VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

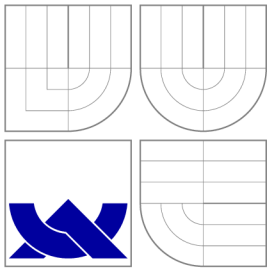
APLIKACE PRO TABLET S OS ANDROID URČENÁ PRO SERVISNÍ TECHNIKY PLYNOVÝCH KOTLŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

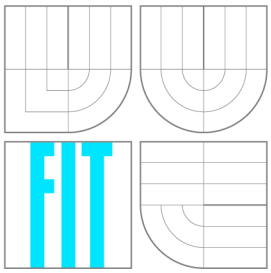
AUTOR PRÁCE
AUTHOR

ADAM PODOLA

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

APLIKACE PRO TABLET S OS ANDROID URČENÁ PRO SERVISNÍ TECHNIKY PLYNOVÝCH KOTLŮ

ANDROID APPLICATION FOR GAS FURNACE SERVICEMEN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ADAM PODOLA
AUTHOR

VEDOUCÍ PRÁCE — Ing. IGOR SZŐKE, Ph.D.
SUPERVISOR

BRNO 2014

Abstrakt

Tato bakalářská práce popisuje specifikaci požadavků, návrh a implementaci aplikace pro zařízení s operačním systémem Android, primárně využitelnou servisními techniky plynových kotlů. Aplikace umožňuje budování patřičné znalostní báze, využitelné při úkonech prováděných servisním technikem. Uživatelské rozhraní aplikace je postaveno na funkcionalitě nabízené fragmenty. Pro uchování dat se používá SQLite databáze v kombinaci s externím úložištěm zařízení.

Abstract

This thesis describes the required specifications as well as the design and implementation of an application for devices running Android operating system which is primarily used by service technicians of gas boilers. This application allows the creation of an appropriate knowledge base which is useful for operations performed by a service technician. The application user interface is based on the functionality provided by the fragments. SQLite database is used in combination with an external storage device to preserve data.

Klíčová slova

Android, databáze, SQLite, aktivita, fragment, úložiště, rozložení, záměr, QR kód, servisní technik

Keywords

Android, database, SQLite, activity, fragment, storage, layout, intent, QR code, service technician

Citace

Adam Podola: Aplikace pro tablet s OS Android určená pro servisní techniky plynových kotlů, bakalářská práce, Brno, FIT VUT v Brně, 2014

Aplikace pro tablet s OS Android určená pro servisní techniky plynových kotlů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Igora Szőkeho, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Adam Podola
19. května 2014

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Igoru Szőkemu, Ph.D. za čas strávený během odborného vedení této bakalářské práce, konzultace a cenné připomínky.

Obsah

Kapitola 1

Úvod

V současné době se na trhu plynových zařízení určených pro vytápění budov pohybuje nemalé množství výrobců, kteří vyrábějí různě složitá plynová zařízení, jenž musí být udržována v provozuschopném stavu servisními techniky po celém světě. S rostoucí vyspělostí těchto zařízení se úměrně zvyšují požadavky na znalosti a zkušenosti potřebné pro odbornou údržbu. Proto výrobci poskytují podpůrné materiály ve formě servisní dokumentace v tištěné nebo elektronické formě, která napomáhá při servisních úkonech prováděných na plynových zařízeních vlastněných fyzickými či právnickými osobami.

Předmětem práce je návrh a implementace aplikace, pracující v současné době na nejrozšířenější mobilní platformě Android, která zjednoduší evidenci zákazníků vlastnících plynová zařízení, na kterých servisní technik provádí servisní úkony, při kterých odhaluje závady a hledá způsoby jejich řešení, přičemž při tomto procesu může být nápomocna budovaná znalostní báze ve formě historie prováděných servisních úkonů a uchovávaná servisní dokumentace. Dále se tato aplikace snaží o urychlení celého procesu servisního úkonu, počínaje objednáním zákazníka a konče závěrečnou fakturací za provedenou práci, užité náhradní díly a další položky.

Kapitola 2

Operační sytém Android

Od 21. října 2008 se jedná o otevřený operační systém založený na linuxovém jádře, vyvíjený skupinou `Open Handset Alliance`[1] zahrnující 84 firem z oblasti nejen mobilních telefonů. V současné době se jedná o nejrozšířenější mobilní platformu. Pro programování aplikací se využívá Android SDK (Software Development Kit) obsahující balík ADT (Android Development Tools), který obsahuje veškeré potřebné nástroje budoucího vývoje.

2.1 Verze

Před začátkem návrhu jakékoli aplikace musí být rozhodnuto, pro které verze platformy Android bude výsledný produkt dostupný. Obecně lze říci, že čím nižší verzi podporujeme, tím více se musí řešit zpětná kompatibilita. Samozřejmostí je testování na všech podporovaných platformách. Abychom nemuseli vlastnit širokou škálu zařízení, ADT poskytuje funkci simulace, při které se zvolí parametry (např. velikost paměti RAM, rozlišení displaye) a verze platformy. Tímto způsobem lze odhalit chyby, které překladač nebyl schopen detekovat. Pro zachování zpětné kompatibility se využívá knihoven *Support Library*. Tabulka 2.1 zachycuje procentuální podíl verzí platforem Android. Na základě získaných informací musí být rozhodnuto, které verze platformy jsou doposud užívané a které by bylo účelné podporovat. Data jsou získávána prostřednictvím aplikace `Google Play Store`, verze `API` nedosahující procentuálního podílu alespoň 0.1% nejsou zahrnuty. Z tabulky mimo jiné můžeme pozorovat klesající podíl zařízení s verzí `API` nižší než 16. Dále je zřejmé, že pokud chceme podporovat téměř 70% zařízení, stačí nám vytvořit aplikaci pro `API` 16 až 19. Tabulka je aktualizována každých sedm dní, dostupná na [2]. Zde se čtenář taktéž dozví procentuální rozložení velikostí displaye a hodnot rozlišení, jenž aktuálně využívají uživatelé aplikací. I tato skutečnost musí být brána v potaz.

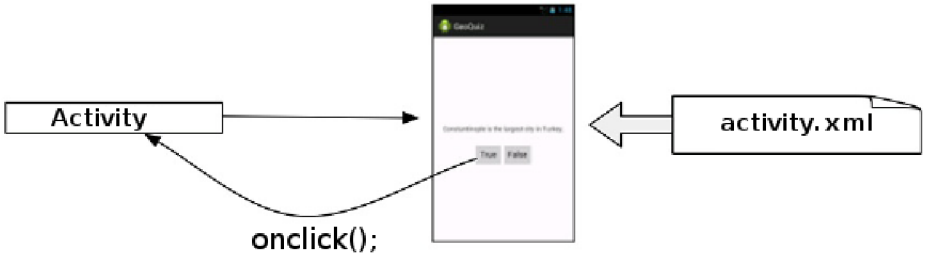
2.2 Aplikace

Každá aplikace se skládá alespoň z jedné `Aktivity` (formálně se jedná o instanci třídy `Activity` a dále tuto značím velkým písmenem) a `layoutu`. `Aktivita` má za úkol správu uživatelské interakce s dotykovou obrazovkou zařízení, tudíž implementuje žádanou funkcionalitu aplikace. `Layout` definuje množinu objektů a jejich umístění na obrazovce, se kterými může uživatel interagovat. Je psán formou `XML` souboru. Objekty `layoutu` mohou být textová pole, tlačítka, spinnery a další komponenty.

[1] http://www.openhandsetalliance.com

Verze platformy	Jmenné označení	API	Podíl [18. 12. 2013]	Podíl [1. 5. 2014]
2.2	Froyo	8	1,6	1.0%
2.3.3 - 2.3.7	Gingerbread	10	24,1	16,2%
3.2	Honeycomb	13	0,1	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	18,6	13.4%
4.1.x	Jelly Bean	16	37,4	33.5%
4.2.x	Jelly Bean	17	12,9	18.8%
4.3	Jelly Bean	18	4,2	8.5%
4.4	KitKat	19	1,1	8.5%

Tabulka 2.1: Procentuální podíl verzí platforem Android[2]



Obrázek 2.1: Vztah mezi podtřídou `Activity` a `Layoutem`

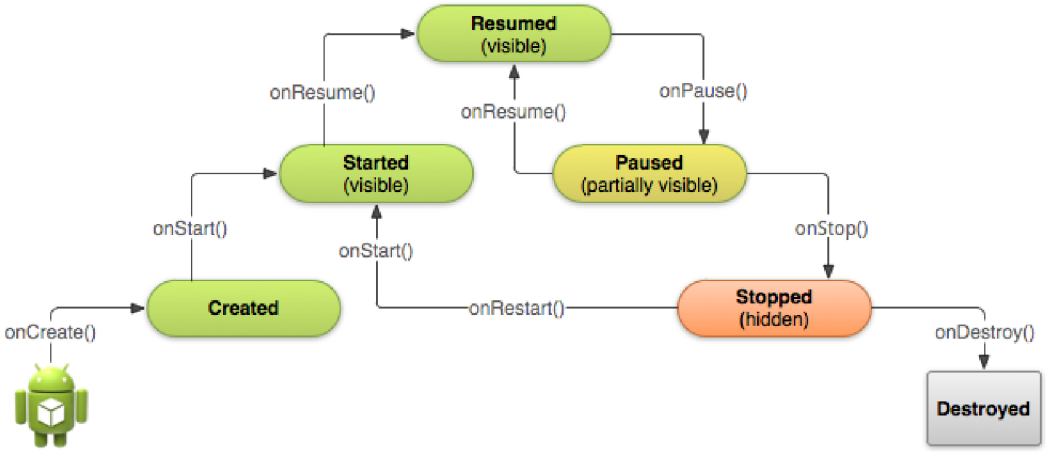
Jednoduchý příklad vzájemného propojení `Aktivity` a `layoutu` je znázorněn na obrázku 2.1, který je digitálně modifikován z [8, s. 25]. V okně, jehož vzhled a obsah popisuje layout souboru `activity.xml`, je definováno textové pole a dvě tlačítka. Po stisku tlačítka **True** je volána metoda `onclick` pro obsluhu jejího stisku, nacházející se ve třídě `Activity`. Metoda, která vykonává obsluhu stisku tlačítka, může být deklarována v souboru `xml` pomocí patřičného atributu komponenty nebo pomocí zpětného volání, které zaregistrujeme na komponentu tlačítka ve třídě *Activity*.

2.3 Aktivita

Příklad uvedený v kapitole 2.2 je velmi jednoduchý, avšak názorný. Ovšem v reálných aplikacích se vyskytuje vyšší počet Aktivit. Pro jejich správu využívá OS Android zásobník LIFO[2]. Pokud uživatel stiskne tlačítko zpět, bude zobrazena ta aktivita, která se nachází na vrcholu zásobníku. Při tomto procesu prochází aktivity jednotlivými stavy jejich životních cyklů, znázorněném na obrázku 2.2. Aktuálně se mohou nacházet v jednom ze čtyřech stavů [7, s. 79]:

- **Aktivní** (*Resumed*) – Aktivita se nachází na vrcholu zásobníku, přijímá vstupy od uživatele a patřičně reaguje svými výstupy. Pokud ke své činnosti potřebuje více zdrojů (operační paměť, výkon procesoru), operační systém postupně maže aktivity nacházející se ve spodních částech zásobníku tak, aby byl schopen zajistit požadavky pro právě aktivní.

[2]Last In, First Out



Obrázek 2.2: Životní cyklus aktivity[3]

- **Pozastavená** (*Paused*) – Aktivita byla vystřídána na vrcholu zásobníku jinou, avšak stále zůstává částečně viditelná. V tomto stavu nemůže přijímat pokyny od uživatele. Pouze v extrémních případech (značný nedostatek paměti) operační systém pozastavenou aktivitu smaže, přičemž tato situace nenastává často.
- **Zastavená** (*Stopped*) – Do tohoto stavu aktivita přechází ve chvíli, kdy není viditelná z žádné své části. Veškeré data s ní spojená nadále zůstávají uložena. Nicméně, pokud systém vyhodnotí nedostatek paměti, může tuto ukončit. Proto, při přechodu z pozastaveného stavu do zastaveného, je nutné explicitně uchovat veškeré hodnoty instančních proměnných. V opačném případě při jejím přechodu do stavu *Destroyed* a opětovném obnovení tato nebudou přístupná.
- **Neaktivní** (*Destroyed, Created*) – Aktivita byla smazána ze zásobníku. V případě znovuobnovení se všechna uložená data načtou z provedené zálohy, pokud byla vytvořena při přechodu z *pozastavená-zastavená*. Taktéž se může jednat o zcela nově vytvářenou aktivitu - ta se pak nachází ve stavu *Created*.

2.3.1 Metody

V rámci životního cyklu aktivity, přechody mezi jednotlivými stavy, jenž jsou znázorněny na obrázku 2.2, operační systém provádí zpětné volání metod[4, s. 72-75]:

- protected void onCreate(Bundle savedInstanceState) – Voláno při vytvoření, přičemž parametr je roven `null`. Pokud je aktivita restartována ze stavu *Destroyed*, parametr obsahuje uložený stav aktivity.
- protected void onStart() – Aktivita ještě není viditelná. Jedná se o nejvhodnější místo získání všech instancí zdrojů, jenž jsou třeba k jejímu běhu.
- protected void onRestart() – Funkcionalita je opačná metodě `onStop()`.
- protected void onResume() – Funkcionalita je opačná metodě `onPause()`.

[3] http://developer.android.com/training/basics/activity-lifecycle/starting.html

- protected void onPause() – Aktivita přechází do pozadí, nastává dealokace zdrojů a uložení provedených změn. Důležitá je rychlost provedení této metody, blokuje nově se načítající aktivitu.

- protected void onStop() – Používá se k další dealokaci zdrojů. Je však dobré mít na paměti situaci, kdy systém nemá dostatek zdrojů a násilně ukončuje aktivitu, tudíž metoda nebude provedena[4].

- protected void onDestroy() – Většina aplikací ji nevyužívá - veškerá dealokace se provádí v metodách `onPause` a `onStop`. Avšak, pokud byla vytvořena nová vlákna v metodě `onStart`, která by mohla potenciálně způsobit únik paměti, jedná se o správné místo k jejich zničení[5].

Přepisování[6] zmíněných metod a přidávání těmto nové funkcionality, je principem vývoje mobilních aplikací pro operační systém Android.

2.4 Fragment

Jedná se o volitelnou vrstvu mezi Aktivitou a Widgetem, přičemž Widgetem rozumíme část uživatelského rozhraní definovaného staticky v `xml` souboru (kapitola 2.2) nebo dynamicky za běhu programu - například tlačítko. Fragmenty byly představeny v roce 2011 ve verzi systému 3.0 - API 11. Pokud tyto chceme využívat ve verzích systému s API 4 až 10, musíme k jejich implementaci využívat metody z knihovny `Android Compatibility Library`. Pro starší zařízení již použitelné nejsou, avšak těch je dle kapitoly 2.1 v současné době využíváno nepatrné množství.

