ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra radioelektroniky

RFID automat

Bakalářská práce

Studijní program: Komunikace, multimédia a elektronika
Studijní obor: Multimediální technika

Vedoucí práce: Ing. Bc. Marek Neruda

Zdeněk Šubčík

Praha 2014

Čestné prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Datum: 23.5.2014

.....
Zdeněk Šubčík

Poděkování

Chtěl bych poděkovat všem, kdo mi pomáhali při tvorbě této práce. Jmenovitě vedoucímu práce Ing. Bc. Marku Nerudovi, dále pak rodině a přítelkyni za trpělivost.

Anotace:

Cílem této bakalářské práce je ukázat možnosti využití funkce NFC v mobilním telefonu s operačním systémem Android. Jako ukázka slouží aplikace NfcRobot, ovládající výdejový automat, kde pro přihlašování uživatelů a načítání zboží slouží NFC tagy. Dále aplikace ukazuje možnosti komunikace pomocí emulované sériové linky přes Bluetooth. Aplikace byla vytvořena ve vývojovém prostředí Eclipse.

Klíčová slova: Android, Bluetooth, NFC

Summary:

The intention of this work is to show the potential of NFC in cell phone run by operating system Android. As an example there was developed application NfcRobot, which is operating RFID robot. To sign up users or to identify goods, NFC tags are used. Furthermore, application reflects the possibilities of communication via Bluetooth by emulating serial bus.
Application was developed in development system Eclipse.

Index Terms: Android, Bluetooth, NFC

Obsah

Použité zkratky:

ACL	The Asynchronous Connection-oriented Logical
ANSI	American National Standards Institute
API	Aplication Programming Interface
CDD	Compatibility Definition Document
CDMA	Code Division Multiple Access
GPS	Global Positioning System
GSM	Groupe Spéciel Mobile (*fr)*, Globální systém pro mobilní komunikaci
HW	Hardware
ISM	Industrial, Scientific and Medical band
ISO	International Organization for Standardization
L2CAP	The Logical Link Control and Adaption Protocol
MAC	Media Access Control
OBEX	Object Exchange (Communication Protocol)
OS	Operační systém
RFCOMM	The RadioFrequency Communications
SCO	Synchronous Connection Oriented
SDK	Software Development Kit
SEQUEL	Structured English Query Language
SIG	Bluetooth Special Interest Group
SQL	Structured Query Language
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver / Transmitter
USB	Universal Serial Bus

Seznam obrázků:

Seznam tabulek:

1 Úvod

V posledních letech se mobilní telefony staly běžnou součástí každodenního života. Původní koncept mobilního telefonu jako prostého komunikátoru se postupně měnil, přidávaly se nové funkce a možnosti, například připojení k internetu pomocí bezdrátové sítě Wi-Fi nebo komunikace přes Bluetooth a NFC. Tzv. chytré mobilní telefony jsou ovládány operačním systémem. Nejrozšířenějším je OS Android od společnosti Google Inc. Android je otevřený operační systém, což mimo jiné znamená, že je možné systém rozšířit o vlastní aplikace.

Cílem této práce bylo navrhnout aplikaci na OS Android, která bude obsluhovat výdejový automat. Tato aplikace má vytvořenou databázi uživatelů, kteří se před nákupem přihlásí pomocí NFC tagu. Databázi uživatelů spravuje administrátor systému, může uživatele nově registrovat nebo naopak z databáze odstranit. Administrátor dále spravuje i databázi zboží, které je při nákupu rozpoznáváno opět pomocí NFC tagu.

Další demonstrací možností využití mobilního telefonu je připojení zámku pomocí modulu komunikujícího s telefonem přes Bluetooth, zámek se odemyká při přihlášení uživatele a zamyká po jeho odhlášení. Pro komunikaci s modulem je přes Bluetooth emulována sériová linka, bezdrátové odemykání zámku je tedy jen ukázka využité této funkce telefonu.

2 Teoretická část

2.1 Operační systém Android

Android je operační systém vyvinutý původně pro mobilní zařízení s dotykovým displejem, jako je mobilní telefon nebo tablet. Později se však rozšířil i například do set-top boxů, digitálních fotoaparátů nebo díky projektu Android-x86 [1] byl předinstalován i do méně výkonných notebooků jako alternativní operační systém k OS Windows.

Android je *open-source project*[1] [2], běží na Linuxovém jádře (*Linux kernel* [3]) a jeho zdrojové kódy jsou dostupné pod licencí *Apache Software Licence, Version 2.0* [4]).

2.1.1 Vývoj OS Android

Vývoj operačního systému Android započal roku 2003 ve společnosti Android Inc. Prvotní myšlenka byla vytvořit systém zohledňující požadavky a nastavení uživatele, přizpůsobivější než v té době se prosazující Symbian nebo Windows Mobile, jak uvedl spoluzakladatel Android Inc. Andy Rubin v interview pro Business Week [5].

I díky finančním problémům byla společnost Android Inc roku 2005 odkoupena společností Google Inc., která tímto způsobem chtěla podle Andy Rubina vyvinout vlastní operační systém a vstoupit tak na trh s mobilními telefony s operačním systémem. Původní koncept ovládaný výhradně pomocí vysouvací QWERTY klávesnice s navigačními tlačítky a bez dotykového displeje, podobný Blackberry phone OS, byl po zveřejnění první verze iPhone od společnosti Apple roku 2007 zcela přepracován.

Výsledkem několikaletého vývoje byla první verze OS Android, představená v říjnu 2008 na telefonu HTC Dream, v ČR distribuovaným pod názvem T-Mobile G1 (Obr. 1). Systém prošel od té doby značným vývojem, který je naznačen v kapitole 2.1.2.

Obr. 1 - HTC Dream, první mobilní telefon s OS Android (převzato z [23])

1 **Open-source project:** Pod veřejnou licencí je za určitých podmínek k dispozici zdrojový kód a dokumentace a je možné zdrojový kód legálně modifikovat

2.1.2 Verze systému

V současné době (květen 2014) je aktuálně nejvyšší verze Android 4.4.2 KitKat (API level 19) [6]. Na tuto verzi je v této kapitole odkazováno jako na nejnovější. Jednotlivé verze OS Android jsou odlišeny hned několika způsoby.

2.1.2.1 Kódové označení verze

Tím nejméně vypovídajícím způsobem odlišení verzí je kódové označení odkazující na názvy zákusků, přičemž každé nové označení začíná na další písmeno v abecedě. Toto označení vzniklo hlavně z toho důvodu, že mezi uživateli se komolila oficiální označení jednotlivých verzí. Společnost Google tedy hledala jednoduché a zapamatovatelné označení, z čehož následně vyplynulo i abecední řazení.

Kódové označení se nemění vždy s novou verzí, například označení „Jelly Bean“ se používá pro Android verze 4.1, 4.2 a 4.3 Jako první byl takto pojmenován Android 1.5 Cupcake (3. verze OS, proto písmeno „C“), nejnovější je Android 4.4 KitKat.

2.1.2.2 Číslo verze

Jednotlivé verze jsou nejčastěji rozlišovány pomocí číslování. První oficiálně vydaná verze je Android 1.0, nejnovější má číslo 4.4.2. Číslování se odvíjí od zásadních změn v systému. Je možné říct, že verze 1.x byly spíše zkušební. Verze 2.x byly zacílené téměř výhradně na mobilní telefony. Právě tzv. chytré mobilní telefony významně rozšířily uživatelskou základnu, díky čemuž se Android rozšířil i na tablety. Na tablety byly zaměřeny verze 3.x, které přinášely zásadní změny ve využití většího displeje tabletu oproti mobilnímu telefonu. Přinesly totiž podporu fragmentů[2], které umožňují zobrazení více paralelně řízených prvků na displej najednou. Verze 4.x pak sjednocují verze 2.x pro mobilní telefony a verze 3.x pro tablety a umožňují tak vyvíjet aplikace různě se zobrazující na menších a větších displejích. Zpětná kompatibilita aplikací je řešena pomocí *Android Support Library*[3].

2.1.2.3 Úroveň API (*API level*)

Nejdůležitější je rozlišení verzí podle úrovně API[4]. Každá API může mít jiný přístup ke knihovnám nebo podporovat jiné funkce telefonu. Každá API přináší více nebo méně důležité změny, jako například podpora vyšší verze Bluetooth (od API level 6 Android podporuje Bluetooth 2.1), podpora více-jádrových procesorů (od API level 11) apod. Vždy je ale třeba dbát na zpětnou kompatibilitu.

2.1.3 Požadavky na hardware

Pro každou nově vydanou verzi Androidu je vydán i *Android Compatibility Definition Document* (dále jen *CDD*), tedy dokument stanovující hardwarové a softwarové požadavky na zařízení, na která může být nainstalován OS Android v patřičné verzi. Tento dokument může být dále revidován.

---

2 **Fragment:** fragmenty slouží k řešení zobrazení na větších displejích, podpora více aktivních fragmentů zobrazených na jednom displeji současně. Od verze Android 3.0 (API level 11) by měly být všechny nově vyvíjené aplikace fragmentově orientované.

3 **Android Support Library:** knihovna pro zpětnou podporu aplikací psaných pro vyšší úroveň API

4 **API:** *Aplication Programming Interface*, rozhraní pro programování aplikací

S téměř každou novou verzí stoupají i požadavky na hardware. První verze cílená na více různých zařízení (všechna tato zařízení ale spadala do kategorie mobilních telefonů), tedy verze Android 1.6, měla hardwarové požadavky cílené právě jen na mobilní telefony. Android 1.6 tedy ve svém *CDD* striktně vyžadoval kameru, akcelerometr, GPS přijímač, magnetický kompas a podporu GSM a CDMA, naopak měl poměrně malé nároky na paměť. Pro srovnání jsou v tab. 1 uvedeny vybrané parametry pro Android 1.6 [7] a Android 4.4. [8].

Tab. 1 - Srovnání požadavků na parametry zařízení pro Android 1.6 a Android 4.4

	Android 1.6	Android 4.4
Displej		
Rozlišení	min. 240 x 320 pixelů, min. úhlopříčka 2,6"	min. 320 x 426 pixelů, min. 120 dpi
Poměr stran	nedefinováno	1,333 (4:3) ~ 1,86 (16:9)
Barevná hloubka	nedefinováno	min. 16-bit
Grafika		
	nedefinováno	Podpora OpenGL ES 1.0, OpenGL ES 2.0
		Doporučená podpora OpenGL ES 3.0
Paměť		
RAM	min. 32 MB	min. 340 MB, doporučeno min. 512 MB
Volná paměť pro aplikace	min. 32 MB	min. 1 GB, doporučeno min. 2 GB
Procesor		
	ARMv5	ARMv7 32-bit
		MIPS
		procesor s architekturou x86

Verze Android 2.x a 3.x už cílily i na tablety, proto Android přestal vyžadovat podporu například GSM nebo GPS, v *CDD* po tyto verze Androidu jsou výše zmíněné funkce vedeny jen jako doporučené.

Protože se Android začal postupně využívat kromě mobilních telefonů a tabletů i třeba v set-top boxech, musely být hardwarové požadavky postupně upravovány i pro tato zařízení. Oproti verzím 1.x například ztratilo smysl vyžadovat u všech zařízení kameru nebo akcelerometr. Proto jsou už téměř veškeré senzory a adaptéry v *CDD* pro Android verze 4.0 a vyšší uváděny jako doporučené, nikoli povinné.

2.1.4 Rozšíření systému aplikacemi třetích stran

Protože je Android otevřený operační systém, nabízí možnost rozšíření aplikacemi třetích stran (nezávislí vývojáři). Těm dává k dispozici SDK[5] pro vývoj a testování vlastních aplikací. Ke každé nové verzi systému (vyšší verze API) je vydán nový SDK, spolu s ukázkami zdrojového kódu a aktualizovanými knihovnami. Vývojář si může při psaní aplikace zvolit, pro kterou úroveň API aplikaci vyvíjí - vždy musí udat nejnižší požadovanou úroveň API a cílovou úroveň API. Jako cílovou úroveň API je doporučováno uvádět nejvyšší současnou úroveň. Nejnižší podporovanou úroveň API vývojář volí podle toho, pro jakou cílovou skupinu aplikaci vyvíjí a jaké nástroje chce ve své aplikaci použít.

Stále je třeba zohledňovat zastoupení jednotlivých verzí OS Android mezi uživateli. Podle oficiální statistiky [9], platné k 1. 4. 2014, používá stále téměř 18% uživatelů Android 2.3.x (API level 10), tedy verzi z roku 2011. Protože zastoupení všech nižších verzí je (podle stejné statistiky) dohromady méně než 1.3%, má v současné době

5 **SDK:** *Software Development Kit*, sada knihoven, která umožňuje tvorbu aplikací pro předem definovaný software (např. určitá verze OS) a hardware

(květen 2014) smysl volit nejnižší podporovanou úroveň API 10.

Na druhou stranu, chce-li vývojář využít možností některých vyšších API (např. Action Bar[6], přidáno v API level 11), často i v *Android Support Library* chybí zpětná podpora, a pak nezbývá než volit minimální úroveň API podle tohoto kritéria.

Google Inc. rovněž nabízí možnost distribuce takto vyvíjených aplikací. Aby Google Inc. omezil šíření podvodných nebo rovnou škodlivých aplikací mezi uživateli a zároveň, aby umožnil stahování aplikací z jediného zdroje, byl zaveden Google Market (distribuční služba, umožňující přístup k aplikacím pro OS Android a nahrávání vlastních aplikací), později sloučen se službou Google Music (on-line obchod s hudbou) a přejmenován na Google Play. Stahování aplikací je podmíněno založením uživatelského účtu, nahrávání pak založením účtu vývojářského. Google vede databázi škodlivého softwaru a umožňuje uživatelům před instalací nových aplikací do jejich zařízení odeslat tuto aplikaci na server společnosti Google a nechat zkontrolovat, zda se nejedná o software vedený v databázi jako škodlivý. I přes veškeré snahy Google Inc není tato služba stoprocentně spolehlivá a stále se vyskytuje škodlivý software distribuovaný přes Google Play. Příkladem může být aplikace DroidCleaner, tvářící se jako aplikace pro údržbu systému Android, která po nainstalování a připojení zařízení k PC infikovala operační systém PC virem typu trojský kůň [10].

