

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Mikrosystém pro řízení vztahů se zákazníky

Štěpán Šanda

Bakalářská práce
2014

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2013/2014

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: Štěpán Šanda
Osobní číslo: I10221
Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie
Název tématu: Mikrosystém pro řízení vztahů se zákazníky
Zadávající katedra: Katedra informačních technologií

Z á s a d y p r o v y p r a c o v á n í :

Cílem práce je navrhnout a vytvořit mobilní mikro CRM systém pro vybranou mobilní platformu s důrazem na správné uživatelské rozhraní, které se bude využívat návrhové vzory dané platformy.

Teoretická část:

V této části bude popsán princip CRM systémů, výběr správné mobilní platformy a UI/UX designové návrhové vzory pro danou platformu.

Implementační část:

Bude navržen a implementován základní mikro CRM systém pro vybranou mobilní platformu, který bude komunikovat s webovou službou na serveru v jazyce C # . Webová služba bude poskytovat data z databáze a také je do databáze nahrávat.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: tištěná/elektronická

Seznam odborné literatury:

HARDY, Brian. Android programming: the big nerd ranch guide. 1st ed. Atlanta, GA: Big Nerd Ranch, 2013. ISBN 03-218-0433-3.

FREEMAN, Adam. Pro ASP.NET MVC 4: interaction design solutions for developers. 4th ed. Berkeley, Calif: Apress, c2013, xxvii, 423 p. ISBN 14-302-4236-1.

DYCHÉ, Jill. The CRM handbook: a business guide to customer relationship management. Boston: Addison Wesley, c2002, xxiv, 307 p. ISBN 02-017-3062-6.

Vedoucí bakalářské práce:

Ing. Zdeněk Šilar

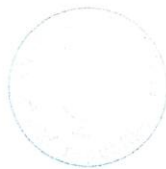
Katedra informačních technologií

Datum zadání bakalářské práce: **20. prosince 2013**

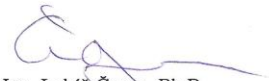
Termín odevzdání bakalářské práce: **9. května 2014**



prof. Ing. Simeon Karamazov, Dr.
děkan



L.S.



Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2014

Prohlášení autora

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 29. 04. 2014

Štěpán Šanda

Poděkování

Tímto bych chtěl poděkovat panu Ing. Zdeňku Šilarovi, Ph.D. za poskytnuté materiály a rady při tvorbě bakalářské práce. Dále bych chtěl poděkovat rodině za podporu při studiu.

ANOTACE

Cílem práce je navrhnout a implementovat mobilní systém pro řízení vztahů se zákazníky, který bude komunikovat s webovou službou implementovanou v jazyce C#. Teoretická část bude věnována definici systému CRM, výběru mobilní platformy a doporučením pro tvorbu uživatelského rozhraní. V praktické části bude poté implementována webová služba, která bude komunikovat s databází a zajišťovat přenos dat mezi klientem a službou. Dále pak mobilní aplikaci sloužící jako klient pro komunikaci se serverem.

KLÍČOVÁ SLOVA

Android, webová služba, REST, Java, CRM, MSSQL

TITLE

Microsystem for customer relationship management

ANNOTATION

The aim of this thesis is to design and implement a mobile system for customer relationship management, which will communicate with the Web Service that will be implemented using C# language. The theoretical part will be devoted to the definition of CRM, mobile platform selection and recommendations for creating user interfaces. In the practical part will be implemented Web Service that will communicate with the database and provide data transfer between client and service. Furthermore, the mobile client that will communicate with server and provide user interface for controlling the system.

KEYWORDS

Android, Web Service, REST, Java, CRM, MSSQL

OBSAH

ÚVOD.....	12
1 Definice CRM.....	13
1.1 Historie.....	13
1.2 Důvody nasazení.....	13
1.3 Bezpečnost.....	13
1.4 Budoucnost	14
1.5 Mobilní CRM.....	14
1.5.1 Výhody.....	14
1.5.2 Požadavky.....	15
1.5.3 Zabezpečení	15
2 Možnosti vývoje mobilního CRM.....	16
2.1 Responzivní web vs. nativní aplikace vs. multiplatformní aplikace.....	16
2.1.1 Responzivní web.....	16
2.1.2 Nativní aplikace	16
2.1.3 Multiplatformní vývoj.....	17
2.1.4 Porovnání	18
2.2 Mobilní platformy.....	18
2.2.1 iOS	18
2.2.2 Android	19
2.2.3 Windows 8	19
2.2.4 Porovnání	20
3 Android	21
3.1 Historie.....	21
3.2 Vývojové nástroje	22
3.3 Komponenty Android aplikace	23
3.3.1 Aktivita	23
3.3.2 Fragmenty	24
3.3.3 Pohledy	25
3.3.4 Manifest	26
3.3.5 Zdroje.....	26
3.4 Doporučení při vytváření uživatelského rozhraní.....	26
3.4.1 Gesta	27

3.4.2	Navigace	27
3.4.3	Action Bar	28
3.4.4	Potvrzení a informování uživatele	28
4	Webové služby	29
4.1	SOAP	29
4.2	REST	30
4.2.1	Jednotné rozhraní	31
4.2.2	Bez-stavový	31
4.2.3	Uložitelný do mezipaměti	32
4.2.4	Klient – Server	32
4.2.5	Vrstvený systém	32
4.2.6	Kód na vyžádání	32
4.3	Formáty dat	32
4.4	Ukázka rozhraní RESTful služby	32
5	Implementace CRM	34
5.1	Webová služba	34
5.1.1	REST	34
5.1.2	Web Api 2	36
5.1.3	Návrhový vzor MVC	36
5.1.4	Entity Framework	36
5.1.5	Asynchronní přístup k databázi	38
5.1.6	Přístup k datům pomocí LINQ	38
5.1.7	Navázání modelu	39
5.1.8	Autorizace a autentizace uživatele	40
5.1.9	Zabezpečení	42
5.1.10	Ukládání do mezipaměti	42
5.2	Databáze	43
5.2.1	Možnosti tvorby databáze	43
5.2.2	Entity a vazby	44
5.2.3	Publikování databáze	44
5.3	Mobilní aplikace	45
5.3.1	Obrazovky a navigace	45
5.3.2	Fragmenty	46

5.3.3	Knihovna Volley	46
5.3.4	Knihovna Gson	48
5.3.5	Implementace MVC v Androidu	50
5.3.6	Návrhový vzor Singleton	51
5.3.7	Export do PDF souborů a uložení do Google Disku	51
5.3.8	Záměr navigace, volání a emailu	52
5.4	Použité nástroje.....	52
6	Testování.....	57
6.1	Webová služba.....	57
6.2	Android aplikace.....	57
6.3	Reálné testování.....	58
7	ZÁVĚR	59
8	Literatura	60
9	Přílohy.....	62

SEZNAM ILUSTRACÍ A TABULEK

Obrázek 1 Aktuální zastoupení verzí Android na zařízeních [6].....	22
Obrázek 2 Životní cyklus aktivity	24
Obrázek 3 Ukázka využití fragmentů [6]	25
Obrázek 4 Zobrazení seskupení View pomocí ViewGroup	25
Obrázek 5 Navigace zpět a) systémové zpět b) tlačítko <i>Up</i>	27
Obrázek 6 Navigace pomocí tlačítka zpět vs. <i>Up</i>	28
Obrázek 7 Action Bar	28
Obrázek 8 Potvrzovací dialog a informační toast	28
Obrázek 9 Webová služba	29
Obrázek 10 Web Api MVC diagram	36
Obrázek 11 Zabezpečené HTTPS spojení	42
Obrázek 12 Přihlašovací obrazovka	45
Obrázek 13 MVC diagram v podání Androidu	50
Obrázek 14 UML diagram návrhového vzoru Singleton.....	51
Obrázek 15 Ukázka spuštění GPS navigace	53
Obrázek 16 Ukázka IDE Visual Studia 2013	53
Obrázek 17 Ukázka IDE Android Studio	54
Obrázek 18 Fiddler	55
Obrázek 19 Ukázka programu GenyMotion.....	56
Tabulka 1 Historie verzí OS Android [3]	21
Tabulka 2 Podporovaná gesta a jejich akce	27
Tabulka 3 Ukázka HTTP požadavků REST služby.....	33
Tabulka 4 Ukázka URI adres entity zákazník	35

SEZNAM ZKRATEK

CRM	Customer Relationship Management
SSL	Secure Socket Layer
VPN	Virtual Private Network
HTTPS	Hypertext Transfer Protocol Secure
HDPI	High Dots Per Inch
HDReady	High Definition Ready
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
UI	User Interface
MVC	Model View Controller
GUI	Graphic User Interface
OS	Operating System
LINQ	Language-Integrated Query
IDE	Integrated Development Environment
XML	Extensible Markup Language
JSON	JavaScript Object Notation
GPS	Global Positioning System
ADT	Android Development Tools
SDK	Software Development Kit
ADB	Android Debug Bridge
URL	Uniform Resource Locator
CRUD	Create, Read, Update, Delete
HTTP	Hypertext Transfer Protocol
SOAP	Simple Object Access Protocol
REST	Representational State Transfer
URI	Uniform Resource Identifier
ORM	Object/Relational Mapping
OOP	Object Oriented Programming
SQL	Structured Query Language
DDL	Data Definition Language
SMS	Short Message Service
RAM	Random Access Memory
HW	Hardware
SW	Software
RSA	Rivest, Shamir, Adleman (jména autorů)

ÚVOD

Práce je věnována především implementaci CRM systému, který slouží firmám pro správu informací o zákaznících, zaměstnancích, produktech distributorech a prodejkách. Ke správné funkci takového systému, bylo potřeba implementovat i webovou službu, která bude dostupná pro klienty z internetu.

Námět na tuto práci vznikl při poptávce ředitele jedné firmy, která měla zájem o takový systém, ale jejím požadavkům nevyhovoval žádný z nabízených na trhu. Firma měla specifické požadavky, které se týkaly zabezpečení, možnosti lokalizovat zaměstnance, a také potřebovala mít data uložena u sebe bez placení paušálních poplatků.

Úvod práce je věnován seznámení čtenáře s definicí CRM systému, co si pod tímto pojmem lze představit a jaké je jeho využití v praxi. Po krátkém úvodu do definice CRM systému následuje vysvětlení problematiky mobilních CRM systémů, jaké jsou jejich výhody a rizika. Dále je probráno, jaké jsou možnosti vývoje mobilního CRM a po krátkém zhodnocení následuje také výčet a charakteristika základních mobilních platforem, které jsou v závěru porovnány. Následně je podrobněji popsána jedna z popsaných mobilních platforem, čtenář se seznámí s krátkou historií, aktuálním zastoupením na trhu a také s vývojem pro danou platformu. Na konci teoretické části je obeznámen s pojmem webová služba a možnostmi její implementace.

Praktická část práce se zabývá především samotnou implementací CRM systému. Je zde popsána implementace webové služby, databáze a mobilní aplikace. Jsou zde uvedeny vybrané technologie, použité knihovny, postupy pro implementaci a také výčet a krátký popis použitých nástrojů. Na závěr následuje kapitola o průběhu testování systému, kde bude probráno, jakým způsobem byl systém testován během vývoje a také na konci vývoje v praxi.

1 Definice CRM

Pod zkratkou CRM se skrývá Customer Relationship Management. Nejvýstižnější volný český překlad je řízení vztahů se zákazníky. Jedná se o systém, který se snaží pomocí různých technologií organizovat, automatizovat a synchronizovat prodeje, marketing, zákaznický servis a technickou podporu. Neustále klade důraz na zvyšování hodnoty vztahů se zákazníky. [1]

1.1 Historie

S tímto pojmem se v oblasti informačních technologií setkáváme již několik let. V minulosti si tyto technologie mohly dovolit pouze velké společnosti, které měly finanční prostředky na drahou implementaci a provozování těchto systémů.

Z důvodu vývoje technologií, zejména internetu a elektronické komunikace, se kompletně mění pohled na využití CRM systémů výhradně pro velké firmy. Naopak CRM se stává velice důležitou záležitostí pro společnosti a organizace různých velikostí. Proto můžeme na současném trhu najít velkou spoustu zahraničních i českých dodavatelů CRM systémů, kteří jsou zaměřeni především na střední a malé firmy. CRM tedy již není pouze výhradou velkých společností, ale i menší firmy mají možnost vybrat si ze široké nabídky obecných i specificky zaměřených CRM systémů dle jednotlivých odvětví.

1.2 Důvody nasazení

Nasazení CRM systému automaticky neznamená zvýšení obrátu. Zaručeně ovšem přináší možnosti zvýšení výkonu celé firmy i zlepšení obchodních výsledků.

CRM zaručuje mít úplný a vždy aktuální přehled o obchodě, zákaznících, prodejkách a dalších důležitých informacích. Poskytuje všem zaměstnancům okamžitý přístup k aktuálním datům firmy. Snaží se ulehčit administrativu a minimalizovat papírování.

1.3 Bezpečnost

Základní funkcí CRM systémů je získávat a shromažďovat data o zákaznících, prodejkách i o celé firmě. Je tedy nutné tato data dostatečně zabezpečit proti jejich úniku nebo ztrátě. Ztráta důležitých dat nebo únik citlivých informací by mohly mít pro firmu fatální následky. Zabezpečení CRM systémů je tedy jednou z nejdůležitějších a zároveň nejvíce podceňovaných věcí při jejich implementaci.

Z právního pohledu provozovatel CRM systému, který shromažďuje data o zákaznících, je povinen systém dostatečně zabezpečit proti úniku informací o zákaznících. Také je potřeba vyřešit požadavek zákazníků na neposkytování informací třetím osobám a společností.

1.4 Budoucnost

Většina prodejců již nabízí cloudové řešení jejich systémů a také řešení software jako služba. Klasická vlastní implementace na vlastních serverech se stala výhradou pouze velkých firem, kde je potřeba vysoká integrace s ostatními firemními systémy. Software jako služba nabízí nesporné výhody pro menší firmy. Zákazník si zaplatí za služby některého online CRM systému a nemusí se starat o správu serveru a údržbu systému. Samozřejmostí jsou automatické zálohy, zabezpečení systému na vysoké úrovni a technická podpora od poskytovatele systému. Odpadá tedy nutnost platit zaměstnance, který by se o tyto věci musel starat.

Většina těchto cloudových řešení poskytuje pro ovládání systému webové rozhraní. S tím souvisí další z nových trendů, a to používání pokročilých technologií, jako například JQuery, Ajax, HTML5 a dalších. Tyto technologie poskytují možnost pracovat se systémem interaktivně, poskytovat funkce typu „Drag & Drop“¹ a možnost pokročilého grafického rozhraní.

Dalším z nových trendů CRM systémů je podpora mobilních zařízení, která umožňují spravovat a sledovat data firmy téměř odkudkoliv, kde je možné připojení k internetu. Veškeré informace a aktuální dění ve firmě může tedy majitel nosit doslova ve své kapse. To mu umožňuje ihned reagovat na nouzové situace, aniž by musel být ve firmě nebo doma u počítače.

1.5 Mobilní CRM

Umožňuje zaměstnancům používat jejich mobilní telefony, tablety nebo notebooky k přístupu do firemního CRM odkudkoliv. Nejlepší mobilní CRM systémy umožňují využívat veškeré funkce systému bez omezení a pracovat i v režimu bez internetu.

1.5.1 Výhody

Hlavní výhodou mobilního CRM je zaručeně přístup k firemním datům nejen z firmy. Manažeři v současné době hodně cestují a tak je potřeba mít na cestách všechny důležité informace vždy po ruce. Díky mobilnímu CRM odpadá nutnost před schůzkami tisknout reporty, přípravy na schůzky či aktuální prodeje a obraty. Stačí, pokud má u sebe například tablet. Manažer má možnost vidět a řídit výkon celého týmu ať je ve firmě v kanceláři, na služební cestě v cizině nebo na dovolené. Ve svém zařízení může sledovat aktuální dění ve firmě a aktivitu všech zaměstnanců.

Mobilní CRM funguje, i když zrovna není internet k dispozici. Data se jednoduše synchronizují, až bude připojení aktivní. Důležité dokumenty se dají stáhnout do tabletu, takže s nimi může uživatel pracovat i bez internetu.

¹ http://www.w3schools.com/html/html5_draganddrop.asp

Obchodníci s sebou nemusí vozit produktové katalogy a ceníky. Mohou produkty prezentovat na svých tabletech a dokumenty mohou jednoduše sdílet přímo s obchodními partnery.

1.5.2 Požadavky

Mobilní CRM potřebují ke své funkčnosti přístup k serveru z internetu. To tedy znamená, že server, na kterém je umístěná databáze a celá aplikační logika systému, musí být přístupný z vnější nezabezpečené sítě. To sebou ovšem nese jistá bezpečnostní rizika. Pokud je server dostupný z veřejné sítě, je nezbytné ho dostatečně zabezpečit, jelikož kdokoli může server napadnout.

1.5.3 Zabezpečení

Webová služba musí zabezpečit přístup pouze autorizovaným uživatelům. Toho lze dosáhnout za použití autentifikačních a autorizačních filtrů. Autentifikační filtr ověřuje pravost uživatele. Zjišťuje, zda je uživatel ten, za kterého se vydává, například pomocí ověření uživatelského jména a hesla. Autorizační filtr naopak ověřuje, zda má již ověřený uživatel na dané operace práva, či nikoli.

Vzhledem k tomu, že veškerá komunikace probíhá po nezabezpečené veřejné síti, je nutné ji zabezpečit pomocí SSL. Při nezabezpečené komunikaci hrozí riziko, že ji může kdokoli odposlouchávat, a dostat se tak k citlivým datům, jako jsou například přihlašovací údaje. Pomocí SSL můžeme zaobalit klasickou HTTP komunikaci na aplikační vrstvě a zašifrovat ji. Toto spojení je označované jako HTTPS (HTTP over SSL).

Další možností, jak zabezpečit komunikaci, je použití VPN. VPN je nástroj, který umožňuje zařízením posílat a přijímat data skrze veřejnou nezabezpečenou síť, jako kdyby byla přímo připojena do privátní sítě, a přitom využívat funkčnosti, bezpečnosti a politiky privátní sítě. Veškerá komunikace takto spojených počítačů je šifrována, a tak ji můžeme považovat za bezpečnou. Nastavení VPN probíhá na klientském operačním systému. To ovšem může být problém, pokud na jednom zařízení chceme provozovat dvě aplikace vyžadující VPN spojení. Na druhou stranu aplikace se nemusí vůbec starat o zabezpečení spojení, protože se připojení tváří, jako když je ve stejné privátní síti.

V neposlední řadě stojí za zmínění také zabezpečení samotné aplikace na straně klienta. Protože mobilní CRM systém je spuštěn na nějakém mobilním zařízení (notebook, tablet, telefon), musí se počítat se situacemi jako ztracení či odcizení zařízení. V takovém případě by mohlo dojít ke zneužití informací a popřípadě nekonzistenci dat. Proto je nutné dostatečně zabezpečit samotnou mobilní aplikaci požadavkem na ověření autority.

2 Možnosti vývoje mobilního CRM

V této kapitole budou probrány možnosti vývoje mobilního CRM systému, porovnání mobilních platforem, základní prvky a návrhové vzory uživatelského rozhraní vybrané platformy.

2.1 Responzivní web vs. nativní aplikace vs. multiplatformní aplikace

Existuje několik možností jak implementovat mobilní CRM systém. Každá z těchto možností má svá specifika, výhody a omezení. Níže jsou popsány tři nejčastější možnosti implementace mobilních řešení.

2.1.1 Responzivní web

Jedná se o metodu vytváření webových stránek, které optimalizují své zobrazení a obnos přenášených dat pro všechny druhy nejrůznějších zařízení (počítače s HDPI monitorem, notebooky s HDReady displejem, tablety, mobily atd.). Jedná se o speciální způsob stylování HTML dokumentu pomocí CSS3, který umožní rozpoznat vlastnosti zařízení, na němž se má stránka zobrazit, a poté ji správně naformátovat, nahrát správné optimalizované obrázky a popřípadě zablokovat nebo zprovoznit specifické prvky na webu. Toho lze dosáhnout především pomocí prvku CSS3 Media Queries², kde můžeme specifikovat jak má web vypadat při různých rozlišeních displeje.

HTML5 dává vývojářům do rukou mocný nástroj, který umožňuje kromě klasického zobrazování webových stránek také vytváření webových aplikací, které dokážou získávat ze zařízení informace o aktuální poloze, přistupovat k fotoaparátu, fungovat bez připojení k internetu, umožňují vykreslovat grafické prvky použitím SVG nebo CSS3 2D/3D, ukládání dat do paměti zařízení a mnoho dalšího.