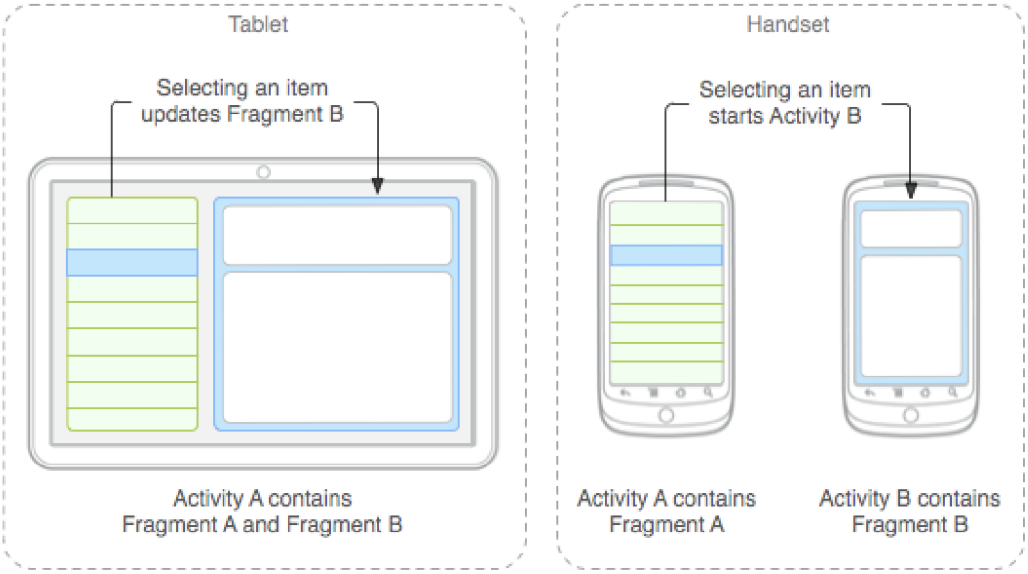
Fragmenty jsou výhodné v situaci, kdy má být aplikace přizpůsobena displayům zařízení, které se příliš odlišují svou velikostí. Jako příklad mohu uvést mobilní telefon s úhlopříčkou 4” a tablet, jenž má podstatně větší úhlopříčku, a to 10”. Z toho plyne, že na tabletu je k dispozici více místa, které je vhodné zaplnit, a tím využít výhodu ve formě méně kroků provedených uživatelem, potřebných k získání požadovaných informací. Názorný příklad znázorňuje obrázek 2.3.

Dále předpokládám názornost na příkladu zákazníka, kdy na obrázku vlevo, znázorňující situaci pro tablet, je v Aktivitě *A* dostupný jejich seznam. Pokud uživatel vybere jméno, zobrazí se napravo od seznamu detail zákazníka. Toto chování se odehrává pouze v jedné Aktivitě, která spravuje dva Fragmenty. Jiné chování nastává v případě mobilního telefonu s menší velikostí displaye, které je znázorněno na obrázku vpravo. Aktivita *A* obsahuje taktéž seznam zákazníků, avšak již není dostatek potřebného místa pro zobrazení detailu zákazníka, jako je tomu v případě tabletu. Proto po akci vybrání se musí spustit Aktivita *B*, která patřičný detail zobrazí. Tuto funkcionalitu lze implementovat i bez využití fragmentů. To má však za následek velké množství redundantního kódu a s tím související problémy. Mnohem výhodnější se jeví využití fragmentů, kdy je seznam zákazníků umístěn ve Fragmentu *A* a jeho detail ve Fragmentu *B*.

Mezi další výhody fragmentů lze zařadit:

- Schopnost modifikace uživatelského rozhraní za běhu.

[4] http://developer.android.com/reference/android/app/Activity.html#onStop()
[5] http://developer.android.com/training/basics/activity-lifecycle/starting.html
[6] Overriding
[7] http://developer.android.com/guide/components/fragments.html



Obrázek 2.3: Využití fragmentů[7]

- Fragmenty přijímají vstupy od uživatele a mohou na ně reagovat stejně, jako aktivity (kapitola 2.3).
- Znovupoužitelnost – tato vyplývá z výše uvedeného příkladu.
- Vnořování – fragmenty se mohou vnořovat, vnořený je ovládán svým „rodičem“.
- Vlastní životní cyklus – mnohem komplikovanější, než je tomu u životního cyklu Aktivity. Pro čtenáře, v případě zájmu, mohu doporučit [6, s. 233 - 259].

Aplikace, která je předmětem této práce, staví na principu využití fragmentů. Jejich další výhody či nevýhody vyplynou z kapitoly implementace.

2.5 Projekt

Každý projekt, který se využívá pro programování aplikací pro OS Android, musí mít náležitou adresářovou strukturu. Ta, mimo objektově orientovaného paradigmatu jazyka Java, zajišťuje rozdělení částí definice vzhledu uživatelského rozhraní a funkcionality, což je výhodné zejména při týmové práci. Základní výčet souborů a adresářů projektu následuje[9, s. 54][5, s. 59-60]:

- /src – Hlavní zdrojové soubory v jazyce Java ve zvoleném jmenném prostoru. Po vytvoření projektu obsahuje instanci třídy `Activity`, jenž je volána při spuštění aplikace.
- /res – Prostředky přibalené ke zkopilovanému kódu aplikace.
- /res/drawable – Sada adresářů obsahující soubory (obrázky ikon ve formátu PNG aj.) vykreslené na obrazovkách s různými druhy rozlišení. Příponou adresáře může být upřesněno, pro který typ obrazovky je soubor určen - např. zdroj umístěný v adresáři drawable-ldpi se vykreslí pouze na obrazovce se středním rozlišením. Výhodou třídění

do podsložek je urychlení běhu aplikace, kdy OS nemusí přepočítávat velikost obrázku, ale pouze vybere aktuálně nejvhodnější.

- /res/layout – Soubory definující rozvržení uživatelského rozhraní. Také může existovat s příponou - např. layout-large obsahuje rozvržení pro zařízení s úhlopříčkou od 7".
- /res/values – `XML` soubory obsahující hodnoty užitých řetězců, čísel a barev. Pokud vyžadujeme lokalizaci ve více jazycích, využijeme `/res/values-cs`, ve které je obsažen soubor `strings.xml` definující, v tomto případě, textové řetězce pro českou lokalizaci aplikace.
- /res/menu – Adresář obsahuje `XML` soubory definující vlastnosti MENU aplikace (vzhled, obsah či podmenu).
- /libs – Adresář s knihovnami ve formátu JAR.
- /bin – Adresář překladu.
- /assets – Může být použito pro úložiště souborů, jejichž název nebude po kompilaci změněn a budou umístěny do výsledného `.apk` souboru. Ty pak můžeme procházet pomocí `AssetManager`. Tohoto se využívá, pokud chceme ukládat textury.

AndroidManifest

Důležitý souborem je `AndroidManifest.xml`, umístěný v kořenové složce aplikace, pomocí kterého jsou definovány informace, jenž musí být přístupné před spuštěním aplikace. Obsahuje strukturu a `metadata` (např. informace o umístění hlavní ikony, použité schémata zobrazení) aplikace, její komponenty, požadavky a v základu dále:

- Unikátní jméno balíčku – jednoznačně identifikující aplikaci, verzi kódu, která je důležitá pro službu `Google Play` při rozlišování publikovaných verzí aplikace pro poskytování aktualizací uživatelům.
- Podporovanou minimální a cílovou verzi `API`. Druhá odpovídá verzi, na které byla aplikace vyvíjena a testována. Pokud uživatel aplikaci spustí na zařízení, jenž má cílovou verzi vyšší než deklarovanou v souboru `AndroidManifest`, bude zapnut režim zpětné kompatibility. Proto je doporučován vývoj a testování aplikace na nejnovější verzi `API`.
- Hardwarové požadavky – kamera, více dotykový displaye, požadavek na specifickou klávesnici zařízení.
- Práva, která aplikace potřebuje pro bezproblémový chod - např. přístup k informaci o aktuální lokaci uživatele, přístup k SMS, zápis souborů do externího úložiště.
- Deklarace hlavní Aktivity, která bude spuštěna při startu aplikace a všech ostatních včetně jejich požadavků.
- Aplikace třetích stran vyžadované pro běh aplikace.

Výše zmíněný výčet není konečný, avšak pro vyvíjenou aplikaci dostačující. Bližší informace jsou dostupné na webových stránkách Android Developers[8].

[8] http://developer.android.com/guide/topics/manifest/manifest-intro.html

2.6 Uchování dat

Většina aplikací pracuje, i v omezené míře, s daty, které potřebuje uchovávat po celou dobu své existence. Například se může jednat o uživatelské nastavení, ve kterém je možné zvolit výchozí hodnoty textových formulářů, přihlašovací údaje ke vzdálené službě, emailovou adresu, na kterou si uživatel přeje zasílat důležité informace, nebo se může jednat o modifikaci celkového vzhledu (velikost a typ písma). Existuje však nemalý počet aplikací, které pro svou činnost vyžadují ukládání většího počtu dat ve formě textových údajů, fotografií a multimédií. OS Android poskytuje pro všechny výše zmíněné požadavky mechanismy, jak data ukládat a manipulovat.

2.6.1 Key-Value Sets

V překladu množina klíč-hodnota, implementována třídou `SharedPreferences`[9]. Tato poskytuje základní metody pro správu a manipulaci těchto hodnot, ty však mohou nabývat pouze primitivních datových typů - *boolean, float, integer, long, string*. Od verze (viz. kapitola 2.1) API 11 byla přidána podpora pro předávání kolekce hodnot datového typu *string*. Při vytváření se volí mód - privátní nebo veřejný. V případě veřejného se jedná o jeden z možných způsobů mezi-aplikační výměny informací (v případě, že oběma aplikacím je znám název množiny). Ukládání dat je vnitřně prováděno do souboru.

2.6.2 Soubory

Platforma Android poskytuje práci se soubory, které mohou být uloženy v úložišti jednoho ze dvou typů[3]:

(a) interní úložiště (Internal storage),

(b) externí úložiště (External storage).

Interní úložiště zajistí přístup k souborům v každém časovém okamžiku výhradně a pouze naší aplikací (pro sdílení interních souborů s ostatními aplikacemi se dá docílit využitím vhodných mechanismů). Dle povahy souboru máme na výběr ze dvou možností uchování, a to dočasné nebo perzistentní. U dočasného (`cache` - až 1MB) musíme počítat s okamžikem nedostatku paměti, ve kterém operační systém automaticky maže dočasné soubory aplikacím, a to libovolně a bez upozornění. Pokud se uživatel rozhodne aplikaci odinstalovat, systém se postará o odstranění veškerých perzistentně uložených dat spjatých s aplikací.

Externí úložiště bývá nejčastěji realizováno formou výměnné karty MicroSD[10]. Z toho vyplývá, že nemusí být vždy přístupné (uživatel kartu vyjme nebo připojí zařízení k počítači), data zde uložená jsou přístupná všem ostatním aplikacím. Výhodnou vlastností oproti internímu úložišti bývá jeho vyšší kapacita. Pro externí úložiště se rozhodneme ve chvíli, pokud aplikace pracuje s velkým množstvím dat ve formě souborů, avšak s patřičným ohledem na jejich bezpečnost. Náš záměr musíme deklarovat formou přístupových práv v souboru `AndroidManifest` (kapitola 2.5) - právo pro čtení je implicitní, pro zápis jej musíme uvést explicitně. Pokud zařízení nevlastní paměťovou kartu, je využita část interního úložiště, kterou systém označí jako externí.

[9] http://developer.android.com/reference/android/content/SharedPreferences.html
[10] Flash paměťová karta odvozená od staršího formátu SD.

2.6.3 SQLite

Jedná se o relační databázový systém využívající dialekt jazyka SQL. Podporuje téměř celý standart SQL-92, kromě následujícího[11]:

- Částečné vnější spojování pravé (*right*) a úplné (*full*).
- Příkaz *alter table* pro změnu struktury tabulky – smazání sloupce (*drop column*), rozšíření sloupce (*alter column*), přidání omezení (*add constraint*).
- Omezená práce s *triggery*, pohledy (*views*).

I přes zmíněná omezení se jedná o ideálního kandidáta pro aplikace nevyžadující vzdálené ukládání dat na databázovém serveru. Jedná se o multiplatformní knihovnu, napsanou v jazyce *C*, šířenou pod licencí `public domain`, díky čemuž může být využita v softwaru určeného pro komerční účely.

SQLite databáze zajišťuje možnost současného vícenásobného čtení a maximálně jeden zápis v každém časovém okamžiku. Její transakce splňují vlastnosti ACID - atomičnost, konzistenci, izolovanost a trvanlivost veškerých změn provedených v databázi. Není spuštěna v samostatném procesu, protože se jedná o knihovnu. Proto při čtení či zápisu velkého množství dat je vhodné zmíněné operace provádět v samostatném procesu nebo vlákně[1].

Každý databázový systém využívá statickou typovou kontrolu. Datový typ vkládané hodnoty je porovnáván s datovým typem přiřazeným danému sloupci právě modifikované tabulky. Tím je zaručena kontrola, potažmo povolení nebo zamítnutí prováděné operace. U SQLite systému je tomu jinak. Vlastností samotné hodnoty je jeho datový typ, ale ne hodnota datového typu sloupce tabulky, do které se ukládá nebo čte samotná hodnota. Toto tvrzení umožňuje uložit do sloupce tabulky, který označíme při jejím vytváření datovým typem *integer* jakoukoli hodnotu libovolného datového typu - například textový řetězec, jehož datový typ chápeme jako *string*. Jedinou vyjímku tvoří sloupce tabulek, které při vytváření označíme jako *integer primary key*. Tyto mohou nabývat pouze celočíselných hodnot[12].

Operační systém Android poskytuje plnou podporu pro práci s SQLite databází, soubor s daty ukládá do perzistentního interního úložiště (kapitola 2.6.2). Počet databází není omezen a každá z nich je uchována v jediném souboru s příponou `dbm`. Doporučení pro manipulaci s databází je využití třídy `SQLiteOpenHelper`, zajišťující její pohodlné vytváření a otevírání.

[11] http://www.sqlite.org/omitted.html
[12] http://www.sqlite.org/datatype3.html

Kapitola 3

Analýza existujících řešení

V kapitole budou postupně popsány již existující aplikace pro servisní techniky dostupné pro platformu Android, které poskytují přímo výrobci plynových zařízení nebo jejich smluvní partneři. Nejedná se o úplný výčet všech dostupných aplikací od veškerých společností zabývajících se výrobou či distribucí plynových zařízení. Následující byly vybrány z hlediska dostupnosti – není třeba vlastnit přihlašovací údaje servisního technika a ceny – nabízeny ke stažení zdarma.

3.1 Viessmann Spare Part App

Název aplikace pro českou lokalizaci zní „Náhradní díly“, dostupná prostřednictvím Google Play[1]. Umožňuje vyhledávat v katalogu náhradních dílů pro část výrobků firmy `Viessmann`[2], a to pomocí `fulltext` vyhledávání dle názvu náhradního dílu, jeho čísla nebo pomocí čárového kódu umístěného na štítku náhradního dílu, načteného prostřednictvím fotoaparátu. Rovněž poskytuje historii vyhledávání a nápovědu. Uživateli je k dispozici funkce vytváření seznamů, do kterých je umožněno vkládat vyhledané položky. Aplikace umožňuje kontrolu kompatibility daného dílu se spotřebiči. Po přihlášení uživatele, které může provést pouze certifikovaný servisní technik, je k dispozici technická dokumentace ve formátu `pdf`. Ukázku aplikace znázorňuje obrázkem 3.1(a).

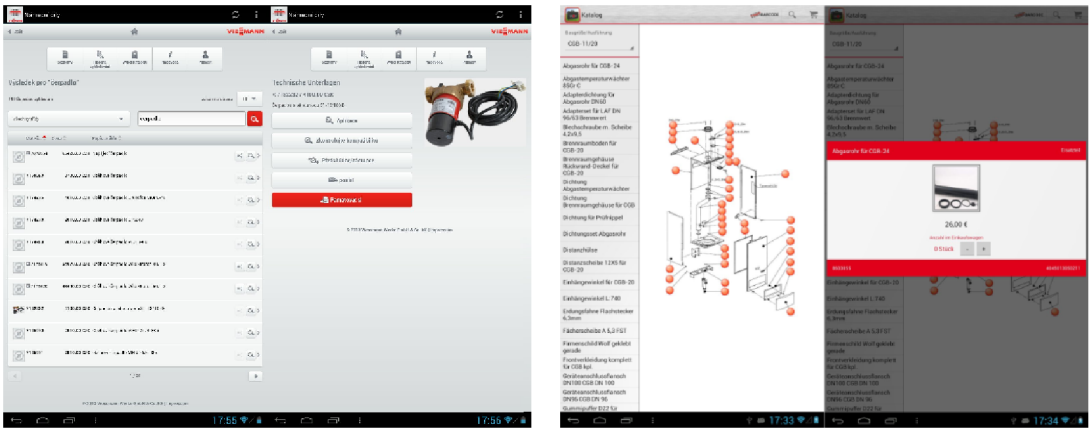
Pro čtení čárových kódů je využita externí aplikace `Barcode Scanner`[3]. Tato umožňuje dekódovat informace obsažené v čárových a QR kódech.