2.1.5 Activity

Activity je základní stavební blok aplikací pro OS Android, specifická právě pro OS Android. Standardem pro aplikace pro jiné platformy je statická metoda *main( )*, která se volá po startu aplikace. Oproti tomu v OS Android je po spuštění aplikace vytvořena *Activity* a o její běh se stará samotný operační systém. Tato první *Activity* je spuštěna po každém startu aplikace a následně může volat další *Activity* nebo, v případě jednodušších aplikací, může řídit celý chod aplikace sama.

2.1.5.1 Životní cyklus Activity

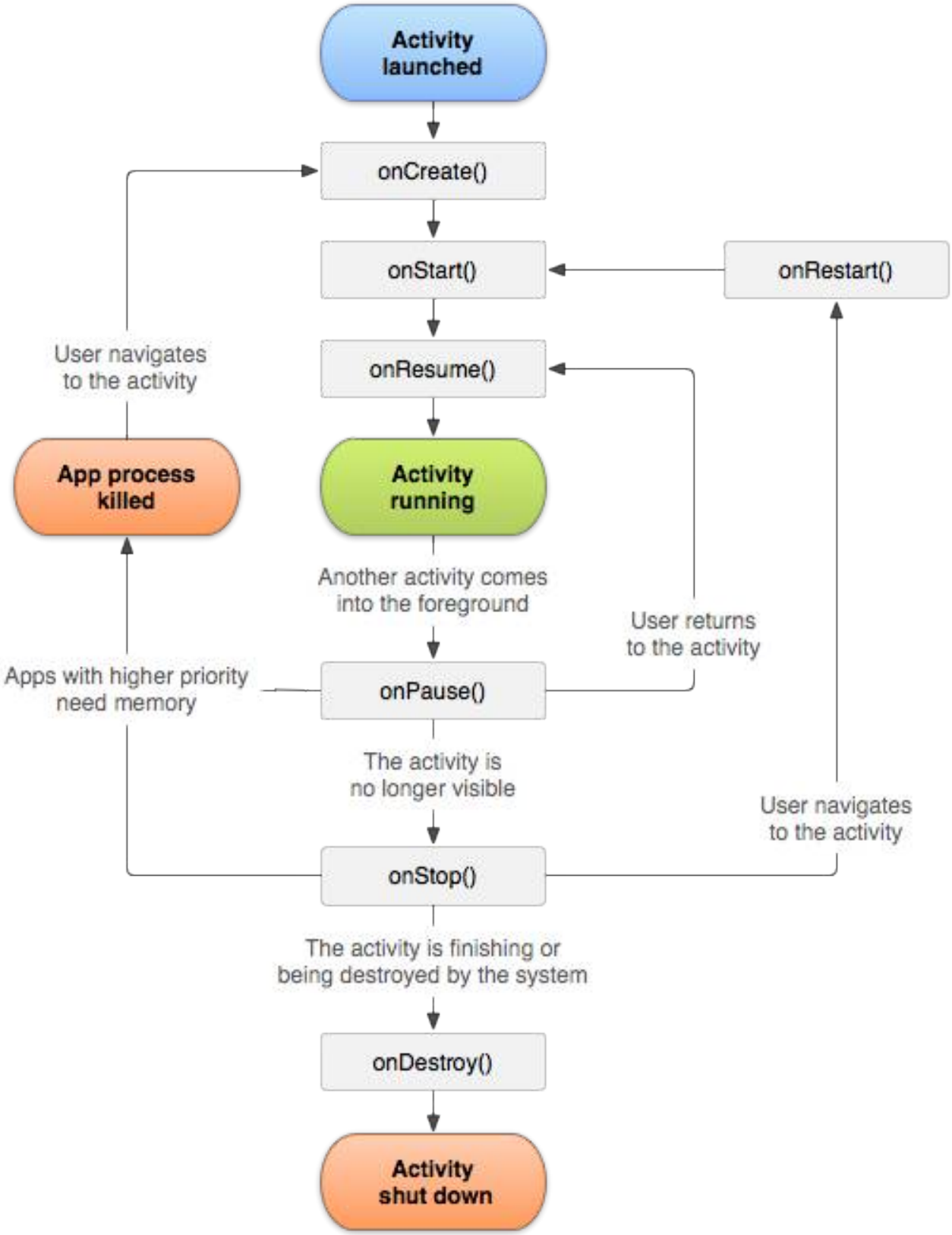
Protože aplikace může být kdykoli přerušena ať už uživatelem nebo akcí systému s vyšší prioritou (například příchozí volání na telefonu), musí mít každá *Activity* ošetřen stav pro uložení svých dat a jejich následnou obnovu při opětovném zobrazení aplikace. Dále musí mít definované chování při spouštění, při ukončování atd. Z těchto důvodů byl vytvořen přesně definovaný životní cyklus *Activity*, popsaný na obr. 2. Každá metoda v životním cyklu *Activity* má svůj smysl, např. *onCreate(Bundle savedInstanceState)* je volána vždy při vytváření *Activity* a definuje se v ní mimo jiné příslušný *layout* pro správné zobrazení *Activity* na displeji zařízení. Naopak metoda *onDestroy( )* je volána při ukončování *Activity*, v ní je třeba ošetřit zrušení přístupu k databázím nebo ukončení jakékoli jiné služby, která je využita v této *Activity* a má být spolu s ní ukončena. Běh aplikace lze rozdělit na tyto fáze [11]:

1. Aktivní stav: Lze říct, že *Activity* je v aktivním stavu, když je zobrazena v popředí, neboli je navrchu zásobníku *Activities*. V této chvíli má nejvyšší prioritu a právě ona zpracovává podněty z uživatelského rozhraní.

Během vytváření *Activity* se volají postupně v tomto pořadí metody *onCreate( )*,

6 **Action Bar:** nástrojová lišta přidaná do API level 11. Svou funkcí nahrazuje HW tlačítko „Menu“, které nadále slouží k zobrazení naposledy spuštěných a právě běžících aplikací

onStart( ) a *onResume( )*. Kterákoli z nich může být vývojářem přepsána[7], nejčastěji bývá takto přepsána metoda *onCreate( )*.



Obr. 2 - Schéma životního cyklu Activity (převzato z [25])

7 **Přepsání metody:** uvádí se anotace *@Override* před metodou, ta ukazuje na přepsání systémem předdefinované metody

2. Přerušení: Pokud získá jiná *Activity* vyšší prioritu, přejde předchozí aktivní *Activity* do stavu přerušení. Tehdy je volána metoda *onPause( )*, ve které by měla být uložena dosud neuložená data, která *Activity* přijala (např. pozice při posouvání obsahu okna, řetězce v polích pro editaci textu apod.). Data mohou být obnovena při přechodu zpět do aktivního stavu nebo uložena a zpracována při ukončení *Activity*.

Po fázi přerušení může *Activity* přejít zpět do aktivního stavu nebo být zastavena a její činnost ukončena.

3. Zastavení: V této fázi je činnost *Activity* zastavena, *Activity* běží na pozadí a pro uživatele není viditelná. Její případná činnost je definována v metodě *onStop( )*. Z této fáze bývá *Activity* většinou ukončena (volá se metoda *onDestroy( )*), může však být obnovena a získat zpět nejvyšší prioritu.

4. Obnovení: Do této fáze se dostane aplikace v případě, že ji systém chce znovu udělit nejvyšší prioritu, ale její činnost již byla zastavena. *Activity* ještě není zničena (metodou *onDestroy( )*) a tedy si pamatuje svůj stav nastavený v metodě *onCreate( )*. Protože tedy není volána metoda *onCreate( )*, kde se obnovují data uložená ve fázi přerušení,volá se místo ní metoda *onRestart( )*, která uložená data obnoví.

Důležitá je ještě skutečnost, že *Activity* ve fázi přerušení nebo zastavení může být ukončena samotným OS, pokud *Activity* s vyšší prioritou potřebuje více systémových prostředků. V takovém případě se *Activity* po opětovném zavolání bude muset znovu vytvořit (metodou *onCreate( )*).

2.1.5.2 Android Manifest a oprávnění aplikace

Při instalaci aplikací třetích stran je třeba zajistit, aby aplikace nezasahovaly do soukromí uživatele bez jeho vědomí a zároveň aby měl uživatel přehled, ke kterým systémovým prostředkům bude mít aplikace přístup. Android toto řeší pomocí oprávnění (*permissions*).

Obchod Google Play musí před instalací aplikace uživatele informovat, ke kterým prostředkům bude mít přístup. Zjistit to lze ze souboru **AndroidManifest.xml**, který musí každá aplikace mít a ve kterém jsou uloženy základní informace o této aplikaci. Zde je několik takových informací i s ukázkovými částmi zdrojového kódu:

- pro které úrovně API je aplikace určena:

```
<uses-sdk
      android:minSdkVersion="10"
      android:targetSdkVersion="19" />
```

- která oprávnění má aplikace:

```
<uses-permission android:name="android.permission.BLUETOOTH" />
```

- informace o ikoně a názvu aplikace, který se bude zobrazovat v zařízení:

```
<application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
</application>
```

- výčet všech *Activities*, které může aplikace volat (*Activities*, které nejsou v AndoridManifest.xml uvedeny, nemohou být aplikací volány):

```
<activity
      android:name="my_application.NewActivity" />
```

Dále musí Android Manifest obsahovat informace o adresářové struktuře aplikace, o *Activity*, která se jako výchozí spouští spolu s aplikací apod.

2.2 NFC

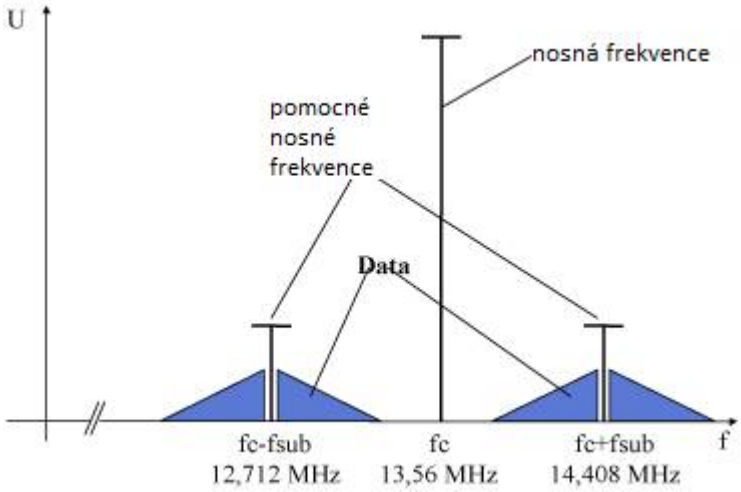
NFC (*Near Field Communication*) je technologie umožňující zabezpečenou bezdrátovou komunikaci na krátkou vzdálenost. Princip je podobný jako u RFID (*Radio Frequency Identification*), jen s tím rozdílem, že NFC má maximální dosah 4 - 20 cm.

2.2.1 Princip a standardy

NFC pracuje na frekvenci 13,56 MHz a dosahuje přenosové rychlosti až 424 kbit/s na vzdálenost do 10 cm. Pro přenos se používají dva typy modulací, BPSK (*Binary-Phase Shift Keying*) a ASK (*Amplitude Shift Keying*) s hloubkou modulace 100% nebo 10% [12].

Během vývoje technologie byly přijaty různé standardy v závislosti na výrobci zařízení. Postupně se jejich počet ustálil na třech, přičemž každé aktivní zařízení musí podporovat všechny tyto standardy. Těmito standardy jsou:

NFC-A: kompatibilní podle ISO/IEC 14443 A (MiFARE)
NFC-B: kompatibilní podle ISO/IEC 14443 B
NFC-F: kompatibilní podle ISO/IEC 18092 a JIS X6319-4 (FeliCa)



Obr. 3 - Frekvenční spektrum přenášeného signálu při použité load modulaci (převzato z [26])

Při přenosu dat mezi aktivním a pasivním zařízením dochází zároveň k přenosu energie. Pro přenos dat se proto používá *load modulace* [13], která podle specifikace ISO/IEC 14443 používá dvě pomocné nosné o frekvencích vzdálených o 848 kHz od hlavní nosné (viz obr. 3). Data pak mohou (ale nemusejí) být přenášena pouze v okolí pomocných nosných. To s sebou ale nese další komplikace ohledně rekonstrukce signálu na straně přijímače, protože signál přenášený v postranních lalocích může být i o 80 dB slabší oproti signálu na nosné frekvenci 13,56 MHz. Pro správný příjem takového signálu je nutné použít vysoce selektivní pásmové propusti.

NFC-F standard je definován jinou specifikací než NFC-A a NFC-B. Používá jiné kódování a narozdíl od standardů podle specifikace ISO/IEC 14443 nepoužívá pomocné nosné (viz tab. 2).

Tab. 2 - Srovnání parametrů jednotlivých standardů NFC (převzato z [12])

NFC standard	Typ zařízení	Modulace	Přenosová rychlost	Nosná frekvence
NFC-A	aktivní	ASK 100%	106 kbit/s	13,56 MHz
	pasivní	Load modulace (ASK)	106 kbit/s	13,56 MHz +- 848 kHz
NFC-B	aktivní	ASK 10%	106 kbit/s	13,56 MHz
	pasivní	Load modulace (BPSK)	106 kbit/s	13,56 MHz +- 848 kHz
NFC-F	aktivní	ASK 10%	212 / 242 kbit/s	13,56 MHz
	pasivní	Load modulace (ASK)	212 / 242 kbit/s	13,56 MHz

2.2.2 Režimy přenosu

Komunikace probíhá mezi dvěma zařízeními, aktivním a pasivním. Aktivní může být NFC čtečka nebo mobilní telefon s podporou NFC, s vlastním zdrojem napájení. Toto zařízení iniciuje vzájemnou komunikaci. Pasivní zařízení je NFC tag, nemá vlastní zdroj energie. Energie je mu dodávána aktivním zařízením pomocí elektromagnetických vln. Komunikovat spolu mohou i dvě aktivní zařízení, dvě pasivní ale už z principu komunikovat nemohou. Rozlišujeme tři různé režimy přenosu:

Card emulation režim: NFC zařízení se chová jako pasivní bezkontaktní čipová karta, komunikující podle standardu ISO/IEC 14443. Komunikaci inicializuje NFC čtečka, má tedy úlohu aktivního zařízení.