Největší výhodou tohoto řešení je jeho multiplatformnost³. Webové aplikace fungují na jakémkoliv typu zařízení, které má k dispozici webový prohlížeč s podporou HTML5.

2.1.2 Nativní aplikace

Vývoj nativních aplikací vyžaduje programování v nativním jazyce platformy. Různé platformy jsou založeny na různých jazycích. Některé z hlavních mobilních platforem a jejich nativní jazyky jsou iOS – C/Objective-C, Android – Java, Windows Phone – C#, Windows 8 RT – C#.

Mezi hlavní výhody nativního vývoje aplikací patří výkon. Za další pozitivum lze pokládat také takzvaný *User Experience* – neboli volně přeloženo, uživatelský pocit. Uživatelé budou mít stejné grafické prostředí jako OS dané platformy. To zahrnuje UI prvky, přechody mezi obrazovkami, ovládací prvky a další. Dalšími výhodami nativního vývoje jsou nástroje pro vývoj aplikací, a to nejen v rámci předem připravených nástrojů od tvůrců těchto OS

² <http://www.vzhurudolu.cz/prirucka/css3-media-queries>

³ <http://www.techterms.com/definition/multiplatform>

(Apple/Google/Windows), ale také široká škála open source nebo i placených knihoven poskytujících řešení mnoha častých problémů.

Za jedinou nevýhodu nativního vývoje se dá považovat skutečnost, že aplikace nebude multiplatformní, a tedy bude fungovat pouze na OS, pro který byla vyvíjena. Pokud by aplikace měla vyjít na více platform, je nutné ji pro jiný OS znovu naprogramovat. Toto přináší několik dalších nevýhod, jako je údržba kódu. Pokud je potřeba v aplikaci udělat nějakou změnu, je nezbytné upravovat každou aplikaci zvlášť a následně ji také zvlášť testovat. S tím souvisí další problém s počtem lidí pracujících na projektu. Například expert na programování pro Android obvykle také nebývá expertem na iOS, a proto je nutné zaměstnat dalšího vývojáře, který se bude starat o druhou platformu. Z toho plyne nutnost najmout vývojářský tým pro každou platformu, na které by aplikace měla fungovat.

2.1.3 Multiplatformní vývoj

HTML5 je sice také multiplatformní způsob vývoje aplikací, ale stále je vázané na zobrazování ve webovém prohlížeči a postrádá hlavní výhody nativního vývoje. Existuje několik nástrojů pro multiplatformní nativní vývoj aplikací. Velice populárním pro tvorbu multiplatformních nativních aplikací se stal Xamarin⁴. Jedná se o nástroj umožňující programovat nativní aplikace v programovacím jazyce C# pro většinu mobilních platform. Funguje na frameworku⁵ Mono⁶ respektive Mono Touch, který je určen pro mobilní vývoj. Jedná se o open-source implementaci Microsoft .NET frameworku.

Pomocí Xamarinu mohou vývojáři sdílet okolo 80 % kódu mezi všemi podporovanými platformami. Zbytek jsou jen specifické věci pro danou platformu, jako přístup k ukládání souborů, GUI aplikace a podobně. Aplikace tak na každé platformě bude vypadat jako nativní s *User Experience* dané platformy. Aby bylo možné dosáhnout co největšího počtu sdíleného kódu, je doporučeno pro vývoj používat návrhový vzor MVC.

Další výhodou je možnost využít jakékoliv knihovny určené pro iOS i Android bez jakýchkoliv omezení.

Xamarin jako takový je zdarma, ovšem pro publikování aplikace na Android nebo iOS je třeba zakoupit Xamarin Android, respektive Xamarin iOS. Pro Windows a Windows Phone, které se defaultně vyvíjí v C#, není nutné žádnou licenci kupovat. Tyto licence nejsou levné a pro tým lidí se mohou velice prodražit. Naštěstí existují i studentské licence, na kterých lze vyvíjet i komerční aplikace. Je tedy třeba nejdříve zvážit, zda se investice do Xamarinu vyplatí, nebo zda jít cestou klasického nativního vývoje, kde bude potřeba vícečlenný tým.

⁴ <http://xamarin.com/>

⁵ <http://www.techterms.com/definition/framework>

⁶ <http://www.mono-project.com/>

2.1.4 Porovnání

CRM systémy bývají velice komplikované a komplexní. U velkých firem je mnohdy požadovaná pokud možno 100% dostupnost a funkčnost systému. I malé výpadky pro ně mohou znamenat velké problémy nebo finanční ztráty.

Použití Xamarinu má velkou výhodu v možnostech multiplatformnosti aplikací, použití jazyka C# a vývoji v kvalitním IDE. Přestože se Xamarin zdá jako ideální volba pro tvorbu CRM systému, je to poměrně nový projekt, který stále není plně stabilní a obsahuje pár chyb. Protože Xamarin není oficiálně podporován od výrobců platform, mohou nastat situace nekompatibility při příchodu nových verzí platformy. V takovém případě se chyby opravdu špatně hledají a odstraňují.

HTML5 je i přes jeho oficiální označení ve vývoji, většina jeho prvků je stabilních a s největší pravděpodobností se již do vydání mnoho měnit nebudou. Nabízí jak multiplatformnost, přístup k hardware zařízení, jako je fotoaparát a GPS, tak také možnost pracovat i bez připojení k internetu. Stejně jako Xamarin je tento jazyk poměrně novinkou na poli vývoje mobilních aplikací, a proto pro něj neexistuje dostatek různých knihoven, které při běžném vývoji usnadňují práci. HTML5 se pro tvorbu firemních mobilních aplikací zatím stále moc nehodí, alespoň do té doby, dokud nebude oficiálně dokončeno a nebude pro něj existovat dostatek knihoven.

Jako nejlepší varianta pro vývoj firemního CRM systému se stále jeví nativní vývoj, který poskytuje bez kompromisů, a hlavně nejspolehlivější vývoj mobilních aplikací. Nativní vývoj je pro většinu firem, zabývajících se vývojem mobilních aplikací, stále jedinou a hlavní možností, jak produkovat kvalitní firemní aplikace pro zákazníky. Proto bude v této bakalářské práci pro vývoj firemního CRM systému použit nativní vývoj pro jednu vybranou mobilní platformu.

2.2 Mobilní platformy

Při volbě nativního vývoje je velice důležité vybrat správnou platformu pro dané účely. Je nutné zvážit mnoho faktorů a specifikace daných platform. Mobilní CRM systémy se v současné době budou s největší pravděpodobností používat více na tabletech než na mobilních telefonech, jelikož nabízejí větší obrazovku pro zobrazení informací a pohodlnější ovládání. Aktuálně se na trhu nachází tři majoritní OS pro tablety – iOS, Android a Windows 8.

2.2.1 iOS

Jedním z nejoblíbenějších OS na tabletech se stal operační systém iOS od společnosti Apple. Je to dáno zejména velkým množstvím kvalitních aplikací optimalizovaných pro tablety v obchodu pro iOS – App Store.

Cena těchto tabletů v porovnání s konkurencí je poměrně vysoká. Zákazník musí platit za prémiovou kvalitu produktu, která ne vždy koresponduje s rozdílem ceny oproti konkurenci.

Pro nativní vývoj iOS aplikací je nutné splnit několik kritérií. Za prvé je nutné iOS aplikace vyvíjet na operačním systému Mac OS, který je dostupný pouze pro počítače a notebooky společnosti Apple. Tyto počítače a notebooky se také projevují vyšší pořizovací cenou, takže se jedná o poměrně vysokou investici do začátku. Dále se aplikace pro iOS musí programovat v jazyce Objective-C, který je objektově orientovaným rozšířením jazyka C. Protože se tento jazyk používá výhradně pro programování iOS a Mac OS X aplikací, nemá většího uplatnění, a tak jsou vývojáři nuceni učit se jazyk, který lze uplatnit pouze na těchto dvou systémech. Oproti klasickým moderním objektově orientovaným jazykům, jako je například Java a C#, je zápis kódu Objective-C složitější, méně přehledný a také implementace základních věcí složitější. Pro možnosti instalace takové aplikace do zařízení je nutné její publikování na Apple Store, což vyžaduje mít zaplacený vývojářský účet. Základní cena se pohybuje okolo \$99 za rok a po vypršení platnosti účtu se aplikace od vývojáře z marketu odstraní. Je proto nutné každý rok licenci obnovit.

2.2.2 Android

Díky otevřenosti systému a velkému výběru nejrůznějších zařízení se stal velmi oblíbeným u uživatelů také operační systém Android od firmy Google. Android je rozsáhlá open source platforma, která vznikla zejména pro mobilní telefony a tablety a je založena na linuxovém jádře. Android lze najít také v chytrých televizích, přehrávačích, brýlích od Google, automobilových počítačích nebo také chytrých hodinkách.

Na rozdíl od systému iOS, který je poskytován pouze na produktech od společnosti Apple, je Android open source, a tedy jej mohou ostatní výrobci zařízení bezplatně používat. To v praxi znamená, že se na trhu vyskytuje nepřeborné množství zařízení s tímto operačním systémem. Díky tomu si mohou zákazníci vybrat produkt, který přesně splňuje jejich požadavky, a to jak HW výbavou, tak také cenou. Na trhu se vyskytují velice levné tablety se základní HW konfigurací, cenově od pár tisíc, až po špičkové nejmodernější tablety s cenou přesahující dvacet tisíc korun.

Vytvářet aplikace pro android lze pod Mac OS X, Windows nebo Linuxem. Android využívá pro tvorbu svých aplikací programovací jazyk Java, který je velmi znám a lze s ním programovat od klasických desktopových aplikací po serverové služby, na které se Java využívá v současné době nejvíce.

Vývoj aplikací pro Android je oproti konkurenci iOS velmi levný. Vývojáři stačí jakýkoliv počítač nebo notebook s téměř jakýmkoliv operačním systémem podporujícím Javu. Vývojové prostředí jsou taktéž zdarma. Jediný poplatek svázaný k publikováním aplikací na obchod Google Play je \$25 poplatek za vývojářský účet.

2.2.3 Windows 8

Nejnovějším hráčem na trhu s mobilními operačními systémy se stal Microsoft s Windows Phone 8 a Windows 8. Windows Phone 8 (WP8) vznikl ze starší verze WP7.5, která trpěla mnoha nedostatky. WP8 je oproti WP7.5 téměř nový operační systém postavený na novém

jádře, který se snaží poučit z chyb jeho předchůdce. I přesto by se ale dalo říci, že systém stále trpí některými dětskými chorobami, které Microsoft pomalu opravuje. Windows 8 (W8) je rozdělený do dvou verzí. Samotný systém W8 je klasickým nástupcem jeho předešlé verze Windows 7. Jeho hlavní změna se týká právě dotykového rozhraní určeného pro zařízení s dotykovými displeji. Druhá základní verze je určena zejména pro mobilní zařízení, která nedisponují takovým výkonem jako klasické desktopy. Obsahuje pouze dotykové rozhraní, na kterém není možné spouštět klasické desktopové aplikace, ale pouze aplikace pro něj určené.

Vývoj W8 aplikací vyžaduje samozřejmě operační systém s plnohodnotnými W8 a nainstalované vývojové prostředí Visual Studio. Operační systém W8 bývá předinstalován na většině nových notebooků a také existují studentské licence programu MSDNAA⁷ pro nekomerční využití. Visual Studio je poměrně drahý nástroj. Nicméně je také dostupný v programu MSDNAA pro nekomerční využití nebo existují takzvané Express edice. Express edice jsou základní verze Visual Studia, které jsou zdarma a mohou se využívat i ke komerčním účelům.

Microsoft poskytuje vývojářům několik možností, jak mohou vytvářet aplikace pro W8. První možností, zřejmě pro spoustu začátečníků tou nejjednodušší, je JavaScript a HTML/CSS. JavaScript slouží pro naprogramování aplikační logiky a HTML s CSS pro uživatelské rozhraní aplikace. Další možností je C#, Visual Basic nebo C++ a XAML. Stejně jako v předchozím případě XAML slouží pro tvorbu uživatelského rozhraní. Poslední možnosti jsou C++ a DirectX, které jsou vhodné zejména pro programování náročnějších programů či her. Vytvářet aplikace pro W8 je tedy poměrně snadné, protože Microsoft poskytuje široký výběr programovacích jazyků spolu s kvalitním IDE.

2.2.4 Porovnání

W8 zatím nejsou vhodným kandidátem pro implementaci. Vzhledem k menšímu výběru tabletů s tímto OS a také protože Microsoft platformu stále vyvíjí a mění, těší se zatím malé uživatelské oblíbenosti.

iOS bohužel trpí v České republice horším servisem a také vysokou cenou. Pokud by si menší firma, pro kterou bude CRM zaměřený, chtěla tento systém pořídit, musela by pro zaměstnance pořídit zařízení s tímto OS. To ovšem znamená poměrně vysoké pořizovací náklady.

Nejlepší volbou pro implementaci mobilního CRM se zdá být Android vzhledem k velkému výběru tabletů všech cenových kategorií, možnosti rozšíření paměti o levné paměťové karty, možnost nahrávat soubory do tabletu jako na paměťové médium, velké množství aplikací v obchodě, vývoj v programovacím jazyce Java a také minimum poplatků spojených s vývojem.

⁷ <http://e5.onthehub.com/WebStore/ProductsByMajorVersionList.aspx?ws=6cfb2d32-836f-e011-971f-0030487d8897&vsro=8>

3 Android

Android je operační systém založený na jádru Linuxu. Za jeho vývoj je zodpovědný Android Open Source Project (AOSP), který je primárně pod vedením Google. Jedná se tedy o open source platformu určenou především pro mobilní zařízení. V současné době má Android 80% zastoupení na trhu s chytrými telefony. [2]

3.1 Historie

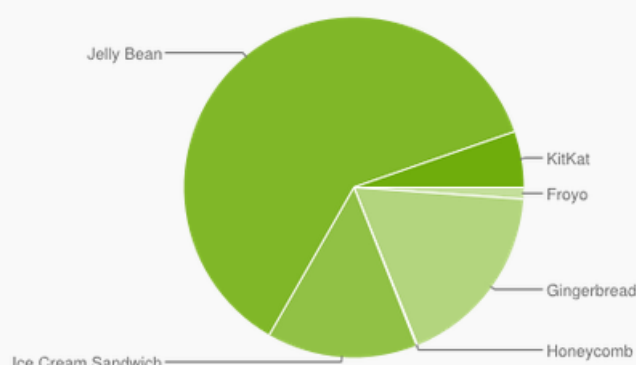
Android Inc. byl založen v říjnu 2003 v Kalifornii. Později v srpnu roku 2005 byl odkoupen společností Google Inc., jehož tým vyvinul platformu založenou na linuxovém jádru určenou pro mobilní zařízení. V říjnu roku 2008 byl poprvé uveden telefon T-Mobile G1 vyrobený společností HTC s operačním systémem Android a spolu s ním uvolněno SDK 1.0 pro vývojáře.

V Tabulce 1 jsou znázorněna data vydání jednotlivých verzí systému Android společně s nejzajímavějšími novinkami, které přinesly. A na Obrázku 1 aktuální procentuální zastoupení jednotlivých verzí.

Tabulka 1 Historie verzí OS Android [3]

Datum vydání	Verze	Stručný výpis novinek
30. duben 2009	1.5 Cupcake	bluetooth, widgety, nahrávání videí, funkce kopírovat a vložit, nová softwarová klávesnice
15. září 2009	1.6 Donut	Android market, mazání více souborů najednou, rychlé vyhledávání, vylepšená aplikace fotoaparátu, převod textu do řeči, grafické změny
26. říjen 2009	2.0, 2.1 Eclair	vylepšené UI, prohlížeč s podporou HTML5, podpora diody, živé tapety, optimalizace HW
20. květen 2010	2.2 Froyo	možnost instalovat aplikace na paměťovou kartu, WiFi hotspot, noční režim, Adobe Flash, optimalizace odezvy systému
6. prosinec 2010	2.3, 2.4 Gingerbread	velká grafická změna prostředí, podpora NFC, nová softwarová klávesnice, podpora nových senzorů, nové Google mapy
22. února 2011	3.0, 3.1, 3.2 Honeycomb	optimalizace pro tablety, verze pouze pro tablety, nový prohlížeč, funkce USB Host, vylepšený multitasking
19. říjen 2011	4.0, 4.0.4 Ice Cream Sandwich	zásadní změna UI, panoramatické focení, ukazatel přenesených dat, vylepšený správce kontaktů
27. červen 2012	4.1, 4.2 Jelly Bean	Google Now, notifikační lišta, vylepšený fotoaparát, podpora více účtů
Očekávané vydání 2014	5.0 Key Lime Pie	nová očekávaná verze, očekává se podpora Open GL ES 3.0 a Bluetooth se sníženou spotřebou

Version	Codename	API	Distribution
2.2	Froyo	8	1.1%
2.3.3 - 2.3.7	Gingerbread	10	17.8%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	14.3%
4.1.x	Jelly Bean	16	34.4%
4.2.x		17	18.1%
4.3		18	8.9%
4.4	KitKat	19	5.3%



*Data collected during a 7-day period ending on April 1, 2014.
Any versions with less than 0.1% distribution are not shown.*

Obrázek 1 Aktuální zastoupení verzí Android na zařízeních [6]

3.2 Vývojové nástroje

Android standardně poskytuje základní nástroje pro vývoj aplikací. V této podkapitole budou uvedeny a krátce popsány základní z nich. [4]

Android SDK

Android Software Development Kit (SKD) obsahuje nezbytné nástroje a knihovny pro vytváření, kompilování a testování aplikací pro Android.

Android Debug Bridge (ADB)

ADB je nástroj poskytovaný uvnitř SDK, který umožňuje spojení s virtuálním nebo reálným zařízením za účelem ladění aplikace.

Vývojová prostředí

Google poskytuje dvě základní vývojová prostředí pro vývoj Android aplikací.

Android Developer Tools (ADT), plugin určený pro Eclipse IDE. Jedná se o sadu komponentů, které rozšiřují základní prostředí Eclipse o nástroje umožňující a usnadňující vývoj Android aplikací.

Momentálně Google pracuje na novém vývojovém prostředí, založeném na IntelliJ IDEA. Aktuální verze je 0.5.5, která je již plně funkční pro běžný vývoj. Do budoucna se počítá s tím, že se Android Studio stane hlavním nástrojem pro vytváření Android aplikací. Celé prostředí je uzpůsobeno tak, aby vývojářům poskytlo co nejsnadnější vývoj aplikací pro Android. Veškeré potřebné nástroje jsou po instalaci ihned k dispozici a výborně integrované do prostředí.

Dalvik Virtual Machine

Dalvik Virtual Machine (Dalvik) je virtuální stroj sloužící ke spouštění Java aplikací na systému Android. Dalvik používá vlastní bajtový kód, který se liší od bajtového kódu Javy. Je nutné tedy třídy Javy nejdříve převést do bajtového kódu Dalvik a až poté je možné aplikaci spustit.

Programovací jazyk

Pro vývoj Android aplikací se primárně používá programovacího jazyka Java, který slouží především pro psaní logiky programu. Pro konfigurační soubory se používá jazyk XML.

3.3 Komponenty Android aplikace

Android aplikace je samostatný program, který lze spouštět a používat nezávisle na ostatních aplikacích.