Aplikace je dostupná v českém jazyce, avšak pouze s částečnou lokalizací. Nevýhodou je nutnost pracovat pouze s dostupným internetovým připojením. Navštívené položky se neukládají do žádné vyrovnávací paměti. Tento způsob řešení má za následek vysoký datový přenos a při špatné kvalitě internetového připojení pomalé odezvy aplikace.

3.2 Wolf Ersatzteile

Katalog náhradních dílů pro výrobky společnosti `Wolf`[4]. Podobně jako v předchozí kapitole 3.1, je k dispozici `fulltext` vyhledávání jak náhradního dílu, tak samotného výrobku nebo skenování čárového či QR kódu umístěného na štítku komponenty zařízení. Pokud

[1] http://play.google.com/store/apps/details?id=com.viessmann.etapp
[2] http://www.viessmann.com/com/en.html
[3] http://play.google.com/store/apps/details?id=com.google.zxing.client.android
[4] http://www.wolf-heiztechnik.de/cz/pkp/home.html



(a) Viessmann Spare Part App (b) Wolf Ersatzteile

Obrázek 3.1: Uživatelské rozhraní analyzovaných aplikací.

štítek chybí a nejsme schopni určit typ komponenty, můžeme využít postupu, kdy pomocí `fulltext` vyhledávání nalezneme zařízení, přičemž se automaticky zobrazí jeho detailní schéma (obrázek 3.1(b) vlevo), ve kterém lze komponentu snadno identifikovat. Ihned může být vložena do nákupního košíku (obrázek 3.1(b) vpravo). Objednání probíhá pomocí automaticky zaslaného emailu dodavateli.

Výhodnou vlastností je možnost práce bez internetového připojení (mimo funkci objednávání), veškeré soubory potřebné pro běh aplikace jsou staženy při instalaci. Aplikace vyniká intuitivním ovládáním, což dokládá i hodnocení uživatelů na stránce věnované aplikaci[5]. Nevýhoda spočívá v existenci pouze německé lokalizace.

3.3 Shrnutí

V kapitole 3 byly popsány aplikace poskytované dvěma výrobci plynových zařízení. Jak již bylo zmíněno, nejedná se o úplný výčet dostupných aplikací pro platformu Android, avšak pokud některý z výrobců tuto poskytuje, její užívání a funkce jsou postaveny na podobných principech popsaných v kapitolách 3.1 a 3.2. Aplikace podporují funkcionalitu, jako databázi výrobků a s nimi spojené komponenty, schémata komponent, komfortní vyhledávání, skenování čárových či `QR` kódů. Každý z výrobců poskytuje svou vlastní aplikaci, neexistuje žádná, která by tyto sjednocovala.

Byl zjištěn nedostatek - *Viessmann Spare Part App* pracuje, podobně jako většina ostatních, pouze s dostupným internetovým připojením, což se může jevit jako velmi omezující podmínka. Uživatel musí vlastnit mobilní telefon s patřičným datovým tarifem. Taktéž se může nacházet v místě bez pokrytí signálu `GSM` nebo `WIFI`, a proto je aplikace v danou chvíli nepoužitelná. Tato skutečnost klade důraz vytvořit aplikaci, která bude schopna pracovat bez dostupného internetového připojení.

Žádná ze zkoumaných aplikací nepodporuje databázi zákazníků, historii oprav plynových zařízení a některé z dalších funkcionalit popsané v kapitole 4.2. Tímto je zajištěno vytváření softwaru, který jsem na trhu dostupných aplikací pro operační systém Android nenalezl.

[5] http://play.google.com/store/apps/details?id=com.wolf_heiztechnik.replacement_parts

Kapitola 4

Návrh aplikace

Samotné etapě návrhu aplikace předcházela fáze specifikace požadavků, ve které jsem použil některé z metod pro získávání informací od uživatele:

(a) analýza existujícího softwarového systému,

(b) pozorování prací u zákazníka,

(c) interview.

Metodě (a) byl věnován prostor v kapitole 3. Využitím (b) jsem byl schopen odhalit část slabých míst v pracovním postupu servisního technika, ve kterých může být vytvářená aplikace nápomocna. Použitím metody (c), společně s informacemi získanými pomocí (b), bylo cílem vytvořit slovní popis, který by co možná nejpřesněji charakterizoval požadavky kladené na datovou strukturu aplikace a seznam funkčních požadavků.

Po získání potřebných informací jsem určil způsob ukládání dat a provedl návrh uživatelského rozhraní.

4.1 Pracovní postup servisního technika

Ohlášení servisního úkonu (fáze *I.*) zákazníkem probíhá pomocí telefonního hovoru nebo emailovou formou. Získávají se údaje o jménu a příjmení zákazníka, adresa bydliště, kontaktní informace, výrobce a produkt plynového zařízení. Pokud je ohlašována závada, zákazník specifikuje, jak se tato projevuje. Dále je dohodnuto datum provedení servisního zásahu. V případě ohlášení pomocí emailové korespondence, zákazník v naprosté většině nedoloží popsané informace, proto servisní technik obratem kontaktuje klienta telefonním hovorem.

Před datem provedení může nastat fáze přípravy (*II.*). Servisní technik, v případě pochybností, studuje dokumenty k plynovému zařízení. Pokud si je jist příčinou závady, objednává od dodavatele náhradní díly, které využije při budoucím servisním úkonu.

Fáze (*III.*) začíná dnem provedení sjednaného servisního zásahu. Pokud se servisní technik dostane do časové tísně z důvodů zpoždění při provádění servisních úkonů u předcházejících zákazníků, kontaktuje následující klienty s požadavkem na pozdější obsloužení. V opačném případě se dostaví na dohodnutou adresu bydliště a započíná samotný úkon. Během jeho provádění lokalizuje a opravuje případné závady, studuje dokumenty a eventuálně mění náhradní díly. V případě, kdy potřebný díl nevlastní a tento musí být objednán,

ukončuje servisní zásah a celý proces se po jeho doručení navrací k fázi *II.* společně s budoucím kontaktováním zákazníka, které je patrné ve fázi *I.* Mohou nastat situace, kdy případnou opravu, či jiné záležitosti, potřebuje konzultovat s technickou podporou poskytovanou výrobcem plynového zařízení.

V poslední fázi (*IV.*) probíhá závěrečné nastavení zařízení dle parametrů udávaných výrobcem v dokumentaci. Poté nastává seznámení zákazníka s prováděnými úkony a definování způsobu platby, případně její okamžité provedení.

4.1.1 Rozbor pracovního postupu

Výsledná aplikace si klade za cíl zjednodušit a zrychlit všechny pracovní fáze popsané výše.

I. Při ohlašování servisního úkonu v případě, kdy již servisní technik v minulosti prováděl úkony na plynovém zařízení zákazníka, si nemusí znovu značit potřebné identifikační údaje, jelikož ty má uloženy z dřívější doby. Celý proces fáze *I.* se tím výrazně zrychlí. Pokud klient ohlašuje závadu, kterou má technik poznačenou v aplikaci jako vyřešenou a opravu zvládne nekvalifikovaná osoba, může na základě této historie poradit zákazníkovi s postupem pro její vyřešení bez nutnosti servisního zásahu technika.

II. V případě opakující se závady na plynovém zařízení technik objednává stejné náhradní díly, které použil v minulosti. Odpadá tím nutnost případného studia dokumentů.

III. Budování znalostní báze, jenž může být využita během provádění jednotlivých fází.

IV. Není nutná celková kalkulace, jelikož ji aplikace provede automaticky.

Mezi další výhody mohu zařadit fakt, kdy veškeré podpůrné materiály má servisní technik přístupné v jediné aplikaci, čímž se eliminuje nevýhoda popsaná v kapitole 3 – každý výrobce poskytuje svou vlastní aplikaci (pokud ji vůbec poskytuje). Na základě znalostní báze může servisní technik zákazníkovi doporučit koupi nového zařízení, kdy v průběhu jeho životnosti částky za údržbu překročily nepřípustnou mez. Taktéž v případě opakujících se závad bude známo jejich řešení.

4.2 Specifikace požadavků

Na základně diskusí provedených s budoucími uživateli s ohledem na informace obsažené v kapitole 4.1, byl vytvořen text specifikace požadavků, který definuje strukturu dat a funkční požadavky kladené na výslednou aplikaci. Následuje vytvořený text:

Zákazník může být fyzická nebo právnická osoba, o které bude možno uložit následující údaje: Jméno, příjemní, adresa bydliště a typ objektu, který je vázán s adresou bydliště - rodinný dům, činžovní dům, malý dům, budova společnosti, budova instituce. Dále telefonní číslo, emailovou adresu, číslo bankovního účtu. Zákazník může vlastnit více plynových zařízení. U každého plynového zařízení je nutné uchovávat informace o výrobním a objednacím čísle (každý výrobce uvádí minimálně výrobní číslo, u kterého se nepředpokládá jedinečnost napříč výrobci), typ paliva - zemní plyn, propan butan, elektřina. Dále, zda si zákazník přeje být informován servisním technikem o pravidelných servisních prohlídkách s možností volby data, na základě kterého má být servisní prohlídka provedena a datum uvedení do provozu.

Dále každému plynovému zařízení bude možno přiřadit jeden z množiny QR kódů pro jeho jednoznačnou identifikaci.

Plynová zařízení jsou vyráběna výrobci, jejich výrobek se nazývá produktem. Každý výrobce může vyrábět více typů produktů a může poskytovat servisní podporu. O výrobci bude možno zadat název a umístění jeho webové prezentace. Servisní podpora, vázána na konkrétního výrobce, může být jedna ze tří kategorií: hlavní servisní technik, náhradní díly, obchodní zástupce. U servisní podpory bude možno zadat jméno a příjmení osoby, která ji poskytuje, telefonní číslo a emailovou adresu.

Produkt spadá do jedné ze dvou kategorií - kondenzační či nekondenzační. O produktu bude možno uložit informace o jeho názvu a výkonu.

Ke každému produktu bude existovat možnost přiřadit dokumenty, nacházející se uložené v zařízení na libovolném místě, ve formátu *pdf*, *png*, *jpg*. Dokument může být servisní manuál, uživatelský manuál nebo fotografie.

Na plynovém zařízení jsou prováděny servisní úkony. Servisní úkon může být jeden ze tří kategorií - oprava, roční prohlídka, uvedení do provozu. Servisní úkon bude obsahovat datum provedení, datum splatnosti, sazbu DPH, cenu cestovného bez DPH, cenu provedené práce bez DPH, celkovou cenu použitých náhradních dílů bez DPH (cena jednotlivých náhradních dílů se může měnit), celkovou cenu bez DPH a celkovou cenu s DPH. Typ platby za servisní úkon (hotově nebo převodem, v případě druhém může být uloženo číslo faktury). Servisní úkon může být označen jako splněný či nesplněný. Při každém servisním úkonu může být vyměněn náhradní díl či náhradní díly. Každý servisní úkon může obsahovat nalezené závady. Závada obsahuje svůj popis a případně možnost popisu jejího odstranění. Servisní úkon může být záruční nebo pozáruční.

Náhradní díl má následující údaje: katalogové číslo, název, cena bez DPH, EAN kód. Náhradní díl poskytuje výrobce pro všechny typy plynových zařízení. Je vyžadováno, aby aplikace pracovala v tzv. `offline` režimu - bez nutnosti připojení k síti Internet.

4.2.1 Funkční požadavky

Aplikaci využívá v každém časovém okamžiku pouze jeden uživatel - servisní technik (aktér), kterému je umožněno na základě rozboru fáze specifikace požadavků (4.2) provádět následující úkony:

1. přidat, upravit, smazat, zobrazit **výrobce**;
2. přidat, upravit, smazat, zobrazit **produkt**, vázaný k výrobci;
3. přidat, smazat, zobrazit **dokument**, vázaný k produktu;
4. přidat, upravit, smazat, zobrazit **náhradní díl**, vázaný k výrobci;
5. přidat, upravit, smazat, zobrazit **servisní podporu**, vázanou k výrobci;
6. přidat, upravit, smazat, zobrazit **zákazníka**;
7. přidat, upravit, smazat, zobrazit **plynové zařízení**, vázané k zákazníkovi;
8. přidat, upravit, smazat, zobrazit **servisní úkon**, vázaný k plynovému zařízení;
9. přidat, smazat, zobrazit **závadu**, vázanou k servisnímu úkonu;
10. přidat, smazat, zobrazit **použitý náhradní díl**, vázaný k servisnímu úkonu.

Operace nad daty

Na základě získané znalostní báze aplikace umožňuje následující operace nad daty:

(a) Vyhledání plynového zařízení dle přiřazeného QR kódu.

(b) Vyhledání náhradního dílu dle přiřazeného EAN kódu.

(c) Generování výpisu neuzavřených servisních úkonů.

(d) Generování výpisu nejbližších ročních prohlídek.

(e) Zobrazení historie veškerých servisních úkonů, které jsou vázány k danému plynovému zařízení.

Každému plynovému zařízení, dle specifikovaných požadavků na začátku kapitoly 4.2, může být přidělen QR kód pro jeho jednoznačnou identifikaci, aby byla zajištěna funkčnost operace (a). Proto musí být zvoleno vhodného mechanismu získání množiny připravených QR kódů, kterou si bude moct uživatel vytisknout a dále s ní pracovat.

Hodnota DPH, se kterou se pracuje v servisních úkonech, musí být předdefinovatelná, aby ji uživatel nemusel při každém novém servisním úkonu znovu vyplňovat. Proto bylo implementováno *nastavení*, ve kterém může být operace definování výchozí hodnoty provedena. Definovaná hodnota se zobrazí u nově vytvářeného servisního úkonu, u editovaného však zůstává hodnota původní.

