VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

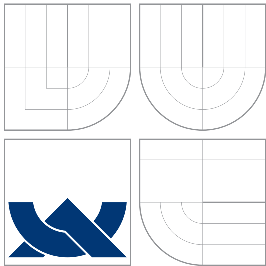
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SPRÁVA NÁKLADŮ NA PROVOZ AUTOMOBILU PRO ANDROID

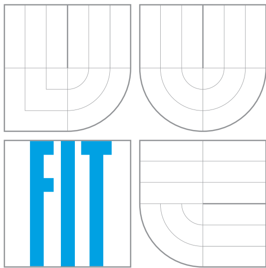
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE | Bc. VLADIMÍR KLEŠTINEC
AUTHOR

BRNO 2011



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

SPRÁVA NÁKLADŮ NA PROVOZ AUTOMOBILU PRO ANDROID

COSTS OF RUNNING A CAR ADMINISTRATION – AN ANDROID APPLICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. VLADIMÍR KLEŠTINEC
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK KUBÍČEK
SUPERVISOR

BRNO 2011

Abstrakt

Práce se zabývá vytvořením aplikace pro operační systém Android, která bude poskytovat efektivní správu nákladů na osobní automobil. Důležitou částí práce je komunikace a synchronizace aplikace se vzdáleným serverem. Celá aplikace si klade za cíl poskytovat přívětivé uživatelské rozhraní, které se bude efektivně ovládat.

Abstract

This work deals with creation of an application for an Android operating system, which will provide effective management of car costs. The important part of this work is communication and synchronization with a remote server. Whole application aims to provide friendly user inteface, which will be very effective to use.

Klíčová slova

Android, Java, síťová komunikace, REST, náklady na osobní automobil.

Keywords

Android, Java, network communication, REST, cost of running a car.

Citace

Vladimír Kleštinec: Správa nákladů na provoz automobilu pro Android, diplomová práce, Brno, FIT VUT v Brně, 2011

Správa nákladů na provoz automobilu pro Android

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Radka Kubíčka.

....
Vladimír Kleštinec
23. května 2011

Poděkování

Chtěl bych poděkovat vedoucímu této práce Ing. Radku Kubíčkovi za jeho odbornou pomoc při vytváření této práce. Dále pak panu Antonínu Šplíchalovi za vytvoření ikon pro tuto aplikaci.

© Vladimír Kleštinec, 2011.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **3**

2 Základní teorie a pojmy **5**
2.1 Přehled operačních systémů pro mobilní zařízení 5
2.1.1 Symbian OS 5
2.1.2 iOS OS 6
2.1.3 Windows Mobile 7
2.1.4 Android 9
2.1.5 Shrnutí 10
2.2 XML 10

3 SDK Android **12**
3.1 Vývojové prostředí a nástroje 13
3.2 Emulátor 15
3.3 Aktivity 16
3.4 Views 17
3.5 Aplikační manifest 18
3.6 Uvedení na trh – Android Market 18

4 Specifikace **20**

5 Návrh **21**
5.1 Funkčnost 21
5.2 Rozhraní aplikace 22
5.3 Databáze 23
5.3.1 Mobilní zařízení - SQLite 23
5.3.2 Server 27
5.4 Síťová komunikace 28
5.4.1 Vlastní protokol 29
5.4.2 WCF 30
5.4.3 Webové služby 30
5.4.4 Shrnutí 31
5.5 Rozhraní webové služby 32
5.5.1 Jednoduché operace 32
5.5.2 Hromadné operace 37
5.6 MVC 40
5.7 Architektura 41

6 Implementace 43
6.1 MVC . . . 43
6.2 Webová služba . . . 44
6.2.1 Běhové prostředí . . . 45
6.2.2 Konfigurace a autentizace . . . 45
6.2.3 Zabezpečení komunikace . . . 47
6.2.4 Implementační detaily . . . 47
6.3 Dotazy na webovou službu ze systému Android . . . 47
6.3.1 Nezabezpečené spojení . . . 47
6.3.2 Zabezpečené spojení . . . 49
6.4 Serializace objektů . . . 49
6.4.1 Webová služba . . . 50
6.4.2 Mobilní zařízení . . . 50
6.5 Změny konfigurace a zpracovávání úkonů na pozadí . . . 51
6.6 Prvky uživatelského rozhraní aplikace . . . 53
6.6.1 Action bar . . . 53
6.6.2 Quick Actions . . . 53
6.6.3 Vlastní prvek . . . 54
6.7 Grafy . . . 54
6.8 Práce se záznamy spotřeb . . . 56
6.9 Spuštění synchronizace . . . 56

7 Budoucí rozšíření 58
7.1 Webový portál . . . 58
7.2 C2DM . . . 58
7.3 Plánování údržeb . . . 59
7.4 Synchronizace obrázků . . . 59

8 Závěr 60

Kapitola 1

Úvod

V dnešním světě se stala mobilní zařízení cenově dostupná pro obrovskou část populace. Tento segment trhu zažívá v posledních letech obrovský rozvoj. Masový zájem o tato zařízení je způsoben zlevněním „chytrých“ mobilů a hlavně novými operačními systémy dodávanými s těmito přístroji. Z těžkopádného ovládání pomocí stylusu bylo přistoupeno k intuitivnímu ovládání pomocí prstů, díky absenci klávesnice mohou být displeje mnohem větší než u starších zařízení s klasickou klávesnicí. Na takto veliké zobrazovací ploše se již dá relativně pohodlně prohlížet internet, pročítat a zpracovávat dokumenty, případně zobrazovat grafy. Zařízení zkrátka dostávají úplně nové možnosti.

Symbolem dnešní doby je mobilita. Lidé chtějí mít všechny informace dostupné kdekoliv a okamžitě. S mobilitou úzce souvisí doprava. Lidé se chtějí přesunovat mezi různými místy, městy, státy a mezi těmito body se dopravují nejčastěji pomocí osobních automobilů. Správa nákladů na osobní automobil je tudíž velmi užitečná a mnohdy žádaná funkce. Je dobré mít přehled o celkových nákladech na svůj automobil, kolik bylo projeto za poslední měsíc kilometrů, jaká byla průměrná spotřeba minulý týden, kolik bylo celkem zaplaceno za oleje a nové pneumatiky. Všechny tyto informace by si uživatel musel poznamenávat a tvořit statistiky ručně, případně pomocí jiného software třetích stran.

Cílem této diplomové práce je vyvinout aplikaci, která bude umožňovat efektivní správu nákladů pro osobní automobil. Bude zajišťovat správu jak jednorázových výdajů, jako jsou opravy, výměny olejů, návštěvy autoservisů, tak pravidelných výdajů za tankování. Protože grafy jsou velmi přehlednou a uživatelsky přívětivou formou prezentace informací, vývoj nákladů bude zobrazitelný v grafové podobě. Data zadaná v aplikaci budou ukládána do lokální databáze, v případě změny telefonu bude možno tyto údaje a informace exportovat a znovu importovat do jiného zařízení. Pokud by uživateli ani toto nevyhovovalo, aplikace bude umožňovat synchronizaci se vzdáleným úložištěm. Uživatelé tohoto softwarového díla tak budou mít neustále přehled o ceně provozu jejich automobilu. Aplikace by měla být maximálně uživatelsky přivětivá a na první pohled jasně a jednoduše použitelná.

Hlavním problémem již existujících systémů je kromě složitého zadávání údajů také malá variabilita možností tankování. Tyto systémy umožňují pouze tankování do plné nádrže. Pokud nechce uživatel nabrat celou nádrž, například z důvodu nedostatku financí nebo drahého paliva, aplikace data buď vůbec nepřijmou nebo přestanou počítat statistiky, v horším případě započítají částečné natankování špatně. Tento problém bude vytvořená aplikace samozřejmě řešit.

Na začátku práce je čtenář seznámen se základním přehledem trhu a nejvýznamnějšími operačními systémy, s jejich stručnou historií a možnostmi vývoje pro danou platformu. Po této úvodní kapitole je rozebíráno SDK systému Android, přičemž jsou popsány důležité

nástroje a také emulátor. Následuje kapitola 4, kde je prezentována základní představa o funkčnosti, kterou by měla navržená aplikace poskytovat. V kapitole 5 je obsažen popis návrhu databáze jak na vzdáleném serveru, tak na mobilním zařízení, přičemž je zde také probrán architektonický návrh aplikace. V rámci kapitoly 6 jsou probrána zajímavá implementační specifika jak na straně serveru, tak na straně mobilního klienta. Jsou zde také zmíněny problémy, na které bylo při vývoji naraženo. Ve zbývající kapitole 7 jsou popsána navrhovaná rozšíření do budoucna.

Kapitola 2

Základní teorie a pojmy

2.1 Přehled operačních systémů pro mobilní zařízení

Jednotlivé mobilní operační systémy se vyvíjejí nezadržitelným tempem, mají rozdílné architektury, způsoby ovládání a není jednoduché s jejich vývojem udržet krok. Liší se také podporou poskytovaných vývojářských nástrojů, přičemž každý z nich podporuje jiné programovací jazyky a má jinak rozsáhlé a dostupné SDK[1]. Přehled těch nejznámějších a nejpoužívanějších systémů je uveden níže.

2.1.1 Symbian OS

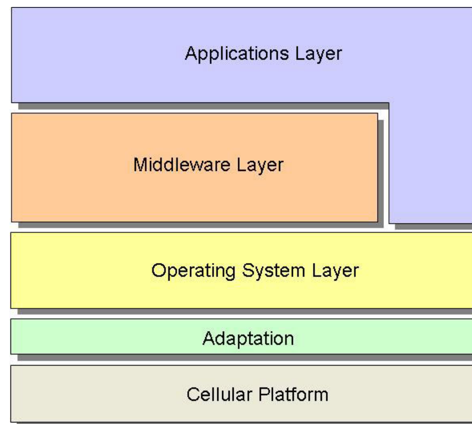
Tento velmi rozšířený operační systém vznikl v roce 1998 na základě spolupráce firem Ericsson, Nokia, Motorola a Psion. Symbian stále žije hlavně díky firmě Nokia, která je tradiční výrobce mobilních telefonů, avšak i přesto je víceméně na ústupu. Aktuální stabilní verze systému je 9.5 s grafickým uživatelským rozhraním S60 od firmy Nokia. Dříve bylo ještě k dispozici uživatelské rozhraní UIQ[2] od firmy Ericsson, ale jeho vývoj byl již utlumen. S60 v současné verzi 3 samozřejmě podporuje dotykové displeje. Mnoho společností dodává velmi užitečné knihovny pro Symbian, bohužel je řada z nich placená. Symbian podporuje velké množství programovacích jazyků, hlavními jsou Java, Python, Open C/C++ a Symbian C++. Open C++ je implementace standardního C++ známého z desktopu, oproti tomu Symbian C++ je lehce výkonnější a dovoluje využít celý potenciál zařízení. Všechny vývojářské nástroje jsou k dispozici zdarma. Standardní SDK je postaveno na IDE[3] Eclipse, instalace přidá automaticky plugin do vývojového prostředí. SDK je dostupné v různých verzích podle platformy, pro kterou je aplikace určena, samozřejmostí je dodávaný emulátor, ve kterém lze jednoduše otestovat aplikaci.

Základní architektura systému je složena z pěti vrstev a je zobrazena na obrázku 2.1. Následuje popis jednotlivých vrstev.

Application Services Layer je nejvyšší vrstva systému, skládá se z aplikačně specifického UI[4] a komponent jádra.

Generic Middleware se skládá z domén, které poskytují služby aplikační vrstvě. Například sítě, lokační služby a multimédia.

[1]Software Development Kit – soubor nástrojů zajišťujících podporu vývoje.
[2]User Interface Quartz – uživatelské rozhraní v Symbian OS.
[3]Integrated Development Environment – integrované vývojové prostředí.
[4]User Interface – uživatelské rozhraní.



Obrázek 2.1: Architektura Symbian OS [25].

OS Services Layer poskytuje vysokoúrovňové služby jádra jako například komunikaci, sítě, grafiku. Zahrnuje také nízkoúrovňové služby systému jako knihovny a utility, které poskytují k abstraktnímu hardwaru programovatelné rozhraní.

Adaptation integruje obecný platformový software s telefonní platformou. Je implementován tvůrci zařízení podle referenční implementace.

Cellular Platform implementuje pro každé zařízení specifický software, který vykonává služby potřebné pro běh Symbian platformy.

Více informací o možnostech tohoto operačního systému je možno nalézt v [26].

2.1.2 iOS OS

Tento mobilní operační systém vyvíjí firma Apple, jeho hlavní výhoda je způsob interakce s uživatelem, který byl velmi inovativní v době svého uvedení. Podporuje totiž již od první verze *multi-touch screen*. To znamená, že systém detekuje stisknutí několika bodů na dotykovém displeji současně. Například Microsoft podporuje tuto funkci až v nyní uváděných Windows Phone 7. Vývojářské nástroje s názvem Xcode tools jsou volně ke stažení, ovšem pouze pro Mac OS X, vyvíjet na jiné platformě není možné. Xcode tools obsahují emulátor a všechny nástroje potřebné pro základní editaci, kompilaci a testování kódu. Aplikace mohou být psány nativně v jazyce C na nejnižších úrovních OS nebo v jazyku Objective-C na vyšších úrovních.

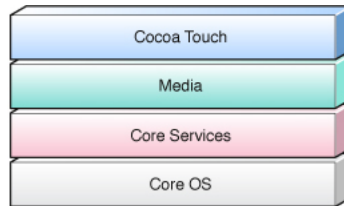
Architektura systému má čtyři základní vrstvy, které jsou znázorněny na obrázku 2.2. Pro bližší informace o systému, kompletní referenci a SDK je možno se podívat do literatury [2].

Nyní následuje bližší popis jednotlivých vrstev.

Core OS je na nejnižším stupni této architektury. Je zde jádro operačního systému, síťová infrastruktura a správa souborového systému.

Core Services zahrnují frameworky poskytující služby pro manipulaci s řetězci, správu kolekcí, URL utility, správu uživatelských kontaktů a nastavení.

Na vyšších vrstvách se mixují technologie založené na C a ObjC.



Obrázek 2.2: Architektura iOS [1].

Media je vrstva, jejíž služby a frameworky jsou závislé na předchozí vrstvě *Core services*, poskytují grafické a multimediální služby *Cocoa Touch* vrstvě. Tyto služby umožňují 2D a 3D kreslení, *Core Audio*, video přehrávání, *OpenGL ES*.

Cocoa Touch vrstva přímo podporuje aplikace pro iOS. Nejdůležitější frameworky jsou:

- *UIKit framework*, poskytující objekty, které aplikace zobrazuje v rámci uživatelského rozhraní.
- *Foundation framework* definuje základní chování objektů, zavádí mechanismy pro jejich správu a poskytuje primitivní datové typy, kolekce a služby operačního systému.

2.1.3 Windows Mobile

Vývoj tohoto uzavřeného operačního systému sahá do roku 2000, kdy byl vydán Pocket PC 2000. Windows Mobile jsou založeny na Windows CE (Windows Embedded Compact), která jsou určena do vestavěných (embedded) zařízení. Windows Mobile je momentálně ve verzi 6.5, která oproti verzi 6 nepřináší žádné výrazné změny. V současnosti se začíná objevovat nejnovější verze Windows Phone 7, která je od základů kompletně inovovaná a neposkytuje zpětnou kompatibilitu. Vývojářské nástroje jsou k dispozici zdarma, avšak pouze pro platformu Microsoft Windows. Zaintegrují se do standardní instalace Visual Studia (minimální verze 2005 SP1, maximálně 2008), vývojář pak může aplikace vyvíjet stejně, jak je zvyklý pro desktop. Součástí SDK je samozřejmě plnohodnotný emulátor. Nástroje jsou v různých verzích, v závislosti na cílové platformě. Mimo současné SDK je také balíček takzvaných *Remote Tools*, které jsou neocenitelnou pomůckou pro nahlížení do útrob, ať už reálného zařízení nebo emulátoru. Je zde Windows CE Remote File Viewer, který dovolí procházet soubory obsažené v zařízení, dále Windows CE Remote Process Viewer, díky kterému máme možnost sledovat běžící procesy a ukončovat je. Poslední je Windows CE Remote Registry Editor, poskytující plnou kontrolu nad registry připojeného zařízení.

V současnosti jsou na trhu tři produktové řady:

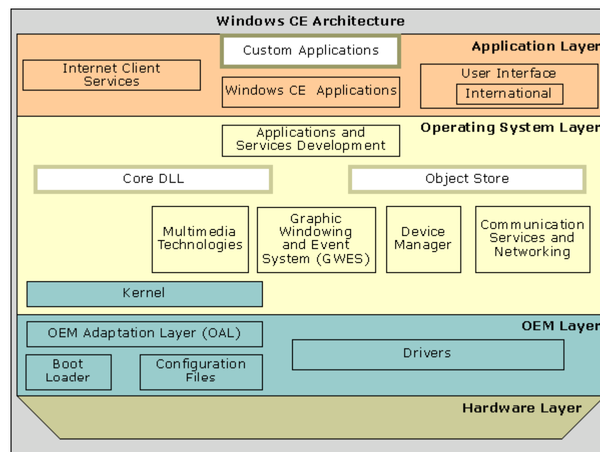
Classic – určena pro zařízení bez telefonního modulu.

Standard – podporuje zařízení s telefonním modulem, ale bez dotykového displeje.

Professional – určena pro nejvýkonnější zařízení s dotykovým displejem a telefonním modulem.

API je uspořádáno do sekcí podle účelu použití. Jsou jimi například: *Application and Services*, kde můžeme najít nástroje umožňující jednoduchou synchronizaci s počítačem pomocí *ActiveSync*. Dále je tu *Pocket Outlook Object Model (POOM)* poskytující přístup ke kontaktům, událostem a schůzkám v kalendáři. V sekci Audio se nachází jednoduché API *Wave Form* umožňující práci se zvukem. Sekce *Phone Features*, která zpřístupňuje správu SMS zpráv a volání. Obsahuje *SIM management*, což je API poskytující přístup k SIM kartě, dále také *Radio Interface Layer (RIL)*, který umožňuje získat informace z vysílačů BTS. Pro správu napájení je zde API *Power Management*. V části *Security* se nacházejí knihovny pro efektivní správu certifikátů, hashování a autentizaci [10].

Celá architektura Windows CE je na obrázku 2.3.



Obrázek 2.3: Architektura Windows CE [9].

Aplikace je možno vyvíjet nativně v jazyce C++, za podpory knihoven MFC a ATL. Tyto knihovny jsou pro programátory známé z desktopových systémů. Dále Windows Mobile umožňují vývoj nenativních aplikací v jazyce C# na platformě .NET Compact Framework. Jedná se o podmnožinu klasického .NET Frameworku známého z desktopových počítačů. Tento framework je ve verzi 2.0 (nejnovější verze 3.5) dostupný standardně ve všech zařízeních s operačním systémem Windows Mobile 6. Vývoj aplikací na platformě .NET je rychlý a efektivní. Každé WM 6 zařízení obsahuje v ROM kromě .NET Compact Frameworku 2.0 SP2 také SQL Server 2005 Compact Edition. Tento SQL server je optimalizován pro přenosná zařízení a umožňuje jednoduchou synchronizaci s klasickými MS SQL servery.

Windows Mobile se postupně stává okrajovým systémem. Jeho tržní podíl upadá a jeho význam do budoucna zůstane hlavně ve specializovaných zařízeních firem Motorola a Symbol. V segmentu mobilních telefonů je nahrazován systémem Windows Phone 7, kdy se začínají pomalu objevovat zařízení s tímto novým OS.

Kompletní reference včetně podrobnějších informací je volně dostupná na webových stránkách [9]. Výborně popsané vývojové prostředí a způsoby programování v .NET Compact Frameworku lze nalézt v knize od Luboslava Lacka [7].

2.1.4 Android

Samotný název Android je souhrnné označení pro operační systém a soubor klíčových aplikací pro tuto platformu vyvíjených pod záštitou Open Handset Alliance, jejímž hlavním členem je společnost Google. Tento OS byl uveden na trh teprve v roce 2007, od té doby však jeho podíl raketově stoupá. Aplikace je možno vyvíjet nativně v jazyku C++, doporučováno je nicméně psát aplikace v jazyce Java, pro který společnost Google vytvořila vlastní knihovny. Nejedná se ovšem o plnou implementaci jazyka Java, protože společnost Google neimplementovala všechny standardní knihovny, za což je kritizována. Oficiálně podporované vývojové prostředí je Eclipse (3.3 a novější), není ovšem problém vyvíjet i v jiných editorech. Avšak použití jiných vývojových prostředí není nejvhodnější, protože bychom přišli o ADT plugin, který do Eclipse dodává velmi dobrou podporu pro vývojáře. Více informací je možno nalézt v [14] a v dalších kapitolách této práce. Základní úvod do architektury je uveden v [12].

Základní architektura sytému je zobrazena na obrázku 2.4. Na tomto obrázku je vidět grafické znázornění do pěti logických úrovní. Ty budou nyní podrobněji popsány.

Obrázek 2.4: Architektura systému Android, obrázek byl převzat z [12].

