VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

APLIKACE PRO ŘEŠENÍ FIREMNÍCH PROCESŮ V SYSTÉMU ANDROID

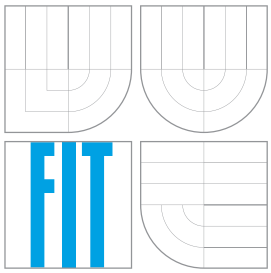
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARCEL KALAI

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

APLIKACE PRO ŘEŠENÍ FIREMNÍCH PROCESŮ V SYSTÉMU ANDROID

AN ANDROID APPLICATION SUPPORTING BUSINESS PROCESSES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARCEL KALAI
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ SMRČKA, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Táto bakalářská práce poskytuje sondu do problematiky vývoje mobilních ERP aplikací. Ilustruje vývoj frameworku, který ulehčí vytváření klientské mobilní aplikace k ERP systému a demonstrační aplikace. Seznámime se s problematikou ERP systémů a platformou Android. Výsledkem bude sada nástrojů, která umožní správu dat, synchronizaci a přidávání modulů, kterých funkčnost bude představena na demostrační aplikaci.

Abstract

This bachelor thesis offers a closer look at mobile ERP application development. It illustrates framework development that facilitates the creation of a customer's mobile application for the ERP system as well as the creation of a demonstration application. We will get acquainted with the area of ERP systems and the Android platform. As a result we will obtain a tool set for data management, synchronization and attaching of modules, functionality of which will be presented by the demonstration application.

Klíčová slova

Android, ERP, informačný systém, synchronizace, Java, Google, databáze

Keywords

Android, ERP, information system, synchronization, Java, Google, database

Citace

Marcel Kalai: Aplikace pro řešení firemních procesů v systému Android, bakalářská práce, Brno, FIT VUT v Brně, 2013

Aplikace pro řešení firemních procesů v systému Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Aleš Smrčka, Ph.D.

.....
Marcel Kalai
15. května 2013

Poděkování

Rád by som poďakoval vedúcemu svojej bakalárskej práce Ing. Alešovi Smrčkovi, Ph.D. za venovaný čas, ochotu a podnetné rady a takisto spoločnosti myWAC Technologies s.r.o. za spoluprácu a poskytnuté informácie.

Obsah

Kapitola 1

Úvod

Každá stredná až veľká firma, ktorá chce svoju činnosť riadiť efektívne, používa nejakú formu podnikového informačného systému. Tieto systémy nájdeme najčastejšie označené ako ERP systémy. V tejto práci budeme používať najmä práve toto označenie. Skratka ERP pochádza z anglického *enterprise resource planning* čiže podnikové plánovanie zdrojov. Je to informačný systém, ktorý modeluje množstvo rôznych firemných procesov. Jedná sa napríklad o nákup, predaj, výrobu, správu majetku, účetníctvo, personalistiku či vzťah zo zákazníkmi (CRM). Často rieši aj triviálnejšie potreby zamestnancov ako správu elektronickej pošty či kalendára.

Implementácií ERP systémov je mnoho, väčšina ale končí u zamestnanca, ktorý pracuje s počítačom v kancelárii. Trendy sa však zmenili. Ľudia chcú mať informácie vždy poruke a vždy aktuálne. Život je rýchlejší – mobilnejší. Napomáha tomu rýchly rozvoj mobilných technológií. Dnešné mobilné telefóny sú výkonnejšie ako prvé počítače, ktoré sme vlastnili. Stále viac ľudí vlastní *smartfón* či tablet. Vznikajú tak požiadavky na offline klientov, mobilných klientov či terminály. Výrobcovia ERP sa týmto požiadavkom snažia vyhovieť. Tieto aplikácie ale nie sú primárnym produktom firmy. Tá preto na ich vývoj vyhradí najmenšiu nutnú časť prostiedkov. Výsledkom takéhoto snaženia je, podľa mojich skúseností, jednoduchá aplikácia, ktorá síce spĺňa požiadavky, no v prípade potreby pridania ďalšej funkcionality je vývoj veľmi neefektívny. Buď je nutné okrem samotnej novej funkcionality upraviť ešte množstvá starého kódu, alebo je nutné vytvoriť nový kus softvéru.

Cieľom tejto práce je preto poskytnúť sondu do vývoja mobilnej ERP aplikácie a vytvoriť sadu nástrojov alebo *framework*, ktorý uľahčí implementáciu mobilnej aplikácie k ERP systému. Aplikácia má znížiť množstvo kódu, ktorý musí programátor napísať predtým, ako sa vôbec dostane k písaniu obslužného kódu samotného ERP systému. Ďalším dôležitým aspektom je čo najmenšia réžia pri pridávaní nových modulov. Demonštrovať to budem pri použití systému myWAC. Ako platformu som si zvolil systém Android pre jeho otvorenosť a rýchlorastúci podiel na trhu.

V nasledujúcej kapitole sa letmo zoznámime s ERP systémami a vývojom pre platformu Android. Ďalšia kapitola sa venuje analýze požiadavkov mobilnej ERP aplikácie a frameworku. Na ňu naväzuje kapitola, ktorá sa venuje návrhu a implementácii aplikácie na základe týchto požiadavkov. Potom prichádza na rad kapitola, v ktorej popisujeme testovanie na demonstračnej aplikácii. Záver pozostáva zo zhrnutia zistení pri vytváraní tejto práce a možností rozšírenia frameworku.

Kapitola 2

Architektúra ERP systémov a vývoj aplikácií pod systémom Android

ERP systémy bývajú väčšinou postavené na architektúre klient-server (dvojvrstvá architektúra) alebo na trojvrstvej architektúre. Výsledkom väčšinou býva desktopová klientská aplikácia, ktorá po sieti komunikuje so serverom, na ktorom je dátové úložisko. Rozdiely bývajú taktiež napríklad v metóde dodania. Poznáme dva základné modely:

- **Model on-premise** – Systém je nainštalovaný na serveroch firmy, ktorá ho využíva. V tomto prípade musí mať firma prostriedky na chod a údržbu. Na aktualizáciách a správe sa podieľa spoločne s firmou dodávateľa systému. Táto metóda sa používa častejšie.
- **Model on-demand** – Tento spôsob je známy aj ako SaaS (Software as a Service). ERP systém je na serveroch dodávateľa a firma k nemu pristupuje cez internet. Výhodou je, že firma nemusí vydávať náklady na údržbu a správu systému.

2.1 Prehľad podnikových systémov

Na trhu sa v súčasnosti nachádza veľké množstvo ERP systémov. Líšia sa mohutnosťou spracovania, podporou platforiem, architektúrou či cenou. Pokúsim sa predstaviť aspoň pár produktov figurujúcich na českom trhu. Ako som naznačil, jedná sa o komerčné produkty určené primárne pre firmy s vacsim mnozstvom zamestnancov.

Informačný systém myWAC

Informačný systém vyvýjaný firmou myWAC Technologies s.r.o. [12]. Predstavený bol v roku 2002, pričom jeho predchodca, Intranet Office, ktorý IS myWAC položil základy, vznikol už v roku 1996. Ponúka komplexné riešenie firemných procesov, ERP, CRM, HRM a DMS. Výhodou je dynamický vývoj, denné online aktualizácie a podpora pre integrovaný e-shop. Prácu spríjemnia aj aplikácie ako elektronická pošta či kalendár. Systém je prístupný z každého zariadenia s pripojením a browserom, pretože klientská časť je implementovaná ako webové rozhranie. Licencované sú jednotlivé aplikácie a cena sa odvíja aj od počtu licencií (teda užívateľov).

ABRA Gx

Systémy ABRA Gx[1] predstavujú radu ERP systémov líšiacich sa mohutnosťou spracovania [13]. Vyvíja ich spoločnosť ABRA Software a.s. Táto pôsobí na trhu od roku 1991. Každá varianta systému Gx ponúka na výber rôzny databázový systém a rôzne moduly, pričom konfiguráciu je možné postupne rozširovať a meniť. Serverová časť môže bežať na platformách Windows a Linux. Klientská časť je riešená desktopovou aplikáciou, ktorá komunikuje so serverom. Výhodou je podpora komunikácie s okolím pomocou webového rozhrania, exportov a jednoduché napojenie na ďalšie z produktov spoločnosti Abra (napr. Mobilný obchodný systém).

Helios

Helios je podnikový informačný systém z produkcie Asseco Solutions a.s. Pokrýva potreby všetkých veľkostí v rôznych oblastiach podnikania a verejnej správy [11]. Verzie tohoto systému sa líšia rozsiahlosťou spracovania a možnosťami systému. Riešenie pre malé firmy je softvér s podporou jednoduchého ERP, riešenie pre veľké firmy je komplexný informačný systém, ktorého súčasťou je podpora CRM, Business Intelligence a tvorba špecializovaných riešení. Výhodou je možnosť výberu z pripravených riešení smerovaných na konkrétne odvetvie priemyslu či služieb.

2.2 Vývoj v systéme Android

Android je open-source platforma určená pre mobilné zariadenia ako chytré telefóny, tablety či PDA [10][2]. Zastrešuje operačný systém postavený na jadre Linuxu, middleware (SQLite, OpenGL, Dalvik VM atď.), aplikačné rozhranie a základné aplikácie. S projektom začala spoločnosť Android Inc., tú neskôr odkúpila spoločnosť Google Inc. Ďalším dôležitým krokom bolo založenie konzorcia *Open Headset Alliance*, ktorému platformu Android spoločnosť Google Inc. predala. Toto spoločenstvo združuje spoločnosti podieľajúce sa na výrobe mobilných telefónov, čipov či softvéru za účelom vytvoriť otvorený štandard pre mobilné zariadenia. Jadro Androidu je navrhnuté tak, aby mohol fungovať na rôznych zariadeniach nezávisle na type hardvéru. K samotnej platforme patria aj nástroje pre efektívny vývoj aplikácii (SDK[2]).