Může obsahovat následující komponenty:

- aktivity (Activities)
- služby (Services)
- záměry (Intents)
- oběžníky (Broadcast receivers)
- poskytovatelé obsahu (Content providers)

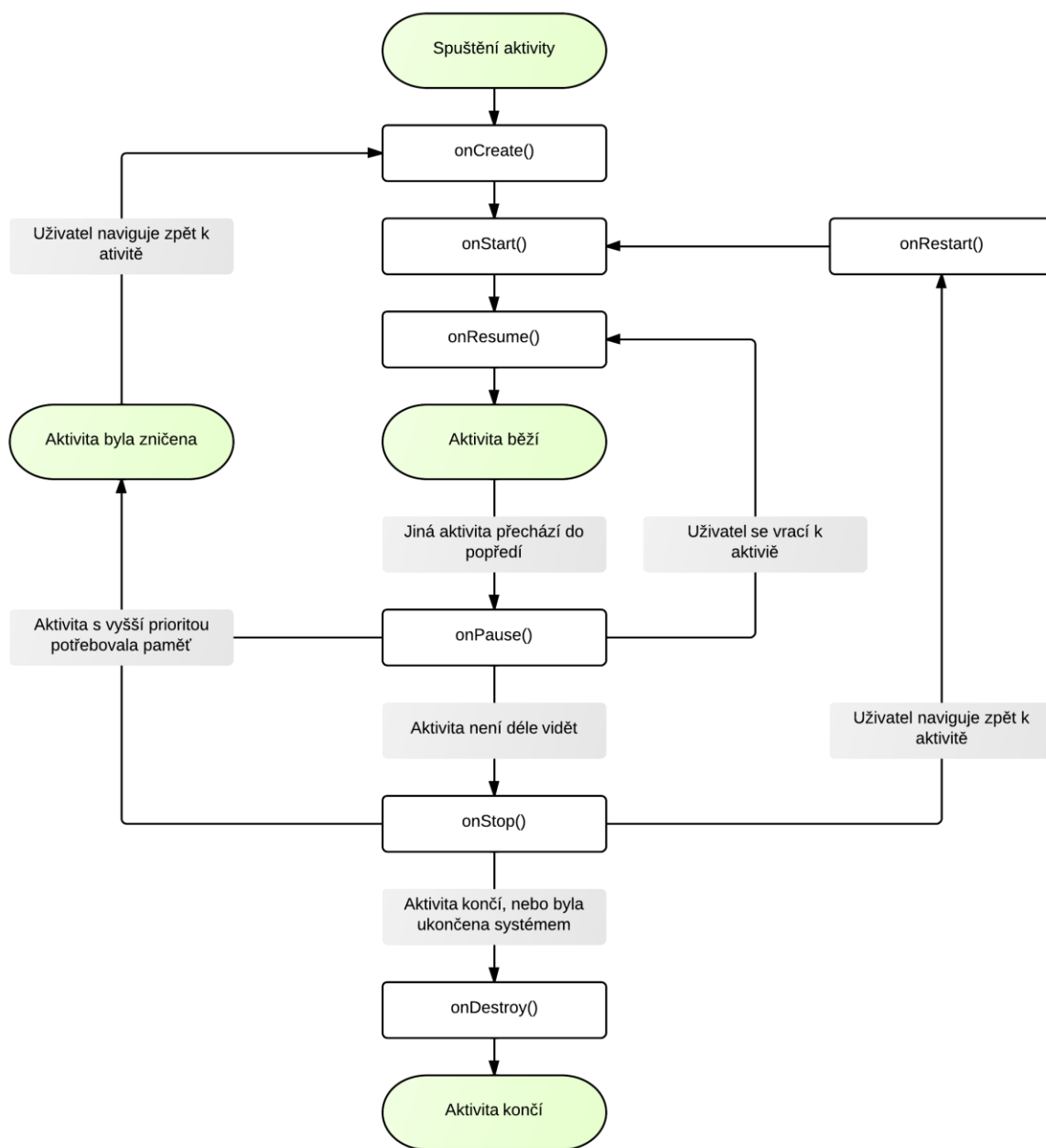
3.3.1 Aktivita

Jedná se o vizuální reprezentaci aplikace. Každá aktivita zobrazuje vlastní okno, ve kterém je vykresleno uživatelské grafické rozhraní, jež umožňuje interakci s uživatelem. Okno obvykle zaplňuje celou obrazovku displeje, ale může být i menší a vznášet se nad ostatními okny pod ním. Aplikace se většinou skládá z více aktivit, které na sebe navazují. Obvykle se zde vyskytuje jedna hlavní, která je tak také nazývána a zobrazí se uživateli po spuštění aplikace.

Každá aktivita může spouštět další, které mohou, ale nemusí, vracet výsledek. Pokaždé když je spuštěna nová aktivita, stará se zastaví, ale systém ji zachová v zásobníku, takže po skončení spuštěné aktivity se ta spouštěcí obnoví.

Pokud je aktivita zastavena, je notifikována o změně stavu skrze metody zpětného volání. Existuje několik metod, které aktivita může obdržet v důsledku změny jejího stavu. Jedná se o stavy vytváření, pozastavení, zastavení, zrušení a pokračování aktivity. Každé z těchto zpětných volání jí poskytuje možnost provést specifické akce. Nejčastěji se jedná o uložení a znovunačtení jejího stavu.

Tyto stavy jsou vyvolávány v souvislosti s životním cyklem aktivity, který je znázorněný na Obrázku 2.

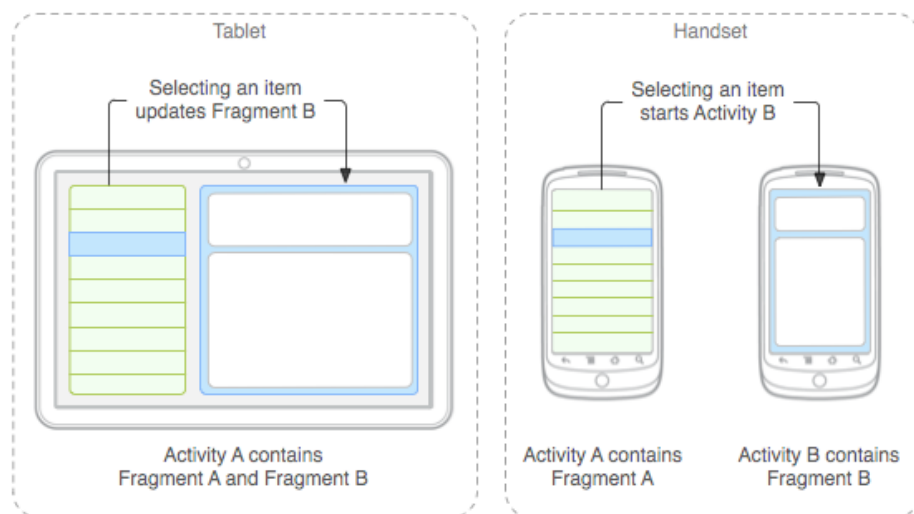


Obrázek 2 Životní cyklus aktivity

Změna stavu nemusí být vyvolána vždy jen spouštěním, nebo vypínáním aktivity, ale také například rotací displeje, změnou konfigurace zařízení, ukončením aplikace systémem pro nedostatek volných zdrojů a dalšími podobnými důvody. [5]

3.3.2 Fragmenty

Fragmenty jsou komponenty, které běží v kontextu aktivity. Zapouzdřují kód aplikace, a tak je možné jejich znovuvyužití v dalších aktivitách. Obrázek 3 znázorňuje využití fragmentů.



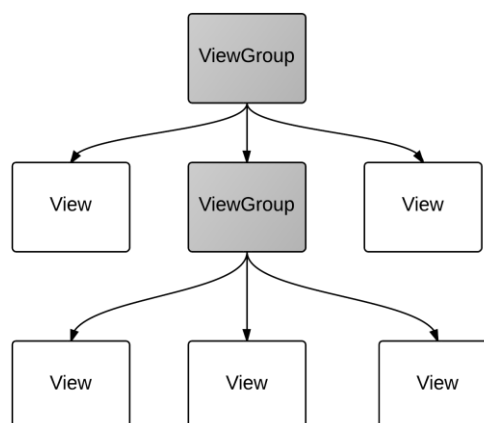
Obrázek 3 Ukázka využití fragmentů [6]

Fragmenty také kopírují životní cyklus aktivity, v jejímž kontextu jsou spuštěny, a poskytují velmi podobné metody pro zpracování změny stavu.

3.3.3 Pohledy

Uživatelské rozhraní je typicky definováno ve speciálních XML souborech. Lze ho ovšem definovat či měnit jeho vlastnosti přímo v kódu aplikace. Je možné definovat stejný XML soubor v různých konfiguračních složkách, které určují, pro jakou velikost zařízení se mají použít. Android poté sám rozhodne podle typu zařízení, jaký XML soubor vybrat pro dané zařízení. Lze tak jednoduše oddělit rozhraní pro mobilní zařízení a tablety.

Veškeré UI elementy v Android aplikacích jsou složeny pomocí pohledů (Views) a seskupení pohledů (ViewGroup). View je element, který se zobrazuje na obrazovce a umožňuje uživateli komunikovat s aplikací jako například tlačítko, obrázek, textové pole a podobné. ViewGroup je objekt, který v sobě uskupuje pohledy a specifikuje v jakém pořadí a na jaké místo se mají vykreslit (Obrázek 4).



Obrázek 4 Zobrazení seskupení View pomocí ViewGroup

3.3.4 Manifest

Jedná se o XML soubor, který je uložený v kořenové složce aplikace. Každá aplikace musí tento soubor obsahovat. Jsou zde uvedené základní informace o aplikaci, které jsou poskytovány OS Android. Mimo jiné, poskytuje následující informace: [4]

- název Java balíčku aplikace – tento název musí být unikátní pro každou aplikaci, proto se jeho název nejčastěji vytváří pomocí následující konvence, zpětné URL adresy;

```
com.firm_name.project_name.version
```

- popis komponent aplikace – aktivity, služby, oběžníky a poskytovatele obsahu;
- oprávnění aplikace přístupu k zabezpečeným API – jedná se například o oprávnění přistupovat na internet, číst kontakty a podobné. Pokud aplikace nemá v manifestu patřičná oprávnění, nelze tyto API v kódu použít;
- minimální úroveň API;
- požadované vybavení zařízení, například požadavek fotoaparátu.

3.3.5 Zdroje

V Android aplikacích se zdroji myslí soubory typu obrázky, zvuky a konfigurační soubory. Ty jsou uchovávány odděleně od zdrojového kódu. Zdrojové soubory musí být ukládány ve složce `/res` a příslušné podsložce pro daný typ souboru. [4]

Přehled základních podporovaných předpon názvů složek:

- `/res/drawables` – obrázky a XML soubory popisující vykreslování objektů
- `/res/layout` – XML soubory popisující rozvržení UI aktivit a fragmentů
- `/res/menu` – vlastnosti položek hlavního menu
- `/res/values` – soubory, které definují vzhled aplikace

3.4 Doporučení při vytváření uživatelského rozhraní

Jedná se o všeobecná doporučení, jak by měla aplikace vypadat či jak by se měla za různých okolností chovat. Hlavní motivací je rychlá orientace uživatele v nové aplikaci. Při použití doporučených zavedených vzorů uživatelského rozhraní se uživatel nemusí seznamovat s funkčností grafických prvků. Například tři tečky vyvolají kontextové menu, šipka zpět zavede uživatele na předchozí obrazovku, nebo je známe, že na horní straně se vyskytuje panel s ovládacími prvky. Protože uživatel zná tyto návrhy z ostatních aplikací, které se jimi řídí, je pro něj velice snadné se v aplikaci zorientovat a proto je ovládání velice intuitivní. V této kapitole budou popsány základní doporučení pro vytváření uživatelského rozhraní.

3.4.1 Gesta

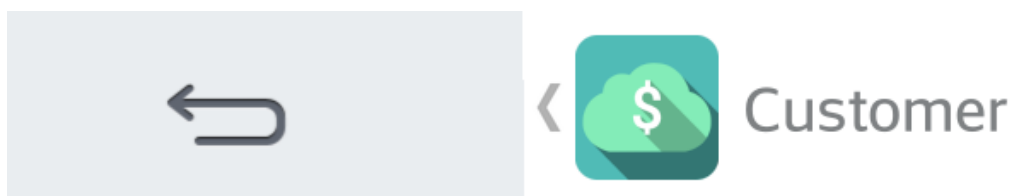
Gesta dovolují uživateli manipulovat obrazovkou pomocí předem definovaných pohybů po obrazovce. Tabulka 2 popisuje základní gesta a jejich význam.

Tabulka 2 Podporovaná gesta a jejich akce

Gesto	Základní chování
Stisknutí	Základní funkce vybrání objektu
Dlouhé stisknutí	Režim výběru dat. Umožňuje vybrat jednu nebo více položek a dále s nimi manipulovat. Nemělo by se využívat pro vyvolání kontextové nabídky.
Tažení	Posouvá obsah stejným směrem jako je pohyb tažení. Tahy jsou rychlé a efektivní.
Dlouhé stisknutí a tažení	Neboli technika zvaná <i>Drag & Drop</i> , uchopení objektu a jeho interaktivní přetažení na jiné místo.
Dvojité stisknutí	Přiblíží obsah na širší obrazovky a podruhé obsah oddálí. Také se používá jako druhotné gesto pro označení obsahu.
Dvojité stisknutí a tažení	Mění velikost obsahu tažením od nebo ke středu gesta.
Oddálení / Přiblížení dvou prstů	Přiblížení či oddálení obsahu

3.4.2 Navigace

Správná navigace v aplikaci je jedna z nejvíce intuitivních věcí, které uživatel očekává. Verze Androidu 2.3 a nižší implementovaly navigaci uvnitř aplikací pouze pomocí systémového tlačítka zpět. Se zavedením Action Bar⁸ ve verzi Android 3.0 se objevilo tlačítko *Up* (Obrázek 5). [6]

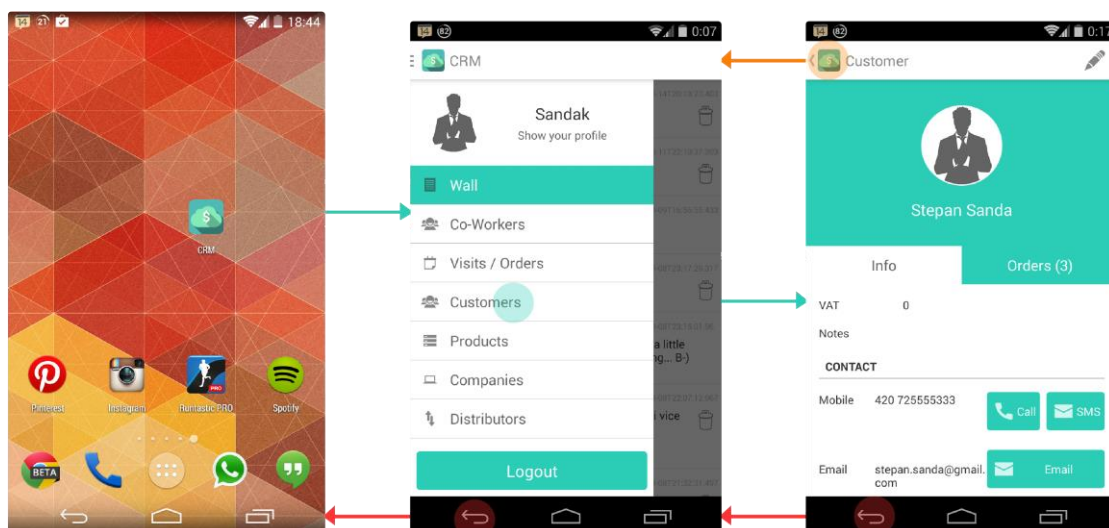


Obrázek 5 Navigace zpět a) systémové zpět b) tlačítko *Up*

Tlačítko *Up* se používá k navigaci v rámci jedné aplikace. Je založené na hierarchickém uspořádání obrazovek na sebe navazujících. Je-li obrazovka na nejvyšší úrovni, obvykle hlavní spouštěcí obrazovka, neměla by obsahovat tlačítko *Up*.

Tlačítko zpět se používá k navigaci v chronologickém pořadí historie spuštěných obrazovek. Je tak založeno spíše na čase, kdy byla obrazovka zobrazena než na hierarchii obrazovek dané aplikace (Obrázek 6).

⁸ <http://developer.android.com/guide/topics/ui/actionbar.html>



Obrázek 6 Navigace pomocí tlačítka zpět vs. *Up*

3.4.3 Action Bar

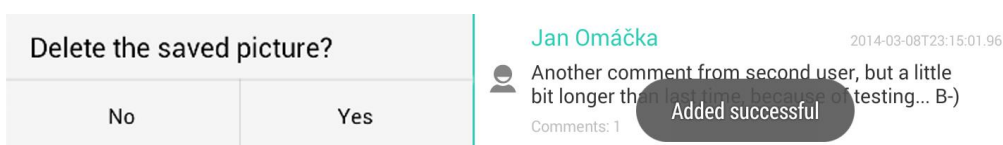
Poskytuje několik základních funkcí a nachází se na horní straně obrazovky. Levá strana obvykle obsahuje ikonu aplikace, která slouží jako tlačítko *Up*. Pravá strana pak obsahuje název aktuální obrazovky s možností rolovacího menu pro výběr aktuálního zobrazení, pokud to daná obrazovka umožňuje. Jako příklad lze uvést obrazovku kalendáře, kde lze měnit zobrazení roční, měsíční, týdenní a agenda. Pravá strana obsahuje nejdůležitější akční tlačítka pro rychlé a intuitivní ovládání. Tlačítka, která se nevejdou na obrazovku, jsou automaticky seskupena pod menu, které je značeno třemi vertikálními tečkami a zobrazeno na konci panelu (Obrázek 7).



Obrázek 7 Action Bar

3.4.4 Potvrzení a informování uživatele

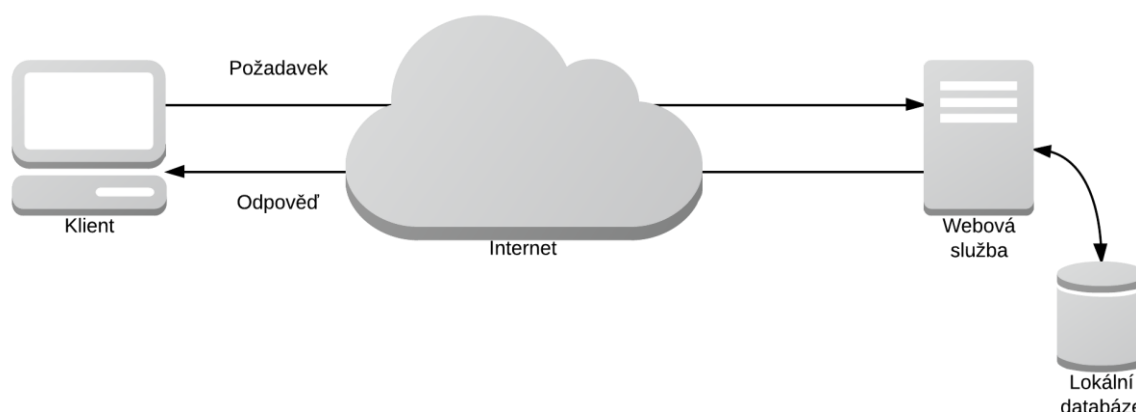
V některých případech vhodné uživatele požádat o potvrzení akce. Nejlepším příkladem je akce mazání. Stornovací tlačítko dialogu by mělo být pokaždé na levé straně a potvrzovací naopak na pravé. Vhodné je také informovat uživatele o dokončení akce. Uživatel by měl být informován o úspěšném, či neúspěšném dokončení akce, jako například ukládání dat. K tomuto informování se využívá takzvaných toustů (Obrázek 8).



Obrázek 8 Potvrzovací dialog a informační toast

4 Webové služby

Webovou službu lze popsat jako software, který poskytuje komunikaci mezi dvěma zařízeními skrze síť. Obvykle je spuštěna na serveru, který je přístupný z internetu a má svoji URL adresu, přes kterou se lze k webové službě připojit a provádět základní operace, jež poskytuje. Jedná se především o operace vytváření, čtení, editaci a mazání. Pro tyto operace se používá nejčastěji zkratka CRUD. Komunikace probíhá pomocí HTTP požadavků mezi klientem a službou. Obvykle je webová služba připojena k lokální databázi, ze které poskytuje data a zároveň tam nahrává data od klienta (Obrázek 9).



Obrázek 9 Webová služba

Webovou službu lze napsat v jakémkoliv programovacím jazyce, který podporuje komunikaci po síti. Nejčastěji se jedná o jazyky Java a C#. Nicméně vybraný programovací jazyk pro implementaci není vůbec podstatný, jelikož komunikace mezi klientem a službou probíhá pomocí HTTP požadavků, které podporuje téměř každý vyšší programovací jazyk. Lze tak například propojit webovou službu vytvořenou v jazyce C# s klientem napsaným v jazyce Java.

V současné době se webové služby nejčastěji implementují pomocí dvou základních způsobů. Jedná se o protokol SOAP a REST, které budou podrobněji popsány v následujících dvou podkapitolách.

4.1 SOAP

SOAP⁹ je zkratka pro *Simple Object Access Protocol*. Jedná se o komunikační protokol mezi dvěma aplikacemi, které komunikují přes internet. Specifikuje formát odesílaných zpráv. Je platformě nezávislý, založený na XML a doporučený konsorciem W3C od roku 2003. [7]

Protokol SOAP byl vytvořen pro komunikaci mezi aplikacemi skrze internet za pomoci HTTP, protože je podporovaný veškerými internetovými prohlížeči, servery i programovacími jazyky. SOAP poskytuje vývojářům možnost, jak komunikovat mezi

⁹ <http://www.w3.org/TR/soap12-part0/>

aplikacemi běžícími na odlišných operačních systémech a implementovanými v různých programovacích jazycích.

