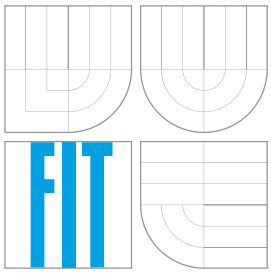
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KLIENT VIRTUÁLNÍ ČEKÁRNY PRO ČEKÁRNU PROVOZOVATELE

VISTUAL WAITING ROOM CLIENT FOR SERVICE PROVIDER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

RADIM KŘEK

VEDOUCÍ PRÁCE
SUPERVISOR

MARTIN KOLÁŘ, M.Sc.

BRNO 2016

Zadání bakalářské práce/18797/2015/xkrekr00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Křek Radim**
Obor: Informační technologie
Téma: **Klient virtuální čekárky pro čekárnu provozovatele služeb**
Vistual Waiting Room Client for Service Provider
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s algoritmy pro realizaci klienta "virtuální čekárny" na mobilní platformě.
2. Vytvořte koncepci klienta "virtuální čekárny" pro mobilní platformu, například tablet, s operačním systémem Android a diskutujte vlastnosti.
3. Navrhněte postup implementace klienta do čekárny provozovatele služeb a diskutujte možnosti takové implementace.
4. Implementujte klienta čekárny a demonstrujte funkčnost na vhodném příkladě.
5. Diskutujte dosažené výsledky a možnosti pokračování práce.

Literatura:

- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3 zadání

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

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% 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: **Kolář Martin, M.Sc.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

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

Na různých místech se často musí čekat a tvoří se dlouhé fronty a nikdo neví kdo je právě na řadě. Proto se čím dál častěji vytváří systémy založené na virtuálních frontách, pro ulehčení těchto situací. Tato práce se zabývá vytvořením aplikace pro systém Android, sloužící jako klient systému založeném na virtuálních frontách. Popisuje také vytvoření serveru a komunikačního protokolu jakožto nezbytné komponenty pro fungování aplikace.

Abstract

Very often we get into situation where we have to wait a long time in queue wher nobody knows who's next. Therfore we have more systems based on Virtual queues. This bachelor's thesis deals with a description of the creation of Android application which will work as client for Virtual queue based system. It also describes the server and communication protocol as an essential components for the correct functioning of the application.

Klíčová slova

Android, virtuální fronta, vyvolávací systém, XML

Keywords

Android, virtual queue, systems for calling up, XML

Citace

Radim Křek: Klient virtuální čekárny pro čekárnu provozovatele, bakalářská práce, Brno, FIT VUT v Brně, 2016

Klient virtuální čekárny pro čekárnu provozovatele

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Martina Koláře, M.Sc.

.....

Radim Křek

16. května 2016

© Radim Křek, 2016.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Kapitola 1

Úvod

V dnešní moderní přetechnizované době se setkáváme s počítači nebo jinou elektronikou téměř všude. To vede lidi k digitalizaci veškerých dat a využívání technologií k co největšímu ulehčení práce. Příkladem můžou být různé rezervační systémy.

Cílem souboru bakalářských prací na ÚPGM Fakulty Informačních technologií, je vytvořit systém Virtuální čekárny, který by poskytl provozovatelům služeb možnost kontrolovat vytíženost jednotlivých služeb. Zároveň dává zákazníkům možnost rezervovat si služby na přesný čas nebo se zařadit do fronty sledovat kdy se dostane na řadu.

Tato bakalářská práce se zabývá návrhem a implementací klienta pro virtuální čekárnu, umístěného přímo v čekárně poskytovatele. Ten bude reprezentován aplikací vytvořenou speciálně pro tablet se systémem android. Aplikace bude umožňovat zákazníkovi zařazení do fronty podle jeho výběru.

V následující kapitole se věnuji popsání principu Virtuální čekárny a představení již existujících systémů. Další kapitola se poté zabývá technologiemi, které byly při vývoji použity. Velkou část kapitoly zabírá popis platformy Android. Jsou zde popsány jeho součásti a životní cyklus aplikace. V kapitole zabývající se návrhem aplikace je také uveden popis testovacího serveru, který byl vytvořen pro potřeby testování aplikace. Poslední kapitola se zabývá již samotnou implementací aplikace.

Kapitola 2

Virtuální čekárna

Ve světě se s pojem „Virtuální čekárna“ moc nesetkáme. Častější pojmenování bývá „Virtuální fronta“, které i lépe vystihuje podstatu systému.

Virtuální fronta je koncept, který je využíván pro zvětšení efektivity a zmírnění zátěže systému příchozími požadavky. Vede také ke většímu uživatelskému komfortu. Virtuální fronty nemají žádnou striktní formu, proto se s nimi můžeme setkat v nejrůznějších podobách a pracovních odvětví.

Nejčastěji virtuální frontu využívají Call-centra, která ji využívají pro rozdělení volajících klientů mezi operátory. V takovém případě funguje Virtuální fronta jako mezistupeň klienta a operátora. Pokud by Call-centrum Virtuální frontu nemělo, musí klient čekat na telefonu, doku nebude volný operátor, nebo zavěsí a pokusí se dovolat později obr.2.1. V případě využití virtuální fronty je po zavolání klient zařazen do databáze čekajících klientů a může zavěsit. V momentě kdy je volný operátor je klientovi zavoláno zpět. Obr.2.2.

U virtuálních front se můžeme setkat s mnoha typy. Majoritní většinou jsou fronty využívající systém FIOF[1], která která obsluhuje nejdříve ty zákazníky, kteří do ní vstoupili dříve. Občas je také využíváno typu kdy je ze zařazených zákazníku náhodně vylosován jeden a ten je obsloužen.

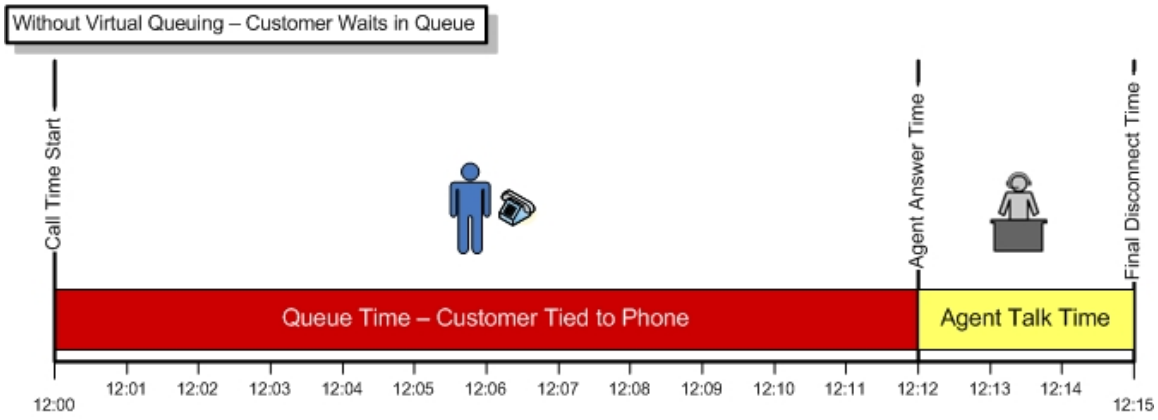
2.1 Vyvíjený systém Virtuální čekárny

Systém Virtuální čekárny, vyvíjený v rámci několika bakalářských prací na Fakultě Informačních Technologií VUT v Brně, je software umožňující jakémukoliv poskytovateli nakonfigurovat si Virtuální čekárnu podle vlastních preferencí.

Zákazník bude mít možnost vstoupit a interagovat s frontou prostřednictvím několika klientů v závislosti na situaci a jeho volbě. Nejdůležitějším klientem je tzv. mobilní klient, který bude mít zákazník v podobě mobilní aplikace nainstalovaný ve svém chytrém telefonu či tabletu, připojenou na server, který obsahuje čekárnu, ke které chce mít zákazník přístup.