2.2.1 Architektúra systému Android

Predtým, než sa vydáme ďalej, by bolo vhodné konkrétnejšie popísať, čo všetko prináša a z čoho sa skladá Android. Android SDK nám prináša nástroje a API, vďaka ktorým môžeme využívať jeho funkcionality. Tu je zoznam tých významnejších:

- Aplikačný framework, ktorý nám umožňuje znovupoužiteľnosť aplikačných prvkov naprieč aplikáciami a nahradenie natívnych aplikácií.
- Virtuálny stroj Dalvik, ktorý je optimalizovaný pre mobilné zariadenia a našu aplikáciu interpretuje nezávisle na HW platforme.
- Integrovaný webový prehliadač založený na technológii WebKit.

[1] V súčasnosti ABRA G1, ABRA G2, ABRA G3, ABRA G4
[2] Software Development Kit

- Optimalizovaná a hardvérovo akcelerovaná grafika zahŕňajúca aj knižnice pre 2D a 3D grafiku.
- SQLite databáza pre uloženie dát aplikácie.
- Podpora pre prácu s bežnými audio, video a obrazovými formátmi.
- Využívanie GSM, EDGE, 3G či 4G sietí k telefonovaniu alebo prenosu dát.
- Správa Wi-Fi
- Knižnice pre Bluetooth a NFC[3]
- Nástroje pre jednoduchú integráciu GPS, akcelerometra, kompasu či kamery do našich aplikácií.
- Vývojové prostredie, ktoré prináša množstvo nástrojov pre ladenie, profilovanie, emulátor a *plugin* pre Eclipse IDE.

Teraz už vieme niečo viac o tom, čo nám Android ponúka. Nevieme ale ako je postavený a ako na sebe jeho časti závisia. Architektúru systému môžeme rozdeliť do nasledujúcich piatich vrstiev:

Aplikácie Android ponúka so základnou inštaláciou aj balík aplikácií. Medzi nimi nájdeme napríklad kontakty, email, kalendár alebo SMS klienta. Do tejto vrsty patria okrem týchto natívnych aplikácií aj aplikácie nainštalované užívateľom.

Aplikačný framework Aplikačný framework prináša základné stavebné prvky pre vytváranie aplikácií. Programátor má vďaka jeho API možnosť ovládať hardvér zariadenia, telefonovať, spravovať užívateľské rozhranie a mnoho iného.

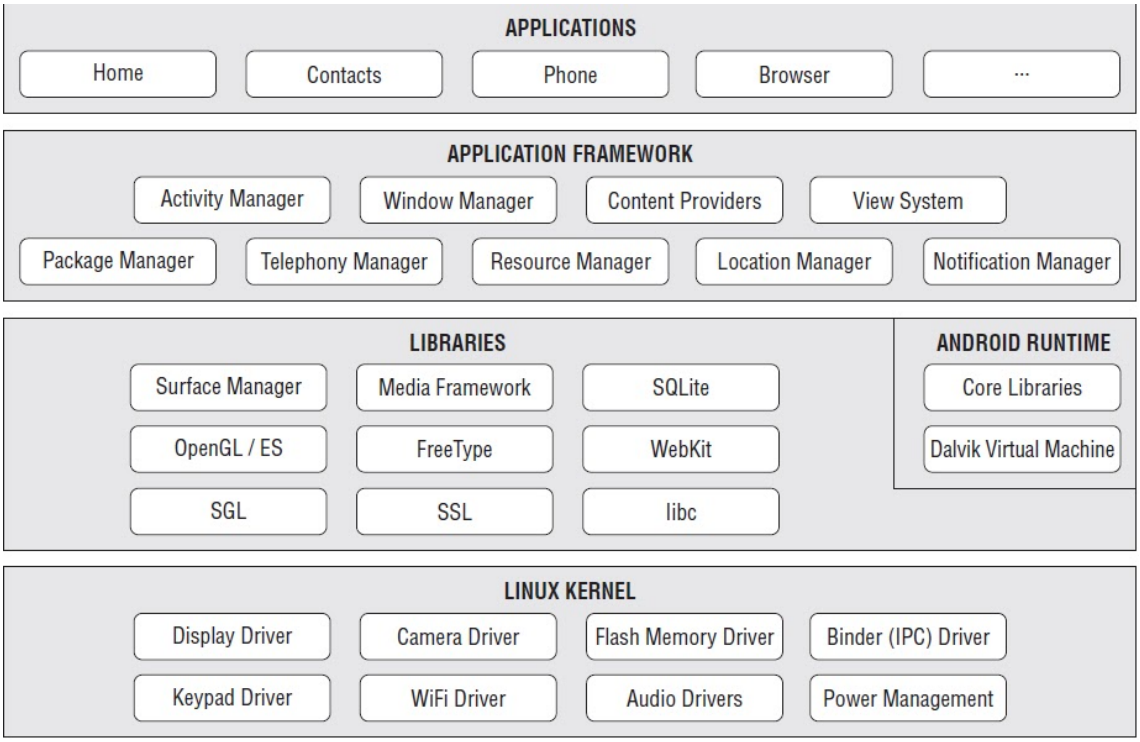
Knižnice Keďže systém Android beží na Linuxovom jadre je prirodzené, že využíva množstvo C/C++ knižníc. Medzi ne patria nasledujúce:

- systémové C knižnice
- multimediálne knižnice pre prácu s audiom a videom
- Surface Manager
- LibWebCore
- FreeType
- SQLite

Android Runtime Toto prostredie by sa dalo nazvať srdcom Androidu. Vďaka nemu nie je Android len ďalšou iplmementáciou Linuxu.

Linuxové jadro Táto vrstva sa stará okrem iného o bezpečnosť, správu pamäte, správu procesov a ovládače zariadení. Taktiež oddeľuje hardvér od zvyšku softvéru. Android využíva jadro verzie 2.6 a od verzie androidu 4.0 aj jadro Linux 3.x.

[3] Near Field Communication



Obrázek 2.1: Architektúra systému Android, prevzané z [9].

2.2.2 Základné koncepty systému

Pri vytváraní aplikácií pre systém Android používame niektoré zo zakladných stavebných blokov aplikačnej platformy. Každý blok má iné vlastnosti, chovanie a jeho použitie má špecifický význam. Tieto komponenty spája tzv. manifest aplikácie. Toto sú základné komponenty, z ktorých pozostávajú aplikácie pre Android:

- **Aktivity** – Reprezentujú jednu obrazovku aplikácie. Užívateľské rozhranie aplikácie je typicky tvorené jednou alebo viacerými aktivitami, ktoré sú medzi sebou previazané. V systéme Android ju reprezentuje trieda `Activity`.
- **Služby** – Bežia na pozadí a vykonávajú dlhotrvajúce operácie, operácie ktoré nepotrebujú užívateľské rozhranie alebo poskytujú možnosť zdieľať funkcionalitu medzi procesmi. Služba je implementovaná triedou `Service`.
- **Poskytovatelia obsahu** – Nástroj pre zapuzdrenie prístupu k dátam aplikácie. Vďaka tomuto prvku je možné svoje dáta zdielať ostatným aplikáciám. Poskytovateľ obsahu je implementovaný triedou `ContentProvider`.
- **Prijímače broadcastu** – Tieto komponenty plnia funkciu brány alebo uzlu. Načúvajú správam (zámerom) šíriacim sa systémom a na ich základe vykonávajú akcie. Prijímač broadcastu je implementovaný triedou `BroadcastReceiver`.
- **Zámery** – Systém zasielania správ, ktorým komunikujú jednotlivé komponenty aplikácie a systému medzi sebou. Implementovaný je triedou `Intent`.

Pre lepšie pochopenie významu týchto blokov si predstavme, že vytvárame aplikáciu IM klienta. Základom užívateľského rozhrania takejto aplikácie a vstupným bodom by bol zoznam kontaktov. Ten by bol implmentovaný vyššie spomínanou aktivitou. Určite budeme potrebovať ďalšiu aktivitu. Napríklad pre obrazovku, na ktorej sa bude odohrávať konverzácia. Prepínať medzi týmito obrazovkami budeme práve zámermi. Stačí zaslať zámer, v ktorom si vyžiadame konkrétnu aktivitu. Budeme chcieť uchovávať záznam konverzácie. K tomu nám pomôže práve poskytovateľ obsahu, ktorého vytvoríme nad našou databázou. Aby takáto aplikácia mala nejaký zmysel, bolo by dobré, keby sme sa vedeli pripojiť k internetu a ostať pripojení. Ideálnym kandidátom na túto úlohu je služba. Dokážeme ju spustiť aj ukončiť z aktivity pomocou zámeru. Na pozadí môže udržiavať pripojenie na server a v prípade, že prijme správu, vytvorí upozornenie či zapíše správu cez poskytovateľa obsahu do databázy. Aby sme zapojili aj prijímač broadcastu, môžeme vytvoriť taký prijímač, ktorý bude počúvať zmenu stavu pripojenia. To nám umožní spúšťať a ukončovať službu, ktorá prijíma správy na základe toho či máme aktívne pripojenie.

Unikátnou vlastnosťou systému Android je, že aplikácia má možnosť volať komponenty inej aplikácie. Napríklad, ak chceme, aby naša aplikácia používala fotoaparát, nemusíme vyvíjať vlastnú aplikáciu pre fotenie. Takúto aplikáciu má používateľ už pravdepodobne nainštalovanú. Možno ich má aj viac. Stačí, aby sme vyslali zámer, ktorým systému povieme, že chceme vyfotiť obrázok. Systém túto správu zachytí, nájde odpovedajúcu aplikáciu a spustí príslušnú aktivitu. Po fotení sa automaticky vrátime späť do našej aplikácie a pre užívateľa to vyzerá akoby bol fotoaparát súčasťou našej aplikácie.

Keďže sa s týmito aplikačnými blokmi naprieč touto prácou ešte budeme stretávať (možno s výnimkou prijímače broadcastu), pokúsim sa ich funkciu a vlastnosti popísať bližšie.