Zpráva SOAP je obyčejný XML dokument obsahující následující prvky:

- envelope – identifikující dokument XML jako zprávu SOAP,
- header – obsahující hlavičkové informace dokumentu,
- body – obsahující data požadavku či odpovědi,
- fault – obsahující chyby a informace o stavu.

Jedná se o procedurálně orientovaný protokol. To znamená, že URL adresa například pro vytvoření nové objednávky může vypadat následovně:

```
http://www.example.org/VytvorNovouObjednavku
```

VytvorNovouObjednavku je název dané metody poskytované webovou službou.

Ukázka SOAP požadavku: [8]

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

4.2 REST

REST neboli Representation State Transfer je styl softwarové architektury, který byl poprvé popsán v dizertační práci Roye Fieldinga, jednoho ze spoluautorů protokolu HTTP, Architectural Styles and the Design of Network-based Software Architectures¹⁰ v roce 2000. Využívá se především pro jednotné a jednoduché přistupování ke zdrojům. Zdroji rozumíme jakákoliv data nebo stavy aplikace, které je možné konkrétními daty popsat. Na rozdíl od SOAP, který je orientován procedurálně, je REST orientován datově. Všechny zdroje mají svoji jednoznačnou URI a definované základní operace pro práci s nimi. [9] [10]

¹⁰ <http://jpkc.fudan.edu.cn/picture/article/216/35/4b/22598d594e3d93239700ce79bce1/7ed3ec2a-03c2-49cb-8bf8-5a90ea42f523.pdf>

REST lze popsat šesti základními vlastnostmi: [11]

- jednotné rozhraní,
- bez-stavový,
- uložitelný do mezizpaměti,
- klient – server,
- vrstvený systém,
- kód na požádání.

4.2.1 Jednotné rozhraní

Jednotné rozhraní mezi klientem a serverem zjednodušuje a odděluje architekturu, která umožňuje každou část vyvíjet nezávisle. Čtyři hlavní zásady jednotného rozhraní jsou:

1. Zdrojově založený – jednotlivé zdroje lze identifikovat pomocí URI. Samotné zdroje lze koncepčně oddělit od reprezentací, které jsou vráceny klientovi. Například server neposílá jeho databázi, ale raději HTML, XML nebo JSON soubor, který reprezentuje záznamy v databázi a například určuje, že jsou kódované v UTF-8, v závislosti na žádosti od klienta nebo implementaci serveru.
2. Manipulace se zdroji skrze reprezentovaná data – pokud má klient k dispozici reprezentaci nějakých dat, včetně připojených meta dat, má dostatek informací pro upravování nebo mazání zdroje na serveru. Za předpokladu, že k tomu má dostatečná oprávnění.
3. Samopopisné zprávy – každá zpráva má dostatek informací k pochopení, jak zprávu zpracovat. Například jak zpracovat obsah zprávy, který je specifikovaný typem internetového média, dříve známý jako MIME typ. Odpovědi také uvádějí vlastnosti uložitelnosti do mezizpaměti daného zdroje.
4. Hypermedia jako aplikační stav – používá se anglická zkratka HATEOS . Klient zasílá stav pomocí obsahu v těle zprávy, parametry v řetězci, informacemi v hlavičce zprávy a požadovanou URI. Služba zasílá stav klientovi v obsahu těla zprávy, kódu odpovědi a hlavičce zprávy. Mimo jiné HATEOS také znamená, že vrácené tělo zprávy obsahuje i dodávající URI pro získání samotného objektu, nebo souvisejících objektů.

4.2.2 Bez-stavový

Každý požadavek musí v sobě obsahovat dostatek potřebných informací, aby ho služba mohla zpracovat. Klient nesmí počítat s tím, že si o něm služba pamatuje nějaké informace. To znamená, že se nevyužívá takzvaných *session*, které přetrvávají mezi mnoha požadavky od stejného klienta. Jako příklad si můžeme uvést autentizaci uživatele. Pokud služba vyžaduje pro daný požadavek autentizaci uživatele, každá zpráva od klienta musí obsahovat autorizační hlavičku, která poskytne službě dostatek informací k autentizaci uživatele.

4.2.3 Uložitelný do mezipaměti

Stejně jako na webu, klienti mohou data ukládat do mezipaměti. Proto musí odpověď obsahovat implicitní nebo explicitní informace o tom, zda se mají data uložit do mezipaměti a popřípadě jak dlouho zůstávají validní.

4.2.4 Klient – Server

Jednotné rozhraní odděluje server od klienta. Toto rozdělení znamená, že klienti se nestarají o ukládání dat na straně serveru. O to se stará výhradně server, a to přispívá k přenositelnosti kódu klienta. Naopak servery se nezabývají uživatelským rozhraním nebo stavu klienta, a to zjednodušuje implementaci serverů a umožňuje jejich větší škálovatelnost. Server a klient lze vyvíjet také samostatně do té doby, dokud se nezmění rozhraní.

4.2.5 Vrstvený systém

Klient nemůže zjistit, zda je připojen přímo ke koncovému serveru, nebo zprostředkovateli uprostřed cesty. Zprostředkovatelské servery umožňují vyrovnávat zatížení a poskytují sdílené *cache*. Vrstvy mohou být také vyžadovány z bezpečnostních důvodů.

4.2.6 Kód na vyžádání

Servery umožňují dočasně rozšířit logiku nebo upravit funkčnost klienta převodem logiky, kterou lze spustit na klientské straně. Jako příklad lze uvést zkompileované komponenty Java appletů nebo skriptů na straně klienta jako JavaScript.

4.3 Formáty dat

Pro výměnu dat mezi klientem a serverem se nejčastěji používají tři standardizované formáty:

- RSS,
- XML,
- JSON.

4.4 Ukázka rozhraní RESTful služby

Pomocí čtyř základních HTTP metod lze snadno implementovat CRUD operace nad zdroji. HTTP metoda jednoznačně specifikuje, jaká operace se s daty má provést, a URI jednoznačně reprezentuje zdroj, nad kterým se má operace provést. V Tabulce 3 je ukázka HTTP požadavků REST služby, porovnané se SOAP požadavky. [12]

Tabulka 3 Ukázka HTTP požadavků REST služby

HTTP metoda	Šablona URI	Ekvivalent RPC operace (například SOAP)	Význam
GET	users/{username}	getUserAccount	dotaz na profil daného uživatele
POST	users/{username}/foto	setUserProfileFoto	nahrává fotku do profilu specifikovaného uživatele
PUT	users/{username}/passsword	setUserNewPassword	mění heslo danému uživateli
DELETE	users/{username}	deleteUser	maže zadaného uživatele
GET	users/	getAllUsersAccounts	dotaz na kolekci uživatelských profilů

Stavové kódy slouží k informování o stavu požadavku. Nejčastějšími odpověďmi jsou následující kódy: [13]

- 200 OK – operace proběhla v pořádku
- 201 Created – zdroj byl vytvořen
- 204 No Content – používá se nejčastěji jako odpověď při použití metody DELETE pro mazání
- 400 Bad Request – požadavek je neplatný a vrací chybu
- 401 Unauthorized – požadavek, respektive uživatel, není autorizovaný k dané operaci
- 404 Not Found – zdroj nebyl nalezen.

Přehledný výpis všech stavových kódů s jejich popisem je možné dohledat na webových stránkách Restapitutorial¹¹.

¹¹ <http://www.restapitutorial.com/httpstatuscodes.html>

5 Implementace CRM

Při implementaci praktické části CRM systému byl kladen velký důraz nejen na správnou funkčnost, ale také na výběr správných technologií a nástrojů pro implementaci. Dále pak na správný návrh OOP, využití základních návrhových vzorů, jednotné a jednoduché uživatelské grafické rozhraní aplikace, které se řídí doporučeními vybrané platformy pro implementaci, a správné použití architektury REST pro komunikaci mezi službou a klientem.

Pro bezchybnou funkčnost mobilního CRM systému bylo potřeba vytvořit tři následující základní bloky. Databázi, která bude uchovávat a poskytovat veškerá potřebná data. Webovou službu dostupnou z internetu, která bude poskytovat a nahrávat data od klienta do databáze, poskytovat jednotné rozhraní, které splňuje definici REST architektury, a v neposlední řadě řešit autentizaci a autorizaci klienta. Poslední částí je poté mobilní aplikace, která slouží uživatelům pro ovládání celého CRM systému. Poskytuje uživatelské grafické rozhraní a veškeré podstatné funkce pro správnou funkčnost. Implementace těchto tří základních bloků bude popsána v následujících podkapitolách.

5.1 Webová služba

Základním kamenem celého CRM systému je webová služba, která se bude starat o logiku celé aplikace, zabezpečení systému a poskytovat API rozhraní, dostupné z internetu pro mobilní zařízení. Implementace webové služby zabrala zhruba stejně času jako implementace samotné mobilní aplikace. Jedná se o poměrně komplexní projekt obsahující desítky tříd a tisíce řádků kódu, proto byla nesmírně důležitá správná organizace a architektura aplikace.

5.1.1 REST

Nejprve bylo nutné se rozhodnout, jakým způsobem bude webová služba implementována. Jako softwarová architektura byla vybrána architektura REST, která se těší v posledních letech velké oblibě a vychází o ní mnoho odborných článků. Mezi hlavními důvody výběru byla hlavně jednoduchost implementace, platformě nezávislá komunikace pomocí HTTP požadavků, čitelnost URI a také podpora XML a JSON formátů pro přenos dat, které jsou dobře lidsky čitelné a v mobilních zařízeních lehce zpracovatelné.

V Tabulce 4 je ukázka URI adres pro CRUD entity zákazník. Každá entita v systému má svoji vlastní URI adresu a možnosti pro CRUD. Ovšem některé operace jsou omezeny uživatelskými rolemi, a to zejména operace mazání, které v některých případech může provádět pouze uživatel s přidělenou administrátorskou rolí.

Tabulka 4 Ukázka URI adres entity zákazník

HTTP metoda	URI adresa	Význam
POST	api/Customers/	nahrání nového zákazníka
GET	api/Customers/	vrací kolekci všech zákazníků
GET	api/Customers/{customerId}	vrací zákazníka dle vloženého ID
PUT	api/Customers/{customerId}	upravuje zákazníka dle vloženého ID
DELETE	api/Customers/{customerId}	maže zákazníka dle vloženého ID

Na každý přijatý požadavek od klienta služba odpoví příslušným HTTP kódem. Ukázky budou znovu prezentovány na příkladu controlleru zákazníka. Při vytváření nového zákazníka pomocí metody POST je, za předpokladu úspěšného nahrání, navrácen kód 201 Created, spolu s URI adresou právě nahraného zákazníka a samotným vytvořeným zákazníkem. Ve Web Api 2, které bude popsáno dále, takováto odpověď vypadá následovně:

```
return CreatedAtRoute<CustomerViewModel>("DefaultApi", new { id =
customer.CustomerId }, customerView);
```

Odpověď na požadavek specifického zákazníka metodou GET nebo aktualizaci zákazníka metodou PUT, za předpokladu úspěšného provedení, vrací kód 200 Ok spolu se zákazníkem a vypadá následovně:

```
return Ok(customerView);
```

A nakonec v případě požadavku DELETE pro smazání zákazníka vrací za předpokladu úspěšného provedení metody stavový kód 204 No Content a vypadá následovně:

```
return StatusCode(HttpStatusCode.NoContent);
```

V případě chyby na straně serveru vrací Web Api 2 automaticky chybové kódy 5xx Server Error, kde je popsána vyskytnutá chyba. V případě špatného požadavku od klienta vrací automaticky stavové kódy 4xx Client Error, které specifikují chybu v požadavku klienta. Nejznámější klientské chybové kódy jsou 400 Bad Request v případě špatného požadavku, 401 Unauthorized v případě chybějící nebo neplatné autorizační hlavičky, nebo také 404 Not Found v případě, kdy zdroj nebyl nalezen.

Kromě úspěšných návratových stavových kódů a automatických chybových kódů, o které se stará Web Api, jsou v controlleru také implementovány explicitní návratové kódy. Například pokud se klient pokouší vložit zákazníka, který se již v databázi nachází, server navrací stavový kód 409 Conflict. V případě nevalidního modelu zákazníka poslaného v těle zprávy vrací stavový kód 400 Bad Request s popisem chyby.

```
if (existingCustomer != null)
    return Conflict();
```

Dále také při žádosti o specifického zákazníka, který se v databázi nenachází, server vrací stavový kód 404 Not Found. A v neposlední řadě také, pokud se klient snaží volat operaci, ke které nemá dostatečné oprávnění, server vrací stavový kód 401 Unauthorized.

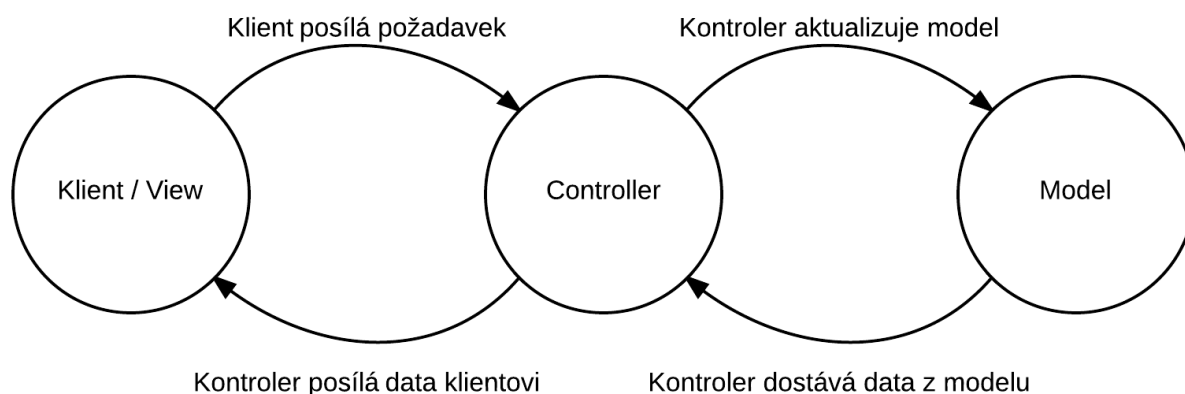
5.1.2 Web Api 2

Po rozhodnutí softwarové architektury služby bylo potřeba se zaměřit, na jaké platformě službu implementovat. Možností se vyskytuje nespočet. Základními z nich jsou Java a ASP.NET. Po krátkém porovnání možností obou základní platform byla zvolena platforma ASP.NET v jazyku C#. Hlavním důvodem pro vybrání právě ASP.NET byl framework Web Api ve verzi 2, pomocí kterého lze velmi jednoduše a rychle implementovat HTTP API. Web Api byl vyvinut za účelem stát se ideálním frameworkem pro implementaci RESTful API. Web Api také poskytuje výborné možnosti pro jednoduchou implementaci zabezpečení, autorizaci a autentizaci uživatele. Dalšími aspekty pro výběr ASP.NET byl výborný ORM framework a LINQ, které budou podrobněji popsány dále. [14]

5.1.3 Návrhový vzor MVC

MVC znázorněný na Obrázku 10 je návrhový vzor, který dělí aplikaci do tří základních komponent: [15]

- model – uchovávající data,
- controller – reagující na události od uživatele či klienta a aktualizující model,
- view – starající se o prezentaci dat z modelu uživateli.



Obrázek 10 Web Api MVC diagram

Web Api vychází z tohoto návrhového vzoru. V praxi to pak znamená, že požadavek od klienta zpracuje příslušný controller, který aktualizuje a načítá data z modelu a ty poté odesílá zpět klientovi, který data zobrazí ve formě vhodné pro uživatele, aby s nimi mohl dále pracovat. Při implementaci webové služby byl kladen velký důraz na dodržení konvence tohoto vzoru.

5.1.4 Entity Framework

Microsoft ADO.NET Entity Framework (EF) je framework objektově/relačního mapování (ORM), které umožňuje vývojářům pracovat s relačními daty stejně jako s klasickými objekty v OOP a odstraňuje potřebu psát klasické dlouhé kódy pro přístup do relační databáze za použití často složitých a nepřehledných SQL dotazů. I přesto je samozřejmě možné v případě

zájmu SQL dotazů využít. Při použití EF vývojáři k datům v databázi přistupují pomocí LINQ a poté s nimi manipulují jako s klasickými instancemi objektů. Entity Framework mimo jiné také poskytuje vývojářům předem implementované nástroje pro lazy loading¹², překlad dotazů a sledování změn, takže se vývojáři mohou zaměřit na logiku aplikace a odpadá nutnost řešit základní přístup k datům. [16] [17]

Nejdříve je nutné Entity Framework stáhnout a přidat jej do projektu. Naštěstí lze použít nástroj, který to velmi zjednodušuje. Ve Visual Studiu, které bylo pro implementaci webové služby použito, lze najít nástroj NuGet¹³. Ten slouží jako klientský nástroj nejen pro stahování balíčků z repozitářů. Hlavním nastaveným repozitářem je NuGet galerie, která je centrálním repozitářem, používaným všemi autory balíčků a zákazníky. Obsahuje velké množství balíčků, které stačí přes tento nástroj pouze stáhnout a ihned je možné je využívat v projektu. Stejně tak byl stáhnut Entity Framework a tím přidán do projektu. Jeho úspěšné stažení je možné zkontrolovat v záložce reference.

Samotný postup ORM mapování a vytváření databáze pomocí EF bude popsán v následující kapitole, která se zabývá tvorbou databáze. V této kapitole budou popsány ještě praktické ukázky připojení k databázi v kódu.

Ke správnému připojení k databázi je potřeba nastavit správný připojovací řetězec nazývaný *connection string*, který se nachází v souboru `Web.config`, kde je uloženo základní nastavení celé webové služby. Připojovací řetězec obsahuje přístupové údaje k databázi, jako název databáze, přihlašovací jméno a heslo, dále také název připojení, který se poté používá v kódu aplikace, a další potřebné věci, které se liší dle druhu databáze.

Ukázka připojovacího řetězce, pro vytvoření a připojení k lokální databázi:

```
<connectionStrings>
  <add name="DefaultConnection" connectionString="Data
Source=(localdb)\Projects;AttachDbFilename=|DataDirectory|\Nazev.mdf;
Initial Catalog=Nazev;Integrated Security=True"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

Tento řetězec zajistí při prvním spuštění služby vytvoření lokální databáze formátu `.mdf` ve složce `AppData` a dále k ní poskytuje přístup. Druhým příkladem je připojovací řetězec zajišťující vytvoření a připojení k databázi na serveru.

```
<connectionStrings>
  <add name="DefaultConnection"
connectionString="server=sql5.aspone.cz;uid=jmeno;pwd=heslo;database=
nazevDb;Integrated Security=False"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

Tyto připojovací řetězce nastaví Entity Framework automaticky v průvodci pro zakládání nové databáze, nebo připojování k již existující. Později je možné je měnit dle potřeby.

¹² <http://www.entityframeworktutorial.net/lazyloading-in-entity-framework.aspx>

¹³ <http://www.nuget.org/>

Samotný přístup k datům z databáze je implementován v daném controlleru. Při vytváření připojení k databázi se využívá bloku `using`, který zajistí, že instance databáze bude správně propuštěna v okamžiku, kdy jí již není třeba, aby dále zbytečně neblokovala zdroje. Pomocí bloku `using` tedy vývojář již nemusí myslet na správné propuštění instance, vše je zajištěno implicitně při opuštění bloku.

```
using (var db = new ApplicationDbContext())
{
    // zde se provádí operace s databází
    db.SaveChangesAsync(); // v případě změn v db, je nutné je uložit
}
```

Na prvním řádku je vytvořena instance objektu, který slouží pro připojení do databáze a umožňuje do ní přistupovat, jako by to byla běžná kolekce dat. Tento objekt byl automaticky vytvořen EF a v jeho nastavení je vyplněn název připojovacího řetězce `DefaultConnection` pro připojení ke správné databázi. V bloku `using` se přistupuje k databázi a při opuštění bloku je instance `db` propuštěna. Před opuštěním bloku je v případě jakýchkoliv provedených změn v databázovém kontextu nutné změny uložit, jinak se neprojeví v databázi.

5.1.5 Asynchronní přístup k databázi

Pro přístup do databáze bylo použito asynchronního přístupu. C# poskytuje velmi jednoduchou syntaxi pro asynchronní provádění operací pomocí dvou klíčových slov `async` a `await`. EF od verze 6 podporuje asynchronní přístup do databáze. Jako ukázka poslouží následující kód:

```
public async Task<IHttpActionResult> PutCustomerCompany( ... )
{
    ...
    var customers = await db.Customers.ToListAsync<Customer>();
    ...
}
```

Ve chvíli, kdy se přistupuje do databáze a čeká se na odezvu databáze, může vlákno obsluhovat jiný požadavek a ve chvíli, kdy se vrátí výsledky z databáze, vlákno přeruší svou aktuální činnost a bude pokračovat ve zpracovávání předchozího požadavku.

