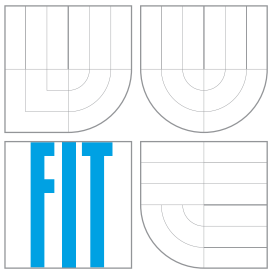
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# MOBILNÍ APLIKACE PRO PODPORU ŘÍZENÍ DOPRAVNÍCH STAVEB

MOBILE APPLICATION SUPPORTING MANAGEMENT OF TRAFFIC CONSTRUCTIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. RADEK OBOŘIL

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. ADAM HEROUT, Ph.D.

BRNO 2015

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav počítačové grafiky a multimédií Akademický rok 2014/2015

# Zadání diplomové práce

Řešitel: **Obořil Radek, Bc.**
Obor: Bezpečnost informačních technologií
Téma: **Mobilní aplikace pro podporu řízení dopravních staveb**
**Mobile Application Supporting Management of Traffic Constructions**
Kategorie: Uživatelská rozhraní

Pokyny:

1. Vyhledejte a zhodnoťte existující aplikace pro podporu projektového řízení v terénu.
2. Analyzujte požadavky na aplikaci pro podporu řízení dopravních staveb. Zaměřte se na precizní dokumentaci zakázek (nových i dokončených-nedokumentovaných), v terénu pomocí GPS, map a fotoaparátu. Zohledněte možnost práce v offline režimu a odloženou synchronizaci se serverem. Zahrňte možnost prohlížet data ze serveru.
3. Navrhněte dílčí prvky uživatelského rozhraní a dílčí řešení synchronizace se serverem. Ověřte jejich použitelnost pomocí experimentů.
4. Integrujte navržená dílčí řešení do jedné aplikace nasaditelné v praxi pro dokumentování dopravních zakázek.
5. Testujte vytvořenou aplikaci na dostatečné množině uživatelů.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování výsledků projektu.

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Bez požadavků.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Herout Adam, doc. Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2014
Datum odevzdání: 27. května 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
*vedoucí ústavu*

# Abstrakt

Tato práce se zabývá vývojem aplikace pro podporu řízení dopravních staveb. Výsledné řešení se skládá z mobilní aplikace a serverové části. Mobilní aplikace umožňuje pomocí integrovaného fotoaparátu, map a GPS precizně dokumentovat průběh dopravních staveb v terénu a nasbíraná data odesílat na server. Serverová část zahrnuje webovou službu pro komunikaci s klienty a webovou aplikaci pro správu databáze a zpracování nabíraných dat. Kromě návrhu a implementace jsou diskutovány využité vlastnosti a technologie mobilních zařízení. Práce se také zabývá současnými nejvýznamnějšími mobilní platformami a rozebírá koncepty a technologie multiplatformního vývoje, zejména nástroj Xamarin a framework MvvmCross, které byly použity při vývoji aplikace.



# Abstract

This thesis deals with development of an application supporting management of traffic constructions. The resulting solution consists of a mobile application and a server application. The mobile application allows users to document progress of a construction using integrated camera, maps and GPS, and send collected data to the server. The server application includes a web service for client-server communication, and a web application for database management and processing of the collected data. Apart from the design and implementation, the features and technologies of mobile devices utilized by the application are discussed. The thesis also reviews current most significant mobile platforms and analyses concepts and technologies of cross-platform development, especially Xamarin and MvvmCross, which were used for development of the above-mentioned application.



# Klíčová slova

Mobilní aplikace, webová služba, podpora dopravních staveb, multiplaformní vývoj, Android, iOS, Windows Phone, Xamarin, MvvmCross

# Keywords

Mobile application, web service, road construction, cross-platform, Android, iOS, Windows Phone, Xamarin, MvvmCross

# Citace

Radek Obořil: Mobilní aplikace pro podporu řízení
dopravních staveb, diplomová práce, Brno, FIT VUT v Brně, 2015

# Mobilní aplikace pro podporu řízení dopravních staveb

## Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně pod vedením pana Doc. Ing. Adama Herouta a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Radek Obořil
26. května 2015

## Poděkování

Tímto bych chtěl poděkovat panu Doc. Ing. Adamu Heroutovi, vedoucímu této práce, za odbornou pomoc, ochotu a čas, kterou mi při její tvorbě věnoval. Dále bych rád poděkoval Ing. Petru Křenkovi a společnosti Q2 Interactive za poskytnutí zdrojů, potřebných při tvorbě této práce a Ing. Romanu Kopřivovi za konzultace specifikací.

