Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Návrh a implementace aplikace pro evidenci ozáření RTG při vyšetření

Jakub Hamadej

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jakub Hamadej

Informatika

Název práce

Návrh a implementace aplikace pro evidenci ozáření RTG při vyšetření

Název anglicky

Design and implementation of an X-ray dosage registry system

Cíle práce

Bakalářská práce se zabývá návrhem a vývojem mobilní aplikace. Hlavním cílem je návrh a implementace aplikace pro archivaci informací o vyšetřeních s dávkou RTG záření. Dílčím cílem bude analyzovat stávající situaci v této oblasti.

Metodika

Metodika řešení práce je založena zejména na prototypovém přístupu. Dále bude provedena analýza funkčnosti aplikací, které používají QR kódy a aplikací, které na základě dat vytváří a archivují seznamy. Na základě zjištěných poznatků bude provedena identifikace slabých míst těchto aplikací. Tyto slabá místa budou ve vytvořené aplikaci posílena.

Aplikace bude implementována pro mobilní zařízení s OS Android v jazyce Java s pomocí nástroje Android Studio. Po dokončení bude aplikace otestována a bude demonstrován její provoz a komunikace s PC stanicí.

Výsledná aplikace bude zhodnocena, zkušenosti z jejího návrhu a implementace budou shrnuty a budou nastíněna případná další možná budoucí rozšíření aplikace.

Doporučený rozsah práce

35-40 stran

Klíčová slova

Android, Java, Android Studio, QR, Zdravotnictví, Mobilní

Doporučené zdroje informací

Android Developers [online]. [cit. 2016-04-01]. Dostupné z: http://developer.android.com/sdk/index.html

HEROUT, Pavel. Učebnice jazyka Java. 5. rozš. vyd. České Budějovice: Kopp nakladatelství, 2011. ISBN 978-80-7232-398-2.

LACKO, Ľ. Vývoj aplikací pro Android. Brno: Computer Press, 2015. ISBN 978-80-251-4347-6

Oracle [online]. [cit. 2016-20-01]. Dostupné z: http://www.oracle.com/technetwork/java/index.html

---

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 07. 02. 2017

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Návrh a implementace aplikace pro evidenci ozáření RTG při vyšetření" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 12. 3. 2017 Jakub Hamadej

Poděkování

Rád(a) bych touto cestou poděkoval(a) panu Ing. Jiřímu Brožkovi, Ph.D. za jeho ochotu vést mou práci a jeho neocenitelné rady v průběhu celého procesu.

Návrh a implementace aplikace pro evidenci ozáření RTG při vyšetření

Souhrn

Tato bakalářská práce se zabývá návrhem a vývojem aplikace pro mobilní zařízení s OS Android pro zaznamenání a archivaci vyšetření, při kterém byla přítomná dávka RTG záření. Zároveň je zde kladen důraz na splnění bezpečnosti přenosu dat při interakci mezi zařízeními, toto je ošetřeno použitím QR kódu jako média. Celá aplikace je psána v jazyce Java za pomoci vývojového nástroje Android Studio. Při vývoji je kladen důraz na využívání obecných principů tvoření uživatelsky přívětivého uživatelského rozhraní. Dále jsou použity obecné principy pro tvoření přehledného a čitelného kódu, případné komplikace jsou řešeny za pomoci návrhových vzorů, pokud možno.

Teoretická část popisuje různé části QR kódu, platformu Android a její architekturu. Jsou zde popsány dílčí části aplikace (aktivity, layout …) a jejich zakomponování do celku.

V praktické části autor popisuje krok po kroku postup tvoření této aplikace. Ať už se jedná o tvorbu objektů, definování metod, řešení problémů či zabránění potencionálním komplikacím. V celé této části jsou uvedeny konkrétní bloky kódu, které jsou doprovázeny vysvětlením jejich funkčnosti, popřípadě jejich úděl v konkrétním procesu.

Klíčová slova: Android, Android Studio, Java, Databáze, QR kód, Zdravotnictví, RTG záření, Aplikace, Mobilní aplikace

Design and implementation of an X-ray dosage registry system

Summary

This thesis describes the design and development of an application for mobile devices with Android operation system for recording and archiving examinations, where a radiation dose was present. At the same time, there is an emphasis on meeting the security of data transmission during the interaction between the devices, this is handled by using a QR code as a medium. The application itself is written in Java with the help of Android studio, a development tool. During the development, the emphasis is on the use of general principles of creating a user-friendly user interface. There are also used general principles for the formation of a clearly readable code. Problems are solved by using design patterns, where possible.

The theoretical part describes the different areas of a QR code, the Android platform and its architecture. There are described sub-sections of the application (Activities, layouts, …) and their integration to the whole.

The practical part describes the step-by-step procedure for creating this application. Whether it is about creating objects, defining methods, problem solving or avoiding potential complication. Throughout this section are specific code blocks that are accompanied by an explanation of their function or their role in a process.

Keywords: Android, Android Studio, Java, Database, QR code, Healthcare, Radiation, Application, Mobile application

Obsah

1 Úvod

Motivací a hlavní pohnutkou výběru tohoto tématu je současná situace nejen v českém zdravotnictví, co se týče archivace a sdílení záznamů o ozáření pacienta.

V této chvíli mezi sebou jednotlivá oddělení, polikliniky, nemocnice či jiná zdravotnická pracoviště tyto data nesdílí. Je zde kompletní absence jakéhokoliv centrálního archivu pro tyto účely. Tím pádem nikdo neví, jakému byl pacient vystaven záření za poslední dobu a není možno určit, jestli je bezpečné vůbec pacienta v tuto chvíli dalšímu záření vystavit.

Není pochyb o tom, že je pouze otázkou času, než se prokáže, jaké problémy pacientům tento přístup skutečně způsobuje. Již několik let se neoficiálně polemizuje o státem vydaném ekvivalentu občanské karty s čipem, který by s sebou byl občan povinen neustále nosit a která by sloužila právě k nahrávání těchto vyšetření.

Vzhledem k neustále se měnící politické situaci, vývoji technologií a změně České IT infrastruktury se nabízí několik možných řešení. Jako velmi atraktivní se jevila možnost použití tzv. chytrých telefonů, což jsou dnes už v podstatě malé počítače. Narazilo se ale hned na několik problémů, největší z nich byly legislativní: člověk nemající chytrý telefon přijde o možnost archivace a tím tedy o prevenci potencionálně zdraví škodlivého vyšetření. Není z hlediska státu možno takto občany dělit, stát musí, podobně jako u občanské karty, být schopen všem poskytnout stejné základní služby bez ohledu na jejich finanční situaci. Je nepřijatelné považovat prevenci zdraví škodlivého vyšetření jako tzv. prémiovou službu (např. typu odlehčená sádra) za příplatek.

Nicméně bez rentgenového záření se člověk ještě dlouhou dobu neobejde a je tedy vhodné se na tento problém podívat i z dlouhodobějšího hlediska. Ve světě, obzvláště pak v norských zemích či ve Francii se na základě Práva o přístupu k internetu [1] diskutuje o tom, že přístup k internetu, a tedy k informacím, by měl být považován za základní právo a stát by ho měl občanovi poskytovat. Před několika lety by tento výrok vypadal směšně, není tedy vyloučeno, že nás v příští dekádě nepotká stejná situace na téma chytré telefony.

Ačkoliv si nyní trochu protiřečím, s největší pravděpodobností se tato služba ve svém počátku stane komerčně prémiovou nebo bonusovou, ať už samostatně nebo jako součást jiného již používaného programu, kterých se v této sféře již několik vyskytuje. Až již postupem času bude stát nucen zareagovat.

Co se týče bezpečnosti IT v zdravotnictví, zde je situace nekompromisní. Žádná data, obzvláště pak ta o pacientech, nesmí být kompromitována jakýmkoliv způsobem. Tím pádem jsou zde zavedeny i velmi přísné instrukce, kdo a jak smí s počítačovou stanicí zacházet, popřípadě kdo smí vstoupit do systému s jakým oprávněním. Těchto opatření existuje celá řada.

Pro vytvoření této aplikace jsem zvažoval několik způsobů přenosu dat, mezi nejlepší kandidáty patřily metody přenosu po USB, Wi-Fi, Bluetooth nebo NFC (Near field communication). Žádná z těchto metod však nesplňuje hned několik bezpečnostních zásad a vystavuje nemocniční zařízení riziku infekce nežádoucím kódem. Uchýlil jsem se k méně tradiční metodě a tou je QR kód. Ten je sice hrubě omezen maximální možnou velikostí přenosu dat, ale ve všech bodech splňuje tyto přísné restrikce. Při zvolení tohoto způsobu totiž nedojde k absolutně žádné interakci mezi dvěma zařízeními.

2 Cíl práce a metodika

2.1 Cíl práce

Tato bakalářská práce se zabývá návrhem a zejména vývojem mobilní aplikace. Hlavním cílem je návrh a implementace aplikace pro archivaci informací o vyšetřeních s dávkou RTG záření. Dílčí cíle jsou následující:

- Nastudování jazyka Java a nástroje Android Studio
- Najít vhodnou metodu archivace dat, tedy příslušnou databázi
- Zajistit čtení a vhodnou interpretaci QR kódu
- Analýza již vytvořeních aplikací
- Zhodnocení reálného využití aplikace

2.2 Metodika

Metodika řešení práce je založena zejména na prototypovém přístupu. Dále bude provedena analýza funkčnosti aplikací, které používají QR kódy a aplikací, které na základě dat vytváří a archivují seznamy. Na základě zjištěných poznatků bude provedena identifikace slabých míst těchto aplikací. Tato slabá místa budou ve vytvořené aplikaci odstraněna.

Aplikace bude implementována pro mobilní zařízení s OS Android v jazyce Java pomocí nástroje Android Studio. Po dokončení bude aplikace otestována, zhodnocena a bude demonstrován její provoz v reálném čase. Nejprve však bude proveden průzkum jazyka Java, nástroje Android studio a jejich možností.

Výsledná aplikace bude zhodnocena, zkušenosti z jejího návrhu a implementace budou shrnuty a budou nastíněny případné možnost jejího dalšího rozvoje.

3 Teoretická východiska

3.1 QR kódy

3.1.1 Historie

Píše se rok 1960, Japonsko právě zažívá věk prudkého hospodářského růstu. Na každém rohu je možno spatřit širokou škálu komodit, přes obchody s jídlem až k prodejnám s oblečením. V této době se standardně používaly pokladny, které se musely manuálně obsluhovat a veškerý vstup byl uživatel nucen zadat. Toto s sebou neslo velikou časovou náročnost, což vedlo k vývoji a používání čárového kódu [2], který vídáme dodnes.

Časem a postupným užíváním však vyšlo najevo, že tato technologie má své limity a nejsou velké, kód je schopen obsahovat informace v hodnotě pouze 20 alfanumerických znaků, což stačilo na zadání názvu a ceny zboží. Postupným rozvojem průmyslu ale bylo velmi žádoucí zadat těchto informací více, a tak byl vyvíjen tlak pro vyvinutí kódu, který by byl schopen pojmout podstatně větší množství informací [3]. Toho se ujala japonská firma DENSO WAVE (INCORPORATED) [4], která roku 1994 ohlásila vznik a používání QR kódu 1.

Postupně se tento kód začal využívat v automobilovém průmyslu a hlavně logistice, kde dodnes slouží jako nenahraditelný nástroj. Však největším krokem, který stojí za dnešní popularitou tohoto kódu je fakt, že tento typ kódu byl krátce po svém uveřejnění poskytnut zdarma komukoliv k používání. Tento krok se začal projevovat až v roce 2002, při výskytu prvních chytrých telefonů a od té doby je popularita kódu na vzestupu [3].

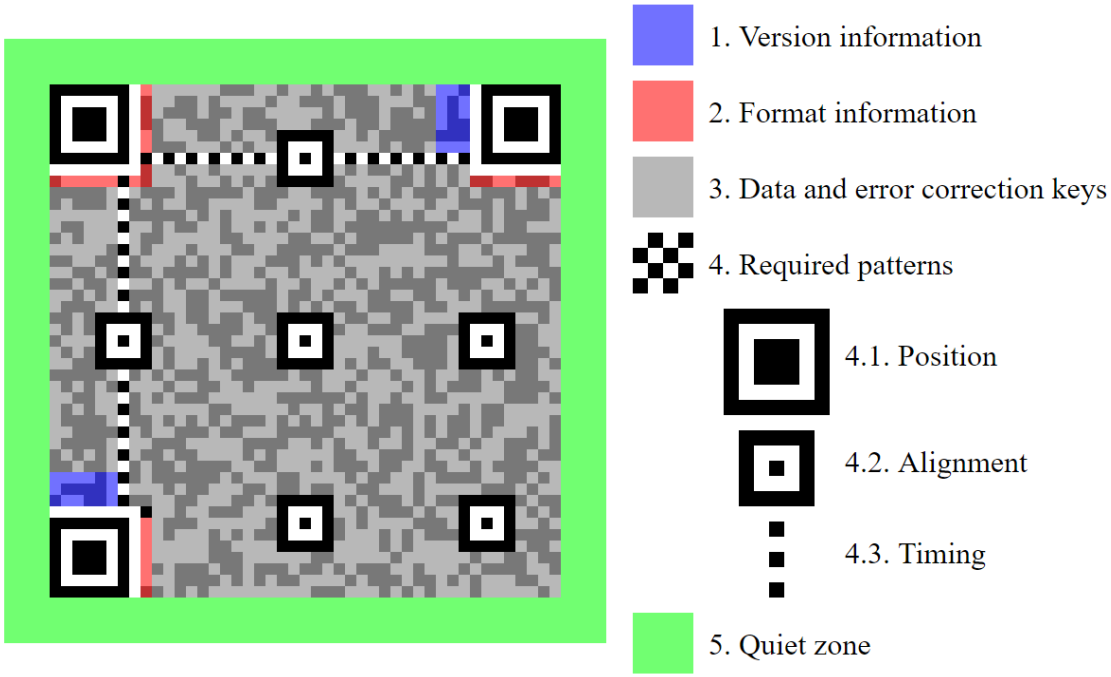
Za zmínku rozhodně stojí schválení kódu jako: AIM standard v roce 1997, standardní 2D kód pro japonský průmysl v roce 1999 a mezinárodní standard ISO v roce 2000. Dnes je standardní do takové míry, že tvrdit jeho používání po celém světě není žádná nadsázka [3].

[1] Quick response – rychlá odezva

3.1.2 Jak funguje?

Na rozdíl od běžného čárového kódu, který ukládá informace pouze v jedné dimenzi – rozměru, je tento kód dvourozměrný. Tento jednoduchý fakt mu umožňuje ukládat několikrát až několiksetkrát větší množství informací (záleží na rozměrech matice). Samotný kód se skládá z několika částí, každá má svou roli.