`Async` dovoluje v metodě použít klíčové slovo `await`, které říká, že následující metoda bude prováděna asynchronně. Jak konvence názvu značí, metoda `ToListAsync` se provede asynchronně. Protože je použit asynchronní přístup do databáze, musí být návratová hodnota nastavená jako `Task<datovýTyp>`.

5.1.6 Přístup k datům pomocí LINQ

LINQ je integrovaný jazyk pro dotazování nad kolekcemi dat, který byl představen v .NET verzi 3.5. Umožňuje snadno a rychle provádět dotazy nad kolekcemi dat. Ty následně třídit, propojovat a vyhledávat v nich. [18]

Na rozdíl od SQL dotazů je jeho zápis mnohem kratší, přehlednější a intuitivní. Jeho použití je velmi široké. Od vyhledávání v poli řetězců, listu objektů po vyhledávání v aplikačním kontextu databáze, ve spojení s EF.

Například dotaz pro vyhledání jednoho zákazníka v databázi dle jeho id vypadá následovně:

```
var customer = await db.Customers.Where(c => c.CustomerId == id)
    .Include(s => s.Address)
    .Include(s => s.Contact)
    .Include(s => s.Company)
    .FirstOrDefaultAsync<Customer>();
```

`db` je aplikační kontext databáze, popsany dříve, a `Customers` je kolekce všech zákazníků v databázi. Podmínka `Where`, jak název již sám napovídá, slouží ke specifikaci, o jakého zákazníka se jedná. Příkaz `Include` říká, že mají být z databáze také stažena data s adresou, kontaktem a firmou zákazníka. V případě, že by příkaz `Include` byl vynechán a později bychom chtěli k těmto datům přes instanci zákazníka přistupovat, musel by se provést nový dotaz do databáze a čekat na jeho zpracování. Jelikož je předem známo, že tato data budou v metodě potřeba, je vhodné využít příkazu `Include` a data načíst najednou. Poslední částí dotazu je příkaz `FirstOrDefaultAsync`, který říká, že má dotaz vybrat pouze jednoho zákazníka, a to asynchronním způsobem. Pro navrácení kolekce více zákazníků v listu, splňující podmínku v klauzuli `where`, se použije `ToListAsync`.

Zajímavou částí dotazu jsou lambda výrazy `c => c.CustomerId == id`. Jedná se o anonymní funkce, které se mohou použít k vytvoření delegátů nebo výrazů. Tyto výrazy byly přidány do jazyka C# zejména kvůli užitečnosti v LINQ.

Způsobů zápisu takového LINQ dotazu je několik, tento je asi nejvíce výstižný a na první pohled je vidět, co má dotaz za úkol. Zkráceným ekvivalentem pro načtení specifické objednávky, dle jejího id, může být tento zápis:

```
var order = await db.Orders.FirstOrDefaultAsync(o => o.OrderId == id);
```

Dotaz pro smazání objektu z databáze vypadá pak následovně:

```
db.Orders.Remove(order);
```

Dotaz pro přidání zákazníka do databáze, kde proměnná `customer` je nově vytvořená instance zákazníka:

```
db.Customers.Add(customer);
```

5.1.7 Navázání modelu

Navazování modelu, anglicky též *binding*, je činnost, kdy od klienta přijde XML nebo JSON objekt v těle požadavku a server se jej snaží rozebrat a správně poskládat do příslušného objektu, například zákazníka. O navazování se stará automaticky Web Api, které se snaží navázat objekt v parametrech dané URI.

```
public async Task<IHttpActionResult> PostCustomer(CustomerBindingModel
customerBinding)
{
    if (!ModelState.IsValid)
        return BadRequest(ModelState);
    ...
}
```

Po samotném navazování je nutné zkontrolovat validitu modelu. Jedná se o proces, který má zabránit klientovi v zasílání nevyžádaných dat. Například je velmi dobré zabránit tomu, aby uživatel s rolí `User`, při aktualizaci jeho entity, přidal do těla zprávy `"admin"="true"`, a to změnilo jeho roli v databázi. Samotná metoda `ModelState.IsValid` ovšem implicitně zajišťuje pouze kontrolu, zda jsou vyplněny všechny vlastnosti nastavené jako vyžadované a jejich omezení, jako například jméno delší než dva znaky. Nezajišťuje již dále, zda klient nepřidal něco navíc, co je v rozporu s logikou aplikace. Je proto nutné se o tuto validitu postarat samostatně. K řešení tohoto problému je možné využít tři základní možnosti.

- První možností je programovat proti rozhraní a v parametru nastavit typ modelu jako rozhraní, které obsahuje pouze vlastnosti, jaké jsou žádané.
- Druhou možností je použití anotací `[blacklist]` a `[whitelist]`, kde je možné nastavit, jaké vlastnosti model může či nemůže mít. Jak již z názvů plyne, `[blacklist]` se stará o vlastnosti, které model nesmí obsahovat, a `[whitelist]` naopak o vlastnosti, které model může obsahovat. Všeobecně je doporučeno používat spíše `[whitelist]`, protože při použití `[blacklist]` se může lehce stát, že zapomenou zakázat nějakou vlastnost.
- Třetí možností je potom vytvořit objekt výhradně pro bindování, který obsahuje pouze dovolené vlastnosti. Tato možnost byla použita v této práci.

5.1.8 Autorizace a autentizace uživatele

Nejdříve je důležité popsat rozdíl mezi autentizací a autorizací uživatele. Autentizace uživatele znamená, že server zná totožnost uživatele. Například když se přihlásí pomocí jména a hesla. Naproti tomu autorizace uživatele je rozhodování, zda má daný autentizovaný uživatel oprávnění k dané akci.

Po úspěšné autentizaci uživatele Web Api nastaví uživatele do globální proměnné `Thread.CurrentPrincipal`, skrze kterou je možné získat informace o uživateli z jakékoliv části aplikace. Jinak řečeno, pro každý požadavek se nastaví uživatel, který ho zaslal. Toto je standardní cesta v .NET. Pro ASP.NET se uživatel nastavuje jako `HttpContext.Current.User`. Viz následující ukázka:

```
private void SetPrincipal(IPrincipal principal)
{
    Thread.CurrentPrincipal = principal;
    if (HttpContext.Current != null)
    {
        HttpContext.Current.User = principal;
    }
}
```

Dalším krokem v pořadí po autentizaci uživatele jsou autorizační filtry. Autorizační filtry se spustí před zahájením funkce controlleru. Není-li požadavek autorizován, filtr vrátí chybovou odpověď a akce je zamítnuta.

Web Api poskytuje zabudovaný autorizační filtr, který kontroluje, zda má, či nemá uživatel oprávnění pro provedení dané akce. Pokud nemá, automaticky navrací stavový kód 401 Unauthorized a akci zamítne. Pro použití toho filtru se využívá atributů. Filtry můžeme aplikovat globálně na celý controller, nebo individuálně na každou akci zvlášť. Následující kód ukazuje globální použití autorizačního filtru na controller zákazníka, který říká, že pro přístup k akcím tohoto *controlleru* musí být uživatel autorizovaný:

```
[Authorize]
public class CustomersController : ApiController { ... }
```

V druhé ukázce je nastaven individuální filtr pro akci mazání zákazníka, který říká, že pro vyvolání této akce musí být uživatel autorizovaný a také musí mít nastavenou roli administrátor:

```
[Authorize(Roles = "Admin")]
public async Task<IHttpActionResult> DeleteCustomer(int id){ ... }
```

Dále atributy umožňují povolit neautorizovaný přístup k akci controlleru, který vyžaduje autorizaci, pomocí atributu `[AllowAnonymous]`, nebo také výčet povolených uživatelů `[Authorize(Users="Stepan,Honza")]`.

V této práci bylo využito Individual Accounts, které Web Api poskytuje od verze 2. Individual Accounts se starají o celý proces administrace uživatelů, včetně jejich rolí. Při zakládání projektu stačí pouze zaškrtnout autorizaci pomocí Individual Accounts a Visual Studio při zakládání projektu zařídí vytvoření potřebných tříd a také lokální databáze pro ukládání informací o účtech.

Individual Accounts používají pro autentizaci uživatele klasických Bearer tokenů¹⁴. Klient zašle žádost o token metodou POST na adresu `/api/Token`, v hlavičce požadavku nastaví formát těla následovně:

```
Content-Type: application/x-www-form-urlencoded
```

A v těle požadavku zašle své přístupové údaje v následujícím formátu:

```
grant_type=password&username=User0&password=Password0
```

Server vrátí Bearer token spolu s údaji o jeho vypršení, daty vystavení a dalšími informacemi. Klient poté pro každou další žádost musí do hlavičky požadavku připojit přijatý token a tím serveru prokáže svoji identitu:

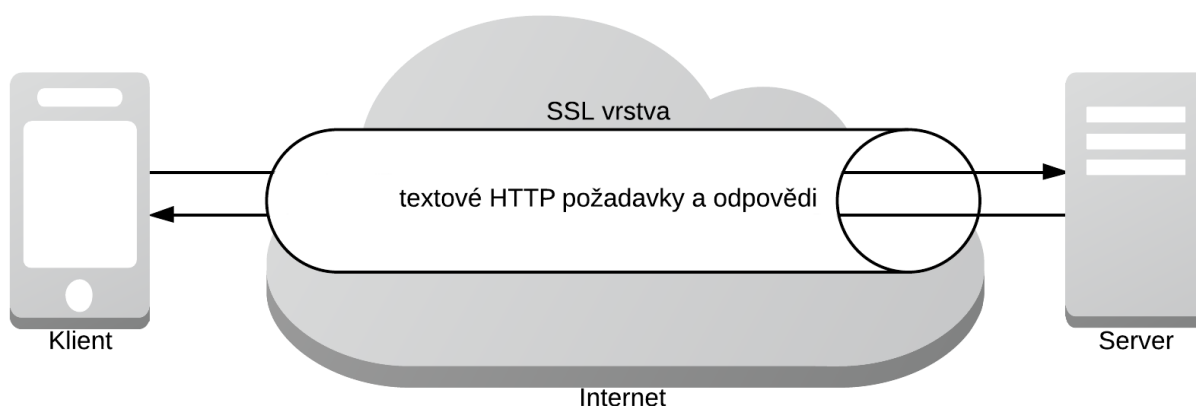
```
Authorization: Bearer mGFC5hnzIB1rBw3cz3PpRbcQv...
```

¹⁴ <http://stackoverflow.com/questions/5925954/what-are-bearer-tokens-and-token-type-in-oauth-2>

5.1.9 Zabezpečení

Protože klient se serverem komunikují skrze HTTP zprávy, které jsou v textové podobě posílány přes internet, je velmi nebezpečné touto cestou posílat citlivá data, jako například přihlašovací jméno a heslo při žádost o token nebo samotný požadavek, který obsahuje token v hlavičce.

Proto v této práci bylo zvoleno HTTPS spojení (Obrázek 11). V nastavení projektu stačí pouze nastavit možnost `SSL Enabled` na hodnotu `true` a poté správně nastavit SSL na straně serveru. Protože klient ví, ke kterému serveru se připojuje, a důvěřuje mu, není třeba SSL certifikát vystavený oficiální certifikační autoritou, který je placený. Stačí tedy jen nepodepsaný vygenerovaný certifikát, který je zdarma.



Obrázek 11 Zabezpečené HTTPS spojení

5.1.10 Ukládání do mezipaměti

Data se načtou z databáze a odešlou klientovi. Při příštím požadavku o ta samá data se zkontroluje, zda jsou stále validní, a pokud ano, místo přístupu do databáze se ihned odešlou data uložená v mezipaměti. Ukládání do mezipaměti je možné jak na straně serveru, tak na straně klienta. V této části bude popsáno pouze ukládání na straně serveru a ukládání na straně klienta bude popsáno v další podkapitole zabývající se implementací klienta. [19] [20]

K nastavení ukládání dat do mezipaměti na straně serveru byla použita knihovna `ASP.NET Web API CacheOutput`, dostupná na severu GitHub, kterou je možné stáhnout z NuGet. Možností nastavení ukládání je mnoho, a proto zde budou popsány jen ty základní možnosti nastavení, které byly použity při implementaci.

V následující ukázce je náhled toho, jak vypadá nastavení ukládání na straně serveru pro akci vrácení všech zákazníků:

```
[CacheOutput(ClientTimeSpan = 86400, ServerTimeSpan = 86400)]
public async Task<IHttpActionResult> GetCustomers() { ... }
```

Použitím atributu `CacheOutput` je zde nastaveno, aby server uložil výsledky do mezipaměti po dobu 86400 vteřin a aby spolu s výsledky odeslal klientovi informaci o tom, že je možné

data do mezipaměti uložit a používat je jako aktuální po stejnou dobu. Po vypršení tohoto intervalu nejsou již validní a je třeba je znovu načíst z databáze. V případě, že se data v databázi změní, musí se označit jako nevalidní, aby při další žádosti o tato data, byla načtena nová. Toto lze nastavit poměrně jednoduše atributem pro *controller*, jak je ukázáno v následujícím příkladu:

```
[AutoInvalidateCacheOutput]
public class CustomersController : ApiController{ ... }
```

Tato anotace značí, že pokud jsou v *controlleru* zavolány metody POST, PUT nebo DELETE, které data upravují, knihovna automaticky data v mezipaměti pro tento controller označí jako nevalidní. Dále je ještě možné nastavit data jako nevalidní explicitně za použití následujícího atributu:

```
[InvalidateCacheOutput("GetPost", typeof(PostsController))]
[InvalidateCacheOutput("GetPosts", typeof(PostsController))]
public async Task<IHttpActionResult> DeleteComment(int id){ ... }
```

Při spuštění akce `DeleteComment` se nastaví nejen uložená data v mezipaměti pro controller `Comments` jako nevalidní, ale také pro controller `Posts`. Toto je nutné v případě, že spolu data úzce souvisí a při změně dat jednoho controlleru se stávají nevalidní i data controlleru jiného.

5.2 Databáze

Vzhledem k již vybrané platformě pro implementaci webové služby byl výběr poměrně snadný. Přímou nabídkou databáze MSSQL, která je výborně propojena s frameworkem .NET a EF.

5.2.1 Možnosti tvorby databáze

Při využití ORM EF jsou celkem tři možnosti, jak vytvořit databázi.

- Database first – z již hotové databáze se vygeneruje logický model a odpovídající třídy v projektu.
- Model first – nejdříve jsou v logickém modelu pomocí grafického návrháře vytvořeny entity, vazby mezi nimi a ostatní náležitosti a poté se zvolí možnost generovat databázi z modelu. Nicméně výsledek nebude vygenerování databáze, jak by se mohlo zdát, nýbrž vygenerování DLL skriptu, který se spustí v databázi a vygeneruje vše potřebné.
- Code first – tento přístup je asi nejpřívětivější pro většinu vývojářů. Není zde nutnost pracovat s jakýmkoliv modelem nebo vazbami mezi entitami. Vývojář nadefinuje v databázovém kontextu entity jako klasické třídy a EF se poté postará o vygenerování správných entit a vazeb mezi nimi automaticky.

Při implementaci byl vybrán přístup *code first*. Důvodů k jeho vybrání bylo několik. Zaprvé tento postup je Microsoftem preferovaný. Zadruhé při použití Individual Accounts se tohoto přístupu využívá automaticky, kdy se z tříd starajících se o administraci uživatelů, vygeneruje příslušná databáze. Proto je velmi snadné tento kontext rozšířit o entity potřebné

k implementaci CRM systému. Základní uživatel z Individual Accounts, který obsahoval základní přihlašovací informace a role, byl podděděn a rozšířen o patřičné vlastnosti. [21]

V následujícím kódu je ukázka třídy modelu firmy:

```
public class Company
{
    public Company()
    {
        Customers = new List<Customer>();
        Changes = new List<Change>();
    }
    public int CompanyId { get; set; }
    [Required]
    public string Name { get; set; }
    public string Image { get; set; }
    public int? ContactId { get; set; }
    public int? AddressDataId { get; set; }
    public virtual Contact Contact { get; set; }
    public virtual AddressData Address { get; set; }
    public virtual ICollection<Customer> Customers { get; set; }
    public virtual ICollection<Change> Changes { get; set; }
}
```

Jedná se o jednoduchou třídu specifikující entitu firmy. Atribut `required` značí, které vlastnosti jsou vyžadovány. V databázové terminologii to znamená - `is not null`. Id firmy není nastaveno jako vyžadované, protože jej generuje databáze a není potřeba jej při vytváření nové instance vkládat ručně. Pomocí dalších atributů lze specifikovat další vlastnosti, jako například minimální délku řetězce jména a podobné. Zajímavou částí je objekt `Contact` spolu s kolekcemi `Customers` a `Changes`. Tímto se značí vazby mezi objekty. Jak je z kódu zřejmé, firma má jeden kontakt, jednu adresu a může obsahovat mnoho zákazníků. Otazník zde značí možnost - `is null`. Tím tedy říká, že vazba mezi firmou a kontaktem je 0:1, stejně tak mezi firmou a adresou.

5.2.2 Entity a vazby

V příloze A se nachází model finální databáze CRM systému. V modelu z důvodu přehlednosti chybí entita `Changes`, která má vazby s mnoha ostatními entitami. Tato entita udržuje historii změn nad určenými tabulkami, takže je později možné obnovit data nebo dohledat kdo danou změnu provedl a kdy.

5.2.3 Publikování databáze

Jedná se o vytvoření databáze na straně serveru. Publikování databáze je v případě *code first* automatické. Při prvním spuštění aplikace je databáze vygenerována na adrese, kterou obsahuje připojovací řetězec. V případě, že je připojovací řetězec správně nastaven, není potřeba nic dalšího nastavovat a export databáze na server proběhne automaticky. Případné změny v databázi se provádí podle historie migrací. Po každé změně v kódu modelů je třeba nejdříve změny uložit následujícím příkazem v *Package Manager Console*:

```
Add-Migration Init
```

Kde `init` je název prováděné změny. Dalším krokem po úspěšném uložení změn je jejich nahrání do databáze. K tomu slouží další velmi jednoduchý příkaz, který se znovu zadá do konzole:

```
Update-Database
```

5.3 Mobilní aplikace

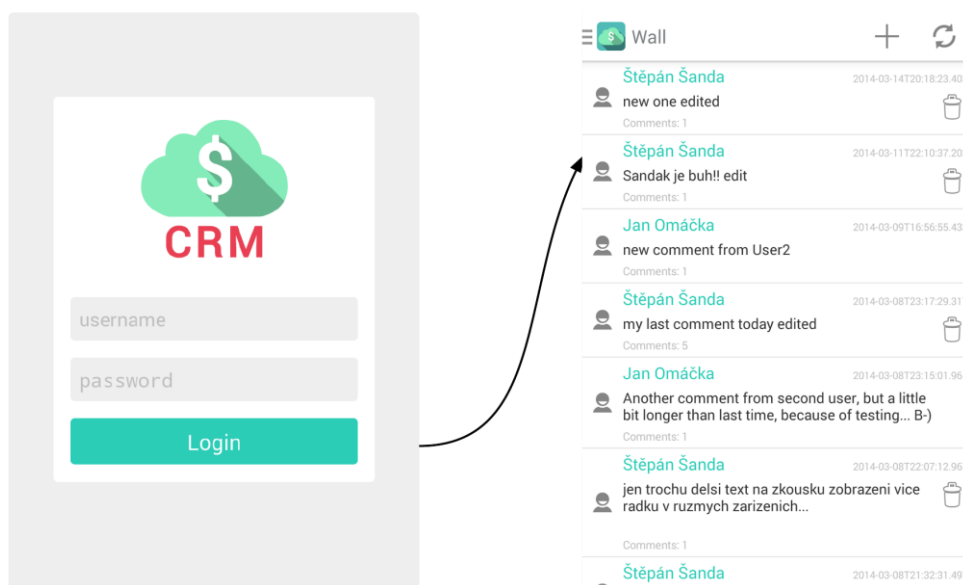
Jako mobilní platforma pro implementaci klienta byl vybrán OS Android, který poskytuje všechny vývojové nástroje zcela zdarma, a zařízení s tímto systémem jsou všeobecně cenově dostupná. Další výhodou je nativní programovací jazyk Java, který je velmi známý, a proto není potřeba se učit novým věcem. Díky tomu je možné v projektu využívat knihovny napsané v jazyce Java.