# Obsah

# Kapitola 1

# Úvod

I přes velký rozmach webových technologií, mobilních technologií a ERP systémů v posledních letech, mnoho firem stálé používá pro správu zdrojů, zakázek a firemních procesů zastaralé systémy, které často spoléhají výhradně na tištěné a ručně psané podklady. To je dáno rozmanitým zaměřením firem, specifiky a rozdíly ve firemních procesech, pro jejichž modelování je často nutné vyvinout software na míru. Na trhu je tedy stále prostor pro produkty tohoto zaměření, jejichž nedílnou součástí se v poslední době stávají také aplikace pro mobilní zařízení.

Dle výzkumné a poradenské agentury Gartner překonal v prvním kvartále roku 2013[1] prodej tzv. chytrých mobilních telefonů prodej telefonů klasických[2]. Dá se tedy očekávat, že v budoucnosti bude většina uživatelů mobilních telefonů vlastnit právě chytré telefony. Často se jedná o přistroje s výkonem, kterého bylo ještě před několika lety možné dosáhnout pouze na osobních počítačích při řádově vyšší spotřebě elektrické energie. Tato zařízení také zpravidla podporují datové přenosy přes síť GSM, určování pozice pomocí družicových navigačních systémů a jsou vybaveny integrovanými kamerami. Dalším jejich společným znakem je možnost vývoje nativních aplikací pro jejich mobilní operační systémy, které mohou těchto vlastností využívat.

Jedním z oborů, ve kterých je možné využít moderní technologie chytrých mobilních telefonů jsou dopravní stavby. Dopravními stavbami se v kontextu této práce rozumí výstavba, údržba nebo rekonstrukce dopravních komunikací. Primárně jsou uvažovány komunikace pro osobní dopravu, tedy silnice a dálnice. Silniční síť v České Republice dosahovala v roce 2013 celkové délky 55 716 kilometrů[3]. Náklady na silniční infrastrukturu se v ČR pohybují ročně okolo 60 mld. Kč [4]. Stabilně se tedy jedná o aktivní odvětví průmyslu.

Úkolem této práce je vytvořit mobilní aplikaci pro podporu řízení dopravních staveb. Aplikace umožňuje řízení prací a jejich precizní dokumentaci přímo v terénu pomocí integrovaného fotoaparátu a technologií pro určování polohy. Pořízená dokumentace je odesílána na server pro další zpracování.

Kapitola 2 analyzuje zadání a potřebné vlastnosti cílových zařízení. Dále identifikuje technologická úskalí, která byla při vývoji aplikace nutné překonat. Kapitola 3 popisuje současný stav trhu s chytrými mobilními telefony. Zabývá se možnostmi multiplatformního

[1] Zdroj: [8]

[2] Rozumí se mobilní telefony běžící na a proprietární platformě (tj. bez standardního mobilního systému), jejichž sadu funkcí není možné uživatelsky rozšířit. Někdy jsou označovány jako tzv. *hloupé telefony*. V angličtině se používá výraz feature phone.

[3] Jedná se o součet délek dálnic a silnic první, druhé a třetí třídy.

[4] Zdroje: [6, 4]

vývoje a blíže popisuje technologie, které byly vybrány pro implementaci aplikace, především nástroj Xamarin a framework MvvmCross. Kapitoly 2 a 3 vznikly z větší části v rámci semestrálního projektu. Kapitola 4 prezentuje návrh jednotlivých částí výsledné aplikace. Následující kapitola 5 je věnována její implementaci. Testování aplikace je popsáno v kapitole 6.