Applications – základní aplikace dodávané společně s OS. Jedná se například o emailový klient, kalendář, program na zasílání SMS a internetový prohlížeč. Všechny tyto aplikace jsou napsány v jazyku Java.

Application framework – poskytuje základní služby systému:

- *Content provider* dovoluje aplikacím jednoduše sdílet data (jako například kontakty) mezi sebou.

- *Resource manager* zajišťuje přístup k neprogramovým zdrojům jako je rozložení displeje a lokalizované řetězce.
- *Notification manager* umožňuje všem aplikacím zobrazit vlastní upozornění ve status baru.
- *Activity manager* spravuje životní cyklus aplikací.
- *Telephony manager* poskytuje přístup o telefonních službách na zařízení. Aplikace mohou použít metody pro zjištění telefonních služeb a jejich stavu.

Libraries – množina C/C++ knihoven používaných různými komponentami systému. Přistupuje se k nim pomocí aplikačního frameworku. Jedná se například o SQLite knihovny sloužící pro efektivní uložení dat databáze nebo LibWebCore poskytující výkonné jádro internetového prohlížeče.

Android Runtime – Dalvik Virtual Machine (DVM) a knihovny jádra operačního systému. Jedná se o implementaci virtuálního stroje jazyka Java. Každá aplikace má svůj vlastní proces se svým vlastním DVM, zařízení může mít spuštěno několik DVM současně. DVM spouští soubory v Dalvik Executable (*.dex) formátu; tento formát byl navržený s ohledem na minimální spotřebu paměti.

Linux kernel – Android je postaven na Linuxovém jádru verze 2.6. Toto jádro vytváří pro zbytek systému abstraktní vrstvu nad hardwarem telefonu.

2.1.5 Shrnutí

Operačních systémů do mobilních zařízení existuje velké množství. Liší se od sebe jak podporou programovacích jazyků, kvalitou dokumentace, velikostí uživatelské základny, tak i obtížností a stylem programování. Každý operační systém má svá pro a proti.

Systém iOS vyniká uživatelsky přívětivým rozhraním, na druhou stranu však programátora nutí pracovat v nepříliš rozšířeném Objective C. Windows Mobile je pomalu umírající platforma, která má však stále co nabídnout firemní klientele z důvodu bezchybné synchronizace s Exchange servery a také díky dědictví obrovského množství v minulosti vytvořených programů a knihoven. Operační systém Symbian je také na velkém počtu zařízení a jeho tržní podíl je v řádu desítek procent. Tento podíl se však pomalu zmenšuje, z čehož těží hlavně nové operační systémy v čele s Android. Tento nový systém má obrovský tržní potenciál a velmi příjemně se pro něj programuje. Vývojáři znalí programovacího jazyka Java budou mít přechod na tuto platformu značně ulehčen.

2.2 XML

Aplikace dokáží v dnešní době pracovat s elektronickými daty, představujícími například hudbu a obrázky. Stále je zde však značná potřeba ukládání dokumentů založených na textových datech. Skvělou volbou pro uložení textových dat je jazyk XML (Extensible Markup Language). Byl vyvinut konsorciem W3C na základě zkušeností s předchozími formáty dat. Je to volný a zdarma dostupný standard, proto pro něj existuje řada volně dostupných parserů a nástrojů podporující práci s ním v nejrůznějších jazycích a editorech. Dovoluje šíření informací mezi různými architekturami počítačů a mezi různými aplikacemi, bez nutnosti několikanásobné konverze. Dokument XML obsahuje specifické elementy, entity a značky (tagy). Jednotlivé elementy se mohou vnořovat. Jazyk XML nám umožňuje definovat obsah dokumentu odděleně od jeho formátování.

XML je značkovací jazyk, který je podmnožinou jazyka SGML. SGML (Standard Generalized Markup Language) je ještě obecnější než XML. XML je ve skutečnosti meta-jazyk, čili jazyk, který se používá k popisu jiných jazyků. Nemá žádný pevně daný seznam jmen elementů. Poskytuje uživateli naprostou volnost při vytváření nových elementů. Doporučuje se samozřejmě používání jmen, která mají pro danou aplikaci smysl. Přílišná volnost ovšem může být i na škodu. Abychom se v záplavě značek neztratili a dodrželi korektnost dokumentu, je zde jazyk DTD (Document Type Definition). DTD definuje povolené prvky. DTD definici je poté možno použít pro validující parser. Tento softwarový analyzátor pak kontroluje validitu XML dokumentu vůči DTD definici. Kromě DTD jsou i novější technologie pro definici struktury dokumentu. Je jím například XML Schema. XML Schema je nástupcem DTD a má mnohem větší vyjadřovací schopnosti při popisování XML dokumentů [21].

Jazyk XML má v dnešní době velmi rozsáhlé využití ve značně různorodých polích působnosti. V systému Android je XML využíváno ve velké míře, je jím popisován jak vzhled jednotlivých grafických prvků, tak i celých aktivit. Je používán pro ukládání všech hodnot – například barev, velikostí a v neposlední řadě i jazykových lokalizací.

Kapitola 3

SDK Android

Programy lze vyvíjet v libovolném editoru, doporučené je ovšem používat vývojové prostředí Eclipse. Jako vhodnější se jeví použít starší verzi Galileo (3.5), případně Ganimede (3.4). Tato verze je o dost svižnější a netrpí nepříjemnou ztrátou odezvy při použití automatického doplňování kódu (vývojáři zvyklí programovat pod Windows jej jistě znají pod názvem IntelliSense). IDE je dále rozebráno v sekci 3.1.

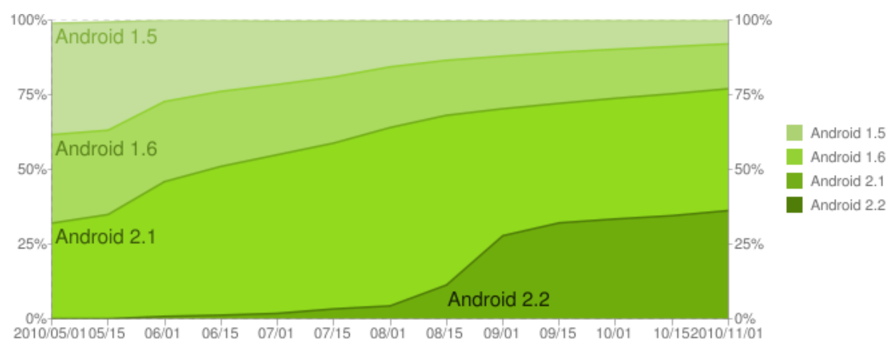
SDK je k dispozici pro 3 platformy. Jsou jimi

Windows – XP a vyšší, jsou podporovány i 64bitové verze.

Linux – je doporučeno vyvíjet pod 32bitovou distribucí.

Mac OS X – verze 10.5.8 a novější (pouze architektura x86).

SDK je v několika stupních, některé jsou již zastaralé, jiné se stále používají. Toto je jistá nevýhoda systému Android. Systém je v relativně hodně verzích a pokud je cílem pokrýt co nejširší skupinu uživatelů, musí se použít nízká verze SDK. Jednotlivé verze jsou samozřejmě zpětně kompatibilní, takže program fungující na verzi 1.5 funguje i na vyšších verzích. Na obrázku 3.1 je vidět, jak se mění tržní podíl jednotlivých verzí. Při vývoji aplikace se zadává takzvaná minSDK, což je minimální verze, která je vyžadována pro bezproblémový chod programu. Stručný přehled jednotlivých aktuálně používaných verzí je uveden níže.



Obrázek 3.1: Zastoupení jednotlivých verzí [13].

< **1.5** – Zařízení s verzí menší než 1.5 se již skoro nevyskytují a nové aplikace už moc často tuto verzi nepodporují.

1.5 – Jak je vidět z grafu, podíl těchto zařízení rychle klesá. Přesto stojí za úvahu jejich podpora. Oproti předchozím přidává mnoho nových prvků pro tvorbu GUI. Mezi nimi například *AppWidget framework* pro tvorbu *widgets* umístěných na hlavní obrazovce. Zavádí také aplikačně definované hardwarové požadavky. Je tak možné definovat například nutnost přítomnosti trackballu pro správnou funkčnost aplikace. Je zde také *Input method framework* s funkcí automatické predikce textu.

1.6 – Tato zařízení se již vyplatí podporovat. Oproti předchozí verzi přibyla podpora VPN sítí, dále byl přidán framework pro podporu uživatelských gest. Linuxové jádro systému bylo změněno na 2.6.29 a byl přidán vyhledávací box, který poskytuje rychlý, efektivní a konzistentní způsob hledání napříč mnoha zdroji.

2.1 – Zde je zajímavá podpora Bluetooth 2.1 a uživatele jistě potěší podpora Exchange a webový prohlížeč podporující HTML5. Pomocí multitouch technologie byla zlepšena možnost rychlého psaní dvěma prsty. Tato verze je v současné době velmi rozšířená, její tržní podíl je skoro 40%.

2.2 – Mezi novinky patří například podpora push notifikace přímo v zařízení pomocí *Cloud To Device Messaging (C2DM)*, přenosný wifi hotspot umožňující ostatním zařízením připojit se. Push notifikace je velká přednost této verze. Jakmile chceme dát vědět své aplikaci o nějaké změně z okolního světa (například přišel e-mail), využijeme push notifikace. V nižších verzích systému je nutné je nahrazovat službami třetích stran.

2.3 – Zatím nejnovější verze. Do systému Android byla přidána podpora SIP telefonie. Vývojářům tak nic nebrání začít ve vývoji aplikací pro internetovou telefonii. Systém dále dostal podporu více kamer, byla přidána podpora pro nové senzory, jako například barometr a lineární akcelerometr. Je zde nově také podpora pro „*extra large screens*“, tím se myslí hlavně tablety.

Bližší informace o SDK lze nalézt v [15]. Velmi užitečnou částí SDK je Emulátor, který je zdarma dostupný. Ten je podrobněji rozebrán v sekci 3.2.

Základem pro každou aplikaci v systému Android jsou aktivity a pohledy. Vzájemně se doplňují a umožňují tvorbu GUI. Tvorba uživatelského rozhraní je zde založena na XML popisu rozložení prvků. Tento postup se trochu odlišuje od tradičního konstruování GUI v kódu, ale je velmi intuitivní a vhodný pro popis rozložení prvků. Programátor není nicméně do ničeho nucen a má možnost vytvářet uživatelské rozhraní i tradičním způsobem za pomocí objektů přímo ve zdrojovém kódu.

V rámci systému je dostupné velké množství knihoven. Pokud bychom ale nenašli knihovnu, kterou hledáme, není problém použít libovolnou externí knihovnu. Zde je však třeba pečlivě vybírat, protože knihoven dělajících konkrétní úkol může být více, ovšem jen málo z nich je opravdu kvalitních.

Aplikace bude vyvíjena pro Android verze 2.1 a vyšší, vývoj pod touto verzí systému byl zvolen, protože protože nejmarkantnější podíl zařízení na trhu je verze 2.1 a vyšší.

3.1 Vývojové prostředí a nástroje

Jak již bylo řečeno, hlavní vývojové prostředí pro vývoj aplikací pod OS Android je Eclipse. Toto vývojové prostředí vzniklo v roce 2001 ve firmě IBM a v lednu roku 2004 byla založena nezisková organizace Eclipse Foundation. Tato organizace se nyní stará o vývoj

tohoto prostředí. Toto vývojové prostředí není určeno pouze pro jeden programovací jazyk, existují rozšíření pro celou řadu jiných. Toto IDE je možné rozšířit celou škálou pluginů poskytujících nejrůznější funkčnost. Od rozšíření podpory pro správy verzí, přes refaktorizace až po specializovaná rozšíření zajišťující kontrolu správnosti kódu.

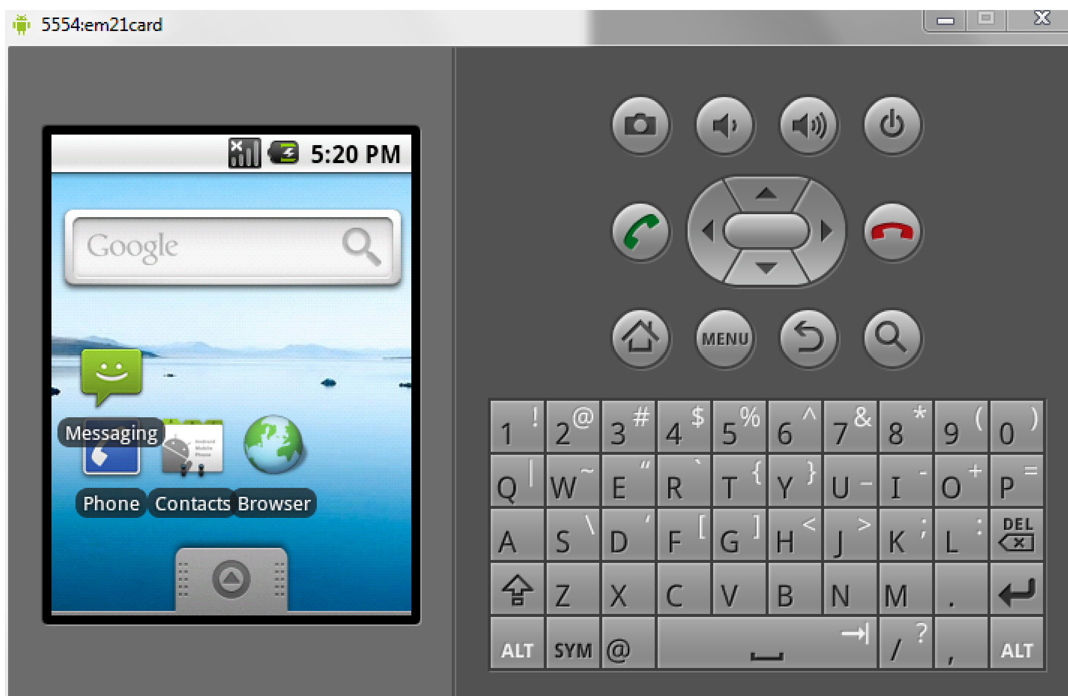
Mezi jedno takovéto rozšíření patří ADT (Android Development Tools [11]). V rámci SDK je řada nástrojů, které dokáží velice zpříjemnit vývoj aplikace. Jejich soupis je v následující tabulce 3.1.

Tabulka 3.1: Nástroje z SDK Android [16].

Nástroj	**Popis**
ADT plugin	Plugin do IDE Eclipse umožňující jednoduchý překlad, testování a ladění aplikací.
Android Emulator	Plnohodnotný emulátor.
Hierarchy Viewer	Poskytuje vizuální reprezentaci hierarchie layoutů.
layoutopt	Analyzuje efektivitu layoutu.
Draw 9-patch	Podpora pro tvorbu obrázků. Tyto obrázky jsou používány například pro tlačítka. Mají nadefinováno, jak se roztahovat, a díky tomu vypadají dobře i při různých rozlišeních.
Dalvik Debug Monitor Service	Integruje se do Eclipse, umožňuje nahlížet přímo do útrob zařízení, prohlížet souborový systém a spravovat procesy.
Traceview	Provádí grafickou analýzu logů, které může program za běhu vytvářet.
mksdcard	Pomáhá vytvořit obraz disku, který můžeme připojit k emulátoru pro simulaci SD karty.
sqlite3	Umožňuje přistupovat k databázovým souborům vytvořených na zařízení. Dovoluje spouštět nad těmito soubory SQL příkazy.
monkey	Zasílá na vstup programu pseudonáhodné posloupnosti uživatelských vstupů a provádí tak stress aplikace. Tato data pak může ukládat pro další analýzu. Umožňuje tak odhalit chyby na vstupu programu, s kterými programátor nepočítal.
monkeyrunner	Python skripty zadané do tohoto programu slouží k testování aplikací. Program spouští příkazy v závislosti na dodaném skriptu, může také pořizovat snímky emulátoru. Primárně je určen pro unit testy a testy UI. Umožňuje také například automatický překlad a instalaci balíků do zařízení.
zipalign	Optimalizační nástroj, díky kterému budou dekompresovaná data zarovnána vzhledem k začátku souboru. To ulehčí práci runtime OS Android a programy poběží rychleji.

3.2 Emulátor

Velmi užitečnou součástí SDK je emulátor založený na QEMU, což je open source emulátor a virtualizér. Je na něm samozřejmě možno spouštět a testovat téměř jakékoliv programy pro Android. Při spuštění umožňuje vybrat, jaké AVD[1] chceme emulovat. Umožňuje emulaci SMS, GPS i volání, bohužel však nepodporuje fotografický vstup, což může být pro jisté aplikace značný nedostatek. Emulátor je zobrazen na obrázku 3.2.



Obrázek 3.2: Emulátor systému Android verze 2.1.

Emulátor má velmi dobrou podporu pro síťovou komunikaci. Pro vývoj aplikací využívajících více zařízení je možno propojit jednotlivé instance emulátoru mezi sebou a ty pak mohou vzájemně komunikovat. Není ani problém přistupovat z emulátoru na internet a testovat tak už přímo služby nasazené na jiných strojích. Jak již bylo zmíněno, emulátor má i některá omezení. Mezi hlavní patří:

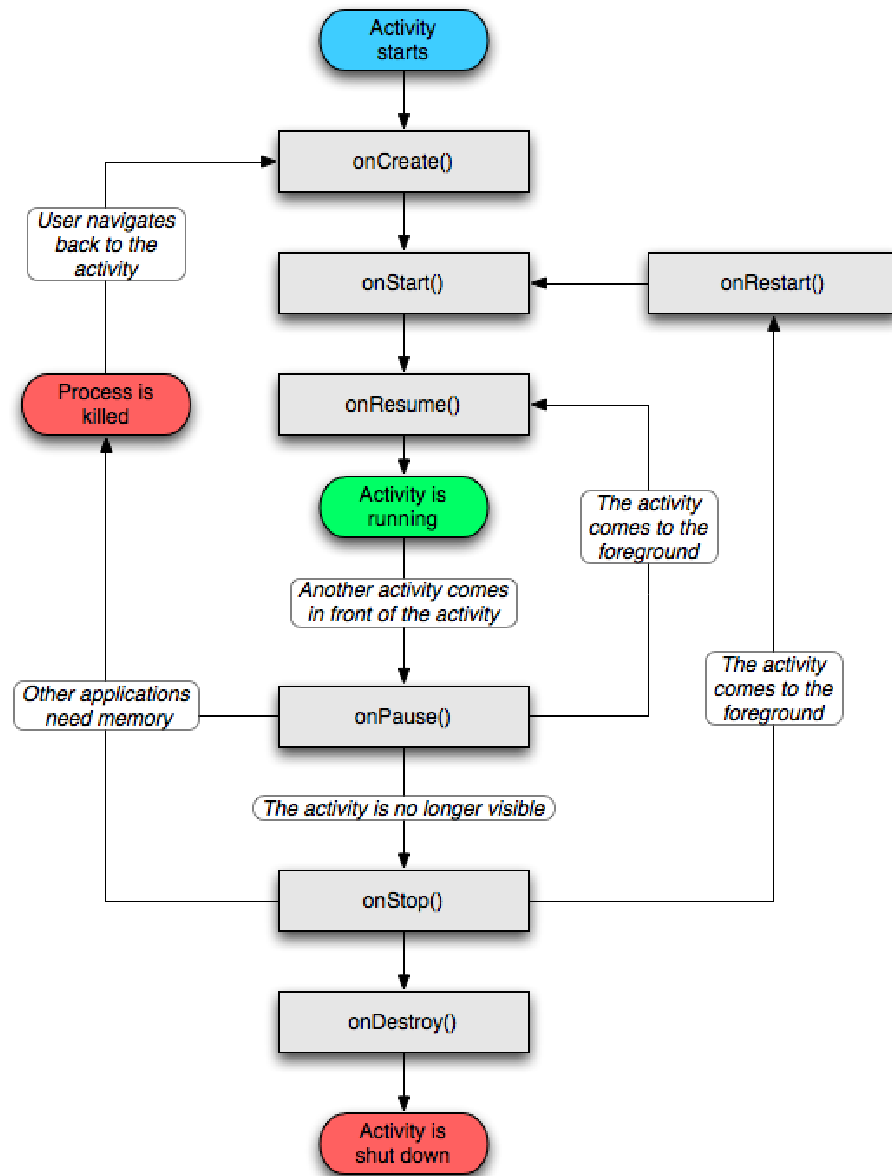
- Chybějící podpora pro USB připojení.
- Chybějící podpora pro nahrávání videa a pořizování fotografií.
- Chybějící podpora Bluetooth.
- Chybějící podpora pro zjištění stavu připojení.

I přes tyto nedostatky se jedná o výborný emulátor. Jedinou jeho citelnou nevýhodou je značná pomalost při ladění aplikací. Na slabších počítačových sestavách je ladění pomocí emulátoru značně pomalé.

[1] Android Virtual Device.

3.3 Aktivity

Aktivita je základním stavebním kamenem pro vývoj aplikací na Google Android. Prakticky všechny aktivity interagují s uživatelem a vyžadují od něj uživatelský vstup, případně mu sdělují nějaký druh informace. Aktivita má na starosti vytvoření okna, do kterého se vkládá uživatelské rozhraní. Abychom mohli aktivitu spustit pomocí volání `Context.startActivity`, musí být deklarována v souboru `AndroidManifest.xml`, který je popsán blíže v části 3.5. Obrázek 3.3 zobrazuje životní cyklus aktivity.