Alternativou, pro zákazníky bez chytrých zařízení, je tablet fyzicky umístěný přímo

[1] First In, First Out



Obrázek 2.1: Call-Centrum bez virtuální fronty

Obrázek 2.2: Call-Centrum s virtuální frontou

v čekárně poskytovatele, ne kterém bude předinstalována a spuštěna speciální aplikace, podobná mobilnímu klientu, umožňující zákazníkovi vstup do Virtuální čekárny. Právě tímto klientem se zabývá tato bakalářská práce.

2.2 Již existující systémy

Virtuální fronty a systémy na nich založené jsou velmi prestižním tématem. Díky tomu se na trhu pohybuje velké množství firem, které tyto systémy vytváří. I když na trhu najdeme těchto systémů velké množství všechny se dají zařadit do úzké kategorie v rámci Virtuálních front. Většinou se totiž jedná o „Vyvolávací systémy“.

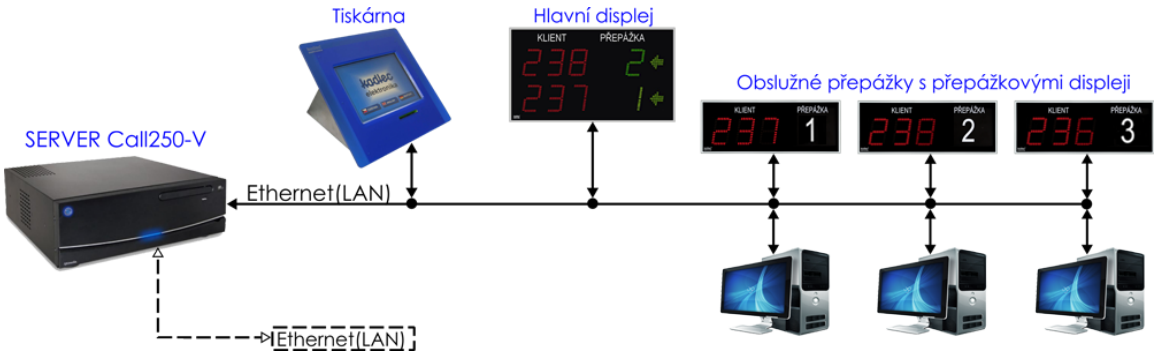
Vyvolávací systémy působí pouze v místě provozovatele a slouží k rozložení příchozích zákazníků mezi jednotlivé přepážky. Systém bývá centralizovaný, složený z více komponent připojených k hlavnímu serveru. Občas se u vyvolávacích systémů setkáme s možností rozšíření o on-line přístup k systému.

2.2.1 Systémy Call

[4] S těmito vyvolávacími systémy se setkáme pravděpodobně nejčastěji. Objevují se totiž ve velké míře na úřadech či poštách. Systém existuje v několika verzích, ale architektura je vesměs stejná. Dále se budu zabývat verzí Call 250-V, z důvodů jeho podobnosti k vyvíjenému systému Virtuální čekárny.

Komunikace mezi jednotlivými komponentami systému je řešena pomocí síťové komunikace. Všechny komponenty jsou tedy připojeny na podnikovou síť a komunikují se serverm, který řídí celý systém. Celou architekturu lze vidět na obrázku 2.3. Tento systém lze napojit na systém WebCall od stejné firmy. Tím získá zákazník možnost on-line objednání termínu či přehled aktuálního stavu vytíženosti jednotlivých přepážek.

Ukázku systému s připojeným webovým rozhraním lze vidět na stránkách města Ostrava.[2]



Obrázek 2.3: Schéma systému Call 250-V

Převzato z: `http://www.kadlecelektro.cz/produkty/vyvolavaci-systemy/call-250-v/`

2.2.2 Systém EVIPA

[5][1][6] Evipa[3] je systém navržený Martinem Horským, Studentem Faktulty Elektrotechniky a komunikačních technologií VUT. Systém není univerzální, ale byl vyvinut konkrétně pro lékařská zařízení. Systém je složen ze dvou prvků. Elektronického terminálu v čekárně, který funguje jako recepce. Do něj vloží příchozí pacient kartičku pojišťovny čímž je ohlášen v ordinaci lékaře. Druhou komponentou systému je počítač v ordinaci, na kterém běží speciální software umožňující komunikaci s terminálem v čekárně. Na počítači je poté možno sledovat kdo se nachází v čekárně případně jestli byl objednán, nebo se vrací z vyšetření.

[2] https://objednavky.ostrava.cz/eobs/
[3] Elektronická evidence pacientů

Kapitola 3

Použité technologie

Tato kapitola pojednává o technologiích, které byly využity pro tvorbu aplikace. Hlavním tématem je popis platformy Android, na kterou byla aplikace cílena.

3.1 Android

Android je operační systém, který je převážně cílen na mobilní hardwarové platformy. Systém je založen na linuxovém jádře a obsahuje základní uživatelské rozhraní.

Android byl vyvíjen společností Android Inc., ale tu velmi brzo odkoupila společnost Google, která je hlavním vývojářem systému až do dnešních dob. Významným spolu účastníkem vývoje systému je konsorcium OHA[1] která sdružuje významné technologické společnosti působících na poli mobilních platforem a výrobků s nimi spojenými. Mezi tyto společnosti patří např. HTC, Intel, LG, Motorola, Qualcom a další. Cílem konsorcia je vývoj otevřeného standard pro mobilní zařízení. Pro vývoj aplikací se využívá Android SDK[2], jenž poskytuje API[3] a potřebné nástroje. Aplikace jsou většinou vyvíjeny v programovacím jazyce Java.

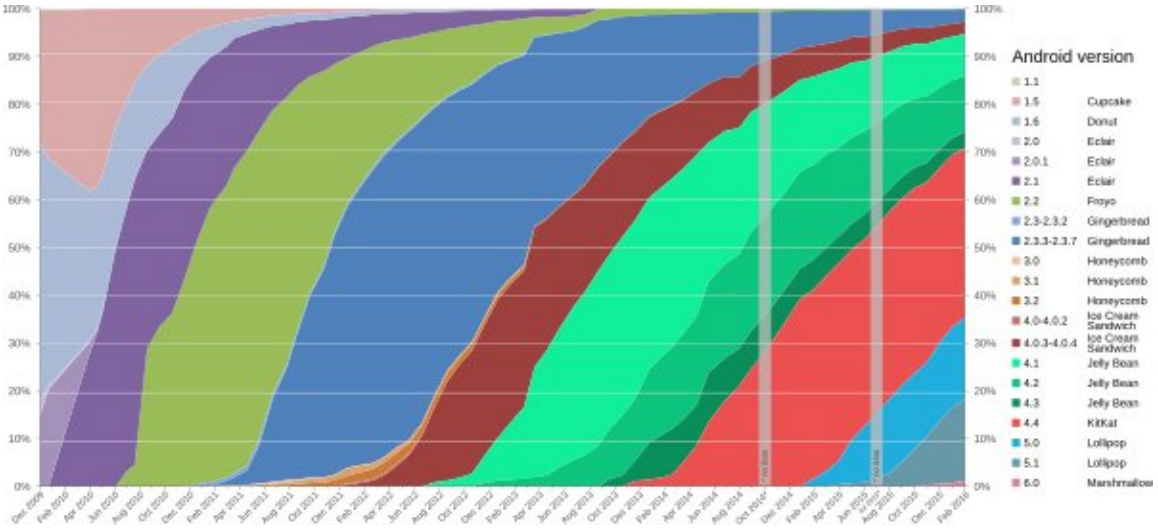
3.1.1 Verze systému Android

Systém je vyvíjen postupně od roku 2003. Dne 5. 11. 2007 kdy bylo založeno konsorcium OHA byl také Android oficiálně veřejně představen. První telefon s tímto systémem se na trh dostal roku 2008 a byl jím T-mobile G1, který byl vyroben firmou HTC. Společně s tímto telefonem byl uveden i Androoid SDK 1.0.