Aktivity

Aktivita (v originále *Activity*) je aplikačná komponenta, ktorá poskytuje obrazovku aplikácie, s ktorou môže pracovať užívateľ (GUI) [1]. Typicky aktivita beží v okne, ktoré plní celu obrazovku mobilného zariadenia, ale môže sa vyskytovat aj v malom okne, ktoré pláva nad ostatnými. V systéme Android je implementovaná triedou `Activity`.

Typicky je aplikácia tvorená viacerými aktivitami, ktoré su medzi sebou voľne previazané. Väčšinou býva jedna z aktivít označovaná ako hlavná. Tá sa potom využíva ako vstupný bod aplikácie. Každá aktivita môže otvoriť ďalšie aktivity. Systém potom uchováva zásobník otváraných aktivít a uzmožňuje spätnú navigáciu.

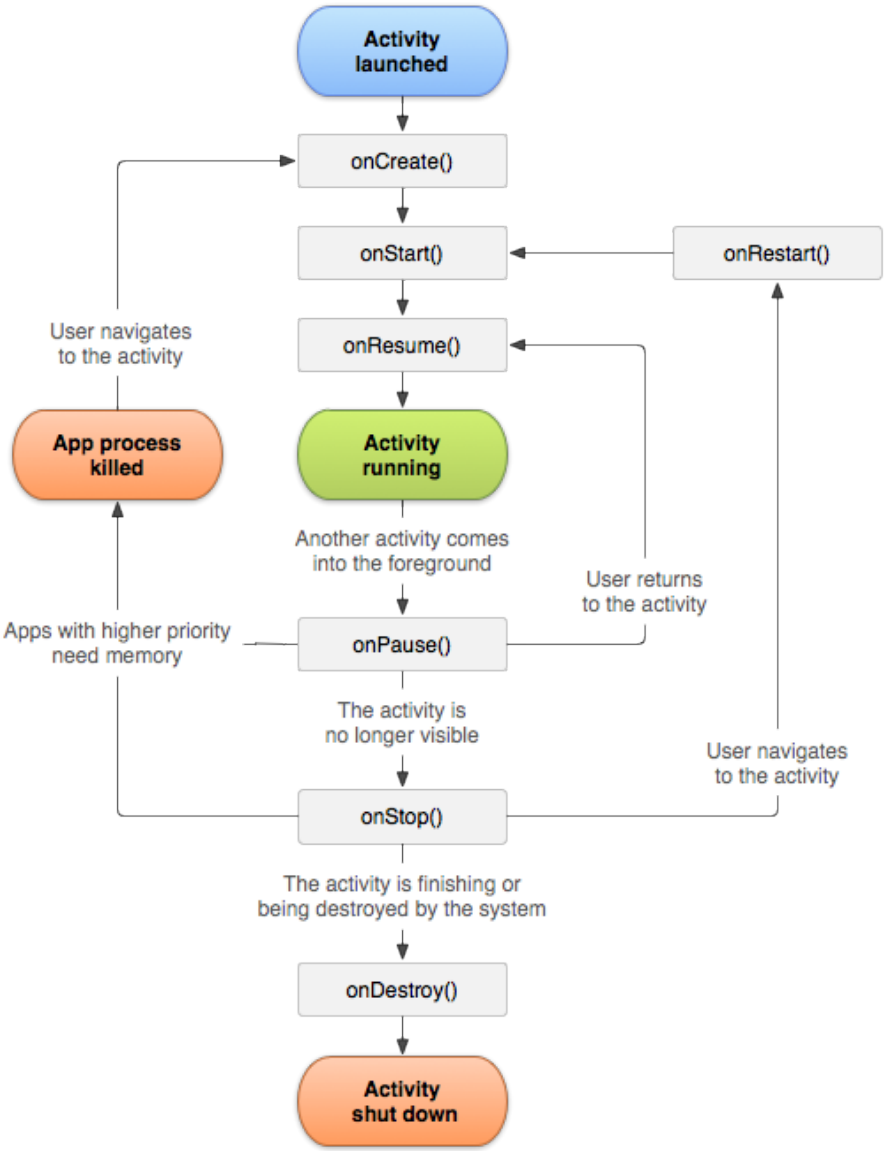
Keď je aktivita zastavená, pretože bola spustená nová, systém ju notifikuje o tejto zmene pomocou systému spätných volaní. Existuje ich niekoľko a každé z týchto volaní dáva priestor aktivite reagovať na zmenu. Napríklad pri zastavení môže uvoľniť zdroje, ktoré potom zas pri obnovení alokuje. Zmeny stavu aktivity nazývame aj jej *životným cyklom*.

Správnym využitím životného cyklu aktivity, ktoré spolieha na implementovanie spätných volaní, sme schopní vytvoriť efektívne a flexibilné aplikácie. Aktivita môže existovať v týchto troch stavoch:

- **Obnovená** – Aktivita je na popredí a má *focus*. Tento stav niekedy označujeme aj ako "bežiaca".
- **Pauznutá** – Na popredí je iná aktivita, ale táto aktivita je ešte stále viditeľná. V tomto stave je aktivita stále nažive tzn. je načítaná v pamäti a patrí pod okenný ma-

nažér. V extrémných prípadoch sa môže stať, že ju systém ukončí z dôvodu nedostatku pamäti.

- **Zastavená** – Aktivita je na pozadí. Nemá focus a nie je vidieť. Aktivita je ale stále nažive, ale nepatrí už pod okenný manažer. Môže byť ukončená v prípade, že systém potrebuje pamäť.



Obrázek 2.2: Životný cyklus aktivity, prevzané z [1].

Obrázok 2.2 ilustruje možnosti zmeny stavu aktivity. Štvorce označujú spätné volania, ktoré je možné implementovať a ktoré sa vykonajú pri zmenách stavu.

Služby

Služba (v originále *Service*) je aplikačná komponenta, ktorá vykonáva dlho bežiace operácie [8]. Služba neumožňuje vytvoriť grafické užívateľské rozhranie. Môže ju spustiť iná komponenta a ona môže ostať bežať aj po tom, čo sa tá komponenta ukonči. Komponenta má taktiež možnosť spojiť (v originále *bind*) sa so službou a komunikovať s ňou. Služba môže napríklad prehrávať hudbu alebo komunikovať po sieti, a to všetko na pozadí. Nadobúda dve podoby:

- **Spustená** – Služba sa považuje za spustenú, ak ju nejaká aplikačná komponenta spustí pomocou metódy `startService()`. Takto spustená služba môže bežať ako dlho chce a to aj po tom ukončení komponenty, ktorá ju spustila. Tento spôsob sa používa pre jednoduché operácie ktoré nepotrebuju vracať nejakú hodnotu. Napríklad sťahovanie súboru cez internet. Po dokončení operácie by sa mala služba sama ukončiť.

- **Viazaná** – Služba je "viazaná", ak sa na ňu naviaže nejaká aplikačná komponenta zavolaním `bindService()`. Táto podoba služby ponúka klient-server rozhranie, ktoré umožňuje komponente so službou obojsmerne komunikovať. Služba v tomto prípade beží iba tak dlho, ako komponenta, na ktorú je viazaná.

Napriek tomu, že popisujeme dve osobité podoby služieb, jedna služba môže kombinovať obe.

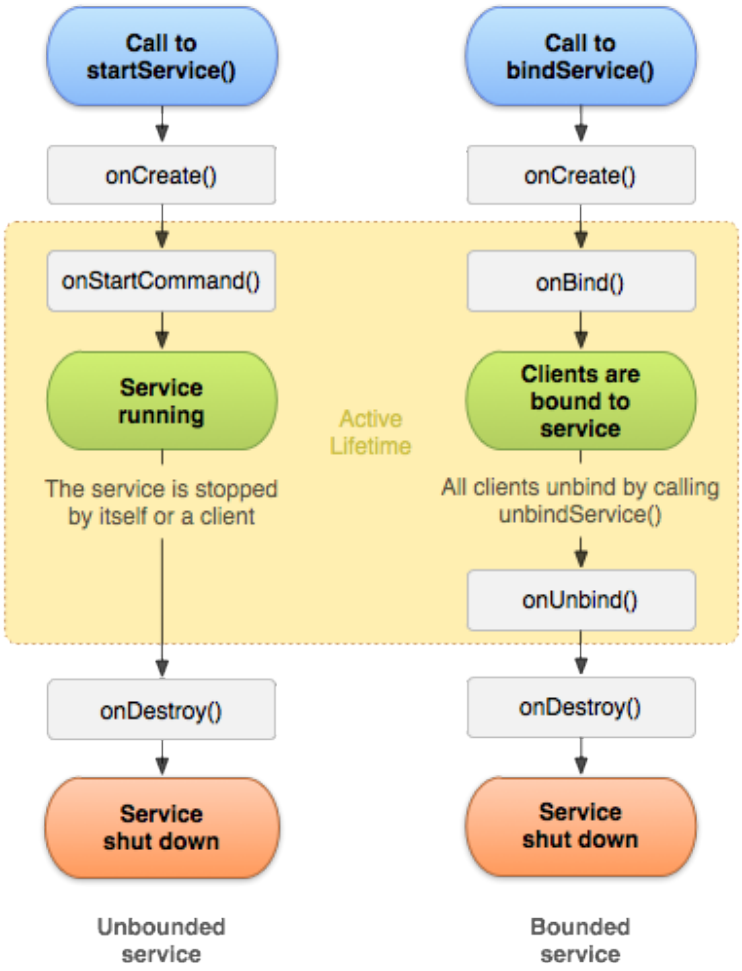
Na vytvorenie služby je potrebné vytvoriť triedu, ktora dedí od triedy `Service`. Pri jej implementácii musíme nahradiť niektoré spätné volania. Tie najdôležitejšie sú:

- **onStartCommand()** – Systém zavolá túto metódu v prípade, že ju niektorá komponenta spustila použitím `startService()`. Po dokončení je služba spustená a môže bežať "donekonečna".

- **onBind()** – Systém zavolá túto metódu v prípade, že sa s ňou chce spojiť nejaká komponenta použitím `bindService()`. Implementácia tejto metódy musí poskytnúť rozhranie, s ktorým bude komponenta komunikovať. To docieli vrátením objektu `IBinder`.