Obrázek 1 - rozložení QR kódu
Zdroj: Data Genetics Dostupné z: http://datagenetics.com/blog/november12013/anat2.png



1) Informace o verzi
2) Informace o formátu
3) Klíče k datům a opravě chyb
4) Mandatorní vzory
   a) Pozice
   b) Zarovnání
   c) Načasování
5) Tichá zóna

Kolik informací je kód schopen obsáhnout závisí na jeho verzi a tím úměrně i na jeho rozměrech, maximálně však 3Kb. Kód verze 1 je matice o rozměrech 21 x 21 modulů. Kód verze 40 je matice o rozměrech 177 x 177 modulů. Pro postup z dané verze na verzi o jeden stupeň vyšší se přidají 4 moduly v obou rozměrech [5].

Počet chyb způsobených při čtení (záměna černé za bílou, např. daná špatným osvětlením, poškození dokumentu vodou …), ze kterých se můžeme zotavit, je dán úrovní opravování chyb při vytváření kódu a je na uživateli, aby ji specifikoval. Čím větší úroveň

zvolí, čím méně dat se do kódu vejde, je tedy potřeba zvážit, jak se bude daný kód používat. Například jedná-li se o kód na vizitku, kde je nebezpečí poničení kódu poměrně vysoké (namočení, přeložení, pomuchlání …), je vhodné použít úroveň opravy chyb co nejvyšší. Pokud se jedná o kód, který bude zobrazován pouze v elektronické podobě, vystačíme si se základní úrovní – nebo žádnou. Celkem existují 4 úrovně [6]:

1) Úroveň L: Zhruba 7 % nebo méně chyb může být opraveno
2) Úroveň M: Zhruba 15 % nebo méně chyb může být opraveno
3) Úroveň Q: Zhruba 25 % nebo méně chyb může být opraveno
4) Úroveň H: Zhruba 30 % nebo méně chyb může být opraveno

Obrázek 2 - různé verze QR kódu Zdroj: MeetHeed Dostupné z: http://qrcode.meetheed.com/images/qrcode_lots.png

Směrem zleva doprava a od shora dolů jsou obrázky následující: Velké množství dat, malé množství dat, střední množství dat a malé množství dat včetně opravy chyb úrovně H.

Velká výhoda a dá se říct i důvod dnešní popularity QR kódu tkví v tom, že se nečte opticky odrazem světla jako čárový kód, ale digitálně pomocí identifikace klíčových modulů. Tato skutečnost umožňuje čtení QR kódů na dnešních chytrých telefonech.

3.1.3 Výskyt a obsah

V dnešní době není možné projet se linkou MHD, prolistovat časopis anebo se jen tak projít po ulici, aniž byste na každém kroku nenarazili na QR kód v jedné z jeho forem. Může se jednat o miniaturní QR kód na vizitce, normální kód na plakátu na ulici anebo velký kód na dokumentu.

Obrázek 3 - příklad umístění QR kódu na lahvi vína
Zdroj: QR code press
Dostupné z: http://www.qrcodepress.com/wp-content/uploads/2012/01/QR-Codes-on-Wine-266x250.jpg

Obrázek 4 - příklad umístění QR kódu na vizitce
Zdroj: 708 media
Dostupné z: http://www.708media.com/wp-content/uploads/2011/06/qr-communiceert-transparent-qr-code-business-card.jpg

Obrázek 5 - příklad umístění QR kódu na inzerční ploše
Zdroj: QR code tracking
Dostupné z: http://qrcodetracking.net/wp-content/uploads/2013/01/sunexpress-qr-code-scavenger-hunt-poster-300x300.jpg

Využití je obrovské, umístěn může být téměř kamkoliv a informace, které je schopen obsáhnout, jsou bezkonkurenční. Typově se řadí takto [7]:

1) Kontaktní: Data jsou uložena v kontaktní kartě, která může obsahovat jméno, telefonní číslo, emailovou adresu a podobně.
2) Kalendářní: Vytvoření kalendářní události
3) URL: Jednoduchá forma odkazu, která je po skenování otevřena v prohlížeči
4) Email: Otevře se nový email s daným adresátem
5) Telefonní číslo: Vytočí číslo
6) Jednoduchý text: Bezúčelný text, který nemá přiřazenou žádnou akci, ale je možné jej specificky použít v rámci jiné aplikace.
7) Geo. poloha: Odkaz na lokaci, možnost otevření mapy

Výše uvedené typy nejsou definitivní! Jedná se o předdefinované možnosti, které jsou dány nastavením operačního systému přístroje, tedy jeho schopností zpracovat data na vstupu v dané formě. Nejdůležitější je zde bod č. 6, tedy zpracování surových dat programátorem definovanou cestou.

3.2 Android

3.2.1 Obecně

„Android je open-source platforma na bázi Linuxu určená hlavně pro mobilní zařízení, tedy chytré telefony, tablety, fotoaparáty a navigace. V současnosti se stále více přidávají i konvertibilní zařízení, tj. tablety s odpojitelnou klávesnicí, které se hodí i k méně náročnému podnikovému nasazení. Platforma Android se objevuje i v televizních přijímačích a různých dalších zařízeních.“ [8, s. 49]

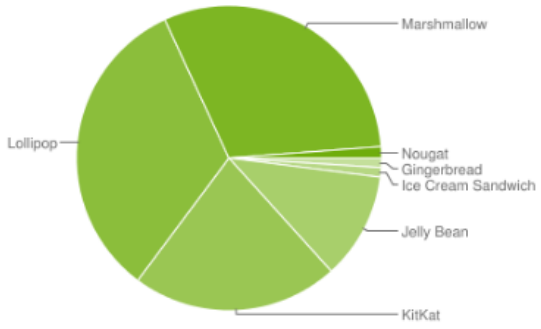
„Největší výhodou a zároveň nevýhodou platformy je její otevřenost a možnost úprav, ať už ze strany výrobců nebo uživatelů. Úpravy se netýkají jen konfigurace či widgetů, ale i firmwaru. Pro Android je k dispozici nejvíce aplikací, mnohé jsou však pochybné kvality, jelikož proces jejich schvalování není tak přísný jako u iOS či Windows. Tablety a telefony s Androidem dodává hodně firem. Je to záruka dynamičtějšího vývoje nových zařízení,

například v porovnání s Applem, kde je celý vývoj hardwaru v režii jedné firmy, a zároveň to představuje problém, protože aplikace běží na přístrojích s různým rozlišením displeje a různým výkonem procesoru a grafiky. V praxi to znamená různý komfort uživatelského rozhraní. Na rozdíl od ostatních platforem nevydává aktualizace operačního systému centrálně Google, ale výrobci zařízení, takže se u různých zařízení můžete setkat s různými verzemi.“ [8, s. 49]

Jednoduše řečeno se tedy jedná o rozšířený operační systém pro širokou škálu mobilních zařízení. Samozřejmě během své několikaleté historie prošel řadou menších i větších změn a objevilo se jej několik verzí. [8, s. 50-52] Tato práce se nezabývá historií Androidu a její pitvání by nepřineslo žádný přínos. Nicméně samotný fakt, že existuje několik verzí a velmi malé procento používá nejnovější verzi, je pro nás velmi důležitý. [9]

Ještě před vytvořením projektu se totiž musíme rozhodnout, jakou minimální verzi Androidu bude naše aplikace podporovat. Důvodů pro naše rozhodnutí může být celá řada, ale nejpodstatnějším faktorem by měl být počet uživatelů s danou verzí systému.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.0%
4.1.x	Jelly Bean	16	4.0%
4.2.x		17	5.7%
4.3		18	1.6%
4.4	KitKat	19	21.9%
5.0	Lollipop	21	9.8%
5.1		22	23.1%
6.0	Marshmallow	23	30.7%
7.0	Nougat	24	0.9%
7.1		25	0.3%



Obrázek 6: Procentuální počet uživatelů k jednotlivým verzím Androidu. Aktuální k 6. 2. 2017
Zdroj: Android Developers
Dostupné z: https://developer.android.com/about/dashboards/index.html

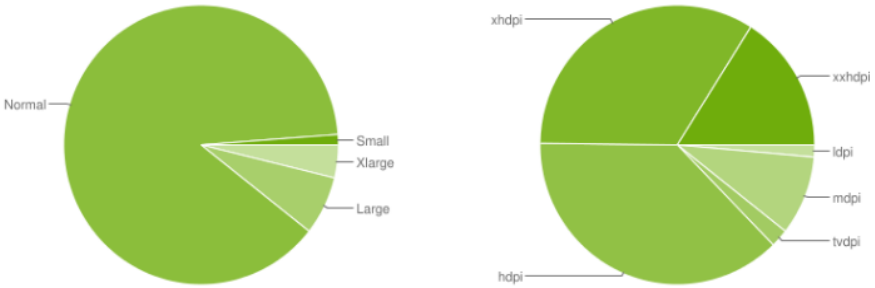
Různé verze podporují různé API a tím pádem i různé objekty. Je ale možné na starší verzi Androidu používat aplikace, které mají funkce či objekty přidané až ve verzi novější, pokud programátor vše správně nastaví. Tyto věci „navíc“ budou zabaleny přímo do aplikace a uživatel je bude tedy moci plně využívat, tzv. *backwards compatibility*. [10] Logicky se ale tím velikost aplikace zvětší a při nadměrném používání může dojít k velmi nežádoucím hodnotám. Programátor je tedy vystaven před svoje první dilema, ještě, než začne tvořit.

První možnost je aplikaci vytvořit pro nejnovější verzi systému a tím pádem bude mít plnou podporu všech funkcionalit. Na druhou stranu tím přijde o většinu uživatelů, protože ti nebudou mít možnost tuto aplikaci ani stáhnout. Druhou možností je vytvořit aplikaci pro nízkou verzi Androidu, tím pádem ji budou moci používat téměř všichni (vytvářet aplikaci pro verzi Androidu = tato verze bude minimální požadavek, vyšší verze ji jsou schopny spustit také). Opět je i zde druhá strana mince, tímto krokem si odepřeme přístup od veškerých nových výdobytků novějších verzí. Popřípadě je zde ještě již zmíněná třetí verze zakomponování novějších funkcionalit přímo do aplikace. [8, s. 56-58]

O trošku méně důležité, ale přece jenom se jedná o podstatný faktor, je rozlišení přístroje, který bude uživatel používat. Například na tabletu by aplikace navržená čistě pro mobilní telefony vypadala komicky. Opět se tedy musí programátor zamyslet, na jaká zařízení je aplikace cílená. Poté může pro každé žádoucí rozlišení vytvořit unikátní zacházení s ním, a to zejména pomocí fragmentů. [11] [12]

Jednoduchým částečným ošetřením tohoto problému je relative layout. [13] Ten při vytváření umožňuje objekty a vzdálenosti mezi nimi nastavit v relativních jednotkách, takže poměr vzdáleností zůstane na zařízení s jiným rozlišením stejný. Toto však není profesionální přístup ale absolutní minimum, které by aplikace měla obsahovat.

	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	1.2%						1.2%
Normal		2.6%	0.2%	36.6%	32.8%	16.0%	88.2%
Large	0.2%	3.9%	1.9%	0.4%	0.4%		6.8%
Xlarge		2.7%		0.5%	0.6%		3.8%
Total	1.4%	9.2%	2.1%	37.5%	33.8%	16.0%	

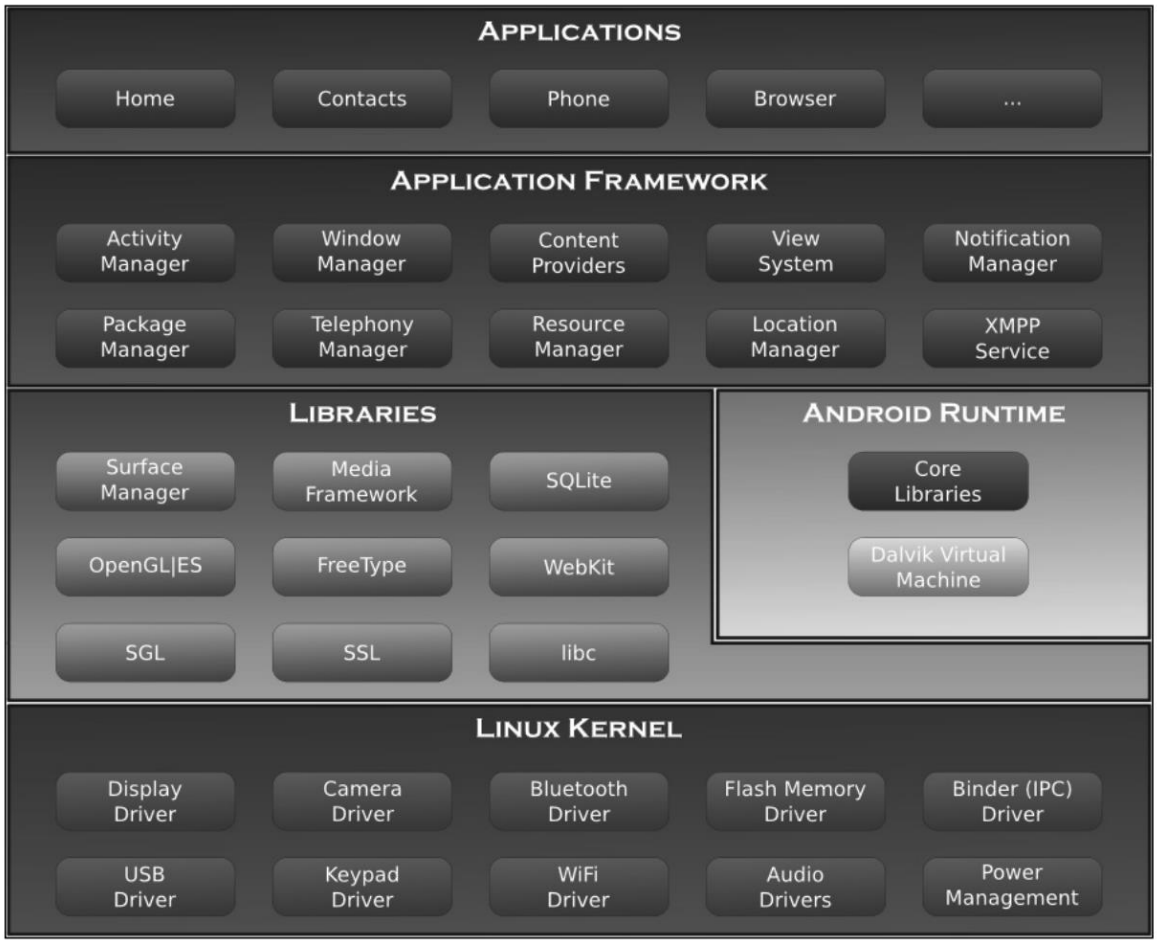


Obrázek 7: Různé rozlišení zařízení s operačním systémem Android. Aktuální k 6. 2. 2017
Zdroj: Android Developers
Dostupné z: https://developer.android.com/about/dashboards/index.html

3.2.2 Architektura

K vývoji je nutné alespoň základní porozumění o principech a architektuře operačního systému, na kterém aplikace poběží. Architektura Androidu je zde popsána podle schématu na obrázku na následující straně systémem „zdola nahoru“, tedy od nejnižší architektonické vrstvy. [8, s. 58]