Obrázek 3.3: Životní cyklus aktivity [15].

Aktivity v systému jsou spravovány pomocí zásobníku. Každá nově spuštěná aktivita je umístěna na vrchol zásobníku a stává se z ní běžící aktivita. Pokud byla předtím spuštěna

jiná aktivita, ta zůstává umístěna v zásobníku a dostane se opět na popředí až s odvoláním současné aktivity.

Celý životní cyklus aktivity probíhá mezi voláními metod `OnCreate` a `OnDestroy`. Všechna nastavení globálního stavu aplikace se provádí v metodě `OnCreate`. V metodě `OnDestroy` se uvolňují všechny alokované zdroje dané aktivity, jako například ukončení všech otevřených spojení nebo zavření používaných souborů.

Aktivita je viditelná mezi metodami `OnStart` a `OnStop`. Mezi těmito dvěma voláními je viditelná – neznamená to však, že je na popředí. Může být překryta aktivitou, která není zobrazena přes celý displej, nebo poloprůhlednou aktivitou. Tyto metody mohou být v průběhu života aktivity volány vícekrát, jak je aktivita zobrazována a schovávána.

Mezi voláními metod `OnResume` a `OnPause` je aktivita plně aktivní a zobrazena na popředí, je tedy viditelná a není překryta žádnými jinými aktivitami. V tomto stavu aktivita komunikuje s uživatelem. Přechod mezi těmito dvěma stavy je velmi častý.

Pokud se změní konfigurace aktivity (změna orientace displeje, jazyka, ...), je ukončena zavoláním metody `OnDestroy` a následně znovu vytvořena. Toto se děje, protože aplikační zdroje, stejně tak vzhledy, se mohou lišit v závislosti na konfiguraci.

Jelikož celý životní cyklus aktivity řídí systém, nepřichází programátor vůbec do styku s instancí dané aktivity. Pokud chceme do aktivity předat informace, musíme tak udělat s využitím třídy `Intent`, pomocí které je možno předat jakýkoliv parametr jednoduchého datového typu, případně datový typ, který je serializovatelný.

Bližší informace ohledně aktivit je možno nalézt v dokumentaci [17], popřípadě v [8].

3.4 Views

View je základním stavebním kamenem uživatelského rozhraní, který zaobírá čtvercový prostor displeje a je odpovědný za vykreslování a obsluhu událostí.

Je bázovou třídou pro všechny nejběžnější prvky uživatelského rozhraní jako jsou tlačítka, textová pole a nápisy. Jednotlivým instancím tříd odvozených od *View* se typicky nastavují vlastnosti jako například:

- Obrázek pro *ImageView* nebo textová hodnota pro *TextView*.
- Nastavení metody pro obsluhu patřičných událostí, ať už kliknutí na tlačítko nebo například změna textu v textovém poli.
- Nastavení viditelnosti *View*. Ten může být viditelný, neviditelný nebo vůbec nezobrazený a to tak, že není ani počítáno s jeho pozicí v layoutu (oproti neviditelnému).

S třídou *View* přímo souvisí třída *ViewGroup*, která je bázovou třídou pro všechny layouty, což jsou neviditelné kontejnery obsahující jiné *View* nebo *ViewGroup*. *ViewGroup* definují jejich vzájemné pozice a rozložení *View* a jiných *ViewGroup* na displeji.

Přidání *View* do jednotlivých *ViewGroup* může být provedeno buď pomocí XML, nebo přímo v kódu s využitím objektů.

Objekty *View* a *ViewGroup* jsou úzce provázány s aktivitami. Aktivity mají na starosti vytvoření okna a vložení uživatelského rozhraní. Toto vložení probíhá v případě, že je *ViewGroup* popsán pomocí XML souboru voláním metody `setContentView` instance třídy aktivity, která vloží chtěné uživatelské rozhraní do dané aktivity.

3.5 Aplikační manifest

Aplikační manifest je XML soubor, který musí být obsažen v každém projektu pro systém Android. Jestliže chce systém spustit jakoukoliv aplikační komponentu, musí nejdříve vědět, že tato komponenta existuje. Tato informace je obsažena právě v aplikačním manifestu. Tento soubor (`AndroidManifest.xml`) je umístěn v kořenovém adresáři aplikace a musí v něm být zmíněny všechny používané komponenty.

V aplikačním manifestu je definováno relativně velké množství položek. Mezi hlavními lze uvést:

- Aplikační komponenty.
- Minimální verze požadovaného API pro běh aplikace.
- Oprávnění, která jsou potřebná pro správný běh aplikace, může jít například o přístup k internetu nebo zápis do externího úložiště.
- Požadované programové a hardwarové vlastnosti daného zařízení, jako například fotoaparát či modul Bluetooth.
- Je zde specifikováno jméno balíčku s aplikací a toto jméno slouží jako jedinečný identifikátor aplikace.
- Obsahuje seznam knihoven, proti kterým musí být aplikace sestavena.
- Určuje, který proces bude hostit naši aplikaci.

Mezi aplikační komponenty patří aktivity (*activities*), služby (*services*), *broadcast receivers* a *content providers*.

Aktivity jsou hlavními stavebními kameny každé aplikace komunikující s uživatelem, oproti tomu služby běží na pozadí a nemají obvykle žádné uživatelské rozhraní. Registrací *broadcast receivers* se aplikace zapíše k odebírání zpráv vysílaných například systémem nebo jinými aplikacemi. Aplikace takto dostává informace například o startu systému nebo o aktivaci nastaveného alarmu. *Content providers*, neboli poskytovatelé obsahu, poskytují v rámci systému Android data aplikacím. Jednotlivé aplikace takto mohou snadno sdílet například uživatelovy kontakty.

3.6 Uvedení na trh – Android Market

Na systému Google Android jsou poměrně přímočaré způsoby distribuce softwaru. Aplikace je sice možno instalovat přímo z lokální instalace (na rozdíl od iOS), toto řešení se však hodí buď pro testovací účely, nebo pro nasazení aplikací pro omezené spektrum uživatelů.

Hlavním distribučním kanálem je Android Market, který je obsažen jako předinstalovaná součást programového vybavení systému. Zde si mohou uživatelé stáhnout a nainstalovat aplikaci, která může být jak zdarma, tak placená. Veškerou distribuci a vybírání poplatků tak vývojář nechává na bedrech společnosti Google a může se tak soustředit na svůj projekt. Uživatelé navíc mají důvěru v placené aplikace, protože samotná platba je velmi jednoduchá a stejná pro všechny aplikace. Nemusejí se tak vypořádat s různými distributory a odlišnými způsoby plateb.

Pokud aplikace využívá mapové podklady, musí být finální verze aplikace zaregistrována pod jedinečným APID.

Aplikace jsou publikovány v instalačních `*.apk` souborech. Tyto soubory musí být podepsány privátním klíčem před umístěním na Android Market. Privátní klíč nemusí být podepsán žádnou certifikační autoritou a je absolutně dostačující použít certifikát vydaný námi samotnými. Systém testuje datum expirace certifikátu pouze při instalaci a později na toto datum není brán zřetel. Pro podepsání stačí standardní utility `Keytool` a `Jarsigner`.

Jakmile je aplikace podepsána, použije se pro optimalizaci velikosti podepsané aplikace utilita `zipalign`.

Požadavky vyžadované Android market serverem:

- Privátní klíč, kterým je podepsána aplikace, musí být validní minimálně do 22. října 2033.
- Aplikace musí definovat jak `android:versionCode`, tak `android:versionName` v manifestu. Hodnota atributu `android:versionName` je zobrazována uživatelům při aktualizacích, oproti tomu hodnota `android:versionCode` slouží pro interní záznamy Android marketu.
- Atributy `android:label` a `android:icon` musí být také definovány, aby byla aplikace správně nabídnuta uživatelům.

Distribuci aktualizací programů obstarává Android Market automaticky. Jakmile nahrajeme na server novou verzi aplikace, jsou uživatelé automaticky upozorněni o dostupnosti nové verze.

Licencování je taktéž zajišťováno externě a vývojář má možnost nastavit, jaká licenční politika mu nejlépe vyhovuje. Aplikace si může za běhu ověřovat stav licence na danou aplikaci pro specifického uživatele.

Kapitola 4

Specifikace

Cílovou platformou této práce je operační systém Android, přičemž implementačním jazykem je Java. Účelem vyvíjené aplikace je poskytovat efektivní správu nákladů na osobní automobil. Zároveň si klade za cíl mít jednoduché a pro uživatele přívětivé uživatelské rozhraní. Pro každý automobil umožní aplikace vytvořit a spravovat samostatný profil obsahující základní informace o vozidle. Jednotlivé náklady mohou být různých druhů, přičemž aplikace bude spravovat tyto základní typy:

- Náklady na spotřebu (pohonné hmoty).
- Náklady na servis.
- Náklady pravidelného charakteru.

Všechny v současné době existující aplikace, které se zabývají správou nákladů pro osobní automobily, mají jednu společnou nevýhodu. Pokud natankujeme plnou nádrž, aplikace počítá průměrnou spotřebu bez problémů. V případech, kdy netankujeme plnou nádrž, tyto aplikace přestanou počítat průměrnou spotřebu, případně ji začnou počítat chybně. Jedním z problémů, které tato aplikace řeší, je samozřejmě i toto chování. Vyvíjená aplikace proto bude implementovat tyto druhy tankování:

Do plné – označuje natankovaní pohonných hmot do plné nádrže.

Částečné – označuje natankování pouze určité části nádrže. Při tomto typu tankování se aplikace musí vypořádat se správnými výpočty průměrné spotřeby.

První – označuje první natankování.

Chybné – označuje natankování, jehož hodnoty nesouhlasí s ostatními zadanými hodnotami, například vychází záporná průměrná spotřeba v důsledku chybně zadaných množství tankovaných pohonných hmot.

Všechny údaje budou po zadání plně editovatelné, v případě změny údajů se patřičně přepočítají a promítnou do všech příslušných statistik. Patřičné údaje aplikace umožní prezentovat vizuálně pomocí grafů.

Tyto údaje a profily uživatelů i automobilů se mohou na přání uživatele synchronizovat se vzdáleným serverem. Je potřeba zajistit, aby tento vzdálený server počítal s budoucím rozšiřováním aplikace. Údaje se tedy nebudou na server pouze ukládat, ale případně i stahovat dle potřeb. Toto nastane například v případě, kdy uživatel zadá data na jiném zařízení.

Kapitola 5

Návrh

Tato kapitola obsahuje podrobný návrh aplikace, přičemž v jednotlivých částech jsou popsány jeho důležité body. Správný návrh je při tvorbě projektu velmi důležitý krok, protože s případnými chybami v návrhu a rozhodnutími, která zde provedeme, se budeme potýkat po celou dobu vývoje projektu. Správný návrh je klíčový pro úspěšný vývoj celé aplikace.

V sekci 5.1 je rozebrána hlavní funkčnost námi navrhovaného programu. Je zde rozebráno, co by program měl umět a jak by se měl chovat. Sekce 5.2 popisuje návrh základního rozhraní aplikace. Samotné rozhraní se značně liší od desktopových aplikací, protože oproti nim máme velice omezenou plochu displeje. Navíc musí být jednotlivé ovládací prvky dostatečně velké, vzhledem k tomu, že zařízení jsou ovládána většinou pouze prsty. Návrh databáze a zvolení vhodného databázového systému je poté popsáno v sekci 5.3. Následuje sekce 5.4 se zabývá volbou nejvhodnějšího způsobu síťové komunikace pro potřeby a budoucí rozvoj programu. V sekci 5.5 je popsán návrh rozhraní webové služby. V posledních dvou sekcích 5.6 a 5.7 je probírána architektura aplikace.

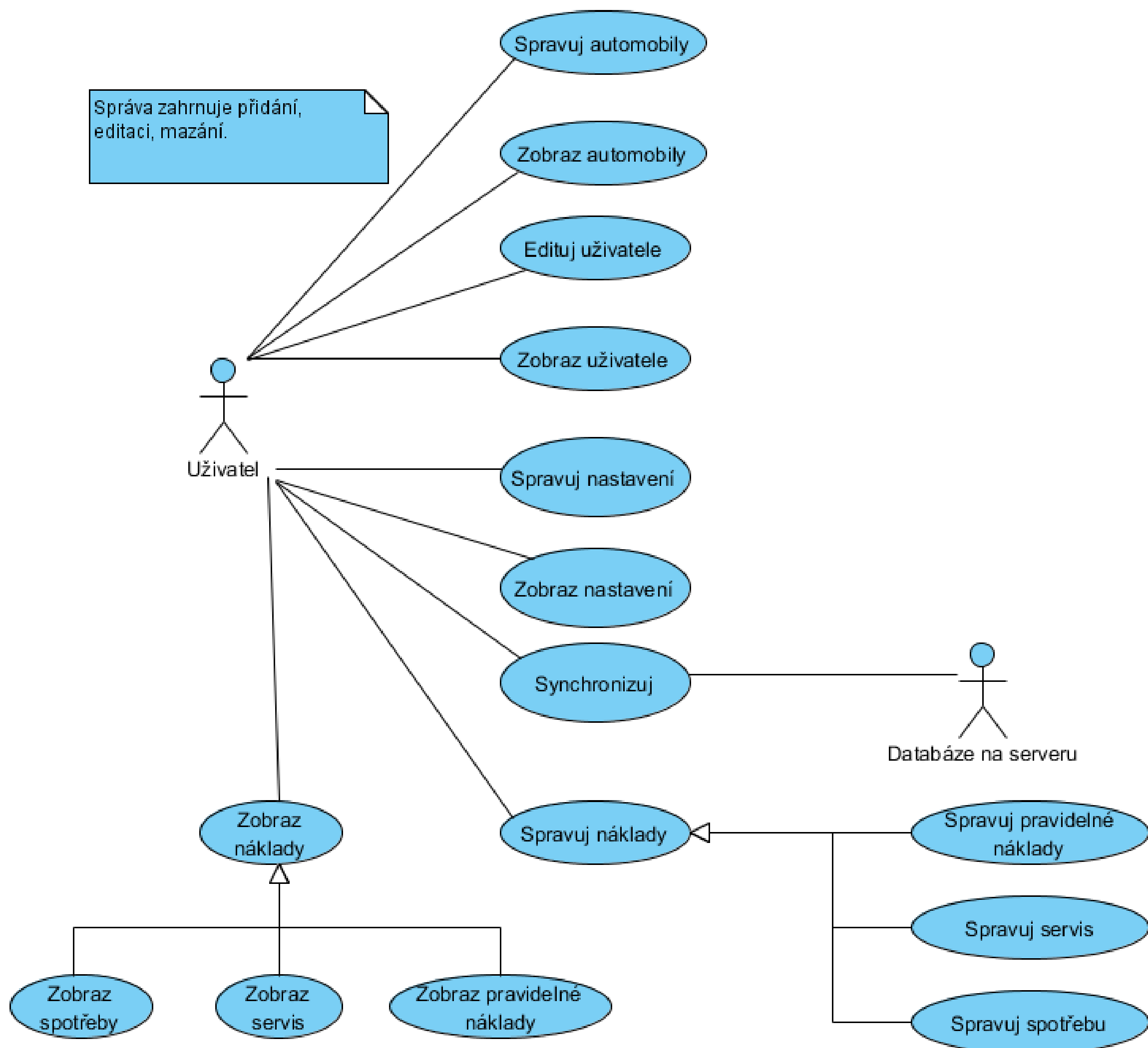
5.1 Funkčnost

Základní funkčnost aplikace je zobrazena pomocí diagramu případu užití na obrázku 5.1. Jak je vidět, v tomto diagramu případu užití existují dva aktéři.

- Uživatel.
- Databáze na serveru.

Aktér **Uživatel** reprezentuje reálného uživatele telefonu. Jak vidíme, **Uživatel** si může zobrazit seznam dostupných automobilů a zároveň je může také spravovat. Pod pojmem „správa“ je myšleno vytvoření nového automobilu a vyplnění všech potřebných atributů, dále pak jeho editace a případné smazání. **Uživatel** také může editovat své údaje a samozřejmě je i zobrazit. **Uživatel** má také možnost synchronizovat své záznamy se záznamy ze serveru. Při této volbě jsou všechna data sesynchronizována s patřičnými záznamy na serveru. Aplikace musí mít pro synchronizaci dostatek informací, proto má uživatel také možnost editovat a zobrazit nastavení, kde se vyplní všechny potřebné položky. V tomto případu použití se vyskytují také jednoduché dědičnosti. Obě se týkají nákladů za automobil. První dědičnost slouží pro zobecnění správy výdajů a druhá pro zobecnění zobrazení výdajů.

Náklady jsou tří druhů, a to:



Obrázek 5.1: Diagram případů užití.

- Spotřeba.
- Servis.
- Pravidelné náklady.

Správa spotřeb slouží pro zadávání hodnot spotřebovaného paliva. Správa v tomto smyslu opět obsahuje vytváření, editaci a mazání jednotlivých záznamů. Náklady na servis jsou náklady za výměny olejů, pneumatik a jiných nutných výdajů. Pravidelné náklady jsou pak, jak již název napovídá, opakující se výdaje. Jako příklad můžeme uvést dálniční známku a výdaje na technické kontroly nebo různá pojištění.

5.2 Rozhraní aplikace

Rozhraní aplikace by mělo být jednoduché a maximálně intuitivní. Návrh uživatelského prostředí je v mnoha ohledech jiný než návrh uživatelského prostředí pro desktopové systémy.

Musí se počítat jak s velikostí displeje, tak s nepřesností ovládání pomocí prstů. Aplikace musí dovolit pohodlně zadat všechny potřebné údaje způsobem, který bude pro uživatele nejefektivnější.

5.3 Databáze

Aplikace potřebuje ukládat data. Jako vhodné řešení se jeví ukládání dat do lokální databáze, která je obsažena přímo v telefonu. Zařízení s systémem Android v sobě obsahují zabudovanou SQLite databázi. Oproti tomu pro serverovou část je již lepší nasadit plnohodnotnou databázi, protože poskytuje lepší podporu pro multiuživatelský přístup. Zároveň by tato databáze měla být alespoň částečně rozšířena mezi hostingovými službami.

Aplikace využívá databázi v mobilním zařízení pro uložení všech uživatelem zadaných dat. Zaznamenávají se do ní uživatelé, informace o automobilech a samozřejmě také informace o jednotlivých výdajích za osobní automobil. Tímto jsou myšleny všechny pravidelné výdaje, výdaje za servis a pak samozřejmě náklady na palivo.

5.3.1 Mobilní zařízení - SQLite

Pro uložení dat v mobilním zařízení je nejvhodnější použít SQLite databázi přímo v mobilním zařízení, protože je nedílnou součástí každého dodávaného zařízení. SQLite je takzvaná embedded databáze. Neběží v nezávislém vlastním procesu, ale koexistuje s aplikací, která ji využívá. Její kód se proplétá s kódem aplikace. V tradičním přístupu, kdy server běží v separátním procesu, musí mezi klientem a serverem existovat nějaký druh komunikace. Nejčastěji se jedná o TCP/IP připojení, pomocí kterého jsou zasílány příkazy na server a následně i navraceny výsledky. Výhoda embedded přístupu je v tom, že není nutno jakkoliv nastavovat administrátorská a přístupová práva, klient i server běží v rámci jednoho procesu a tím je redukována režie spojená se síťovou komunikací. Proces přistupuje přímo k databázovému souboru, který je zapsán na paměťovém médiu. Databáze SQLite podporuje transakce, přičemž jsou zajištěny všechny podmínky ACID (atomičnost, konzistence, izolovanost, trvanlivost). Pro další informace lze nahlédnout do referenční literatury [5] a [18].