- **onCreate()** – Systém volá túto metódu pri prvom spustení služby. Volaná je v prípade spustenej aj viazanej služby.

- **onDestroy()** – Systém volá túto metódu v prípade, že sa služba už nepoužíva a končí svoju činnosť. Toto je posledné volanie, ktoré služba obdrží, preto by v ňom mala napríklad uvoľniť alokované prostriedky.



Obrázek 2.3: Životný cyklus služby, prevzané z [8].

Obrázok 2.3 ilustruje životný cyklus služby. Časť vľavo popisuje službu "spustenú" a časť vpravo službu "viazanú".

Poskytovatelia obsahu

Poskytovateľ obsahu (v originále *Content provider*) sprostredkováva prístup k štrukturovaným dátam [5]. Dáta zapuzdruje a poskytuje mechanizmy na ich ochranu. Poskytovateľ tvorí štandardné rozhranie, ktoré spája dáta jedného procesu s kódom bežiacim v inom procese.

Ak chceme pristúpiť k datám, ktoré sú zapuzdrené poskytovateľom obsahu, využívame na to objekt `ContentResolver`. Ten komunikuje s inštanciou triedy, ktorá dedí od triedy `ContentProvider`. Poskytovateľ takto prijíma požiadavky na vykonanie rôznych dátových operácií. Aby `ContentResolver` vedel, ktorého poskytovateľa má pri požiadavkoch adresovať, používa platforma Android systém *URI*[4] (označujú sa aj ako *content URI*). URI pozostáva zo symbolického názvu poskytovateľa (jeho *autority*) a označenia tabuľky prípadne cesty.

[4]Uniform Resource Identifier

Platforma Android obsahuje poskytovateľov, ktorí spravujú audio, video, kontaktné informácie či kalendár. V prípade, že chceme vytvoriť vlastného poskytovateľa, musíme implementovať triedu, ktorá dedí od triedy `ContentProvider`.

Zámery

Tri z hlavných komponent systému Android sú aktivované pomocou špeciálnych správ, ktoré nazývame *zámery* (v originále *Intent*) [7]. Systém zámerov poskytuje prostriedok pre neskorú väzbu, za behu, medzi aplikačnými komponentami, z jednej alebo viacerých aplikácií. Zámer je objektom triedy `Intent` a uchováva statické dáta s popisom operácie, ktorú chceme vykonať alebo popisom udalosti, ktorá sa udiala v systéme. Okrem toho môže obsahovať napríklad rôzne dáta potrebné pre komponentu, ktorá zámer príjme. Prakticky môže obsahovať nasledovné polia:

- **Názov komponenty** – Plne kvalifikovaný názov triedy, ktorá má spracovať zámer a názov balíčku, v ktorom sa trieda nachádza. Reprezentovaný je objektom `ComponentName`. Toto pole nie je povinné. Ak nie je zadané, systém nájde príjemcu na základe ostatných polí.
- **Akcia** – Textový reťazec pomenovávajúci akciu, ktorá sa má vykonať. Využiť sa pritom môžu akcie definované v systéme alebo je možné vytvoriť vlastné.
- **Dáta** – URI a *MIME*[5] typ dát, na ktoré je zameraná akcia. Na základe typu MIME môže systém sám zvoliť aplikáciu, ktorá s týmto typom dát dokáže pracovať. Napríklad, ak užívateľ chce zobraziť obrázok, systém zobrazí zoznam aplikácií, ktoré sú tohto úkon schopné.
- **Kategória** – Textový reťazec obsahujúci dodatočné dáta o type aplikácie, ktorá by sa mala zavolať. Dostupné kategórie obsahuje trieda `Intent`.
- **Doplnky** – Ide o dodatočné dáta, predané komponente, ktorá spracuje zámer. Organizované sú ako kľúč-hodnota.
- **Príznaky** – Rôzne príznaky riadiace to, ako má byť komponenta spustená. Väčšina z nich sa viaže na aktivity a všetky sú definované v triede `Intent`.

Zámery delíme do dvoch kategórií na základe toho ako sa spracovávajú:

- **Explicitné zámery** – Majú nastavený názov komponenty a to určuje ich cieľ. Keďže tieto názvy väčšinou nie sú obecne známe, využívajú sa tieto zámery predovšetkym na komunikáciu vovnútri aplikácie.
- **Implicitné zámery** – Nemajú definovaný cieľ (názov komponenty je prázdny). Využívajú sa hlavne na volanie komponent iných aplikácii.

Ak chceme umožniť, aby komponenta bola adresovaná implicitným zámerom, musíme definovať jej filter zámerov. Ten môže obsahovať akcie, kategórie a dáta, ktoré komponenta dokáže obslúžiť. Filter zámerov sa definuje v manifeste aplikácie.

[5] viacúčelové rozšírenie internetovej pošty, angl. multipurpose internet mail extension

Manifest aplikácie

Každá aplikácia musí obsahovať súbor AndroidManifest.xml vo svojej koreňovej zložke [3]. Manifest obsahuje všetky podstatné informácie o aplikácii a prezentuje ich takto systému Android. Okrem iného manifest plní tieto úlohy:

- Pomenováva Java balíček aplikácie. Tento názov slúži ako unikátny identifikátor aplikácie.
- Popisuje komponenty, z ktorých pozostáva aplikácia a pomenováva triedy, ktorými sú implementované. Systému poskytuje informácie o tom, akými schopnosťami komponenta oplýva a za akých podmienok môže byť spustená.
- Určuje, v ktorých procesoch budú komponenty bežať.
- Deklaruje prístupové prava, ktoré aplikácia potrebuje pre svoj beh.
- Deklaruje ptístupové práva, ktoré iné aplikácie potrebujú na to, aby mohli pristupovať ku komponentám aplikácie.
- Obsahuje zoznam tried, ktoré sa využívajú na profilovanie a testovanie. Tieto triedy sú prítomné počas vývoja a sú odstraňované pri publikovaní aplikácie.
- Deklaruje minimálnu verziu API, ktorá je potrebná pre beh aplikácie.
- Obsahuje zoznam knižníc, ktoré aplikácia vyžaduje.

2.2.3 Vývojové nástroje platformy Android

Platforma Android, okrem iného, prináša aj rozsiahly balík vývojových nástrojov. V texte tejto práce sa môžeme stretnúť aj s označením *SDK* (z angl. software development kit). V balíku môžeme nájsť napríklad *debugger*, knižnice, emulátor a vzorové kódy aplikácií. Vývoj je možný na platformách Linux, Mac OS X aj Windows. Pre efektívny vývoj môžeme využiť vývojové prostredie Eclipse s nainštalovaným ATD[6] pluginom. V ňom môžeme, okrem tvorby kódu, napríklad upravovať XML zdroje alebo vytvárať grafický návrh aktivít. Aplikácie pre systém Android sú písané v jazyku Java. Nástroje SDK tento kód preložia a spoločne s ďalšími zdrojmi (textové reťazce, obrázky, atd.) zabalia do jedného súboru s príponou *.apk*.

Dôležitým nástrojom v tomto balíku je *adb*[7]. Pozostáva z troch častí:

- Klient, ktorý sa nachádza v počítači programátora. Je možné ho spustiť v príkazovom riadku.
- Server, ktorý beží na pozadí v počítači programátora. Stará sa o komunikáciu medzi klientom a *démonom*.
- Démon, ktorý beží na pozadí v mobilnom zariadení alebo v emulátore.

Tento nástroj má mnoho funkcií. Okrem iného ním môžeme inštalovať aplikácie, prenášať súbory, zobrazovať logy aplikácií a pristupovať k príkazovému riadku zariadenia.

[6] Android Development Tools
[7] Android Debug Bridge

Kapitola 3

Analýza a špecifikácia požiadavkov mobilnej ERP aplikácie

Cieľom tejto práce je vytvoriť framework, čiže sadu podporných nástrojov pre jednoduché vytvorenie mobilnej aplikácie podnikového informačného systému. Jednou z hlavných čŕt je možnosť jednoduchého pridávania modulov k existujúcej aplikácii. Predtým, než som sa pustil do návrhu frameworku, snažil som sa zistiť, čo všetko tvorí ERP systém a čo všetko musí programátor zaistiť, aby mohol vytvoriť mobilnú aplikáciu k takému systému. Študoval som informačný systém myWAC. Zistenia sú zhrnuté do týchto základných blokov:

- **Dáta** - štrukturované uloženie množstva dát
- **Synchronizácia** - automatická aktualizácia dát
- **Obslužný kód** - aplikačná logika, spracovanie dát
- **GUI** - zobrazenie dát, kontrola vstupných dát

Mojou úlohou je podporovať vytváranie takýchto aplikácií. Preto sa pri špecifikácii nesmiem zamerať len na požiadavky ERP aplikácie, ale aj na požiadavky programátora, ktorý takéto aplikácie tvorí.

3.1 Dáta a ich synchronizácia

Ako každý informačný systém aj ERP systém potrebuje zabezpečiť dáta, nad ktorými bude pracovať. Naskýtajú sa nám dve možnosti:

- **online prístup** - výhodou je práca nad skutočnými dátami informačného systému, nevýhodou je nutnosť internetového pripojenia pre fungovanie aplikácie a množstvo komunikácie na sieti
- **lokálna kópia** - výhodou je možnosť pracovať v offline režime, nevýhodou je nutnosť synchronizácie dát a väčšie nároky na úložný priestor mobilného zariadenia

Rozhodol som sa pre druhú možnosť. Dáta chceme mať vždy a všade nezávisle na tom, či sme v dosahu GPRS alebo Wi-Fi siete. Je neprípustné, aby sa manažér na ceste nedostal k svojim kontaktom alebo skladník nemohol naskladniť tovar. Dáta teda budú ukladané lokálne. K dátam potrebujeme vedieť pristupovať z rôznych modulov aplikácie. Dáta chceme mať čo

najaktuálnejšie, preto potrebujeme vytvoriť systém synchronizácie s hlavnou databázou informačného systému. Musíme myslieť na obmedzené prostriedky mobilných zariadení a preto minimalizovať množstvo prenášaných a uložených dát. Keďže sa môže jednať o citlivé podnikové dáta, mal by byť možný šifrovaný prenos. Nesmieme zabudnúť na ďalší prvok zadania tejto práce a to, že okrem nášho úložiska by sme mali vedieť utilizovať aj niektoré existujúce podsystémy platformy Android.