Od té doby bylo vydáno mnoho verzí. Specifikem je že každá verze kromě číselného označení má „kódové“ označení, které vždy začíná následujícím písmenem abecedy než bylo označení předchozí verze a je názvem nějaké sladkosti. K datu 1. 3. 2016 je nejnovější verzí Android 6.0.1 Marshmallow. Bylo ale již ohlášeno že se pracuje na nové verzi s inter-

[1] Open Handset Aliance
[2] Software Development Kit
[3] Application Programming Interface

ním označením „N“. Přesto že má Android již mnoho verzí, tak kvůli ukončení podpory od výrobců zařízení, se objevuje velké množství uživatelů, kterým běží na svém zařízení starší verze systému. Z tohoto důvodu byla aplikace vyvíjena pro Android verze 4.4, díky čemuž ji bude možno spustit na více jak 50% uživatelských zařízeních. Přibližné využívání jednotlivých verzí je možné vidět na obrázku 3.1.



Obrázek 3.1: Graf rozšířenosti jednotlivých verzí Systému

3.1.2 Architektura systému

Systém je složen ze softwarových komponent, které jsou rozděleny do pěti sekcí a čtyřech hlavních vrstev viz obrázek 3.2. Hranice mezi jednotlivými sekcemi je však někdy hůře definovatelná.

Linuxové jádro

Nejnižší vrstvou architektury je jádro systému, které je založeno na Linuxovém jádru. Díky tomu získal Android jednu z jeho hlavních výhod a to jednoduchou modifikaci pro různá zařízení. Android také těží z dalších funkcí Linuxu a to např. správa paměti, zabudované ovladače či souběžný běh aplikací, které běží jako samostatné procesy. Díky všem těmto vlastnostem napomáhá Linuxové jádro k zvýšení stability celé architektury.

Knihovny

V této vrstvě nalezneme knihovny založené jak na Javě tak na C/C++ které zapouzdřují klíčové funkce systému. Tyto knihovny jsou vývojářům poskytnuty pomocí Application frameworku.

Android runtime

Pravděpodobně nejdůležitější částí celé aplikace je Android Runtime. Tato část se nachází v druhé vrstvě a obsahuje „Dalvik Virtual Machine“, což je virtuální stroj velmi podobný JVM[4]. Důvodem proč byl vyvinut vlastní virtualizační nástroj jsou licenční podmínky. Java a její knihovny jsou volně přístupné, kdežto JVM nikoliv. Dalším důvodem bylo provedení optimalizací pro mobilní platformy a to zvláště v oblasti úspory energie.

Při překladu aplikace pro android, neprobíhá překlad přímo do Dalvik byte kódu, ale je nejdříve přeložena klasickým Java kompilátorem do Java byte kódu a teprve poté přeložena do Dalvik byte kódu. Při spuštění běží každá aplikace ve vlastním procesu s vlastní instancí Dalvik Virtual Machine.

Application framework

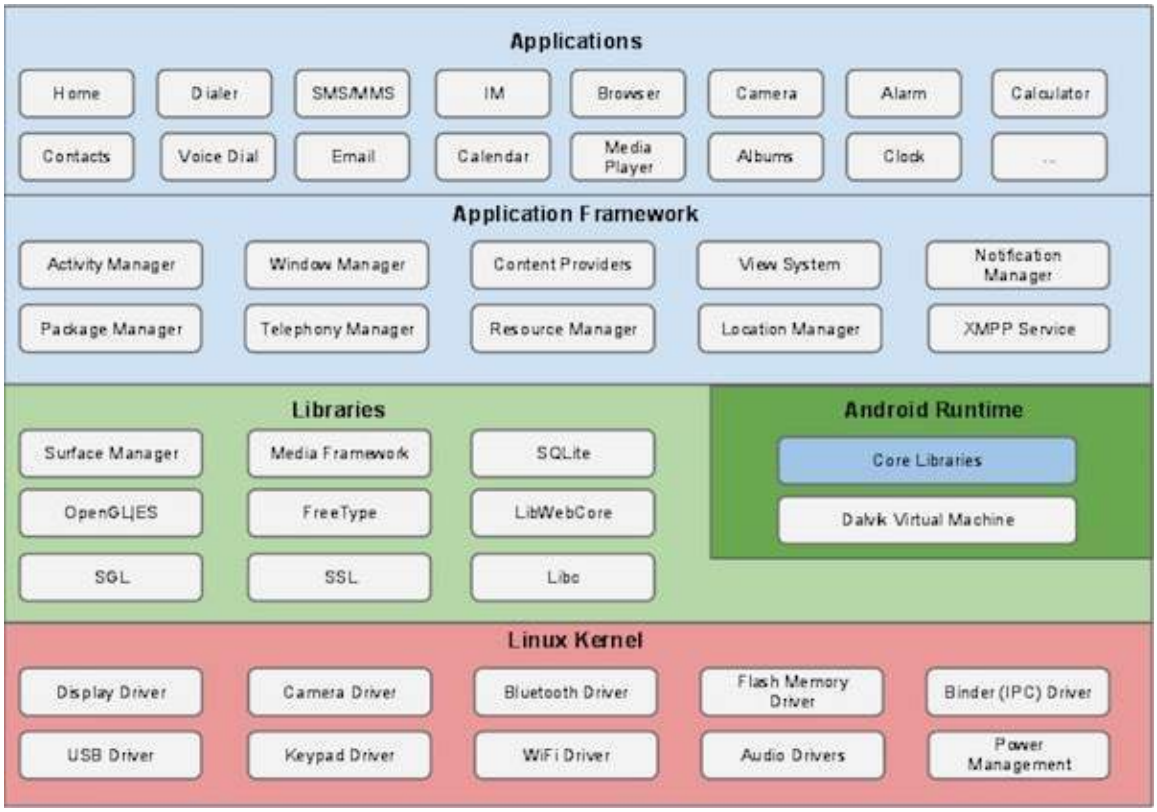
Tato vrstva poskytuje velké množství vysoko úrovňových služeb, které můžou vývojáři používat pro své aplikace. Tyto služby jsou jim poskytnuty jako knihovny jazyka Java. Klíčovými službami jsou:

- **Activity Manager** - řídí životní cyklus aplikace
- **Content Providers** - zajišťuje sdílení dat mezi jednotlivými aplikacemi
- **Notifications Manager** - díky této knihovně může aplikace vyvolávat upozornění a notifikace pro uživatele
- **View System** - obsahuje soubor pohledů (Views), které umožňují vytvořit uživatelské rozhraní aplikace

Aplikace

Tuto vrstvu není třeba popisovat do detailu, jelikož s ní se dostávají uživatelé do styku neustále. Vrstvu totiž tvoří veškeré aplikace, které jsou v mobilním zařízení obsaženy. A to jak vestavěné tak i později doinstalované aplikace.

[4] Java Virtual Machine



Obrázek 3.2: Základní vrstvy systému Android

3.1.3 Základní části aplikace pro Android

Při programování aplikace, je nutné pracovat s různými prostředky, které nám SDK nabízí. Avšak nalezneme zde několik prvků, které jsou využívány téměř vždy. Dalo by se říci že jsou základními stavebními kameny každé aplikace. Vytvoříme je jako třídu dědící od korespondující třídy, která se nachází v základní knihovně systému Android.

Android Manifest

Jedná se o XML soubor, který je obsažen v každé aplikaci. V tomto souboru jsou zapsány hlavní nastavení aplikace. Např. určení cílové verze SDK pro kterou je aplikace určena nebo speciální knihovny které aplikace využívá. Dále lze také v manifestu nalézt seznam oprávnění které má aplikace povoleny. Velmi důležitou informací, kterou v manifestu nalezneme, je určení výchozí Aktivity vyvíjené aplikace. Viz obrázek 3.3

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Obrázek 3.3: Nastavení hlavní aktivity

Activity

Aktivita je základním prvkem při vývoji uživatelského rozhraní. Jedna instance aktivity reprezentuje jednu „obrazovku“ pro interakci s uživatelem.