# Kapitola 2

# Specifikace a analýza požadavků

Tato kapitola analyzuje požadavky na aplikaci. Zabývá se jejím zadáním a popisuje vlastnosti a technologie cílových zařízení, které jsou pro funkci výsledné aplikace zásadní.

## 2.1 Zadání

Společnost Q2 Interactive, sídlící v Brně, se rozhodla vytvořit produkt pro podporu dopravních staveb. Jedná se o řešení umožňující správu prováděných prací a jejich dokumentaci v terénu pomocí mobilního zařízení, které bude možné integrovat jako součást ERP systému. Tento produkt není zadán konkrétním klientem. Měl by mít hlavně demonstrační úlohu pro získávání nových klientů z oboru dopravních staveb a sloužit jako základ pro vývoj produktů, nasaditelných do reálného provozu. Návrh aplikace je proto v některých ohledech zjednodušen.

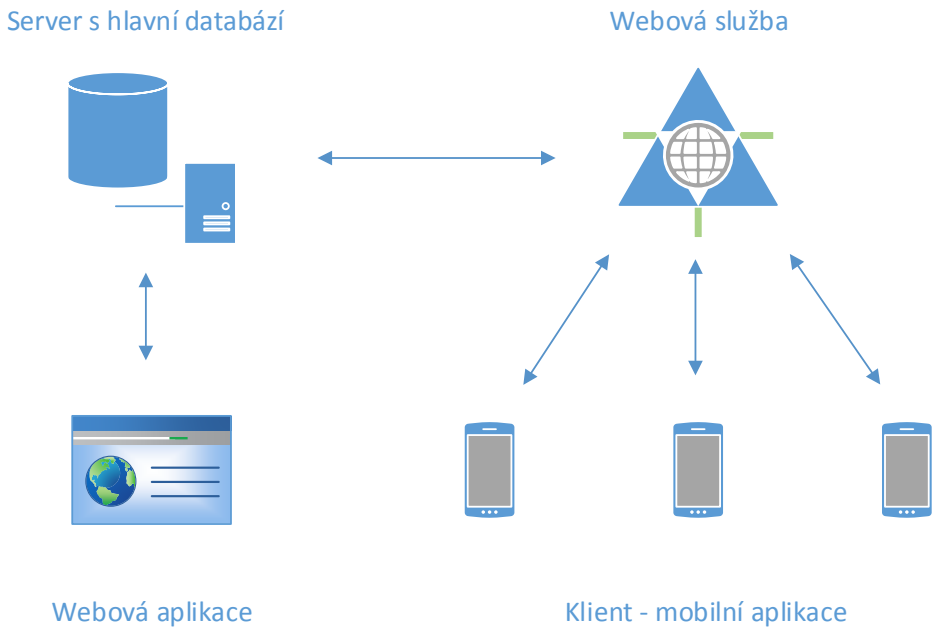
Přestože aplikace nebyla vyvíjena pro konkrétního klienta, impulz pro její vytvoření vznikl na základě zájmu manažerů firem z oboru dopravních staveb, se kterými společnost Q2 Interactive spolupracovala v oblasti internetového marketingu. Požadavky, zachycené v následujících sekcích byly získány převážně na základě neformálních rozhovorů s Ing. Petrem Křenkem, jednatelem společnosti Q2 Interactive a Ing. Romanem Kopřivou, obchodním manažerem společnosti Značky Morava, a.s.

## 2.2 Architektura řešení

Požadované řešení je aplikace typu klient – server. Jeho schéma je znázorněno na obrázku 2.1. Jednotlivé části jsou popsány v následujícím textu.