Základním pilířem, nejnižší vrstvou architektury Androidu, je upravené jádro populárního operačního systému Linux. Úpravy se týkají redukce funkcí a jejich přizpůsobení možnostem mobilních zařízení. Vývojáři běžných aplikací s jádrem do přímého kontaktu však nepřijdou, ale mohou použít ADB (Android Debug Bridge) k příkazovému rozhraní Linux Shell. Skrz něj potom mohou zadávat jádru operačního systému příkazy. [8, s. 59]



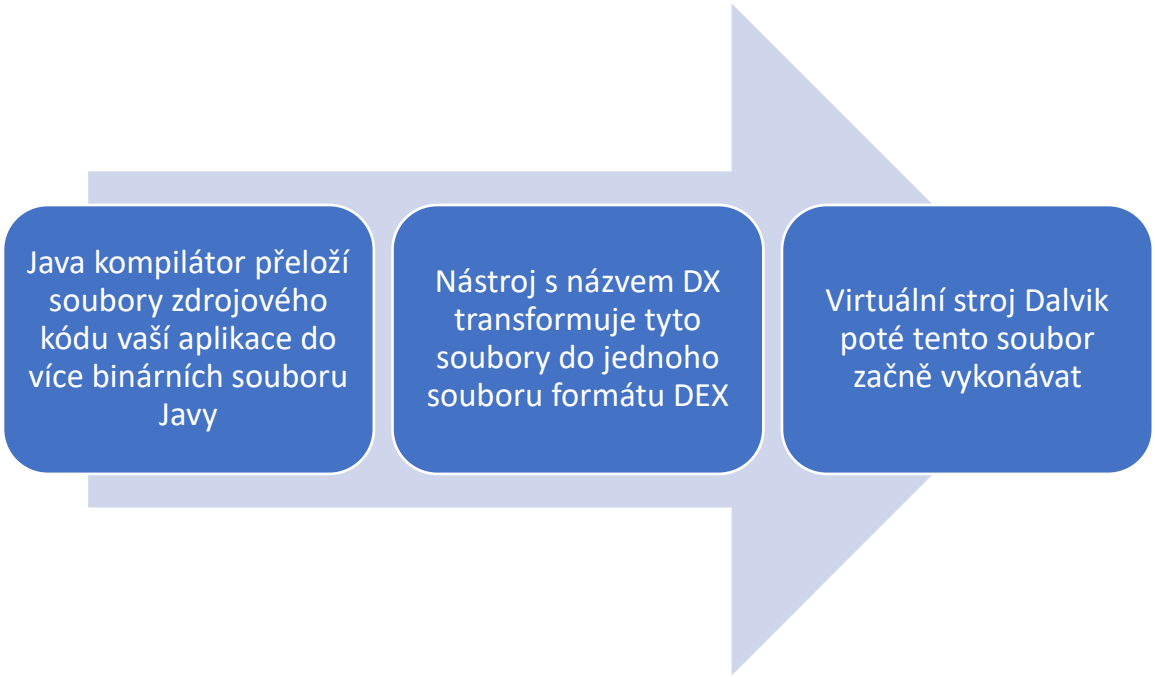
Obrázek 8: Schéma architektury OS Android
Zdroj: Odborná literatura. Dostupné z: [8, s. 58]

Nad jádrem je situovaná vrstva knihoven, které poskytují přímý přístup aplikací k různým komponentám systému Android. Jsou to nativní knihovny napsané v C/C++. Tvoří mezivrstvu mezi různými komponentami vyšších vrstev a linuxovým jádrem. [8, s. 60]

V neposlední řadě se je ještě obsažen Android Runtime, který obsahuje sadu základních knihoven. Každá aplikace pro Android je samostatný proces využívající vlastní instanci virtuálního stroje DVM (Dalvik Virtual Machine). Tento zabezpečuje běh spustitelných souborů s příponou DEX. Soubory DEX vznikly kompilací klasických souborů CLASS a JAR. Jsou kompaktnější než klasické soubory CLASS. Dalvik je optimalizovaný pro mobilní zařízení, to znamená, že bere v úvahu omezení možnosti napájení, menší paměť a podobně. Současně může běžet více instancí virtuálního stroje.

DVM na mobilních zařízeních se dá považovat za jakousi analogii Java Virtual Machine na klasických počítačích, na rozdíl od JVM jsou aplikace ale spouštěny ve svých procesech. [8, s. 60]

Co se tedy stane, když spustíte aplikaci napsanou v jazyku Java? [8, s. 60]



Obrázek 9: Jak probíhá kompilace na operačním systému Android.
Zdroj: Vlastní tvorba

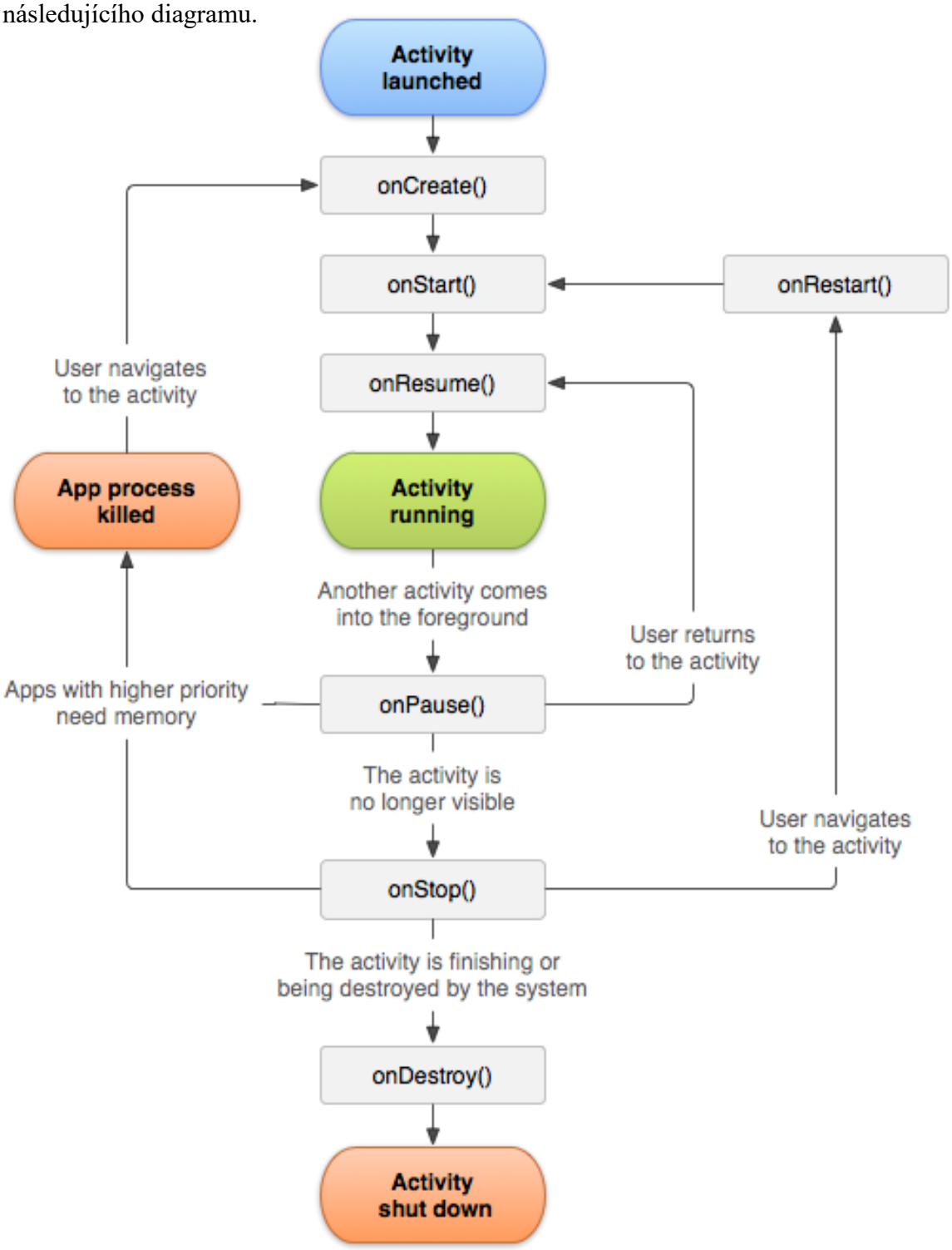
3.2.3 Aktivita a její životní cyklus

Aplikace pro Android se skládají z více na sobě nezávislých komponent – aktivit a služeb. Operační systém poté sám určuje, kdy budou instance aktivit vytvořeny, odsunuty či smazány. Aktivita reprezentuje jednu obrazovku s uživatelským rozhraním. V kódu Javy je aktivita objekt odvozený od třídy Activity (později *AppCompatActivity*). Většina aplikací – nejen pro mobilní zařízení – se ale obvykle skládá z několika těchto obrazovek. Na platformě Android se nazývají Activity. [8, s. 65]

Správně navržená aplikace je taková, kde jednotlivé aktivity fungují každá samostatně. To znamená, že mají přesně definované vstupy a výstupy. Pokud je všechno takto vymyšleno a posléze i propojeno, jeví se aplikace uživateli jako jeden kontinuální proces. Většinou aktivita zabírá celou plochu displeje, ale není to pravidlo. Co je ale podstatné: Každá aktivita má svůj vlastní životní cyklus a je potřeba počítat s tím, že instance aktivit či služeb mohou být v odůvodněných případech odstraněny. Například pokud začnou docházet systémové zdroje, systém se pokusí odstranit nepoužívané komponenty. [8, s. 66]

Jedna z aktivit je definována jako hlavní. Tato aktivita se zobrazí hned po spuštění aplikace. Po spuštění další aktivity se pozastaví, ale zůstane v paměťové oblasti nazývané *Back Stack*. Do této oblasti se aktivita ukládá po spuštění.

Samotný životní cyklus aktivity je definovaný metodami, které se spouštějí v přesně definovaných situacích v určeném pořadí. Životní cyklus lze nejlépe pochopit z následujícího diagramu.



Obrázek 10: Schéma životního cyklu aktivity
Zdroj: Android Developers
Dostupné z: https://developer.android.com/images/activity_lifecycle.png

3.2.4 **Intent (záměr)**

Aby se mohl uživatel pohybovat mezi jednotlivými aktivitami, je zapotřebí použít třídu Intent. Intent, přeloženo jako „záměr“ přesně odpovídá významu slova. Její využití spočívá pouze v záměru ovlivnit nějakým způsobem aktivitu nebo zjistit její stav. To může znamenat přesunutí se na jinou aktivitu, pozastavení aktivity, zkontrolovaní služby … nejedná se však o statickou třídu, je tedy potřeba vytvořit si její instanci. Syntaxe vypadá následovně [14] :

```
Intent intent = new Intent(context (source), target);
startActivity(intent);
```

Obecně je tedy vytvořen intent, který spojuje místo A s místem B. Samozřejmě je třída Intent o něco obsáhlejší a poskytuje rozmanitou konfiguraci. Konfigurace umožňuje například otevření aktivity ve specifickém režimu, provedení akce, přidání parametrů … [14]

Nejpodstatnější je funkce **putExtra**. Tato metoda slouží k předávání hodnot mezi komponentami. Je možné odevzdat jakýkoliv objekt *Serializable* nebo *Parcelable*. Objekty se mapují pomocí textových řetězců. [8, s. 70]

```
intent.putExtra(object);
```

Není tedy nutné vytvářet redundantní veřejně přístupné proměnné k přenosu dat. Načtení dat v následující aktivitě proběhne jednoduše statickou funkcí **getIntent**. Tím je načten intent, konkrétní načtení proměnné z intentu ale záleží na typu uložené datové proměnné. V závislosti něm je použita příslušná funkce třídy intent, například [14] :

```
Intent intent = getIntent();
Long long = intent.getLongExtra(String name, Long defaultvalue);
Char char  = intent.getCharExtra(String name, Char defaultvalue);
```

Pro jiné datové typy je postup analogický.

Další zajímavou zmíněnou vlastností je možnost spouštět aktivitu v konkrétním režimu pomocí tzv. Flags, konkrétně metody **addFlags**. [8, s. 70] Metodě můžou být předány následující hodnoty:

- **FLAG_ACTIVITY_NO_ANIMATION** – vypne animaci při spouštění aktivity
- **FLAG_ACTIVITY_NO_HISTORY** – spuštěná aktivita se neuloží do zásobníku. Když uživatel aktivitu opustí, bude ukončena.
- **IFLAG_ACTIVITY_CLEAR_TOP** – pokud už je spouštěná aktivita v zásobníku, obnoví se a přesune se navrch zásobníku.
- **FLAG_ACTIVITY_SINGLE_TOP** – pokud je spouštěná aktivita na vrcholu zásobníku, obnoví se a nespustí se nová aktivita

Příklad nastavení příznaku metodou addFlags:

```
Intent intent = new Intent();
Intent.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY);
```

Navíc jsou zde ještě další způsoby spuštění aktivity, které mohou být definovány v **AndroidManifest.xml** atributem **android:launchMode:** [8, s. 70]

- **standard** – vždy bude vytvořena nová instance
- **singleTop –** vždy bude vytvořena nová instance, pokud už není instance aktivity na vrcholu zásobníku.
- **singleTask** – bude vytvořena nová úloha, a pokud instance aktivity už existuje, dojde k přesměrování na ni. Typickým příkladem je webový prohlížeč či přehrávač multimédií.
- **singleInstance –** jako singleTask, ale v rámci úlohy je vždy jen jedna aktivita.

Tlačítko **Zpět** funguje vždy bez ohledu na hodnotu parametru **android:launchMode.**

3.2.5 **Layout**

Layout, česky tedy rozložení, definuje vizuální strukturu uživatelského rozhraní, jako například aktivita nebo widget. Layout je možno deklarovat dvojím způsobem [15] :

- **Deklarace elementů uživatelského rozhraní v souboru XML** – Android poskytuje přímočarý XML slovník, který odpovídá třídám a podtřídám, jak pro widgety, tak pro layout
- **Instancování layout elementů za běhu –** Aplikace je schopna vytvářet objekty View a ViewGroup (a manipulovat jejich vlastnostmi) pomocí příkazů.

Je k dispozici obrovské množství nástrojů, jak uživatelské rozhraní vytvořit (nastavení okrajů, vložení objektů, …). Každá aktivita ale musí být propojena s konkrétním layoutem, aby se vytvořilo dané uživatelské rozhraní. Toto probíhá přes funkci **setContentView(R.layout.activity)** typicky při vytváření aktivity v metodě **onCreate.** Uskutečněním tohoto příkazu se aktivita propojí s konkrétním layoutem a má možnost interakce s jeho objekty.