Peer-to-peer režim: Jedná se o obousměrnou komunikaci dvou nebo více aktivních zařízení. Protože zde není žádné pasivní zařízení, není nutné jej napájet a proto tento režim spotřebuje nejméně energie. Dosahuje se zde nejvyšších přenosových rychlostí, a to až 424 kbit/s.

Reader/writer režim: Zde se jedná o komunikaci mezi aktivním zařízením a pasivním NFC tagem. Aktivní zařízení iniciuje spojení a napájí NFC tag, dosahuje se přenosové rychlosti do 106 kbit/s. Právě tento režim podporuje všechny tři standardy NFC-A, NFC-B a NFC-F. Po detekci NFC tagu nejprve aktivní zařízení zkusí postupně získat odezvu na každý z těchto standardů pro přenos a podle odezvy nastaví odpovídající komunikační standard.

2.2.3 NFC tagy

NFC tagy jsou pasivní zařízení, komunikující s aktivními NFC zařízeními (mobilní telefon s NFC, NFC čtečka). Jedná se vlastně o malé paměťové úložiště, které po navázání komunikace může do aktivního zařízení přenést malé množství uložených dat. Byly definovány 4 typy NFC tagů, jejichž parametry jsou uvedeny v tabulce 3.

Tab. 3 - Parametry jednotlivých standardizovaných typů NFC tagů (převzato z [12])

	ISO/IEC standard	Kompatibilní produkt	Přenosová rychlost	Paměť	Anti-kolize
Typ 1	14443 A	Innovision Topaz	106 kbit/s	96 B ~ 2 kB	Ne
Typ 2	14443 A	NXP MIFARE	106 kbit/s	48 B ~ 2 kB	Ano
Typ 3	JIS X6319-4	Sony FeliCa	212 / 424 kbit/s	< 1 MB	Ano
Typ 4	14443 A / B	NXP DESFire, JCOP, ...	106 / 212 / 424 kbit/s	< 32 kB	Ano

2.2.4 Využití technologie NFC

NFC lze využít k bezkontaktním platbám, jako identifikaci, pro docházkový systém apod. V dnešní době se prosazuje začlenění NFC do mobilních telefonů, které má u sebe dnes téměř každý. Takový telefon je pak možné využít jako bezkontaktní platební kartu. V případě odcizení lze postupovat stejně jako při ztrátě platební karty, tedy kontaktovat banku a platby z tohoto telefonu zablokovat. Výhodou je např. on-line prodloužení platnosti karty (platebních transakcí z telefonu), není nutné se osobně dostavit do banky.

Dalším využitím je nahrazení klasických klíčů, NFC tag lze využít pro odemčení domovních dveří nebo dveří automobilu. Právě u automobilu lze NFC tag využít i pro uložení individuálního nastavení klimatizace, zrcátek, rádia apod., např. u firemních automobilů, které využívá více lidí.

Dalším využitím může být nákup jízdenek v městské hromadné dopravě. V současné době mnoho dopravců provozuje službu, kde je možné zakoupit jízdenku odesláním SMS ve správném tvaru. Jednodušším řešením by byl terminál se čtečkou NFC a nákupem jízdenky prostým přiložením telefonu a potvrzením.

Praktickým a již dnes používaným využitím NFC je konfigurace Wi-Fi sítě nebo párování zařízení pro Bluetooth komunikaci [14].

2.3 Bluetooth

2.3.1 Vývoj a specifikace

Technologie Bluetooth je definovaná specifikací 802.15.1 [15]. Vznikla již roku 1994 u výrobce mobilních telefonů Ericsson jako výsledek snahy propojit mobilní telefony bezdrátovou, energeticky nenáročnou (*low-power*) a levnou technologií. Roku 1998 založil Ericsson spolu se společnostmi IBM, Intel, Nokia a Toshiba organizaci SIG (*Bluetooth Special Interest Group*) pro vývoj Bluetooth technologie. O rok později byla vydána první specifikace Bluetooth 1.0. V současné době (květen 2014) je nejnovější specifikací Bluetooth 4.0 z roku 2010.

Bluetooth specifikace je vlastně spojení několika různých specifikací. Tyto specifikace jsou celkem čtyři [16]:

Bluetooth Core Specification: definuje vlastní Bluetooth architekturu a obecné podmínky užití.

Bluetooth Transport Specification: popisuje různé způsoby komunikace mezi dvěma zařízeními (např. přes UART nebo USB).

Bluetooth Protocol Specification: popisuje komunikační protokoly vyšší úrovně (např. RFCOMM nebo OBEX).

Bluetooth Profile Specification: popisuje a definuje transportní protokoly.

2.3.2 Parametry Bluetooth

Technologie Bluetooth byla vyvinuta za účelem bezdrátového přenosu dat na krátké vzdálenosti. Protože jsou data vysílána na vzdálenosti řádově desítek metrů, stačí nízký výstupní výkon vysílače (desítky mW), a komunikace přes Bluetooth je proto vhodná pro přenosná zařízení napájená vlastní baterií (např. mobilní telefony). Pracovní frekvence Bluetooth je 2,4 GHz, spadá do mezinárodně vyhrazeného ISM frekvenčního pásma. ISM (*Idustrial, Scientific and Medical*) frekvenční pásmo slouží pro průmyslový, vědecký a zdravotnický obor, schválená zařízení na ní lze provozovat bez licenčních poplatků. ISM pásma ale nemají garanci proti rušení, proto je Bluetooth komunikační protokol velmi robustní z hlediska interference s ostatními zařízeními na stejné pracovní frekvenci.

K identifikaci Bluetooth zařízení slouží 48-bitová jedinečná MAC adresa. Bluetooth připojení funguje na principu *master-slave*, přičemž k zařízení typu *master* může teoreticky současně aktivně komunikovat až se 7 zařízeními (omezeno 3-bitovým identifikátorem) a pasivně může být připojeno až 255 zařízení.

Tab. 4 - Přehled základních parametrů Bluetooth komunikace (převzato z [16])

Pracovní frekvence	2,4 GHz
Maximální přenosová rychlost	3 Mbps (spec. 2.0) 24 Mbps (spec. 3.0 a 4.0)
Výstupní výkon vysílače	1 mW (class 1) 2,5 mW (class 2) 100 mW (class 3)
Dosah ve volném prostředí	1 m (class 1) 10m (class 2) 100 m (class 3)
Maximum připojených zařízení	7 aktivních, 255 pasivních
Adresování	48-bit MAC

Bluetooth zařízení se dělí podle specifikací, novější specifikace se zaměřují na vyšší dosah při nižším výstupním výkonu vysílače a na zabezpečení přenosu pomocí šifrování. Dále se zařízení dělí na třídy (class 1, class 2 a class 3), každá třída má určitý maximální povolený výstupní výkon a s tím souvisí i maximální dosah. Přehled parametrů Bluetooth je shrnut v tab. 4.

2.3.3 Komunikační protokoly

Velmi často je Bluetooth využíváno k nahrazení kabelového spojení s příslušenstvím. Může tak sloužit k připojení bezdrátové náhlavní soupravy (hands-free) k mobilnímu telefonu, připojení bezdrátové klávesnice k počítači apod. Dalším využitím je bezdrátová výměna dat, ať už jde např. o sdílení multimediálního obsahu nebo vytvoření sériové linky a datovou komunikaci.

Různá využití Bluetooth umožňují Bluetooth protokoly. Těch je specifikováno velké množství, zde je popis jen těch nejčastěji používaných [17].

RFCOMM: *The Radio Frequency Communications* je tzv. spolehlivý (*reliable*) protokol. To znamená, že se ujistí, že všechny odeslané pakety byly přijaty a v případě nespolehlivého přenosu ukončí spojení. Data odesílá jako *stream* - odesílá pakety, které nemají pevně danou délku. Podle Bluetooth specifikací byl tento protokol vytvořen pro emulaci sériové linky RS-232 přes protokol L2CAP. Ačkoli je protokol velmi podobný TCP, narozdíl od tohoto protokolu RFCOMM nabízí jen 30 volných portů. Jedná se o nejčastěji podporovaný Bluetooth komunikační protokol, v minulosti často i jediný podporovaný, např. pro Windows XP Bluetooth API.

L2CAP: *The Logical Link Control and Adaption Protocol* je paketově orientovaný protokol, umožňuje nastavení úrovně spolehlivosti. Výchozí délka paketu je 672 bytů, ale po navázání spojení stabilního spojení je možné odesílat pakety o délce až 65353 bytů. Pakety musí být přijaty vždy ve stejném pořadí, jako byly odeslány.

Protokol umožňuje nastavení tří úrovní spolehlivosti. Pokud není odeslaný paket přijat, může být paket odeslán znovu. To závisí na nastavené úrovni spolehlivosti:

- nikdy neodesílat paket znovu (metoda doručení paketů *best-effort* - snaží se co nejrychleji a nejefektivněji pakety odeslat, ale nekontroluje, zda pakety druhá strana přijme)
- vždy odeslat paket znovu, dokud není paket přijat nebo není ukončeno spojení (výchozí stav, *reliable*)
- pokud nebyl paket do daného časového limitu přijat, je přesunut na konec fronty dat čekajících na odeslání (stále ale není klasifikováno jako *reliable*)

Nastavování úrovně s sebou nese určitá rizika. Nastavení spolehlivosti pro určité zařízení bývá uloženo a je použito i při příštím připojení tohoto zařízení, nenastaví se výchozí nastavení. Ostatní připojená zařízení to ale neovlivní. Dalším rizikem je fakt, že L2CAP slouží jako transportní protokol i pro RFCOMM, a tedy změna úrovně spolehlivosti na této vrstvě ovlivní i spolehlivost RFCOMM protokolu.

ACL: *The Asynchronous Connection-oriented Logical* transportní protokol není sám o sobě příliš používaný, ale zapouzdřuje veškerá L2CAP spojení. Tím pádem ho využívá i RFCOMM, který používá L2CAP jako transportní vrstvu. Trochu tedy připomíná

protokol IP, který nebývá používán přímo pro přenos dat, ale zapouzdřuje protokoly vyšších úrovní.

SCO: *Synchronous Connection-Oriented* transportní protokol je paketově orientovaný protokol s metodou doručení paketů *best-effort*. Byl vytvořen pro přenos audio signálu s datovým tokem přesně 64 kb/s, tedy pro přenos telefonního hovoru. Tento protokol není použitelný pro transport kvalitního audio signálu a například pro poslech hudby v bezdrátových sluchátkách. K tomu se používá protokol L2CAP.

Ačkoli nedoručené pakety nejsou znovu odeslány, je tu zaručena určitá kvalita přenosu. SCO připojení vždy zaručuje přenosovou rychlost 64 kb/s a v případě více Bluetooth připojení z jednoho zařízení bude mít SCO nejvyšší prioritu. I proto nemůže mít žádné zařízení více než 3 aktivní SCO připojení, navíc mezi dvěma zařízeními může být nejvýše jedno SCO připojení.

2.4 SQL databáze

2.4.1 Jazyk SQL

Jazyk SQL (*Structured Query Language*) byl vyvinut ke konci sedmdesátých let 20. století firmou IBM, ke které se postupně přidávaly další firmy, např. Relational Software (dnešní Oracle Corporation). Jejich cílem bylo vyvinout takový jazyk, jehož příkazy by byly syntakticky podobné běžnému jazyku (angličtině). Tak vznikl jazyk SEQUEL (*Structured English Query Language*), později přejmenován na SQL. Protože se jazyk velice rychle rozšířil a prosadil, bylo nutné jej standardizovat. Roku 1986 byl přijat ANSI[8] standard SQL-86 a o rok později i ISO[9] standard. Původní standard prošel zatím dvěma revizemi, v současné době (květen 2014) nejnovější standard je z roku 1999, nazýván SQL3.

Název jazyka SQL bývá do češtiny překládána jako strukturovaný dotazový jazyk. SQL je neprocedurální jazyk. Původně byl jazyk vyvinut pro ovládání produktu DB2 společnosti IBM, tedy k ovládání relačního databázového řídicího systému, později se začal využívat i pro jiné účely a na jiných platformách [18].

2.4.2 Relační databáze

Relační databázový systém se od obecného databázového systému liší tím, že používá množinově orientovaný jazyk (např. SQL). Množinově orientovaný jazyk nepracuje s daty jednotlivě, ale po skupinách, množinách. Pro ukládání dat se většinou používá tabulka, kde řádky reprezentují jednotlivé záznamy (entity) a sloupce pak definují jejich vlastnosti (atributy). Atributy mají pevně daný datový typ můžou mít definovanou doménu, tedy množinu přípustných hodnot.

Pro relační databázi jsou důležité vazby mezi jednotlivými záznamy. Ty lze rozdělit na tři základní typy [19]:

- **1:1** - Tento typ vazby se používá, pokud hledáme vazbu mezi atributy, které jsou pro každou entitu jedinečné. Příkladem může být číslo občanského průkazu v databázi zaměstnanců firmy, každý zaměstnanec bude mít právě jedno číslo průkazu a žádní dva nebudou mít toto číslo stejné.
- **1:N** - Tento typ vazby se používá, pokud jedna entita může mít více různých atributů, ale žádný z atributů není společný pro různé entity. Příklad: jeden člověk může vlastnit více kreditních karet, ale žádnou z těchto karet nemůže vlastní zároveň i někdo jiný.
- **M:N** - Zde neplatí žádná z předchozích omezení. Příkladem může být zápis studentů do různých předmětů, kdy více studentů může mít zapsaný stejný předmět a zároveň může mít každý student zapsané libovolné předměty.