Většina aplikací se sestávají z více než jedné Aktivity, kdy každá je nezávislá na na ostatních. Avšak je nutné mít v aplikaci jednu Aktivitu, která je označena jako výchozí a je zobrazena při spuštění aplikace. Každá aktivita má možnost spustit jinou aktivitu a předat jí řízení. Při zobrazení jiné Aktivity, je předchozí pozastavena, ale není zrušena pouze uložena na zásobník. Aktivita, která se nachází na vrcholu zásobníku je aktivní, má focus, a komunikuje s uživatelem.

Aktivita se může nacházet v různých stavech. Při přechodech mezi jednotlivými stavy se volají klíčové metody aktivity, díky kterým je možno např. připravit data před vykreslením aktivity. Nejdůležitější metodou je metoda `onStart()`, která se volá při vytvoření aktivity nebo při jejím obnovení. Využívá se pro nastavení klíčových zdrojů, které jsou potřeba během celého běhu aktivity.

Přehled všech stavů a metod je uveden na obrázku 3.4.

Intent

Intent je komponenta úzce spjatá s Aktivitami. jedná se o Asynchroní zprávy využívající se pro zasílání zpráv mezi jednotlivými komponentami aplikace a právě převážně mezi Aktivitami.

Intent je také využívám ve chvíli, kdy je zapotřebí otevřít nové okno (aktivitu) aplikace.

Service

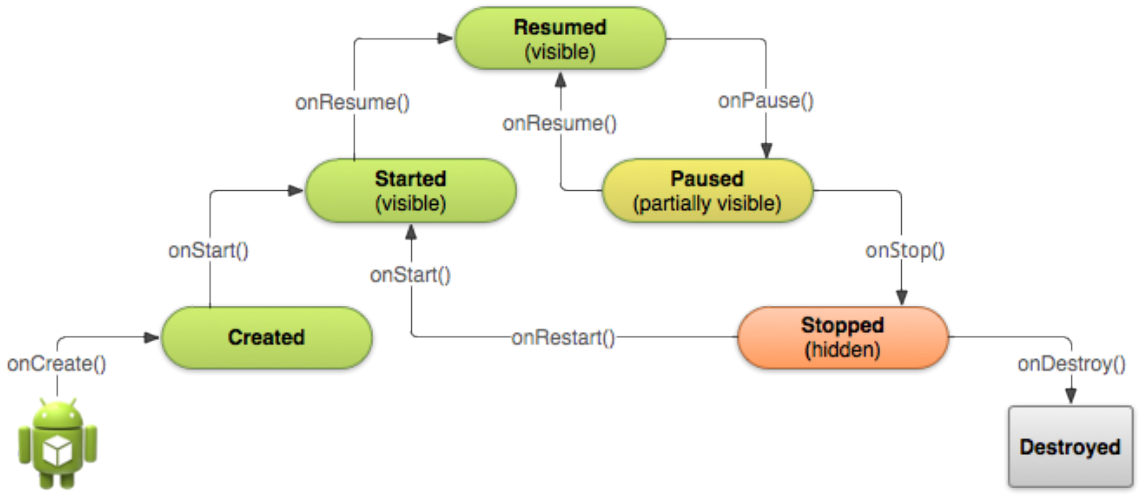
Service, služba, je komponenta, která běží na pozadí a stará se o zpracování dlouho trvajících akcí. Díky tomu je možné aby aplikace zpracovávala data a neblokovala uživateli práci s jinými aplikacemi nebo některou z aktivit aplikace.

Broadcast reciever

je stejně jako Service nevizuální komponenta a využívá se pro zachytávání oznámení od jiných aplikaci nebo systému a k reakci na ně. Příkladem oznámení může být zpráva o nízkém stavu baterie. Aplikace jako taková může využívat buďto systémové broadcast recievery, nebo si vytvořit vlastní.

Content provider

Content provider je komponenta, která se slouží k poskytování dat mezi aplikacemi ale i mezi jednotlivými aktivitami v aplikaci. Nezáleží na tom zda jsou data uložena v databázi nebo souborovém systému. Content provider poskytuje rozhraní pro práci s těmito daty a aplikace nemusí řešit jak data získat. K těmto datům mají přístup i ostatní aplikace pokud je povoleno.



Obrázek 3.4: životní cyklus aktivity

3.2 XML

XML celým názvem „Extensible Markup Language“ je rozšiřitelný značkovací jazyk. Je vyvíjen konsorciem W3C, které má veřejně přístupné jeho standardy. XML bylo vytvořeno kvůli potřebě standardizovaného formátu zasílání informací, aby na přečtení příchozí zprávy nemusel mít příjemce konkrétní software ve kterém byla zpráva vytvořena. V dnešní době takovýmto speciálním typem souboru je např. formát souborů „.docx“.

Od počátků vývoje bylo u XML dbáno na podporu i jiných jazyků než angličtiny, proto bylo jako defaultní kódování zvolen formát „Unicode“, s tím že je připuštěno změnit toto kódování na libovolné jiné.

XML se vyznačuje vysokou informační hodnotou. Pomocí XML značek je možno naznačit význam jednotlivých částí dokumentu.

DTD

Kvůli své rozšiřitelnosti se XML také často nazývá meta jazykem, nadřazeným značkovacím jazykem, protože je v rámci XML možné definovat vlastní jazyky pomocí DTD[5] popisu.

XML nedefinuje žádné značky. Dokumenty psané v XML obsahují vždy set vlastních značek, vytvořených speciálně pro daný dokument. Tyto značky je možno definovat v již zmíněném DTD souboru, avšak je to nepovinné. Při použití DTD lze automaticky kontrolovat vytvářený dokument, zda odpovídá definici v něm napsané.

XSLT a Zobrazení XML

Jazyk XML je zaměřen pouze na předání informačních hodnot. Neobsahuje žádné prostředky, pro definici stylu dokumentů. Tím vzniká naprosté oddělení informační a vzhledové vrstvy. V případě potřeby je možno pro definici stylů použít jiný jazyk. Mezi nejznámější patří CSS[6].

Další možností, která se u XML využívá je XSLT[7]. Tyto předpisy nedefinují na rozdíl od CSS vzhled, ale jsou využívány pro transformace XML do jiného formátu. Pro transformaci je kromě zdrojového XML souboru zapotřebí šablona, podle které se transformace provádí.

[5] Document Type Definition
[6] Cascading Style Sheets
[7] Extensible Stylesheet Language Transformations

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="example.xsl"?>
<Article>
  <Title>My Article</Title>
  <Authors>
    <Author>Mr. Foo</Author>
    <Author>Mr. Bar</Author>
  </Authors>
  <Body>This is my article text.</Body>
</Article>
```

Příklad XML
Zdroj: https://developer.mozilla.org/en-US/docs/Web/API/XSLTProcessor/Basic_Example

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:template match="/">
    Article - <xsl:value-of select="/Article/Title"/>
    Authors: <xsl:apply-templates select="/Article/Authors/Author"/>
  </xsl:template>
  <xsl:template match="Author">
    - <xsl:value-of select="." />
  </xsl:template>
</xsl:stylesheet>
```

Příklad XSLT šablony pro převod XML z předchozího příkladu
Zdroj: https://developer.mozilla.org/en-US/docs/Web/API/XSLTProcessor/Basic_Example

```
Article - My Article
Authors:
- Mr. Foo
- Mr. Bar
```

Výstup XSLT transformace
Zdroj: https://developer.mozilla.org/en-US/docs/Web/API/XSLTProcessor/Basic_Example

Kapitola 4

Návrh aplikace

V této kapitole rozeberu návrh aplikace a testovacího serveru, který byl vytvořen pro potřeby testování aplikace.