3.2.6 **Android Studio**

Po celou dobu práce měl autor možnost pracovat s vývojářským nástrojem Android Studio, který byl průběžně aktualizován. Instalace tohoto nástroje, který je zdarma dostupný na oficiálních stránkách *https://developer.android.com/studio/index.html* , je rychlá a jednoduchá. Program poskytuje úplně vše od vývojového prostředí, přes debugger, až po emulaci virtuálních zařízení. O tomto úžasném nástroji a jeho možnostech je napsána celá řada knih i oficiálních dokumentací. [16] Před jeho existencí a současnou podobou bylo potřeba projít poměrně složitou a zdlouhavou procedurou, které byl autor naštěstí ušetřen. [8, s. 1 - 48]

4 Vlastní práce

4.1 QR kódy

4.1.1 ZXING

Svou práci autor začal tím nejpodstatnějším. Odpovědí na otázku, jak bude tato aplikace QR kódy číst a hlavně, jak následně naloží s danými daty.

Bylo prozkoumáno několik open source knihoven, které vyhovovaly méně nebo více. Po této fázi se autor rozhodl zahrnout do svého programu knihovnu Zxing („Zebra Crossing)“ [17], její zařazení je provedeno přidáním následujících řádků kódu do Gradle scrips, build.gradle (Module: app):

```
compile 'com.google.zxing:core:3.2.1'
compile 'com.journeyapps:zxing-android-embedded:3.2.0@aar'
```

Tato knihovna poskytuje veškeré prostředky nutné pro čtení QR kódu, nejpodstatnější je třída **IntentIntegrator** a třída I**ntentResult**, potažmo jejich metody.

4.1.2 Zpracování

Spuštění skenování uživatel provede kliknutím tlačítka. Nejdříve je potřeba ho v příslušném layoutu aktivity vytvořit:

```
<Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="SCAN"
        android:id="@+id/bScan"/>
```

A poté v kódu aktivity načíst:

```
Button bScan = (Button) findViewById(R.id.bScan);
```

Následně je vytvořen Listener k tomuto tlačítku, ve kterém je obsažena třída IntentIntegrator:

```
bScan.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                IntentIntegrator integrator = new
                                            IntentIntegrator(activity);
                Integrator
                .setDesiredBarcodeFormats
                (IntentIntegrator.QR_CODE_TYPES);
                integrator.setPrompt("Scan");
                integrator.setCameraId(0);
                integrator.setBeepEnabled(false);
                integrator.setBarcodeImageEnabled(false);
                integrator.initiateScan();
            }
        });
```

V této chvíli už zbývá jen zajistit, jak nakládat s výsledkem tohoto čtení. Pro nakládání s výsledkem je zde použita třída **IntentResult**, pro uložení třída **Examination** (viz. Kapitola 4.1.3. – Třída Examination) a následně je proveden zápis do databáze. Podrobnější zpracování kódu probíhá v samotné třídě Examination.

```
@Override
    protected void onActivityResult (int requestCode, int
                                     resultCode, Intent data) {
        IntentResult result = IntentIntegrator.parseActivityResult
                              (requestCode, resultCode, data);
        if (result != null) {
           if (result.getContents() == null) {
               Toast.makeText(this,
              "Skenování bylo přerušeno", Toast.LENGTH_LONG).show();
           } else {
               Examination examination = new Examination();
               Toast.makeText(this, result.getContents(),
                              Toast.LENGTH_LONG).show();
               try {
                   examination = new
                                 Examination(result.getContents());
                   if(db.addExamination(examination)) {
                       Toast.makeText(this, "Vyšetření bylo úspěšně
                                      zaznamenáno",
                       Toast.LENGTH_SHORT).show();
                   } else {
                       Toast.makeText(this, "Chyba při zapisování
                                      vyšetření do databáze",
                       Toast.LENGTH_SHORT).show();
                   }
               }
               catch (Error e) {
                   Toast.makeText(this, e.getMessage(),
                                  Toast.LENGTH_LONG).show();
               }
```

```
            }
        }
        else {
            super.onActivityResult(requestCode, resultCode, data); }
    }
```

4.1.3 Třída Examination

Tato klíčová třída obsahuje kompletní vyšetření, jak už její název napovídá. Jedná se o objekt s následujícími vlastnostmi:

```
private String date;
private String place;
private String dose;
private String modality;
private long _id;
```

Jelikož jsou její vlastnosti privátní, musí samozřejmě obsahovat příslušné gettery a settery:

```
public long get_id() {
    return _id;
}
public void set_id(long _id) {
    this._id = _id;
}
public String getDate() {
    return date;
}
public void setDate(String date) {
    this.date = date;
}
public String getPlace() {
    return place;
}
public void setPlace(String place) {
    this.place = place;
}
public String getDose() {
    return dose;
}
public void setDose(String dose) {
    this.dose = dose;
}
public String getModality() {
    return modality;
}
public void setModality(String modality) {
    this.modality = modality;
}
```

Uchovává tedy informaci o datu vyšetření, místu vyšetření, dávce a modalitě vyšetření. V případě žádosti o více informací by tyto vlastnosti mohli být expandovány na kompletní DICOM [18] objekt, popřípadě jeho variaci s hlavičkou. Vlastnost _id je určena pouze pro párování s databází.

Dále obsahuje základní prázdný konstruktor:

```
public  Examination () {

    }
```

Nejdůležitější je však konstruktor, který jako parametr bere textový řetězec:

```
public Examination (String stringChain){
        Examination examination = resolveChain(stringChain);

        this.date = examination.date;
        this.place = examination.place;
        this.dose = examination.dose;
        this.modality = examination.modality;

    }
```

Jak je možno vypozorovat, tento konstruktor hned na svém počátku využívá funkci **resolveChain,** která je pro program klíčová. Jak se zobrazeno na další straně, jedná se o funkci privátní, je k tedy k dispozici pouze pro objekty typu Examination. Funkce totiž vezme data poskytnutá čtením QR kódu a velmi specifickým způsobem z nich vytvoří objekt této třídy. Mít tuto metodu k dispozici veřejně či jiným způsobem k dispozici postrádá smysl i účel. Tato funkce tedy obstarává rozkódování poskytnutého textového řetězce na smysluplné vlastnosti a vrátí objekt. Vše se děje následujícím způsobem:

```
private Examination resolveChain (String stringChain) {
        Examination returnExamination = new Examination();
        String[] result = new String[4];
        String tempString = "";
        int t = 0;

        for (int i = 0; i < stringChain.length(); i++) {
            if
(Character.toString(stringChain.charAt(i)).equals("*")) {
                result[t] = tempString;
                t++;
                tempString = "";
            }
            else
            {
                tempString += stringChain.charAt(i);
            }
        }

        if (t!=4) throw new Error("Kód není ve správném formátu
                                  nebo je poškozen");

        returnExamination.date = result[0];
        returnExamination.place = result[1];
        returnExamination.dose = result[2];
        returnExamination.modality = result[3];

        return returnExamination;
    }
```

Funkce je v tuto chvíli nastavena následovně: Předpokládá, že v poskytnutém textovém řetězci (který je jí předáván jako parametr při volání) se vyskytuje speciální znak „*“. Ten jí říká, že co doposud následovalo je konkrétní vlastnost objektu, speciálním znakem vlastnost končí a začíná vlastnost nová.

Pro zamezení redundance dat (a tím pádem i umožnění uložení více dat v kódu) se zde spoléhá na přesné pořadí vlastností objektu. Pokud přijdou v jiném pořadí, než které je nastavené (datum, místo, dávka, modalita), budou přiřazeny chybně. Toto lze však jednoduše ošetřit při generování QR kódu. Pokud je žádoucí „inteligentní“ metoda a nevadí nám ztráta potencionální kapacity QR kódu, bylo by vhodné implementovat formát JSON.

Příklad platného kódu:

Obrázek 11: Příklad platného QR kódu
Zdroj: Vlastní tvorba

21.12.1994*Krajská nemocnice Kladno*20 µGy*Rentgen zubu*

Příklad neplatného kódu, který vyvolá chybnou hlášku:

Obrázek 12: Příklad neplatného QR kódu
Zdroj: Vlastní tvorba

21.12.1994*Krajská nemocnice Kladno*20 µGy*Rentgen zubu

Jinak řečeno: Každý kód, který neobsahuje přesně čtyři znaky „*“ bude vyhodnocen jako chybný, a tak k němu program přistupuje.

4.2 Databáze

4.2.1 Výběr metody

Jakmile je ošetřeno načítání QR kódu a jeho zpracování na objekt dané třídy, je potřeba data někam ukládat. Po analýze několika aplikací, fór a oficiálních doporučených postupů [19] autor, stejně jako drtivá většina ostatních programátorů došel k závěru, že nejlepší možností je použít knihovnu **SQLiteOpenHelper** [20]**,** což je oficiální nástroj pro zacházení s databází v jazyku SQL. Není potřeba ho nijak zabudovávat do našeho programu, protože na platformě Android je nativně k dispozici. Ta poskytuje širokou řadu metod a možností, jak s daty nakládat. Jedná se o profesionální databázi a není tedy nutno řešit, jak funguje ukládání, mazání, indexování a podobně (obecné, ne konkrétní!). Nabízí se zde i možnost data ukládat vlastním způsobem (např. serializace), ale toto se při větším množstvím dat stává velmi neefektivním, mimoto tento způsob dále omezuje možnosti, s jakými lze s daty nakládat. Navíc se oproti předchozí možnosti jedná o velice neintuitivní možnost.

Jako u každé jiné databáze je potřeba databázi vytvořit, v ní vytvořit tabulku (popř. tabulky) a v ní vytvořit sloupce. Dále je potřeba specifikovat konkrétní metody pro mazání, vytváření a aktualizaci sloupců. Toto vše je obsažené ve třídě, která nese název **DatabaseHelper**.

4.2.2 DatabaseHelper

Jak už bylo zmíněno, jedná se o třídu, která má na svědomí veškerou práci s databází od přidávání vyšetření až po jejich mazání. Aby toho byla schopna, používá metody z knihovny SQLiteOpenHelper. Dále je vhodné poznamenat, že při těchto operacích je používán objekt **Cursor** [21]. Jedná se o rozhraní, které je už dle definice používáno pro ukládání/čtení výsledku, který vrátí dotaz nad databází. Opět se jedná o třídu, která je defaultně k dispozici a není nutno ji žádným způsobem přikládat k aplikaci.

Zde probíhá inicializace databáze:

```
public class DatabaseHelper extends SQLiteOpenHelper {

    private static final String DB_NAME = "dbRadioCare2.db";
    private static final int DB_VERSION = 2;

    private static final String TABLE_EXAMINATION =
                                                "examination_table";

    protected static final String COLUMN_ID = "_id";
    protected static final String COLUMN_DATE = "date";
    protected static final String COLUMN_PLACE = "place";
    protected static final String COLUMN_DOSE = "dose";
    protected static final String COLUMN_MODALITY = "modality";

    private static final String[] columns =
                              {COLUMN_ID,COLUMN_DATE,COLUMN_PLACE,
                               COLUMN_DOSE, COLUMN_MODALITY};
}
```

Metoda pro přidání vyšetření do databáze: Jak je vidět, je otevřena již existující databáze a do ní pomocí metody **db.insert** skrz proměnnou values vloženy konkrétní hodnoty sloupců. Pokud metoda selže, vrátí hodnotu 0, v opačném případě vrací hodnotu 1.

```
public boolean addExamination(Examination ex) {

        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues values = new ContentValues();

        values.put(COLUMN_DATE,ex.getDate());
        values.put(COLUMN_PLACE,ex.getPlace());
        values.put(COLUMN_DOSE,ex.getDose());
        values.put(COLUMN_MODALITY,ex.getModality());

        long operationState =
                              db.insert(TABLE_EXAMINATION,
                                        null,values);
        db.close();

        return operationState > 0;
    }
```

Vyšetření tedy nyní program umí ukládat, ale nepochybně bude také potřebovat vyšetření načítat. K tomu tedy slouží metoda **getExaminationById**. Tato metoda jako parametr bere **id** vyšetření, poté ho nalezne a vrátí pomocí dotazu nad databází.

```
public Examination getExaminationById(long id){
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor c = db.rawQuery("SELECT * FROM " +
                               TABLE_EXAMINATION + "
                               WHERE " + COLUMN_ID + " = '" + id
                               + "'",null);

        Examination ex = new Examination();

        if(c.moveToFirst()){
            do{
                ex.set_id(c.getInt(0));
                ex.setDate(c.getString(1));
                ex.setPlace(c.getString(2));
                ex.setDose(c.getString(3));
                ex.setModality(c.getString(4));
            } while (c.moveToNext());
        }
        return ex;
    }
```

Dále je potřeba metoda pro aktualizaci vyšetření, pokud by jej uživatel chtěl ručně upravit, jako parametr bere vyšetření. Samotná aktualizace je obdobně provedena skrz proměnnou values za pomocí metody **db.update**.

```
public boolean updateExamination(Examination ex) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues values = new ContentValues();

        values.put(COLUMN_DATE,ex.getDate());
        values.put(COLUMN_PLACE,ex.getPlace());
        values.put(COLUMN_DOSE,ex.getDose());
        values.put(COLUMN_MODALITY,ex.getModality());

        long operationState =
db.update(TABLE_EXAMINATION,values,COLUMN_ID
                                                                +
                                                               " =
?", new String[]

{String.valueOf(ex.get_id())});
        db.close();
        return operationState > 0;
    }
```

A jako poslední rozdílná akce je potřeba metoda pro mazání vyšetření, je tedy vytvořena metoda pro provedení tohoto úkonu. Jako parametr bere id vyšetření, které chceme smazat a úkon provede pomocí metody **db.delete.**

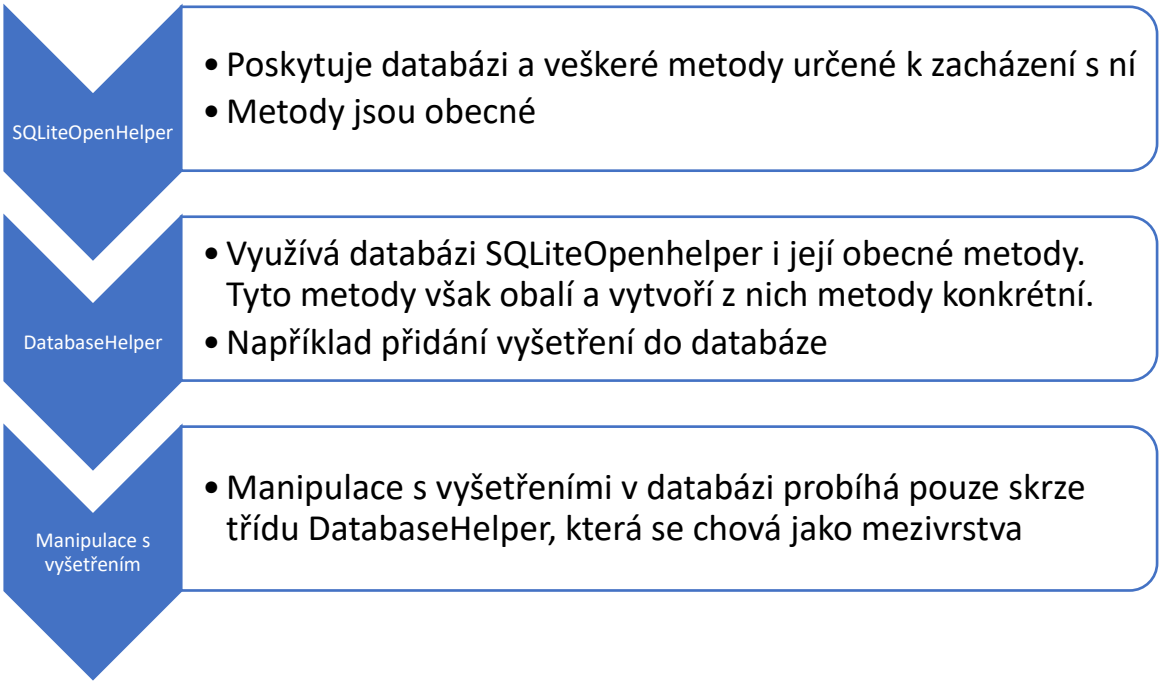
```
public boolean deleteExaminatin(long id) {
        SQLiteDatabase db = this.getWritableDatabase();
        String[] selectionArgs = {String.valueOf(id)};

        long operationState =
                    db.delete(TABLE_EXAMINATION,COLUMN_ID + "=
                              ?",selectionArgs);
        db.close();
        return operationState > 0;
    }
```

Nesmí se však opomenout ještě jedna metoda, která ušetří v budoucnu práci. Jedná se o velice podobnou metodu na získání vyšetření – akorát v tomto případě metoda vrátí vyšetření všechna.

```
public Cursor getExamination(){
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor c =
                    db.query(TABLE_EXAMINATION,columns,
                             null,null,null,null,
                             COLUMN_ID + " DESC");
        return c;
    }
```

Zde by se autor rád na moment pozastavil a jednoduchým diagramem znázornil, jak jsou do sebe jednotlivé úkony zakomponovány.

