbear.com/apis/understanding-soap-and-rest-basics/, 8.1.2013 [cit. 2018-04-18]

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2016/2017

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Hovorka Tomáš	Na Trávníku 1237, Rychnov nad Kněžnou	I14078

TÉMA ČESKY:

Mobilní aplikace využívající principy klient-server a peer-to-peer komunikace

TÉMA ANGLICKY:

Mobile application uses the principles of client-server and peer-to-peer communication

VEDOUCÍ PRÁCE:

doc. Ing. Filip Malý, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl - Navrhnout a implementovat mobilní aplikaci, která bude komunikovat s ostatními mobilními aplikacemi s využitím principů klient-server a peer-to-peer.

1. Úvod
2. Principy komunikace klient-server a peer-to-peer
3. Analýza a návrh
4. Implementace
5. Zhodnocení výsledků
6. Závěr

Literatura

SEZNAM DOPORUČENÉ LITERATURY:

Lacko: Vývoj aplikací pro android
https://www.svetandroida.cz/vyvijime-pro-android-1-uvod-201103/

Podpis studenta: Hovorka ..... **Datum:** 10.10.2017

Podpis vedoucího práce: ..... **Datum:** 10.10.2017