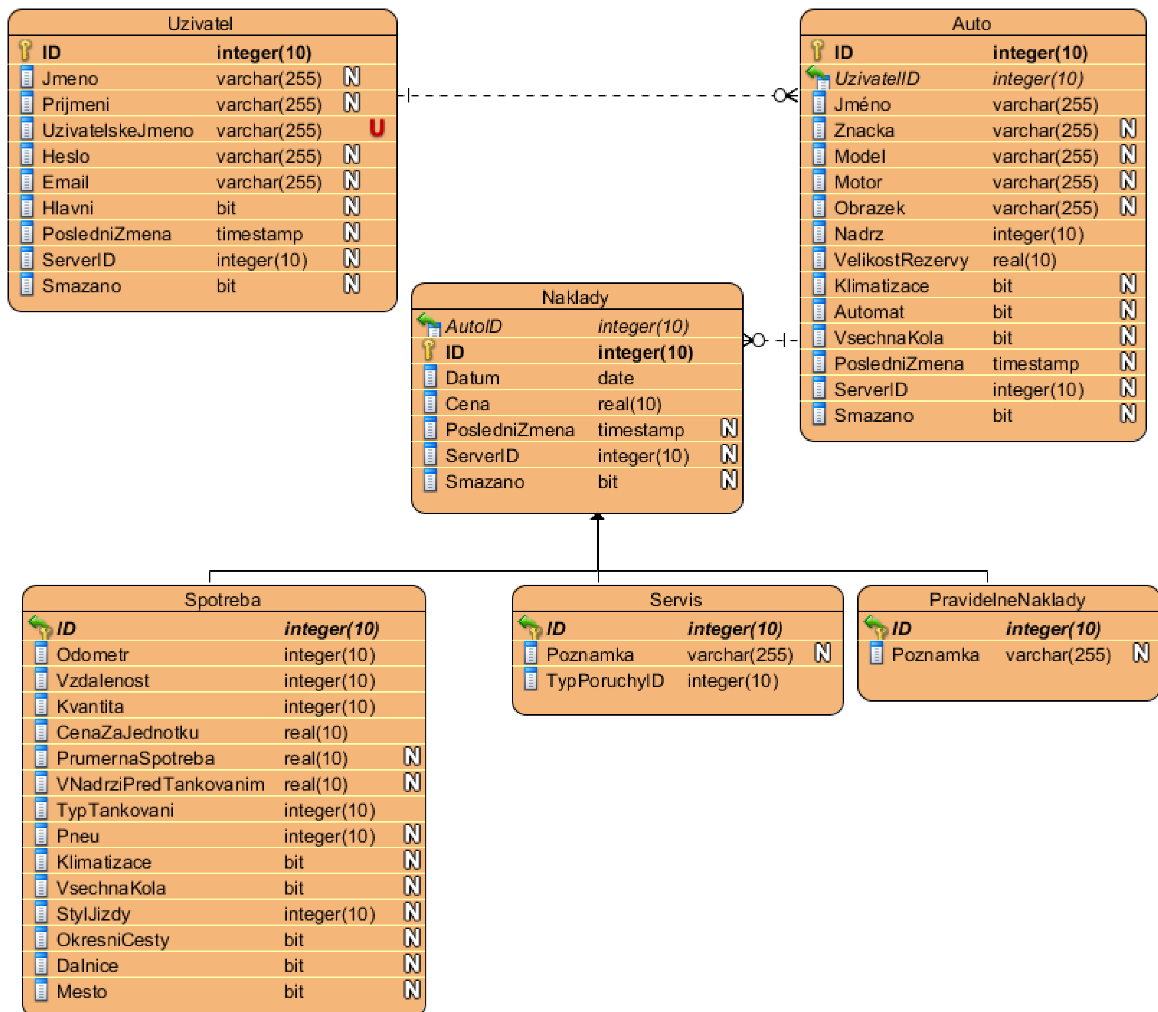
ER diagram na obrázku 5.2 zobrazuje navrženou strukturu databáze na mobilním zařízení. Stojí za povšimnutí, že ve všech tabulkách jsou sloupce **ServerID** a **PosledniZmena**, tato dvě pole jsou zde kvůli synchronizaci. **ServerID** je jednoznačná identifikace daného záznamu na serveru. Sloupec **PosledniZmena** je časové razítko záznamu, obsahuje datum jeho poslední aktualizace. Díky tomu je zřejmé, jestli je aktuální verze záznamu na serveru nebo na mobilním zařízení.

Tabulka Uzivatel

Tato tabulka obsahuje informace o jednotlivých uživatelích systému. Uživatel musí vyplnit své uživatelské jméno a heslo, ostatní položky jsou nepovinné. Uživatel může ke svému účtu doplnit jméno, příjmení a také e-mail. Jak je vidět, je zde vazba 1–N s tabulkou **Auto**, což znamená, že uživatel může spravovat více aut.

ID – Primární klíč této tabulky.

Jmeno – Tento sloupec nese hodnotu reprezentující jméno registrovaného uživatele. V databázi SQLite není datový typ `VARCHAR`, proto je použit datový typ `TEXT`, který v SQLite databázi slouží pro uchovávání řetězců.



Obrázek 5.2: ER diagram zobrazující databázi na mobilním zařízení.

Prijmeni – Sloupec sloužící pro uchovávání příjmení.

UzivatelskeJmeno – Sloupec, jehož hodnota je unikátní. Uživatelské jméno je jedinečné pro každého uživatele a jednotlivé hodnoty se tak mohou v tabulce vyskytovat pouze jednou.

Heslo – Obsahuje hash hesla daného uživatele, který je uložen jako textová hodnota.

Email – Tento sloupec uchovává e-mailovou adresu klienta.

PosledniZmena – V tomto sloupci je uchována hodnota poslední modifikace. Díky informaci obsažené v tomto sloupci máme přehled, jestli je novější záznam na serveru nebo na mobilním zařízení. Jelikož SQLite databáze nemá nativně datový typ pro datum, byl pro uchování hodnoty byl zvolen datový typ `TEXT`. Druhou možností je uchovávat počet milisekund od předem stanoveného data.

Hlavni – Určuje, zda se jedná o hlavní uživatelský účet.

ServerID – Sloupec nese hodnotu ID daného záznamu v databázi na serveru. Díky tomuto údaji jej může přesně a jednoduše identifikovat. Pro jeho uložení je využit datový typ `INTEGER`.

Smazano – Indikace, zda je daný záznam stále aktivní, nebo byl již smazán. Datový typ `BOOLEAN` není SQLite databází podporován, pro jeho reprezentaci se musí použít `INTEGER`, přičemž 1 reprezentuje hodnotu pravda a 0 hodnotu nepravda.

Tabulka Auto

Tato tabulka uchovává všechny potřebné údaje o automobilu. Uživatel musí vyplnit minimálně jméno automobilu, toto jméno je pak používáno dále v aplikaci v různých přehledových a výběrových menu. K automobilu může samozřejmě také doplnit jméno, značku, model, typ motoru a obrázek. Mezi aplikačně důležité údaje patří **Nádrž**, **VelikostRezervy**, **Klimatizace**, **Automat**, a **VsechnaKola**. Pokud má auto vyplněna tato pole, formuláře poté navíc automaticky nabízejí tyto rozšiřující volby. Tabulka **Auto** obsahuje tyto sloupce:

ID – Primární klíč tabulky Auto.

UzivatelID – Cizí klíč do databázové tabulky uživatel, určuje kterému uživateli náleží daný vůz.

Jmeno – Sloupec obsahuje jméno daného auta, pod tímto jménem je automobil reprezentován v potřebných dialozích.

Znacka – Hodnota reprezentující tovární značku automobilu, například Škoda.

Model – Model daného automobilu, například Octavia.

Motor – Definice motoru, kterým je daný automobil osazen. Uživatel zde může zadat ať již objemy válců, výkony nebo kroutící momenty.

Obrazek – Cesta k obrázku, který bude u daného automobilu zobrazen.

Nadrz – Velikost nádrže na palivo, tato hodnota je velmi důležitá při výpočtech množství natankovaného/spotřebovaného paliva.

VelikostRezervy – Velikost rezervy uložena jako `REAL`, což je datový pro čísla s plovoucí desetinnou čárkou na SQLite. Tento údaj má hlavně informativní charakter při zadávání zbylé hodnoty paliva v nádrži.

Klimatizace, Automat, VsechnaKola – Indikují, zda automobil má některou z těchto vlastností, protože tyto hodnoty se mohou značně podílet na hodnotě celkové spotřeby.

PosledniZmena, ServerID, Smazano – Tyto sloupce mají stejný význam a datový typ jako u předešlé tabulky **Auto**.

Tabulka Naklady, Spotreba, Servis, Pravidelne naklady

Tabulka **Naklady** je ve vztahu N–1 s tabulkou **Auto**, díky čemuž může automobil mít více záznamů o nákladech. Tyto jednotlivé tabulky spolu velmi úzce souvisí. Tabulky **Spotreba**, **Servis** a **PravidelneNaklady** jsou specializací tabulky **Naklady**. Tabulka **Naklady** je jejich generalizací. Společné znaky jednotlivých tabulek jsou sdruženy v generalizované tabulce **Naklady**. Specializované tabulky pak nesou specifické rysy jednotlivých nákladů na automobil.

Spotreba – Tato tabulka je specializací nejčastějšího druhu nákladů, a to nákladů na jednotlivá tankování pohonných hmot.

Servis – Tato tabulka je taktéž specializací nákladů. Udržuje záznamy o všech servisních a ostatních jednorázových výdajích.

PravidelneNaklady – Opět se jedná o specializaci nákladů. Tabulka sdružuje pravidelné výdaje na auto.

Všechny tabulky také mají společné sloupce **Datum** označující datum vzniku nákladu. Tabulky sdílejí také sumarizovanou cenu, která je uložena jako `REAL`. Zbylé sloupce pro uložení poslední změny, ID záznamu na serveru a data poslední změny, jsou stejné jako v ostatních tabulkách.

Tyto tabulky obsahují následující sloupce :

Odometr – tento sloupec udává hodnotu odometru kdy se daný záznam uskutečnil.

Vzdalenost – obsahuje hodnotu ujeté vzdálenosti od posledního tankování. Při výpočtu průměrné spotřeby je toto základní údaj.

Kvantita – nese hodnotu počtu natankovaných jednotek pohonných hmot.

CenaZaJednotku – reprezentuje cenu jedné jednotky paliva pohonných hmot.

PrumernaSpotreba – nese vykalkulovanou hodnotu průměrné spotřeby na základě tankování.

VNadrziPredTankovanim – obsahuje hodnotu počtu jednotek paliva v nádrži před natankováním. Bez tohoto údaje bychom nemohli spolehlivě vypočítat množství spotřebovaného paliva, pokud bychom netankovali do plné nádrže.

TypTankovani – určuje, jestli bylo tankováno do plné nádrže, případně pouze částečně, nebo dokonce poprvé. Hodnota této enumerace se ukládá jako celé číslo.

Pneu – značí, na jakých pneumatikách bylo ježděno, zda-li se jednalo o letní, zimní či celoroční.

Klimatizace a VsechnaKola – indikují, zda v průběhu jízdy byla zapnuta klimatizace nebo náhon všech čtyř kol.

StylJizdy – enumerace určující jakým stylem jízdy bylo jezděno.

OkresniCesty, Dalnice, Mesto – určují, na jakých cestách jsme v jezdili. Spotřeby se u automobilů výrazně liší pro městské prostředí, dálnice a okresní cesty.

Poznamka – uživatel může zaznamenat poznámku o servisním úkonu.

TypPoruchy – enumerace reprezentující typ opravy.

5.3.2 Server

Samotný server slouží jako frontend databáze. Tuto databázi bude využívat více klientů, kteří se zároveň pomoci ní mohou synchronizovat. Na serveru je několik možností ohledně vhodné volby databáze. Kvalitních databázových systémů je mnoho. Po rozumné úvaze a průzkumu podpory databází na hostingových službách se jako vhodní kandidáti jeví tyto databázové systémy:

- MySQL,
- PostgreSQL,
- SQLite.

Databáze SQLite by byla logická volba, protože je používána i na telefonu a měli bychom tedy stejné možnosti jako na mobilním zařízení. Příkazy jazyka SQL použité na mobilním zařízení by fungovaly i na serveru. Tato databáze ale pro svoji embedded povahu není mezi hostingovými službami prakticky rozšířena. Navíc by byl problematický vícenásobný přístup do databáze. SQLite jej sice umožňuje, ale tento přístup není tak vysoce sofistikovaný jako u velkých databází využívajících serverového přístupu, proto tato možnost není již dále rozebírána. Oproti tomu, zbylé dvě kandidátské databáze jsou v hostingových službách hojně nasazovány.

MySQL je relační databázový systém běžící jako server poskytující mnohouživatelský přístup do databází. Je to nejpoužívanější databázový systém pro online aplikace. Běží na mnoha počítačových architekturách a na více než dvaceti různých platformách. Jedná se o open source databázový systém volně dostupný pod licencí GNU GPL. Databáze samozřejmě podporuje transakce a jsou k dispozici ovladače pro provádění dotazů nad mnoha různými jazyky, jako například C/C++, Java, Perl, Python a mnoho jiných. Další informace lze nalézt v [23] [28].

PostgreSQL je open source objektově relační databáze. Má za sebou více než 15 let vývoje a běží na všech významných operačních systémech (Linux, UNIX, Mac OS X, Windows, Solaris a další). Je zde plná podpora pro cizí klíče, pohledy, příkazy join, databázové triggery a uložené procedury. Tato databáze má nativní programové rozhraní pro jazyky C/C++, Java, Ruby, Python, Perl, .NET, Ruby a další. Více informací lze nalézt v [19].

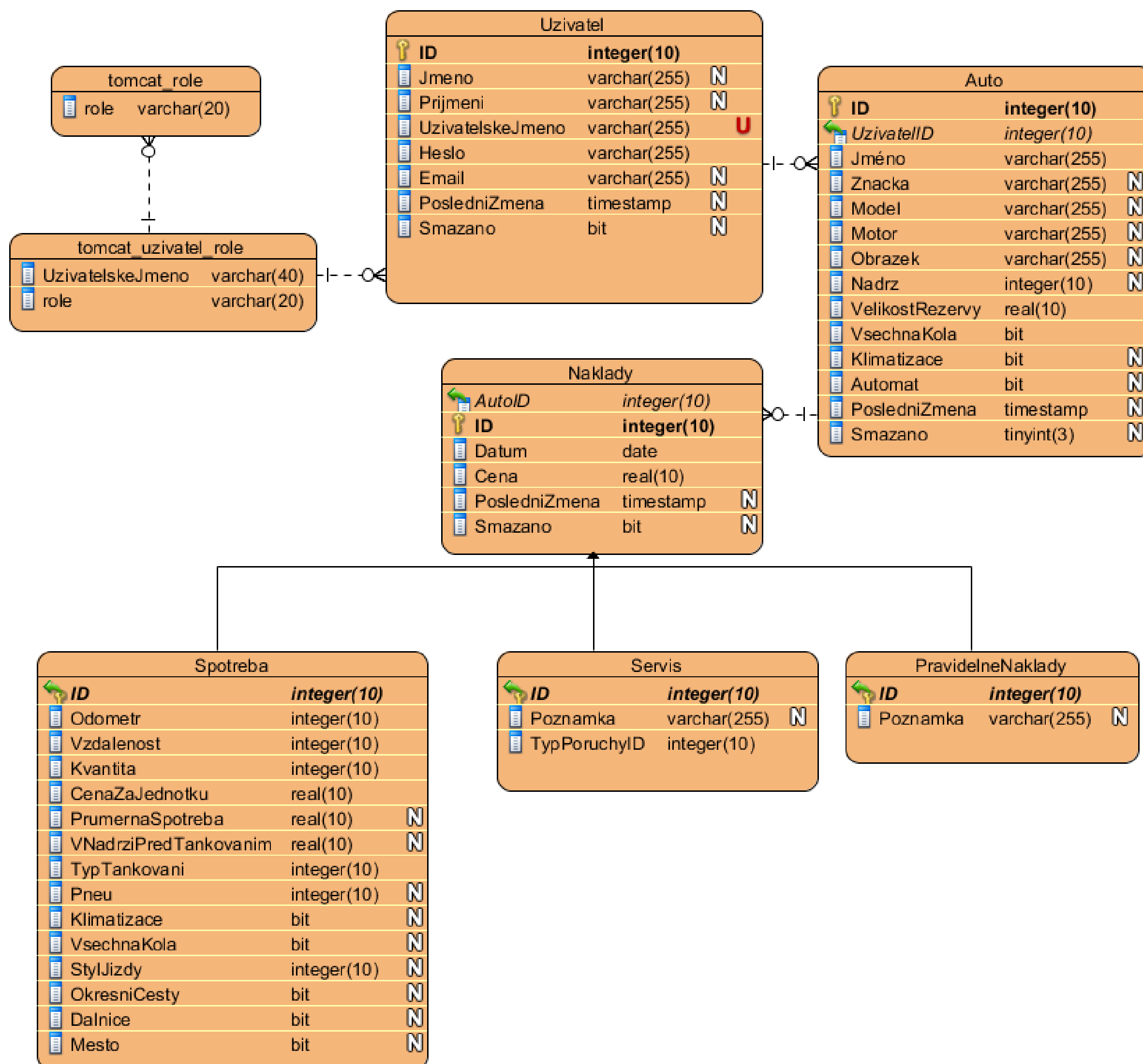
Obě zbývající databáze mají svoje přednosti. Databáze MySQL je ale pro účely této práce vhodnější, jedním z důvodů je ten, že toto databázové řešení je velmi rozšířeno na mnoha webových hostingových službách. Pokud by ji chtěl uživatel zprovozňovat sám, je velká pravděpodobnost, že ji již někdy použil a má s ní zkušenosti. Pokud si bude uživatel chtít zprovoznit vlastní server, nebude se tak muset s největší pravděpodobností učit nové věci, což je z uživatelského hlediska velmi pozitivní.

ER diagram zobrazující strukturu databáze na serveru je zobrazen na obrázku 5.3.

Serverová část databáze je velmi podobná databázi na mobilním zařízení, která je popsána výše. Chybí zde ovšem sloupec **ServerID**, které na serveru postrádá smysl a není proto potřeba.

Jsou zde navíc dvě tabulky, a to **tomcat_role** a **tomcat_uzivatel_role**. Tyto dvě tabulky jsou v serverové části z důvodu automatické autentizace uživatele webovým serverem.

Tabulka **tomcat_role** obsahuje pouze jeden sloupec, a to **role**. V tomto sloupci jsou uvedeny všechny role, které mohou na daném serveru existovat.



Obrázek 5.3: ER diagram zobrazující databázi na serveru.

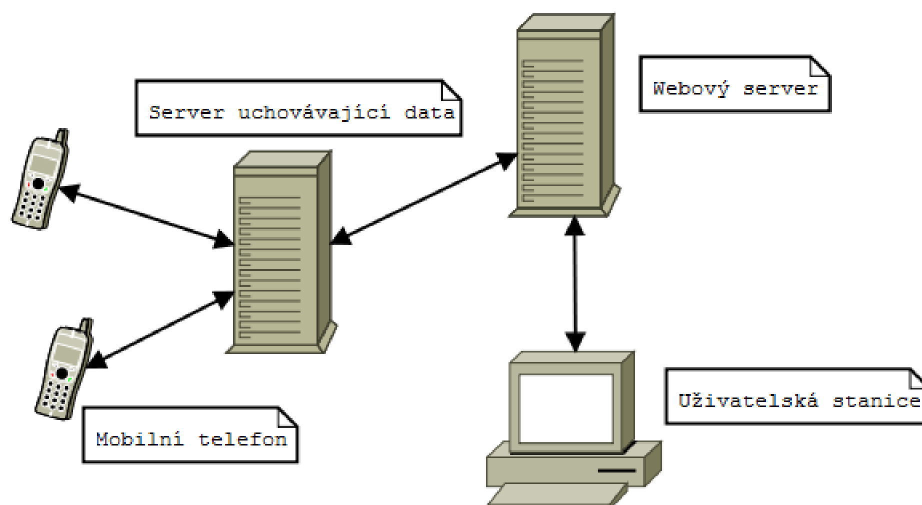
Tabulka **tomcat_uzivatel_role** je vazební tabulka mezi uživateli a jejich rolemi. Sloupec **UzivatelskeJmeno** je cizím klíčem do tabulky **Uzivatel** a sloupec **role** je cizím klíčem do tabulky **tomcat_role**. V této tabulce je tedy přesně definováno, který uživatel má jakou roli.

5.4 Síťová komunikace

Aplikace má umožňovat synchronizaci databáze se vzdáleným serverem. Schéma navržené pro komunikaci je zobrazeno na diagramu 5.4.

Centrální částí celého návrhu je server uchovávající data. Jeho technologická implementace bude předmětem diskuze v dalších částech této práce. Tento server je ústřední bod pro synchronizaci dat. Jeho komunikační protokol musí být dostatečně otevřený, aby s ním mohli komunikovat jednotliví mobilní klienti. Tito klienti mohou být různé mobilní

platformy, primárně však Google Android. Server musí zastiňovat databázi a musí k ní poskytovat rovnoměrný přístup pro všechny klienty. Jako zdroj pro získávání a ukládání dat jej může používat i webový server, který dále poskytuje data jednotlivým klientským stanicím přistupující na tento server. Uživatelé tak mohou editovat svá data jak přes mobilní rozhraní, tak přes webové rozhraní. V obou případech se data uloží do stejného datového úložiště.



Obrázek 5.4: Diagram znázorňující komunikaci.

Pro implementaci síťové komunikace je možno zvolit více rozdílných technologií a přístupů. Tyto jednotlivé technologie budou rozebrány v následující části textu. Následně bude zvolena nejvhodnější technologie pro potřeby navrhované aplikace.

5.4.1 Vlastní protokol

První možností je vlastní správa síťové komunikace pomocí skupiny tříd z balíčku `java.net`. V tomto balíku jsou obsaženy všechny potřebné třídy pro síťovou komunikace za pomoci soketů. Je zde podporována jak nezabezpečená komunikace (`TcpSocket`), tak zabezpečená SSL komunikace (`SSLSocket`).

Pomocí jak zabezpečené, tak nezabezpečené komunikace by probíhala výměna informací. Aby si obě strany mohly tyto informace vyměňovat a efektivně spolu komunikovat, musel by být pečlivě navržen komunikační protokol. Při jakékoliv změně (například databáze) by obě strany musely rozšířit protokol a reflektovat tyto změny.