S pridávaním nových modulov sa budú meniť aj požiadavky na synchronizované dáta. Pretože náš framework má byť jednoducho rozšíriteľný, musíme na tento aspekt myslieť. Pridanie nových objektov pre synchronizáciu musí byť možné bez úpravy zdrojových kódov aplikácie.

3.2 Obslužný kód a GUI

Aby akýkoľvek modul ERP systému plnil nejakú funkciu, bude okrem dát potrebovať aj obslužný kód a pravdepodobne aj užívateľské rozhranie. Pod obslužným kódom myslíme aplikačnú logiku aplikácie (funkcie, výpočty, spracovanie dát). Je to kód, ktorý vytvorí objednávku alebo vystaví faktúru. Aby k týmto úkonom užívatelia mohli pristupovať, potrebuje aplikácia užívateľské rozhranie. Programátor by mal mať možnosť jednoducho tento kód a GUI vytvoriť. Môže sa pritom jednať o pár jednoduchých funkcií, ale tiež aj o zložitú spleť tried. Užívateľské rozhranie musí zvládnuť vytvoriť jednoduchý formulár, ale aj zobraziť veľké množstvo dát.

3.3 Modularita frameworku

Ako som už spomínal, chceme, aby náš framework podporoval pridávanie modulov. V ideálnom prípade by pridanie takéhoto modulu malo znamenať nulový zásah do existujúcej aplikácie. To sa netýka len zdrojových kódov. Chceme, aby bolo možné na mobilné zariadenie inštalovať len konkrétny modul. Užívateľ tak nebude musieť preinštalovať celú aplikáciu pri každej malej zmene niektorého z modulov.

Kapitola 4

Návrh a implementácia frameworku a klientskej aplikácie

Aplikácia je vyvíjaná pre systém Android, preto sa musíme zamyslieť nad možnosťami a obmedzeniami, ktoré prináša.

4.1 Reprezentácia dát vo formáte JSON

Pre reprezentáciu dát v rôznych častiach aplikácie bol zvolený formát JSON. Oproti XML je tento formát jednoduchší a jeho spracovanie je rýchlejšie. Tieto vlastnosti ho robia ideálnym kandidátom pre webové a mobilné platformy. JSON formát je založený na dvoch štruktúrach:

- Kolekcia párov názov/hodnota. V rozličných jazykoch býva označovaná aj ako objekt alebo asociatívne pole.
- Triedený zoznam hodnôt. Označovaný býva aj ako pole či vektor.

4.2 Uloženie a zdieľanie dát aplikácie

Keďže aplikácia potrebuje ukladať množstvo štrukturovaných dát, jasnou voľbou úložiska je databáza. Android obsahuje integrovaný databázový systém SQLite, ktorý je pre naše požiadavky postačujúci. Mali by sme vedieť, že SQLite, na rozdiel od iných databázových systémov, využíva systém dynamického typovania. Typ dát nie je pevne určený stĺpcom tabuľky, v ktorej sú dáta uložené. Základné dátové typy SQLite databázy sú:

- NULL – Dátový typ NULL hodnoty.
- INTEGER – Ukladá celé číslo so znamienkom na 1, 2, 3, 4, 6, alebo 8 bajtov v závislosti od veľkosti.
- REAL – Ukladá číslo s pohyblivou radovou čiarkou. Uložené býva na 8 bajtov.
- TEXT – Slúži na uloženie textových reťazcov.
- BLOB – Využíva sa primárne na uloženie nešpecifikovaných binárnych dát.

4.2.1 Vytváranie tabuliek

Dátová vrstva ERP systému štandardne obsahuje množstvo tabuliek (objektov), ktorých počet sa často zvyšuje. Jedným z požiadavkov na túto aplikáciu je práve schopnosť reagovať na tieto zmeny. Musí vedieť vytvoriť novú tabuľku na základe poskytnutej definície. Definícia musí obsahovať:

- názov tabuľky
- názvy stĺpcov
- dátové typy stĺpcov

Okrem toho, definícia každého stĺpca môže obsahovať:

- predvolené hodnoty
- veľkost
- obmedzenie na NULL hodnotu
- obmedzenie na unikátnosť hodnoty

Definícia tabuľky je realizovaná ako JSON objekt, ktorého každá vlastnosť je zároveň názvom tabuľky. Hodnota vlastnosti je vždy objekt, ktorý musí obsahovať vlastnosť **fields**. Pod kľúčom fields je uložený objekt, ktorého každá vlastnosť je názvom stĺpca tabuľky. Objekt stĺpca môže obsahovať tieto vlastnosti:

- **type** – Jediná povinná vlastnosť stĺpca. Povolené hodnoty sú textové reťazce TEXT, REAL a INTEGER.
- **size** – Môže byť použitý na obmedzenie veľkosti stĺpca. Prípustná hodnota je kladné celé číslo. Nie je povinný.
- **notNull** – Obmedzenie, ktoré určuje možnosť uloženia NULL hodnoty. Jeho hodnota môže byť len Booleova hodnota *true* (označovaná aj ako pravda) alebo *false* (označovaná aj ako nepravda). V prípade, že nie je nastavená, berie sa ako nepravda.
- **unique** – Obmedzenie na unikátnosť hodnoty, vkladanej do stĺpca tabuľky. Môže nadobúdať dvoch hodnôt, a to pravda a nepravda. Ak nie je nastavená berie sa ako nepravda.
- **default** – Nastavuje predvolenú hodnotu stĺpca. Vlastnosť nie je povinná.

Nasledujúci príklad nám ukazuje jednoduchú definíciu tabuľky *goods*. Tabuľka ukladá názov tovaru a jeho cenu. Názov musí byť unikátny, nesmie obsahovať NULL hodnotu a je dátového typu TEXT. Cena je dátového typu REAL a v prípade, že pri vytváraní záznamu v tabuľke, nie je zadaná hodnota, hodnota sa nastaví na 0.

Môžeme si všimnúť, že príklad neobsahuje definíciu primárneho kľúča. ERP klient pri vytváraní tabuľky podľa definície automaticky pridáva ďalšie dva stĺpce:

- **_id** – Plní funkciu primárneho kľúča. Jeho hodnota sa tvorí automaticky.
- **_oid** – Slúži na uloženie primárneho kľúča z databázy ERP systému. Tým preväzuje záznam tabuľky medzi mobilnou aplikáciou a hlavným systémom.

```
{
  goods : {
    fields : {
      name : {
        type : "TEXT",
        size : 50,
        unique : true,
        notNull : true,
      },
      price : {
        type : "REAL",
        default : 0.0
      }
    }
  }
}
```

Obrázek 4.1: Príklad zápisu definície tabuky

Aplikácia definíciu spravuje a vygeneruje SQL príkaz, ktorým vytvorí tabuľku v databáze. Neskôr sa do tejto tabuľky budú synchronizovať dáta ERP systému. Pre lepšiu predstavu uvedieme SQL príkaz, ktorý aplikácia vygeneruje na základe vyššie uvedeného príkladu.

```
CREATE TABLE goods (  _id INTEGER PRIMARY KEY AUTOINCREMENT,
                      _oid TEXT,
                      name TEXT(50) UNIQUE NOT NULL,
                      price REAL DEFAULT 0.0
                   )
```

4.2.2 Zdieľanie dát aplikácie medzi modulmi

Bežne platforma Android, pre zvýšenie bezpečnosti mobilných zariadení, neumožňuje pristupovať jednej aplikácii k dátam inej aplikácie. To je docielené linuxovými právami prístupu a tým, že každá aplikácia má vlastného užívateľa, pod ktorým beží. Aby ale tvorcovia Androidu umožnili posunúť vývoj mobilných aplikácií o krok ďalej, vytvorili systém, ktorým sa v systéme Android dáta zdieľať dajú. Na tento účel využívame prvok systému, tzv. poskytovateľa obsahu.

Poskytovateľ obsahu bol už v tejto práci popísaný. Naša implementácia bude využívať len dve metódy a to metódu `query()` a metódu `update()`. Oproti bežnému použitiu budú ich parametre upravené tak, aby bolo možné volať SQL príkaz priamo. Nie je to najbezpečnejšia varianta, ale v opačnom prípade by bolo nutné meniť implementáciu poskytovateľa podľa toho, ako by sa menili požiadavky na náročnosť SQL príkazov.

Metóda `query()` je deklarovaná takto:

```
public Cursor query(Uri uri, String[] projection, String selection,
          String[] selectionArgs, String sortOrder)
```

Implementácia použitá v aplikácii využíva len parameter `selection` a `selectionArgs`. Do parametra selection vkladáme kompletný SQL požiadavok. V prípade, že chceme využiť možnosť nahradenia znaku `?` hodnotou, môžeme túto hodnotu vložiť do parametra selectionArgs. Výsledok je vrátený štandardne ako `Cursor`.

Metóda `update()` je deklarovaná takto:

```
public int update(Uri uri, ContentValues values, String where,
          String[] selectionArgs)
```

V tomto prípade aplikácia využíva iba parametre `where` a `selectionArgs`. Parameter where slúži na zadanie SQL príkazu. Tento príkaz po spustení nevráti žiadne dáta, preto treba zvážiť, ktoré príkazy budeme touto metódou spúšťať. Znova, v prípade, že chceme využiť možnosť nahradenia znaku `?` hodnotou, môžeme využiť parameter selectionArgs.