Pro přenos dat mezi serverem a klientem byl vybrán formát JSON, vzhledem k jeho jednoduchému zpracování na straně Androidu za použití knihovny GSON, která bude popsána níže. Webová služba implementovaná pomocí Web Api ponechává výběr na klientovi a umí přijímat a odesílat data jak v JSON tak XML formátu.

5.3.1 Obrazovky a navigace

Tvorba obrazovek a navigace v aplikaci je složitější proces, než se na první pohled zdá. Ovládání aplikace musí být pro uživatele logické a také je potřeba aplikaci otestovat pro různé velikosti obrazovek zařízení.

Jako hlavní obrazovka při prvním spuštění je přihlašovací obrazovka požadující přihlašovací údaje. Ta je zobrazena pouze při prvním spuštění, nebo pokud se uživatel odhlásí. Pokud jsou po spuštění aplikace načteny uložené přihlašovací údaje, zobrazí se jako první obrazovka `Wall`, která poskytuje výpis nových firemních zpráv. Přihlašovací obrazovka a obrazovka `Wall`, jsou zobrazeny na Obrázku 12.



Obrázek 12 Přihlašovací obrazovka

Pro navigaci v aplikaci se přímo nabízelo použít navigaci Navigation Drawer¹⁵. Ta poskytuje velmi intuitivní navigaci napříč celou aplikací, pokud poskytuje více než tři hlavní obrazovky. Návrh struktury navigace celé aplikace lze najít v příloze B. [22]

5.3.2 Fragmenty

Byly využity pro vytváření uživatelského rozhraní, kvůli optimalizaci pro různé velikosti obrazovek telefonů a tabletů. Využití fragmentů sice vyžaduje psaní více kódu, ale nabízí možnosti zobrazení různých rozložení obrazovky podle velikosti obrazovky a také při rotaci neztrácí uživatelsky vyplněná data.

5.3.3 Knihovna Volley

Jedná se o knihovnu vydanou společností Google v květnu 2013, která velmi usnadňuje práci se síťovou komunikací. Přestože Android ve svém SDK poskytuje vše potřebné k síťové komunikaci, je to poměrně zdlouhavý proces, v němž se vývojáři musí starat o spoustu věcí jako je například asynchronní zpracování požadavků. Volley je velmi moderní knihovna, která poskytuje vývojářům jednoduché nástroje, jež činí síťovou komunikaci v Android aplikacích jednodušší a rychlejší. Volley výborně odstiňuje vývojáře od nízkoúrovňových detailů HTTP komunikace a tím mu dovolí se soustředit na psaní hezkých a čistých RESTful HTTP požadavků. Navíc jsou všechny požadavky zpracovávány asynchronně v jiném vlákne, aniž by přitom blokoval hlavní vlákno, což je od verze Androidu 3.0 vyžadováno. Pokud se požadavek zpracovává v hlavním vlákne, čímž ho blokuje, Android bude reagovat vyhozením nepříjemné výjimky `android.os.NetworkOnMainThreadException`. Blokování hlavního vlákna síťovou komunikací má vážné následky v podobě nereagujícího či sekajícího se grafického uživatelského rozhraní a to může způsobit nechvalně známou hlášku - aplikace neodpovídá.

Hlavní výhody této knihovny jsou především vysokoúrovňová API, která umožňují vytvářet asynchronní RESTful HTTP požadavky, globální prioritní fronta požadavků, která umožňuje požadavkům nastavit jejich prioritu, poskytuje dobré API pro zrušení požadavků dle různých filtrů, a robustní zpracování ukládání dat do mezipaměti. [23]

Volley bohužel momentálně nemá téměř žádnou oficiální dokumentaci a tak je nutné se vyrovnat s tím, že je potřeba hledat návody a postupy na internetu od vývojářů, kteří již s knihovnou mají své zkušenosti.

Klíčové třídy Volley knihovny jsou:

- `RequestQueue` – fronta obsahující HTTP požadavky, které se musí zpracovat.
- `Request` – základní rodičovská třída všech požadavků, která obsahuje základní informace o požadavku, jako je například HTTP metoda. Od této třídy se dědí další více specifické třídy.
- `StringRequest` – HTTP požadavek, kde je očekávaná odpověď v podobě řetězce.

¹⁵ <https://developer.android.com/design/patterns/navigation-drawer.html>

- `JsonObjectRequest` – HTTP požadavek, kde je očekávaná odpověď v podobě JSON objektu.
- `JSONArrayObjectRequest` – HTTP požadavek, kde se očekává návratový typ v podobě pole JSON objektů.

Při implementaci aplikace, byly tyto základní třídy podděny a doplněny o autorizační hlavičku, která je potřeba pro autentizaci klienta. A tak tedy není nutné při každém požadavku z různých míst aplikace psát stejný kód a pokaždé vkládat autorizační hlavičku. Jako ukázka poslouží část kódu třídy `MyStringRequest`:

```
public class MyStringRequest extends StringRequest {
    ... // Konstruktory
    @Override
    public Map<String, String> getHeaders() throws AuthFailureError {
        Map<String, String> headers = new HashMap<String, String>();
        UserAccountInfo uac = UserAccountInfo.getInstance();
        headers.put("Content-Type", "text/html");
        if (uac != null) {
            headers.put("Authorization", uac.getmTokenType() + " " +
                uac.getmAccessToken());
        }
        return headers;
    }
}
```

V případě, že token nebyl nalezen, nebo již vypršel a aplikace má k dispozici uživatelské přihlašovací údaje, třída `VolleyController` automaticky zašle požadavek o nový token. To již řeší další třída `VolleyController`, která se jak již název napovídá, stará o zpracování všech požadavků. Tato třída byla implementovaná z důvodů zjednodušení základních činností při práci s HTTP požadavky. Poskytuje základní metody, které se starají o přijetí a zařazení požadavků do fronty, zrušení požadavků, získání či obnovu přístupového tokenu a poskytuje globální konstanty řetězců pro všechny požadavky.

Protože byl `VolleyController` implementován dle návrhového vzoru Singleton, který bude pospán níže, byla zajištěna jedna globální fronta požadavků v celém kontextu aplikace. Pro vložení požadavku do fronty poskytuje metodu `addToRequestQueue`, která přijme požadavek, nastaví token pro zrušení, za jak dlouho a kolikrát se má požadavek opakovat v případě neúspěchu a také možnost uložit výsledek do mezipaměti, je-li tak v odpovědi dovoleno:

```
public <T> void addToRequestQueue(Request<T> req, String tag) {
    req.setTag(TextUtils.isEmpty(tag) ? TAG : tag);
    req.setRetryPolicy(new DefaultRetryPolicy(20 * 1000, 1, 1.0f));
    req.setShouldCache(true);
    getRequestQueue().add(req);
}
```

Vložení požadavku do fronty odkudkoliv z celé aplikace poté vypadá následovně:

```
VolleyController.getInstance(getActivity()).addToRequestQueue(req, TAG);
```

Následuje ukázka globálních konstanty připojovacích řetězců k webové službě, které jsou potom přidány k URL každého požadavku a URL pro získání autorizačního tokenu:

```
public static final String TOKEN_URL="https://sandak.aspone.cz/bp/Token";
public static final String MAIN_URL ="https://sandak.aspone.cz/bp/api";
```

Metoda `cancelPendingRequests` slouží pro zrušení všech požadavků, které nebyly vyřízeny a odpovídají filtru v parametru metody. Objekt `tag` bývá obvykle unikátní řetězec označující danou aktivitu:

```
public void cancelPendingRequests(Object tag) {
    if (mRequestQueue != null)
        mRequestQueue.cancelAll(tag);
}
```

Zrušení požadavku probíhá nejčastěji při ukončování dané aktivity nebo pokud uživatel nechce již déle na odpověď čekat a žádost zruší tlačítkem zpět. Proto jsou ve fragmentech přepsány metody `onDestroyView` a `onKeyDown` pro zachycení stisku tlačítka zpět.

```
VolleyControler.getInstance(getActivity()).cancelPendingRequests(TAG);
```

Dále byla implementována třída pro překládání chyb požadavků do uživatelsky čitelného stavu. Třída s názvem `VolleyErrorHelper` poskytuje statickou metodu, která v parametru přijímá chybu, jež server poslal v odpovědi, tu zpracuje a vrátí řetězec vysvětlující význam chyby. V případě, že se jedná o chybu špatné autentizace, je automaticky zaslán požadavek o nový token.

```
public static String getMessage(Object error, Context context) {
    if(error instanceof TimeoutError) {
        return ....getString(R.string.generic_server_down);
    }else if (isServerProblem(error)) {
        return handleServerError(error, context);
    }else if (isNetworkProblem(error)) {
        return ....getString(R.string.no_internet);
    }
    return context.getResources().getString(R.string.generic_error);
}
```

O stavu požadavku, ať úspěchu či vrácené chybě, je uživatel informován pomocí toastů:

```
Toast.makeText(getActivity(), VolleyErrorHelper.getMessage(error,
getActivity()), Toast.LENGTH_SHORT).show();
```

5.3.4 Knihovna Gson

Jedná se o open source Java knihovnu vydanou společností Google, také známou jako Google Gson, která velmi zjednodušuje a automatizuje převod Java objektů do formátu JSON a obráceně z JSON řetězců vytvoří instance Java objektů. Gson také výborně podporuje genericitu a dědičnost Javy a umožňuje vynechání Java anotací, které bývají obvykle pro takovéto automatické převody nutností. [24]

Gson poskytuje pro převod dvě základní velmi intuitivní metody pro převod. A to `toJson` a `fromJson`. Jejich využití je velmi přímočaré a znázorňuje ho následující blok kódu:

```
Gson gson = new Gson();
Post post = gson.fromJson(response.toString(), Post.class);
```

Odpověď, která přišla z webové služby jako řetězec ve formátu JSON se pomocí instance `gson` převede na patřičný objekt a vytvoří jeho instance. K tomu, aby převod proběhl úspěšně, je potřeba aby třída `Post` splňovala základní pravidla.

```
public class Post implements Serializable {
    @SerializedName("Text")
    private String mText;
    @SerializedName("Username")
    private String mUsername;
    @SerializedName("FirstName")
    private String mFirstName;
    @SerializedName("LastName")
    private String mLastName;
    @SerializedName("Date")
    private String mDate;
    @SerializedName("PostId")
    private int mPostId;
    @SerializedName("Comments")
    private ArrayList<PostComment> mComments;
    public Post(){ ...}
    ... // Další metody třídy včetně get a set metod
}
```

První podmínkou je implementace rozhraní `Serializable`. Další podmínkou jsou správné proměnné třídy, jejichž názvy se musí shodovat s názvy v řetězci JSON, nebo se musí použít anotace, která udává, že daná proměnná se váže s daným názvem v JSON.

Zpracování odpovědi obsahující pole objektů je také velmi snadné:

```
Gson gson = new Gson();
Post[] posts = gson.fromJson(response.toString(), Post[].class);
PostsController postsControllerList = PostsController.getInstance();
for (int i = 0; i < posts.length; i++)
    postsControllerList.addPost(posts[i]);
```

Pro převod instance objektu do JSON řetězce se využívá metody `toJson` a pro vytvoření instance JSON objektu je třeba použití bloku odchyťavajícího výjimky při nezdařilém převodu:

```
Gson gson = new Gson();
String jsonString = gson.toJson(customer);
try{
    JSONObject jsonObject = new JSONObject(jsonString);
} catch (JSONException e) {
    e.printStackTrace();
}
```

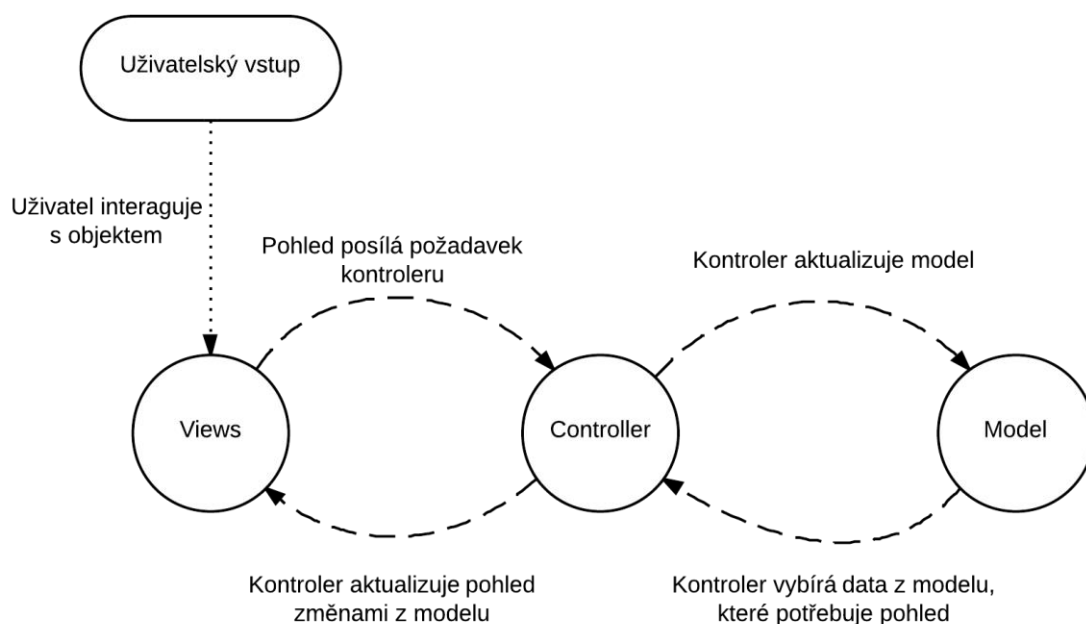
Pro pohodlnější práci s Volley požadavky a zpracování JSON převodu byla implementována třída `GsonRequest<T>` jako potomek třídy `Request<T>` z knihovny Volley. Tato třída automaticky konvertuje instanci předaného objektu na JSON řetězec a odpověď z JSON řetězce vytvoří instanci daného typu objektu, takže již není nutné převod provádět pokaždé ručně. Ukázka části její implementace se nachází v příloze C.

5.3.5 Implementace MVC v Androidu

Návrhový vzor byl již krátce popsán v předchozí kapitole zabývající se implementací webové služby. Tento návrhový vzor byl využit i pro implementaci mobilní aplikace CRM systému. V tomto případě se dají třídy MVC popsat následovně:

- Model je objekt, který drží data aplikace a *business* logiku. Třídy modelu obvykle představují reálné objekty, s nimiž aplikace pracuje, jako například zákazník, firma, objednávka a podobně.
- Pohled je objekt, který ví jak sám sebe vykreslit na obrazovku a jak reagovat na uživatelský vstup, jako dotek dané komponenty. Jednoduché pravidlo určující co pohled je, zní: „Pokud to uživatel může vidět na obrazovce, poté je to pohled.“
- Kontroler je objekt, který spojuje pohled s modelem. Obsahuje aplikační logiku. Je navržený tak, aby věděl, jak reagovat na různé zachycené události pohledem a jak zpracovávat data mezi modelem a pohledem. V Androidu se obvykle jedná o aktivity nebo fragmenty.

Obrázek 13 znázorňuje jak MVC model funguje v podání Androidu, například pokud je akce vyvolaná stisknutím tlačítka na obrazovce.



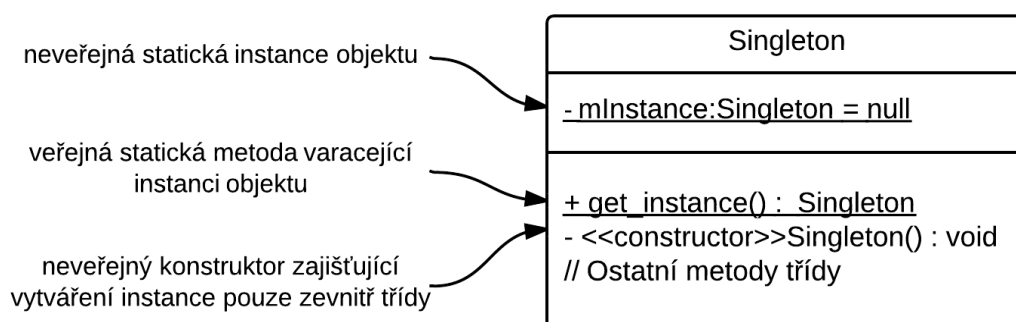
Obrázek 13 MVC diagram v podání Androidu

Využití MVC vzoru přináší několik výhod. Zejména ve větších projektech pomáhá velmi zpřehlednit kód aplikace. Odděluje kód do logických tříd, které pomáhají vývojářům navrhovat a rozumět aplikaci jako celku. Vývojář může nad aplikací myslet jako nad různými

vrstvami místo jednotlivými třídami. MVC také pomáhá znovupoužitelnosti tříd. Jelikož jsou třídy rozděleny do vrstev a třídy nejsou vázané na ostatní vrstvy, je mnohem snazší je znovu použít na jiném místě aplikace. [5]

5.3.6 Návrhový vzor Singleton

Návrhový vzor Singleton omezuje vytváření instance objektu pouze na jednu instanci. Toto je užitečné, pokud je požadováno, aby v celé aplikaci byla nejvíce jedna instance této třídy. Implementaci jedináčka zobrazuje UML diagram třídy `Singleton` na Obrázku 14. [15]



Obrázek 14 UML diagram návrhového vzoru Singleton

Tento návrhový vzor byl v aplikaci využit na několika místech, kde bylo logické vytvářet maximálně jednu instanci dané třídy napříč celou aplikací. Jedná se zejména o třídy spravující kolekce načtených objektů ze serveru, které je možné využívat z více míst aplikace.

Jako příklad lze například uvést třídu `Customers`, která spravuje kolekci všech zákazníků načtených ze serveru. Protože se se zákazníky pracuje na několika místech aplikace jako například v aktivitách `Customers`, `Visits`, `Orders` a `Companies`, je vhodné poskytovat pouze jednu globální instanci této třídy. Odpadá poté nutnost v ostatních aktivitách znovu načítat stejný obsah v případě, že již nějaká z předešlých aktivit zákazníky načetla.

5.3.7 Export do PDF souborů a uložení do Google Disku

Aplikace umožňuje provedené návštěvy a její objednávky exportovat do souboru PDF a uložit na firemní cloudový účet Google Disk. Pro export a naformátování PDF souboru byla použita knihovna `iText`¹⁶, která nabízí jednoduchou práci s PDF soubory. Pro tento účel slouží třída `PdfCreator`, která poskytuje metodu `makePDF` s parametrem přebírajícím konkrétní návštěvu. Nahrání uloženého PDF souboru do Google Disk¹⁷ byl trochu složitější proces. Nejdříve bylo nutné aplikaci registrovat v Google Developer Console¹⁸, kde bylo potřeba zadat vygenerované klíče pomocí konzolového nástroje Java `keytool`, a tím aplikace získá přihlašovací podepsaný certifikát.