### 2.2.1 Klient

Roli klienta představuje mobilní aplikace pro použití v terénu. Jejím hlavním účelem je pořizování přesné obrazové dokumentace pomocí integrované kamery mobilního zařízení. Kromě obrazové dokumentace obsahují záznamy také informace o místě pořízení, především přesnou polohu, získanou pomocí lokalizační technologie mobilního zařízení. Podle polohy umožňuje aplikace určit další informace o místě, např. číslo silnice, okres nebo katastr. Záznamy dokumentují hlavně vykonané práce na zakázkách, ale aplikace také umožňuje pořizování záznamů, které se vykonaných prací přímo netykají - např. dokumentování škod, způsobených přírodními živly, apod.




Obrázek 2.1: Schéma aplikace.

Aplikace také podporuje řízení prací – umožňuje vytváření a změny jednotlivých úkolů, které spadají do zakázky, přímo v terénu. Při vytváření úkolů je aplikace schopna určovat délky úseků komunikace.

Pro spojení se serverem při práci v terénu využívá aplikace mobilního internetového připojení. Musí proto počítat s jeho omezenou dostupností a umožňovat odloženou synchronizaci. Aplikace by ideálně měla být multiplatformní, minimálně musí fungovat na zařízeních běžících na platformě Android.

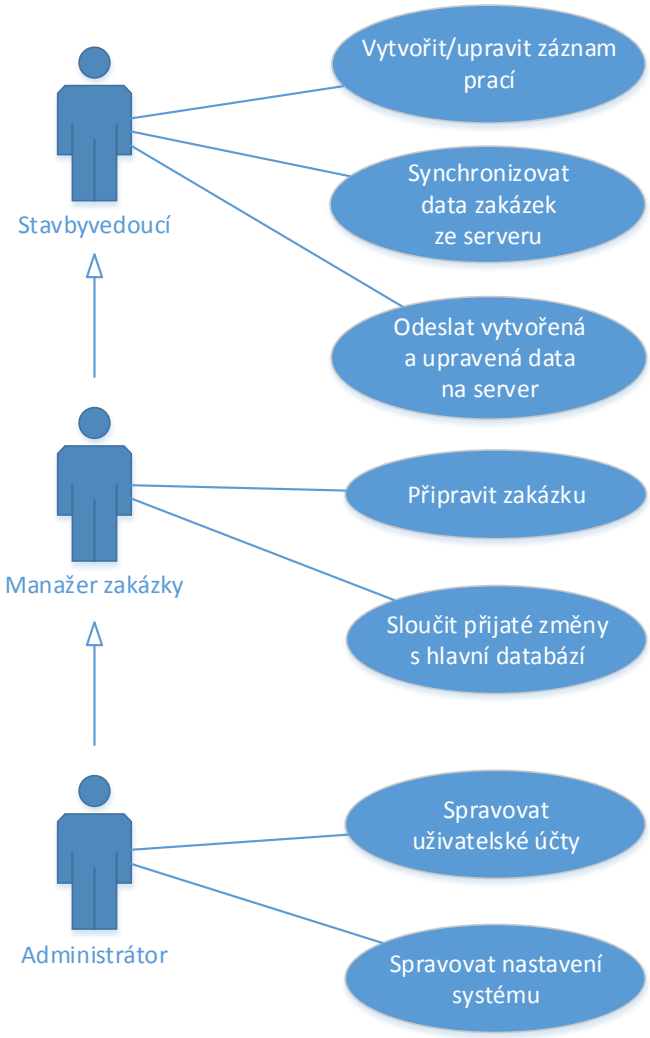
### 2.2.2 Server

Serverovou aplikaci je možné rozdělit na dvě následující části:

**Webová služba** – zajišťuje komunikaci s klienty. Umožňuje především přenos dat nasbíraných v terénu na server, a to včetně obrazové dokumentace. Dalším úkolem je synchronizace aktuálních uložených dat k zakázkám a nastavení na klientská zařízení. Pro tyto účely poskytuje vhodné aplikační rozhraní (API).

**Webová aplikace** – jejím hlavním účelem je správa zakázek a zpracování dat, která byla nasbírána v terénu. Umožňuje také provádění operací, ke kterým nejsou mobilní zařízení vhodná. Obzvlášť takové, které zahrnují zadávání delších částí textu, nebo jiné operace, které se na mobilních zařízeních provádějí obtížně kvůli malým rozměrům obrazovky. Zejména se jedná o přípravu nových zakázek, která zahrnuje přesnou definici jednotlivých úkolů.