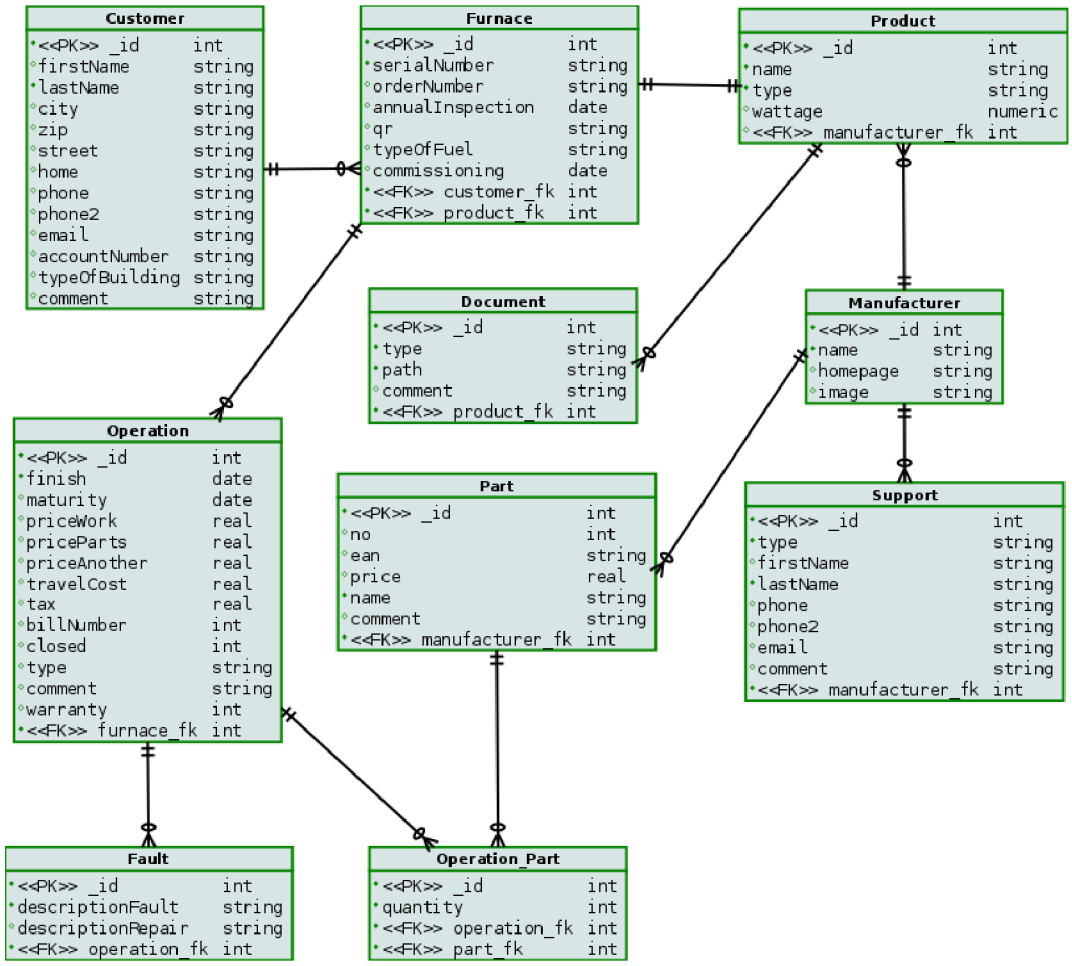
4.3 Uložení dat

Na základě specifikace požadavků uvedené v kapitole 4.2 je zřejmé, že navrhovaná aplikace musí manipulovat s velkým množstvím dat ve formě textových řetězců a číselných hodnot. Nabízí se dva způsoby. Prvním může být uložení serializovaných dat do souboru (kapitola 2.6.2), například pomocí značkovacího jazyka XML[1], a v případě potřeby jejich opětovného získání využít deserializaci. Druhý způsob může být použití SQLite databáze, se kterou operační systém Android dokáže pracovat (více v kapitole 2.6.3). Rozhodl jsem se právě pro tento z důvodů komfortnější implementace a předchozích zkušeností s databázemi, se kterými se pracovalo v některých školních projektech. Taktéž je potřeba provádět zpracování dat, které plyne nejen z požadovaných operací vyjmenovaných v kapitole 4.2.1. V případě využití prvního způsobu by byla implementace této funkcionality značně složitá. Dokumenty budou ukládány do externího úložiště zařízení. Důvod tohoto řešení vyplývá z principu, který je uveden v kapitole 2.6.2.

Obrázek 4.1 znázorňuje strukturu navržené SQLite databáze. Po dodatečné konzultaci s uživatelem byl přidán do vybraných tabulek sloupec `comment` pro možnost uložení libovolného textového komentáře. Tento požadavek není ve specifikaci až na jednu vyjímku uveden. Sloupce, u kterých je zakreslena plná kružnice, nemohou nabývat prázdné hodnoty (NULL). Opačně je tomu v případě kružnice nevyplněné.

Názvy tabulek a jejich sloupců korespondují k informacím uvedeným v textu specifikace, taktéž jejich datové typy. Avšak pro pochopení stojí za zmínku uvést několik poznámek k vybraným tabulkám navržené databáze:

[1] Extensible Markup Language



Obrázek 4.1: Relační model databáze

- Customer – slouží pro uchování informací o zákazníkovi. Veškeré sloupce (dále ve výčtu uvažuji sloupce bez primárního klíče) nabývají hodnot datového typu *string*. Například telefonní číslo, phone a phone2, může být ve formě prostých číslic, avšak tato varianta by neumožňovala uložení čísla v mezinárodním formátu. Číslo bankovního účtu taktéž obsahuje znaky, které nejsou číselné - pomlčka oddělující předčíslí, lomítko pro specifikaci kódu banky. Podobná situace nastává v tabulce Support.

- Document – obsahuje sloupec path. Uchovává hodnotu úplné cesty k místu uložení dokumentu v zařízení.

- Operation – sloupce closed, warranty, billNumber nabývají datového typu *integer*, přičemž první dva jsou využity k uchování hodnoty datového typu *boolean*. Ten však není podporován databází SQLite. Proto je využito zmíněného datového typu, kdy hodnotu **1** uvažuji jako *True*, podobně **0** jako *False*. Do třetího sloupce v pořadí se ukládá číslo faktury. Pokud uživatel označí metodu platby jako „hotově“, je zde uložena hodnota *False*, resp. **0**.

- Operation_Part – Při servisním úkonu může být vyměněn žádný, právě jeden, ale i více dílů. Díl může být použit v žádném, právě jednom, ale i ve více servisních

úkonech. Tvrzení odpovídá vztahu M:N mezi tabulkami Part a Operation. Proto musí být využito pomocné vazební tabulky.

4.4 Uživatelské rozhraní

Grafické uživatelské rozhraní je primárně zaměřeno na přístroje vlastnící display s velikostí úhlopříčky alespoň 7", jenž v textu souhrnně nazývám pojmem „tablety". Abych cílil na vyšší počet uživatelů, rozhodl jsem se i pro podporu zařízení s menší úhlopříčkou, v textu téže označovaných jako „mobilní telefony". Návrh uživatelského rozhraní pro tablet je znázorněn pomocí obrázku 4.2(a), kde jednotlivé číslice zastupují:

I. Název aktuální sekce, ve které se uživatel nachází.

II. Povolené akce v této sekci - přidání nového záznamu nebo uložení nově přidávaného.

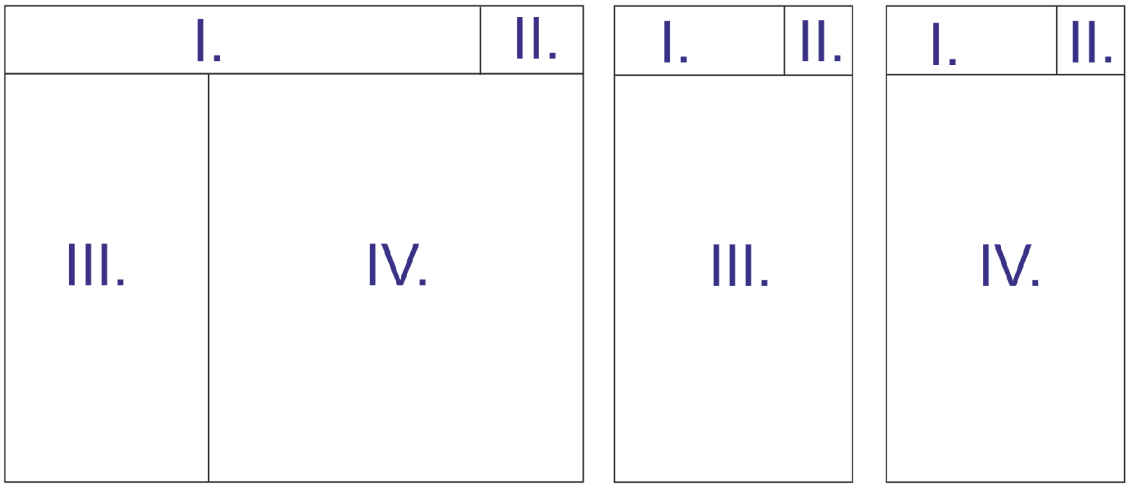
III. Část uživatelského rozhraní zobrazující:

   (a) Seznam uložených záznamů, pak část *IV.* zobrazuje detail vybraného záznamu.

   (b) Souhrnné informace z předcházejících sekcí, poté část *IV.* zobrazuje seznam prvků společně s jejich detailními informacemi.

   (c) Požadované informace během přidávání či úpravě záznamu. Části *III.* a *IV.* budou sloučeny.

Uživatelské rozhraní navržené pro mobilní telefony znázorňuje obrázek 4.2(b), ve kterých *I.* a *II.* plní stejnou funkci jako v případě tabletu, *III.* zobrazuje seznam uložených záznamů. Pokud bude libovolný vybrán, část *IV.* zobrazí jeho detail. Tímto návrhem bude uživatel mobilního telefonu ochuzen o zmíněné souhrnné informace, pro které není dostatek místa, avšak tyto nejsou pro používání aplikace zcela nezbytné. Slouží pro zobrazení informací z předcházejících kroků, které uživatel provedl.



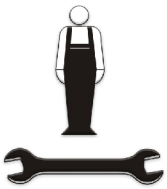
(a) Tablet (b) Mobilní telefon

Obrázek 4.2: Návrh rozvržení uživatelského rozhraní

Kapitola 5

Implementace

Dle zadání bakalářské práce byla aplikace zveřejněna prostřednictvím online distribuční služby **Google Play Store** společnosti Google v sekci *Nástroje* ve dvou jazykových lokalizacích - angličtina a čeština. V první variantě byl zvolen název **Boiler Servicemen**, ve druhé **Opravář servisních kotlů**. Na obrázku 5.1 je znázorněna ikona aplikace. Při jejím grafickém návrhu byl kladen důraz na jednoduchost, aby v seznamu aplikací byla od ostatních odlišná tzv. „na první pohled“. Aplikace je dostupná zdarma, bez žádných vstupních poplatků.



Obrázek 5.1: Ikona aplikace Boiler Servicemen

Aplikace je naprogramována v programovacím jazyce Java s využitím patřičných knihoven pro vývoj pod operačním systémem Android. Cílová verze API (kapitola 2.1) je rovna hodnotě 18, minimální podporovaná hodnotě 13. Těmito volbami, ke dni 1. května 2014, podporuji 82,8% zařízení využívající službu **Google Play**. K procentuálnímu výsledku se dojde prostým sečtením patřičných hodnot z tabulky 2.1.

5.1 Architektura

Programové řešení je rozděleno do celkového počtu sedmi balíčků[1], ve kterých podřetězec „cz.podola.furnanceservicemenapp“ značí jmenný prostor[2] aplikace. Následuje výčet balíčků:

(a) cz.podola.furnanceservicemenapp – hlavní jádro aplikace starající se o správu uživatelského rozhraní.

(b) cz.podola.furnanceservicemenapp.db.adapter – adaptéry sloužící pro přístup k jednotlivým tabulkám SQLite databáze znázorněné obrázkem 4.1. Taktéž obsahuje třídu

[1] Anglický název - packages
[2] Anglický název - namespace. Důležitá je jeho unikátnost v rámci služby Google Play.

DBA, jenž je zodpovědná za vytvoření a naplnění části databáze během prvního spuštění aplikace po její instalaci. Pokud v budoucnu bude pozměněno schéma databáze, právě tato nabízí mechanismy, kterými lze požadované změny aplikovat.

(c) cz.podola.furnanceservicemenapp.db.model – obsahuje třídy reprezentující modely, ve smyslu jednotlivých sloupců, tabulek databáze. Tyto využívají třídy uživatelského rozhraní a adaptéry přistupující k databázi. Slouží pro pohodlné předávání a získávání dat z jednotlivých tabulek databáze.

(d) cz.podola.furnanceservicemenapp.files – vlastní jedinou třídu *Files*. Zjišťuje stav externího úložiště, kopíruje, maže a otevírá z něj soubory, resp. dokumenty.

(e) cz.podola.furnanceservicemenapp.qr – v balíčku existuje pouze jediná třída, *Scanning*. Tato získává řetězec jednoznačně identifikující plynové zařízení, který se nachází zakódován ve vygenerovaném a posléze načteném QR kódu.

(f) cz.podola.furnanceservicemenapp.adapter – obsažené využívají třídy uživatelského rozhraní odvozené od třídy *ListFragment*.

(g) cz.podola.furnanceservicemenapp.dialog – využívány uživatelským rozhraním pro získávání vstupů od uživatele nebo jeho informování, a to formou dialogových oken.

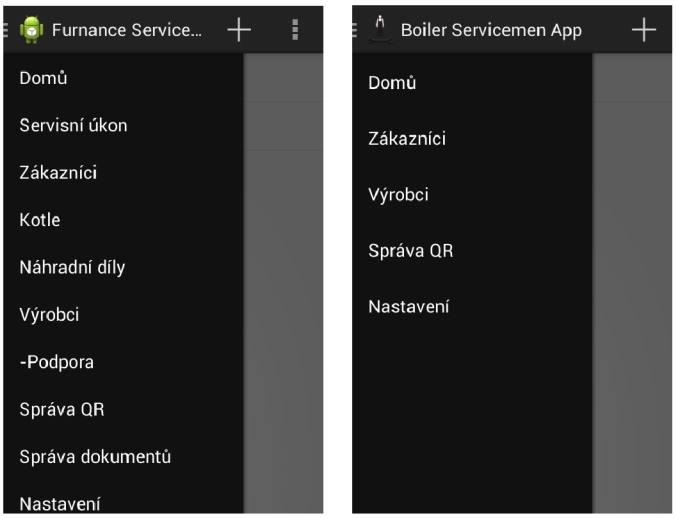
5.1.1 Navigace

Pro účely navigace napříč aplikací jsem zvolil vyjíždějící menu od levého okraje obrazovky směrem k jejímu středu, které nese název Navigation Drawer[3]. Důvodem této volby bylo původní navržení celkem deseti kategorií, jak je znázorněno na obrázku 5.2(a). Tento počet byl v průběhu implementace značně redukován, a to na hodnotu pět. Příčinou bylo uvědomění si závislostí, které jsou patrné z funkčních požadavků (kapitola 4.2.1). Například kategorie *Náhradní díly*, potažmo náhradní díl je vždy vázán ke konkrétnímu výrobci. Proto je nadbytečné, aby měl v globálním menu aplikace svou kategorii při uvědomění si situace, kdy uživatel bude chtít vypsat všechny náhradní díly daného výrobce, kterého bude muset nejprve ze seznamu všech výrobců vybrat. Tudíž kategorie *Náhradní díly*, *Podpora* a *Správa dokumentů* byly sloučeny do jediné kategorie - *Výrobci*. Zde jsou všechny zmíněné skupiny přístupné jiným způsobem, který bude dále v textu představen. Obdobná situace nastala u skupin *Zákazníci*, *Kotle*, *Servisní úkon* a jejich sloučení do skupiny *Zákazníci*. Ukázka výsledného menu je zobrazena na obrázku 5.2(b).

Vyjíždějící menu nemá implementované podkategorie z důvodů popsaných začátkem této podkapitoly. Maximální hloubka zanoření v kategorii *Zákazníci* nabývá hodnoty pět v případě tabletu, šest u mobilního telefonu. Aby byl uživatel schopen v každé úrovni zanoření určit, ve které se momentálně nachází a co která znamená, byl implementován další prvek navigace - Action Bar[4]. Jedná se o lištu lokalizovanou na vrchní straně zobrazovaného obsahu. V tomto prvku je uveden nadpis aktuální úrovně a tlačítko „+“, pokud to v této dává smysl, sloužící pro přidání nového záznamu. Jeho ukázka je viditelná na obrázcích 5.2.

Funkcionalita navigace byla implementována prostřednictvím bázové třídy *BaseActivity*. Ostatní třídy implementující funkci Aktivity, popsané v kapitolách 2.2 a 2.3, dědí právě od této bázové třídy, a to z důvodu zajištění jednotného zobrazení, obsahu a struktury vysouvacího menu a navigační lišty. V případě nutnosti modifikace prvků navigace

[3] http://developer.android.com/design/patterns/navigation-drawer.html
[4] http://developer.android.com/guide/topics/ui/actionbar.html



(a) Původní návrh (b) Současná podoba

Obrázek 5.2: Globální menu

postačí provedení změn právě v této bázové třídě. Implementace vyjíždějícího menu není triviální záležitostí, při implementaci jsem využil volně dostupné a doporučované postupy na webových stránkách určených pro vývojáře[3].