¹⁶ <http://itextpdf.com/>

¹⁷ <https://drive.google.com/>

¹⁸ <https://console.developers.google.com/project>

```
keytool -exportcert -alias androiddebugkey -keystore path-to-debug-or-  
production-keystore -list -v
```

Dále bylo nutné pomocí správce SDK stáhnout příslušné balíčky *Google Play Services*, ty vložit do projektu a poté nastavit v manifestu oprávnění pro práci s Google Diskem. [25]

5.3.8 Záměr navigace, volání a emailu

Při zobrazení konkrétního zákazníka aplikace umožňuje v případě vyplněných informací volat či posílat SMS na zákaznickovo číslo, zaslat email či navigovat na jeho adresu. Toho bylo docíleno pomocí implicitních záměrů, které v Androidu umožňují spouštět ostatní aplikace podle typu žádosti. Instance zákazníka obsahuje ověřovací metody, které zpracovávají potřebná data a vrací je ve správném formátu pro předání při volání implicitního záměru. Například při volání či psaní SMS model navrácí správný formát řetězce čísla, které i kontroluje jeho správně zadanou předvolbu.

```
public String getCallMobileIntentString() {  
    String number = "tel:";  
    number += phoneVerification(getmMobAreaCode(), getmMobNumber());  
    if (number.length() < 9) return null;  
    return number;  
}
```

Záměr pro volání poté vypadá následovně:

```
if (mCustomer.getCallTelephoneIntentString() != null)  
    ImplicitIntents.makeCall(getActivity(),  
        mCustomer.getCallTelephoneIntentString());
```

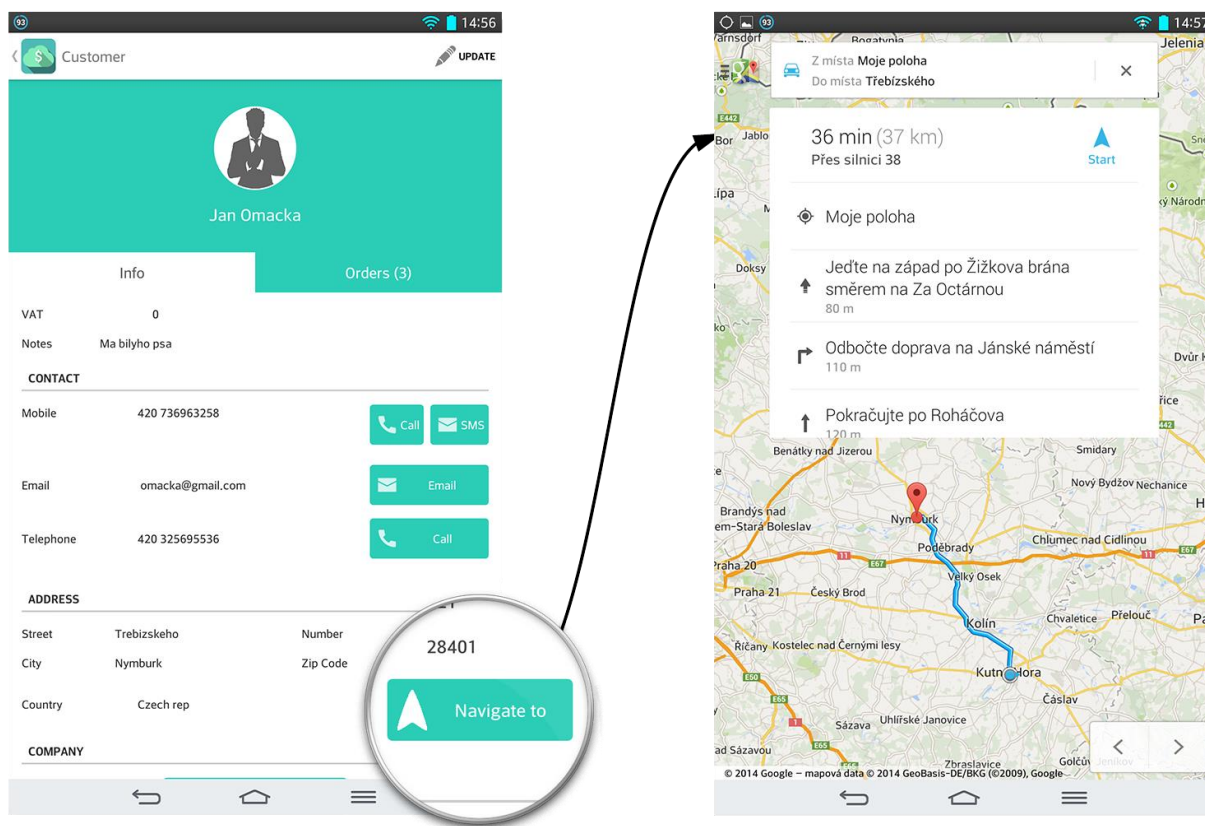
Správný formát záměru pro navigaci na danou adresu je trochu složitější. Aplikace musí správně naformátovat řetězec adresy zákazníka. O to se stará metoda `getNavigationAddressIntentString`. Pro navigaci byla využita aplikace Google Mapy, které jsou součástí každého Android zařízení. Na Obrázku 15 je vidět obrazovka detailu zákazníka s tlačítky pro volání, posílání SMS a emailů, a také zobrazení spuštění navigace s nastavenou adresou zákazníka.

5.4 Použité nástroje

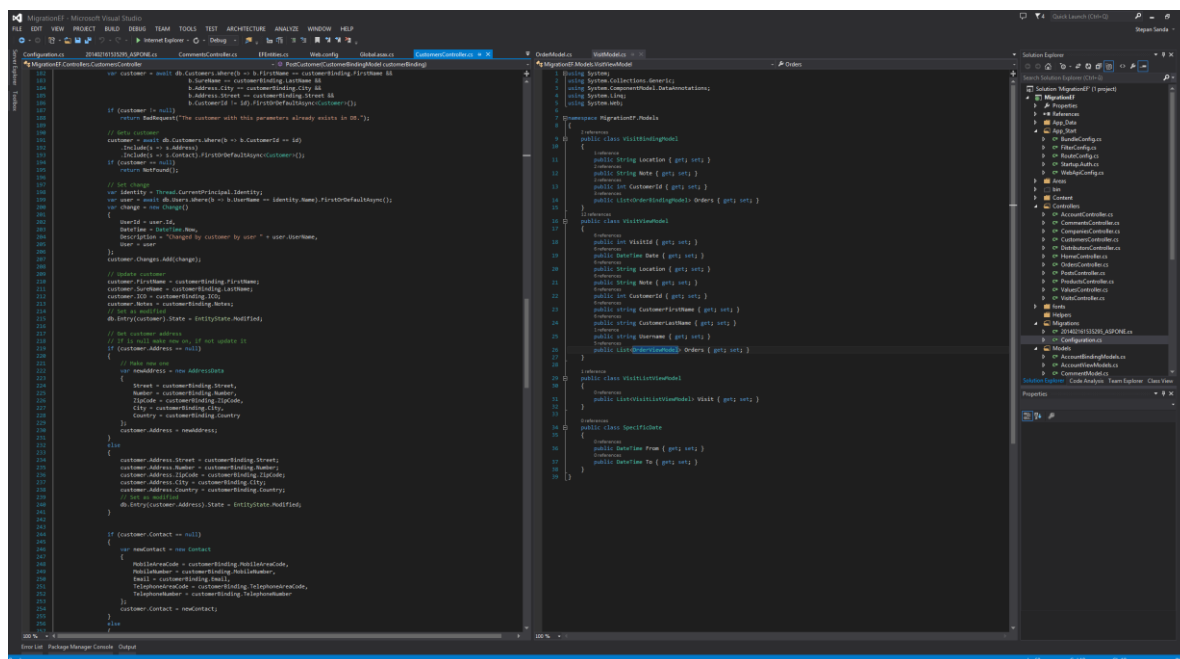
Pro vývoj aplikace bylo použito mnoha nástrojů. Zde budou uvedeny ty nejvýznamnější z nich spolu s jejich krátkým popisem.

Pro implementaci webové služby bylo využito vývojové prostředí Visual Studio 2013 (Obrázek 16). Jedná se o jedno z nejvyspělejších vývojových prostředí vůbec, které poskytuje obrovské množství nástrojů a doplňků pro usnadnění vývoje. Jako každé současné moderní vývojové prostředí poskytuje možnost nastavení tmavého vzhledu, díky kterému je možné pracovat dlouho dobu bez větší únavy očí. Výhodou je i možnost se přímo z prostředí připojit ke vzdálenému serveru či databázi nebo vytvořit vlastní lokální server pro testování služby. Díky přítomnosti nástroje NuGet, je přidání doplňku velmi jednoduché. Velmi se osvědčil

doplněk nazvaný ReSharper¹⁹, který vývojáři pomáhá a velmi dobře napovídá, jak správně psát, pojmenovávat a formátovat kód. Tím se kód často stává čitelnější, čistější i kratší.



Obrázek 15 Ukázka spuštění GPS navigace

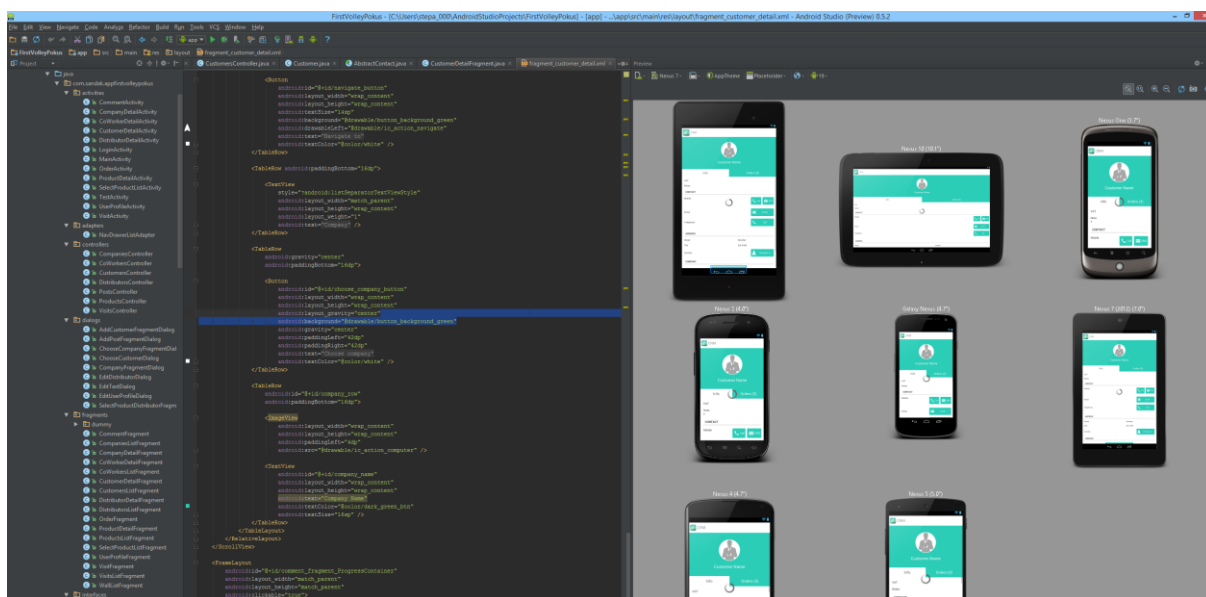


Obrázek 16 Ukázka IDE Visual Studio 2013

¹⁹ <http://www.jetbrains.com/resharper/>

Přestože k tvorbě databáze bylo využito přístupu *ccode first*, kde se databáze generuje automaticky podle kódu, bylo třeba databázi na serveru spravovat. A pro tyto účely bylo využito Microsoft SQL Server Management Studio, pomocí kterého se během testování mohla kontrolovat správnost dat v databázi, exportovat diagram databáze, upravovat, vkládat a mazat data. Exportovaný diagram databáze lze najít v příloze A.

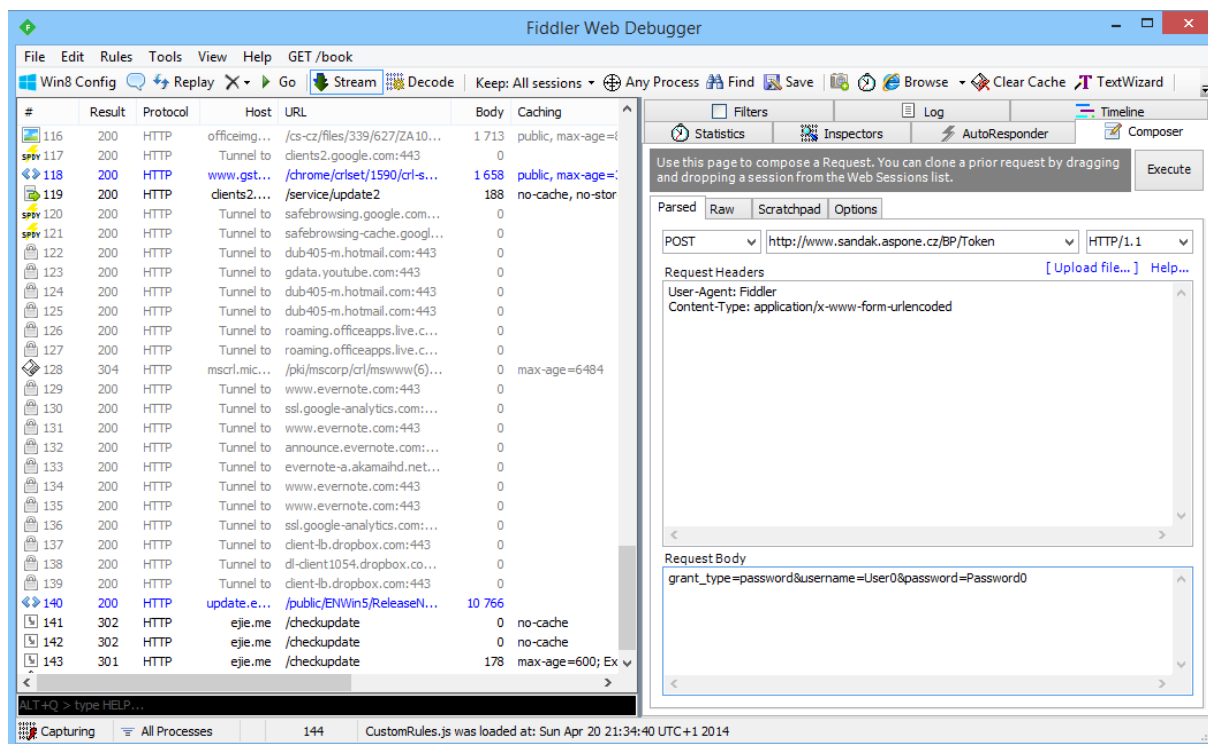
Pro vývoj mobilní Android aplikace bylo použito Android Studio aktuální verze 0.5.5 Preview (Obrázek 17). Jedná se sice stále o beta verzi, nicméně již plně funkční pro reálný vývoj. Za celý vývoj aplikace nenastal žádný problém či chyba. Jedná se o vývojové prostředí založené na IntelliJ IDEA, vývojovém prostředí pro jazyk Java. Android Studio je vyvíjeno přímo Googlem a ihned po jeho instalaci je možné začít psát aplikace. Není potřeba extra stahovat a instalovat Android SDK a další potřebné věci, vše je po instalaci k dispozici. Nutno podotknout, že prostředí je výborně přizpůsobeno vývoji pro Android. Obsahuje všechny potřebné funkce, výborný návrh XML layoutů, šablony pro aktivity, fragmenty a další části Androidu. Stejně jako Visual Studio 2013 poskytuje tmavý vzhled, který ulehčí očím při delší práci.



Obrázek 17 Ukázka IDE Android Studio

Pro testování HTTP požadavků a odpovědí byl použit nástroj Fiddler²⁰ (Obrázek 18). Jedná se o nástroj, který je poskytovaný zdarma a používá se zejména pro sledování síťové komunikace. Při vývoji byl využíván zejména ve chvíli vývoje webové služby, kdy ještě nebyl k dispozici klient, který by její správnou funkčnost mohl otestovat. Dále byl také použit pro testování komunikace mezi klientem a serverem. Jelikož Fiddler poskytuje i proxy, stačilo na mobilním zařízení nastavit pod stejnou síť proxy adresu Fiddleru a poté bylo možné veškerou komunikaci mezi službou a klientem sledovat. To značně urychlilo celý vývoj, jelikož některé chyby odhalit bylo velmi snadné.

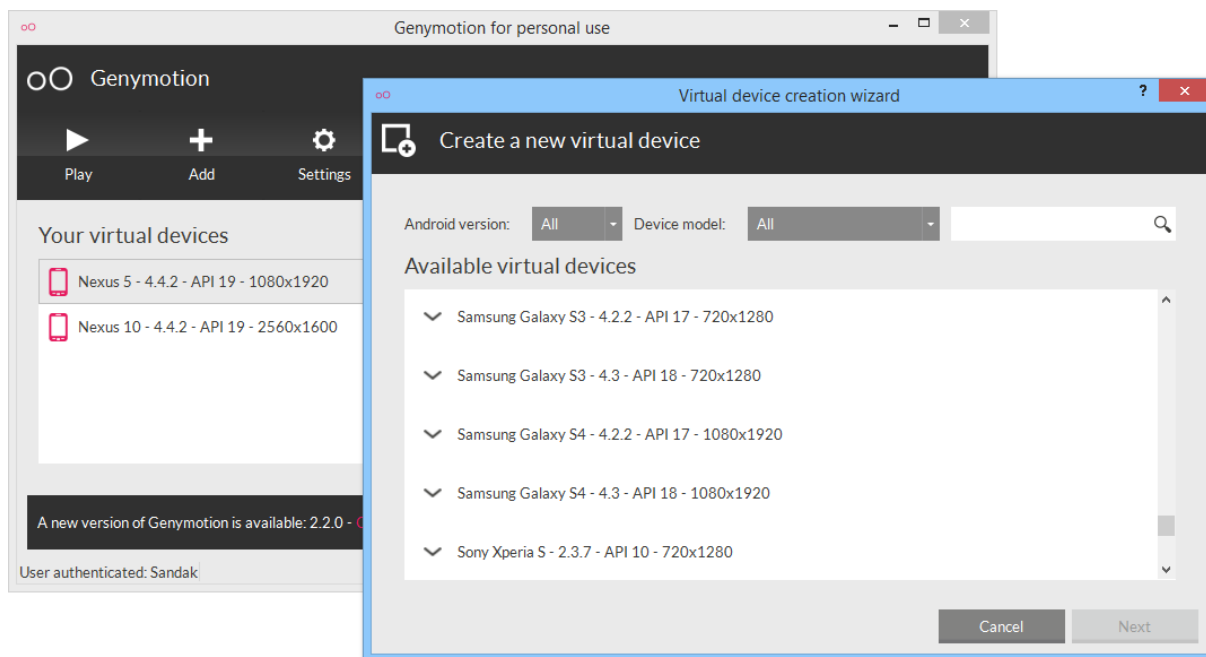
²⁰ <http://www.telerik.com/fiddler>



Obrázek 18 Fiddler

V některých případech byl vzhledem k nedostatku HW zařízení pro testování nutné využít Android emulátor. Zejména pro test rozvržení obrazovek na různě velkých obrazovkách. Obyčejný Android emulátor, který je součástí Android Studio, není moc použitelný. I přes poměrně silnou HW výbavu počítače, na kterém vývoj probíhal, 8Gb RAM a přítomnost SSD disku trvá spuštění emulátoru okolo deseti minut a jeho reakce jsou velmi pomalé. Naštěstí existuje nástroj GenyMotion²¹ (Obrázek 19), který obyčejný emulátor nahrazuje. Jedná se o poměrně dobře zpracovaný nástroj, který poskytuje mnoho užitečných nástrojů, jako je například testování chování aplikace při vybité baterii nebo testování GPS signálu a podobné. GenyMotion využívá pro virtualizaci Android zařízení poměrně známý VirtualBox, program sloužící pro virtualizaci operačních systémů. Na úvodní obrazovce programu je možné vytvářet nová zařízení buď dle existujících zařízení na trhu, nebo je možné zadat specifické HW a SW vlastnosti.

²¹ <http://www.genymotion.com/>



Obrázek 19 Ukázka programu GenyMotion

Mezi další použité nástroje patří cloudový zápisník Evernote²², který sloužil pro psaní poznámek, ukládání nápadů, chyb k vyřešení a také JSON objektů, pro rychlejší testování. Dále pak Git pro verzování a ukládání projektu na serveru Bitbucket²³, který poskytuje studentským účtům zdarma i privátní repozitáře. Paralelně s GitHub byl pro zálohování celých složek projektů a důležitých dokumentů použit cloud DropBox²⁴. Posledním zde uvedeným nástrojem je Chrome aplikace pro tvorbu diagramů LucidChart²⁵. Tento program byl využit pro kreslení většiny zde uvedených diagramů a obrázků.

²² <https://evernote.com/>

²³ <https://bitbucket.org/>

²⁴ <https://www.dropbox.com/>

²⁵ <https://www.lucidchart.com/>

6 Testování

Během celého vývoje systému bylo nutné všechny jeho části postupně testovat. V této kapitole bude podrobněji popsán průběh testování.

6.1 Webová služba

Webová služba byla nejdříve vyvíjena pouze na lokálním serveru pro testování. Později byl ve vánoční 50% slevě zakoupen roční základní účet na serveru Aspone.cz, který umožňuje připojení k databázi pomocí SQL Server Management Studia.

Pro testování služby bylo potřeba klienta. Jelikož ještě klient nebyl hotový, byl využit program Fiddler popsáný v předchozí kapitole. S jeho pomocí bylo možné simulovat reálnou HTTP komunikaci, včetně metod, hlaviček, těla a podrobné informace o odpovědi.

Při testování na lokálním serveru byly odezvy služby velmi rychlé. Při publikování služby na server byla odezva velmi různorodá a v průměru znatelně pomalejší. Po delším testování, byl vyloučen vliv rychlosti internetu na přenos, jelikož byly přenášeny velice malé bloky dat, a také často služba reagovala velmi rychle. Po dlouhém experimentování bylo zjištěno, že webová služba je poměrně složitější a její spuštění na serveru zabere mnoho HW zdrojů, hlavně paměti RAM. Jelikož byl zakoupen pouze běžný účet, který nabízí pouze 100Mb RAM a aplikace při základním vytížení dosahuje běžně okolo 350Mb RAM, což je běžné, nebylo možné dosáhnout takového snížení náročnosti služby pomocí základní optimalizace. Aplikace po spuštění pomalu zvyšovala své nároky na zdroje dle přijatých požadavků a ve chvíli, kdy přesáhla daný limit, byla automaticky restartována a klient poté musel čekat na její znovuspuštění. Naštěstí ve webovém nastavení zakoupeného hostingu bylo možné zaškrtnout políčko povolující 32bitový režim. Tím se náročnost aplikace snížila při běžném provozu na 90Mb RAM, což je přijatelný výsledek.