Jelikož hlavními klienty serveru jsou mobilní telefony, komunikace mezi serverem a klientem tak může probíhat po relativně nestabilním síťovém spojení zajišťovaném GSM sítí. Toto spojení by mohlo být v průběhu synchronizace v nepříznivých podmínkách relativně často přerušováno. Server by si tak musel pamatovat stav komunikace s klientem a navazovat komunikaci v bodu, kde skončili synchronizaci, případně vrátit synchronizační změny a začít od znovu, jakmile bude dostupné připojení.

5.4.2 WCF

Windows Communication Foundation je jednou z mnoha technologií, kterou přinesl .NET framework 3.0 a která je s každou další verzí rozšiřována. Tato technologie slouží jako API pro výstavbu SOA[1].

WCF vlastně nereprezentuje jednu samostatnou technologii, ale sjednocuje a zaštiťuje velké množství technologií jiných. WCF je navrženo pro podporu distribuovaných výpočtů, kde jsou služby poskytovány klientům a ti je konzumují. Služba může mít více klientů a klienti mohou využívat více služeb.

Klienti se na službu připojují pomocí *Endpoint*. Každá služba vystavuje jeden nebo více těchto přístupových bodů, které specifikují na jaké URL adrese je daná služba dostupná, a zároveň vazebné vlastnosti (*binding properties*), které specifikují způsob, jakým jsou data přenášena. Před zahájením komunikace musí mít obě strany nastaveny stejný *Endpoint* a příslušné *binding properties*, jinak mezi sebou nebudou schopny komunikovat.

Na zařízeních s .NET frameworkem je možno na základě konfigurací služeb automaticky vygenerovat klienty a programátor pak pouze volá metody vygenerovaného objektu. Při správně konfiguraci se tak vůbec nemusí starat o jakákoliv síťová volání a vše je obstaráno automaticky vygenerovaným klientem. Na jiných systémech je však nutno tyto metody invokovat ručně, což může být v závislosti na nastavení *Endpoint* a *binding properties* poměrně komplikované.

5.4.3 Webové služby

Klasické webové služby

Klasické webové služby jsou populární hlavně v prostředí velkých podniků. Jejich rozhraní je popsáno pomocí automaticky zpracovatelného formátu WSDL[2], který je založen na XML. Služba popsaná jazykem WSDL velmi často využívá SOAP[3] a XML Schema pro poskytování služeb. Zprávy jsou zabaleny do SOAP obálky, kde součástí této obálky je hlavička a tělo zprávy, přičemž samotné informace přenášené zprávou jsou uloženy v části těla.

RESTful webové služby

RESTful webové služby jsou založeny na principech REST[4], což je styl softwarové architektury pro distribuované systémy. Pokud technologie odpovídá myšlenkám REST, nazýváme ji RESTful – odtud pojem RESTful webové služby.

Tato architektura se skládá z klientů a serveru. Jednotliví klienti vysílají požadavky na server, zde je požadavek přijat a zpracován serverem. Ten následně zašle klientovi nazpět odpověď.

Tento koncept velmi využívá protokolu HTTP[5], který je stejně jako REST bezstavový, navíc jsou s ním spojeny rozsáhlé prostředky pro popis (odpovědní kódy, HTTP metody, ...). Avšak REST není omezen pouze na protokol HTTP, ale může využívat jakýkoliv jiný protokol, který poskytuje v dostatečně bohaté míře prostředky pro popis.

[1] Service-oriented Architecture.
[2] Web Services Description/Definition Language.
[3] Simple Object Access Protocol.
[4] REST – Representational State Transfer.
[5] HTTP – Hypertext Transfer Protocol.

Pro základní operace s daty (takzvané CRUD[6]) a pro manipulaci s poskytovanými zdroji dané služby se používají tyto HTTP metody:

GET – Slouží k získání dat daného zdroje.

POST – Slouží primárně ke vkládání dat.

PUT – Pomocí tohoto příkazu provádíme aktualizace jednotlivých poskytovaných zdrojů.

DELETE – Tímto příkazem vymažeme daný zdroj.

Jednotlivé zdroje, které webová služba poskytuje, jsou identifikovány pomocí jejich URL. Služba podporuje přenášení těchto zdrojů ve formátu XML a JSON, přičemž některé služby mohou podporovat oba tyto formáty. Pokud služba podporuje JSON formát přenosu zpráv, lze s ní komunikovat prostřednictvím technologie AJAX, což ji dělá zajímavou pro vývojáře webových aplikací.

Oproti klasický webovým službám popsaných komplexním jazykem WSDL a přenášených pomocí SOAP zpráv, mají webové služby založené na REST velmi jednoduché přístupové rozhraní, se kterým se velmi pohodlně pracuje. Klienti HTTP jsou dostupní na prakticky všech myslitelných platformách, takže s těmito webovými službami lze efektivně komunikovat z velkého množství klientů. RESTful webové služby se navíc velmi hodí do prostředí mobilních aplikací. Mobilní zařízení stále nemusí disponovat plnohodnotným připojením, a tak SOAP zprávy mohou představovat zbytečnou a nechtěnou zátěž pro již tak malou kapacitu přenosového média. Pro další informace o webové službě je možno nahlédnout do literatury [20].

Samotná popularita webových služeb na bázi REST neustále stoupá a začíná převyšovat klasické webové služby na bázi WSDL a SOAP.

5.4.4 Shrnutí

V této sekci byly diskutovány jednotlivé možnosti síťové komunikace. Jako první možnost bylo zváženo využití soketů a komunikace pomocí vlastního protokolu. Tato možnost se však příliš nehodí do oblasti mobilních komunikací, protože by protokol musel být poměrně odolný proti nárazovým výpadkům spojení. Navíc z pohledu budoucího rozšiřování služby na jiné systémy je tato možnost zbytečně náročná.

Druhou možností je využití některé z variant komunikace poskytované ve WCF od firmy Microsoft. Tato možnost však také není příliš vhodná, protože volání služeb založených na WCF není taktéž příliš pohodlné ze systémů, které nejsou vyvíjeny formou Microsoft. To samozřejmě neplatí, pokud bychom vyvíjeli pomocí WCF například RESTful webovou službu.

Třetí možností je použití klasické webové služby popsané pomocí WSDL využívající v rámci komunikace SOAP. Tyto webové služby balí jednotlivé zprávy do SOAP obálek. Každá zpráva mezi klientem a službou tak obsahuje data zaobalená ve velice obsáhlém XML. Kromě dat obsahuje tato zpráva i velké množství dodatečných popisných informací, které nemusí být ani využívány. Kromě toho, že je zpráva velice obsáhlá, což je pro použití v mobilních zařízeních nevhodné, je také problematické parsování a vytváření dotazů. Dotazy musí být v SOAP obálce, s tou lze na systému Android pracovat pomocí *ksoap2* knihovny. Jednotlivé části dat se však musejí manuálně parsovat a použití tak není příliš efektivní a pohodlné.

[6]Create, Read, Update, Delete.

Poslední zvažovaná možnost je použití RESTful webové služby. Oproti klasickým webovým službám hodnocených v předchozím odstavci, je komunikační protokol těchto služeb podstatně jednoduší a není zatížen žádnými obálkami a daty navíc. To je jeden z důvodů, proč se skvěle hodí pro použití v mobilních aplikacích – nízká zátěž sítě. V systému Android je navíc zabudovaný klient protokolu HTTP, který umožňuje snadnou tvorbu dotazů. Samotné poskytované zdroje se dají serializovat do XML pomocí serializéru a není tak nutné tvořit obsah zprávy ručně.

Z výše uvedeného vychází jako nejlepší kandidát RESTful webová služba. V dalším návrhu aplikace proto bude počítáno s využitím této technologie.

5.5 Rozhraní webové služby

Pro synchronizaci byla zvolena webová služba na bázi REST. Jelikož aplikace počítá do budoucna s rozšiřováním, je potřeba dostatečně podrobně a důsledně navrhnout veřejné rozhraní pro synchronizaci z jiných systémů. Toto rozhraní musí poskytovat možnosti pro:

Jednoduché operace – Rozhraní sloužící pro provádění atomických operací nad daty. Nejedná se zde o hromadné operace, protože použití tohoto veřejného rozhraní je zamýšleno pro účely jednoduchých nedávkových operací. Předpokládaní klienti tohoto rozhraní jsou webové aplikace, které ze své povahy mají neustálý přístup k internetu, takže mohou akce uživatele ihned provádět a reflektovat.

Synchronizační operace – Toto rozhraní slouží pro vykonávání atomických dávkových operací nad daty. Využíváno je pro synchronizaci s vyvíjenou aplikací a do budoucna ji mohou využívat například stejně zaměřené aplikace vyvinuté pro jiné platformy mobilních zařízení.

Webové služby na bázi REST se začínají objevovat stále častěji. Příkladem může být na HTTP založené rozhraní populární služby sloužící pro shromažďování odkazů na webové stránky *delicious*, jejichž API pro vytváření těchto tagů je k dispozici pro nahlédnutí zde [29]. Toto API nevyužívá plně všechny myšlenky REST, které byly popsány v podsekci 5.4.3. Všechny dotazy (vytváření, mazání, úprava) jsou totiž prováděny pouze pomocí HTTP příkazu GET a další příkazy jako PUT, POST, DELETE nejsou vůbec využívány. Navržená webová služba přijímá a poskytuje zdroje serializované do XML.

5.5.1 Jednoduché operace

Zde je popsáno rozhraní webové služby pro jednotlivé zpřístupněné prostředky.

Automobily

Tabulka 5.1: Získání automobilu s daným ID.

Získání automobilu s daným ID	
URL	/data/{uzivatel}/car/{id}
Metoda	GET
Navrací	200 OK & XML popis automobilu 401 Unauthorized

Postup získání automobilu s daným ID je zřejmý z tabulky 5.1. Cesta je parametrizována dvěma parametry. Prvním parametrem je **uživatelské jméno** uživatele vlastnícího automobil {uzivatel} a druhým parametrem je **id** požadovaného vozidla, náležícího danému uživateli {id}. Vysláním požadavku metodou *GET* na adresu z tabulky dostaneme XML, které obsahuje data daného automobilu. Server může odpovědět návratovým kódem *200 OK* značícím, že vše proběhlo v pořádku, a společně s ním zašle data o automobilu. Pokud se nezdařila autentizace, server odpoví kódem *401 Unauthorized*. Ukázka struktury XML souboru je zobrazena ve výpisu 5.1.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<car>
    <AC>true</AC>
    <AT>true</AT>
    <active>false</active>
    <allWheels>false</allWheels>
    <brand>VW</brand>
    <deleted>false</deleted>
    <id>0</id>
    <lastChange>2011-04-23T11:25:58.000+0200</lastChange>
    <model>Passat</model>
    <name>Zlutak</name>
    <reserve>5.0</reserve>
    <serverID>4</serverID>
    <tank>70</tank>
    <uzivatelID>4</uzivatelID>
</car>
```

Výpis 5.1: Ukázka serializovaného souboru s daty o automobilu.

Formát pro přenos datumu je zobrazen v serializovaném poli lastChange. Jedná se o textovou reprezentaci data a času založenou na normě ISO8601. Hodnoty jsou řazeny od významnějších po méně významné. Jednotlivé hodnoty mají pevně daný počet číslic, takže pokud je číslice reprezentující hodnotu menší délky než udává standard, je zleva dorovnána nulami (například 2009-1-2 je zarovnáno na 2009-01-12). Standardním oddělovačem jednotlivých hodnot je „–“, pro hodnoty času je to „:“. Součástí zde zvoleného formátu je i specifikace časového pásma.

Tabulka 5.2: Vložení automobilu.

Vložení automobilu	
URL	/data/{user}/car/
Metoda	POST
Navrací	200 OK & XML popis automobilu s vyplněným ID 401 Unauthorized

Pro vložení automobilu je potřeba postupovat podle tabulky 5.2. Na adresu uvedenou v popisu se vyšle *HTTP POST* příkaz obsahující automobil, který chceme vložit, v XML formátu specifikovaném ve výpisu 5.1. Cesta je opět parametrizována. Jak je vidět, součástí cesty je i uživatelské jméno uživatele. Pokud se toto jméno neshoduje se zaslanou autentizací, server odpoví *401 Unauthorized*. V případě, že se data vložila bez problémů, server odešle nazpět pro potvrzení úspěšného vložení automobil s vyplněným polem **ServerID** společně s kódem *200 OK*.

Tabulka 5.3: Změna automobilu.

Změna automobilu	
URL	/data/{user}/car/
Metoda	PUT
Navrací	200 OK & XML popis automobilu 401 Unauthorized

Postup aktualizace automobilu je zobrazen v tabulce 5.3. Vysláním požadavku *HTTP PUT* na adresu z tabulky provedeme aktualizaci záznamu. Pole **ServerID** musí být předvyplněno na hodnotu ID automobilu na serveru, aby webová služba zaktualizovala správný záznam. Pokud byl požadavek zpracován a vše proběhlo v pořádku, oznámí tuto skutečnost služba odeslaným návratovým kódem *200 OK* a vkládanými daty v podobě, ve které je vložila.

Tabulka 5.4: Vymazání automobilu s daným ID.

Vymazání automobilu s daným ID	
URL	/data/{user}/car/{id}
Metoda	DELETE
Navrací	200 OK 401 Unauthorized 404 Not found

Při mazání záznamu se žádná data společně s dotazem, na rozdíl od předchozích případů, neodesílají. Klient vyšle *HTTP DELETE* příkaz na adresu uvedenou v tabulce 5.4. Zde se jako parametry cesty udává přihlašovací jméno klienta a **ID** automobilu, který si přejeme smazat. Pokud vše proběhlo v pořádku, server odpoví návratovou hodnotou *200 OK*, pokud nastala chyba autorizace, server odpoví návratovou hodnotou *401 Unauthorized*. V případě chybějícího ID odpoví server hodnotou *404 Not found*.

Spotřeba

Tabulka 5.5: Přehled operací nad spotřebami.

Rozhraní pro spotřeby	
Metoda	**URL**
GET	/data/{user}/consumption/{id}
POST	/data/{user}/consumption/
PUT	/data/{user}/consumption/
DELETE	/data/{user}/consupmtion/{id}

Tabulka 5.5 zobrazuje postup volání jednotlivých metod pro práci se záznamy o spotřebě. Jsou zde opět všechny metody pro vytvoření, upravení, smazání a přečtení záznamu. Odpovědní kódy jsou stejné jako pro přístup k automobilům. Všechny přístupy jsou ošetřeny autentizací a v případě, že se uživatel pokouší přečíst například data jiného uživatele, než pro kterého se prokáže autentizací server, mu odpoví kódem stavu *401 Unauthorized*,

případně *404 Not found*. Příklad formátu XML, který webová služba produkuje a přijímá pro záznam o spotřebě, je na výpisu 5.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<consumption>
    <autoID>4</autoID>
    <date>2011-04-21T19:53:47.000+0200</date>
    <deleted>false</deleted>
    <id>0</id>
    <lastChange>2011-04-23T10:25:58.000+0200</lastChange>
    <price>45.0</price>
    <serverID>1</serverID>
    <AC>false</AC>
    <allWheels>false</allWheels>
    <averageConsumption>0.0</averageConsumption>
    <cityRoads>false</cityRoads>
    <countryRoads>false</countryRoads>
    <distance>0</distance>
    <highwayRoads>false</highwayRoads>
    <inTankBeforeFillUp>23.5</inTankBeforeFillUp>
    <odometer>5000</odometer>
    <pricePerUnit>45.0</pricePerUnit>
    <quantity>1.0</quantity>
    <rideStyle>Moderate</rideStyle>
    <tankType>First</tankType>
    <tires>Winter</tires>
</consumption>
```

Výpis 5.2: Ukázka serializovaného souboru se záznamem spotřeby.

Pravidelné náklady

Tabulka 5.6 obsahuje shrnutí možných operací nad pravidelnými náklady. Jsou zde všechny CRUD metody, pomocí kterých můžeme jednoduše manipulovat s záznamy o pravidelných nákladech.

Tabulka 5.6: Přehled operací nad pravidelnými náklady.

Rozhraní pro pravidelné náklady	
Metoda	**URL**
GET	/data/{user}/regularexpenses/{id}
POST	/data/{user}/regularexpenses/
PUT	/data/{user}/regularexpenses/
DELETE	/data/{user}/regularexpenses/{id}

Formát XML souboru, který webová služba produkuje a přijímá pro záznam o pravidelných výdajích, je na výpisu 5.9.

```
<?xml version="1.0" encoding="UTF-8"?>
<regularExpense>
    <autoID>4</autoID>
    <date>2011-04-22T13:46:14.000+0200</date>
    <deleted>false</deleted>
    <id>0</id>
    <lastChange>2011-04-22T13:46:24.000+0200</lastChange>
    <price>5648.0</price>
    <serverID>5</serverID>
```

```
    <note>nereknu vam nic</note>
</regularExpense>
```

Výpis 5.3: Ukázka serializovaného souboru s pravidelným výdajem.

Servis

Tabulka 5.7: Přehled operací nad servisní náklady.

Rozhraní pro servisní náklady	
Metoda	**URL**
GET	/data/{user}/servis/{id}
POST	/data/{user}/servis/
PUT	/data/{user}/servis/
DELETE	/data/{user}/servis/{id}

V tabulce 5.7 je zobrazen přehled operací, které mohou být vykonávány nad servisními výdaji. Formát XML souboru, který webová služba produkuje a přijímá pro záznam o servisním výdaji, je na výpisu 5.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<repair>
    <autoID>4</autoID>
    <date>2011-04-22T12:45:01.000+0200</date>
    <deleted>false</deleted>
    <id>0</id>
    <lastChange>2011-04-22T12:45:12.000+0200</lastChange>
    <price>556.0</price>
    <serverID>4</serverID>
    <note>hřebík</note>
    <servisType>NotSpecified</servisType>
</repair>
```

Výpis 5.4: Ukázka serializovaného souboru se servisním výdajem.

Registrace

Aby služba mohla autentizovat své uživatele, musí poskytovat možnost registrování nových uživatelů. Tabulka 5.8 zobrazuje možnosti registračního dotazu.

Tabulka 5.8: Registrace do služby.

Provádění registrace	
URL	/register
Metoda	POST
Navrací	200 OK & řetězec 403 Forbidden & řetězec

Registrace se provádí pomocí metody *HTTP POST* a odesláním společně s ní data o novém uživateli ve formátu prezentovaném na výpisu 5.5. Hodnoty elementů **id** a **serverID** jsou ignorovány a nejsou vloženy do databáze. Heslo musí být dodáno v podobě po zpracování hashovací funkcí MD5. V případě úspěšné registrace je navráceno ID uživatele,

pod kterým je veden na webové službě. Pokud se registrace nezdařila, služba vrátí *403 Forbidden* a společně s ním jednu z hodnot enumerace *LogOrRegEnum* specifikující blíže důvod chyby (například již existující uživatelské jméno).

```
<?xml version="1.0",encoding="UTF-8"?>
<user>
        <name>Karel</name>
        <surname>Pyšný</surname>
        <userName>karel</userName>
        <password>4a8a08f09d37b73795649038408b5f33</password>
        <id>0</id>
        <serverID>1</serverID>
        <email>karel@email.cz</email>
        <lastChange>2011-04-23T10:25:58.000+0200</lastChange>
        <deleted>false</deleted>
</user>
```

Výpis 5.5: XML formát pro reprezentaci uživatele.

Přihlášení

Klienti se mohou přihlásit na službu a ověřit si tak platnost dodaných uživatelských údajů. Tabulka 5.9 zobrazuje možnosti přihlašovacího dotazu. Společně s *HTTP POST* požadavkem se zasílají uživatelská data v podobě zobrazené na výpisu 5.5. Pokud byla registrace úspěšná, vrací služba ID uživatele jako potvrzení. Pokud se registrace nezdařila, služba navrátí *403 Forbidden* a společně s ním jednu z hodnot enumerace *LogOrRegEnum* specifikující blíže důvod chyby.

Tabulka 5.9: Příhlášení do služby.

Provádění přihlášení	
URL	/data/login
Metoda	POST
Navrací	200 OK & řetězec 403 Forbidden & řetězec

5.5.2 Hromadné operace

Zde je popsáno rozhraní pro hromadné operace. Tyto jsou využívány hlavně pro synchronizační účely, nic však nebrání tomu, aby byly využívány jinak.

Automobily

Tabulka 5.10: Získání automobilů.

Získání automobilů	
URL	data/{user}/cars/{čas}
Metoda	GET
Navrací	200 OK & automobily 401 Unauthorized

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cars>
	<car>
		...
	</car>
	...
</cars>
```

Výpis 5.6: XML formát pro reprezentaci automobilů.

Tabulka 5.10 znázorňuje možnost hromadného získání automobilů. URL adresa má dva proměnné parametry, prvním z parametrů je jméno uživatele, kterému automobily náleží, druhý parametr je čas změny. Tento čas změny je celočíselná hodnota reprezentující počet milisekund od 1. ledna 1970, 00:00:00 GMT. Toto datum a čas bylo zvoleno, protože se jedná o standardní datum počátku objektu jazyka Java *java.util.Date*. Webová služba navrátí všechny automobily, jejichž čas poslední modifikace je pozdější, než toto datum a čas. Tyto jednotlivé automobily budou serializovány jako pole objektů do XML souboru, jehož ukázku můžeme vidět na výpisu 5.6. Pokud se nezdařila autentizace, tak služba vrátí stavový kód *401 Unauthorized*.