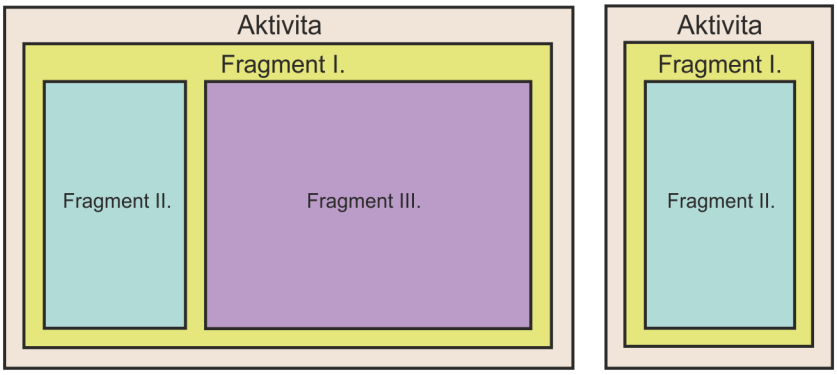
5.1.2 Využití fragmentů

Aby uživatelské rozhraní reflektovalo jeho návrh popsaný v kapitole 4.4, rozhodl jsem se pro využití funkcionality nabízené fragmenty. Další výklad je zaměřen k obrázku 5.3. Pro každý pohled, např. zákazníci a jejich seznam společně s detailem vybraného, existuje *Aktivita* odvozená od třídy *BaseActivity*. Jejím úkolem je nastavit nadpis sekce v liště navigace a vytvořit *Fragment I.*, jenž umístí do připraveného kontejneru typu *FrameLayout* definovaného v layout[5] souboru. Tímto způsobem je zajištěno zachování funkcionality vysouvacího menu, pro které je v layout souboru připraven prvek[6] *ListView*.

Pro rozhodnutí, zda se aplikace nachází v zařízení typu tablet (obrázek 5.3(a)) nebo mobilní telefon (obrázek 5.3(b)), je využito odlišné definice rozložení prvků na obrazovce dle její velikosti za běhu programu. V prvním případě operační systém načítá rozložení z umístění (více v kapitole 2.5) */res/layout/soubor_rozlozeni.xml*, ve druhém to bude z lokace */res/layout-large/soubor_rozlozeni.xml*. Tyto soubory mají odlišnou definici, kdy v případě tabletu jsou k dispozici kontejnery typu *FrameLayout* pro *Fragment II.* a současně *Fragment III.* V situaci mobilního telefonu bude k dispozici pouze *Fragment II.* Kontrolou existence kontejneru pro *Fragment III.* se zjišťuje, na kterém typu zařízení aktuálně aplikace pracuje.

Fragment I. spravuje své „potomky“ ve formě Fragmentů. Tyto zasílají svému „rodiči“ informace o jejich interakci s uživatelem. Tudíž, *Fragment I.* v závislosti na interakcích svých potomků reaguje formou jejich překreslení, záměně nebo startem nové *Aktivity*. Zmíněné chování závisí na aktuálním kontextu, ve kterém se aplikace nachází. Mimo jiné z toho vyplývá, že *Fragment I.* tvoří mezivrstvu mezi Aktivitou a ostatními fragmenty. V případě

[5]XML soubor obsahující rozmístění prvků na obrazovce. Jeho vysvětlení v kapitole 2.3
[6]Fakticky se jedná o statickou definici objektu, dále tento budu nazývat prvkem.



(a) Tablet (b) Mobilní telefon

Obrázek 5.3: Fragmenty

důslednosti, tento nemusí být vůbec využit a jeho funkcionalitu by mohla plně převzít *Aktivita*. Důvodem, proč *Fragment I.* využívám, je co možná nejvyšší odstínění vlivu *Aktivity* na výsledné programové řešení aplikace, jelikož se jedná o novější technologii, která se v současné době začala hojně využívat.

Zmíněná funkcionalita se neobejde bez vzájemné komunikace Fragmentů, kterou rozlišuji následovně:

- Vstupní – nastává při vytváření Fragmentu s nutností předání dodatečných informací ve formě argumentů, bez kterých tento nemůže korektně pracovat. Argumentem může být například cizí klíč řádku tabulky, jenž má být použit v SQL příkazu pro získání dat z databáze, která mají být Fragmentem zobrazena. Tyto argumenty nesmějí být předávány prostřednictvím konstruktoru Fragmentu, protože Fragment, stejně jako Aktivita, prochází životním cyklem, ve kterém může být zničen a následně opět vytvořen. Tyto parametry by nebyly v případě znovuvytvoření přístupné a docházelo by k nekorektnímu chování. Proto se musí argumenty předávat pomocí objektu *Bundle*. Dalším druhem vstupní komunikace v řešení aplikace je nutnost u některých Fragmentů překreslení jejich obsahu. Pro tento případ je implementována metoda, jenž může být zavolána jeho rodičem.
- Výstupní – Fragment definuje rozhraní[7], které jeho rodič musí implementovat (programově hlídáno). Díky tomu může Fragment zasílat zprávu pomocí zpětného volání svému rodiči, který na tuto patřičně reaguje.

5.1.3 Knihovny cizích zdrojů

Při řešení programové části bylo využito následujících knihoven cizích zdrojů. Jejich funkcionalita není stěžejní ve smyslu zadání bakalářské práce.

ActionBarSherlock

Knihovna[8] zajišťující jednotné zobrazení navigační lišty na zařízení s verzí API 5 a vyšší. Díky jejímu využití jsem zamezil nežádoucímu chování, které je znázorněno obrázkem 5.4.

[7] Anglický název - interface
[8] http://actionbarsherlock.com

Při použití stejného motivu má tato lišta různý vzhled na zařízeních s odlišnými verzemi operačního systému Android - API 16 na obrázku 5.4(a), API 13 na 5.4(b). Při vytváření třídy implementující funkci Aktivity tato nedědí od třídy *Activity*, nýbrž od *SherlockActivity*. Při využití Fragmentů, popsaných v kapitole 2.4, nedochází k dědění od třídy *Fragment*, nýbrž od *SherlockFragment*. Podobná situace nastává u ostatních podtříd Aktivit a Fragmentů.

(a) API 16 (b) API 13

Obrázek 5.4: Porovnání vzhledu lišty Action Bar

Znázorněná situace je naznačena na jednoduchém příkladu. Pokud se využívá dalších funkcí, které poskytuje navigační lišta, je rozdíl ve výsledném vzhledu dosti markantní a obecně se zvyšuje při podpoře nižších verzí operačního systému. Proto bylo této knihovny využito s ohledem na další budoucí vývoj aplikace.

aFileDialog

Operační systém Android, resp. knihovny používané pro vývoj, neposkytují vestavěný prohlížeč souborů, s uživatelským rozhraním, uložených v zařízení či externím úložišti. Proto bylo využito knihovny **aFileDialog**[9], která pracuje na zařízeních s verzí API 8 a vyšší. Poskytuje základní funkce pro práci se soubory a složkami. Tato knihovna může být využita formou Aktivity nebo dialogového okna.

Barcode Scanner

Při řešení tuto aplikaci využívám pro skenování EAN a QR kódů. Nejedná se tudíž o knihovnu, avšak existuje způsob, kterým ji do projektu (kapitola 2.5) začlenit ve formě externí knihovny. Proto je tato netypicky součástí podkapitoly 5.1.3. Při včlenění do výsledného řešení odpadá nutnost explicitní instalace této aplikace, kterou musí provést uživatel, pokud chce využívat zmíněné funkce. Avšak velká nevýhoda nastává při vydání jejich nových verzí. V této situaci by bylo nutno importovat novou verzi knihovny a zpřístupnit novou verzi mé aplikace. Proto výhodnější řešení spočívá v použití **Barcode Scanner**[10] jako externí aplikace, za předpokladu zachování stejného komunikačního rozhraní mezi aplikacemi.

5.2 Databáze

Pro manipulaci s databází SQLite, popsanou v kapitole 2.6.3, je využito tříd z balíčku „cz.podola.furnanceservicemenapp.**db.adapter**“ (v dalším textu budou balíčky uváděny bez jmenného prostoru). Každé tabulce navržené databáze, jenž je znázorněna obrázkem 4.1, zde náleží jedna třída, jejíž název koresponduje s názvem tabulky doplněný o příponu „DBA[11]“. Každá tato třída definuje pomocí statických proměnných svůj název a jména sloupců, které tabulka obsahuje. Dále poskytuje metody, které manipulují s daty své tabulky

[9] http://code.google.com/p/afiledialog/
[10] http://play.google.com/store/apps/details?id=com.google.zxing.client.android
[11] Z anglického spojení - DataBase Adapter

v databázi. Důvodem implementace typu „každé tabulce odpovídá jedna třída“ je snaha o určité zapouzdření funkcionalit prováděných nad konkrétní tabulkou.

Třída *DBA*, obsažená v balíčku **db.adapter**, definuje jméno databáze a její verzi. Při každém spuštění aplikace pomocí instance této třídy je ověřováno, zda je databáze vytvořena. Při záporném výsledku je volána metoda *onCreate* vnitřní třídy *DBHelper*, která databázi vytvoří za pomocí SQL příkazů, jenž jsou definovány s využitím statických proměnných (poskytují ostatní třídy balíčku). Pokud v budoucnu bude změněno číslo databáze v kladném směru, bude volána metoda *onUpdate*, pomocí které lze strukturu databáze částečně (omezení SQLite databáze jsou uvedeny v kapitole 2.6.3) modifikovat.

Modelem budu dále nazývat třídu, která vlastní atributy s názvy a datovými typy odpovídajícími sloupcům tabulky databáze. Dále tato poskytuje veřejné metody pro získání a nastavení každého atributu. Modely jsou pak využívány pro komunikaci mezi Fragmenty a třídami z balíčku **db.adapter**, kdy záznamy získané z databáze jsou předávány formou objektu patřičného modelu tabulky při dotazu na konkrétní záznam nebo kolekci těchto modelů, pokud se předpokládá více výsledků na dotaz provedený nad patřičnou tabulkou. Modely se taktéž využívají pro předání nového záznamu, jenž má být zapsán do databáze.

5.3 Uživatelské rozhraní

V následujících podkapitolách bude konkrétněji popsána implementace funkcionality uživatelského rozhraní. Bude užito konvence, kdy objekt uživatelského rozhraní definován staticky v layout souboru (textové pole, tlačítko aj.) nazývám „prvkem“. Užité třídy a z nich případně vytvořené instance budou zvýrazněny kurzívou. Při výkladu bude využita příloha B, jenž obsahuje snímky výsledné aplikace, v nichž údaje o osobách jsou smyšlené.

Veškeré ikony využité při implementaci uživatelského rozhraní byly získány z volně dostupných zdrojů v prostředí sítě Internet. Výjimku tvoří pouze hlavní logo aplikace znázorněné obrázkem 5.1.

5.3.1 Výrobci

Pro správu výrobců slouží okno znázorňující obrázek B.1(a). Levý sloupec obsahuje seznam výrobců reprezentovaný třídou *ManufacturerListFragment*, pravý sloupec byl implementován pomocí třídy *ManufacturerDetailFragment*. Tento zobrazuje základní informace o výrobci (název společně s umístěním webové prezentace) a dále první tři záznamy produktů, náhradních dílů a servisní podpory. Každý ze zmíněných představuje v pořadí *ProductListFragment*, *PartListFragment*, *SupportListFragment*. Další výklad bude vázán k *ProductListFragment*, přičemž stejná situace nastává u ostatních zmíněných.

Fragment je umístěn v prvku *FrameLayout*, jenž má definovanou fixní výšku a který je umístěn v prvku *LinearLayout*, jehož funkce spočívá v nastavení odlišného pozadí pro vizuální oddělení formou obdélníku. Pokud *ProductListFragment* obsahuje dva a méně záznamů, uživatelskou interakci ve zmíněném obdélníku, kde nejsou zobrazeny prvky pomocí *ProductListFragment*u, zachytí místo něj jeho rodičovský prvek *FrameLayout*. Obsluha interakce spočívá ve startu patřičné *Aktivity*. Pro přidání či úpravu výrobce je využito dialogové okno znázorněné obrázkem B.1(b).

5.3.2 Produkty

Pokud uživatel zvolil v okně výrobců (kapitola 5.3.1) produkt, je spuštěna nová Aktivita, jejíž vzhled je zobrazen obrázkem B.2(a) skládající se ze dvou sloupců. Levý je definován Fragmentem *ProductLeftFragment*, v němž jsou zobrazeny základní informace o zvoleném výrobci z předcházejícího kroku společně s jeho ikonou. Abych zamezil případným sporům s firmami výrobců, rozhodl jsem se místo použití oficiálního loga, které může mít registrovánu ochrannou známku, využít počátečního písmena názvu výrobce a zobrazovat ikonu obsahující právě ono písmeno[12]. Dalším argumentem pro zmíněné řešení je možnost uživatelem definovaných výrobců, proto není vázán pouze na předem definované společně s jejich logy.

Pravý sloupec okna obsahuje seznam uložených produktů, přičemž obrázek B.2(b) zobrazuje dialog sloužící pro přidání či úpravu produktu. Pro výběr kategorie, do které přidávaný či upravovaný produkt náleží, je využito prvku *Spinner* s obecným adaptérem *SpinnerRowImageTextViewAdapter* plnící jeho řádky. Tento je využit i v ostatních případech, kdy řádek *Spinner*u zobrazuje pouze obrázek a text.

5.3.3 Dokumenty

Obrázek B.3(a) zobrazuje seznam uložených dokumentů. Levý sloupec obsluhuje *DocumentLeftFragment*, jenž k zobrazení svého obsahu využívá *ProductLeftFragment* (činnost vysvětlena v kapitole 5.3.2) a *ProductDetailFragment*, jehož úkolem je zobrazit informaci o produktu, který byl v předchozím kroku vybrán pro zobrazení k němu přiřazených dokumentů.

Přidávání dokumentů, zobrazující snímek B.3(b), se provádí formou dialogového okna, ve kterém má uživatel na výběr ze tří kategorií dokumentu – uživatelský manuál, servisní manuál, fotografie. Při stisku tlačítka „Vyberte soubor“ se vytvoří nový dialog, jenž je implementován pomocí knihovny aFileDialog popsané v kapitole 5.1.3. Před samotným vytvořením se nastaví filtrování pro soubory s příponou *pdf*, *jpg* a *png*. Tímto způsobem může uživatel označit soubor nacházející se v interním i externím úložišti.

Dokumenty jsou ukládány do externího úložiště při využití objektu třídy *Files* (více začátek kapitoly 5.1) a jeho poskytovaných metod. Nejprve se zjišťuje dostupnost externího úložiště. Poté je volána metoda *getExternalFilesDir*[13] s parametrem *null*. Tato vrací objekt třídy *File*, jenž má nastavenou absolutní cestu adresáře v externím úložišti, do kterého aplikace smí zapisovat svá data. Výhoda užití zmíněného adresáře nastává při případné odinstalaci aplikace, kdy veškerá data zapsaná aplikací v umístění získané zmíněnou metodou, budou automaticky mazána. Kopírování souboru probíhá v novém vlákně, během této operace je uživateli zobrazeno oznámení ve formě **Progress Dialog**u[14]. Po dokončení této operace může uživatel původní soubor smazat.

Do sloupce *path* tabulky DOCUMENT (více obrázek 4.1) bude uložena absolutní cesta uloženého dokumentu, přičemž tato nabývá tvaru „rootDir/document/id/soubor“, ve které rootDir, document, id, soubor značí postupně:

- cestu navrácenou pomocí metody *getExternalFilesDir*,
- zvolený název složky pro dokumenty,

[12] Jestliže počáteční písmeno nepatří do množiny znaků anglické abecedy, bude zobrazen jednoduchý obrázek znázorňující továrnu.

[13] http://developer.android.com/reference/android/content/Context.html#getExternalFilesDir(java.lang.String)

[14] http://developer.android.com/reference/android/app/ProgressDialog.html

- primární klíč nově vytvářeného záznamu v tabulce DOCUMENT,
- původní název ukládaného souboru.

Externí úložiště nemusí být vždy přístupné, proto při výpisu dokumentů zobrazuji jejich dostupnost, jak je patrné v pravé části snímku B.3(a). SQLite databáze nabízí možnost uložení obecně jakéhokoli souboru, já ji však nevyužil z důvodu úspory místa interního úložiště.