Vyhledávání v databázi funguje na principu klíčů. Klíče se rozdělují do tří hlavních skupin podle svého využití:

- **Primární klíč:** Primární klíč je jednoznačný identifikátor záznamu. Může jím být jediný atribut (ten pak musí mít vazbu 1:1) nebo více různých atributů, které společně jednoznačně identifikují záznam. Hodnota ve sloupci, který je primárním klíčem,

8 **ANSI:** *American National Standards Institute*, nezisková organizace pro vytváření standardů v USA. Je členem ISO.

9 **ISO:** *International Organization for Standardization*, mezinárodní organizace pro normalizaci

nesmí být *null*, tedy prázdná, nedefinovaná hodnota. Často se používá umělý primární klíč v podobě sloupce s jedinečnými identifikátory. Ty mají často podobu přirozeného čísla, kdy každý nový záznam získá identifikátor o jedna vyšší než nejvyšší dosud uložený záznam. Další možností je vytvořit identifikátor z času přidání záznamu do databáze (je třeba rozlišovat dostatečně krátké časové intervaly, aby nemohlo být ve stejný časový interval přidáno více záznamů). V tomto případě slouží identifikátor i k určení času přidání záznamu do databáze, není tedy pouze umělým primárním klíčem.

- **Kandidátní klíč:** Jedná se o atribut nebo skupinu atributů jednoznačně definujících záznam v databázi. Některé kandidátní klíče se stanou primárními klíči, zbytek slouží jako alternativní klíče.
- **Cizí klíč:** Cizí klíče jsou skupiny polí, slouží k vyjádření vztahů (relací) mezi jednotlivými záznamy v databázi.

3 Praktická část

3.1 Obecný popis a řešení zadání

Výdejový automat je řízen aplikací nainstalovanou na mobilním telefonu LG-E610v s operačním systémem Android, ver. 4.1.2. (API level 16). K načítání výrobků a přihlašování uživatelů se používají NFC tagy. Zboží bude uskladněno v prostoru uzamčeném elektromagnetickým zámkem, který je ovládán Bluetooth modulem RN-42, připojeným k telefonu.

Mobilní telefon má jen jednu NFC anténu, většinou pevně vlepenou do zadního krytu (jako i v tomto případě), a proto musí být v tomto demu společná pro přihlašování uživatelů i načítání zboží.

K řízení a komunikaci s uživatelem slouží aplikace NfcRobot, obecně popsaná a kapitole 3.1.1. Podrobně rozepsaný princip i s vyjmutými částmi zdrojového kódu je v kapitole 3.3.

Připojení zámku k telefonu je řešeno přes Bluetooth. Je to z toho důvodu, že mobilní telefon, který byl zapůjčen k testování aplikace, nemá USB OnTheGo port. To znamená, že přes USB umožňuje komunikovat pouze jako ovládané zařízení (*slave*), neumožňuje ale připojení externího zařízení (v roli *host*). Dalším rozhraním by mohlo být NFC, ale to už je jednak použito pro načítání zboží a přihlašování uživatelů, za druhé má příliš malý dosah. Řešení přes bezdrátovou síť Wi-Fi by bylo příliš energeticky náročné, zbývá tedy možnost propojení pomocí Bluetooth. Jako rozhraní mezi mobilním telefonem a zámkem je použit Bluetooth modul RN-42, který na jednom ze svých výstupů spíná zámek přes tranzistor. Podrobnější popis se nachází v kapitole 3.4.

3.1.1 Princip aplikace

Po svém spuštění aplikace ověřuje, zda zařízení podporuje funkce NFC a Bluetooth. Pokud ano, zapínají se Bluetooth a NFC adaptéry a vyhledává se Bluetooth modul obsluhující zámek. Po navázání komunikace s modulem se zobrazí výchozí obrazovka aplikace a čeká se na přihlášení uživatele pomocí NFC tagu.

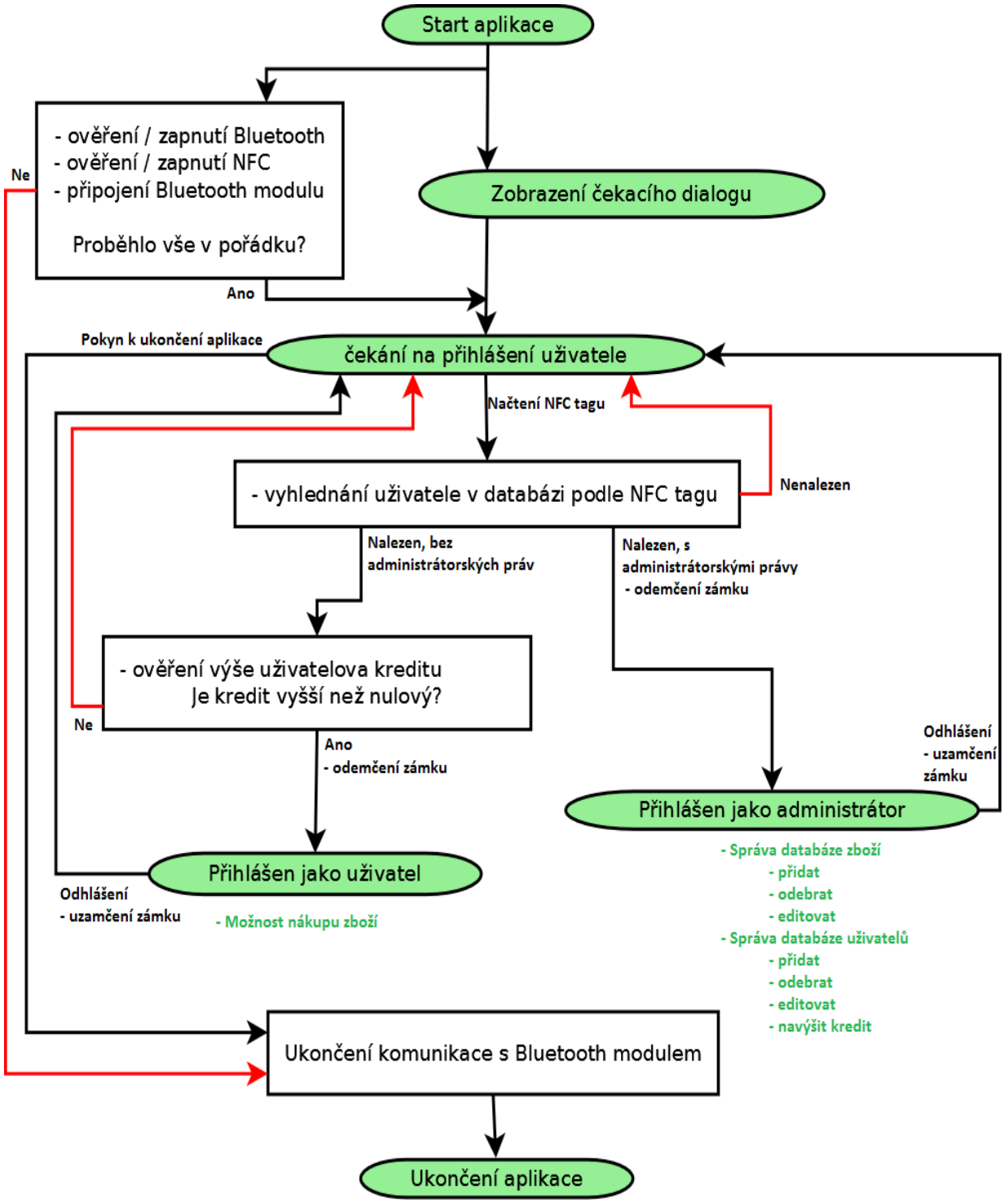
Aplikace rozlišuje dvě úrovně oprávnění, jak je naznačeno na obr. 4. První z nich je úroveň administrátora, druhá úroveň uživatele. V případě přihlášení administrátora je aplikace přesměrována do administrátorské části aplikace, kde lze spravovat databázi zboží a uživatelů. V případě přihlášení běžného uživatele je aplikace přesměrována do uživatelské části, která nabízí možnost nákupu zboží. V obou případech se po přihlášení odemyká zámek a po odhlášení se zámek opět uzamkne. Po odemčení zámku má uživatel přístup ke zboží, opatřenému NFC tagem. Při vyjmutí zboží z úložného prostoru se načte NFC tag zboží (v případě nakupujícího běžného uživatele). Stejně tak se NFC tag načte při vložení nového zboží do úložného prostoru (v případě administrátora přidávajícího nové zboží do databáze).

Aplikace funguje v *landscape* i *portrait* módu, tedy na výšku i na šířku displeje a je možné kdykoli za běhu aplikace změnit orientaci displeje bez nebezpečí nestandardního chování aplikace nebo ztráty neuložených dat (například při vyplňování údajů o novém zboží). Při změně orientace displeje nebo přerušení například příchozím hovorem je totiž právě aktivní *Activity* ukončena, a poté, co by se měla znovu na displeji objevit, se celá znovu spouští. Během toho by se ztratila data například v *EditText* polích nebo právě zobrazené dialogy. Aby ke ztrátě dat nedošlo, je třeba vždy před ukončením *Activity* data

nějakým způsobem uložit a při obnovení *Activity* obnovit i data.

To je ošetřeno pomocí metody *onSaveInstanceState(Bundle outState)*, která se spouští při ukončení *Activity* a například metodou *outState.putString(String name, String data)* lze uložit řetězec, který se při startu může opět obnovit z dat uložených v argumentu metody *onCreate(Bundle savedInstanceState)*.

Schéma celé aplikace je na obr. 4.



Obr. 4 - Naznačení běhu aplikace NfcRobot

3.1.2 Obsluha NFC adaptéru

Po zapnutí NFC adaptéru systém automaticky čte a zobrazuje načtené NFC tagy, a to i v případě, že je spuštěna a je právě aktivní aplikace, která NFC adaptér využívá. Aby se tomuto samovolnému čtení z NFC tagů zabránilo, je nutné i do té *Activity*, která s NFC adaptérem přímo nepracuje, implementovat metody pro jeho obsluhu a potlačit tak nežádoucí intervenci systému. Těmito metodami se přepíše výchozí nastavení obsluhy NFC adaptéru a *Activity* tak přebírá kontrolu nad čtením a zpracováním NFC tagů.

Metody pro obsluhu NFC adaptéru jsou rozšířením metod popsaných na oficiálních internetových stránkách pro vývojáře [20]. Celý kód, který obsahují všechny *Activities* nevyužívající NFC adaptér pak vypadá takto:

```
@Override
 protected void onResume() {
    super.onResume();
    PendingIntent pendingIntent = PendingIntent
          .getActivity(getApplicationContext(), 0, new
          Intent(this,getClass())
          .addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
    NfcAdapter nfcAdapter = NfcAdapter.getDefaultAdapter(this);
    nfcAdapter.enableForegroundDispatch(this, pendingIntent, null, null);
}
@Override
 protected void onNewIntent(Intent intent) {
    Toast.makeText(this, "Now you are not allowed to read from NFC tag!" ,
                Toast.LENGTH_SHORT).show();
    super.onNewIntent(intent);
}
```

3.1.3 Řešení administrátora při prvním spuštění

Protože tato aplikace nemůže správně fungovat, pokud v databázi uživatelů není alespoň jeden administrátor, je třeba zajistit, aby vždy alespoň jeden existoval. Bez administrátora není možné spravovat databázi zboží a vytvářet nové uživatelské účty - jen administrátor může zaregistrovat dalšího administrátora.

Při prvním spuštění aplikace nebo po smazání veškerých dat aplikace v nastavení systému Android je databáze uživatelů prázdná. Tehdy se aplikace po spuštění a zapnutí NFC a Bluetooth adaptéru přesměruje do *UsersActivity* namísto *SignUpActivity*. Je to nutný krok, protože ze *SignUpActivity* lze pokračovat pouze přihlášením uživatele, což není s prázdnou databází uživatelů možné.

V *UsersActivity* se čeká na vytvoření uživatelského účtu. Při ukončení této *Activity* se ověří, zda byl účet vytvořen, a pokud ano, aplikace se vrátí do *SignUpActivity* a dál běží tak, jak je popsáno v kapitole 3.1.1. Pokud účet vytvořen nebyl, zobrazí se varovný dialog se dvěma možnostmi:

- **Create account**: Uživatel je přesměrován do *GetTagActivity* a je mu umožněno účet založit.
- **No, exit**: Aplikace bude ukončena. Nebyl vytvořen administrátorský účet, proto se bude při příštím startu aplikace situace opakovat.

Takto vytvořený administrátorský účet bude mít jedinečný identifikátor *_id* roven 1, protože se jedná o první záznam v databázi uživatelů. Aby se zajistilo, že někdy

v budoucnu nebudou smazány všechny administrátorské účty a ponechány pouze ty bez administrátorských práv, nelze tento účet z databáze odstranit ani ho editovat.

3.1.4 Použité příkazy SQLite databáze

Pro uložení zboží i uživatelských účtů využívá aplikace SQLite databázi. Její výhodou je poměrně snadná obsluha a jednoduchost. O komunikaci s databázemi se starají třídy *Database.java* a *Users.java*. Obě pracují na stejném principu, mají metody pro přidání položky do databáze, odstranění položky z databáze, vybrání jedné, více nebo všech položek z databáze a pak další potřebné metody.