Tabulka 5.11: Hromadné operace s automobily.

Hromadné operace s automobily	
URL	data/{user}/cars
Metoda	POST
Navrací	200 OK & automobily 403 Forbidden

Primárně pro synchronizační účely slouží hromadné operace s automobily zaznačené v tabulce 5.11. Služba přijme data ve formátu specifikovaném ve výpisu 5.6. V poli přijatých automobilů jsou přijata jak data určená ke vložení, tak data pro aktualizaci a vymazání. Služba data roztřídí na základě jednotlivých parametrů. Pokud automobil nemá vyplněno pole **serverID**, tak se jedná o vkládání dat, pokud jej má vyplněno, jedná se o aktualizaci. Data určená k smazání jsou rozpoznána podle příznaku **deleted**.

Vše je vykonáno jako jediná transakce, takže nedochází k nekonzistentnosti dat. Pokud by se operace neprováděly v rámci jedné transakce, mohlo by v případě špatného připojení k síti, které je u mobilních zařízení poměrně časté, dojít k nekonzistentnosti dat. Pokud by příkazy byly striktně rozděleny na vkládání, aktualizaci a mazání, mohlo by se klientovi, který se pokouší o vložení a aktualizaci dat, podařit provést pouze vložení dat a aktualizace by zůstala neprovedena. Služba by tak až do další synchronizace měla data v nekonzistentním stavu, protože by chyběly aktualizace dat.

V případě úspěšné transakce navrátí služba kód *200 OK* a automobily, přičemž u vkládaných dat, je vyplněno pole **serverID**. Při chybě autorizace oznámí služba chybu kódem *403 Forbidden*.

Spotřeby

Tabulka 5.12 zobrazuje možné metody pro hromadnou práci se spotřebami. Jsou podporovány příkazy *HTTP GET* pro získání spotřeb novějších než specifikované datum. Dále pak *HTTP POST* pro hromadné vložení, aktualizaci a smazání dat. Výpis 5.7 zobrazuje formát XML souboru, v jakém se serializují jednotlivé spotřeby. Jak je zřejmé, jednotlivé spotřeby jsou uzavřeny v elementu **consumptions**.

Tabulka 5.12: Rozhraní pro spotřeby.

Rozhraní pro spotřeby	
Metoda	**URL**
GET	data/{user}/consumptions/{čas}
POST	data/{user}/consumptions

```
<?xml version="1.0" encoding="UTF-8"?>
<consumptions>
    <consumption>
        ...
    </consumption>
    ...
</consumptions>
```

Výpis 5.7: XML formát pro reprezentaci spotřeb.

Servis

Na výpisu 5.8 je vidět formát, v jakém služba posílá data a zabaluje jednotlivé servisní náklady do elementu **repairs**. Hromadné přístupové metody jsou prakticky stejné jako v předešlém případě a jsou shrnuty v tabulce 5.13.

Tabulka 5.13: Rozhraní pro servisní záznamy.

Rozhraní pro servisní záznamy	
Metoda	**URL**
GET	data/{user}/servises/{čas}
POST	data/{user}/servises

```
<?xml version="1.0" encoding="UTF-8"?>
<repairs>
    <repair>
        ...
    </repair>
    ...
</repairs>
```

Výpis 5.8: XML formát pro reprezentaci servisních záznamů.

Pravidelné náklady

Operace s pravidelnými náklady jsou opět stejné jako v předchozích případech. V tabulce 5.14 je zobrazen způsob práce s více pravidelnými náklady, ty jsou reprezentovány ve formátu specifikovaném na výpisu 5.9.

Tabulka 5.14: Získání pravidelných nákladů.

Získání pravidelných nákladů	
Metoda	**URL**
URL	data/{user}/regularexpenses/{čas}
URL	data/{user}/regularexpenses

```
<?xml version="1.0" encoding="UTF-8"?>
<regularExpenses>
    <regularExpense>
        ...
    </regularExpense>
    ...
</regularExpenses>
```

Výpis 5.9: XML formát pro pravidelných nákladů.

5.6 MVC

MVC, neboli *Model-View-Controller*, je velmi populární architektonický vzor, který rozděluje architekturu aplikace do tří nezávislých částí. Díky tomu jsou jakékoliv změny relativně jednoduše proveditelné a nevyžadují rozsáhlé změny kódu v závislých modulech.

Model – Odpovídá vrstvě domény, obsahuje objekty domény, což mohou být jakákoliv data, s kterými si přejeme pracovat. Součástí modelu je také aplikační logika (například různé agregační funkce nad daty).

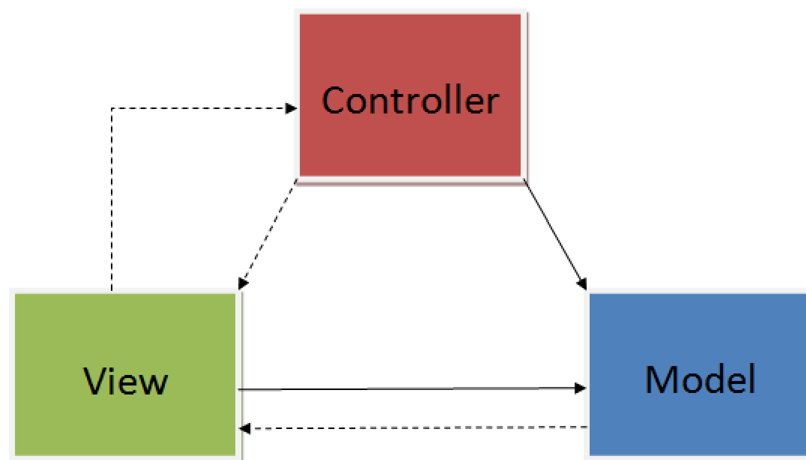
View – Zodpovídá za prezentaci modelu uživateli. Jedná se tedy o uživatelské rozhraní, které si je schopno získat stav z modelu.

Cotroller – Zpracovává a reaguje na události, což jsou typicky události vyvolané akcemi uživatele. *Controller* tedy významně provazuje funkčnost aplikace, protože reaguje na události uživatelského rozhraní a vyvolává případnou změnu modelu.

Prvky této architektury a vazby mezi nimi jsou znázorněny na obrázku 5.5.

Architektura *MVC* může být implementována různými způsoby, ovšem obecně fungují tyto principy:

Uživatel provede požadovanou akci v uživatelském rozhraní, které je reprezentováno částí *View*. Následně je *Controller* informován o této akci (v závislosti na použité technologii může být informován částí *View* – přerušovaná šipka značící možnou vazbu mezi *View* a *Controller*). V reakci na tuto událost vyvolá *Controller* změny v modelu, což je znázorněno plnou šipkou z *Controller* do *Model*. Logika modelu provede patřičné akce, například aktualizuje sumu ceny nebo například vloží nové objekty. Model následně informuje



Obrázek 5.5: MVC architektura.

o změně, což může být realizováno například pomocí návrhového vzoru *Observer*. Vzor *Observer* je použit, protože model nesmí mít v této architektuře přímou vazbu na zbylé dvě části, což je znázorněno přerušovanou šipkou z *Model* do *View*. Můžeme si povšimnout, že v *MVC* nejsou některé věci specifikovány – jako například persistence dat. Není to však chyba, tato specifika jsou ponechána pro řešení jinými modely architektur.

5.7 Architektura

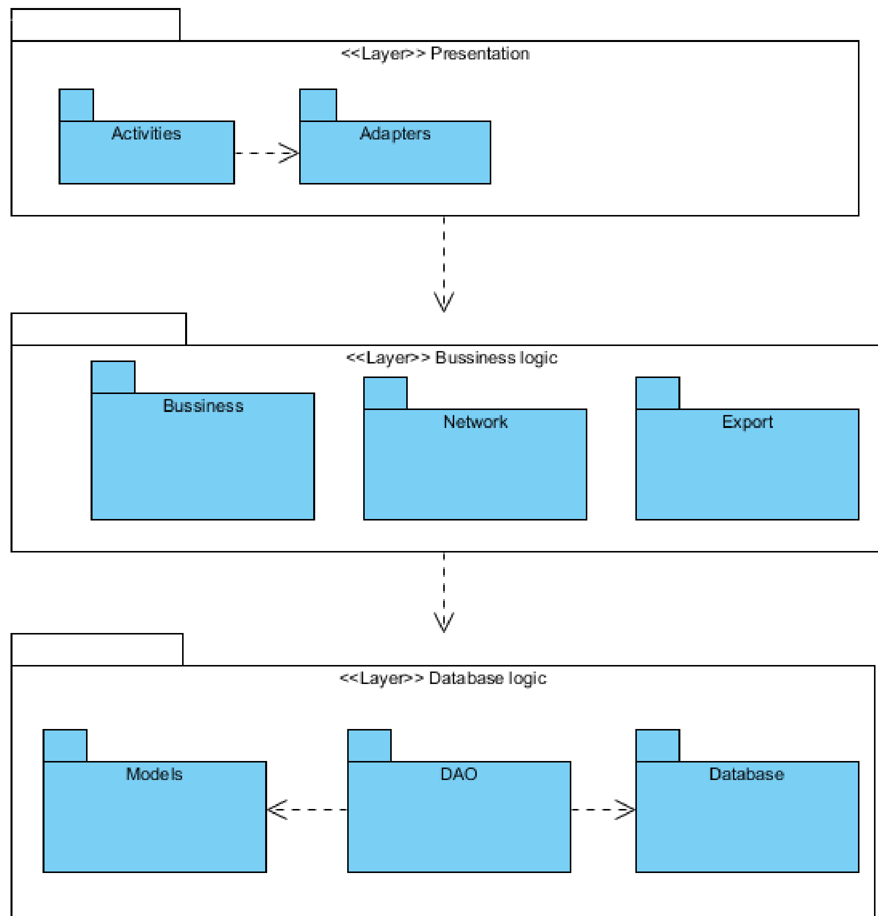
Aplikace je architektonicky rozdělena do tří vrstev, jejichž vzájemné vztahy jsou zobrazeny na diagramu zobrazeném na obrázku 5.6.

Prezentační vrstva se stará o prezentaci dat uživateli. Jsou zde dva hlavní balíčky, které obsahují aktivity a adaptéry. Aktivity komunikují s uživatelem a adaptéry slouží slouží pro efektivní zobrazení dat v seznamech, mřížkách případně expandovatelných seznamech. Tato vrstva závisí na vrstvě aplikační logiky, která je v obrázku popsána jako Bussiness logic.

V této vrstvě je obstarávána veškerá logika aplikace, od správy záznamů, přes síťovou komunikaci včetně podpory synchronizace, až po export dat pro komunikaci s jinými aplikacemi. Veškeré složitější operace jsou tedy prováděny zde.

Aby mohla být tato data spolehlivě ukládána, je zde ještě třetí vrstva, na které je vrstva aplikační logiky závislá. Tato třetí vrstva je pojmenována jako databázová logika a má na starosti persistenci dat. Persistence dat nemusí znamenat pouze ukládání dat do databáze, ale může se zde využívat obecně jakékoliv persistentní úložiště, které uznáme za vhodné. Bylo by tedy zcela akceptovatelné využít zde například serializaci do XML souborů, případně jakékoliv jiné formy. V našem případě ale byla samozřejmě využita databáze. V této nejnižší vrstvě jsou také obsaženy všechny modely, které databázová vrstva ukládá a se kterými pracují vyšší vrstvy.

Tato architektura má výhodu relativní izolovanosti jednotlivých částí a změna v jedné vrstvě neovlivní nijak markantně, případně vůbec, ostatní vrstvy. Pokud bychom se například rozhodli změnit datové úložiště, změny by se prováděly pouze v této vrstvě a nikde



Obrázek 5.6: Rozdělení aplikace do vrstev.

jinde. Rozhraní pro persistenci by zůstalo stejné. Stejně tak změny v uživatelském rozhraní neovlivní vrstvy nižší.

Kapitola 6

Implementace

6.1 MVC

V rámci návrhu v sekci 5.6 byl popsán architektonický vzor MVC. Jeho realizace na platformě Android má svá specifika.

Za *Model* je považována aplikační logika společně s daty. Aby mohl model informovat ostatní části, je využíváno návrhového vzoru *Observer*, který má v jazyku Java velmi dobrou podporu. Třídy, které chtějí informovat o svých změnách, což jsou v našem případě jednotlivé části *Model*, mají jako svého předchůdce ve stromu dědičnosti třídu `java.util.Observable`. Třídy, které chtějí být informovány o změnách proběhlých v *Model*, implementují komplementární rozhraní `java.util.Observer`. O události v *Model*, by měl být informován přímo *View*. To však není možné, protože v systému Android jsou *View* přímo XML soubory, které ze své povahy nejsou schopny reagovat na jakékoliv dynamické události. Aktivita je tedy zcela jistě *Controller*, protože obstarává události, jako jsou kliknutí na tlačítko, a následně říká modelu, co má změnit. Aktivita je ovšem částečně i *View*, protože musí reagovat na změny modelu a reflektovat je.

Následující ukázky na výpisech 6.1 a 6.2 zobrazují základní práci s `java.util.Observer` a `java.util.Observable` v kontextu vývoje pro systém Android.

```
public class MyActivity extends Activity implements java.util.Observer
{
    BussinessLogic log = new BussinessLogic();

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        // přidáme aktivitu jako pozorovatele
        log.addObserver(this);
    }

    @Override
    protected void onDestroy()
    {
        // ostranění aktivity ze seznamu pozorovatelů
        log.deleteObserver(this);
    }

    @Override
    public void update(Observable o, Object arg)
```

```
    {
        // Tato metoda je volána z~modelu,
        // signalizuje změnu a je nutno aktualizovat zobrazení modelu
    }
    ...
}
```

Výpis 6.1: Ukázka práce s třídou Observable a rozhraním Observer v rámci aktivity (*Controller*).

```
public class BussinessLogic extends java.util.Observable
{
    public void Add(int number)
    {
        //provedeni výpočtů a změn modelu
        ...
        // informování pozorovatelů o~změnách
        this.setChanged();
        this.notifyObservers();
    }
    ...
}
```

Výpis 6.2: Ukázka práce s třídou Observable v rámci aplikační logiky *Model*.

V metodě `onCreate` zobrazené na výpisu 6.1 je vidět, jak se aktivita představující *Controller* zapíše jako pozorovatel do aplikační logiky (*Model*). Následně se v metodě `onDestroy` z tohoto seznamu odepíše, aby se ji logika nepokoušela zasílat žádné další zprávy. Toto zapsání může provést, jelikož implementuje již dříve zmíněné rozhraní `java.util.Observer` obsahující metodu `update`, která je volána z *Model* pro informování o změnách. Aktivita tak ví, že má zahájit překreslení pohledu.

Metoda `Add` zobrazená na výpisu 6.2 je samozřejmě pouze demonstrativní. Můžeme si místo ní představit jakoukoliv metodu, která provádí změnu nad daty. Aplikační logika následné zavolá metodu `setChanged`, která nastaví příznak změny a metodu následně metodu `notifyObservers`, která informuje všechny pozorovatele o změně v datech. Obě tyto metody jsou zděděny z nadtřídy `java.util.Observable`.

Informace uvedené v rámci této sekce shrnuje následující výčet:

Model – Aplikační logika společně s daty je potomkem `java.util.Observable`, pro jednoduché informování jiných částí o změnách modelu.

View – Layout je popsaný popsaný pomocí XML souborů, které definují vzhled.

Cotroller – O změnách modelu by měl být informován přímo *View*, to však zde není možné, a tak musí na změny modelu reagovat Aktivita reprezentující *Controller*. Pro příjem informací o změnách implementuje rozhraní `java.util.Observer`.

6.2 Webová služba

V následujících podsekcích je popsána implementace, konfigurace a technologická stránka webové služby. V části 6.2.1 je popsáno běhové prostředí potřebné pro webovou službu. Následuje část 6.2.2, která popisuje konfiguraci serveru. V části 6.2.3 je popsáno, jak efektivně zabezpečit komunikaci se službou, přičemž jednotlivé implementační detaily služby jsou popsány v části 6.2.4.

6.2.1 Běhové prostředí

Webová služba byla vyvíjena a testována na serveru *Apache Tomcat* verze `6.0.14`. Jedná se o open source servlet kontejner, který implementuje technologie *Java Servlet* a *JavaServer Pages*. Je vyvíjen a distribuován pod *Apache License version 2*, což umožňuje jeho volné použití. Pro bližší informace o serveru *Apache Tomcat* je vhodné shlédnout odkaz v literatuře [27].

Pro implementaci RESTful webové služby, která byla zvolena jako nejvhodnější kandidát v rámci návrhu 5.4.3, byla použita referenční implementace standardu *JAX-RS (JSR 311)* [22] pojmenovaná *Jersey*. Bližší informace o *Jersey* jsou v literatuře [24].

6.2.2 Konfigurace a autentizace

Konfigurace prostředí *Tomcat* je prováděna pomocí více XML souborů. Hlavními jsou:

- web.xml
- tomcat_users.xml
- server.xml

Soubor **web.xml** je ve zkrácené podobě zobrazen na výpisu 6.3. Konfigurační soubor definuje nezbytné informace pro správné načtení webové služby. Všimněme si hlavně elementu `<init-param>`. V rámci jeho synovských elementů je definováno, že se o poskytnutí tříd, které mají být zpřístupněny, stará třída `org.carcosts.backend.MyApplication`.

```
<servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>
        com.sun.jersey.spi.container.servlet.ServletContainer
    </servlet-class>
    <init-param>
        <param-name>javax.ws.rs.Application</param-name>
        <param-value>org.carcosts.backend.MyApplication</param-value>
    </init-param>
    ...
```

Výpis 6.3: Popis souboru web.xml část načtení.

V rámci tohoto souboru je řešeno i nastavení autentizace, jehož výňatek je na výpisu 6.4. Uvnitř elementu `<security-constraint>` se nachází elementy `<web-resource-collection>` a `<auth-constraint>`. Element `<web-resource-collection>` obsahuje další podelementy, které definují, jaké zdroje vyžadují autentizaci, a také, pro jaké metody tato omezení platí. Jak je zřejmé, omezení platí pro metody *GET, POST, PUT, DELETE* mapující se v projektu na cestu `/rest/data/*`, přičemž `*` značí, že omezení platí pro všechny zdroje v daných a návazných umístěních. Přístup na dané umístění je zde povolen pouze osobám autentizovaným v roli `uzivatel`.

V elementu `<login-config>` je specifikováno, jakým způsobem se budou uživatelé autentizovat. v našem případě je použita hodnota `BASIC`, značící základní autentizaci *(Basic authentication)*. Uvnitř elementu `<security-role>` jsou definovány dostupné uživatelské role. V našem případě je zde uvedena výše zmíněná role `uzivatel`.

```
...
<security-constraint>
        <web-resource-collection>
```

```
            <web-resource-name>Security</web-resource-name>
            <url-pattern>/rest/data/*</url-pattern>
            <http-method>GET</http-method>
            <http-method>POST</http-method>
            <http-method>PUT</http-method>
            <http-method>DELETE</http-method>
        </web-resource-collection>
        <auth-constraint>
            <role-name>uzivatel</role-name>
        </auth-constraint>
...
<login-config>
    <auth-method>BASIC</auth-method>
</login-config>
<security-role>
    <role-name>uzivatel</role-name>
</security-role>
...
```

Výpis 6.4: Popis souboru web.xml část autentizace.

Soubor `tomcat-users.xml` zobrazený na výpise 6.5, je používán v základním nastavení *Tomcat serveru*. Je vhodný hlavně pro ulehčení vývoje. Obsahuje definice rolí, uživatelů a vztahů mezi nimi. Zásadní slabina tohoto souboru je v tom, že představuje pouze statickou konfiguraci. Není určen pro dynamickou editaci za běhu a nelze proto použít například pro registrace uživatelů a pro naše účely tak není vhodný.

```
<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users>
    <role rolename="tomcat"/>
    <user username="tomcat" password="tomcat" roles="tomcat"/>
</tomcat-users>
```

Výpis 6.5: Popis souboru tomcat-users.xml.

Pokud chceme, aby uživatelé mohli být registrováni dynamicky za běhu, běžné řešení je přes databázi. Musíme nakonfigurovat soubor `server.xml`. Je zde element `<Realm>`, jehož jednotlivé atributy specifikují v ukázce na výpisu 6.6 autentizační nastavení. *Realm* je databáze uživatelských jmen, hesel a jejich rolí.

```
<Realm className="org.apache.catalina.realm.JDBCRealm"
connectionName="realm_access" connectionPassword="realmpass"
connectionURL="jdbc:mysql://localhost:3306/carcosts"
digest="MD5" driverName="com.mysql.jdbc.Driver"
roleNameCol="role" userCredCol="Heslo" userNameCol="UzivatelskeJmeno"
userRoleTable="tomcat_uzivatel_role" userTable="Uzivatel"/>
```

Výpis 6.6: Popis souboru server.xml.