Pro otevírání dokumentů využívám implicitních záměrů[15]. To znamená, že operační systém nabídne uživateli všechny jeho instalované aplikace, které dokáží zobrazit obsah vybraného dokumentu.

5.3.4 Náhradní díly, servisní podpora

Snímek B.4(a) zobrazuje náhradní díly vybraného výrobce. Podobně jako předcházející případy, levý sloupec tvoří *ProductLeftFragment*. Pravý sloupec, *PartListFragment*, zobrazuje seznam náhradních dílů. Pokud uživatel libovolný zvolí, není vytvořena nová Aktivita nebo dialogové okno jako v předcházejících případech, ale dochází k zobrazení detailních informací náhradního dílu (situaci znázorňuje snímek B.4(b)), přičemž tyto jsou reprezentovány pomocí *PartDetailFragment*, který nahradí *PartListFragment*. Pokud uživatel stlačí tlačítko zpět, dochází k opětovnému zobrazení seznamu náhradních dílů.

Při definování nového náhradního dílu dochází k vytvoření nové Aktivity *PartAddActivity*. Uživatel, mimo jiné, vkládá hodnotu EAN kódu náhradního dílu. Může ji definovat ručně nebo využít funkci skenování pomocí aplikace **Barcode Scanner** popsané v kapitole 5.1.3. Z prostorových důvodů snímek situace neuvádím.

Obrázek B.5 ukazuje správu servisní podpory, jejíž implementace staví na stejném principu, jako je tomu u popsané administrace náhradních dílů. Při zobrazení detailu servisní podpory patrného na snímku B.5(b), má uživatel k dispozici dvě tlačítka pro volání a jedno pro odeslání emailové zprávy. Tyto jsou dostupná, jestliže byla patřičná pole při vytváření či úpravě záznamu servisní podpory vyplněna. Jejich funkcionalita je realizována pomocí implicitních záměrů, kdy pro volání bylo využito typu záměru *ACTION_CALL* s nastavením patřičného oprávnění v souboru manifestu (kapitola 2.5). V průběhu testování bylo zjištěno, že na mobilních telefonech, kde se tato funkcionalita předpokládala, nedocházelo k vytvoření hovoru. Proto byl záměr v novějších verzích aplikace změněn na typ *ACTION_DIAL*. Tento nevytváří hovor implicitně, nýbrž vyplňuje pole uživatelského rozhraní operačního systému Android sloužící pro zadávání telefonního čísla explicitně uživatelem. Poté stačí volbu pouze potvrdit. Pro vytváření nové emailové zprávy jsem využil implicitního záměru typu *ACTION_SENDTO*.

5.3.5 Zákazníci

Administraci zákazníků zachycuje snímek B.6(a). Levý sloupec tvořící seznam zákazníků je implementován pomocí *CustomerListFragment*. Po vybrání položky bude zobrazen její detail v pravém sloupci – *CustomerDetailFragment*. Na přání uživatelů byly v posledních verzí aplikace přidány tlačítka pro volání a vytvoření emailové zprávy, jejichž implementace je totožná s popsanou v kapitole 5.3.4. Přidání či úpravu záznamu (obrázek B.6(b)) implementuje třída *CustomerAddActivity*, kde při výběru typu budovy je využito prvku *Spinner*

[15] Anglický název - Implicit intents.

s obecným adaptérem *SpinnerRowImageTextViewAdapter*. Jeho definici řádku používá *CustomerDetailFragment* při zobrazování detailu záznamu, resp. typu budovy.

5.3.6 Plynová zařízení

Tato sekce je dostupná po stisku tlačítka „Kotle“ v okně zobrazující detail zákazníka. Dle specifikace požadavků, uživatel může vlastnit více plynových zařízení. Situace je znázorněna obrázkem B.7(a).

Levý sloupec tvoří *CustomerDetailFragment*, kterému je při vytváření předána informace, aby zobrazil pouze nezbytně nutné informace. Implementace této funkcionality spočívá v označení prvků definovaných v layout souboru *customer_detail.xml*, jenž mají být skryty, elementem „android:tag“ se zvoleným řetězcem. Po naplnění pohledu fragmentu *CustomerDetailFragment* nastává volání metody *setGoneForViewByTag*. Tato má za úkol rekurzivně projít hierarchii objektů typu *View* (hierarchie může obsahovat i objekty typu *ViewGroup*) a přiřadit prvkům s elementem „tag“ nový element „android:visibility“ s hodnotou „GONE“. Tato značí, že prvek nebude ve výsledku zobrazen a nebude zabírat žádné místo.

Pravý sloupec, implementován pomocí *FurnaceListFragment*, zobrazí seznam plynových zařízení zákazníka. Pro zobrazení jednotlivých řádků užívá adaptéru *FurnaceListAdapter*, jenž plní jednotlivé řádky definované layoutem *furnace_list_row_detailed.xml*. Pro zajištění funkcionality oznámení při dlouhém stisku uživatele na řádku zobrazující detail plynového zařízení, má zmíněný layout nastaven vnější okraj určité šířky, jelikož definice odlišného pozadí řádky tuto funkcionalitu ruší.

Akce přidávání a úprava plynového zařízení jsou znázorněny snímkem B.7(b). Funkcionalitu poskytuje *FurnaceAddActivity*. Pro výběr výrobce a jednoho z jeho poskytovaných produktů jsem využil prvky *AutoCompleteTextView*. Uživateli se při interakci s prvním zobrazí seznam všech výrobců, jenž má v aplikaci uložené. Jakmile vybere patřičný záznam, naplní se data druhého prvku, pomocí kterého vybírá konkrétní výrobek. Výhoda použití prvků *AutoCompleteTextView* nastává v případě, kdy uživatel začne zadávat počáteční znaky názvů, pak se nabízené záznamy filtrují a celá operace výběrů se výrazně zrychluje.

Volitelnými informacemi se rozumí datum roční prohlídky a datum uvedení do provozu. První je zobrazen v zeleném rámečku, druhý v pořadí ve žlutém rámečku, patrných na snímku B.7(a). Proces výběru datumů je implementován pomocí *DatePickerFragment*. Jedná se o dialogové okno využívající třídy *DatePickerDialog* poskytované operačním systémem. Pro účely nulování zvolených hodnot slouží tlačítka s textem „reset“, a to zvláště při úpravě záznamu plynového zařízení.

Uživatel může přiřadit plynovému zařízení jeden z množiny předem připravených QR kódů, jenž jsou dostupné ke stažení v kategorii menu „Správa QR“ znázorněném obrázkem 5.2(b). Po procesu načtení textového řetězce reprezentovaného QR kódem nastává jeho kontrola pomocí třídy *Scanning* a její statické metody *getQrCodeInString*. Tato hledá podřetězec v předaném řetězci ve tvaru `App code:<<_XXXXX_>>`, ve kterém znak „`X`“ představuje kladnou číslici, přičemž navrací řetězec všech získaných číslic. Tento bude ve výsledku přiřazen plynovému zařízení. Metoda může navrátit konstantu *null*, pak načtený QR kód, potažmo řetězec, není validního tvaru. Výhoda přiřazení bude zmíněna v kapitole 5.3.9.

5.3.7 Servisní úkony

Po zvolení plynového zařízení se vytváří nová Aktivita, jejíž pohled je znázorněn snímkem B.8(a). Levý sloupec má totožnou implementaci, popsanou v kapitole 5.3.6. Pravý sloupec zobrazuje seznam veškerých provedených servisních úkonů na plynovém zařízení zákazníka, a to pomocí *OperationListFragment*. Jeho řádky vlastní odlišnou barvu pozadí, proto pro vizuální oznámení při dlouhém stisku řádku jsem využil stejného principu, jenž byl popsán u *FurnaceListFragment* v předchozí podkapitole.

Přidávání nebo úprava servisního úkonu je provedena pomocí *OperationAddActivity*. Tuto situaci znázorňuje snímek B.8(b). Uživatel zvolí typ servisního úkonu pomocí prvku *Spinner*, zadá datum dokončení a případný datum splatnosti. Tyto jsou viditelné při výpisu servisních úkonů, jenž jsou řazeny dle data ukončení, na snímku B.8(a) ve žlutých obdélnících – datum splatnosti s ikonou amerického dolaru, datum dokončení s ikonou vlajky znázorňující cíl.

Pomocí prvků *CheckBox* může uživatel definovat, zda prováděný servisní úkon spadá pod záruku hrazenou výrobcem, stav servisního úkonu a způsob platby. Pokud není označena metoda platby jako „hotově“, je umožněno definovat číslo faktury. V opačném případě bude případné číslo faktury z patřičného pole ihned smazáno a jeho editace bude zakázána.

Pro definování použitých náhradních dílů je využit dialog implementovaný pomocí *OperationPartAddDialog*, jenž využívá layout soubor *operation_add_part.xml*. Uživatel má na výběr ze dvou možností určení náhradního dílu. Prvním je využití prvku *AutoCompleteTextView*, přičemž jsou nabízeny pouze ty náhradní díly, které definoval aktuálnímu výrobci plynového zařízení v sekci náhradních dílů popsané v kapitole 5.3.4. Druhým způsobem může být využití funkce skenování EAN kódu, jenž bývá umístěn na štítku obalu, ve kterém byl náhradní díl distribuován dodavatelem. Využití druhého způsobu předpokládá vyplněnou hodnotu EAN kódu při vytváření nebo úpravě záznamu náhradního dílu. Vložení nové závady probíhá formou dialogového okna *FaultDialog*. Uživatel zadává popis závady a volitelný popis opravy. Ukázky obou dialogů neuvádím z prostorových důvodů.

Původním záměrem pro zobrazení užitých náhradních dílů a definovaných závad (dále nazývané souhrnně záznamem) bylo využití prvku *ListView* s patřičným adaptérem definující jeho řádky, jejichž počet se za běhu programu může měnit s ohledem na přidávání či odebírání záznamů. Pokud byl přidán první záznam, řádek se zobrazil v pořádku, ovšem po přidání druhého a následujících byl zobrazen pouze původní řádek, velikost prvku *ListView* nebyla změněna, přičemž funkce posuvu poskytované prvkem *ListView* nebyla funkční. Důvodem zmíněného chování je skutečnost, kdy téměř veškeré layouty využité v aplikaci vlastní kořenový prvek *ScrollView*, jenž zajišťuje funkci posuvu, jestliže se veškerý obsah definovaný souborem layoutu při vykreslování nevleze na obrazovku zařízení. Pokud se uživatel snažil mezi záznamy řádků *ListView* procházet, tuto interakci zachytil kořenový prvek *ScrollView*, čímž požadovaný záměr nemohl být proveden. Řešením problému může být umístění výpisu náhradních dílů a závad do nové Aktivity, zmíněným způsobem by však uživatel ztrácel celkový přehled o servisním úkonu, nehledě na nutnost vykonat více kroků pro akci mazání záznamu. Proto jsem zmíněnou situaci vyřešil využitím prvku *LinearLayout*, do kterého se dynamicky přidávají nové záznamy ve formě prvku *RelativeLayout*. Vykreslování výsledného layoutu se záznamy pak probíhá žádaným způsobem.

5.3.8 Informace

Po stisku tlačítka „Informace“ dostupného v servisním úkonu bude vytvořena nová Aktivita *OperationInfoActivity*, jejíž vzhled znázorňuje snímek B.9(a). Vrchní část tvoří *Document-*

ListFragment a nabízí uživateli veškeré dokumenty, jenž jsou přiřazeny plynovému zařízení, na kterém probíhá servisní úkon.

Spodní část slouží pro zobrazení servisní podpory aktuálního výrobce, kterou implementuje *SupportListFragment*. Při zvolení položky servisní podpory nastává nahrazení Fragmentu *SupportListFragment* za *SupportDetailFragment*, jenž zobrazí patřičný detail (pravý sloupec snímku B.5(b)).

5.3.9 Hlavní obrazovka

Obsluhu hlavní obrazovky zachycenou snímkem B.9(b), provádí *HomeFragment*. Panelem rychlých akcí nabízím uživateli částečné zrychlení práce s aplikací, přičemž tento umožňuje vyhledat:

- Kotel (plynové zařízení) – pomocí aplikace **Barcode Scanner**, v případě načtení QR kódu přiřazeného plynovému zařízení, bude uživateli zobrazena historie servisních úkonů znázorněná snímkem B.8(a). Uživatel tudíž nemusí procházet kroky vyhledání zákazníka, zobrazení plynových zařízení a výběr konkrétního plynového zařízení.
- Náhradní díl – pomocí stejnojmenné aplikace se načítá EAN kód reprezentující náhradní díl. V případě úspěchu bude zobrazen jeho detail. Situaci znázorňuje snímek B.4(b).

Pro splnění zbylých požadavků pro operace prováděnými nad daty je na hlavní obrazovce zobrazeno:

- Neuzavřené úkony – seznam servisních úkonů, jenž jsou při svém vytváření či úpravě označeny jako neuzavřené. Může nastat v situacích popsaných v kapitole 4.1.
- Nejbližší roční prohlídky – seznam nejbližších deseti ročních prohlídek. Datum je nastaven při vytváření či úpravě plynového zařízení, popsané v kapitole 5.3.6.

5.4 Testování

Výčet zařízení, jenž sloužily pro účely testování, znázorňuje tabulka 5.1. Název následovaný příponou *AVD*[16] značí emulované zařízení vytvořené pomocí předdefinovaných profilů. Abych byl schopen aplikaci otestovat na zařízeních s odlišnými verzemi platforem, bylo nutno modifikovat v profilu právě zmíněnou hodnotu.

Zařízení	**Verze platformy**	**Úhlopříčka** [”]	**Rozlišení** [px]
Nexus 4 *AVD*	3.2	4.7	768 x 1280
Galaxy Nexus *AVD*	4.0	4.7	720 x 1280
HTC Desire X	4.1.1	4.0	800 x 480
3Q q-pad LC1016C	4.1.1	10.1	1280 x 800
Nexus 7 *AVD*	4.1.2	7.3	800 x 1280
Acer Iconia A1-810	4.2	7.9	1024 x 768
Nexus 7 *AVD*	4.3	7.3	800 x 1280

Tabulka 5.1: Zařízení užitá při vývoji a testování aplikace

V průběhu vývoje byl kladen důraz na vzájemné prokládání fází implementace a testování, nicméně po publikování první verze aplikace prostřednictvím služby Google Play byly nalezeny následující nedostatky:

- Sériové i objednací číslo plynového zařízení uvedené v textu specifikace byly chápany jako číselné hodnoty. Ve skutečnosti mohou obsahovat alfanumerické znaky.
- Nebyl vysloven požadavek definování dílů při servisních úkonech, jenž neposkytují výrobci plynových zařízení. Jedná se o tzv. spotřební zboží. Řešením bylo přidání do tabulky OPERATION řádek *priceAnother*. Jedná se o dočasné řešení, přičemž v dalších verzích aplikace bude umožněna správa spotřebního zboží podobně, jako je tomu u náhradních dílů poskytovaných výrobcem.
- V některých případech špatně čitelné textové řetězce z důvodu zvolení nevhodné barvy či velikosti. Řešení spočívalo v upravení souboru *styles.xml* definující použité styly.
- Nefunkční úprava produktu.

Postupné zjišťování závad vedlo k vydávání několika aktualizací, jenž tyto opravovaly. Pozitivním výsledkem může být fakt, kdy nebyl zaznamenám pomocí Developer Console žádný pád aplikace.

[16] Android Virtual Devices

Kapitola 6

Závěr

Velkým přínosem byla možnost pozorovat servisního technika během pracovního procesu. Získané poznatky jsem využil ve fázích specifikace požadavků a návrhu uživatelského rozhraní.

Výstupem analýzy existujících řešení jsem dospěl k pozitivnímu závěru, kdy na trhu mobilních aplikací s operačním systémem Android není k dispozici aplikace, jejíž funkcionalita by byla totožná s výslednou aplikací. Tato byla umístěna volně ke stažení pomocí služby Google Play. Dostupností aplikace v české a anglické lokalizaci jsem docílil skutečnosti, kdy aplikaci využívá menší množství českých uživatelů oproti anglicky mluvícím, jejichž počet každým dnem pozvolna roste.