Následuje popis příkazů SQLite databáze, použitých ve třídách *Database.java* a *Users.java*:

insert( ):

Metody pro přidání položky využívají příkaz:

```
database_name.insert(String name, String nullColumnHack, ContentValues
      values)
```

Ta vytvoří nový řádek v databázi, do jednotlivých sloupců vepíše hodnoty definované v proměnné *values* a vrátí identifikátor *_id* nově vytvořeného řádku.

delete( ):

Metody pro odstranění položek využívají příkaz:

```
database_name.delete(String name, String whereClause, String[] whereArgs)
```

Ta vyhledá řádky splňující podmínky definované ve *whereClause* (ve kterém sloupci) a *whereArgs* (argumenty pro porovnání s obsahem v daném sloupci) a vymaže je z databáze. Pokud chceme vymazat jedinou položku, je nejjednodušší dosadit za argument *whereClause = COLUMN_ID + " = ?"* a do argumentu *whereArgs* dosadit *_id* dané položky. Tím se zajistí, že se v databázi vybere jediná položka, a to právě ta s požadovaným *_id*.

Pokud se za argument *whereClause* dosadí *null*, vyberou se všechny řádky databáze. To využívá například metoda *getGoods( )* ve třídě *Database.java*.

update( ):

Metody pro změnu dat u již existujících záznamů v databázi využívají příkaz:

```
database_name.update(String name, ContentValues values, String
      whereClause, String[] whereArgs)
```

Podmínky *whereClause* a *whereArgs* jsou stejné jako u metody *delete( )*, data jsou uložena v proměnné *values* stejně jako u metody *insert( )*.

query( ):

Poslední skupinou jsou metody, vracející jeden nebo více řádků databáze v proměnné typu *Cursor*. Ty využívají příkaz:

```
database_name.query(String name, String columns, String selection,
      String[] selectionArgs, String groupBy, String having,
      String orderBy)
```

3.2 Použité přístroje

• Mobilní telefon **LG-E610v**

• Bluetooth modul **RN-42**

Obr. 5 - Mobilní telefon LG-E610v (převzato z [27])

Obr. 6 - Bluetooth modul RN-42 (převzato z [28])

Tab. 5 - Vybrané parametry mobilního telefonu LG-E610v (převzato z [21])

Operační systém	Android 4.1.2 (ICS), API 16
Velikost displeje	4"
Rozlišení displeje	480x320 pixelů
Počet barev	262 000 barev
Paměť RAM/ Frekvence CPU/GPU	390 MB RAM / 800Mhz (ARMv7 rev 1) CPU / Adreno 200
Interní paměť	2,65 GB
Bluetooth	Ano, v 3.0 (HS) EDR, A2DP, HSP, HFP
NFC	Ano, NFC Beam
Baterie	Li-ion 1500mAh

Tab. 6 - Vybrané parametry Bluetooth modulu RN-42 (převzato z [22])

Pracovní frekvence	2,402 ~ 2,480 MHz
Modulace	FHSS / GFSK
Dosah	až 20 m
Výstupní výkon vysílače	4 dBm
Citlivost přijímače	-80 dBm
Pracovní napětí	3,3 V; povoleno 3,0 ~ 3,6 V
Komunikační rozhraní	Bluetooth, UART

3.3 Popis jednotlivých tříd

3.3.1 WelcomeActivity.java

Tato *Activity* je spuštěna při startu aplikace. Jejím úkolem je ověřit, zda zařízení, na kterém je aplikace spuštěna, podporuje funkce Bluetooth a NFC. Dále se postará, aby byly adaptéry obou funkcí zapnuty.

Jak je zřejmé z obr. 4 a popisu v kapitole 3.1.1, rozdělí se běh aplikace na dvě vlákna. První vlákno se stará o zobrazení informací na displej a může zajišťovat komunikaci s uživatelem, zatímco druhé vlákno vykonává nějakou déle trvající činnost na pozadí. Takovou činností je v tomto případě navazování komunikace s Bluetooth modulem, což trvá řádově několik sekund. Důležité je, aby ostatní vlákna aplikace čekala až do navázání komunikace, jinak hrozí pád aplikace nebo selhání připojení.

První vlákno zobrazí dialog (viz obr. 7), který uživatele informuje, že aplikace navazuje spojení s Bluetooth modulem obsluhujícím zámek. Druhé vlákno, běžící na pozadí, vyhledá modul, připojí se k němu a uvede modul do *command módu*, tedy stavu, ve kterém modul přijímá příkazy posílané po sériové lince přes Bluetooth.

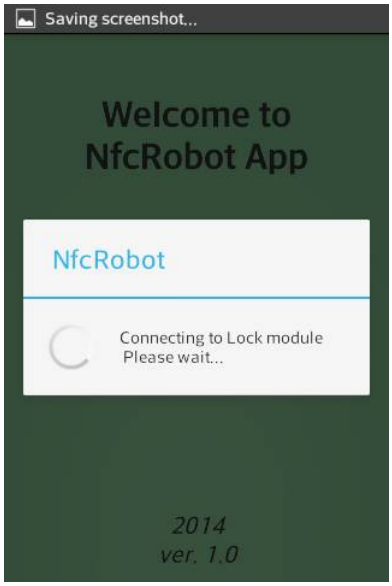
Celý tento proces probíhá ve třídě *MyTask*, jejíž konstruktor *MyTask(ProgressDialog dialg)* definuje v metodě *onPreExecute( )* procesy běžící v popředí (zde zobrazení dialogu) a v metodě *doInBackground(void)* procesy běžící na pozadí. Zde je to metoda *doOnStart( )*, která volá metodu *LockEnable( )* pro připojení modulu, podrobněji popsanou v kapitole 3.4.1. Celá třída *MyTask* vypadá takto:

```
public class MyTask extends AsyncTask<Void, Void, Void> {
        //consructor
    public MyTask(ProgressDialog dialog) {
        this.dialog = dialog;           //define dialog
    }
        //do on main Thread, wait for 'return' from background
    public void onPreExecute() {
        dialog.show();
    }
        //do in background
    public Void doInBackground(Void... unused) {
        doOnStart();
        return null;
    }
        //do after background process is done
    public void onPostExecute(Void unused) {
        dialog.dismiss();               //dismiss dialog
            //if lock is connected
        if (lock_ok) {
            startNextActivity();        //start SignUpActivity
        }
        else {
            Toast.makeText(getApplicationContext(), "Lock module device
                is not available", Toast.LENGTH_LONG).show();
            finish();                   //exit aplication
      }
    }
    ProgressDialog dialog;
}
```

Po návratu do hlavního vlákna příkazem *return* zmizí dialog a vyhodnotí se podmínka, zda byla komunikace navázána. Pokud ne, aplikace vypíše hlášku a ukončí se. Pokud ano, aplikace zavolá *SignUpActivity* pomocí metody *startActivityForResult(intent i, int request)*. Tímto způsobem jsou i později volány všechny *Activities*. Výhodou této metody je, že při ukončení volané *Activity* je možné nastavit jí parametr *RESULT*, který po návratu do mateřské *Activity* informuje, zda byl účel, pro který byla nová *Activity* zavolána, splněn, či nikoli. K vyhodnocení této návratové hodnoty je implementována metoda *onActivityResult(int RequestCode, int ResultCode, Intent data)*. Protože je tato metoda společná pro veškeré *Activities* volané metodou *startActivityForResult( )*, parametrem *RequestCode* je možné rozhodnout, která *Activity* metodu spustila. *ResultCode* nese informaci o úspěšnosti *Activity*. Parametr *data* může nést další data, která je třeba předat z volané Activity.

Zde je příklad užití metody *onActivityResult( )*, kde je vyhodnocen parametr *Result* při spuštění metody ukončením *SignUpActivity*. Pokud vše proběhlo v pořádku, byl nasteven *Result* na hodnotu *RESULT_OK* a *WelcomeActivity* (a s ní celá aplikace) bude ukončena. Ještě před ukončením aplikace je ale třeba ukončit komunikaci s Bluetooth modulem, proto se místo ukončení *Activity* metodou *finish( )* volá *MyTaskExit(ProgressDialog dialog).execute( )*. Princip je stejný, jako při navazování komunikace.

```
@Override
  protected void onActivityResult(int requestCode, int resultCode, Intent
    data) {
          //answer to REQUEST SIGN UP
    if(requestCode == REQUEST_SIGN_UP && resultCode == RESULT_OK) {
                //set dialog2 params
          dialog2 = new ProgressDialog(WelcomeActivity.this);
          dialog2.setMessage("Disconnecting from Lock module\n Please
                wait...");
          new MyTaskExit(dialog2).execute();
    }
    super.onActivityResult(requestCode, resultCode, data);
 }
```



Obr. 7 - Dialog ve WelcomeActivity

Obr. 8 - Výchozí obrazovka pro přihlášení

Po úspěšném ukončení komunikace s modulem je *WelcomeActivity* i celá aplikace ukončena.

3.3.2 SignUpActivity.java

Tato *Activity* je spuštěna po zapnutí NFC a Bluetooth adaptéru a připojení zámku ve *WelcomeActivity*.

Zobrazí se výchozí obrazovka pro přihlášení (viz obr. 8) a je spuštěn *event listener*[10] pro zachycení NFC tagu, pomocí kterého se může uživatel přihlásit. Ve spodní části obrazovky je umístěno tlačítko pro ukončení aplikace. Po kliknutí na něj se zavolá metoda *showExitDialog( )*, která zobrazí potvrzovací dialog. Ten uživateli umožní návrat v případě, že na tlačítko klikl omylem. Funkce hardwarového tlačítka pro návrat je potlačena.

Jak již bylo popsáno v kapitole 3.1.1, aplikace umožňuje přihlášení ve dvou režimech. Prvním z nich je přihlášení běžného uživatele, který je po úspěšné identifikaci NFC tagu a ověření výše kreditu přesměrován na *ShopActivity*. Ověřením hodnoty uživatelova kreditu se zabrání přihlášení uživatele, jehož kredit je nulový nebo záporný. V takovém případě se vypíše varovná hláška a uživatel není přihlášen, zámek se neodemkne.

Druhým režimem je přihlášení administrátora, který je přesměrován na *AdminActivity*, odkud může spravovat databázi výrobků a uživatelů. Při přihlášení v prvním i druhém režimu se vždy odemyká zámek.

Bluetooth modul, obsluhující zámek, byl již ve *WelcomeActivity* uveden do *command módu*, tedy do stavu, ve kterém je možné přijímat pomocí Bluetooth komunikace příkazy, které změní stav na jeho výstupech (log. 1 nebo log. 0). Struktura těchto příkazů je popsána v kapitole 3.4.2.

Rozlišení běžných uživatelů od administrátorů je řešeno již při registraci každého nového uživatele, kde se mu přidělí (nebo nepřidělí) administrátorská práva. Při přihlášení uživatele je tedy podle NFC tagu vyhledán příslušný řádek v databázi uživatelů a ověřeno, zda má uživatel administrátorská práva, či nikoli. Jak již bylo zmíněno v kapitole 3.1.3, vždy musí existovat alespoň jeden administrátor, jejich maximální počet není omezen. Pro ověření, zda v databázi uživatelů je administrátor, je už při vytváření *SignUpActivity* volána metoda *checkAdmin( )*, která ověří, zda již v databázi existuje alespoň jeden uživatel, a pokud žádného nenalezne, zavolá *UsersActivity* a vybídne k vytvoření uživatelského účtu s administrátorskými právy. Takto vytvořený administrátor už nemůže být editován ani vymazán z databáze (viz *AdminActivity* - kapitola 3.3.4), a proto platí, že je-li v databázi uživatelů zřízen alespoň jeden účet, existuje alespoň jeden administrátor.

3.3.3 ShopActivity.java a ShopListFragment.java

Po úspěšném přihlášení je uživatel bez administrátorských práv přesměrován do *ShopActivity*. Je zde implementován stejný posluchač pro zachycení NFC tagu jako v *SignUpActivity*.

V horní části displeje se zobrazí jeho jméno, příjmení a výše kreditu. Pokud je jméno uživatele dlouhé a nevejde se na jeden řádek, je pole, kam se jméno vypisuje, rošířeno o další řádek.

Pod informacemi o uživateli se nachází pole, kam se vypisuje načtené zboží. Pokud není

10 **Event listener:** posluchač události, zpracovává určené události vyvolané operačním systémem nebo uživatelem (např. stisknutí tlačítka nebo, v tomto případě, zahájení komunikace s NFC tagem)

dosud načteno žádné zboží, zobrazí se v něm hláška „Shop list is empty“. Ta zmizí s prvním načteným zbožím (viz obr. 9). O obsluhu tohoto pole se stará *ShopListFragment*, jehož nejdůležitější metodou je *updateList( )*, která zboží po načtení NFC tagu do tohoto pole pomocí metody *SimpleCursorAdapter( )* přidá:

```
  public void updateList() {
       Context ctx = getActivity();           //get Activity context
       Database goods = new Database(ctx);    //open database
             //get Titles of all items and put them into ListView
       String[] from = { Database.COLUMN_TITLE };
       int[] to = { android.R.id.text1 };
       ListAdapter adapter = new SimpleCursorAdapter(ctx,
             android.R.layout.simple_list_item_1, goods.getGoodsSelected(),
             from, to, 0);
       setListAdapter(adapter);

       goods.close();                         //close database
}
```

ShopListFragment dále implementuje *onItemClickListener*, což je metoda starající se o obsluhu události při kliknutí na již načtenou položku v seznamu. V tomto případě zavolá metodu pro zobrazení dialogu s informacemi o zboží, jako je cena a detailnější popis (viz obr. 10). Dialog má pevně dané rozměry, a pokud by se do něj text nevešel, obsah v něm lze posouvat.

Pod polem pro výpis načteného zboží se zobrazuje informace o celkové ceně nákupu, ta se aktualizuje po každé další načtené položce.

Ve spodní části displeje se nachází tlačítko pro ukončení nákupu a odhlášení. Po kliknutí na něj se zobrazí dialog s informací o zbývajícím kreditu (kredit uživatele snížený o cenu nákupu). Pro případ, že na tlačítko pro ukončení klikl uživatel omylem, je zde možnost návratu (tlačítko „Return“), jinak je uživatel po potvrzení tlačítkem „Ok, sign out“ odhlášen, je mu snížen kredit o cenu nákupu a zamkne se zámek. Aplikace se vrací do *SignUpActivity*.