Protože server obsahuje hlavní databázi celého systému, musí webová aplikace umožňovat také správu dat přijatých z terénu a jejich sloučení se stávajícími daty zakázek.




Obrázek 2.2: Diagram případu užití specifikované aplikace.

Z existence více klientů také vyplývá možnost vzniku konfliktů v přijatých datech, k jejichž řešení musí webová aplikace poskytovat vhodné rozhraní.

## 2.3 Aktéři

Ze specifikovaných požadavků vyplývá existence tří následujících aktérů (rolí):

**Stavbyvedoucí** – reprezentuje vedoucího pracovníka v terénu, který má na starosti dokumentaci provedených prací. Jeho interakce se systémem se omezuje na použití mobilní aplikace. Nemá tedy přístup do webové aplikace.

**Manažer zakázky** – má za úkol správu dat v systému. Jeho hlavním úkolem je vytváření nových zakázek a úkolů, ze kterých se jednotlivé zakázky skládají a které je třeba

| Technologie | Rychlost - downlink[1] | Rychlost - uplink[1] | Pokrytí [2] |
|---|---|---|---|
| EDGE | 236 Kbit/s | 59,2 Kbit/s | 96,4% |
| HSPA+ | 42 Mbit/s | 5,76 Mbit/s | 35,4 % |
| LTE | 100 Mbit/s | 57,6 Mbit/s | |

Tabulka 2.1: Přehled rychlostí a pokrytí mobilního datového připojení na území ČR. Zdroje: [24, 28, 26, 31, 30, 7]

v terénu vykonat. Jeho úlohou je dále správa změn, které jsou na server odesílány z terénu, a jejich sloučení s daty v hlavní databázi. Primárně pracuje s webovou aplikací. Sdílí základní práva s aktérem Stavbyvedoucí, tedy může využívat i mobilní aplikaci.

**Administrátor** – jedná se o rozšíření aktéra Manažer zakázky o správu uživatelských účtů, mobilních zařízení, nastavení systému, apod.

Hierarchii uživatelských rolí a jejich hlavní interakce se systémem znázorňuje diagram případů užití na obrázku 2.2. Uživatelé všech rolí (tj. ve webové i mobilní aplikaci) se autentizují pomocí uživatelského jména a hesla. Vytváření uživatelských účtů mají na starosti uživatelé s rolí Administrátor.

## 2.4 Využité technologie a vlastnosti mobilních zařízení

Pro dosažení požadovaných vlastností musí cílové mobilní zařízení podporovat několik technologií. Ty jsou v této části textu podrobněji popsány. Dále jsou diskutovány možnosti jejich použitelnosti v aplikaci.

### 2.4.1 Internetové připojení

Navrhovaná mobilní aplikace má v celém řešení úlohu klienta. Pro správnou funkci tedy vyžaduje datové připojení k serveru. Součástí mobilních sítí ve všech rozvinutých státech světa jsou služby pro přenos datových paketů. Primárně se počítá s fungováním aplikace v podmínkách na území České Republiky. Z tabulky 2.1 vyplývá, že rychlosti různých technologií pro datové přenosy se velmi významně liší. Pokrytí technologií EDGE na území ČR je téměř kompletní. Rychlosti této technologie ale nejsou dostatečné pro synchronizaci obrazové dokumentace úkonů, obzvláště jejich odesílání na server. Úroveň pokrytí technologiemi s vyššími přenosovými rychlostmi není dostatečná k tomu, aby s dostupností rychlejších datových přenosů bylo při návrhu aplikace možné obecně počítat.

Z výše uvedeného vyplývá, že navrhovaná aplikace musí být schopna pracovat v režimu bez připojení a umožňovat odloženou synchronizaci se serverem. Jako další nezbytná funkce se jeví správa stažených podkladů k zakázkám a možnost jejich stahovaní na vyžádání.