4.1 Funkčnost aplikace

Aplikace klienta pro čekárnu provozovatele, bude plnit úlohu přístupového terminálu, který bude využíván klienty, kteří přijdou přímo do čekárny provozovatele využít poskytovaných služeb. Aplikace poskytuje klientům základní možnosti jak interagovat se systémem Virtuální Čekárny. Klient v čekárně pro provozovatele má však omezenou funkčnost a dovoluje zákazníkovi pouze vstoupit do fronty podle jeho výběru a sledovat aktuální stav fronty. Celá aplikace je pak rozdělena na dvě hlavní část. Veřejnou a neveřejnou, ke které má přístup pouze správce.

4.1.1 Chráněná část

Celý systém Virtuální čekárny je koncipován tak, aby byl lehce přenositelný a modifikovatelný podle potřeb poskytovatele. Je také navržen tak aby nebyl závislý na jednom serveru. Z toho důvodu je potřeba aplikaci klienta do čekárny vytvořit tak aby ji bylo možno zadat na které adrese se nachází server ze kterého bude čerpat data. Jelikož se jedná o velmi citlivou informaci je potřeba ji umístit do části ke které obyčejný uživatel aplikace nebude mít přístup. Nazývejme dále tuto část jako Nastavení.

Do Nastavení bude mít přístup výhradně „správce aplikace“, kromě jedné výjimečné situace, který se do Nastavení dostane po zadání hlavního hesla aplikace. Jedinou výjimkou je situace, kdy je aplikace spuštěna poprvé a není nakonfigurována. Tehdy není přístup do Nastavení nijak blokována. Naopak je aplikace vždy po spuštění přesměrována do sekce Nastavení a čeká na vyplnění konfiguračních dat.

Hlavními konfiguračními informacemi jsou IP serveru, ze kterého bude Klient čerpat veškerá data a Hlavní heslo aplikace, kterým bude později sekce Nastavení chráněna.

4.1.2 Veřejná část

Hlavní úlohou aplikace je umožnit zákazníkovi vstoupit do fronty systému. Proto je důležité aby tato funkce byla umístěna na přehledném a lehce dostupném místě. K tomu se hodí nejlépe hlavní obrazovka aplikace nacházející se ve veřejné části. Společně s tlačítkem, které bude obstarávat samotné odeslání požadavku do systému, je potřeba umožnit zákazníkovi vybrat frontu do které se chce zařadit. Ten také očekává nějaký identifikátor, například číselný údaj určující kolikátý zákazník již dnes je, podle kterého se bude moci dohledat ve frontě a kterým jej provozovatel upozorní že je již na řadě.

Druhým prvkem, který zde bude umístěn, avšak také velmi důležitým je výpis aktuálního stavu právě vybrané fronty. Nejlepší bude grafické zobrazení v podobě seznamu nebo tabulky, aby to poskytlo zákazníkovi co nejvíce informaci bez toho, aniž by musel dané zobrazení podrobně zkoumat.

4.2 Testovací server

Pro plnou funkčnost aplikace je velmi důležitý server, který poskytuje data. Tento server byl zadáním jiné bakalářské práce, avšak v době vývoje aplikace nebyl k dispozici. Z toho důvodu bylo nutné vytvořit si vlastní testovací server, který bude poskytovat alespoň základní služby, pro správný běh aplikace.

Server byl postaven jako webová aplikace s databází, aby byla schopna dynamicky reagovat na požadavky od klienta a neposkytovala pouze statická data. Odpovědi vrací server v podobě XML.

4.2.1 Backend serveru

Jak již bylo zmíněno výše, server je vytvořen jako webová aplikace. Je postaven na skriptovacím programovacím jazyce PHP ve verzi 5.6.

Pro rychlý a jednoduchý vyvoj nebyl server psán v čistém PHP, ale byl použit česky Open-Source Nette Framework[1] od Nette Foundation. Ten poskytuje velkou sadu knihoven založenou na MVC[2] modelu, pro vývoj aplikace.

Server je navržen tak, aby i přes vlastní způsob implementace logiky a komunikačního protokolu, byl jednoduše převeditelný na oficiální komunikační protokol, který byl předmětem jiné bakalářské práce.

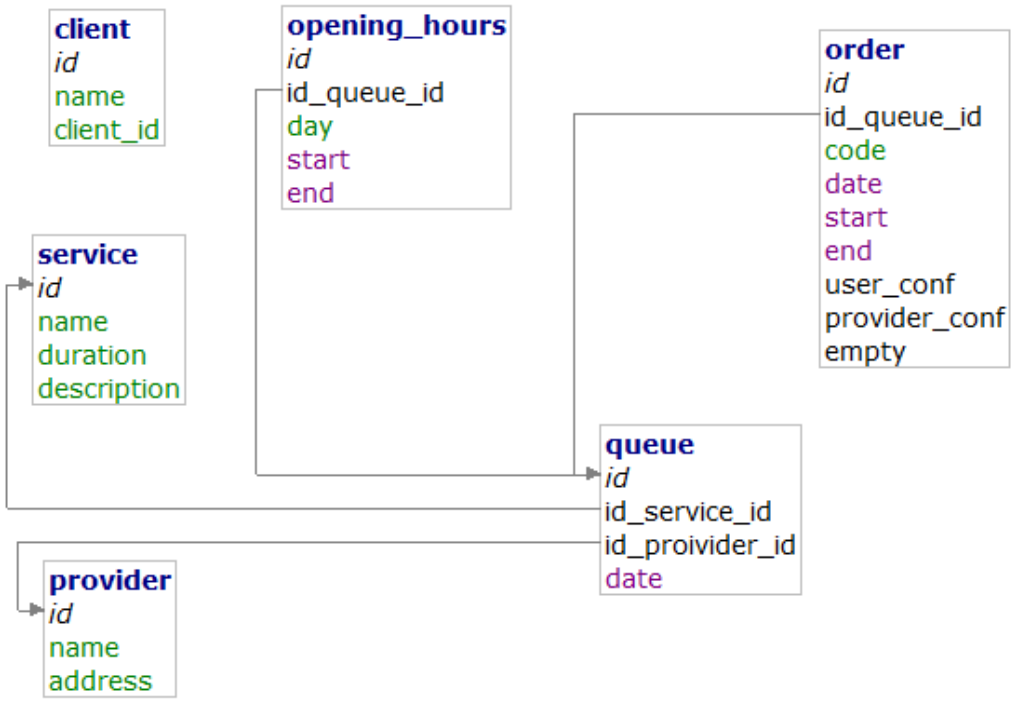
[1] http://www.nette.org
[2] Model View Controler

4.2.2 Databázová část

Server si ukládá informace o připojených klientech, poskytovatelích nebo o aktuálním stavu fronty. Pro tento účel se nejlépe hodí využít databázi. Kvůli rozšířenosti a jednoduchosti využití byl zvolen databázový systém MySQL.

Databáze byla vytvořena podle návrhu Mateje Sadloně, jehož zadáním bakalářské práce bylo vytvořit právě databázovou část k systému „Virtuální čekárny.“ Schéma výsledné databáze je vidět na obrázku 4.1.

Databáze byla vytvořena pomocí Addonu do Nette Frameworku. Přesněji pomocí ORM[3] databázové knihovny Doctrine, která umožňuje objektový návrh a práci s databází. Dále díky této knihovně není potřeba vytvářet složité dotazy pro získání dat z databáze, ale pouze volat metody této knihovny a ta výsledný dotaz vytvoří a to v jazyku, podle toho, jaký typ databáze je používán. Výhodou je také že každý řádek tabulky je reprezentován „Entitou“, PHP objektem, který je knihovnou automaticky naplněn daty z databáze.



Obrázek 4.1: Schéma databáze serveru

[3]Object-relational mapping

4.3 Komunikační protokol

Pro správné fungování celé aplikace je ale potřeba aby si obě předchozí části mezi sebou vyměňovali informace. K tomu bude sloužit komunikační protokol.