Obr. 9 - Shop Activity se dvěma načtenými položkami

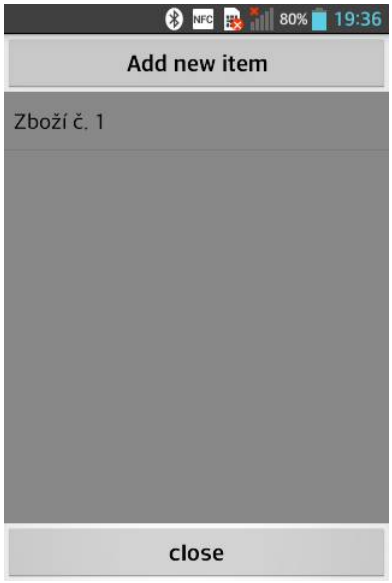
Obr. 10 - Dialog s detaily o nakupovaném zboží

3.3.4 AdminActivity.java

Po přihlášení uživatele s administrátorskými právy je aplikace přesměrována do *AdminActivity*, což je výchozí *Activity*, odsud se lze dostat k databázi zboží i uživatelů. Náhled je na obr. 11.

V horní části displeje se do připraveného *TextView* vypíše jméno a příjmení administrátora pomocí metody *adminName( )*. O něco níže se nachází tlačítka pro vstup do Activities obsluhujících databázi zboží (*GoodsActivity*) a uživatelů (*UsersActivity*) a úplně dole je tlačítko pro odhlášení, zamčení zámku a návrat do *SignUpActivity*. Funkce HW tlačítka „Zpět“ je potlačena.

Obr. 11 - AdminActivity



Obr. 12 - Goods Activity, v databázi je uloženo pouze jedno zboží

3.3.5 GoodsActivity.java a GoodsListFragment.java

Tato *Activity* slouží administrátorovi ke správě databáze zboží. Odsud lze zavolat *Activity* pro přidání nové položky nebo pro editaci již uložené položky v databázi. Dále je zde možné zboží z databáze odstranit nebo zobrazit jeho detailní popis.

V horní části displeje se nachází tlačítko „Add new item“, které volá *GetTagActivity* pro načtení nového NFC tagu, a následně *AddNewActivity* pro vyplnění údajů o novém zboží. Tyto údaje jsou v *GoodsActivity* zpětně získány v metodě *onActivityResult(int requestCode, int resultCode, Intent data)*, princip je popsán v kapitole 3.3.1. Takto získaná data jsou předána metodě *onAddGood( )*, která přidá novou položku do databáze a aktualizuje *ListView* s výrobky (to obsluhuje třída *GoodsListFragment*, použije metodu *updateList( )*, stejně jako *ShopListFragment* v kapitole 3.3.3):

```
public void onAddGood(String title, String details, String tag, String prize)
{
     Database goods = new Database(this);                          //open database
     long id = goods.addGood(title, details, tag, prize);          //add item to
                                                                     database
          //if addGood() return valid id, add item to ListView and update list
     if (id >= 1) {
           ((GoodsListFragment) getSupportFragmentManager().findFragmentById(
                 R.id.goods_list)).updateList(); }
     else {
           String id1 = String.valueOf(id);
           Toast.makeText(this, id1 + "is not valid",
                 Toast.LENGTH_LONG).show();
     }
     goods.close();                                                //close databse
}
```

Již zmíněné *ListView* s výčtem zboží v databázi se nachází pod tlačítkem „Add new item“ a jsou v něm zobrazeny jednotlivé položky z databáze, řazené chronologicky od poslední přidané (viz obr. 12). Při klepnutí na položku se zavolá *ShowGoodActivity* (kapitola 3.3.6), která zobrazí podrobné informace o dané položce - zboží. Při podržení některé položky se zobrazí kontextové menu s možnostmi editovat nebo smazat položky. O jeho obsluhu se stará metoda *onContextItemSelected(MenuItem item)* ve třídě *GoodsListFragment*. Po klepnutí na „Delete“ se zobrazí potvrzovací dialog, umožňující návrat. Po klepnutí na „Edit“ se volá *EditGoodActivity* (kapitola 3.3.7).

Ve spodní části displeje je už jen tlačítko pro návrat do *AdminActivity*.

3.3.6 ShowGoodActivity.java

Tato *Activity* je velice jednoduchá, jejím úkolem je zobrazit informace z databáze do připravených *TextView*. Při jejím vytvoření je třeba zjistit, na kterou položku v databázi uživatel klikl (rozlišuje se podle identifikátoru *id*):

```
long id = getIntent().getLongExtra(EXTRA_ID, -1);
```

Pak se volá metoda *show(long id)*, která najde připravení *TextView* pro výpis jednotlivých sloupců z databáze, zde například pro výpis názvu:

```
TextView title = (TextView) findViewById(R.id.title);
```

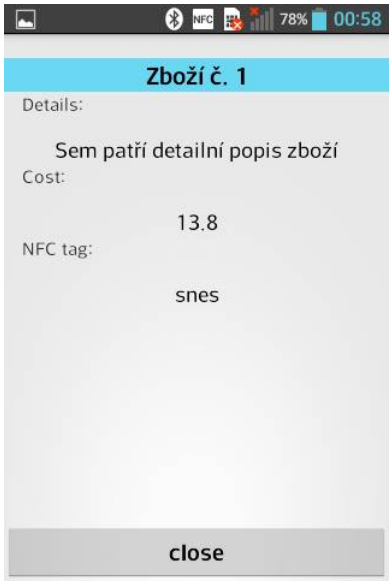
Poté se otevře databáze a pomocí *Cursoru* se identifikují jednotlivé její sloupce (zde opět pro příklad jen sloupec s názvem):

```
Database goods = new Database(this);          //open database
Cursor cursor = goods.getGood(id);            //get good with my id
int titleIndex = cursor.getColumnIndex(Database.COLUMN_TITLE);
```

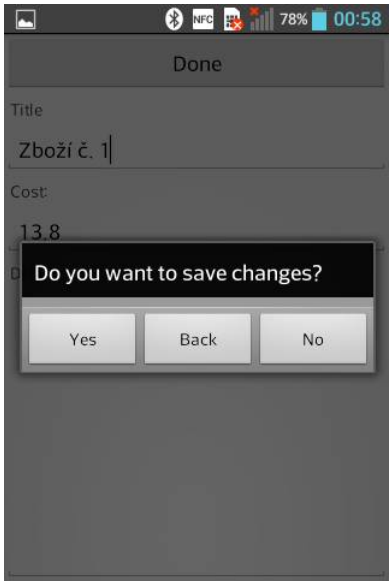
Nakonec se do takto nalezených *TextView* předá obsah patřičných sloupců z databáze:

```
cursor.moveToNext();                          //move to next raw
title.setText(cursor.getString(titleIndex)); //set text to each TextView
```

Náhled je na obr. 13.



Obr. 13 - ShowGoodActivity



Obr. 14 - Dialog při ukončení EditGoodActivity

3.3.7 EditGoodActivity.java

Tato *Activity* slouží k editaci již existujícího zboží v databázi. Lze změnit všechny údaje mimo NFC tagu. Podobně jako u *ShowGoodActivity* se načte zboží v databázi a jednotlivé údaje o něm se vypíšou, tentokrát ale do *EditText* (obdoba *TextView*, ale lze v nich zobrazený text upravit). K tomu slouží metoda *edit(long id)*, funguje podobně jako metoda *show(long id)* v *ShowGoodActivity* (kapitola 3.3.6).

Poté, co uživatel provede požadované změny, má dvě možnosti, jak *Activity* ukončit. První z nich nabízí tlačítko „Done“ v horní části displeje. Zavolá se metoda *onEditGood(View Button)*, která načte řetězce textu z *EditText* do proměnných a ověří, zda jsou kolonky pro název a cenu vyplněny. Pokud nejsou, zobrazí se výstražný dialog s možností návratu nebo potvrzení uložení. Pokud je vše vyplněno, jak má být, pokračuje se do metody *onExitActivity( )*, kde je zboží uloženo do databáze s novými údaji a pod novým id. Pokud toto uložení proběhne úspěšně, je původní záznam o zboží z databáze vymazán , zobrazí se Toast „Successfully updated“ a *EditGoodActivity* je ukončena, aplikace se tedy vrací do *GoodsActivity*.

Druhou možností, jak Activity ukončit, je stiskem hardwarového tlačítka „Zpět“. Jeho obsluhu má na starosti metoda *onBackPressed( )*. Ta zobrazí dialog s dotazem, zda chce uživatel uložit změny (viz obr. 12). Na výběr jsou tyto možnosti:

- **Yes** - Zavolá metodu *onEditGood(View Button)*, která uloží změny a ukončí Activity. Jako parametr se této metodě předá *null*, tedy nulový ukazatel.
- **No** - Ukončí *Activity* bez uložení změn.
- **Back** - Vrátí se do *EditGoodActivity*.

3.3.8 GetTagActivity.java

Tato Activity slouží pouze k načtení NFC tagu nově přidávaného zboží nebo uživatele (v obou případech je využita stejná *Activity*). Pro rozlišení, odkud je volána, je použita metoda *context.getCallingActivity( )*:

```
String EXTRA_PARENT = this.getCallingActivity().toString();
```

Metody pro načtení NFC tagu jsou podobné jako v *SignUpActivity* (kapitola 3.3.2), ale jsou zde navíc metody pro ověření jedinečnosti NFC tagu v databázi. Ty jsou volány přes metodu *checkDuplicity(String tag)*, která podle parametru *EXTRA_PARENT* rozhodne, zda bude nově načtený tag vyhledávat v databázi výrobků nebo uživatelů. Poté zavolá patřičnou metodu - buď *checkTagGood(String tag)* pro vyhledávání v databázi výrobků nebo *checkTagUser(String tag)* pro vyhledávání v databázi uživatelů.

Obě tyto metody pracují na stejném principu. Otevře se databáze, do *Cursoru* se načtou všechny položky z databáze, zjistí se jejich počet a ve smyčce se u každé položky porovná uložený NFC tag s právě načteným NFC tagem. Metody vracejí hodnotu *true*, pokud naleznou shodu. Zde je uvedena metoda *checkTagGood(String tag)*:

```
public boolean checkTagGood(String new_tag) {
     Database goods = new Database(this);   //open databse
     Cursor cursor = goods.getGoods();      //get all items of database
     int raws = cursor.getCount();          //get number of items
     int indexTag = cursor.getColumnIndex(Database.COLUMN_TAG);
           //compare every item's Nfc tag to new_tag
     for(int i = 0; i < raws; i++) {
           cursor.moveToNext();
           if(cursor.getString(indexTag).compareTo(new_tag) == 0) {
                 cursor.close();            //close cursor
                 goods.close();             //close database
                 return true;
           }
     }
     cursor.close();                        //close cursor
     goods.close();                         //close database
     return false;
}
```

Pokud je NFC tag v dané databázi jedinečný, zavolá se metoda *callActivity( )*, která opět podle parametru *EXTRA_PARENT* zavolá *AddGoodActivity* nebo *AddUserActivity*.

3.3.9 AddGoodActivity.java

Tato Activity slouží k vyplnění údajů o nově přidávaném zboží. Uživateli se zobrazí tři pole *EditText* pro výpis názvu, ceny a detailního popisu, a v horní části displeje tlačítko „Done“. Každý *EditText*, do kterého se data vepisují, zobrazuje poloprůhledným písmem nápovědu, co má uživatel do tohoto pole vepsat (jestli název, cenu nebo detailní popis). Do *EditText* pro cenu lze vepsat pouze číslo, může být i desetinné. To je zajištěno ošetřením vstupu v příslušném *layoutu*, konkrétně metodou *android:inputType*, do které se jako parametr předá *„numberDecimal“*.

Po stisknutí tlačítka „Done“ se zavolá metoda *onAddGood(View Button)*, která ověří, zda jsou vyplněna pole pro název a cenu - oba tyto údaje musí každé zboží mít, detailní popis je volitelný. Pokud některý z těchto údajů vyplněn není, zobrazí se varovný dialog

upozorňující na tuto skutečnost. Uživatel může buď potvrdit, že si i přesto přeje zboží do databáze uložit, nebo má možnost návratu. Pokud uživatel uložení potvrdí nebo jsou údaje správně vyplněny, volá se *onExitActivity*. Ta data z *EditText* načte do lokálních proměnných a odsud je uloží do proměnné typu *Intent*[11], která *Activity* slouží jako návratová hodnota (viz kapitola 3.3.1).

Dále lze *Activity* ukončit stiskem hardwarového tlačítko „Zpět“. To má implementovaný *event listener onBackPressed( )*, který zobrazí stejný dialog jako v *EditGoodActivity* (kapitola 3.3.7), tedy dotaz, zda si uživatel přeje uložit data. Jsou zde tyto možnosti:

- **Yes:** uloží zboží do databáze zavoláním metody *onAddGood(View Button)*
- **No:** smaže data z *AddGoodActivity* a vrátí se do *GoodsActivity,* v databázi zůstane původní záznam o zboží
- **Back:** vrátí se zpět do *AddGoodActivity*

Dále jsou tu implementovány metody pro uložení a obnovení dat v případě změny orientace displeje nebo jiného znovuvytvoření *Activity*. Pro uložení dat slouží metoda *onSaveInstanceState(Bundle outState)*, která se volá vždy při ukončení *Activity* jiným způsobem než zásahem uživatele. Do proměnné *outState* vloží metodou *outState.putString(String name, String data)* text z *EditText* s příslušným identifikátorem *name*, tedy například z *EditText* pro název zboží jsou data uložena následovně:

```
outState.putString("title", ((EditText)findViewById(R.id.TitleNew))
     .getText().toString());
```

Obnovení dat a jejich výpis do *EditText* proběhne metodou *restore( )* během vytvoření *Activity*, pokud byla při jejím ukončení nějaká data uložena (podmínka *Bundle savedInstanceState != null* v metodě *onCreate(Bundle savedInstanceState)*). Jako příklad poslouží opět obnovení textu do pole pro název:

```
TextView _title = (TextView) findViewById(R.id.TitleNew);
_title.setText(title);
```

3.3.10 Database.java

Tato třída slouží k obsluze databáze zboží. Je v ní implementována třída *DatabaseHelper*, která se stará o vytvoření databáze (metoda *onCreate(SQLiteDatabase db)*), a její upgrade (metoda *onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)*). Upgarde databáze tato aplikace nevyužívá, a tedy není řešeno zachování dat při volání metody *onUpgrade( )*.