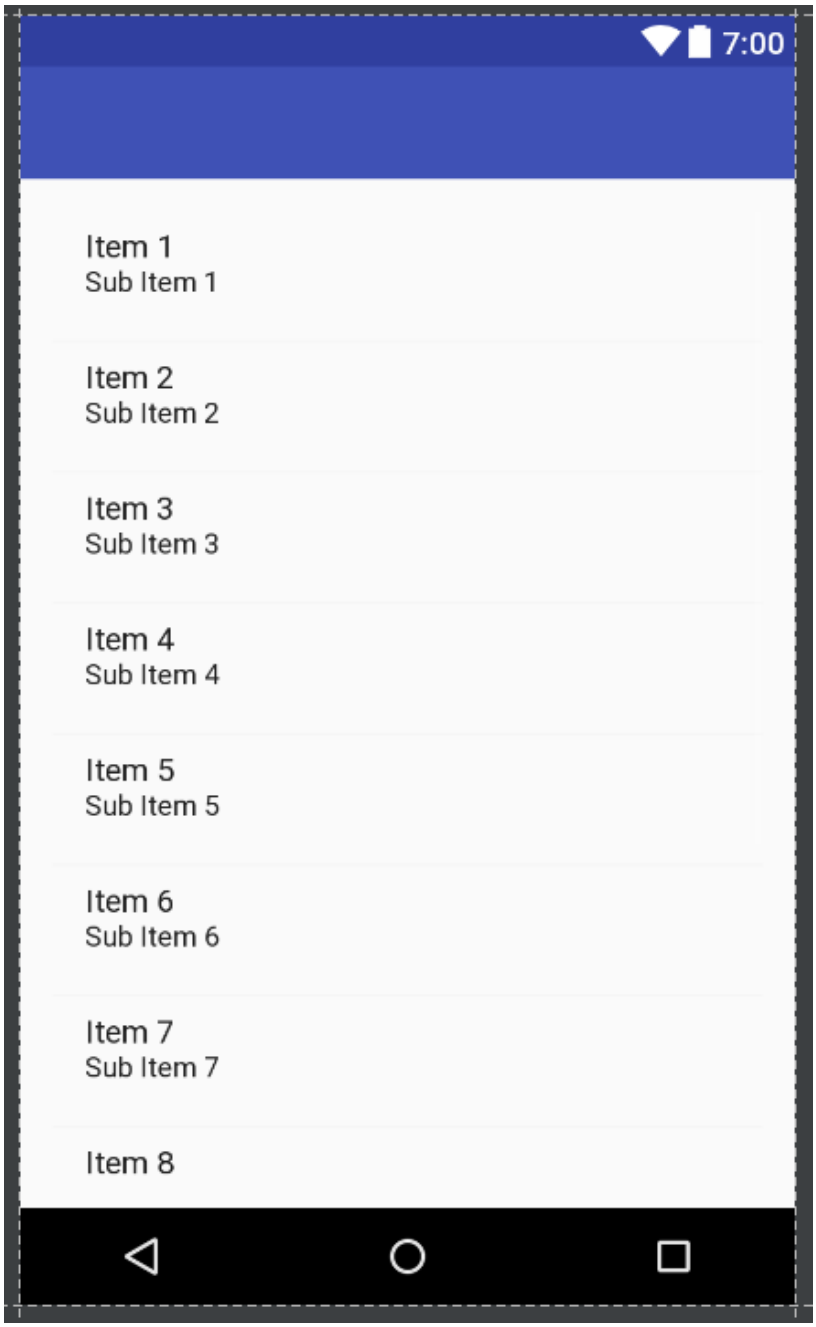


Obrázek 13: Průběh manipulace s vyšetřením.
Zdroj: Vlastní tvorba

4.3 ListView

4.3.1 ListView obecně

ListView je jedna ze základních možností pro vizualizaci objektů, které poskytuje Android Studio. Už z názvu je patrné, o co se jedná: List (seznam) + View (pohled), tedy seznam pohledů. Pod pojmem je možno si představit konkrétní layout, který bude duplikován a z něj bude vytvářen seznam. Zní to trošku zamotaně, ale princip je jednoduchý.



Obrázek 14: Obecný náhled na ListView
Zdroj: Vlastní tvorba

Z obrázku na předchozí straně je to patrné: Oblast „Item 1 Sub Item 1“ představuje View – layout. Konkrétní data jsou variabilní a bude se tedy tvořit seznam.

4.3.2 **Layout**

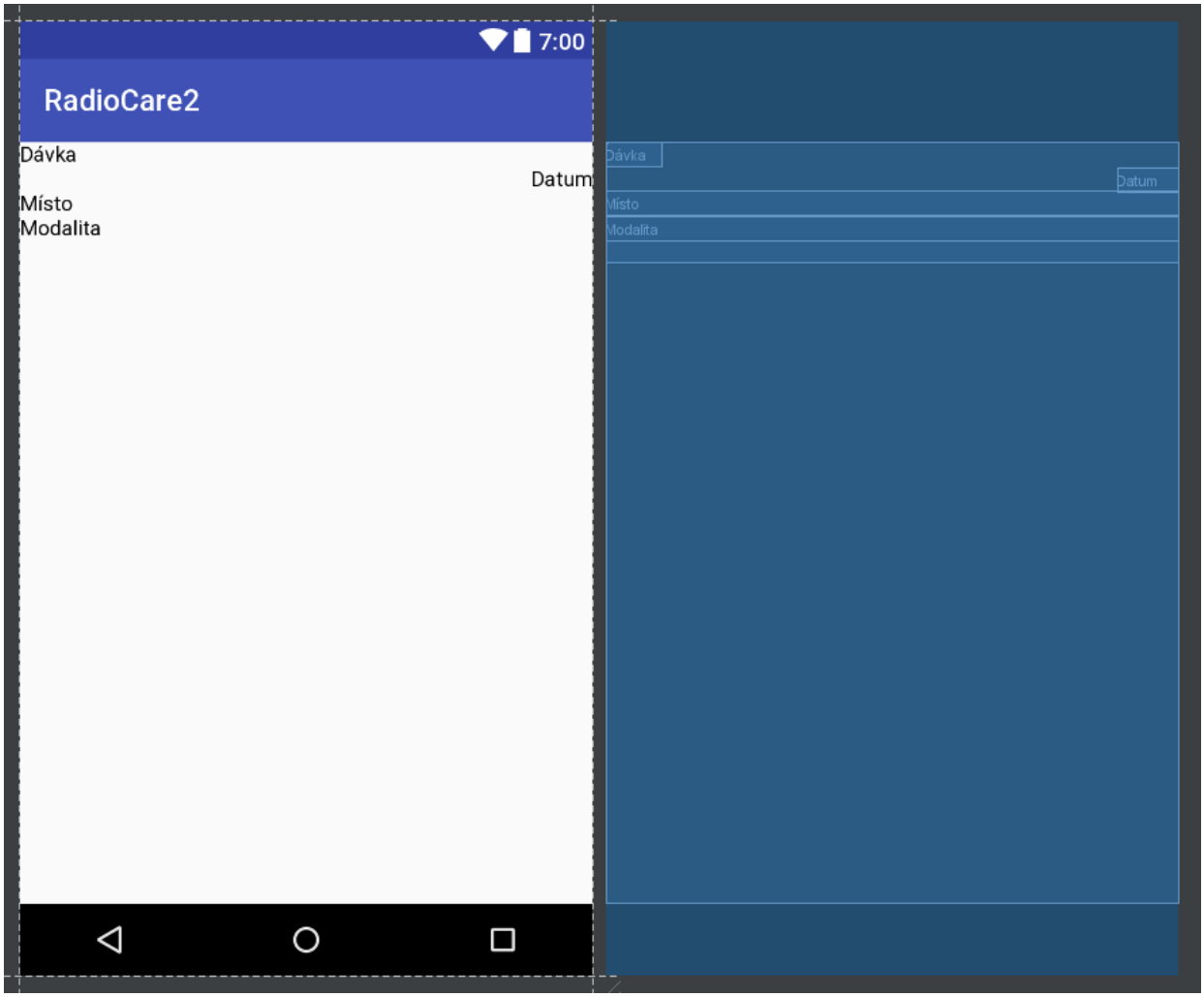
Jako první věc je nutné si tedy vytvořit layout. Je to stejný layout typu .xml, který používají aktivity. Akorát v tomto případě není k žádné aktivitě „natvrdo“ přiřazen.
Všechny layouty můžeme nalézt v „res/layout.“

Autor si vytvořil tedy jeden takový, který obsahuje čtyři objekty TextView. To je opět základní kámen, který poskytuje Android Studio, umožňuje pouze zobrazení textu.
V tomto případě není nutno nastavovat vše relativně, postačí lineární nastavení s vlastností android:gravity, která určuje, jakým směrem bude objekt „přitahován.“

```
<TextView
        android:id="@+id/tvDate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="Datum"
        android:textColor="#000000" />
```

Celý tento layout bude v konečné fázi několikrát naplněn a zobrazen ve vytvořeném ListView. Cokoliv, co je sem tedy přidáno, bude v iteracích vytvořeno. Protože je žádoucí uživateli data pouze zobrazovat, je vhodné, aby prvky v tomto layoutu nebyly žádným způsobem interaktivní. Konkrétní akce mohou později být zakomponovány pomocí ListView, který poskytuje hned několik možností, o tom více později.

Celý layout poté vypadá takto:



Obrázek 15: Náhled na Layout
Zdroj: Vlastní tvorba

4.3.3 Adaptér (Wrapper)

„Návrhový vzor Adapter (nebo také Wrapper) se používá při práci s komponentou, která má nestabilní nebo s naší aplikací nekompatibilní rozhraní. Umožňuje komponentu obalit naším rozhraním, a tak aplikaci úplně odstínit od rozhraní původního.“ [22]

Adaptér je jeden z návrhových vzorů, který obecně navrhuje řešení problému s kompatibilitou. Je zde ale nějaký problém? Skutečně je tomu tak. Jsou vytvořeny čtyři instance TextView a objekt Examination. Pro tento program jsou to ale „jablka a hrušky“.

Musí mu tedy být vysvětleno, kam a jak má jakou vlastnost objektu promítnout/použít. Je tedy vytvořena nová třída **ExaminationAdapter**.

```
public class ExaminationAdapter extends CursorAdapter {

    public ExaminationAdapter(Context context, Cursor c, int
flags) {
        super(context, c, flags);
    }

    @Override
    public View newView(Context context, Cursor cursor,
ViewGroup parent) {
        LayoutInflater inflater = LayoutInflater.from(context);
        View view =
inflater.inflate(R.layout.examination_layout,parent,false);
        bindView(view,context,cursor);
        return view;
    }
```

Jak je možno si všimnout, tato třída dědí ze třídy **CursorAdapter**. To svědčí o povaze způsobu, jakým budou data přenášena. Také se zde objevuje **LayoutInflater**, což je způsob, jak, doslova „nafouknout Layout daty." Použita je i funkce **bindView**, která je definována následovně:

```
@Override
    public void bindView(View view, Context context, Cursor
cursor) {
        DatabaseHelper db = new DatabaseHelper(context);

        int id =
cursor.getColumnIndex(DatabaseHelper.COLUMN_ID);
        int date = cursor.getColumnIndex(db.COLUMN_DATE);
        int place = cursor.getColumnIndex(db.COLUMN_PLACE);
        int dose = cursor.getColumnIndex(db.COLUMN_DOSE);
        int modality =
cursor.getColumnIndex(db.COLUMN_MODALITY);


((TextView)view.findViewById(R.id.tvId)).setText(cursor.getStrin
g(id));

((TextView)view.findViewById(R.id.tvDate)).setText(cursor.getStr
ing(date));

((TextView)view.findViewById(R.id.tvPlace)).setText(cursor.getSt
ring(place));
```

```
((TextView)view.findViewById(R.id.tvDose)).setText(cursor.getStr
ing(dose));

((TextView)view.findViewById(R.id.tvModality)).setText(cursor

.getString(modality));
    }
```

Zde se nachází jádro děje. Jednotlivé vlastnosti id, date, place, dose a modality jsou zde načteny z instance třídy **Cursor**. Poté jsou přetypovány na typ String a pomocí metody **setText** jsou přeneseny do instancí třídy TextView.

4.3.4 Aktivita ListView

Poslední potřebná komponenta je aktivita, kterou uživatel uvidí a kde se bude vše odehrávat. Je tedy vytvořena aktivita **ListExaminationActivity** a k ní i příslušný layout. Ten však bude velmi jednoduchý a bude obsahovat pouze objekt **ListView** (viz. Obrázek 6: Obecný náhled na ListView), popř. pro lepší navigaci i jednoduchý **Toolbar**.

```
<android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?attr/colorPrimary"
        android:minHeight="?attr/actionBarSize"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true">
    </android.support.v7.widget.Toolbar>

    <ListView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingBottom="@dimen/activity_vertical_margin"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/toolbar" />
```

Je potřeba provést ještě poslední krok, a tím je načtení dat. Zde by autor rád upřesnil, jak tato operace bude probíhat, než bude uvedena názorná ukázku. Data o vyšetření se nacházejí v databázi, budou tedy hledávána tam. Poté je potřeba data načíst, aby s nimi mohlo být manipulováno. Data v této formě ale nevyhovují z hlediska kompatibility, musí tedy tedy použit adaptér. Po všech těchto krocích nic nebrání tomu, aby data byla správně zobrazena.