V mém osobním zájmu je pokračovat ve vývoji aplikace. Možnosti rozšíření funkcionality je celá řada. Jako první chci implementovat generování výpisu použitých náhradních dílů za zvolené období. Další bude vytváření kompletních záloh buďto lokálně nebo pomocí synchronizace se vzdáleným serverem, přičemž v druhém případě budu muset klást velký důraz na zabezpečení vzájemného přenosu, jelikož aplikace pracuje s citlivými daty. Někteří uživatelé by ocenili možnost vytvářet a přiřazovat fotografie k servisním úkonům. Pro tuto možnost byl prozatím přidán sloupec do patřičné tabulky, přičemž ukládání bude prováděno stejnou formou, jako je tomu v současné chvíli při práci s dokumenty. Posledním krokem bude implementace možnosti teamové spolupráce, kdy servisní technici budou moci svou databázi vhodně sdílet, přičemž bude existovat způsob globální správy této databáze. Poté může být aplikace nabízena firemním zákazníkům, kteří zaměstnávají jednotlivé servisní techniky. Dalším rozšířením může být generování statistik poruchovosti jednotlivých produktů.

Během implementace jsem si osvojil výhody plynoucí z využití fragmentů pro podporu odlišných velikostí displayů zařízení. Využil jsem získané teoretické znalosti o návrhu databáze a tyto byly zužitkovány při následné implementaci pomocí relačního databázového systému SQLite.

Literatura

[1] About SQLite [online]. [cit. 2014-05-04].
URL http://www.sqlite.org/about.html

[2] Dashboards [online]. [cit. 2014-05-03].
URL http://developer.android.com/about/dashboards/index.html

[3] Saving Files [online]. [cit. 2013-12-27].
URL
http://developer.android.com/training/basics/data-storage/files.html

[4] Conder, S.: *Android Wireless Application Development, Second Edition.* Addison-Wesley Professional, 2010, ISBN 978-0-321-74301-5.

[5] Grant, A.: *Android 4: Průvodce programováním mobilních aplikací.* Computer Press, 2013, ISBN 978-80-251-3782-6.

[6] Komatineni, S.; MacLean, D.: *Pro Android 4.* Paul Manning, 2012, ISBN 978-1-4302-3930-7.

[7] Meier, R.: *Professional Android 2 Application Development.* Wiley Publishing, Inc., 2010, ISBN 978-0-470-56552-0.

[8] Phillips, B.; Brian, H.: *Android Programming: The Big Nerd Ranch Guide.* Big Nerd Ranch, Inc., 2013, ISBN 978-0321804334.

[9] Ujbányai, M.: *Programujeme pro Android.* Grada Publishing, a.s., 2012, ISBN 978-80-247-3995-3.

Dodatek A

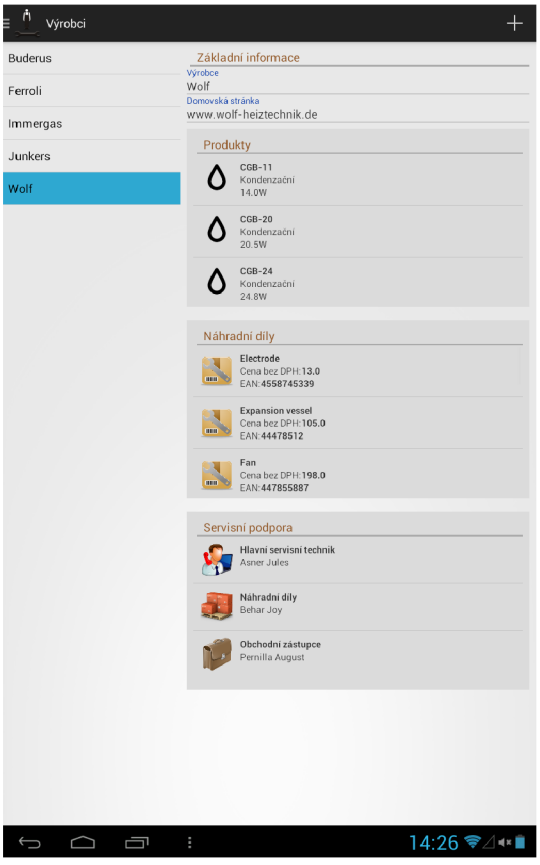
Obsah CD

- **/apk/** – přeložená výsledná aplikace
- **/dokumentace/** – zdrojová dokumentace projektu
- **/manual/** – manuál s popisem zprovoznění projektu
- **/plakat/** – prezentační plakát
- **/projekt/** – projekt pro vývojové prostředí *Eclipse* s obsaženými zdrojovými kódy
- **/projekt-knihovny/** – knihovny třetích stran
- **/zprava/** – elektronická verze této práce ve formátu *pdf*
- **/zprava-tex/** – zdrojové soubory elektronické verze této práce

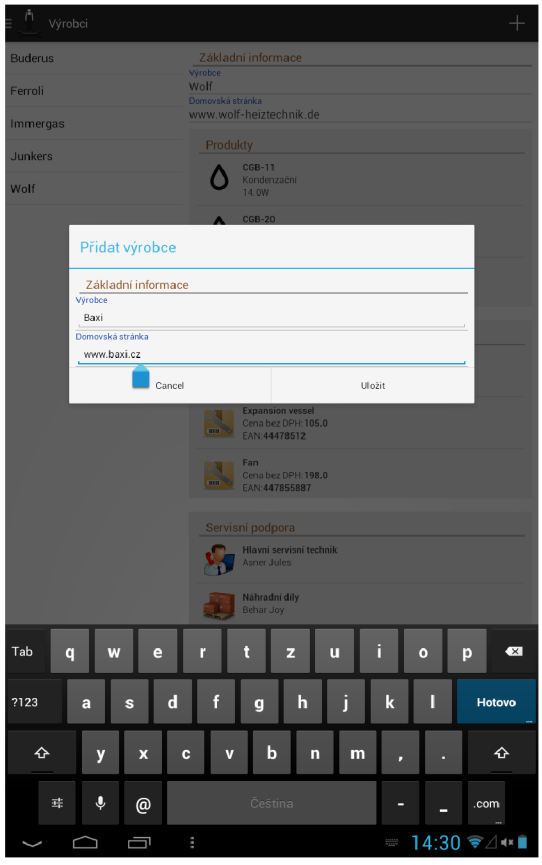
Dodatek B

Snímky aplikace

Příloha zachycuje snímky výsledné aplikace na zařízení *3Q q-pad LC1016C* (parametry dostupné v tabulce 5.1).

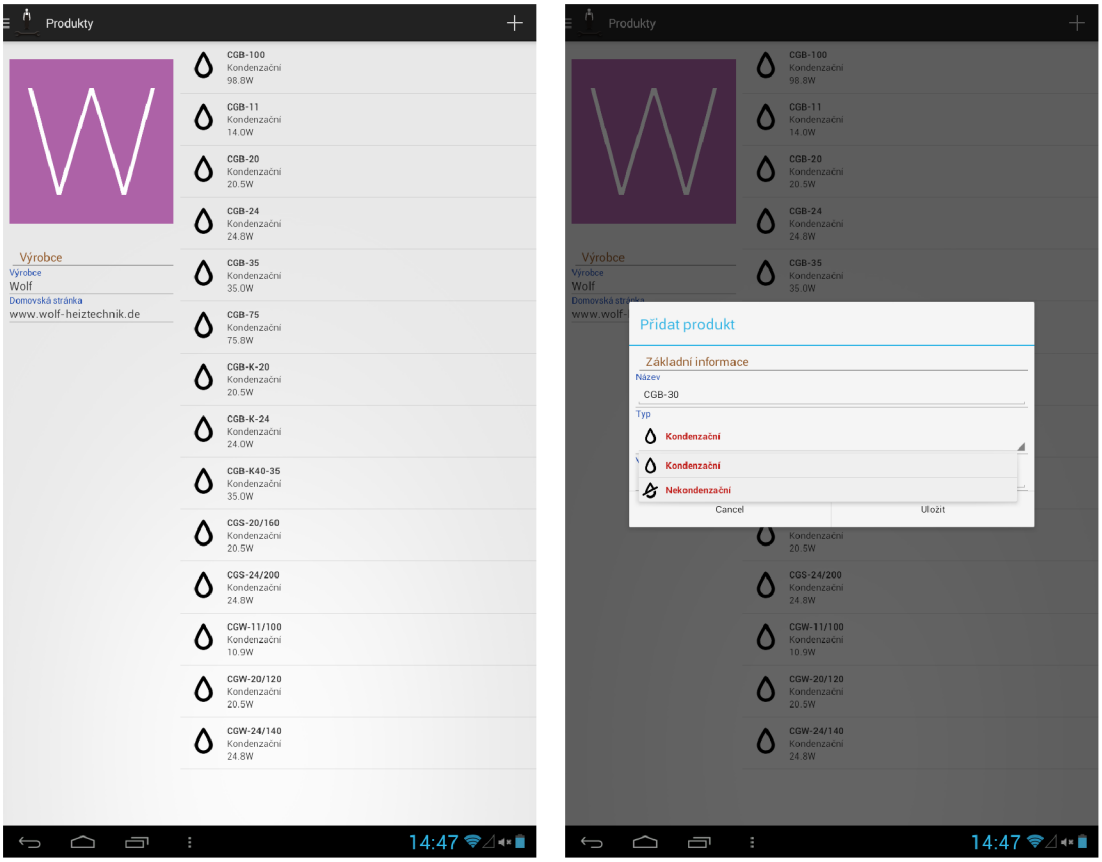


(a) Seznam výrobců s detailem vybraného



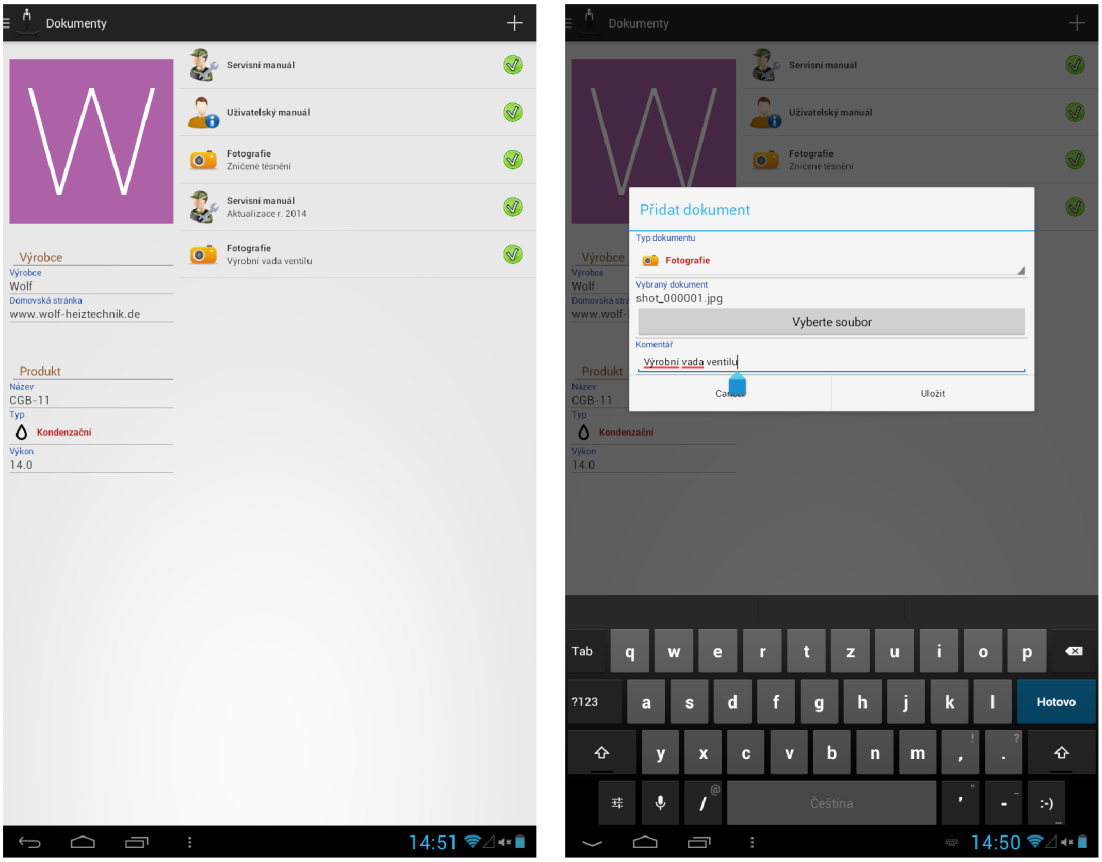
(b) Přidání a úprava výrobce

Obrázek B.1: Výrobci



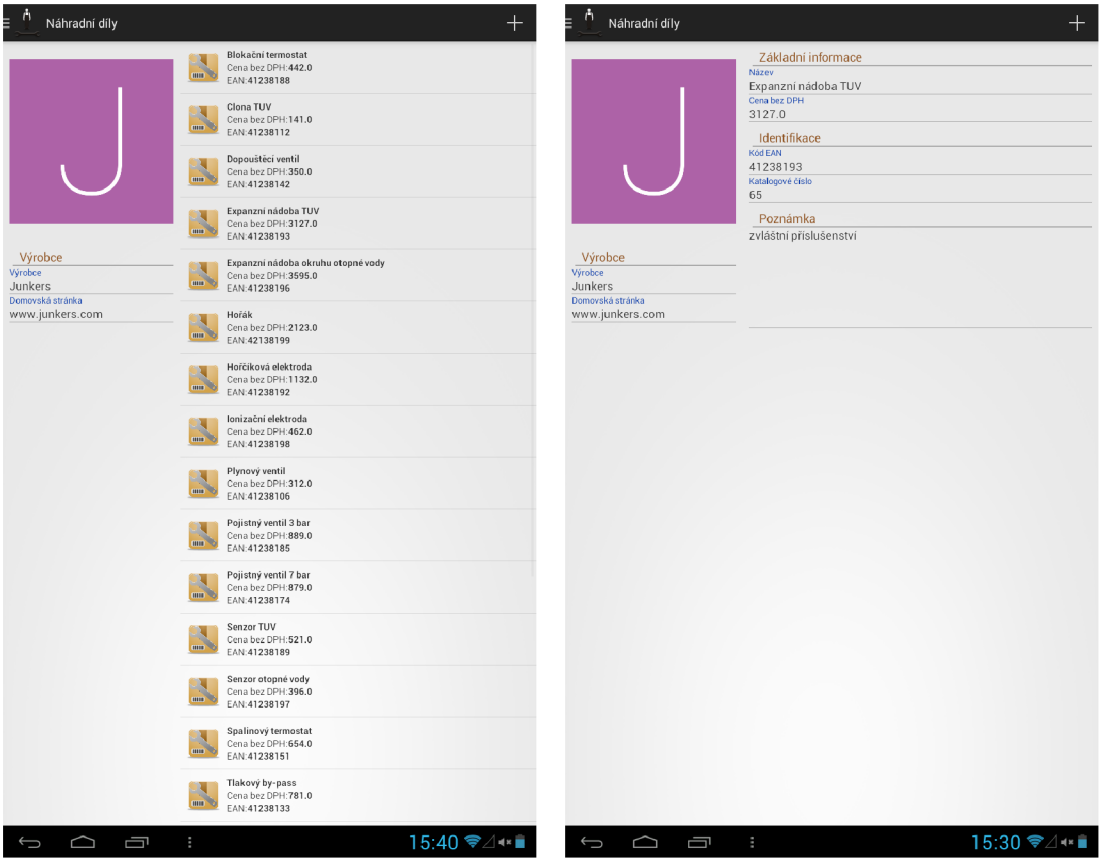
(a) Seznam produktů (b) Přidání a úprava produktu