V této části se budu věnovat popisu protokolu, který byl využíván v průběhu vývoje aplikace a k jejímu testování. Ten se od oficiálního protokolu, který bude využívám v celém systému Virtuální čekárny, mírně liší. A to především v komplexnosti požadavků a odpovědí, kde oficiální protokol je více univerzální.

4.3.1 Princip dotazování serveru

Komunikace směrem od klienta k serveru, je založena na HTTP REST[4] API. Klient tedy komunikuje se serverem podle přesně zadaných webových adres, na kterých nalezne odpověď od serveru.

Adresy se mění podle aktuálního požadavku klienta, ale stále dodržují předem daný formát. Např. pro získání informací může vypadat adresa následovně:

```
http://www.server.cz/get/<object>/<parameter>/<clientId>
```

Kde sekce uvedené ve špičatých závorkách se mění v závislosti na objektu, o kterém se klient snaží získat informace od serveru.

Tím že se jedná pouze o testovací komunikační protokol další popis univerzálních adres neobsahuje. Zde uvádím celkový přehled API adres, které jsou testovacím protokolem definovány.

- **Autorizační adresa** - Adresa, kterou musí kontaktovat klient jako první, aby se u serveru autorizoval a dostával od něj odpovědi
  `http://www.server.cz/authorize/<secret>/<clientName>`
- **Vstup do fronty** - tuto adresu kontaktuje klient v případě, kdy zákazník chce vstoupit do fronty
  `http://www.server.cz/enter/queue/<clientId>`
- **Dotazovací adresa** - jediná univerzální adresa v testovacím protokolu. Je využívána ve chvílích kdy se klient snaží zjistit informace o nějakém objektu. Např.: Poskytovateli, frontách nebo aktuálním stavu fronty.
  `http://www.server.cz/get/<object>/<parameter>/<clientId>`

[4]Representational state transfer

4.3.2 Formát odpovědi serveru

Poté co je server kontaktován klientem, zkontroluje zda je autorizován podle jeho `clientID`, které vždy posílá. Pokud ano zobrazí klientovi odpověď. Pro zjednodušení komunikace je třeba zvolit správný formát dat a odpovědí. V tomto směru testovací protokol částečně odpovídá tomu oficiálnímu a formátuje data za pomoci jazyka XML[5].

Odpovědní zpráva je vždy složena s hlavního objektu `wrserver` který je reprezentován párovým tagem stejného jména. Veškeré informace, které server odesílá, jsou poté vždy obsaženy v tomto objektu. Dalším prvkem, který je vždy v odpovědi obsažen je objekt „code“, který obsahuje číselný návratový kód informující o tom, zda akce proběhla v pořádku nebo nastala chyba. Je zde použito stejné číselné označení chybových stavů jako při HTTP požadavku.

Autorizace klienta

Nejdůležitější zprávou v celé komunikaci je zpráva, kterou klient obdrží jako první. Jedná se o zprávu, kde server uvádí zda je klient autorizovaný a zasílá identifikační kód, kterým se poté klient serveru prokazuje. O autorizaci žádá klient jen jednou a to při startu celé aplikace.

```
<wrserver>
    <code>200</code>
    <clientId>785ed9a27d</clientId>
</wrserever>
```

Odpověď serveru při autorizaci klienta

Výpis objektů

Při komunikaci se serverem se nejčastěji bude objevovat odpověď, která bude popisovat a podávat info o nějakém objektu či objektech. Například výpis poskytovatelů na daném serveru Virtuální čekárny.

Odpověď opět obsahuje základní objekt „wrserver“ a objekt „code“ s informací zda proběhlo vše v pořádku. Nejdůležitějším prvkem odpovědi je kolekce, tvořená podle konvencí XML, obsahující objekty požadované klientem. Kolekce ve svém úvodním tagu obsahuje atribut „count“ udávající počet objektů, které server posílá klientovi. Příkladem této odpovědi je vrácení front na serveru nebo jejich aktuální stav.

[5]Extensible Markup Language

```
<wrserver>
    <code>200</code>
    <queues count="2">
        <queue>
            <id>1</id>
            <serviceId>1</serviceId>
            <name>Fronta 1</name>
        </queue>
    </queues>
</wrserver>
```

Výpis front na serveru

```
<wrserver>
    <code>200</code>
    <orders count="3">
        <order>
            <id>12</id>
            <orderCode>1</orderCode>
        </order>
        <order>
            <id>14</id>
            <orderCode>2</orderCode>
        </order>
    </orders>
</wrserver>
```

Výpis aktuálního stavu fronty

Kapitola 5

Implementace

Celá aplikace byla navržena a vytvořena pro operační systém Android. Pro Implementaci byl použil jazyk JAVA s knihovnami specifickým pro vývoj aplikací pro systém Android.

Při vývoji aplikace jsem se snažil aby byla byly zdrojové kódy psány tak, aby se co nejvíce podobaly MVC[1] modelu pro vývoj softwaru. V jazyku Java však nejsem příliš obeznámen s pravidly, které se používají pro tento typ vývoje a čerpal jsem ze svých znalostí MVC modelu v programovacím jazyce PHP.

5.1 Komunikace se serverem

Jedním z nejdůležitějších prvků aplikace je komunikace klienta se serverem. O tuto funkci se stará třída `NetworkManager`. Jedná se o třídu děděnou od třídy `AsyncTask`. Díky této dědičnosti běží spojení mimo hlavní běh aplikace, tudíž při čekání na odpověď od severu nijak neblokuje aplikaci a zákazník s ní tak může dále pracovat.

Klient komunikuje se serverem pomocí REST API na HTTP protokolu. Tedy snaží se přistoupit na webovou adresu, splňující pravidla aplikačního protokolu. Jelikož se adresy mění v závislosti na tom, co od serveru očekáváme, byla třída NetworkManager tomu přizpůsobena. A to tak, že byla vytvořena univerzální metoda, která přijímá webovou adresu jako parametr. Poté tuto adresu kontaktuje a zachytí odpověď od serveru v podobě `InputStreamu`. Třída NetworkManager se nestará o samotné zpracování dat, ale zachycená data předá dále ke zpracování jiné třídě.

Jakmile jsou data zpracována, předá NetworkManager data tomu objektu, který o ně žádal (dále budu tento objekt nazývat Žadatel). Díky tomu že NetworkManager běží asynchronně oproti celé aplikaci nelze vrátit data přesně na to místo v kódu kde o ně bylo požádáno. A je tedy nutno informovat Žadatele o tom že byla data připravena. Třídy děděné od třídy AsyncTask, tak jako NetworkManager, vždy při dokončení volají vlastní funkci „onPostExecute()“. Ta sice nedokáže v základu předat data tam kam potřebujeme ale dá se k tomu využít.

[1] Model View Controler

Aby byl NetworkManager schopen předat data do správného objektu je potřeba vytvořit rozhraní. V aplikaci je nazváno „AsyncResponse“ viz obrázek 5.1. Toto rozhraní obsahuje jednu funkci „processFinish()“. Poté lze v Žadateli vytvořit objekt, založený na tomto rozhraní, který přepíše chování metody processFinish() o volání nějaké metody z Žadatele (obr. 5.2). Takto vytvořený nový objekt je pak předán NetworkManageru přes jeho konstruktor a právě výše zmíněná funkce onPostExecute() se postará o zavolání funkce ProcessFinish() v něm obsažené. Díky tomu se nám dostanou data až k Žadateli (obr. 5.3).

```
1  package cz.radimkrek.waitingroomclient;
2
3  import java.util.HashMap;
4
5
6  /**
7   * Created by radim on 14.02.2016.
8   */
9  public interface AsyncResponse {
10     void processFinish(HashMap<String,HashMap<String,String>> output);
11 }
```

Obrázek 5.1: Rohraní pro zpracování dat

```
41          NetworkManager networkManager = new NetworkManager(
42                  new AsyncResponse() {
43                      @Override
44                      public void processFinish(HashMap output) {
45                          saveConnection(output);
46                      }
47                  }
48          );
```