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list_examination);
        ListView listView = (ListView) findViewById(R.id.list);
        db = new DatabaseHelper(getApplication());
        Cursor c = db.getExamination();
        ExaminationAdapter adapter = new
                    ExaminationAdapter(getApplication(),c,0);
        listView.setAdapter(adapter);
}
```

Všechny nutné kroky už jsou zavedeny a **ListView** je plně v provozu. Zbývají už jen dvě věci pro doladění. Tou první je zakomponování **Toolbaru**:

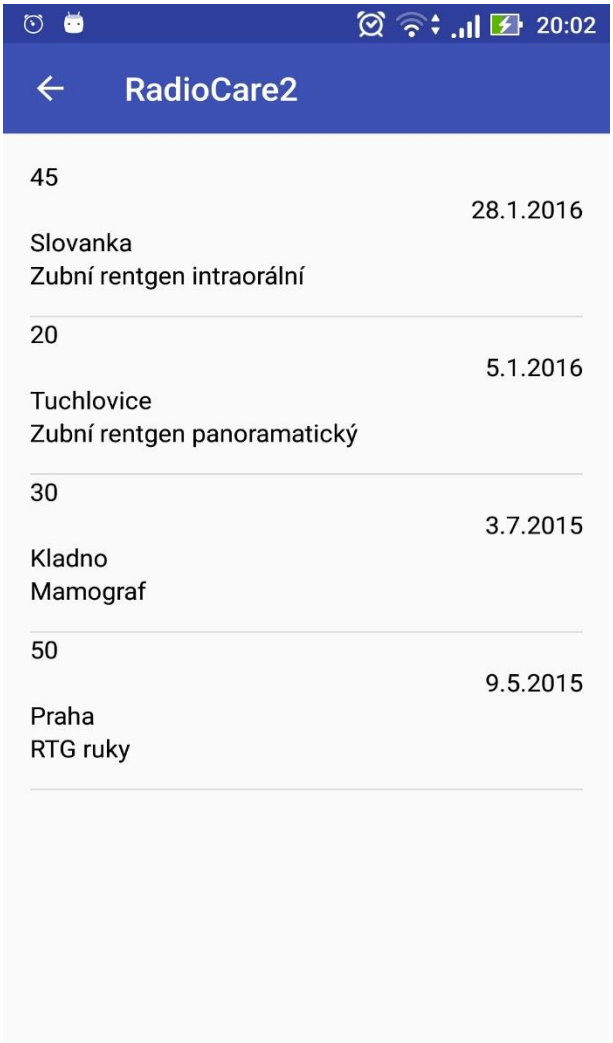
```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        toolbar.setTitleTextColor(Color.WHITE);
```

Druhý úkon umožní po kliknutí na vyšetření v tomto seznamu dané vyšetření upravit, o tom v následující podkapitole.

```
listView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View
view, int position, long id) {
                runUpdateActivity(id);
            }
        });
```

```
private void runUpdateActivity(long id) {
        Intent intent = new Intent(getApplication(),
UpdateExaminationActivity.class);
        intent.putExtra("id",id);
        startActivity(intent);
    }
```

Zde je patrné, že je po kliknutí na konkrétní vyšetření spuštěna pomocí intentu aktivita **UpdateExaminationActivity** a do instance třídy **Intent**, přes kterou je spouštěna, je přikládána informace o **id konkrétního vyšetření.**



Obrázek 16: Reálný příklad ListView
Zdroj: Vlastní tvorba

Samotné spuštění této aktivity je poté nutno zakomponovat, v tomto případě pro něj autor vytvořil následující funkci:

```
public void showListActivity(View view) {
        Intent intent = new
Intent(getApplication(),ListExaminationActivity.class);
        startActivity(intent); }
```

Nesmí být opomenut fakt, že se uživatel může k uložené aktivitě vrátit pomocí tlačítka „zpět“ a zobrazit si staré údaje, které již ale nejsou aktuální, čímž nastane akorát zmatek a nejednotnost informací. Proto je nutné nejen při startu, ale i při každém obnovení aktivity obsah seznamu aktualizovat.

Toho je docíleno přepsáním (@Override – potlačení) původní – defaultní metody onResume:

```
@Override
    protected void onResume() {
        super.onResume();
        db = new DatabaseHelper(getApplication());
        Cursor c = db.getExamination();
        ExaminationAdapter adapter = new
ExaminationAdapter(getApplication(),c,0);
        ListView listView = (ListView) findViewById(R.id.list);
        listView.setAdapter(adapter);

    }
```

Nyní se při každém startu (tím pádem i restartu, viz. obrázek č. 10) a obnovení aktivity seznam aktualizuje. Jedná se vlastně o (téměř) identický kód, který se nachází v metodě onCreate. Správně by tedy měl být vložen do funkce (např. refreshListView) a tato fukce volána. Pro demonstrační účely nechává autor kód takto, ale pointa je jasná.

4.4 Možnosti zacházení s vyšetřením

4.4.1 Vytvoření nového vyšetření pomocí QR kódu

Tato procedura je popsána v kapitole **4.1 QR kódy**, konkrétně **4.1.2 Zpracování** a to včetně kódu, jak se vše děje. Zde je vysvětleno, jak je toto zakomponováno.

Jak je již patrné z odkazované kapitoly, o vše je už postaráno. Nyní jen zbývá vytvořit rozhraní, přes které bude uživatel funkce volat. Pro tento účel bude však stačit obyčejné tlačítko:

```
<Button
        android:id="@+id/bScan"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="12dp" />
```

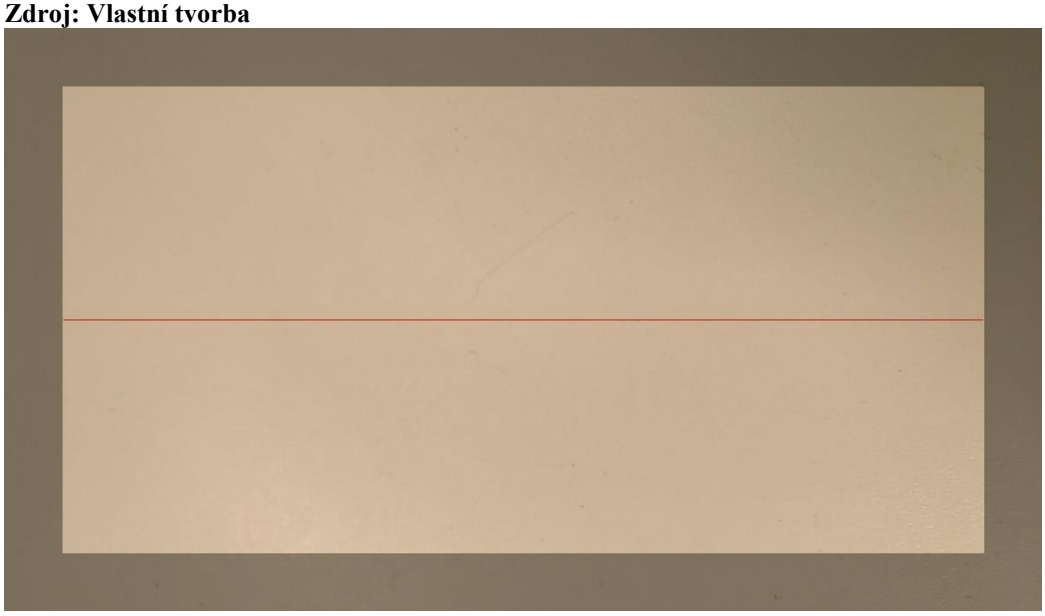
Které na kliknutí provede následující část kódu:

```
bScan.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                IntentIntegrator integrator = new
                                        IntentIntegrator(activity);
                integrator.setCameraId(0);
                integrator.setBeepEnabled(true);
                integrator.setBarcodeImageEnabled(false);
                integrator.initiateScan();
            }
        });
```

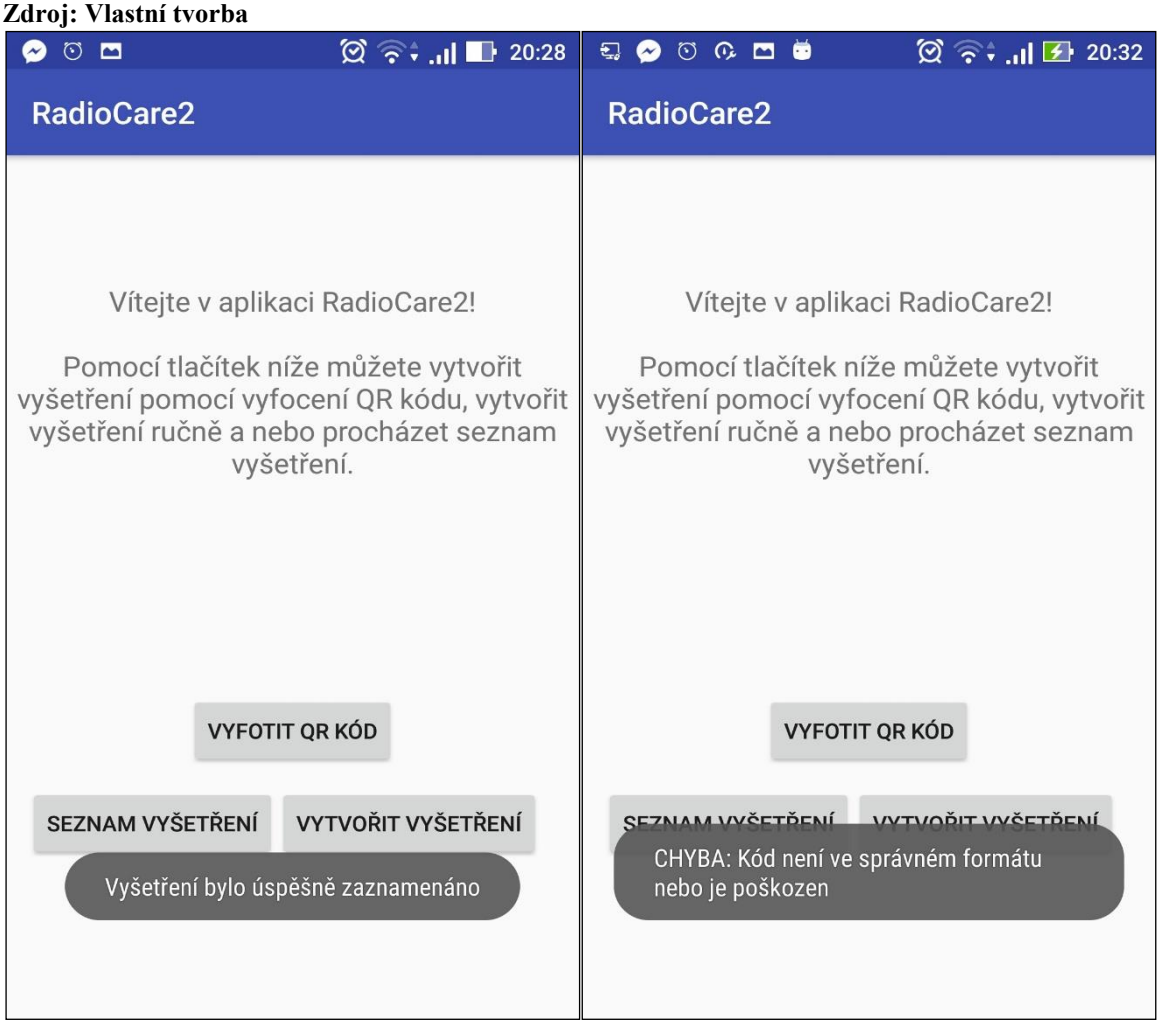
A je hotovo! Po kliknutí na tlačítko bude zahájeno skenování QR kódu a v případě úspěšného skenování a správného formátu kódu bude v databázi vytvořen nový záznam vyšetření s vlastnostmi, které byly v kódu obsaženy. Uživatel bude o výsledku operace informován pomocí widgetu **Toast**[2]. V případě, že bude operace neúspěšná, bude zobrazena chybová hláška

[2] Jedná se o široce používaný widget nativně zabudovaný v operačním systému Android. Slouží k zobrazování zpráv uživateli (viz. např. obrázek č. 18 a 19).

Obrázek 17: Pohled při skenování
Zdroj: Vlastní tvorba



Obrázek 18: Úspěšné skenování
Zdroj: Vlastní tvorba



Obrázek 19: Neúspěšné skenování
Zdroj: Vlastní tvorba

4.4.2 Vytvoření nového vyšetření manuálně

Pro netypické případy (nemohl být vygenerován QR kód, uživatel si ho zapomněl nahrát, …) je vhodné zadat možnost ručního vložení vyšetření. Pro tyto účely je zde opět přítomné jednoduché tlačítko:

```
<Button
        android:id="@+id/bCreateExamination"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Vytvořit vyšetření"
        android:layout_marginRight="24dp"
        android:layout_marginEnd="24dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:layout_marginBottom="103dp" />
```

A je mu přiřazena ativita na kliknutí:

```
bCreateExam.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new
Intent(getApplication(),CreateExaminationActivity.class);
                startActivity(intent); }
});
```

Po kliknutí na tlačítko bude tedy spuštěna aktivita **CreateExaminationActivity.** Zde se nachází RelativeLayout (uspořádání prvků), TextView (pouze zobrazení textu), EditText (je možno do něj zadat text) a tlačítko na vytvoření vyšetření. Je zbytečné uvádět celý layout, autor se tedy omezí na ukázku jedné instance každého prvku:

```
<EditText
            android:id="@+id/etModality"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:ems="10"

tools:backgroundTint="@android:color/background_dark"
            android:layout_alignBaseline="@+id/textView8"
            android:layout_alignBottom="@+id/textView8"
            android:layout_alignParentRight="true"
            android:layout_alignParentEnd="true" />
```

```
<TextView
            android:id="@+id/textView5"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Dávka"
            android:layout_alignBaseline="@+id/etDose"
            android:layout_alignBottom="@+id/etDose"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true" />
```

```
<Button
            android:id="@+id/bCreateExamination"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentEnd="true"
            android:layout_alignParentLeft="true"
            android:layout_alignParentRight="true"
            android:layout_alignParentStart="true"
            android:layout_below="@+id/etModality"
            android:elevation="0dp"
            android:text="Vytvořit" />
```

```
<RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/toolbar"
        android:paddingBottom="@dimen/activity_vertical_margin"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:id="@+id/relativeLayout">
```

Předem je známo, že data pro vytvoření tohoto vyšetření bude zadávat uživatel a z objektů budou požadována. Také je známo, že vyšetření je nutno zapsat do databáze, následovně je tedy připraveno bitevní pole.