Obrázek B.2: Produkty výrobce



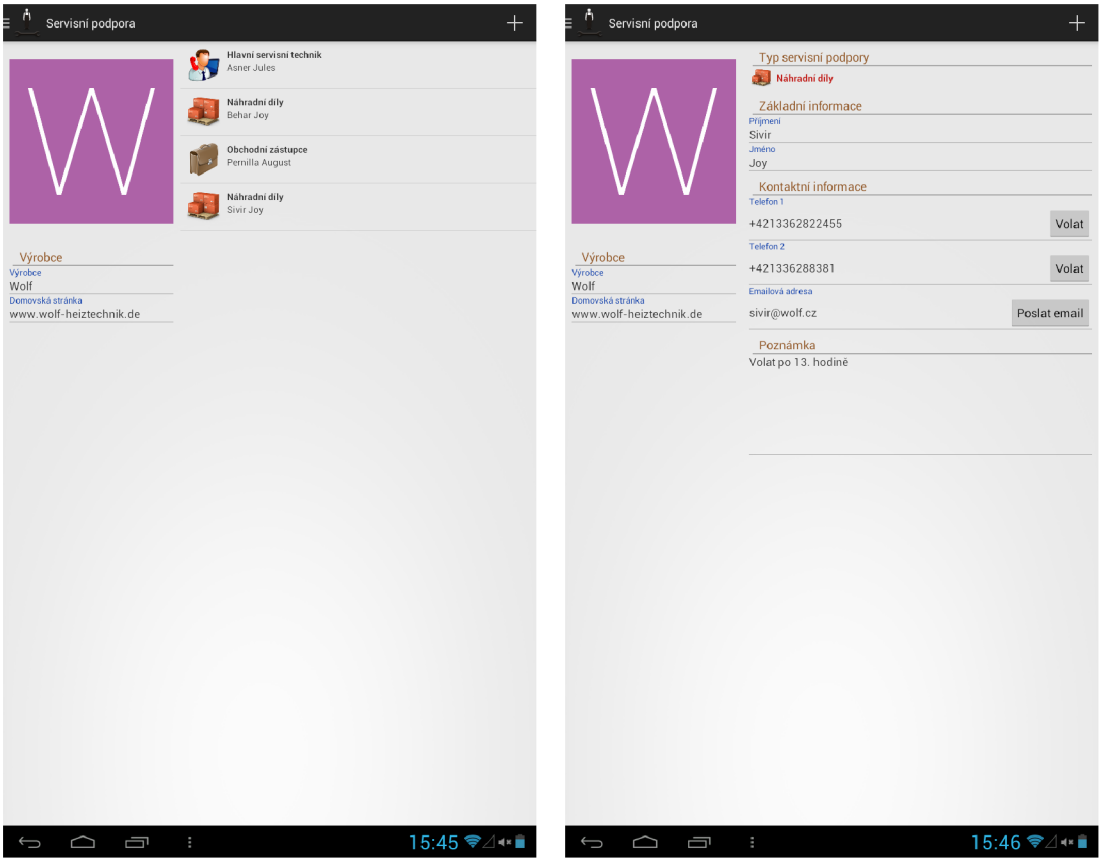
(a) Seznam dokumentů (b) Přidání nového dokumentu

Obrázek B.3: Dokumenty k vybranému produktu



(a) Seznam náhradních dílů (b) Detail náhradního dílu

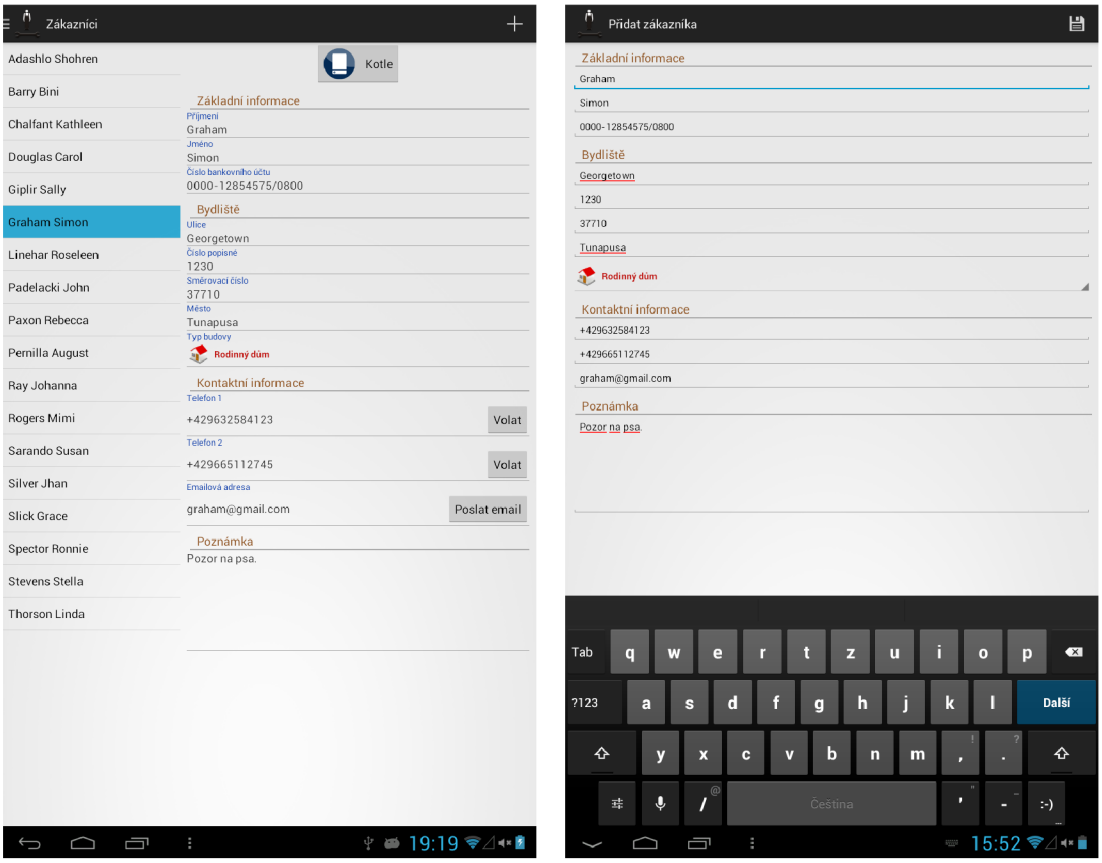
Obrázek B.4: Náhradní díly poskytované výrobcem



(a) Seznam servisních podpor

(b) Detail servisní podpory

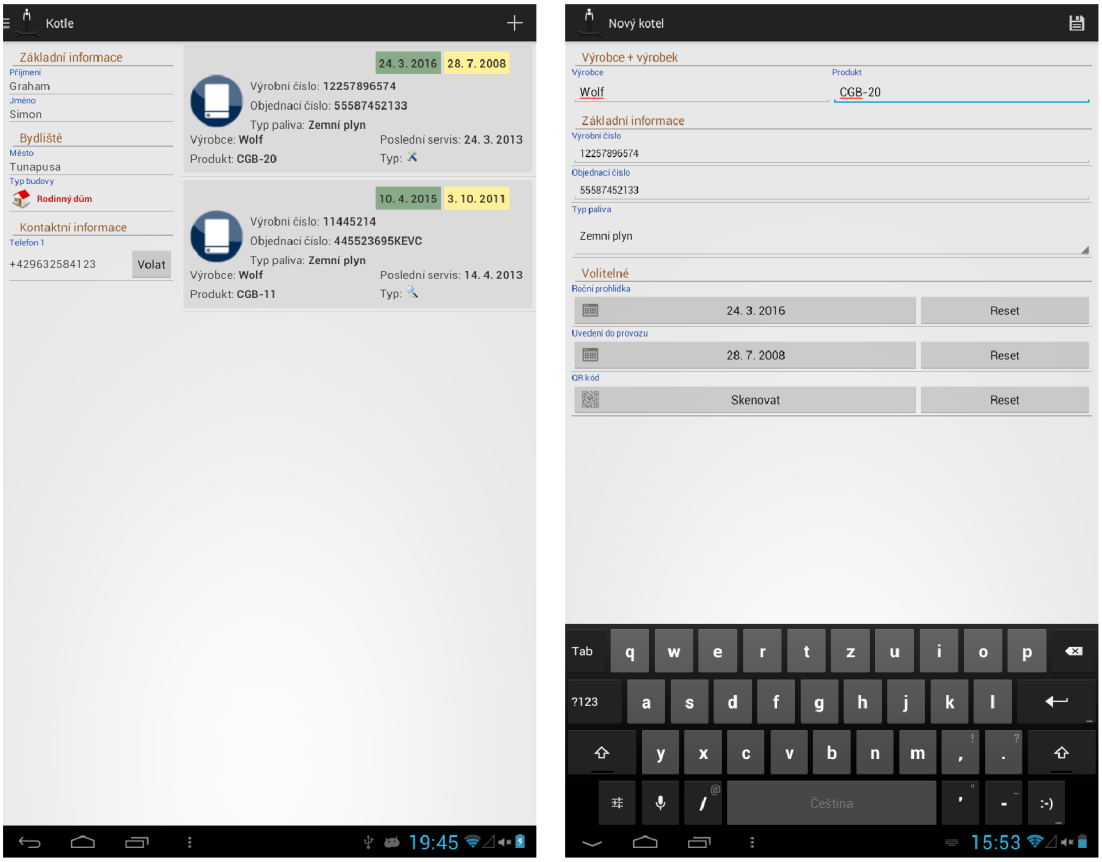
Obrázek B.5: Servisní podpora poskytovaná výrobcem



(a) Seznam zákazníků s detailem vybraného

(b) Přidání a úprava zákazníka

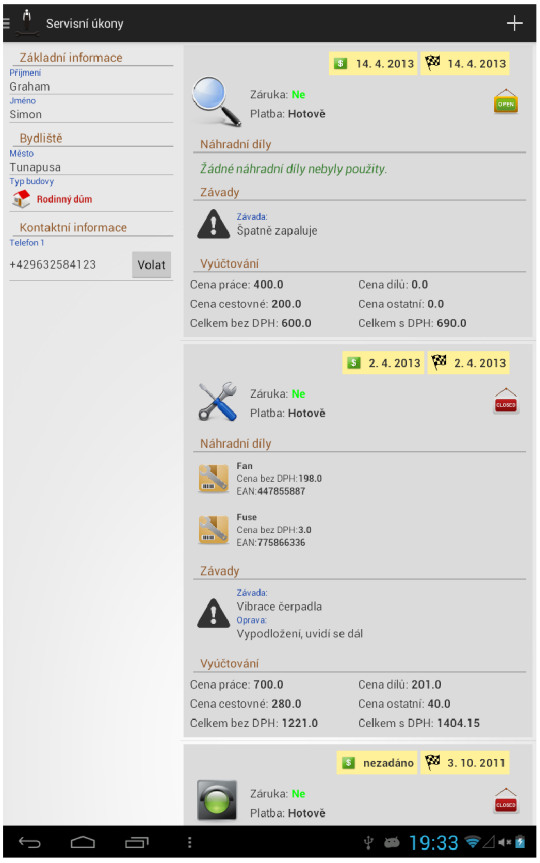
Obrázek B.6: Zákazníci



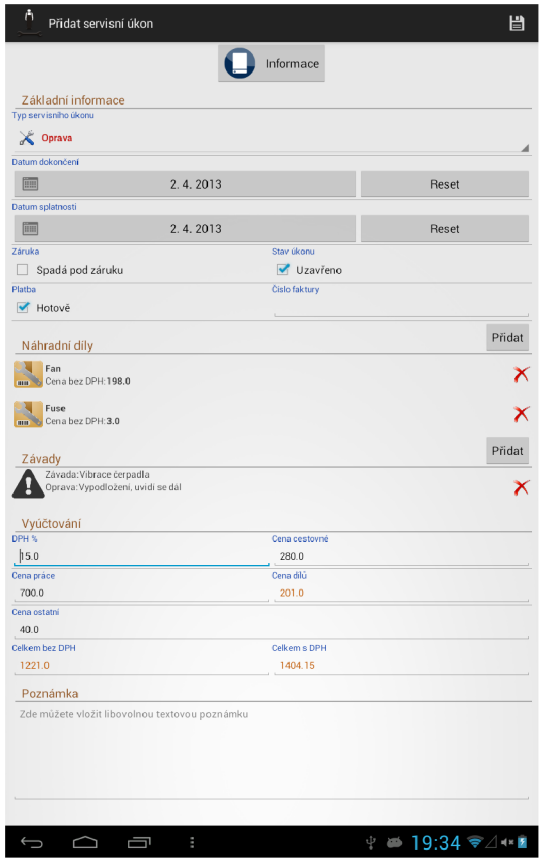
(a) Seznam plynových zařízení zákazníka

(b) Přidání a úprava plynového zařízení

Obrázek B.7: Plynová zařízení

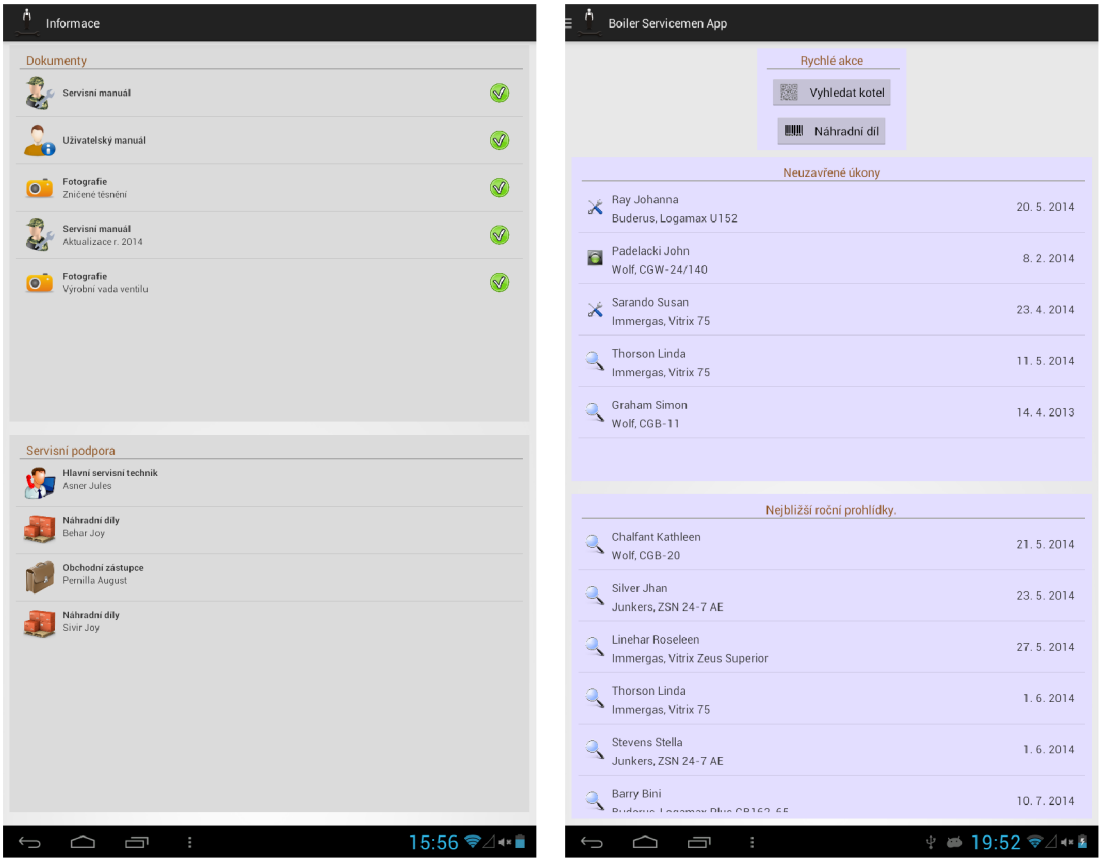


(a) Seznam servisních úkonů provedených na plynovém zařízení



(b) Přidání a úprava servisního úkonu

Obrázek B.8: Servisní úkon



(a) Informace (b) Hlavní stránka

Obrázek B.9: Informace dostupné u servisního úkonu, hlavní obrazovka aplikace