Ve třídě *Database* jsou vytvořeny metody pro práci s databází. První z nich je metoda *getGoods( )*, která vrací všechny řádky databáze v proměnné typu *Cursor*. *Cursor* používají jako návratový typ všechny metody, vracející jeden nebo více řádků z databáze. Další takovou metodou je *getGood(long id)*, která vrací právě jeden řádek z databáze, vybraný podle jedinečného identifikátoru *_id* . Identifikátor *_id* je doporučeno vložit do prvního sloupce databáze, značně pak usnadňuje vyhledávání v databázi. Musí ale být uveden přesně v požadovaném formátu, tedy i s podtržítkem na začátku.

Další metodou je *getGoodsSelected( )*, která vrací všechny řádky z databáze, v nichž je ve sloupci COLUMN_SELECTED hodnota „y“. V tomto sloupci sice mohl být použit typ proměnné *boolean* namísto *String*, ale uložení *Stringu* do databáze je o něco jednodušší.

11 **Intent**: Jedná se o datový typ používaný v jazyce Java. Vyjadřuje úmysl provést nějakou akci, při volání *Activity* jako potomka často slouží jako návratový typ.

Tato metoda se volá ze *ShopActivity*, resp. *ShopListFragment*, kam v proměnné *Cursor* vrátí všechno zboží z databáze, které je aktuálně nakupováno, ale ještě není zaplaceno (teprve pak je z databáze odstraněno, je to pojistka pro případ pádu aplikace). Aby tato metoda měla co vrátit, je nejprve potřeba zboží označit. K tomu slouží metoda *selectGood(long id)*, která využívá metody *database_name.update(String name, ContentValues values, String where, String[] selectionArgs)*. Funguje tak, že v již otevřené databázi *database_name* vyhledá všechny položky splňující podmínku v *selectionArgs* (v tomto případě je sem vložen jedinečný identifikátor *_id* dané položky), a provede příkazy definované v *ContentValues values*. Například příkaz *values.put(COLUMN_SELECTED, "n")* nahradí obsah ve sloupci *COLUMN_SELECTED* druhým argumentem metody *put(String column, String value)*, tedy řetězcem „y“. Celá metoda *selectGood(long id)* pak vypadá takto:

```
public void selectGood(long id) {
     SQLiteDatabase db = openHelper.getWritableDatabase();
     String[] selectionArgs = { String.valueOf(id) };
     ContentValues values = new ContentValues();
     values.put(COLUMN_SELECTED, "y");
     db.update(TB_NAME, values, COLUMN_ID + "= ?", selectionArgs);
     db.close();
}
```

Další metoda *deselectGood(ling id)* pracuje podobným způsobem, jen vkládá jiný řetězec. Poslední metodou sloužící k obsluze právě nakupovaných položek je metoda *deleteSelected( ),* která při ukončení nákupu všechny položky označené metodou *selectGood(long id)* vymaže z databáze.

Předposlední metodou je *deleteGood(long id)*, která položku vybranou identifikátorem *_id* vymaže z databáze a vrací „true“, pokud odstranění proběhne v pořádku. Pokud této metodě předáme argument *_id* neexistující položky, vrátí metoda „false“.

Poslední metodou je *addGood(String title, String details, String tag, String prize)*. Ta opět metodou *put(String column, String value)* vloží do *ContentValues values* data k uložení a poté metodou *database_name.insert(String name, String nullColumnHack, ContentValues values)* vloží nový řádek do databáze a vrátí *_id* nově přidané položky.

3.3.11 UsersActivity.java

Tato Activity je obdobou *GoodsActivity* s tím rozdílem, že tvoří rozhraní pro databázi uživatelů, nikoli zboží. Jsou tu implementované podobné metody se stejnými principy.

Jediný rozdíl je v metodě *onBackPressed( )* volané po stisku HW tlačítka „Zpět“ a metodě *onClickBack(View Button)* volané po kliknutí na tlačítko „Back“. Protože aplikace nemůže fungovat bez alespoň jednoho administrátorského účtu, aplikace se po prvním spuštění (tedy s prázdnou databází uživatelů) přesměruje právě sem, do *UsersActivity*, namísto *SignUpActivity* (podrobněji viz kapitola 3.1.3). Ve svém standardním běhu pokračuje až poté, co je přidán administrátorský účet. Proto se při stisku tlačítka „Zpět“, který by v tomto případě znamenal návrat do *SignUpActivity*, ověřuje, zda už byl vytvořen administrátorský účet, a pokud dosud neexistuje, vypíše se varovný dialog. Uživatel zvolí, zda aplikace nadále zůstává zde v *UsersActivity* a čeká na přidání administrátorského účtu, nebo bude ukončena. Metoda *onClickBack( )* tedy vypadá následovně:

```
public void onClickBack(View Button) {
	Users users = new Users(this);		//open database
	Cursor cursor = users.getUsers();	//get all items from database
	int column_count = cursor.getCount();	//get number of items
	cursor.close();				//close cursor
	users.close();				//close database
	  //if there is no item in database (means no administrator), show dialog
	if(column_count <= 0) {
		showAlertDialog();
		return;
	}
	else {
		setResult(RESULT_OK);
	}
	finish();
}
```

3.3.12 Ostatní třídy

Zbytek *Activities*, tedy *AddUserActivity*, *EditUserActivity* a *ShowUserActivity* jsou v principu velice podobné *AddGoodActivity*, *EditGoodActivity* a *ShowGoodActivity*. Proto nemá smysl je tu podrobněji popisovat.

Stejně tak princip třídy *Users* je stejný jako u třídy *Database*. Tak jako se *Database* stará o databázi zboží, obsluhuje *Users* databázi uživatelů. Nejsou tu metody pro označení právě nakupovaného zboží, je tu ale navíc metoda *editCredit(long id, String credit)*, která pomocí metody *database_name.update(String name, ContentValues values, String where, String selectionArgs)* uživateli s příslušným *id* zvýší kredit o hodnotu definovanou v argumentu *credit*. Matoucí by mohlo být, že *credit* je typu *String* a nikoli *float*. Je to z toho důvodu, že zápis řetězce do databáze je jednodušší, jak už bylo zmíněno u vyplnění a uložení ceny zboží.

Nakonec ještě zmínka o *UsersListFragment*, který obsluhuje pole pro výpis uživatelů stejně jako jeho obdoba *GoodsListFragment* pro zboží. Protože by se v poli pro výpis uživatelů mělo objevit jméno a příjmení, musí se použít jiná metoda než *SimpleCursorAdapter*. Vhodné je použít *ViewBinder*, který v tomto případě umožní vypsat data ze dvou sloupců do jednoho pole. Ve zdrojovém kódu to vypadá takto:

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(ctx,
    android.R.layout.simple_list_item_1, users.getUsers(), from, to, 0);

adapter.setViewBinder( new SimpleCursorAdapter.ViewBinder() {

    @Override
    public boolean setViewValue(View view, Cursor cursor, int columnIndex) {
        TextView text1  = (TextView) view;
        int nameIndex = cursor.getColumnIndex(Users.COLUMN_NAME);
        int surnameIndex = cursor.getColumnIndex(Users.COLUMN_SURNAME);
        String text = cursor.getString(nameIndex) + " " +
                      cursor.getString(surnameIndex);
        text1.setText(text);
        return true;
    }
});
```

Druhou změnou v *UsersListFragment* oproti *GoodsListFragment* je rozšíření kontextového menu, zobrazujícího se po podržení některého uživatele, o volbu „Add credit“. Po jejím zvolení se zobrazí dialog s informacemi o uživateli a výši jeho kreditu, pod tímto textem se nachází pole *EditText*, kam je možno vepsat částku, o kterou se má uživatelův kredit navýšit. Pod tímto polem jsou tlačítka „Back“ pro návrat do *AdminActivity* bez přidání nového kreditu a tlačítko „Add credit“ pro navýšení uživatelova kreditu o hodnotu vepsanou v *EditText*. Pole *EditText* přijímá jen číslo, může být s desetinným rozvojem, prázdné (nevyplněné) pole vrátí hodnotu *0*.

3.4 Připojení zámku a komunikace

Jak už bylo zmíněno v kapitole 2.1, zámek je spínán na výstupu Bluetooth modulu, který komunikuje s telefonem. Schéma zapojení je v příloze. Při spuštění aplikace je třeba modul vyhledat a správně identifikovat, následně navázat komunikaci a uvést modul do *command módu,* tedy do stavu, kdy přijímá příkazy posílané přes Bluetooth po sériové lince. Teprve poté je možné spínat zámek.

3.4.1 Navázání komunikace s modulem

Doporučený postup na internetových stránkách pro vývojáře pro připojení zařízení přes Bluetooth [21] je vyhledat všechna zařízení v dosahu a porovnávat jejich adresy s adresou v paměti. Tento postup vyžaduje další rozdělení běhu aplikace do jednotlivých vláken, jinak bude navazování komunikace nespolehlivé a může zapříčinit nestandardní chování aplikace.

V případě této aplikace je situace zjednodušena tím, že Bluetooth MAC adresa modulu je předem známá a neměnná, a tedy může být uložena přímo v paměti aplikace, nemusí se odnikud získávat. Telefon se proto pokouší připojit přímo k zařízení s touto adresou a všechna ostatní Bluetooth zařízení v dosahu ignoruje.

K připojení modulu k telefonu slouží metoda *LockEnable( )*, která je volána na pozadí při spouštění aplikace (viz kapitola 3.3.1). Po nastavení adresy modulu je vytvořen Bluetooth Socket, tedy spojení pro komunikaci, a to metodou *BT_device.createInsecureRfcommSocketToServiceRecord(UUID uuid)*. To vytvoří jednosměrný komunikační kanál pro odesílání příkazů z telefonu do modulu. Poté se volá metoda *bluetooth_adapter.cancelDiscovery( )*, která ukončí případné vyhledávání zařízení. Vyhledávání může být spuštěno i jinou běžící aplikací, proto je doporučeno tuto metodu volat vždy před připojením k zařízení metodou *BT_socket.connect( )*. Vyhledávání zařízení totiž zabírá velkou část kapacity z komunikačního kanálu Bluetooth, připojení by proto mohlo selhat.

Pokud připojení metodou *connect( )* selže, ať už z důvodu nedostatečné kapacity kanálu nebo proto, že zařízení není v dosahu, pokusí se aplikace uzavřít komunikační Socket metodou *BT_socket.close( )*. Po selhání připojení se vypíše hláška, že modul nebyl nalezen, a aplikace se ukončí.

Po úspěšném připojení lze uvést modul do *command módu*. K tomu slouží příkaz **$$$**, předdefinovaný výrobcem modulu. Protože nelze přímo odeslat řetězec *String*, musí být řetězec napřed načten do pole *byte[ ]* a teprve poté odeslán metodou *output_stream.write(char[ ] buffer)*. Poté je nastavena kontrolní proměnná *boolean lock_ok* na hodnotu *true* a a metoda *LockEnable( )* je ukončena. Aplikace vyhodnotí výstup z ní a podle nastavení kontrolní proměnné *lock_ok* buď pokračuje do *SignUpActivity* nebo se ukončí.

```
public void LockEnable() {
            //set address of module device
      boolean  lock_ok = false;
      String BTAddress = "00:06:66:62:02:DF";       //BT MAC address of module
      BluetoothDevice lock = mBTAdapter.getRemoteDevice(BTAddress);
      BluetoothSocket mBTSocket = null;
      try {
            mBTSocket =lock.createInsecureRfcommSocketToServiceRecord(BT_UUID);
      } catch (IOException e) { };
```

```
        //this should be called even if discovery was not started by this app
    mBTAdapter.cancelDiscovery();
          //connect to module device
    try {
          mBTSocket.connect();
    } catch(IOException e) {
          try {
                mBTSocket.close();
          } catch (Exception e2) { };
          //module device was not found, return to main Thread and exit
            applicaton
          lock_ok = false;
          return;
    }
          //get output stream to send commands
    try {
          OutputStream out = mBTSocket.getOutputStream();
    } catch (IOException e) { };
          //enable command mode by command „$$$“
    String message = "$$$";
    byte[] buffer = message.getBytes();
    try {
          out.write(buffer);
    } catch (IOException e) { };
    lock_ok = true;
    return;
}
```

3.4.2 Ovládání zámku v SignUpActivity.java

V *SignUpActivity* již je modul připojen a uveden do *command módu*, je tedy možné na jeho výstupních pinech měnit logický stav pomocí příkazu zaslaného přes Bluetooth. Pomocí takového příkazu je odemykán a zamykán zámek.

Pro odemčení je třeba přes Bluetooth po sériové lince odeslat příkaz **S*,0202<cr>**. Ten je složen ze tří částí a ukončovacího znaku:

- **S*,** - změna výstupního stavu na pinu
- **02** - definuje pin, na kterém dojde ke změně stavu
- **02** - přepnutí výstupního stavu do log. 1
- **<cr>** - ukončovací znak příkazu. Může být odesláno i <cr> <lf>.

Pro odemčení zámku se volá metoda *setLockOff( )*. Protože metoda *write(byte[ ] buffer)* neumí pracovat s řetězcem, musím příkaz nejprve načíst do pole *byte[ ] buffer* a pak teprve odeslat:

```
public void SetLockOff() {
          //Unlock
    String message = "S*,0202\r";
    byte[] buffer = message.getBytes();
    try {
          out.write(buffer);
    } catch (IOException e) { };
}
```

Pro zamčení zámku se používá metoda *setLockOn( )*, velice podobná té pro odemčení zámku. Jediný rozdíl je v příkazu, zde se odesílá **S*,0200<cr>**. Struktura je stejná, jen se

přepíná do log. 0, tedy znaky **02** před <cr> jsou nahrazeny **00**. Metoda *SetLockOn( )* pak vypadá následovně:

```
public void SetLockOn() {
                    //Lock
      String message = "S*,0200\r";
      byte[] buffer = message.getBytes();
      try {
            out.write(buffer);
      } catch (IOException e) { };
 }
```

Pro odemykání i zamykání by šlo sice implementovat jedinou metodu, která by příkaz přijímala jako argument, například takto: *metoda_ovladani_zamku*(*String message)*. Použitý způsob je ale připraven pro případné pozdější úpravy jedné z těchto metod, např. rozsvícení signalizační LED diody na jiném výstupním pinu modulu, která bude indikovat odemčený zámek.

3.4.3 Ukončení komunikace s modulem

Pro ukončení *command módu* při ukončování aplikace slouží metoda *LockDisable(BluetoothDevice device)*. Ta, podobně jako metoda *lockEnable( )*, běží na pozadí, zatímco hlavní vlákno aplikace zobrazuje čekací dialog. V tomto případě by rozdělení do vláken ani nebylo nutné, nečeká se na navázání komunikace, jen se odešle příkaz „**---**“ ukončený <cr>, kterým je ukončen *commad mód*. To netrvá dost dlouho, aby uživatel zaznamenal prodlevu aplikace. Metoda *LockDisable(BluetoothDevice device)* vypadá následovně:

```
public void LockDisable(BluetoothDevice lock) {
            //disable command mode
      try {
            String discmd = "---\r";
            byte[] buffer2 = discmd.getBytes();
            out.write(buffer2);
      } catch (IOException e) { };
      return;
 }
```

K ukončení komunikace mezi telefonem a modulem dojde po návratu běhu aplikace do hlavního vlákna, tedy poté, co proběhne metoda *LockDisable( )*. Slouží k tomu metoda *bluetooth_socket.close( )*.

3.5 Realizace projektu

Ukázka výdejového automatu je na obr. 15. Jedná se o box, uvnitř kterého je umístěno zboží.

Z vnitřní strany dvířek je vlepená NFC anténa spolu se zadním krytem telefonu. Oddělit zadní kryt s anténou od telefonu bylo bohužel nezbytné, protože telefon nedokáže přečíst NFC tag přiložený k němu zepředu, anténa má příliš krátký dosah. Takto je možné využít NFC anténu k přihlášení uživatele přiložením NFC tagu na naznačené místo na dvířkách, a dále ke čtení nakupovaného zboží při jeho vyjmutí z boxu.

Proti elektromagnetickému zámku je ve dvířkách zabudovaný klasický dveřní mechanismus, zámek nedovolí otevření dvířek, pokud není odemčený. Bluetooth modul, ovládající zámek, je umístěn uvnitř úložného prostoru, ven je vyveden 12 V napájecí adaptér.

Obr. 15 - Ukázka realizovaného NFC automatu

Legenda k obr. 15:
1 - Box pro uložení zboží
2 - Dvířka
3 - Mobilní telefon s aplikací NfcRobot
4 - Elektromagnetický zámek
5- Dveřní protikus k zámku
6 - Zde je z druhé strany dvířek umístěná NFC anténa

4 Závěr

Cílem této práce bylo vytvořit aplikaci pro OS Android, která bude ovládat RFID výdejový automat. Aplikace využívá pro identifikaci zboží i uživatelů NFC tagy, ale bohužel má mobilní telefon, na kterém byla aplikace testována, anténu pro NFC komunikaci vlepenou v zadním krytu přístroje.

Protože je dosah NFC velmi malý, typicky několik cm, není takto umístěná anténa pro toto využití vhodná. Proto byl zadní kryt při realizaci automatu oddělen od telefonu a anténa vodivě propojena k konektorům na zadní straně telefonu. Takto anténa slouží k přihlášení uživatele i načtení nakupovaného zboží. Problém nevhodně umístěné NFC antény, navíc s velmi malým dosahem, by bylo možné řešit i jiným způsobem, nejlépe připojením externího NFC modulu s vlastní anténou, který by se k mobilnímu telefonu připojil pomocí již vytvořené komunikace přes Bluetooth.

Operační systém Android umožňuje emulaci sériové linky přes Bluetooth, čehož tato aplikace využívá pro odemykání a zamykání zámku. Zde bývá většinou nutná spolupráce uživatele při navazování komunikace s předem neznámým zařízením, je třeba ověřit bezpečnostní kód před spárováním přístrojů. Tento krok lze přeskočit, pokud je předem známá Bluetooth MAC adresa zařízení, se kterým chce aplikace komunikovat. Tak tomu je v případě aplikace NfcRobot, která komunikuje s předem známým Bluetooth modulem, proto probíhá navázání komunikace automaticky při startu aplikace.

Největší výhodou tohoto řešení výdejového automatu je jeho snadná obsluha a pro uživatele mobilních telefonů s OS Android nabízí známé a přehledné prostředí. Přihlašování uživatelů může navíc fungovat i prostřednictvím mobilního telefonu s NFC, který má většina lidí spíše po ruce než NFC tag například na klíčích nebo v peněžence.

5 Literatura

[1] Android x86 - Porting Android to x86 [online], poslední revize 13.3.2014, citováno 11.5.2014. Dostupné z: http://www.android-x86.org/#Tested_platforms

[2] Android Developers, [online], citováno dne 11.5.2014. Dosupné z: https://source.android.com/

[3] Linux Kernel Archives - About [online], poslední revize 5.12.2013, citováno 5.11.2014. Dostupné z: https://www.kernel.org/category/about.html

[4] Apache Licence, Version 2.0 [online], citováno 11.5.2014. Dostupné z: http://www.apache.org/licenses/LICENSE-2.0

[5] Google Buys Android For Its Mobile Arsenal - Businessweek [online], poslední revize 16.8.2005, citováno 28.4.2014. Dostupné z: http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal

[6] Android Developer - Android 4.4 APIs [online], citováno 13.5.2014. Dostupné z: https://developer.android.com/about/versions/android-4.4.html

[7] Google Inc - Android Compatibility Definition: Android 1.6 [online], citováno 11.5.2014. Dostupné z: http://static.googleusercontent.com/media/source.android.com/cs//compatibility/1.6/android-1.6-cdd.pdf

[8] Google Inc - Android Compatibility Definition: Android 4.4 [online], poslední revize 27.11.23013, citováno 11.5.2014. Dostupné z: http://static.googleusercontent.com/media/source.android.com/cs//compatibility/4.4/android-4.4-cdd.pdf

[9] Android Developers - Dashboards [online], poslední revize 1.5.2014, citováno 11.5.2014. Dostupné z: http://developer.android.com/about/dashboards/index.html

[10] Infosecurity - DroidCleaner: Android malware that infects PCs [online], poslední revize 4.2.2013, citováno 11.5.2014. Dostupné z: http://www.infosecurity-magazine.com/view/30560/droidcleaner-android-malware-that-infects-pcs/

[11] Xamarin - Android Lifecycle [online], citováno 11.5.2014. Dostupné z: http://docs.xamarin.com/guides/android/application_fundamentals/activity_lifecycle/

[12] Rhode & Schwarz - Near Field Communication (NFC) Technology and Measurments [online], citováno 11.5.2014. Dostupné z: http://cdn.rohde-schwarz.com/dl_downloads/dl_application/application_notes/1ma182/1MA182_5E_NFC_WHITE_PAPER.pdf

[13] RFID Handbook - Active Load modulation [online], citováno 11.5.2014. Dostupné z: http://rfid-handbook.de/about-rfid/active-load-modulation.html?showall=&limitstart=

[14] NEUSTUPA, Zdeněk, STAŠA, Pavel. Technologie NFC. *Sdělovací technika* 4/2013, ISSN 0036-9942

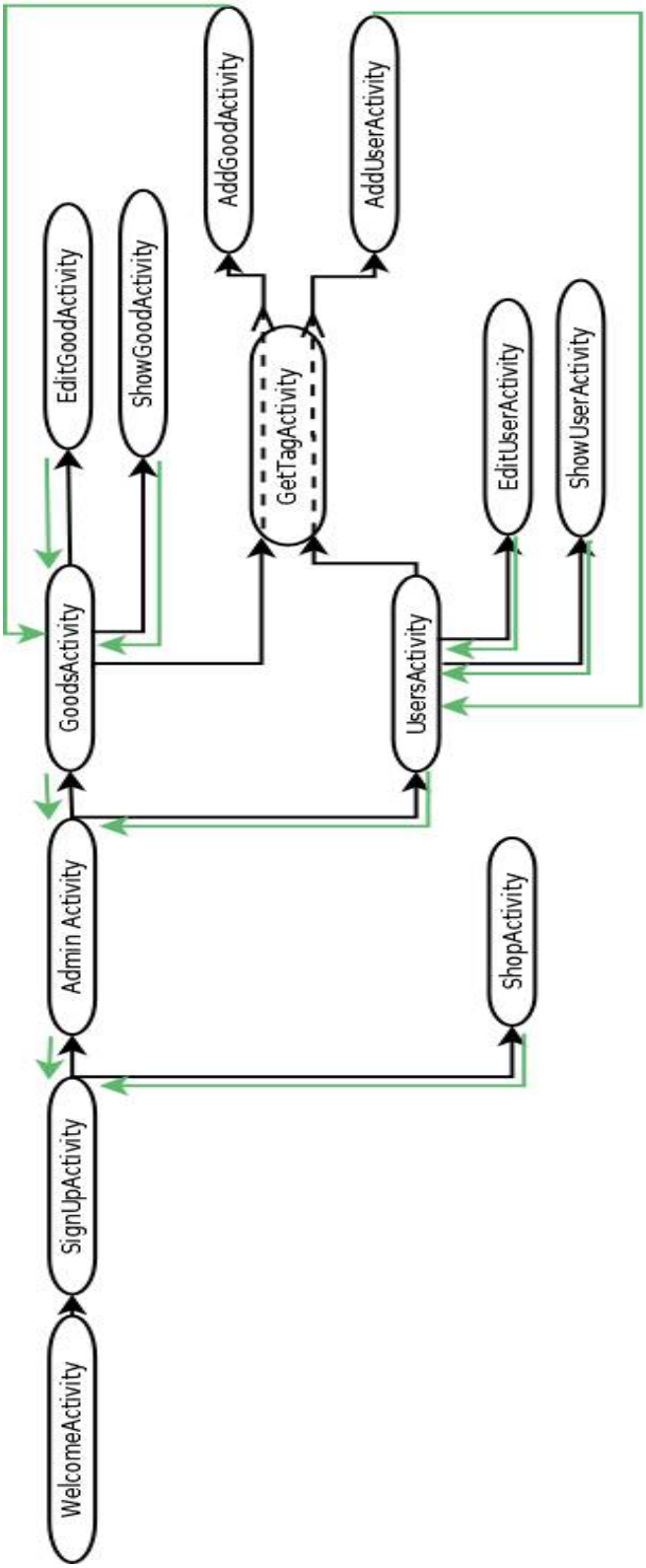
[15] IEEE SA - 802.15.1 - 2002 [online], citováno 11.5.2014. Dostupné z: http://standards.ieee.org/findstds/standard/802.15.1-2002.html

[16] KLINE, Paul, KUMAR, C Bala, THOMPSON, Timothy. *Bluetooth Application Programming with the Java APIs* . 1st. ed. Morgan Kaufmann, 2008. ISBN 978-0-12-374342-8

[17] HUANG, Albert, RUDOLPH, Larry. *Bluetooth Essentials for Programmers*. 1st. ed. Cambridge University Press, 2007. ISBN 978-0-52-170375-8
[18] PLEW, Ronald, STEPHENS, Ryan. *Naučte se SQL za 21 dní: pochopte principy jazyka relačních databází : uplatněte získané dovednosti při tvorbě dotazů a databázových aplikací*. 1st. ed. Computer Press, 2004. ISBN 80-722-6870-8
[19] Interval.cz - Databáze a jazyk SQL [online], poslední revize 4.8.2000, citováno 11.5.2014. Dostupné z: http://interval.cz/clanky/databaze-a-jazyk-sql/
[20] Android Developers - NFC Basics [online], citováno 11.5.2014. Dostupné z: http://developer.android.com/guide/topics/connectivity/nfc/nfc.html
[21] LG Electronics CZ - LG Optimus L5 E610 [online], citováno 11.5.2014. Dostupné z: http://www.lg.com/cz/telefony/lg-E610-optimus-l5
[22] Roving Networks - RN42/RN42N Class 2 Bluetooth Module [online], citováno 11.5.2014. Dostupné z: http://ww1.microchip.com/downloads/en/DeviceDoc/rn-42-ds-v2.32r.pdf
[23] Android Developer - Bluetooth [online], citováno 11.5.2014. Dostupné z: http://developer.android.com/guide/topics/connectivity/bluetooth.html
[24] MyMobiles - HTC Dream [online], citováno 11.5.2014. Dostupné z: http://www.mymobiles.com/phones/htc/dream
[25] Android Developers - Activity [online], citováno 11.5.2014. Dostupné z: http://developer.android.com/reference/android/app/Activity.html
[26] RFID Handbook - Active Load Modulation - Principles [online] citováno 11.5.2014. Dostupné z: http://rfid-handbook.de/about-rfid/active-load-modulation.html?showall=&start=2
[27] HungHao Mobile - LG-E610v Optimus L5 [online], citováno 11.5.2014. Dostupné z: http://www.hunghoamobile.com/sites/default/files/styles/detail_blog/public/lg-e610V-optimus-L5_0.gif
[28] Mbed - RN-42 Bluetooth [online], citováno 11.5.2014. Dostupné z: http://mbed.org/cookbook/RN-42-Bluetooth

6 Přílohy

6.1 Schéma aplikace NFC robot



6.2 Schéma zapojení modulu RN-42 a zámku