Obrázek 5.2: Vytvoření objektu a předání do NetworkManageru

```
23 public class NetworkManager extends AsyncTask<String, Void, HashMap> {
24
25     public AsyncResponse delegate = null;
26
27     public NetworkManager(AsyncResponse delegate) {
28         this.delegate = delegate;
29     }
30
31     @Override
32     protected void onPostExecute(HashMap result) {
33         delegate.processFinish(result);
34     }
```

Obrázek 5.3: Přehled NetworkManageru

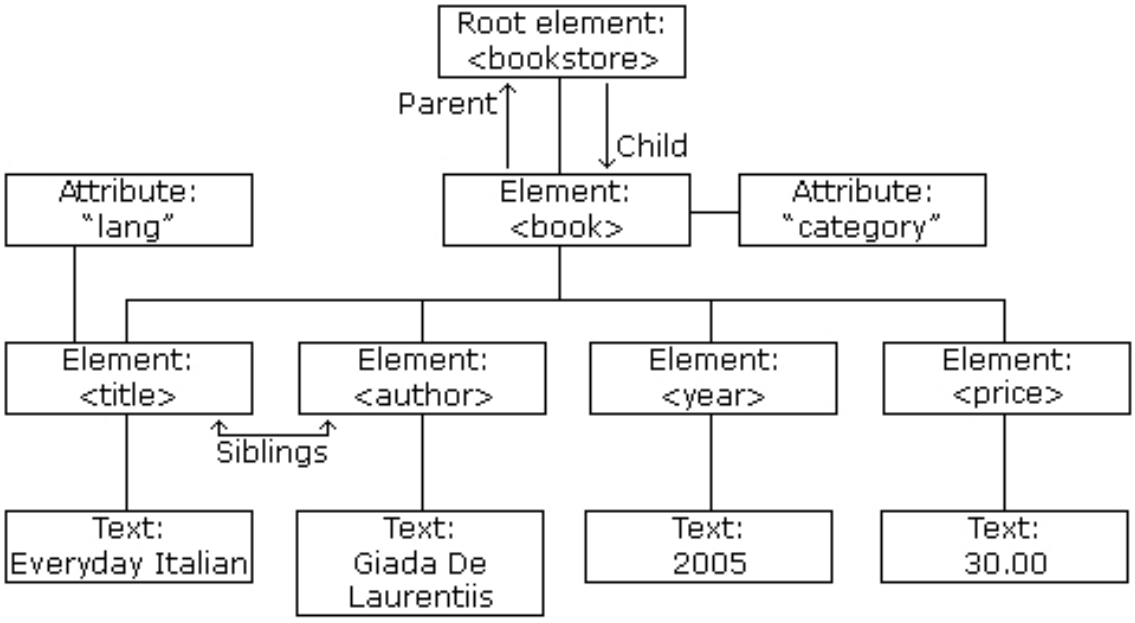
5.2 Parsování XML odpovědi

Veškeré odpovědi, které klient dostává od serveru jsou standardizované podle komunikačního protokolu a jsou posílány ve formátu XML[2].

V Javě je možno zpracovávat XML dvěma způsoby. Proudovým zpracováním a pomocí stromové struktury.

5.2.1 Zpracování stromovou strukturou

Z načtených dat je vytvořena stromová struktura objektů, ve které jsou elementy původního dokumentu reprezentovány objekty. V takovéto struktuře je pak velmi jednoduché se pohybovat, přidávat nebo odebírat uzly, popřípadě s ní manipulovat jakkoliv jinak. V dnešní době naprostá většina programovacích jazyků podporuje strukturu DOM[3] (obr. 5.4) vyvinutý konsorciem W3C[4]. Nevýhodou Stromové reprezentace je velmi velká paměťová náročnost a delší doba načítání dokumentu.



Obrázek 5.4: Stromová struktura DOM
Zdroj: `http://www.w3schools.com/`

[2]Extensible Markup Language
[3]Document Object Model
[4]https://www.w3.org/

5.2.2 Proudové zpracování

Proudové zpracování oproti Stromovému na začátku nenačítá celý dokument, ale prochází jej postupně. Přitom jakmile narazí na jednotlivé prvky XML volá určené metody, které obsahují námi implementovaný kód a starají se o zpracování informací. Oproti stromovému zpracování má také výhodu v paměťové nenáročnosti. Avšak většinu funkcí je potřeba implementovat a také tím že je soubor zpracováván sekvenčně, nelze se v dokumentu vracet.

5.2.3 Implementace Parseru

Při implementaci Klienta bylo zvoleno Proudové zpracování XML souboru, kvůli jednoduchosti implementace.

Logika starající se o parsování XML odpovědi serveru, je obsažena ve třídě „Parser“. Ten přebírá vstupní data v podobě InputStreamu od NetworkManageru a pomocí vestavěné knihovny systému Android, „XmlPullParser“ začne vstupní dokument parsovat. Výsledkem tohoto procesu je „HashMap“, datová struktura velmi podobná poli, ve které jsou uloženy veškeré informace, které byly v XML dokumentu obsaženy. HashMap je poté vrácena NetworkManageru, který se postará o doručení dat tam, kde jsou potřeba. Většinou do některé z funkcí třídy SystemManageru, který data nachystá a zašle k zobrazení.

5.3 SystemManager

Třída `SystemManager` zabírá v rámci celé aplikace jednu z nejdůležitějších pozic. Třída by se dala považovat ze centrum veškerých úkony, které klient provádí. Jedná se o dělící vrstvu, která odděluje část aplikace starající se o grafické rozhraní a část starající se o komunikaci se serverem, kterou reprezentuje `NetworkManager`.

Tato třída obsahuje většinu logiky aplikace a to i plnění dat do šablony. Proto třída obsahuje vždy instanci aktuálně používané Aktivity. Nejčastěji volanou funkcionalitou z této třídy je dvojice funkcí `loadQueueStatus()` a `showQueueStatus()`, kde první se stará o předání správných parametrů pro zavolání API k načtení stavu fronty. Druhá funkce přebírá získané data a převádí je do formátu ve kterém je lze zobrazit v Aktivitě. Funkce `showQueueStatus()` se stará i o samotné zaslání dát do aktivity.

5.4 Ukládání konfiguračních dat

Jelikož aplikace pracuje na principu Klient-Server, je potřeba ji na začátku nakonfigurovat. Je však velmi vhodné aby byli tyto informace o konfiguraci uloženy někde, kde zůstanou i v případě vypnutí aplikace, abychom ji nemuseli konfigurovat znovu. V androidu nalezneme několik způsobů jak toho docílit.

5.4.1 Metody ukládání dat v Androidu

Jednou z možností je uložení informací do souboru ze kterého si je poté vždy přečteme. Nevýhodou takovéhoto ukládání informací, je jejich složitá správa. A to zvláště v případě že by si aplikace ukládala velké množství dat.

Mnohem vhodnější variantou uložení dat je uložení do lokální databáze. Takováto data jsou logicky rozdělená a jejich údržba je velmi jednoduchá. Ovšem tento způsob se hodí pouze v případě že je potřeba ukládat větší množství datových informací.

Další možností je využití tzv. `SharedPreferences`. Jedná se o API poskytnuté v knihovnách android a umožňuje nám ukládat menší množství dat. Těmto hodnotám můžeme nastavit režim přístupu na sdílená data, nebo privátní. Kromě `SharedPreferences` existují také jen `Preferences`. Rozdíl mezi nimi je pouze v tom, že `Preferences` se vztahují k aktuální Aktivitě a `SharedPreferences` jsou stejné napříč celou aplikací.

5.4.2 Použití v aplikaci

V aplikaci je k ukládání využito `SharedPreferences`. Jsou do nich ukládána konfigurační data potřebná pro běh aplikace. Např. IP adresa serveru nebo hlavní heslo aplikace, které umožňuje aplikaci rekonfigurovat.