```
DatabaseHelper db;
EditText etDate;
EditText etDose;
EditText etPlace;
EditText etModality;

private Examination examination = new Examination();
```

Poté je nutno k tlačítku přiřadit funkci na vytvoření vyšetření:

```
Button bCreateExamination = (Button)
      findViewById(R.id.bCreateExamination);

bCreateExamination.setOnClickListener(new View.OnClickListener()
      {
            @Override
            public void onClick(View v) {
                createExamination(examination);
            }
        });
```

Což je samo o sobě důležité, ale nic nedělající. Byla přiřazena k tlačítku funkce **createExamination**, ale žádná taková funkce ještě neexistuje. Zatím.

```
private void createExamination(Examination ex) {
        ex.setDate(etDate.getText().toString());
        ex.setDose(etDose.getText().toString());
        ex.setPlace(etPlace.getText().toString());
        ex.setModality(etModality.getText().toString());

        if (db.addExamination(ex)) {
            Toast.makeText(this,"Vyšetření bylo přidáno",
Toast.LENGTH_SHORT).show();
            backToPreviousActivity();
        } else {
            Toast.makeText(this,"Editace se nepovedla",
Toast.LENGTH_SHORT).show();
        }
    }
```

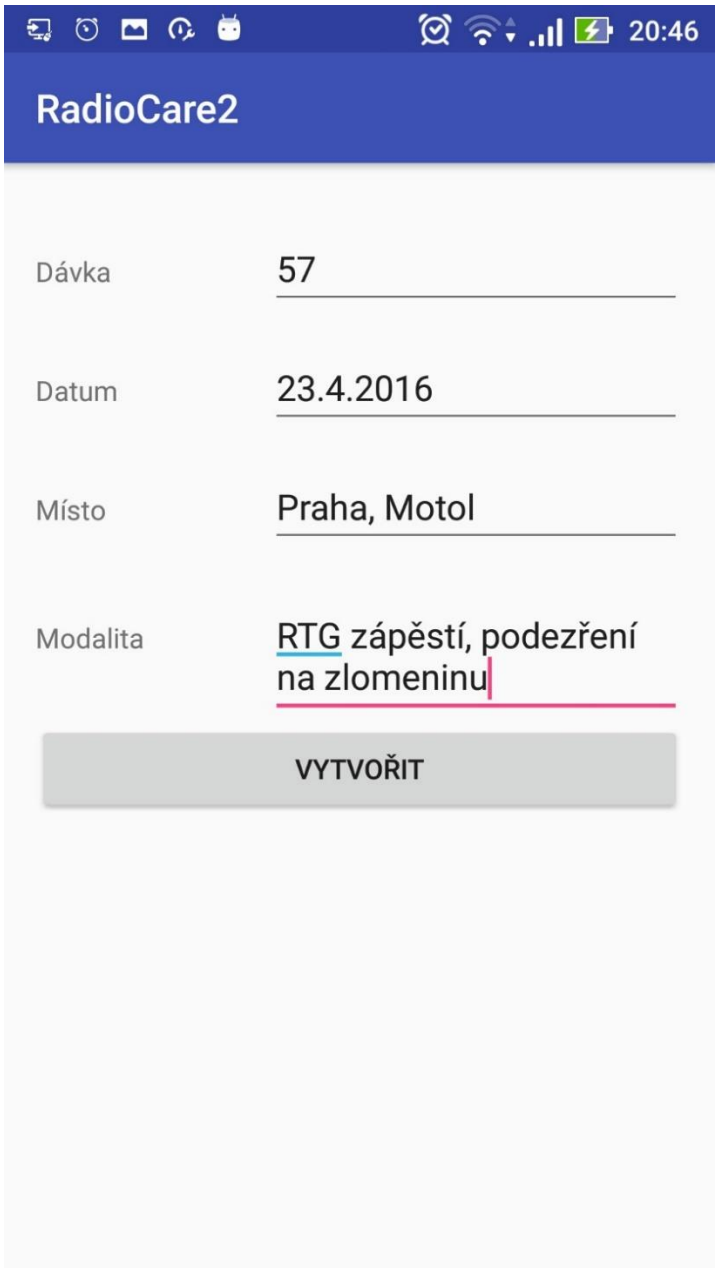
Nyní je v arsenálu funkce, která z objektů vhodně načte data, které zadal uživatel. Poté z nich vytvoří vyšetření, a to zapíše do databáze, o průběhu informuje uživatele. Zbývá toto vše propojit:

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_examination);
        Button bCreateExamination = (Button)
                          findViewById(R.id.bCreateExamination);

        db = new DatabaseHelper(this);
        etDate = (EditText) findViewById(R.id.etDate);
        etDose = (EditText) findViewById(R.id.etDose);
        etPlace = (EditText) findViewById(R.id.etPlace);
        etModality = (EditText) findViewById(R.id.etModality);
}
```

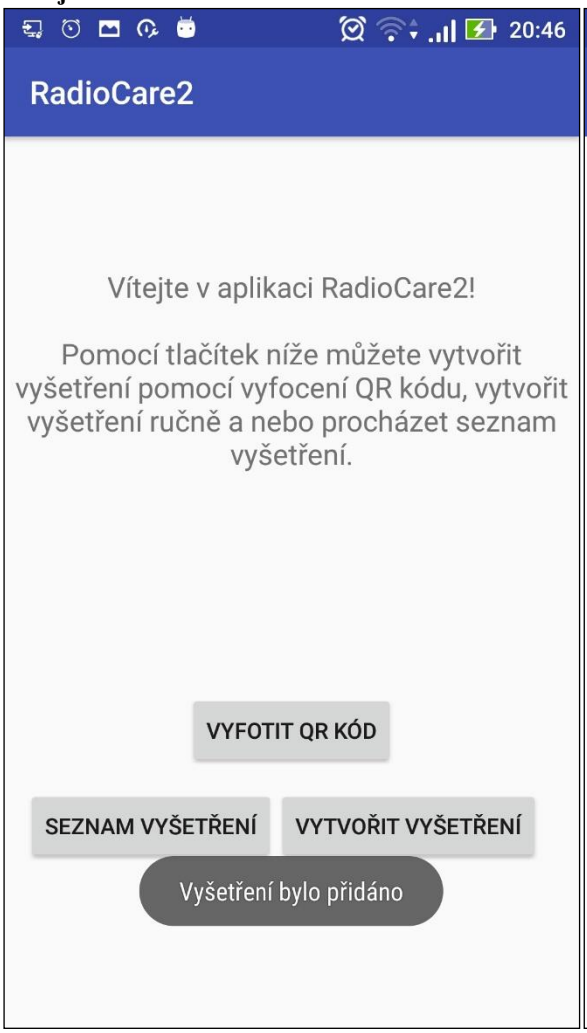
Ještě maličkost. Po založení vyšetření už není důvod setrvávat na tomto místě, je tedy umožněn návrat do menu:

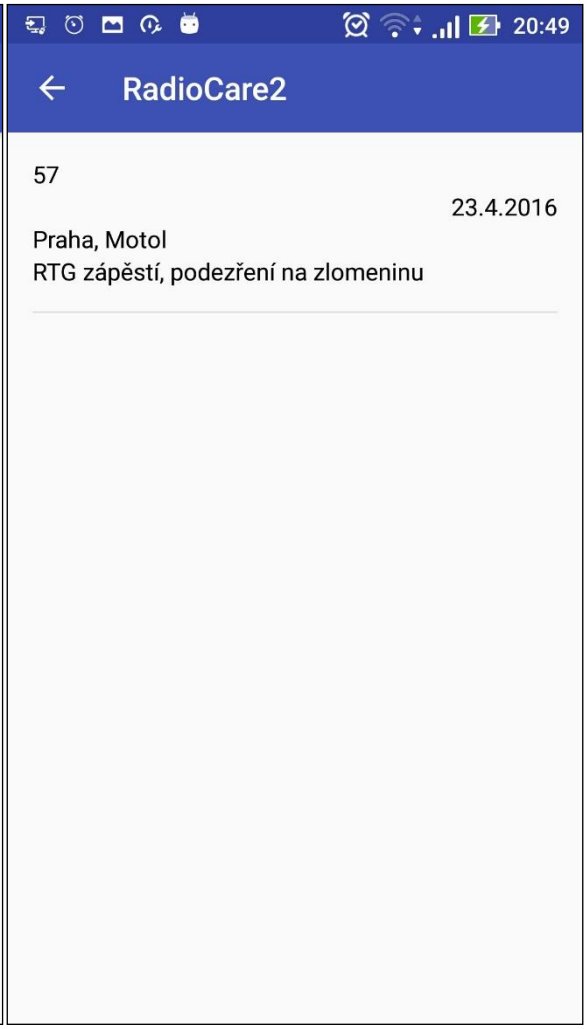
```
private void backToPreviousActivity() {
        Intent intent = new
Intent(getApplication(),MainActivity.class);
        startActivity(intent);
    }
```



Obrázek 20: Ruční zadání vlastností vyšetření
Zdroj: Vlastní tvorba

Obrázek 21: Úspěšné zapsání vyšetření do databáze.
Zdroj: Vlastní tvorba





Obrázek 22: Náhled na vytvořené vyšetření
Zdroj: Vlastní tvorba

4.4.3 Manuální smazání vyšetření

Pro smazání vyšetření je neintuitivní vytvářet samostatnou aktivitu. Je zde tedy použit jeden z méně typických prvků – ImageButton. Jedná se v podstatě o tlačítko, jehož vzhled je nahrazen libovolným obrázkem. Toto tlačítko posléze bude zakomponováno do aktivity pro aktualizaci/upravení vyšetření v následující podkapitole. Obrázek ale musí být neustále k dispozici, je tedy vhodné ho přidat k ostatním zdrojům do složky „res/drawable."

Vytvoření ImageButton je prováděno mimo relativní layout, kde jsou ostatní objekty. Pro demonstraci už je při vytváření tlačítku přiřazena funkce **deleteExamination**, což ale není doporučený postup. V .xml layoutu aktivity vytvoření tlačítka vypadá takto:

```
<ImageButton
        android:id="@+id/imageButton2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:layout_marginEnd="12dp"
        android:layout_marginRight="12dp"
        android:layout_marginTop="10dp"

android:background="@drawable/ic_delete_forever_white_24dp"
        android:onClick="deleteExamination"
        android:paddingTop="0dp" />
```

A nesmí být zapomeno funkci na smazání vyšetření vytvořit:

```
public void deleteExamination(View view) {
        deleteExamination(longId);
    }

private void deleteExamination(long id) {
        if (db.deleteExamination(id)){
            Toast.makeText(this, "Smazání proběhlo úspěšně",
                           Toast.LENGTH_SHORT).show();
            backToPreviousActivity();
        }
        else {
            Toast.makeText(this, "Vybraný záznam nebyl smazán,
                           vyskytla se chyba",
                           Toast.LENGTH_SHORT).show();
        }
    }
```

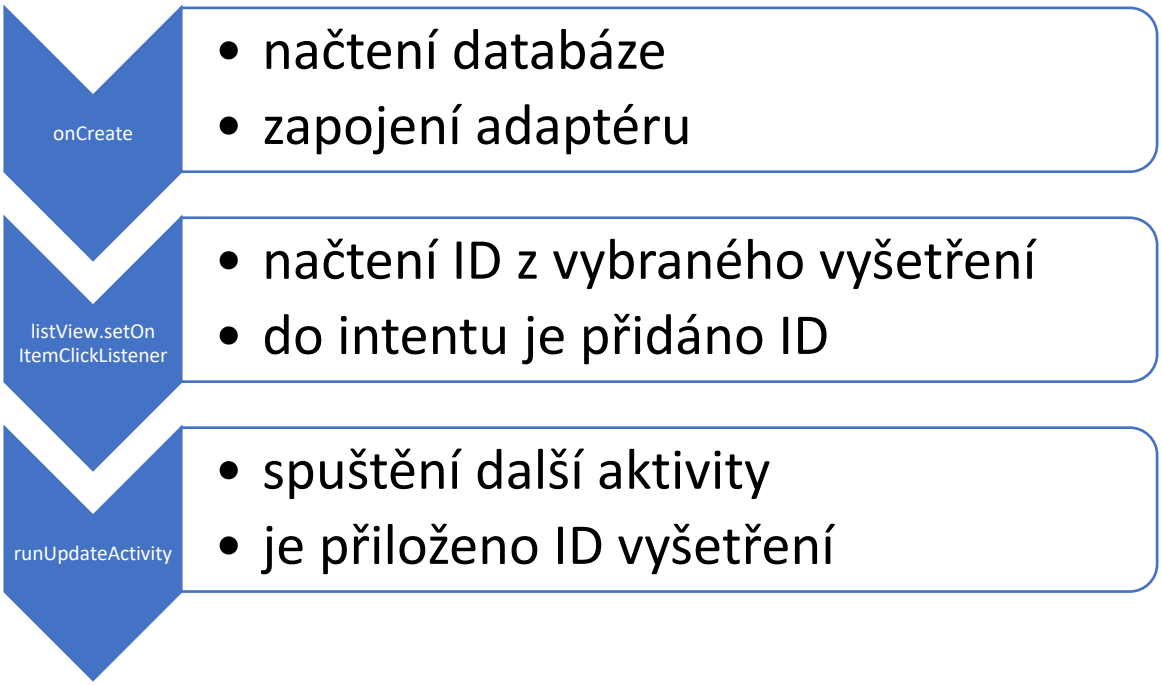
Funkce nalezne a smaže vyšetření v databázi dle unikátního ID. Toto id je v aktivitě uložené ve volně přístupné proměnné id. Dostane se tam, jak je známo, z předchozí aktivity a načtení proběhne ve funkci **onCreate**.

```
longId = intent.getLongExtra("id",-1);
```

4.4.4 Editace vyšetření

Pro upravení vyšetření je zde vytvořena aktivita **UpdateExaminationActivity** s příslušným layoutem. Autor nebude layout nijak rozebírat, protože je kromě absence objektu ImageButton totožný s layoutem aktivity **CreateExaminationActivity**.