4.3 Synchronizácia dát aplikácie

Pojmom synchronizácia budeme označovať synchronizáciu dát medzi ERP systémom a mobilnou aplikáciou. Pre tento účel využijeme prostriedky `AccountManager` a `SyncAdapter`. Vďaka nim je mobilná aplikácia schopná vytvoriť v systéme účet a synchronizačný adaptér, ktorý sa bude starať o synchronizáciu dát. Vlastník mobilného zariadenia s Androidom takéto účty určite pozná už z aplikácií Google, Skype či Facebook.

V tejto práci rozlišujeme dva spôsoby, akými sa synchronizované dáta ukladajú a od nich sa odvíja aj spôsob, akým sa synchronizujú a prenášajú.

- **ukladanie objektu do vlastnej databázy** – pri tejto metóde je objekt uložený v takom tvare, ako je na serveri
- **ukladanie do zdieľanej databázy** – pri tejto metóde sa dáta musia prispôsobiť požiadavkom poskytovateľa obsahu.

4.3.1 Vytvorenie účtu v systéme Android

Android ponúka centralizovaný register užívateľských online účtov. Vďaka nemu má užívateľ prehľad o online službách bežiacich na jeho zariadení a dokáže ich z jedného miesta spravovať. Aplikácia si zas týmto spôsobom vie uložiť informácie potrebné k autentifikácii. Okrem toho sa účet využíva aj na identifikáciu dát v zdieľaných prostriedkoch. Napríklad pri synchronizácii kontaktov potrebujeme vedieť, ktorý kontakt patrí ku ktorému účtu. Ak by sme túto informáciu nemali, po zmazaní účtu by v zozname kontaktov ostalo množstvo zbytočných dát. Na prístup k účtom sa využíva komponenta *AccountManager*. Implementovaná je triedou rovnakého názvu. Na vytvorenie vlastného účtu musíme implementovať rozhrania využívané práve komponentou AccountManager. Prvá časť, ktorú musíme vytvoriť je aktivita, ktorá sa zobrazí užívateľovi pri pridávani účtu. Pomocou nej získame prihlasovacie údaje a ďalšie potrebné informácie. Vytvoríme ju zdedením triedy `AccountAuthenticatorActivity`. Obsahuje metódu `setAccountAuthenticatorResult(android.os.Bundle)`, ktorá nastaví výsledok, ktorý sa vráti tomu, kto volal túto aktivitu. Pred ukončením aktivity by sa mal týmto spôsobom výsledok nastaviť.

Ďalším krokom je implementovanie abstraktnej triedy `AbstractAccountAuthenticator`. V našom projekte je implementovaná ako trieda `Authenticator`. Má sedem abstraktných metód, ktoré treba prepísať:

- **addAccount** – pridá účet zadaného typu,
- **confirmCredentials** – slúži na overenie prihlasovacích údajov,
- **editProperties** – otvára aktivitu, ktorá slúži na upravenie vlastností účtu,
- **getAuthToken** – vyžiada autentifikačný reťazec pre konkrétny účet,
- **getAuthTokenLabel** – vyžiada lokalizovaný popisok pre zadaný typ účtu,
- **hasFeatures** – kontroluje, či daný typ účtu podporuje rôzne funkcie,
- **updateCredentials** – aktualizuje lokálne uložené prihlasovacie údaje.

Pre túto prácu sú podstatné iba dve. Metóda `addAccount()` a `getAuthToken()`. Prvá z nich je deklarovaná nasledovne:

```
public Bundle addAccount(AccountAuthenticatorResponse response,
        String accountType, String authTokenType,
        String[] requiredFeatures, Bundle options)
```

Táto metóda vytvorí explicitný zámer na aktivitu, ktorá si vyžiada potrebné údaje a vytvorí účet. Zámer musí obsahovať `AccountAuthenticatorResponse` pod kľúčom `KEY_ACCOUNT_MANAGER_RESPONSE`.

```
public Bundle getAuthToken(AccountAuthenticatorResponse response,
        Account account, String authTokenType, Bundle options)
```

Predpokladá sa, že pri online komunikácii aplikácia využíva nejaký druh autentifikačného reťazca. Tento reťazec sa potom využíva dovtedy, kým mu neskončí platnosť. Tento prístup je výhodnejší ako zasielanie prihlasovacích údajov pri každom požiadavku na server. Metóda `getAuthToken` má možnosť prihlásiť sa na server a autentifikovať sa. V prípade chyby prihlasovacích údajov alebo inej, môže podniknúť prislušné kroky. Napríklad spustiť aktivitu, ktorá sa postará o nové zadanie hesla.

Teraz, keď už máme triedu `Authenticator`, potrebujeme vytvoriť službu, v ktorej bude bežať na pozadí. Budeme ju nazývať `AuthenticatorService`. Táto trieda je veľmi jednoduchá. Stačí vytvoriť inštanciu triedy Authenticator v metóde `onCreate()`, potom už len v metóde `onBind()` na tej inštancii zavolať metódu `getIBinder()`. Službu musíme pridať do manifestu s týmto filtrom a metadátami:

```
1  <intent-filter>
2     <action android:name="android.accounts.AccountAuthenticator" />
3   </intent-filter>
4   <meta-data android:name="android.accounts.AccountAuthenticator"
5              android:resource="@xml/authenticator" />
```

V tomto príklade vidíme, že atribút `android:resource` ukazuje na ďalši XML súbor. Tento súbor musí byť v takomto tvare:

```
<account-authenticator xmlns:android="http://schemas.android.com/apk/res/
    android"
    android:accountType="typeOfAuthenticator"
    android:icon="@drawable/icon"
    android:smallIcon="@drawable/miniIcon"
    android:label="@string/label"
    android:accountPreferences="@xml/account_preferences"
/>
```

Atribút `android:accountPreferences` nie je povinný a v tomto projekte sa ani nevyužíva. Týmto sme zaistili všetko pre to, aby užívateľ mohol vytvoriť účet našej aplikácie na mobilnom zariadení. Účet však nevie synchronizovať dáta. Na to musíme vytvoriť synchronizačný adaptér.

4.3.2 Vytvorenie synchronizačného adaptéru v systéme Android

Každý účet môže združovať množstvo týchto adaptérov. Na vytvorenie synchronizačného adaptéru potrebujeme implementovať abstraktnú triedu `AbstractThreadedSyncAdapter`. Skutočnú synchronizáciu vykonáva metóda `onPerformSync()`. Jej volanie má v réžii systém Android. Ručne sa dá zaslať iba požiadavok na synchronizáciu, pričom skutočná synchronizácia sa vykoná až po uplynutí určitého času. Požiadavok generuje aj systém, a to buď v istých časových intervaloch, alebo v prípade zmeny dát, s ktorými je adaptér zviazaný. Ďalším krokom je vytvorenie jednoduchej služby. Tak, ako v minulom prípade, v metóde `onCreate()` vytvoríme inštanciu adaptéru. Potom v metóde `onBind()` na tejto inštancii zavoláme metódu `getSyncAdapterBinder()`. Službu je nutné pridať do manifestu s týmto filtrom a metadátami:

```
  <intent-filter>
    <action android:name="android.content.SyncAdapter" />
  </intent-filter>
  <meta-data android:name="android.content.SyncAdapter"
             android:resource="@xml/syncadapter" />
```

Atribút `android:resource` znova ukazuje na XML súbor. Tento súbor by mal byť v tomto formáte:

```
<sync-adapter xmlns:android="http://schemas.android.com/apk/res/android"
    android:contentAuthority="authority"
    android:accountType="accountType"
    android:userVisible="true|false"
    android:supportsUploading="true|false"
    android:allowParallelSyncs="true|false"
    android:isAlwaysSyncable="true|false"
    android:syncAdapterSettingsAction="ACTION_OF_SETTINGS_ACTIVITY"
/>
```

Dôležité sú hlavne tieto dva atribúty:

- **android:contentAuthority** – obsahuje autoritu poskytovateľa dát, ktoré synchronizuje,
- **android:accountType** – obsahuje typ účtu, ku ktorému adaptér patrí.

Pre lepšiu predstavu si ukážeme konkrétny adaptér, ktorým sa synchronizujú kontakty.

```
<?xml version="1.0" encoding="utf-8"?>
<sync-adapter xmlns:android="http://schemas.android.com/apk/res/android"
    android:contentAuthority="com.android.contacts"
    android:accountType="@string/authtoken"
    android:supportsUploading="true"
    android:userVisible="true"
/>
```

4.3.3 Synchronizácia obecného objektu

Obecným objektom rozumieme akýkoľvek objekt ERP systému. Tento objekt sa ukladá do databázy klientskej aplikácie v takej podobe, ako ho dostane od serveru. K tomuto úkonu slúži samostatný synchronizačný adaptér. Celý proces prebieha z pohľadu klientskej aplikácie v tychto krokoch:

1. Získanie definície tabuliek, ktoré sa synchronizujú.
2. Porovnanie definície oproti vytvoreným tabuľkám. Na základe prípadných zmien sa vykonajú príslušné kroky (vytvorenie a zahodenie tabuľky, zmena definície stĺpca).
3. Vyhľadanie upravených objektov. Pre každú zo synchronizovaných tabuliek sa získa zoznam upravených záznamov.
4. Odoslanie upravených objektov na server.
5. Získanie dát od serveru. Server prijme klientské dáta a vygeneruje dáta pre klienta.
6. Spracovanie a uloženie dát.

Na reprezentáciu dát je použitý formát JSON. Objekt predstavuje záznam v tabuľke. Názov každej vlastnosti odpovedá stĺpcu tabuľky. Na základe toho sa vygeneruje SQL príkaz, ktory uloží hodnoty do správnych stĺpcov tabuľky. Každý takýto objekt musí obsahovať vlastnosti `_id` a `_oid`. Vlastnosť `_id` reprezentuje primárny kľúč, ktorý sa používa v mobilnom zariadení a `_oid` primárny kľúč používaný na serveri. Príklad takýchto vidíme na obrázku 4.2.

Špeciálnym prípadom je vytvorenie nového záznamu na mobilnom zariadení. Vtedy je vyplnená iba vlastnosť `_id`. Po odoslaní záznamu na server sa záznam vytvorí a vráti sa aj s vyplneným `_oid`, ktoré sa uloží. Tým sa záznam klientskej aplikácie a serveru spáruje a proces vytvárania sa považuje za kompletný.