Přístupový kanál k `SharedPreferences` je vždy otevřen ve funkci `onStart()` každé aktivity, aby bylo možné s úložištěm kdykoliv pracovat. V hlavní aktivitě se navíc provádí kontrola, zda byla aplikace nakonfigurována či nikoliv. Pokud nakonfigurována nebyla, je uživatel automaticky přesměrován na obrazovku Nastavení a není mu dovoleno odejít, dokud aplikaci nenakonfiguruje tak aby byla schopna komunikace se serverem.

5.5 Grafické rozhraní

Celá aplikace je velmi jednoduchá a obsahuje pouze dvě obrazovky, které korespondují k částem, které byly uvedeny v kapitole 4.1.

Každá obrazovka je složena ze dvou komponent. XML souboru, ve kterém je navrženo jaké prvky se na obrazovce budou nacházet a kde budou umístěny. Každý prvek na obrazovce je v textové podobě reprezentován XML elementem. V tomto souboru lze také nastavit Výchozí hodnoty jednotlivých prvků. V případě že musí být komponenta dynamická je nutno vytvořit instanci prvku v Aktivitě. Ta je druhou komponentu, která tvoří výslednou obrazovku. Obsahuje funkční logiku a umožňuje ze statických prvků obrazovky vytvořit dynamické.

Jelikož je aplikace vyvíjena přímo pro provozovnu provozovatele, je pravděpodobné že velké procento uživatelů nebude technicky zdatných. Proto bylo rozhraní vytvořeno s ohledem na tento fakt co nejpřehlednější a nejjednodušší.

5.5.1 Hlavní obrazovka

Nejdůležitějším prvkem pro uživatele je přehled aktuální fronty. Proto na hlavní obrazovce zabírá největší prostor. Pro vizualizaci tohoto prvku byl zvolen výpis v podobě tabulky o jednom sloupci, pro její velkou přehlednost. Na zdrojové úrovni je výpis vytvořen pomocí standardního grafického prvku systému Android `ListView`. Výpis je třeba udržovat aktualizovaný a proto je při spuštění Aktivity vytvořen `Timer`, který každých 10 sekund volá funkci ze SystemManageru, která se stará o načtení aktuálních dat o všech frontách ze serveru.

Pro zařazení uživatele do fronty slouží dvojice prvku. Výběrový box tvořený komponentou `Spinneru`, který umožňuje výběr fronty. A k samotnému vstupu do fronty slouží tlačítko na které je navázán tzv. `onCLickListener`. Ten je volán při každém kliknutí na tlačítko a obsahuje logiku pro vstup do fronty a kód starající se o zobrazení dialogového okna pro zobrazení identifikátoru, pod kterým je uživatel veden ve frontě.

Při změně výběru ve výše zmíněné komponentě `Spinneru` dojde také ke změně aktivní fronty. Tím se tabulka s výpisem aktuálního stavu fronty překreslí tak, aby odpovídala právě vybrané frontě.

na obrazovce se ještě nachází tlačítko pro správce aplikace, které vyvolá dialog s vyžádáním o zadání hesla. Pokud je heslo zadáno správně aplikace přejde na obrazovku nastavení.

Výslednou obrazovku lze vidět na obrázku 5.5.

Obrázek 5.5: Hlavní obrazovka aplikace

5.5.2 Obrazovka nastavení

Obrazovka nastavení je méně graficky různorodá než hlavní obrazovka. Většina práce je schována ve funkční oblasti, kdy se volají obslužné rutiny kvůli připojení k serveru.

Obrazovka tak obsahuje pouze tři textové vstupní pole, pro zadání konfiguračních dat aplikace. Všechny jsou řešeny pomocí Android prvku `EditText`, s rozdílně nastaveným typem vstupu. Tím je docíleno to, že uživateli je vždy zobrazena klávesnice, která nejlépe vyhovuje povaze zadávané informace. Díky tomu se také zvětší bezpečnost aplikace, protože při zadávání hlavního hesla, jej není vidět, ale jsou zobrazovány pouze zástupné znaky. Rozložení obrazovky lze vidět na obrázku 5.6.

Obrázek 5.6: Obrazovka nastavení

Kapitola 6

Závěr

Cílem této práce bylo navrhnout a implementovat aplikaci pro systém Android, která by vystupovala jako klient, pro systém Virtuální čekárny, umístěný přímo v čekárně poskytovatele. A umožnil tak zákazníkovi se zařadit do fronty. Toho bylo docíleno a vznikla aplikace pro tablet se systémem Android verze 4.4 a vyšší. Ta společně se serverem, který byl také implementován, dokáže splnit všechny výše zmíněn požadavky.

6.1 Přínos práce

Největší přínosem práce bylo seznámení se s vývojem aplikací pro platformu Android, jakož to nejvíce rozšířeným mobilním systémem. Velmi poučná byla také práce se samotným Android SDK. Také jsem získal nové poznatky při návrhu Klient-server aplikace a její následné implementace.

6.2 Možnosti rozšíření

Aplikace má velkou škálu možností pro rozšíření. Nejdůležitějším rozšířením aplikace je propojení s ostatními oficiálními komponentami systému virtuální čekárny. Další možností pro rozšíření je implementace ověření, přihlášení, uživatele a umožnit mu např. objednání na příští termín návštěvy provozovatele.

Jedno ze zajímavějších rozšíření které by zlepšilo možnosti klienta je připojit jej k tiskárně a umožnit tak vytištění lístku s pořadovým číslem zákazníka, které si teď musí pamatovat.

Literatura

[1] Eva Sovová: Vymyslel, aby pacienti nečekali u lékařů zbytečně. Nápad mu mění život [online]. http://idnes.cz/JjliJ, 2015-04-15 [cit. 2016-04-12].

[2] Google, Inc.: Android Developers [online]. http://developer.android.com/about/versions/android-4.4.html, [cit. 2016-04-12].

[3] Google, Inc.: Virtual Hold Technology [online]. http://www.virtualhold.com/products/vht-callback/, [cit. 2016-04-27].

[4] Kadlec Elektro: Vyvolávací systémy [online]. http://www.kadlecelektro.cz/produkty/vyvolavaci-systemy/, [cit. 2016-05-09].

[5] Medingo s.r.o.: Systém Evipa [online]. http://www.medingo.cz/evipa.html, [cit. 2016-05-09].

[6] Michal Mareš: Neklepejte, sestra vychází! Jak český (ne)startup řeší chaos v čekárně u doktora [online]. http://www.forbes.cz/neklepejte-sestra-vychazi-jak-cesky-nestartup-resi-chaos-v-cekarne-u-doktora/, 2016-02-08 [cit. 2016-05-01].

[7] Techtarget Network: REST (representational state transfer) [online]. http://searchsoa.techtarget.com/definition/REST, [cit. 2016-04-30].

[8] TutorialsPoint: Android Architecture [online]. http://www.tutorialspoint.com/android/android_architecture.htm, [cit. 2016-03-01].

[9] W3C: Document Object model [online]. https://www.w3.org/DOM/, [cit. 2016-05-01].

[10] Wikipedia: Android (operační systém) [online]. https://cs.wikipedia.org/wiki/Android_(operační_systém), 2016-01-21 [cit. 2016-03-01].

[11] Wikipedia: Representational state transfer [online]. https://en.wikipedia.org/wiki/Representational_state_transfer, [cit. 2016-04-30].

[12] Wikipedia: Document Object model [online].
`https://cs.wikipedia.org/wiki/Document_Object_Model`, [cit. 2016-05-01].

Příloha A

Obsah CD

Přiložené CD obsahuje:

- adresář se zdrojovými kódy serveru;
- návod pro instalaci serveru
- adresář se zdrojovými kódy aplikace;
- instalační soubor aplikace ve formátu APK pro systém Android;
- adresář se zdrojovými kódy této technické zprávy;
- tuto technickou zprávu v PDF;