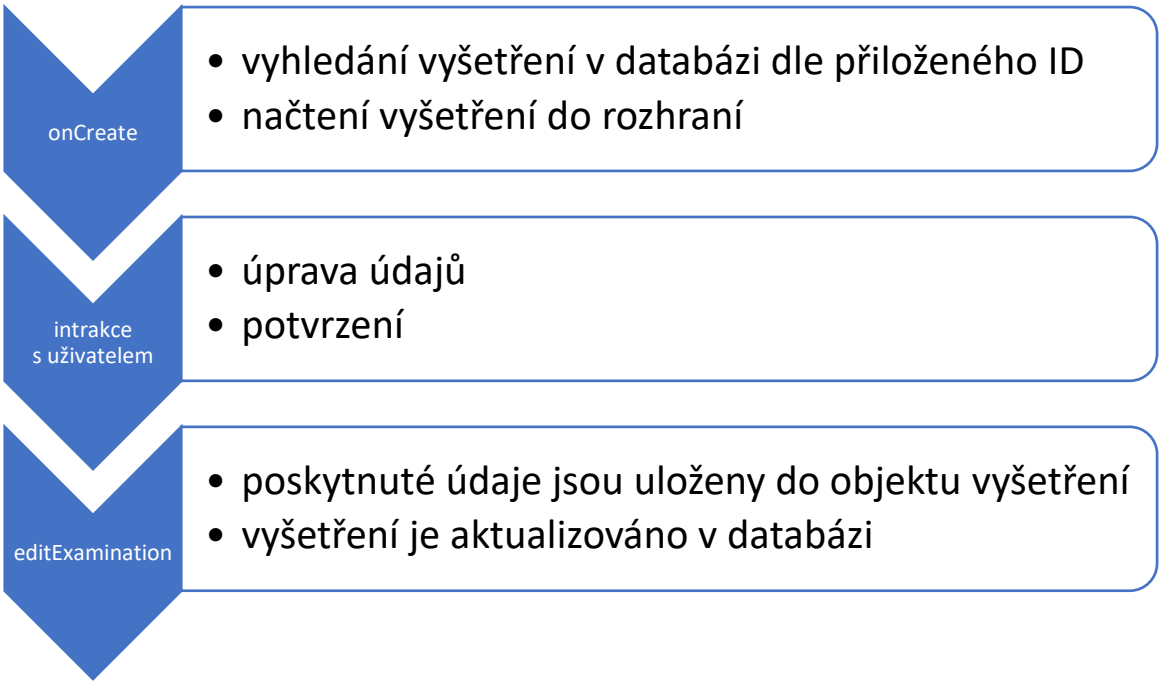
Je načase stručně ujasnit, jak by se věci měly dít: Uživatel se právě kouká na seznam jednotlivých vyšetření, poté na jedno z nich klikne a očekává, že ho bude moci upravit. Co to znamená pro autora (kromě spousty práce)? Bude muset být zaznamenáno id, tedy unikátní hodnota, podle které vyšetření, na které uživatel kliknul, bude nalezeno a v této aktivitě načteno. V předchozích kapitolách autor naznačil, jak celý tento proces probíhá, zde je na něj náhled. Proces začíná aktivitou **ListExaminationActivity**:



Obrázek 23: Průběh procesu editace vyšetření
Zdroj: Vlastní tvorba

Konkrétní kód těchto úkonů je k dispozici v předchozích kapitolách.

Nyní náhled na to, jak probíhá proces v následující aktivitě **UpdateExaminationActivity**:



Obrázek 24: Průběh procesu editace vyšetření
Zdroj: Vlastní tvorba

Nyní, když je proces znám, následuje pohled na jeho konkrétní implementaci. Nejdříve náhled na funkci **onCreate**:

```
@Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_update_examination);

        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        toolbar.setTitleTextColor(Color.WHITE);

        db = new DatabaseHelper(this);
        etDate = (EditText) findViewById(R.id.etDate);
        etDose = (EditText) findViewById(R.id.etDose);
        etPlace = (EditText) findViewById(R.id.etPlace);
        etModality = (EditText) findViewById(R.id.etModality);


        Intent intent = getIntent();
        longId = intent.getLongExtra("id",-1);
        fillControls(longId);
    }
```

Provede se načtení databáze a inicializace objektů, co je ale klíčové:
Do proměnné longId je načteno id vyšetření, které je poskytnuto z předchozí aktivity. Poté je spuštěna funkce **fillControls** s tímto id.

```
private void fillControls(long id){
        ex = db.getExaminationById(id);

        etDate.setText(ex.getDate());
        etDose.setText(ex.getDose());
        etPlace.setText(ex.getPlace());
        etModality.setText(ex.getModality());
    }
```

Tato funkce provede jednoduchou operaci: Najde v databázi podle id konkrétní vyšetření, a to poté načte do rozhraní. V tuto chvíli má uživatel možnost s danými údaji jakkoliv manipulovat. Až bude hotov, potvrdí úpravy stiskem tlačítka, které má přiřazenou následující funkci **editExamination**:

```
public void editExamination (View view){
        ex.setDate(etDate.getText().toString());
        ex.setDose(etDose.getText().toString());
        ex.setPlace(etPlace.getText().toString());
        ex.setModality(etModality.getText().toString());

        if (db.updateExamination(ex)) {
            Toast.makeText(this,"Vyšetření bylo upraveno",
                           Toast.LENGTH_SHORT).show();
            backToPreviousActivity();
        } else {
            Toast.makeText(this,"Editace se nepovedla",
                           Toast.LENGTH_SHORT).show();
        }
    }
```

Operace nejdříve z prvků, do kterých uživatel zadal informace, data načte a vytvoří nové vyšetření. Vyšetření, které jsme načetli, má posléze své vlastnosti nahrazeny těmi, které zadal uživatel. Nechybí funkce backToPrevious, která uživatele po potvrzení úpravy vrátí na seznam veškerých vyšetření.

Nesmí však být opomenuta jedna maličkost. Podobně jako u případu s aktualizací databáze nesmí být neopomenuty možnosti uživatele. Pokud zmáčkne tlačítko „zpět“, ze zásobníku by se tato aktivita nahrála i s předchozími daty, která už neexistují. Program je ošetřený, tudíž uživatel ani nemá možnost s touto aktivitou zacházet a akorát bude lapený ve velmi matoucí situaci. Problém může být vyřešit následovně a je na něm i demonstrován životní cyklus aktivity:

```
@Override
    protected void onRestart() {
        super.onRestart();
        Intent intent = new
Intent(getApplication(),MainActivity.class);
        startActivity(intent);
    }
```

V tuto chvíli je nastavení takové, že jakmile se uživatel pokusí k dané aktivitě dostat zpět, bude přesměrován do menu.

4.5 Nabídka menu

Několikrát je v textu zmíněna nabídka menu. Opět se jedná o aktivitu s příslušným layoutem, která slouží uživateli pro navigaci skrz aplikaci. Při jejím designu, jakož to i u ostatních layoutů, bychom měl být dáván důraz na obecné poučky pro tvoření uživatelského rozhraní. Například kde bude uživatel hledat nejčastěji používaná tlačítka, výsledky dialogového okna, mazání a podobně…

Obrázek 25: Náhled na nabídku menu
Zdroj: Vlastní tvorba

5 Výsledky a diskuze

V průběhu tvoření aplikace vyšlo najevo několik problémů, které se podařilo ošetřit. Zejména se jednalo o způsoby, jakými uživatel může ovlivnit běh programu bez použití autorem poskytnutých prvků či funkcí. Jako nejefektivnější řešení se ukázalo využití životního cyklu aktivity a ošetřit zacházení s ní v konkrétním stádiu.

Další maličkosti vyšly najevo při testování aplikace, aby se skutečně celá aplikace uživateli jevila jako jeden kontinuální proces a tvářila se interaktivně, i toto bylo vyřešeno.

Největší otazník je zde u praktického uplatnění aplikace. Jak už autor uvedl v úvodu této práce, tato aplikace (popřípadě její rozšířená verze) v horizontu několika příštích let nemá šanci stát se oficiálním nástrojem, který by byl používán. Autor ale usuzuje, že po provedení modifikací, což zahrnuje zejména vizuální stránku aplikace a přenos žádoucích informací QR kódem, popřípadě indikaci roční doporučené dávky radiace, formátu JSON a přidání filtru je aplikace ideální k nasazení do reálného života. Jediným háčkem je pouze vygenerování QR kódu po provedení vyšetření.

PC stanice obsahující vyšetření bývají relativně izolované, a to i programy na nich. Z hlediska komfortu a jistých možných právních komplikací by bylo nutno domluvit se s vlastníky těchto programů, které mají vyšetření na starosti, o zakomponování vygenerování QR kódu po vyšetření. Toto se autorovi jeví jako velmi nenáročné a nic nebrání okamžité implementaci tohoto dodatku.

Otazník poté visí ještě nad způsobem přenosu, tím je myšleno konkrétní zařízení. Oddělení pořídí vyšetření a vygeneruje kód, který se musí dostat ke konkrétnímu pacientovi a k nikomu jinému. Nabízí se zde několik možností jako je tisk, zaslání emailem a podobně. Tyto metody s sebou však nesou zbytečné náklady (papír, inkoust, …) nebo nebezpečí úniku informací (email apod.). Jako ideální se zde jeví možnost přenosu přes tablet. Vygenerovaný kód je po zabezpečené bezdrátové síti přenesen na mobilní zařízení, ideálně tablet, které je ukázáno pacientovi. Ten si jej vyfotí a kód je posléze smazán.

Stojí za zmínku, že aplikace neobsahuje kromě hlavního vlákna žádná jiná. Toto není v tuto chvíli potřeba ale pokud by z nějakého důvodu byla například databáze velmi obsáhlá, stojí za zvážení vytvoření asynchronního vlákna, které by práci s databází obstarávalo.

Aplikace pomocí fragmentů není optimalizovaná pro zařízení s větším rozlišením, např. tablety, protože nepředpokládá jejich použití.

6 Závěr

Na začátku této bakalářské práce si autor stanovil za cíl navrhnout a vytvořit mobilní aplikaci s operačním systémem Android, což se povedlo. Aplikace je plně funkční a byla autorem otestována, to včetně zátěžového testu. Získané zkušenosti sloužily k rozšíření aplikace o bloky, které toto nežádoucí zacházení znemožňují a chyby zachytávají. Návrhu aplikace a jeho tvorbě předcházela analýza jiných aplikací, které využívají podobné principy. Zkušenosti z této fáze byly aplikovány. Dílčí cíle práce byly následující:

- Nastudování jazyka Java a nástroje Android Studio

Nastudování jazyka Java provedl autor za podpory odborné literatury a stávajících programů napsaných v tomto jazyce. Autor neměl před počátkem této práce s tímto jazykem žádné zkušenosti a nyní si dovolí tvrdit, že plynule ovládá její základy a některé pokročilejší prvky. Zároveň prozkoumal i možnosti vývojářského nástroje Android Studio a naučil se s ním zacházet.

- Najít vhodnou metodu archivace dat, tedy příslušnou databázi

Po analýze jiných aplikací, prostudování oficiální dokumentace a doporučení došlo k výběru třídy, která velmi efektivně a přehledně zpracovává databázi. Ta je již přítomná v samotném operačním systému Android a je zde tedy programátorovi z velké části usnadněno zacházení s ní.

- Zajistit čtení a vhodnou interpretaci QR kódu

Po analýze jiných aplikací a odborných fór autor našel knihovnu, která je velmi málo používaná, ale perfektně vyhovuje účelům této aplikace. Je zpřístupněna třída, která umožňuje velmi rychle a efektivně číst QR kódy a předat jejich výsledek. Pro samotnou interpretaci přečteného QR kódu byla autorem vytvořena vlastní metoda, která výsledek přetváří na objekt s konkrétními vlastnostmi.

- Zhodnocení reálného využití aplikace

Zhodnocení autor podrobně vyjadřuje v předchozí kapitole **5 – Výsledky a diskuze.** Aplikace byla úspěšně vytvořena a je plně funkční. Po minimálních zásazích je připravená být šířena dál.

7 Bibliografie

[1] Právo o přístupu k internetu. *Article19* [online]. The United Kingdom: article19, 2016 [cit. 2017-02-13]. Dostupné z: https://www.article19.org/resources.php/resource/38429/en/unhrc:-significant-resolution-reaffirming-human-rights-online-adopted

[2] Barcode. *Wikipedia* [online]. Worldwide: Wikipedia, 2015 [cit. 2017-02-17]. Dostupné z: https://en.wikipedia.org/wiki/Barcode/

[3] QR code history. *QRcode* [online]. Japan: QRcode, 2015 [cit. 2017-02-18]. Dostupné z: http://www.qrcode.com/en/history/

[4] *Denso wave* [online]. Japan: Denso wave, 2014 [cit. 2017-03-01]. Dostupné z: https://www.denso-wave.com/en/

[5] QR codes; how do they work?. *CreateQRcodes* [online]. The United States of America: createQRcodes, 2015 [cit. 2017-03-01]. Dostupné z: http://www.createqrcodes.org/how-do-qr-codes-work.html

[6] Creating QR codes. *Swetake* [online]. The United Kingdom: Swetake, 2007 [cit. 2017-03-01]. Dostupné z: http://www.swetake.com/qrcode/qr1_en.html

[7] QR code types. *QRcode* [online]. The United States of America: meetheed, 2017 [cit. 2017-02-26]. Dostupné z: http://qrcode.meetheed.com/question13.php

[8] LACKO, L'uboslav. *Vývoj aplikací pro Android*. 1. Brno: Computer Press, 2015. ISBN 978-80-251-4347-6.

[9] Dashboard. *Android Developers* [online]. The United States of America: Android Developers, 2017 [cit. 2017-01-17]. Dostupné z: https://developer.android.com/about/dashboards/index.html

[10] Backwards compatibility. *Android Developers* [online]. The United States of America: Android Developers, 2017 [cit. 2017-01-03]. Dostupné z: https://developer.android.com/training/backward-compatible-ui/index.html

[11] Multiple screens. *Android Developers* [online]. The United States of America: Android Developers, 2017 [cit. 2017-02-07]. Dostupné z: https://developer.android.com/training/multiscreen/index.html

[12] Fragment. *Android Developers* [online]. The United States of America: Android Developers, 2017 [cit. 2017-02-13]. Dostupné z: https://developer.android.com/reference/android/app/Fragment.html

[13] Relative layout. *Android Developers* [online]. The United States of America: Android Developers, 2017 [cit. 2017-02-23]. Dostupné z: https://developer.android.com/guide/topics/ui/layout/relative.html

[14] Intent. *Android Developers* [online]. The United States of America: Android Developers, 2017 [cit. 2017-02-27]. Dostupné z: https://developer.android.com/reference/android/content/Intent.html

[15] Layout. *Android Developers* [online]. The United States of America: Android Developers, 2017 [cit. 2017-03-01]. Dostupné z: https://developer.android.com/guide/topics/ui/declaring-layout.html

[16] *Android Studio* [online]. The United States of America: Android Developrts, 2017 [cit. 2017-03-02]. Dostupné z: https://developer.android.com/studio/index.html

[17] ZXing. *GitHub, Inc.* [online]. The United States of America: GitHub, Inc., 2017 [cit. 2017-03-03]. Dostupné z: https://github.com/zxing/zxing

[18] DICOM. *Nema* [online]. The united States of America: Nema, 2016 [cit. 2017-03-04]. Dostupné z: http://dicom.nema.org/standard.html

[19] Databases. *Android Developers* [online]. The United States of America: Android Developers, 2017 [cit. 2017-03-02]. Dostupné z: https://developer.android.com/training/basics/data-storage/databases.html

[20] SQLiteOpenHelper. *Android Developers* [online]. The United States of America: Android Developers, 2017 [cit. 2017-03-02]. Dostupné z: https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html

[21] Cursor. *Android Developers* [online]. The United States of America: Android Developers, 2017 [cit. 2017-03-02]. Dostupné z: https://developer.android.com/reference/android/database/Cursor.html

[22] Návrhový vzor adaptér. *Inetwork* [online]. Česká Republika: inetwork, 2014 [cit. 2017-03-08]. Dostupné z: http://www.itnetwork.cz/navrhove-vzory/adapter-wrapper-navrhovy-vzor

8 Seznam obrázků