Atribut `connectionURL` specifikuje cílový počítač, port a databázi, která je v našem případě MySQL. Atributy `connectionName` a `connectionPassword` specifikují přístupové jméno a heslo k databázi. V databázi tedy musí být uživatel minimálně s právy čtení. Vytvoření takového uživatele ilustruje výpis 6.7.

```
CREATE USER 'realm_access'@'localhost' IDENTIFIED BY 'realmpass';
GRANT SELECT ON carcosts.* TO realm_access@localhost;
```

Výpis 6.7: Vytvoření uživatele pro přístup k MySQL databázi.

Atribut `digest` specifikuje typ hashovací funkce pro práci s hesly, přičemž v našem případě se jedná o hashovací funkci *MD5*. Zbylé atributy definují, v jakých tabulkách a

sloupcích se nachází uživatelské jména, hesla a jaké jsou uživatelské role. Tyto hodnoty se musí shodovat s návrhem databáze. Tabulka, která drží údaje o uživatelských rolích, se jmenuje `tomcat_uzivatel`. Uživatelé jsou spravování v tabulce `Uzivatel`, přičemž uživatelská jména jsou ve sloupci `UzivatelskeJmeno` a hesla jsou ve sloupci `Heslo`. Toto nastavení umožňuje serveru autentizovat uživatele vůči databázi. Tento způsob autentizace se nazývá *container managed security* a využívá *JDBC Realm*, což je jedna ze standardních implementací *Realm* využívají pro přístup do databáze *JDBC* ovladač.

6.2.3 Zabezpečení komunikace

Komunikace probíhá standardně přes nezabezpečený protokol HTTP. Pokud chceme zabezpečit přenos dat, je potřeba využít místo protokolu HTTP protokol HTTPS. V rámci těchto úprav musí administrátor serveru povolit standardně na portu 443 přijímání zabezpečeného připojení. Standardně se využívá šifrování pomocí SSL nebo TLS. Tímto způsobem je nejefektivněji dosaženo zabezpečení komunikace.

6.2.4 Implementační detaily

Jak bylo uvedeno v předchozí části této práce 6.2.1, pro práci s *JAX-RS (JSR 311)* byla použita referenční implementace tohoto standardu s názvem *Jersey*. Použití je relativně přímočaré a ve značné míře využívá prvků jazyka Java zvaných anotace. Na výpisu 6.8 je zobrazena ukázka použití, která bude nyní popsána.

```
@Path("/data/{user}/consumption")
public class SingleConsumption extends BaseResponse
{
    @POST
    @Produces(MediaType.APPLICATION_XML)
    @Consumes(MediaType.APPLICATION_XML)
    public Response post(@Context HttpHeaders ui,
        @PathParam ("user") String user,  Consumption cons)
    {
    ...
```

Výpis 6.8: Ukázka použití JAX-WS – Jersey.

Anotace `@Path` nad deklarací třídy určuje, na jaké adrese bude tato třída dostupná. Jak je vidět na ukázce, třída `SingleConsumption` je dostupná na adrese `/data/user/consumption`, kde název ve složených závorkách značí parametrizovatelnou cestu. Hodnota, která je zde zadána, je přejata ve funkci obsluhující daný požadavek pomocí anotace `@PathParam ("user") String user`. Všimněme si, že názvy v anotaci třídy a metody se pro správnou funkci předání parametrů musí shodovat. Pomocí anotace `@POST`, `@PUT`, `@DELETE`, `@GET` nad metodou specifikujeme, jaký HTTP požadavek metoda obsluhuje. Anotací `@Context` získáváme kontextové informace o právě prováděném dotazu, jako například v tomto případě HTTP hlavičku. Pokud chceme specifikovat, jaké vstupní formáty naše třídní metody přijímají a produkují, použijeme anotaci `@Produces` a `@Consumes`.

6.3 Dotazy na webovou službu ze systému Android

6.3.1 Nezabezpečené spojení

Jak již bylo řečeno, RESTful webová služba je založena na protokolu HTTP. Nejvhodnější je proto pro komunikaci použít zabudované HTTP klienty. Na systému Android je imple-

mentována třída `DefaultHttpClient` obsažená v balíčku `org.apache.http.impl.client`. Tato třída poskytuje plnou podporu pro provádění potřebných dotazů. Na výpisu 6.9 je uveden příklad vykonání dotazu na službu. Nejdříve je zvolena metoda, kterou se chceme dotazovat. To je v tomto případě *POST*, proto je zvolen patřičný objekt `HttpPost`. Pokud bychom chtěli použít jinou metodu, stačí pouze zaměnit tento objekt za jiný (například `HttpPut`).

```
//provádíme Http Post dotaz
HttpPost query = new HttpPost("http://server.cz/CarcostBackend/rest/data/car");

//nastavení typu obsahu
query.setHeader(HTTP.CONTENT_TYPE, "application/xml");

// vložení přihlašovacích údajů do dotazu
setAuthentification(usr);

// vložení samotného obsahu dotazu
StringEntity se = new StringEntity(xml, HTTP.UTF_8);
se.setContentType(new BasicHeader(HTTP.CONTENT_TYPE,
                                  "application/xml"));

//nastavení kódování dotazu
se.setContentEncoding(HTTP.UTF_8);
//vykonání dotazu
HttpResponse response = httpclient.execute(query);
//získání návratového kódu
int statusCode = response.getStatusLine().getStatusCode();
```

Výpis 6.9: Dotaz na webovou službu (vkládání).

O dodání přihlašovacích údajů se stará metoda `setAuthentication`, jejíž obsah je probrán níže ve výpisu 6.10. Po zdárném nastavení hlavičky dotazu dodáme samotný obsah dotazu, v tomto případě se jedná o entitu `StringEntity`, které předáme serializovaná data ve správném kódování, což je v našem případě UTF-8, ve kterém očekává a odesílá data i webová služba. Samotný akt vykonání dotazu a získání odpovědi provádí metoda `execute`, která zároveň vrací odpověď od serveru. Z této odpovědi lze jednoduše získat jak HTTP stavové kódy, tak data výsledná, která jsou vrácena. Tato celá odpověď je zapouzdřena v objektu `HTTPResponse`.

Aby mohla služba klienta správně autentizovat, musí klient poskytnout své přihlašovací údaje. Tyto údaje jsou přenášeny v rámci HTTP hlavičky každého dotazu. Jejich poskytnutí na systému Android ukazuje výpis 6.10. Pokud se uživatel úspěšně autentizuje, vykoná služba na základě jeho identity požadované akce a vrátí mu výsledek celé operace.

Třída `DefaultHttpClient` umožňuje získat instanci třídy `CredentialProvider`. Tato třída umožňuje nastavit, případně získat přihlašovací údaje. Společně s přihlašovacími údaji se nastavuje takzvaný autentikační prostor (*authentication scope*). Tento prostor omezuje použití daných přihlašovacích údajů na specifický port a cílového hostitele. V našem případě však použití přihlašovacích údajů nijak neomezujeme, a použijí se tak pro jakéhokoliv hostitele a port.

```
// získání CP
CredentialsProvider cp =  httpclient.getCredentialsProvider();

// vložení přihlašovacích údajů
UsernamePassword up =  new UsernamePasswordCredentials("jmeno", "heslo");
```

```
// nastavení přihlašovacích údajů
cp.setCredentials(new AuthScope(null, AuthScope.ANY_PORT), up);
```

Výpis 6.10: Poskytnutí přihlašovacích údajů

6.3.2 Zabezpečené spojení

V případě, že je na webovém serveru, na kterém běží služba, aktivováno zabezpečené připojení pomocí SSL, musí s touto službou umožňovat zabezpečeně komunikovat i náš klient. Třída `DefaultHttpClient` podporuje zabezpečené přenosy a jednotlivá volání zůstávají stejná jako při nezabezpečeném přenosu. Je však potřeba poskytnout certifikát podepsaný certifikační autoritou, které důvěřuje OS Android.

Pokud není použit certifikát podepsaný důvěryhodnou certifikační autoritou, je při kontaktování serveru vyvolána výjimka `javax.net.ssl.SSLException` obsahující zprávu *Not trusted server certificate*. Pro předejití vyvolání této výjimky existují tři způsoby:

- Důvěryhodný certifikát.
- Povolení používání i nedůvěryhodných certifikátů.
- Vložení našeho certifikátu do skladu důvěryhodných certifikátů v telefonu.

První a nejlepší možností pro provozování zabezpečené komunikace je použití certifikátu vydaného důvěryhodnou certifikační autoritou. Tato volba bude fungovat na všech mobilních zařízeních a také je nejbezpečnější.

Druhou možností je akceptování i nedůvěryhodných certifikátů, to znamená certifikátů nepodepsaných důvěryhodnou certifikačním autoritou. V aplikaci se musí implementovat nový manažer důvěryhodných certifikátů. Tento manažer pak certifikáty nekontroluje a všechny akceptuje bez jakéhokoliv ověření. Tato možnost však není bezpečná, a proto není v aplikaci použita. Akceptování všech certifikátů zavádí vlastně bezpečnostní slabinu do aplikace, a proto se silně nedoporučuje používat tento postup.

Posledním způsobem je manuální vložení námi vydaného certifikátu do skladu důvěryhodných certifikátů. Nejvhodnější je použít *BKS*[1], který je v základu podporován všemi telefony s OS Android. Certifikát v tomto formátu se pak musí umístit na mobilní zařízení. Aplikace musí vytvořit vlastního HTTP klienta (zděděného od `DefaultHttpClient`), který bude při ověřování certifikátů brát v potaz tento nově vytvořený certifikát v *BKS* formátu. Tento způsob ovšem vyžaduje naimportování každého takovéhoto certifikátu do daného mobilního zařízení. Tato možnost je tedy vhodná hlavně pro testovací účely s menším počtem zařízení.

6.4 Serializace objektů

Pro efektivní přenos objektů je použita serializace jednotlivých tříd do XML. Jak na straně webové služby, tak na straně mobilního klienta, má tato serializace vzhledem k použití různých serializérů svá specifika.

[1]Bouncy Castle Keystore Format.

6.4.1 Webová služba

Serializace na serverové straně probíhá automaticky. Stačí pouze použít daný objekt jako parametr dané metody a bude automaticky předán při jejím volání, případně navrátit instanci třídy jako výsledek a bude také automaticky serializována. Jedinou podmínkou je, že objekt musí být serializovatelný. Aby byl objekt serializovatelný a webová služba jej takto mohla automaticky používat, musí splňovat následující dvě podmínky:

- Implementace rozhraní `ISerializable`.
- Anotace `@XmlRootElement`.

Po splnění výše uvedených dvou podmínek funguje serializace bez problémů. Je však nutno upravit formu, v jaké se předává datum.

Jelikož každý serializér může předávat datum v trochu jiné podobě, je potřeba tento formát standardizovat na hodnoty uvedené v návrhu. Na straně serveru se toho dosáhne ve dvou krocích. Nejdříve je potřeba vytvořit třídu, která bude potomkem třídy `XmlAdapter`, v našem případě jsme tuto třídu nazvali `MyJaxDateAdapter`. Tato třída umožňuje pro datový typ dodat jeho vlastní podobu pro převody do XML. Její stručná podoba je vidět na výpisu 6.11. Obsahuje nadefinovaný formát pro datum a čas, dále pak metodu `marschal`, která se stará o převod na textovou formu, a metodu `unmarschal`, která provádí převod opačným směrem.

U jednotlivých vlastností typu `java.uti.Date`, které chceme serializovat následně, nastavíme anotaci `@XmlJavaTypeAdapter(MyJaxDateAdapter.class)`. Tímto je zajištěn úspěšný převod.

```
// objekt umožňující konverzi
public class MyJaxDateAdapter  extends XmlAdapter<String,Date>
{
    public static final String DATEFORMAT="yyyy-MM-dd'T'HH:mm:ss.SSSZ";

    public Date unmarshal(String s) {...}
    public String marshal(Date d) {...}
}

// tato vlastnost objektu se má serializovat pomocí výše definovaného objektu
@XmlJavaTypeAdapter(MyJaxDateAdapter.class)
public void setLastChange(Date lastChange)
{
    this.lastChange = lastChange;
}
```

Výpis 6.11: Vlastní formát pro reprezentaci datumu v XML.

6.4.2 Mobilní zařízení

Na mobilním zařízení s operačním systémem Android nejsou dostupné v základu zcela všechny třídy a knihovny dostupné v JRE[2]. Pro práci s XML dokumenty je tak dostupný například SAX[3] nebo DOM[4]. V základu však chybí XML serializér, který by umožňoval automaticky serializovat objekty bez nutnosti jakýchkoliv manuálních definic. Ruční převod

[2]Java Runtime Environment.
[3]Java's Simple API for XML.
[4]Document Object Model.

objektů z XML a jejich následný zpětný převod z XML by byl značně komplikovaný a vedl by k zanesení řady potenciálních chyb. Tento přístup by byl rovněž značně nepružný z hlediska další rozšiřitelnosti aplikace. Pokud bychom se rozhodli přidat do třídy byť jedinou vlastnost, museli bychom upravit obě metody pro převod mezi reprezentacemi.

Z důvodu absence jakékoliv takovéto třídy a její nesporné praktičnosti byla použita knihovna XStream [3] pro zprostředkování automatické serializace. Nebyla však použita její plnohodnotná edice, ale její verze pro OS Android distribuovaná pod BSD licencí [6]. Knihovna XStream je velmi výkonná a má jednoduché použití. Dovoluje rovněž velmi rozsáhle konfigurovat způsob serializace. Je možno nastavit vlastní konventory hodnot i vlastní názvy pro serializované objekty. Konventory hodnot je výhodné využít například pro vynucení použití vlastního formátu pro datum a čas. Příklad využití konventoru je na výpisu 6.12. Vlastní názvy se dají použít nejen pro samotné objekty, ale také například pro pole objektů.

```
// registrace konventoru do objektu třídy XStream
xstream.registerConverter(new SimpleDateConverter());

// třída pro konvertování data a času
public class SimpleDateConverter extends AbstractSingleValueConverter
{
    public static final java.text.SimpleDateFormat fmt =
          new java.text.SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSZ");

    @Override
    public boolean canConvert(Class type)
    ...
    @Override
    public Object fromString(String str)
    ...
    @Override
    public String toString(Object obj)
    ...
}
```

Výpis 6.12: Využití konventorů pro vlastní reprezentaci data a času.

Postup je podobný jako v postupu řešení obdobného problému na serverové straně. Vytvoří se třída `SimpleDateConverter`, jejímž předkem je abstraktní třída `AbstractSingleValueConverter`. V rámci implementace se naimplementují metody `canConvert`, `fromString`, `toString`. Tyto metody se starají o kompletní převod data na řetězec a zpět. Metoda `canConvert` určuje, jaké datové typy umožňuje třída konvertovat.

6.5 Změny konfigurace a zpracovávání úkonů na pozadí

Aplikace se musí vypořádat s různými událostmi. Mezi nejčastější změnu konfigurace patří otočení displeje. Aplikace musí uživateli aktivitu zobrazit ve stejném stavu, jako byla před změnou konfigurace. Při těchto změnách se totiž celá aktivita zničí a znovu vytvoří s novou konfigurací.

Značnou část tohoto úkolu plní systém Android automaticky a například obsah různých uživatelem vyplněných polí si pamatuje sám. Ulehčuje však vývojáři práci jen při použití správných technik. Pokud například vývojář vytvoří dialog ve vlastní režii, to znamená vytvoří instanci, nastaví vlastnosti a zobrazí dialog voláním metody `show`, musí si také sám hlídat změny konfigurace a znovu obnovit tento dialog, pokud byl zobrazen před změ-

nou konfigurace. Pokud však použije pro zobrazení dialogu metodu aktivity `showDialog`, aktivita sama znovu spustí dialog po změně konfigurace.

Pokud aplikace v některých částech vykonává časově náročnou práci, je vhodné dát tuto činnost do vlákna, které ji bude zpracovávat na pozadí a informovat uživatele až o výsledku dané operace. Zde je ovšem problematická manipulace s uživatelským rozhraním, protože s ním může být manipulováno pouze z hlavního vlákna. K tomuto účelu jsou na platformě Android tzv. *Handlery*, pomoci kterých lze bezpečně poslat zprávu, která bude zpracována v hlavním vlákně.

Další možností je použití objektu `AsyncTask`, který slouží jako podpora pro vykonávání práce na pozadí. Tato třída má čtyři hlavní metody, které jsou obvykle implementovány v odvozených třídách. Jsou jimi:

onPreExecute – Pracuje ve vlákně, které může dělat zásahy do uživatelského rozhraní, obvykle se zde například zobrazí dialog.

doInBackground – Je vykonáváno ve vlákně na pozadí.

onPostExecute – Vykonáváno ve vlákně, kde běží uživatelské rozhraní. Obvykle se zde provádí zrušení dialogu, kterým byl uživatel informován o prováděných akcích.

onProgressUpdate – Slouží pro distribuci informace o aktuálním stavu úkonu uživateli.

Je zde ovšem jeden značný problém, pokud změníme konfiguraci (během vykonávání úkonu na pozadí), objekt `AsyncTask` bude mít referenci na aktivitu, která jej spustila, tato aktivita však byla zničena a byla vytvořena nová aktivita. Pokud se `AsyncTask` pokusí schovat dialog indikující postup, nepodaří se mu to, protože bude pracovat s již neplatnou aktivitou.

Prvním možností, jak toto řešit, je obejít zničení a znovu vytvoření aktivity. V manifestu aplikace je potřeba deklarovat zachytávání změn konfigurace a v rámci aktivity samotné je potřeba manuálně obstarat změnu zobrazení. Tento postup je demonstrován na výpisech 6.13 a 6.14.

```
<activity  android:name=".AsyncTaskSimpleConfigured"
           android:configChanges="keyboardHidden|orientation"
           android:label="AsyncTaskSimpleConfigured"/>
```

Výpis 6.13: Zachycení změny konfigurace v manifestu.

```
@Override
public void onConfigurationChanged(Configuration newConfig)
{
  super.onConfigurationChanged(newConfig);
  setContentView(R.layout.myLayout);
}
```

Výpis 6.14: Zachycené změny konfigurace v aktivitě.

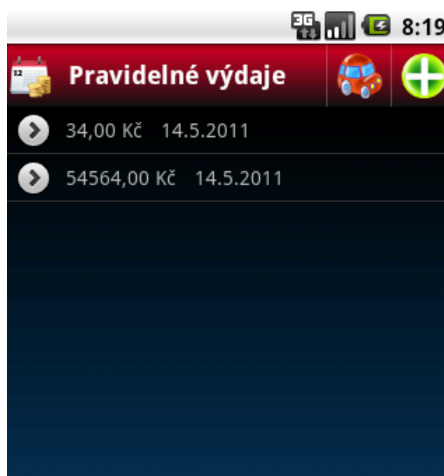
Možnost použitá v aplikaci je ale odlišná. Objekt vykonávající akci na pozadí je rozšířen o metody pro přiřazení aktuálně platné aktivity, přičemž tato metoda je volána po vytvoření nové aktivity. Abychom měli v nově vytvořené aktivitě instanci běžícího úkonu na pozadí, je potřeba využít další metody aktivity. Těsně před zničením aktivity je volána metoda `onRetainNonConfigurationInstance`. V této metodě vrátíme náš objekt třídy `AsyncTask` a vyzvedneme jej nazpět v rámci nového vytváření aktivity pomocí metody `getLastNonConfigurationInstance`. Pokud získáme tento objekt, nastavíme mu odkaz na instanci nové aktivity, a ten tak informuje o průběhu činnosti již nově vytvořenou aktivitu.

6.6 Prvky uživatelského rozhraní aplikace

Aplikace používá různé zajímavé prvky GUI pro interakci s uživatelem. Tyto prvky jsou relativně nestandardní a budou nyní popsány a předvedeny.

6.6.1 Action bar

V aplikaci byl použit tento stále populárnější prvek rozšiřující se mezi novějšími aplikacemi. Byl prvně představen jako jedna z nově vzniklých návrhových strategií pro vývoj GUI v rámci oficiálního klienta sítě sociální sítě Twitter[5]. Tento takzvaný Action bar je zobrazen na obrázku 6.1. Jedná se o horní lištu zobrazenou v rámci aktivity, kde má uživatel poskytnuty všechny potřebné údaje a nejpoužívanější akce. Action bar je použit ve všech aktivitách, které byly implementovány v rámci této aplikace. Pro jeho implementaci nebyly použity žádné knihovny třetích stran a je popsán v XML pouze za pomoci layoutů, stylů, hodnot a rozměrů.



Obrázek 6.1: Ukázka Action bar.

Obrázek 6.1 zobrazuje aktivitu reprezentující přehledový dialog pro pravidelné náklady. V horní části aktivity je zobrazen Action bar. V jeho levé části je ikona reprezentující pravidelné výdaje společně s textem informující uživatele o tom, kde se nachází. V pravé části jsou dostupné akce, které může uživatel provést. V tomto případě první ikona znázorňuje možnost změny automobilu, pro který jsou data v přehledu vedena. Ikona v pravém horním rohu pak umožňuje uživateli přidat nový záznam o pravidelných výdajích. Všechny nejčastější akce tak jsou uživateli dostupné na jedno kliknutí.