4.3.4 Synchronizácia kontaktov a kalendára

Synchronizácia kontaktov a kalendára je implementovaná osobitným synchronizačným adaptérom pre každý zo subsystémov. Tieto prvky synchronizujeme zvlášť, pretože majú rôzne požiadavky, ktoré musíme splniť. O prístup k týmto dátam sa stará poskytovateľ obsahu.

```
{
   "_id":null,
   "_oid":"000000000064",
   "FirstName":"Jozef",
   "LastName":"Novak",
   "Telephone":"666 123 456",
   "Mobil":"798 654 654",
   "Pozice":"",
   "Ulice":"",
   "PSC":"",
   "Mesto":"",
   "Poznamky":"",
   "Customer":"780000000014",
   "Fax":"212 345 789",
   "Email":"",
   "Stat":"000000000005",
   "Datum_Narozeni":"1.1.1950",
   "Titul":"Ing.",
   "Deleted":false,
   "Sys_Created":"5.2.2004 16:31:30",
   "Sys_Modified":"1.6.2012 16:34:26"
}
```


Obrázek 4.2: Príklad dát obecného objektu.

Informácie o tabuľkách, z ktorých tieto systémy pozostávajú nám poskytujú tzv. *kontraktor* triedy (v originále contract class). Tie obsahujú konštanty s názvami stĺpcov, hodnotami, ktoré tieto stĺpce používaju a URI pre adresovanie k dát. Princíp synchronizácie je podobný ako u obecného objektu s nasledujúcimi odlišnosťami:

- Definícia tabuliek sa nezískava zo serveru. Údaje v tabuľkách sú známe vopred z dokumentácie.
- Objekt na mobilnom zariadení nemusí odpovedať objektu na serveri. Dáta, ktoré sa prenášajú môžu byť zoskupením viacerých objektov.
- Dáta je nutné konvertovať na strane serveru do formátu, ktorý očakáva synchronizačný adaptér.

Systém kontaktov

Systém kontaktov je komponenta platformy Android, ktorá slúži na správu dát o osobách [6]. Tieto dáta vidíme napríklad pri spustení vstavanej aplikácie Kontakty. Informácie o kontakte sú uložené v troch tabuľkách. Každá z nich má svoju triedu kontraktora, podľa ktorej sa tabuľka bežne nazýva. Kontakty využívajú týchto kontraktorov:

- **ContactsContract.Contacts** – Tabuľka `Contacts` reprezentuje jednu konkrétnu osobu. Agreguje záznamy z tabuľky `RawContacts`.

- **ContactsContract.RawContacts** – Tabuľka `RawContacts` obsahuje informácie o osobe viažúce sa ku konkrétnemu účtu.
- **ContactsContract.Data** – Tabuľka `Data` obsahuje dáta ako telefónne číslo či emailová adresa.

Systém kalendára

Systém kalendára slúži na prácu s udalosťami užívateľa [4]. Užívateľ môže mať množstvo kalendárov patriacich k rôznym účtom. Vďaka API, ktoré systém Android ponúka, môžeme vytvárať či upravovať kalendár, udalosti, pripomenutia a ďalšie prvky systému kalendára. Pozostáva z väčšieho množstva tabuliek, z ktorých každú reprezentuje trieda kontraktora:

- **CalendarContract.Calendars** – Tabuľka `Calendars` ukladá rôzne informácie o kalendári (napr. názov, farba). Jeden záznam odpovedá jednému kalendáru.
- **CalendarContract.Events** – Tabuľka `Events` uchováva informácie o udalosti. Každá udalosť sa viaže na kalendár. Môže sa objaviť jednorazovo alebo sa môže opakovať viackrát. Agreguje záznamy z tabuliek `Instances`, `Attendees` a `Reminders`.
- **CalendarContract.Instances** – Tabuľka `Instances` obsahuje čas každého výskytu konkrétnej udalosti. Jednorazová udalosť tu bude mať iba jeden prislúchajúci záznam. Pre opakujúcu sa udalosť sa vygeneruje množstvo záznamov.
- **CalendarContract.Attendees** – Tabuľka `Attendees` ukladá informácie o návštevníkoch udalosti.
- **CalendarContract.Reminders** – Tabuľka `Reminders` obsahuje informácie o pripomenutí udalosti. Každý záznam reprezentuje jedno upozornenie. Udalosť ich môže mať viacero.

4.3.5 Riešenie konfliktov

Konfliktom sa pri synchronizácii myslí problém, ktorý vznikne pri pokuse zlúčiť rôzne verzie dát. Obvykle sa to stáva v prípade, keď sa od poslednej sycnhronizácie zmenil rovnaký objekt na dvoch (prípadne viacerých) miestach súčasne. Aby bolo možné tento problém identifikovať a do určitej miery aj vyriešiť, uchovávajú sa na serveri dáta, ktoré boli zaslané každému z klientov. Server vie na ich základe pri synchronizácii zistiť, z akých dát vychádzajú dáta prijaté od klienta. Určí vlastnosti objektu, ktoré boli zmenené na mobilnom zariadení a pokúsi sa tieto zmeny aplikovať na objekt uložený v databáze ERP systému. Takto sa dá automaticky vyriešiť aspoň konflikt, pri ktorom neboli súčasne upravené rovnaké vlastnosti objektu. V prípade, že sa zmeny týkajú rovnakých vlastností, väčšiu váhu má server, a preto mobilný klient o svoje zmeny prichádza. Dostane iba informáciu o tom, že o tieto zmeny prišiel a jeho zmena sa zálohuje na serveri. Tento prístup nie je ideálny, ale je najjednoduchší. Jednou z možností rozšírenia tejto práce je umožniť riešenie konfliktov priamo na mobilnom zariadení.

4.4 Serverová časť frameworku

Serverová časť tvorí rozhranie medzi mobilnou aplikáciou a databázou ERP systému. Implementovaná je ako jednoduchý JSON-RPC server postavený na webovom rozhraní. V

nasledujúcom texte si popíšeme metódy, ktoré je nutné implementovať na to, aby synchronizácia fungovala. Uvedený bude aj príklad konfigurácie, ktorý sa používa pri demonštrácii. Týmto spôsobom môže programátor vytvoriť vlastný server, ktorý bude kompatibilný s týmto frameworkom. Server musí implementovať metódy:

- **getDefinition()** – Metóda nemá žiaden parameter. Vracia definíciu tabuliek, ktoré sa synchronizujú. Ako taká definícia vyzerá popisuje sekcia 4.2.1.
- **syncObject(name, data)** – Metóda má dva parametre. Parameter `name` obsahuje názov tabuľky, ktorá sa synchronizuje. Parameter `data` obsahuje pole zmenených objektov. Vracia zoznam objektov zmenených na serveri. Slúži na synchronizáciu obecného objektu. Formát dát je popísaný v sekcii 4.3.3.
- **syncContact()** – Slúži na synchronizáciu kontaktov.
- **syncCalendar()** – Slúži na synchronizáciu kalendára.
- **authenticate(login, password, clientId)** – Metóda slúži na autentifikovanie klienta. Má tri parametre. Parameter `login` obsahuje prihlasovacie meno a parameter `password` obsahuje heslo. Posledný je parameter `clientId`, ktorý obsahuje špecifický identifikátor mobilného klienta. Ten sa používa na identifikáciu mobilného zariadenia, pretože jeden užívateľ môže mať viac zariadení viacero. Metóda vracia autentifikačný reťazec, ktorý sa ku komunikácii so serverom pridáva ako GET parameter. Nájdeme ho pod kľúčom `authtoken`.

V mojej implementácii správanie synchronizácie obecného objektu ovplyvňuje konfiguračný súbor `profile.json`. Obsahuje objekty, ktoré sa majú synchronizovať. Metóda `getDefinition()` je vďaka nemu schopná vygenerovať definíciu. Ku každému objektu je možné uložiť SQL podmienku, ktorá sa použije pri výbere dát z databázy. Obmedzí sa tým množstvo dát ukladaných na mobilné zariadenie. Môžeme nastaviť aj to, že nechceme pre nejaký objekt povoliť synchronizáciu smerom dovnútra. Výsledok môže vyzerať napríklad takto:

```
{
   Contact : {
      condition : "Deleted = 0",
      acceptData : true
   },
   Faktura : {
      condition : "",
      acceptData : false
   }
}
```

Toto je ale len príklad mojej implementácie. Programátor, ktorý bude používať tento framework si može vytvoriť vlastný systém.

4.5 Zabezpečený prenos dát

Štandardná komunikácia so serverom prebieha protokolom *HTTP*[1]. Jedným z požiadavkov na tento framework bola možnosť zabezpečeného prenosu dát. Preto potrebujeme mať možnosť komunikovať cez protokol *HTTPS*[2]. Môžeme využiť triedu `HttpsURLConnection`. Jej použitie je veľmi jednoduché. Problém je však so systémom overenia certifikátu. Ak na serveri používame certifikát od certifikačnej autority (CA), ktorú mobilné zariadenie nepozná, odmietne spojenie. Aby sme to obišli je nutné vytvoriť vlastný *KeyStore*. Ide o Java komponentu slúžiacu ako repozitár certifikátov. Vytvorenie je vcelku jednoduché. Stačí na to využiť triedu `KeyStore` a zavolať statickú metódu `getInstance()`. Ďalším krokom je import certifikátu. Tento krok je už o niečo náročnejší. Princíp spočíva vo vytvorení *SSL* spojenia so serverom a stiahnutí certifikátu. Ten sa uloží pomocou metódy `setCertificateEntry()`. Aby tento postup nebolo nutné opakovať pri každom spojení, uložíme KeyStore do súboru pomocou metódy `store()`.