6.2 Android aplikace

Android aplikace je nejlepší testovat přímo na reálných zařízeních, protože emulátor je v tomto případě velmi nespolehlivý a přináší více problémů než užitku. Pokud ovšem není k dispozici dostatek reálných zařízení pro testování, je to jediná volba jak aplikaci otestovat pro různé velikosti displejů a různé HW vybavení. Jak bylo popsáno v předchozí kapitole, k těmto účelům byl využit emulátor GenyMotion, který pro toto určení fungoval výborně. Bylo zde vytvořeno několik mobilních telefonů a tabletů lišících se HW i SW parametry.

Pro testování na reálných zařízeních byl k dispozici 4" mobilní telefon Google Nexus 4 s verzí Androidu 4.4 KitKat a 8" tablet LG G Pad, taktéž s Androidem 4.4 KitKat. Protože tablet neobsahuje 3G modul, bylo možné jeho testování pouze s připojením k lokální WiFi nebo pomocí hotspotu vytvořeného na telefonu. Při testování na WiF, 3G i Edge připojení, byla rychlost odezvy dostatečně rychlá, aby neobtěžovala uživatele.

6.3 Reálné testování

V poslední části vývoje byl celý CRM systém na týden zapůjčen jedné firmě, která o podobný produkt měla zájem, na otestování v praxi. Produkt testovalo celkem čtrnáct zaměstnanců na tabletech Nexus 7 verze 2012 32Gb s 3G modulem a Androidem 4.4 KitKat. Do databáze bylo nahráno několik základních produktů jejich nabídky a vytvořeny uživatelské účty pro všechny testovací pracovníky.

Na konci týdne vedoucí zaslal dojmy a připomínky od zaměstnanců. Bylo zjištěno především pár drobných problémů v rozložení obrazovek aplikace a u některých obrazovek problémy při rotaci obrazovky. Jedním z hlavních nalezených problémů byly potíže s diakritikou. Ukázalo se, že Volley knihovna při nastaveném ukládání do mezipaměti volí špatné kódování. Běžně je v odpovědi od serveru kódování UTF-8 určeno, ovšem v případě, že server místo odpovědi navrací kód 304 Not Modified, kde již kódování není určeno, Volley zvolí automatické kódování.

7 ZÁVĚR

Cílem této práce bylo vytvořit základní CRM systém pro správu zákazníků, který lze používat na mobilních telefonech a tabletech. Ze začátku se toto zadání nezdálo moc komplikované. Postupem implementace začínal projekt velmi narůstat a nabíral na své komplexnosti. Pro správnou funkčnost a reálné nasazení bylo potřeba vyřešit mnoho problémů, které na začátku práce nebyly příliš zjevné. Protože jsem na začátku neměl téměř žádné znalosti potřebné pro implementaci webových služeb nebo programování pro OS Android, bylo pro mě toto zadání velkou výzvou a hlavně zajímavou příležitostí naučit se novým věcem.

Projekt byl rozdělen do tří základních částí, které na sebe navazovaly. První částí byl návrh funkčnosti samotného CRM systému, jeho logických celků a databáze potřebné pro ukládání veškerých dat. Pro inspiraci funkcí a možností systému bylo vyzkoušeno několik tuzemských CRM systémů, které nabízí zkušební verzi. Po promyšlení logické funkčnosti systému, byl navrhnout model databáze, kde byly vytyčeny potřebné entity, jejich vlastnosti a vazby mezi nimi.

Následovala implementace webové služby. Po výběru použité technologie pro implementaci se mohlo začít se samotnou implementací. V první řadě bylo řešeno zabezpečení, autorizace a autentizace uživatele. Nejdříve byla snaha o napsání vlastních autorizačních modulů za pomoci RSA asymetrického šifrování s neveřejným a veřejným klíčem. Po jeho implementaci fungovala autentizace uživatele výborně. Nicméně poté byla objevena nová možnost autorizace a autentizace uživatele za pomoci Individual Accounts, které poskytuje Microsoft v nové verzi .NET, od verze Visual Studio 2013. Vzhledem k velmi snadné implementaci byla ručně napsaná autorizace nahrazena Individual Accounts. Po vyřešení autentizace a autorizace uživatele bylo možné začít s implementací funkcí webové služby. Během implementace se projekt pomalu rozrůstal, až bylo časem nutné provést menší refaktorování pro lepší údržbu a přehlednost kódu. Následně po zhotovení webové služby byl zakoupen webový hosting, kam se poté služba publikovala.

Posledním krokem byla implementace samotného klienta pro mobilní platformu Android. Zde bylo využito návrhových vzorů MVC a Singleton, které velmi usnadnily vývoj. Při implementaci byl kladen velký důraz na správné a jednoduché rozvržení obrazovek, aby ovládání bylo pro uživatele snadné a logické.

Veškeré cíle zadání práce byly splněny a vznikl funkční produkt se základní funkčností, který lze použít v praxi bez větších omezení. Stále se však nepodařilo úplně eliminovat problémy s diakritikou při použití mezipaměti a současné řešení je pouze dočasné. Bylo by také vhodné zakoupit sdílený hosting, který bude poskytovat lepší HW zdroje, zejména paměť RAM. Je zde velký prostor pro budoucí rozšiřování funkčnosti a optimalizaci aplikace. Zejména možnost rozšíření o statistiky a grafy prodeje, plánování schůzek a akcí, řádné administrátorské rozhraní a klienta pro import produktů z produktových katalogů distributorů. To jsou ovšem již věci nad rámec zadání této práce, které nabízejí možnost pro její budoucí rozšíření.

8 Literatura

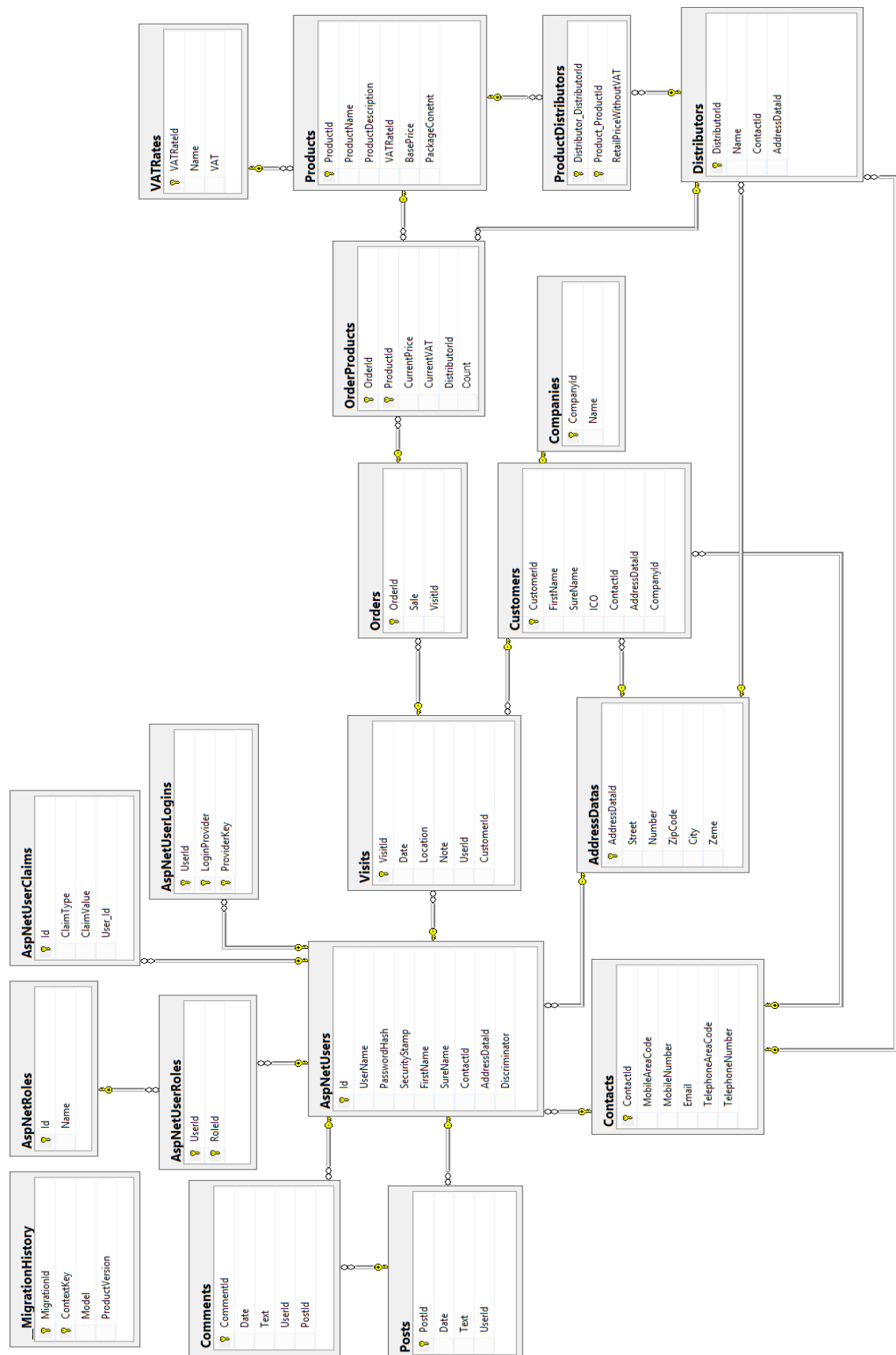
- [1] DYCHÉ, Jill. *The CRM handbook: a business guide to customer relationship management*. Boston: Addison Wesley, c2002, xxiv, 307 p. ISBN 02-017-3062-6.
- [2] FINGAS, JON. *Android climbed to 79 percent of smartphone market share in 2013, but its growth has slowed* [online]. 2014 [cit. 2014-04-21]. Dostupné z: <http://www.engadget.com/2014/01/29/strategy-analytics-2013-smartphone-share/>
- [3] MYSLIVEČEK, David. *Krátké ohlédnutí za historií Androidu* [online]. 2013 [cit. 2014-04-21]. Dostupné z: <http://www.svetandroida.cz/kratke-ohljedniti-za-historii-androidu-201305>
- [4] VOGEL, Lars. *Android Development - Tutorial: Based on Android 4.4* [online]. 2009, 2014 [cit. 2014-04-21]. Dostupné z: http://www.vogella.com/tutorials/Android/article.html#components_fragments
- [5] HARDY, Brian. *Android programming: the big nerd ranch guide* [online]. 1st ed. Atlanta, GA: Big Nerd Ranch, 2013, p. cm. [cit. 2014-04-21]. ISBN 03-218-0433-3.
- [6] GOOGLE. *Design, Develop and Distribute* [online]. 2014 [cit. 2014-04-21]. Dostupné z: <http://developer.android.com/>
- [7] W3SCHOOLS. *SOAP: Introduction* [online]. © 2014 [cit. 2014-04-21]. Dostupné z: http://www.w3schools.com/Webservices/ws_soap_syntax.asp
- [8] *SOAP Example* [online]. 2014 [cit. 2014-04-21]. Dostupné z: http://www.w3schools.com/Webservices/ws_soap_example.asp
- [9] MALÝ, Martin. *REST: architektura pro webové API* [online]. 2009 [cit. 2014-04-21]. Dostupné z: <http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/>
- [10] THIJSEN, Joshua. *The RESTful CookBook: How to do stuff RESTful* [online]. 2012 [cit. 2014-04-21]. Dostupné z: <http://restcookbook.com/>
- [11] FREDRICH, Todd. PEARSON ECOLLEGE. *RESTful Service Best Practices: Recommendations for Creating Web Services* [online]. 2013 [cit. 2014-04-21]. Dostupné z: <https://s3.amazonaws.com/tfpearsonecollege/bestpractices/RESTful+Best+Practices.pdf>
- [12] SKONNARD, Aaron. MICROSOFT. *A Guide to Designing and Building RESTful Web Services with WCF 3.5* [online]. 2008 [cit. 2014-04-21]. Dostupné z: <http://msdn.microsoft.com/en-us/library/dd203052.aspx>
- [13] ECOLLEGE. *Learn REST: A RESTful Tutorial* [online]. 2012 [cit. 2014-04-21]. Dostupné z: <http://www.restapitutorial.com/>
- [14] WASSON, Mike. MICROSOFT. *The basics of building an HTTP service using ASP.NET Web API* [online]. [cit. 2014-04-21]. Dostupné z: <http://www.asp.net/web-api/overview>
- [15] FREEMAN, Eric a Elisabeth FREEMAN. *Head first design patterns*. 1st ed. Sebastopol: O'Reilly, 2004, xxxvi, 638 s. ISBN 05-960-0712-4.
- [16] MICROSOFT. *Entity Framework* [online]. 2014 [cit. 2014-04-21]. Dostupné z: <http://www.asp.net/entity-framework>
- [17] *Entity Framework Tutorial* [online]. © 2014 [cit. 2014-04-25]. Dostupné z: <http://www.entityframeworktutorial.net/>

- [18] ALBAHARI, Joseph, Ben ALBAHARI a Peter DRAYTON. *C# 5.0 in a nutshell*. 5th ed. Sebastopol: O'Reilly, c2012, xvi, 1042 p. In a nutshell O'Reilly. ISBN 978-144-9320-102.
- [19] NOTTINGHAM, Mark. *Caching Tutorial* [online]. © 2014 [cit. 2014-04-21]. Dostupné z: <http://www.jakpsatweb.cz/clanky/caching-tutorial-czech-translation.html>
- [20] TOMAYKO, Ryan. *Things Caches Do* [online]. 2008 [cit. 2014-04-21]. Dostupné z: <http://tomayko.com/writings/things-caches-do>
- [21] ATTEN, John. *Configuring Db Connection and Code First Migration for Identity Accounts in ASP.NET MVC 5 and Visual Studio 2013*. [online]. 2013 [cit. 2014-04-21]. Dostupné z: <http://typecastexception.com/post/2013/10/27/Configuring-Db-Connection-and-Code-First-Migration-for-Identity-Accounts-in-ASPNET-MVC-5-and-Visual-Studio-2013.aspx#Configuring-Entity-Framework-Migrations-and-Seeding-the-Database>
- [22] TAMADA, Ravi. *Android Sliding Menu using Navigation Drawer* [online]. 2013 [cit. 2014-04-21]. Dostupné z: <http://www.androidhive.info/2013/11/android-sliding-menu-using-navigation-drawer/>
- [23] CHAKRABORTY, Arnab. *Asynchronous HTTP Requests in Android Using Volley* [online]. 2013 [cit. 2014-04-21]. Dostupné z: <http://arnab.ch/blog/2013/08/asynchronous-http-requests-in-android-using-volley/>
- [24] SINGH, Inderjeet, Joel LEITCH a Jesse WILSON. *Gson User Guide* [online]. [cit. 2014-04-21]. Dostupné z: <https://sites.google.com/site/gson/gson-user-guide#TOC-Object-Examples>
- [25] GOOGLE. *Google Drive Android API* [online]. 2014 [cit. 2014-04-21]. Dostupné z: <https://developers.google.com/drive/android/>

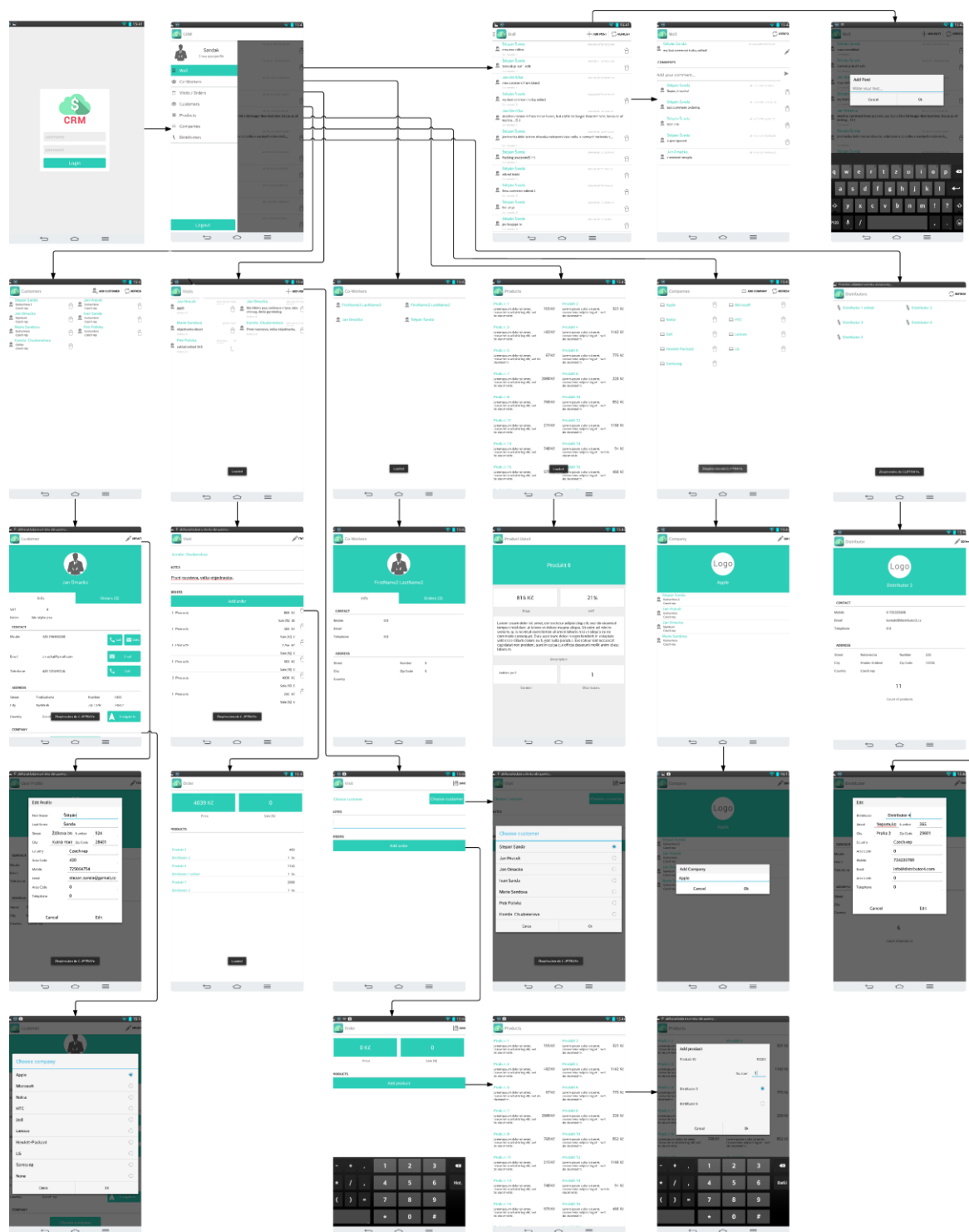
9 Přílohy

Příloha A <i>Model databáze</i>	63
Příloha B <i>Struktura navigace mobilní aplikace</i>	64
Příloha C <i>Zdrojový kód třídy GsonRequest.java</i>	65
Příloha D <i>Ukázka obrazovek mobilní aplikace</i>	66

Příloha A *Model databáze*



Příloha B *Struktura navigace mobilní aplikace*



Příloha C Zdrojový kód třídy *GsonRequest.java*

```
public class GsonRequest<T> extends Request<T>{
    private Gson mGson;
    private Class<T> mClass;
    private Response.Listener<T> mListener;

    public GsonRequest(int method, String url, Class<T> cls,
        Response.Listener<T> listener, Response.ErrorListener errorListener){
        super(method, url, errorListener);
        mGson = new Gson();
        mClass = cls;
        mListener = listener;
    }

    @Override
    protected Response<T> parseNetworkResponse(NetworkResponse response){
        try {
            String jsonString = new String(response.data,
                HttpHeaderParser.parseCharset(response.headers));
            T parsedGSON = mGson.fromJson(jsonString, mClass);
            return Response.success(parsedGSON,
                HttpHeaderParser.parseCacheHeaders(response));
        } catch (UnsupportedEncodingException e) {
            return Response.error(new ParseError(e));
        } catch (JsonSyntaxException je) {
            return Response.error(new ParseError(je));
        }
    }

    @Override
    protected void deliverResponse(T t){
        mListener.onResponse(t);
    }
    ...
}
```

Příloha D Ukázka obrazovek mobilní aplikace