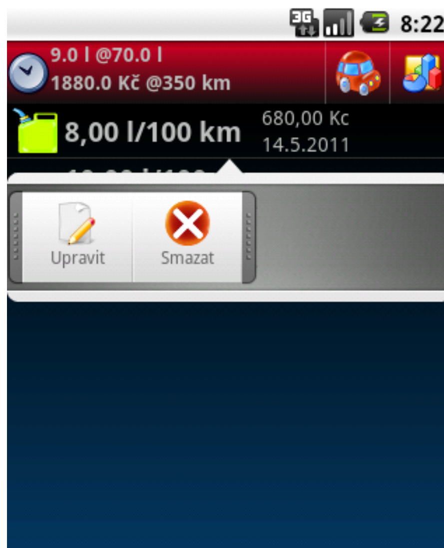
6.6.2 Quick Actions

Quick Actions[6] je také jeden z nových prvků uživatelského rozhraní poprvé představen v oficiálním klientu sítě Twitter.

Tyto rychlé volby byly použity jako knihovna, které je distribuována pod New BSD licencí. Jedná se o rychle aktivované okno vyvolané akcí nějakého elementu na displeji. Jedno

[5]http://android-developers.blogspot.com/2010/05/twitter-for-android-closer-look-at.html
[6]http://www.londatiga.net/it/how-to-create-quickaction-dialog-in-android

z jeho doporučených nasazení je například pro obsluhu dlouhých kliknutí na různých formách přehledových seznamů. V následně zobrazených volbách jsou pak nabídnuty operace s danou položkou seznamu. Na ukázce z obrázku 6.2 je vidět, že tato nabídka rychlých akcí nabízí úpravu a smazání dané položky.



Obrázek 6.2: Ukázka zadávání parametrů jízdy.

6.6.3 Vlastní prvek

V rámci vývoje aplikace byl navržen i vlastní prvek uživatelského rozhraní. Jedná se o jednoduchou lištu pro zadávání stylu a podmínek jízdy pro dané tankování. Použití prvku je intuitivní, vývojář pouze vloží do XML layoutu prvek `<org.carcosts.activities.RideStyleView/>`. Uživatel může jednoduchým kliknutím na ikony zadat parametry jízdy, které jsou nabízeny v závislosti na vozidlu. Ukázka tohoto prvku je na obrázku 6.3.

Obrázek 6.3: Ukázka zadávací lišty.

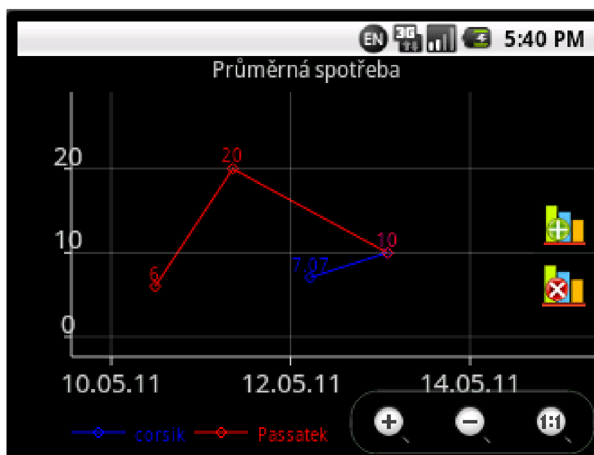
6.7 Grafy

Pro grafickou reprezentaci dat byla použita knihovna AChartEngine[7]. Tato knihovna je distribuována pod licencí Apache 2.0 a její nejnovější verze 0.6 byla vydána v lednu 2011.

Tato knihovna podporuje tvorbu velkého množství grafů, pro které programátorovi poskytuje velké množství konfiguračních možností. Graf může zobrazit přímo jako aktivitu, kterou mu dodá `ChartFactory`. Pokud si chce programátor tyto grafy upravit, může je

[7]http://www.achartengine.org/

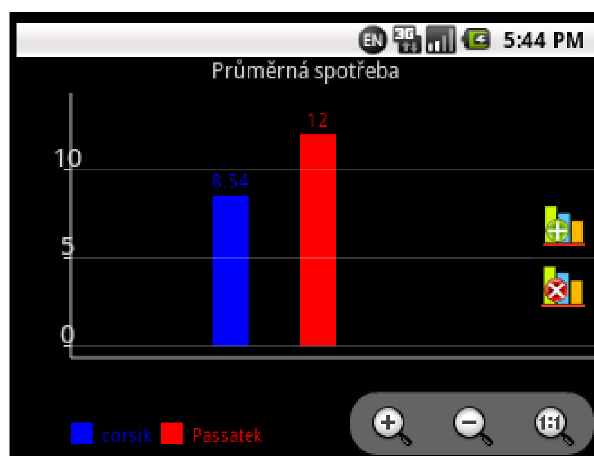
získat jako samostatný objekt třídy `GraphicalView`. Ten pak můžeme jednoduše zakomponovat do layoutu jakékoliv námi spravované aktivity.



Obrázek 6.4: Ukázka spojnicového grafu.

Na ukázce na obrázku 6.4 je vidět, že pro zobrazení průměrné spotřeby jsou zde pro jednotlivé automobily použity spojnicové grafy. Každá spojnice značí jednotlivé spotřeby pro automobil v časovém horizontu. Data tankování jsou zobrazena na ose X a hodnoty průměrné spotřeby jsou vyneseny na ose Y. Stejně tak tomu je na obrázku 6.5, zde jsou ovšem použity grafy sloupcové, které reprezentují průměrnou spotřebu jednotlivých automobilů. Můžeme tak porovnat průměrné hodnoty jednotlivých automobilů.

Do prostředí s grafem byly přidány dvě ikony, které umožňují přidání a odebrání dat pro grafové podklady daného automobilu.



Obrázek 6.5: Ukázka sloupcového grafu.

6.8 Práce se záznamy spotřeb

Záznamy spotřeb jsou uloženy stejně jako ostatní záznamy o nákladech v databázi ve vlastní tabulce. Oproti jiným nákladům však mají jedno specifikum, a to takové, že jednotlivé řádky se navzájem ovlivňují. U jiných záznamů (například servisních), můžeme libovolně mazat, přidávat a upravovat, protože každý záznam je samostatný a neovlivňuje ostatní.

U záznamů spotřeb tomu tak naneštěstí není. Když uživatel zadá první tankování, ještě není vypočítána průměrná spotřeba, nejsou totiž známy všechny potřebné údaje pro výpočet průměrné spotřeby. Tyto údaje jsou k dispozici až se zadáním dalšího záznamu spotřeby, protože teprve s jeho vyplněním program ví, jakou ujel řidič vzdálenost mezi tankováními a kolik na to spotřeboval pohonných hmot. Velmi důležitým údajem pro tyto výpočty je v případě prvního a částečného tankování hodnota množství pohonných hmot v nádrži před tankováním. Díky tomuto údaji může vyvinutá aplikace spočítat průměrnou spotřebu přesně a ihned, na rozdíl od jiných řešení dostupných na trhu. Díky tomuto způsobu výpočtu průměrné spotřeby jsou vlastně jednotlivé spotřeby zobrazovány zpětně, protože u záznamu s novým tankováním je zobrazena průměrná spotřeba, která mohla být dopočítána až díky tomuto záznamu a zobrazuje vlastně průměrnou spotřebu pro minulé tankování.

Závislost jednotlivých záznamů o tankováních klade poměrně vysoké nároky na aplikační logiku. Každá změna, přidání či smazání záznamu, vede kvůli závislostem k automatickému přepočítání návazných spotřeb záznamu.

Pokud přidáme záznam, aplikace jej zařadí na správné místo v závislosti na hodnotě odometru. Pokud je přidán na konec jako nejnovější záznam, je pouze spočítána (s využitím dat předchozího záznamu) průměrná spotřeba a ujetá vzdálenost.

Když přidáme záznam mezi dva jiné záznamy, chová se aplikace stejně jako v předchozím případě, avšak přepočítá i návazný záznam, protože jeho hodnoty musejí být spočítány v referenci na nově platný předcházející záznam.

U úpravy záznamů se musí sledovat, jestli se nezměnila hodnota odometru, a pokud ano, je nutno počítat s tím, že záznam mohl úplně změnit své umístění a také typ tankování. Tím pádem musí být kromě měněného záznamu a jeho sousedů přepočítány i záznamy v původní pozici. Stejně tak u mazání záznamů se musí přepočítávat následující záznam, protože se mu změnil jeho předchůdce.

Při těchto operacích může nastat situace, že uživatel zadá údaje, které se neshodují s ostatními. Pokud by například zadal, že při prvním tankování měl v nádrži 5 litrů PHM a natankoval 10 litrů, měl by tedy logicky v nádrži 15 litrů pohonných hmot. Předpokládejme, že automobil má zadanou kapacitu nádrže 50 litrů. Pokud by uživatel jako následné tankování zadal, že načerpal do plné nádrže 10 litrů PHM, dostáváme se do sporu. Jednoduchým výpočtem bychom došli k tomu, že na danou ujetou vzdálenost spotřeboval -25 litrů paliva, což je samozřejmě nesmysl. Aplikace na toto myslí, a pokud jsou data takto v nepořádku, převede je do stavu neplatných dat a uživatel je má v přehledu graficky zvýrazněny.

6.9 Spuštění synchronizace

Aby měl telefon neustále synchronizovaná data, je potřeba tuto synchronizaci spouštět. Toto spuštění může být provedeno jak manuálně volbou z menu, tak automaticky po zadaném časovém intervalu. Pokud si uživatel přeje, je možno provádět automatickou synchronizaci po určitém počtu hodin. K tomuto účelu byla použita třída `AlarmManager`. Tato

třída umožňuje aplikaci naplánovat akci na nějaký čas v budoucnu. Příklad na výpisu 6.15 prezentuje nastavení jednorázové akce, však může být i opakovaná.

```
// Získáním aktuálního času a přičtení hodin
Calendar cal = Calendar.getInstance();
cal.add(Calendar.HOUR, hours);

// získání service a nastavení alarmu
AlarmManager am = (AlarmManager) getSystemService(ALARM_SERVICE);
am.set(AlarmManager.RTC_WAKEUP, cal.getTimeInMillis(), getPending());
```

Výpis 6.15: Ukázka práce s alarmy.

Alarm je zachycen v takzvaném *Broadcast receiver*, zde je zpráva přijata a zpracována. Pokud aplikace neběží, je příjmem této zprávy automaticky spuštěna.

Kapitola 7

Budoucí rozšíření

V rámci této kapitoly jsou popsána možná budoucí rozšíření vyvíjené aplikace. Tato rozšíření jsou jak na straně síťové komunikace, tak na straně samotné mobilní aplikace.

7.1 Webový portál

Aplikace by se do budoucna mohla rozšířit o webový portál. Uživateli by poskytoval stejnou funkčnost jako mobilní zařízení. Vstupní zařízení na osobních počítačích mohou být pro někoho stále pohodlnější a rychlejší na zadávání dat než mobilní aplikace. Tento internetový portál by komunikoval s webovou službou a přistupoval k jejím datům, díky tomu by tato data byla jednoduše dostupná na mobilních zařízeních.

Mohl by sloužit ať už jako otevřený webový portál nebo běžet pouze v intranetu pro potřeby správy menších firem.

Portál může být realizovaný prakticky v kterémkoliv programovacím jazyce, jediným požadavkem na něj kladeným je schopnost komunikace s RESTful webovou službou.

7.2 C2DM

Android Cloud to Device Messaging je technologie vyvinutá firmou Google, která pomáhá vývojářům přenášet data ze serverů do jejich aplikací.

Služba představuje jednoduchý způsob, kterým může server předat mobilní aplikaci informaci, například aby jej kontaktovala přímo. *C2DM* obstarává všechny činnosti ukládání a doručování zpráv, pokud například mobilní zařízení, kterému se server snaží odeslat zprávu, není dostupné, je zpráva uložena do fronty zpráv čekající na doručení. Zpráva je doručena, jakmile zařízení znovu získá připojení k síti internet. Služba nicméně nezaručuje bezchybné doručování zpráv, je tedy vhodná pro distribuci upozornění na nastalé situace a nehodí se například pro přenášení důležitých dat.

Služba je v současné době spuštěna jako součást laboratoří Google. Pro provoz je potřeba mít svůj Google účet a telefon s operačním systémem Android verze 2.2 a vyšší. Pokud bychom tedy chtěli integrovat tuto funkčnost do naší aplikace, museli bychom zvýšit minimální verzi vyžadovaného SDK na hodnotu odpovídající verzi 2.2.

Webová služba a mobilní zařízení by se rozšířily o schopnost práce s *C2DM*. Pokud by uživatel přes webové rozhraní navrhované v bodě 7.1 zadal a provedl změnu dat, webová služba by vyslala zprávu mobilnímu zařízení pomocí *C2DM*. Toto mobilní zařízení by následně tuto zprávu dostalo a buď by informovalo uživatele o této skutečnosti, nebo by přímo

začalo se synchronizací. Více informací o *C2DM* lze nalézt v literatuře [4].

7.3 Plánování údržeb

Aplikace by mohla obsahovat jednoduchý připomínkový alarm, který by byl dlouhodobého charakteru. Uživateli by sloužil pro plánovaní údržbových informací ohledně automobilu, mohly by se zde zadávat upomínky pro návštěvu technické kontroly nebo měření emisí. Tyto alarmy by mohly být informovat uživatele pomocí notifikačních zpráv. Následně by se mohly integrovat s dalšími službami firmy Google jako kalendář, který by obstarával notifikaci uživateli místo aplikace.

7.4 Synchronizace obrázků

Aplikace v současné podobě umožňuje přiřazení obrázku každému automobilu. Tyto obrázky se načítají ve zmenšené podobě z datového skladu na mobilním zařízení a v informacích uložených o automobilu se uchovává cesta k tomuto obrázku. V rámci synchronizačního procesu se v současné podobě práce nesynchronizují jednotlivé obrázky automobilů.

Jako vhodné řešení ukládání se jeví uložení obrázku přímo do databáze, přičemž zvolená MySQL databáze tuto funkčnost přímo podporuje.

Kapitola 8

Závěr

Tato diplomová práce se zabývá správou nákladů pro osobní automobil. Během návrhu a tvorby aplikace bylo třeba nastudovat rozsáhlé množství materiálů zabývajících se různými technologiemi, prozkoumat a pochopit architekturu systému Android, stejně tak jako principy návrhu aplikací pro tento systém.

Návrh a tvorba uživatelského rozhraní je v systému Android značně odlišná od tradičního přístupu využívajícího tvorby objektů přímo ve zdrojovém kódu, jelikož je jeho tvorba založena na popisu pomocí XML souborů. Tento nový pohled na tvorbu GUI byl pro mne velmi přínosný. Naučil jsem se také používat rozličné nástroje SDK, jako *ADT plugin* a *sqlite3*. V rámci návrhu aplikace byly zvoleny dva databázové systémy, které bude aplikace využívat. Na straně mobilního zařízení je použita SQLite databáze, protože je v základu přítomna na každém zařízení s OS Android. Její použití proto byla nejrozumnější volba. Na vzdáleném serveru je také databázový systém, nejedná se ovšem o SQLite databázi, ale o databázový systém MySQL. Ten byl zvolen, protože nejvíce vyhovoval požadavkům kladeným na serverovou stranu. V rámci síťové komunikace bylo více kandidátských technologií, ale jako nejvhodnější vyšlo použití webové služby na bázi REST.

Při vývoji aplikace byl zvolen úspěšný architektonický vzor MVC, jehož podrobnosti jsou rozebrány jak v části návrhu, tak implementace. Aplikace taktéž používá několik moderních prvků uživatelských rozhraní jakou například *Action bar* a *Quick actions*. Pro zobrazení jednotlivých statistických dat jsou použity grafy a uživatel tak má přehled o vydaných nákladech, navíc může porovnávat statistiky jednotlivých automobilů mezi sebou.

Aplikace dosáhla všech na ni kladených cílů, ať už v oblasti ovládání, tak v oblasti správy nákladů a synchronizace. Uživatel má okamžitý přehled o nákladech na vozidlo, a to nejen nákladů na palivo, ale také na servis a pravidelné výdaje. Jednotlivá data jsou synchronizována se serverem a uživatel tak má svá data neustále zálohována. Tato synchronizace se provádí buď na požádání, nebo v zadaných časových intervalech. Díky tomu si uživatel nemusí manuálně spouštět synchronizaci, protože její spuštění je naplánováno v systému.

V rámci budoucích rozšíření by bylo vhodné vyvinou internetový portál, který by využíval webové služby a umožňoval tak efektivní provázání dat na mobilním zařízení a webovém serveru. Uživatel by si tak mohl vybrat způsob, jakým si přeje data zadat, případně prezentovat.

Literatura

[1] Apple, Inc.: *iOS Architecture Overview* [online]. 2010, [cit. 2010-11-14]. URL http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/IPhoneOSOverview/IPhoneOSOverview.html

[2] Apple, Inc.: *iOS Reference* [online]. 2010, [cit. 2010-11-14]. URL http://developer.apple.com/library/ios/navigation/

[3] Codehaus: *XStream* [online]. 2008, [cit. 2010-04-30]. URL http://xstream.codehaus.org/index.html

[4] Google, Inc.: *Cloud to device messaging* [online]. 2011, [cit. 2011-05-15]. URL http://code.google.com/intl/cs-CZ/android/c2dm/

[5] HIPP, R.: *SQLite* [online]. 2010, [cit. 2010-11-15]. URL http://www.sqlite.org/

[6] JUNGINGER, M.: *XStream for Android* [online]. 2008, [cit. 2010-04-30]. URL http://jars.de/java/android-xml-serialization-with-xstream

[7] LACKO, L.: *Programujeme mobilní aplikace ve Visual Studiu .NET*. Computer Press, 2004, ISBN 80-251-0176-2, 480 s.

[8] MEIER, R.: *Professional Android 2 Application Development*. Wrox, 2010, ISBN 978-0-470-56552-0, 576 s.

[9] Microsoft Corporation: *Windows Mobile Developer Center*[online]. 2009, [cit. 2010-11-14]. URL http://msdn.microsoft.com/en-us/windowsmobile/

[10] Microsoft Corporation: *Windows Mobile 6.5* [online]. 2010, [cit. 2011-05-15]. URL http://msdn.microsoft.com/cs-cz/library/bb158486(en-us).aspx

[11] Open Handset Alliance: *Android ADT* [online]. 2010, [cit. 2010-11-15]. URL http://developer.android.com/guide/developing/eclipse-adt.html

[12] Open Handset Alliance: *Android Architecture* [online]. 2010, [cit. 2010-11-13]. URL http://developer.android.com/guide/basics/what-is-android.html

[13] Open Handset Alliance: *Android Platform Versions* [online]. 2010, [cit. 2010-11-15]. URL http://developer.android.com/resources/dashboard/platform-versions.html

[14] Open Handset Alliance: *Android Reference* [online]. 2010, [cit. 2011-05-20]. URL http://developer.android.com/reference/packages.html

[15] Open Handset Alliance: *Android SDK* [online]. 2010, [cit. 2010-11-15]. URL http://developer.android.com/sdk/index.html

[16] Open Handset Alliance: *Android tools* [online]. 2010, [cit. 2010-11-15]. URL http://developer.android.com/guide/developing/tools/index.html

[17] Open Handset Alliance: *Android Platform Versions* [online]. 2011, [cit. 2011-05-08]. URL http://developer.android.com/reference/android/app/Activity.html

[18] OWENS, M.: *The Definitive Guide to SQLite*. Apress, 2006, 464 s.

[19] PostgreSQL Global Development Group: *PostgreSQL* [online]. 2010, [cit. 2010-11-15]. URL http://www.postgresql.org

[20] RICHARDSON, L.; RUBY, S.: *RESTful Web services*. O'REILLY, 2007, ISBN 978-0-596-52926-0, 448 s.

[21] RUSTY, H.; SCOTT, W.: *XML v kostce*. Computer Press, 2002, ISBN 80-7226-712-4, 456 s.

[22] Sun Microsystems, Inc.: *JAX-RS* [online]. 2010, [cit. 2010-04-30]. URL http://jsr311.java.net/nonav/releases/1.1/index.html

[23] Sun Microsystems, Inc.: *MySQL* [online]. 2010, [cit. 2010-11-15]. URL http://www.mysql.com

[24] Sun Microsystems, Inc.: *Jersey* [online]. 2011, [cit. 2011-04-30]. URL http://jersey.java.net/

[25] Symbian Foundation: *Introduction to Symbian OS* [online]. 2010, [cit. 2010-11-13]. URL http://developer.symbian.org/main/documentation/reference/s3/pdk/GUID-13987218-9427-455E-AC77-ADE6B0E9CD7E.html

[26] Symbian Foundation: *Symbian OS Reference* [online]. 2010, [cit. 2010-11-13]. URL http://developer.symbian.org

[27] The Apache Software Foundation: *Apache Tomcat* [online]. 2010, [cit. 2010-04-30]. URL http://tomcat.apache.org/

[28] VASWANI, V.: *My SQL Database Usage & Administration*. Mc Graw Hill, 2010, ISBN 978-0-07-160550-2, 368 s.

[29] Yahoo! Inc.: *Read/write access to your Delicious bookmarks and tags via an HTTP-based interface* [online]. 2011, [cit. 2011-04-13]. URL http://www.delicious.com/help/api