4.6 Tvorba obslužného kódu a grafického rozhrania modulu

Mojou pôvodnou myšlienkou bolo vytvoriť systém, ktorý by umožňoval vytvoriť obslužný kód a grafické rozhranie na serveri, preniesť ich na zariadenie a tam interpretovať. Takto by sa nové moduly dali pridávať ku klientskej aplikácii bez toho, aby ju bolo nutné preinštalovať. Tento spôsob nie je možný, pretože Java, a teda aj systém Android neumožňuje interpretovanie zdrojových kódov tak ako napríklad JavaScript. Zdrojové kódy musia byť najprv preložené.

Rozhodol som sa pre iný prístup. Modul bude samostatná aplikácia pre Android. Na tento úkon sú najlepšie uspôsobené práve vývojové nástroje platformy Android. Zdrojové kódy a dizajn grafického rozhrania sa asi nedá vytvoriť jednoduchšie ako použitim Eclipse a ADT pluginu. Prvky grafického rozhrania, ktoré ponúka Android, sú pre použitie v ERP mobilnej aplikácii dostačujúce.

4.7 Rozhranie pre pridávanie modulov

Mobilná ERP aplikácia môže pozostávať z množstva modulov. Aby pri pridávaní nových modulov nebolo nutné zasahovať do existujúcej aplikácie, zvolil som prístup, pri ktorom je modul ERP systému samostatná Android aplikácia. Vytváraný framework sa postará o to, aby moduly tvorili celok.

Platforma Android umožňuje previazať rôzne komponenty rôznych aplikácií. Pri vytváraní rozhrania pre pridávanie modulov využívame práve túto vlastnosť systému Android. Základom je správne nastaviť filter zámerov aplikácie, ktorá reprezentuje modul ERP systému. V ďalšom kroku potom môžeme pomocou objektu `PackageManager` a jeho metódy `queryIntentActivities()` získať zoznam aktivít reprezentujúcich moduly ERP systému. Z tohoto zoznamu sa potom vygeneruje základná plocha mobilnej aplikácie a užívateľ odtiaľ bude mocť spustit jednotlivé moduly.

Týmto spôsobom jednoducho dostaneme súčasné aj budúce moduly ERP systému pod jednu strechu. Výsledným efektom je dojem, že všetky moduly patria jednej aplikácii, aj keď každý modul je samostatná aplikácia, ktorá sa inštaluje osobitne.

[1] Hypertext Transfer Protocol
[2] Hypertext Transfer Protocol Secure

Kapitola 5

Testovanie mobilnej aplikácie

Testovanie prebiehalo na demonštračnej aplikácii, ktorá komunikuje s informačným systémom myWAC. Boli využité dve metódy: ručné vysokoúrovňové testovanie jednotlivých komponent aplikácie a nasadenie do prevádzky medzi zamestnancov firmy *myWAC TECHNOLOGIES s.r.o.*

5.1 Ručné testovanie aplikácie

Testovanie prebiehalo počas celého vývoja aplikácie. Využíval som pritom emulátor, ktorý patrí medzi SDK platformy Android a skutočné mobilné zariadenie. Verzie systému Android sa pri tom menili od verzie Android 2.3 až po najnovšiu verziu Android 4.2. Minimálna doporučená verzia API je preto verzia 9. Ako mobilné zariadenie som využíval chytrý telefón *HTC Desire Z* a verziu systému nazývanú *CyanogenMod 7.2*.

Medzi testovanú funkcionalitu patrí:

- Vytvorenie účtu v systéme – Overená bola správna funkcia autentifikácie a uloženie prístupových údajov.
- Import certifikátu – Bolo možné vytvoriť *KeyStore* a importovať certifikát.
- Synchronizácia kalendára – Podarilo sa vytvoriť kalendár a importovať udalosti. Zmeny na mobilnom zariadení sa správne preniesli do systému. Overilo sa správne zobrazenie obyčajných aj opakujúcich sa udalostí.
- Synchronizácia kontaktov – Kontakty sa podarilo preniesť na mobilné zariadenie a overilo sa ich správne zobrazenie v zozname kontaktov.
- Synchronizácia obecného objektu – Podarilo sa vytvoriť databázu a tabuľky na základe definície. Podarilo sa naplniť tieto tabuľky dátami z ERP systému. Overené boli aj možnosti vytvorenia nových záznamov a ich prenos do systému.
- Pridanie modulu – Vytvoril sa modul ERP systému, ktorý využíva dáta cez poskytovateľa obsahu. Jeho funkčnosť sa overila nainštalovaním do systému. Overilo sa jeho správne zobrazenie medzi modulmi ERP aplikácie.

5.2 Nasadenie do prevádzky

Aplikácia bola k dispozícii pre zamestnancov firmy myWAC TECHNOLOGIES s.r.o., ktorí ju využívali primárne na synchronizáciu kontaktov a kalendára. Počas tejto fázy bola zaznamenávaná kompletná komunikácia medzi mobilnou aplikáciou a IS myWAC. Taktiež boli zberané návrhy na vylepšenie a chybové hlásenia. Zistili sa tak napríklad nedostatky ako neposkytnutie možnosti zmeniť prihlasovacie údaje po tom, čo sa zmenili v ERP systéme.

Kapitola 6

Záver

Cieľom tejto práce bolo poskytnúť sondu do vývoja ERP aplikácie na platfome Android. Povedali sme si, čo je vlastne ERP systém, ako býva realizovaná jeho implementácia a predstavili sme si niektoré z ERP systémov ponúkaných na českom trhu. Zoznámili sme sa so systémom Android a vývojom aplikácií pre túto platformu.

Výsledkom je sada nástrojov, ktoré uľahčia implementáciu mobilných ERP aplikácií a demonštračná ERP aplikácia. Pri ich tvorbe som musel preniknúť do funkcie mnohých existujúcich subsystémov platformy Android. Niektoré detajly vnútornej implementácie kontaktov či kalendára nie sú riadne zdokumentované. Pri riešení niektorých problémov som musel študovať zdrojové kódy Androidu.

Pôvodne som mal o výsledku tejto práce inú predstavu. Chcel som vytvoriť omnoho robustnejšie riešenie, ktoré by umožňovalo nasadenie nového kódu za behu a bez užívateľského zásahu. Taktiež som chcel umožniť užívateľovi meniť rôzne parametre aplikácie a odľahčiť tým programátorovi. Riešenie v takomto rozmere by však bolo veľmi náročné a pravdepodobne by presahovalo rámec tejto práce. Vysledný framework ale aj tak uľahčí programátorovi prácu v značnej miere a zdokumentované postupy mu ukážu jednu z možnych ciest, ktorými sa pri vývoji ERP aplikácii vydať.

S demonštračnou aplikáciou boli užívatelia spokojní. Vadilo im napríklad to, že systém Android niekedy nesprávne zlúči kontakty alebo že na zmenu v ERP systéme nie sú upozornení okamžite. Jednou z možností rozšírenia je práve využitie služby *Google Cloud Messaging*, ktorou by sme vedeli upozorniť užívateľa na zmenu dát na serveri a iniciovať synchronizáciu. Ďalšou by bolo napríklad riešenie konfliktov priamo na mobilnom zariadení. Aj napriek pripomienkam však boli užívatelia radi, že majú svoje dáta poruke a môžu s nimi pracovať aj mimo kancelárie.

Literatura

[1] Google: Activities [online].
http://developer.android.com/guide/components/activities.html, [cit. 20013-05-11].

[2] Google: Android, the world's most popular mobile platform [online].
http://developer.android.com/about/index.html, [cit. 20013-05-11].

[3] Google: The AndroidManifest.xml File [online].
http://developer.android.com/guide/topics/manifest/manifest-intro.html, [cit. 20013-05-11].

[4] Google: Calendar Provider [online].
http://developer.android.com/guide/topics/providers/calendar-provider.html, [cit. 20013-05-11].

[5] Google: Content Providers [online].
http://developer.android.com/guide/topics/providers/content-providers.html, [cit. 20013-05-11].

[6] Google: Cotacts Providers [online].
http://developer.android.com/guide/topics/providers/contacts-provider.html, [cit. 20013-05-11].

[7] Google: Intents and Intent Filters [online].
http://developer.android.com/guide/components/intents-filters.html, [cit. 20013-05-11].

[8] Google: Services [online].
http://developer.android.com/guide/components/services.html, [cit. 20013-05-11].

[9] Kumar, S.: Architecture of Android [online].
http://androidprogramz.blogspot.cz/2012/06/architecture-of-android-in-order-to.html, 2012-06-30 [cit. 20013-05-11].

[10] Meier, R.: *Professional Android 4 Application Development.* John Wiley & Sons, Inc., 2012, iSBN 978 1 118 10227 5.

[11] WWW stránky: Helios - podnikový informační systém, ekonomický a účetní software, systém pro veřejnou správu. http://www.helios.eu/.

[12] WWW stránky: Informační systém myWAC. http://www.mywac.cz/.

[13] WWW stránky: Podnikové informační systémy ABRA. http://www.abra.eu/.

Příloha A

Obsah CD

Priložené CD obsahuje tieto položky:

- `projects/` – zložka s kompletnými projektami
  - `myWAC_Mobile_Calendar` – synchronizačný adaptér kalendára
  - `myWAC_Mobile_Contact` – synchronizačný adaptér kontaktov
  - `myWAC_Mobile_ERP` – mobilná ERP aplikácia
  - `myWAC_Mobile_Library` – knižnica frameworku
  - `myWAC_Module1` – modul pre demonštráciu rozhrania na pridávanie modulov
- `bin/` – obsahuje preložene *.apk* balíčky

Příloha B

Manuál

Pre nainštalovanie balíčkov je nutné povoliť inštaláciu aplikácii z neznámych zdrojov. Balíček je potom mozné nainštalovať dvoma spôsobmi:

- Pomocou nástroja adb ktorý je súčasťou Android SDK. V príkazovom riadku zadáme `adb install <NazovBaliku.apk>`.
- Nakopírovaním balíčku na kartu SD a spustením.

Najdôležitejši je balíček `myWAC_Mobile_ERP.apk` pretože sa naň viažu ostatné balíčky. Po nainštalovaní je možné vytvoriť účet v systémových nástrojoch.