[1] Maximální rychlosti udávané třemi největšími operátory v ČR (Telefónica, T-Mobile, Vodafone). Zpravidla jsou nižší než teoretická max. rychlost samotné technologie, což je závislé na použité revizi a režimu provozu. Rychlosti uplinku (odesílání dat) nejsou často operátory udávány a jsou proto získány z jiných zdrojů pro odpovídající typy a režimy provozu dané technologie.

[2] Průměrné pokrytí území třemi největšími operátory v ČR podle Českého telekomunikačního úřadu v době vypracování semestrálního projektu (leden 2014).

### 2.4.2 Integrovaný fotoaparát

První telefon s integrovaným fotoaparátem se v České Republice začal prodávat již v roce 2002[3]. V dalších letech se integrované kamery v mobilních telefonech staly standardem. V dnešní době je jejich kvalita na dostatečné úrovni, aby v některých případech nahradili dedikované digitální fotoaparáty. Z pohledu mobilních vývojářů je podstatné, že využití integrované kamery je podporováno nástroji pro tvorbu nativních aplikací na všech dnes rozšířených platformách.

Pro účely obrazové dokumentace stavebních prací je kvalita kamer v dnešních chytrých telefonech dostačující. Díky jejich standardizaci nemá požadavek na existenci integrované kamery téměř žádný vliv na množství podporovaných cílových zařízení.

### 2.4.3 Systém pro určení polohy

Systémy pro určení polohy, založené na různých technologiích, mají v mobilním odvětví významnou roli. Nejpřesnější a nejpoužívanější jsou globální družicové systémy, tedy takové, jejichž funkce je závislá na družicích na oběžné dráze Země. Nejznámějším takovým systémem je GPS[4] (Global Positioning System).

#### GPS

Obsah této kapitoly vychází z [20] a [27].

Pomocí GPS přijímače lze určit kdekoliv na planetě čas, rychlost a aktuální geografickou pozici včetně nadmořské výšky. Původně byl systém GPS vyvíjen americkým ministerstvem obrany pro vojenské použití. V průběhu vývoje ale bylo rozhodnuto, že bude k dispozici i pro civilní účely. Získal tak mnoho dalších využití a zaznamenal rapidní nárůst uživatelů, kteří se dnes počítají na desítky miliónů. To má několik důvodů, zejména:

- Relativně vysoká polohová přesnost
- Schopnost určovat i rychlost a čas
- Dostupnost signálů kdekoliv na planetě
- Použití služby je standardně bezplatné
- Systém pracuje za každého počasí a je dostupný 24 hodin denně
- Přijímače prošly v posledních letech rapidním vývojem, a to hlavně v oblasti minimalizace rozměrů a nároků na spotřebu elektrické energie

Systém spoléhá na 32 družic na oběžné dráze Země. Komunikace mezi družicemi a přijímačem je dosaženo pomocí rádiových vln. Jejich frekvenční pásma jsou zvolena tak, aby docházelo k nejmenšímu možnému rušení při jejich průchodu atmosférou.

Nejdůležitější funkce systému, tedy určování polohy, je docíleno pomocí výpočtu tzv. zdánlivých vzdáleností od jednotlivých družic. Přijímač synchronizuje své hodiny s hodinami družic a pomocí dalších údajů, které družice vysílají (hlavně parametry jejich dráhy),

[3] Podle [17] se jednalo telefon Nokia 7650 s operačním systémem Symbian S60.

[4] Další významné globální systémy jsou GLONASS, provozovaný ruskou armádou a Galieo realizovaný ESA (Evropská kosmická agentura).

může tyto zdánlivé vzdálenosti určit. Z nich je pak možné po korekci chyb za pomocí trilaterace[5] určit aktuální polohu. Pro určení polohy ve dvojrozměrném prostoru (zeměpisná délka a šířka) je nutné přijímat signál z nejméně tří družic. Pro určení trojrozměrné polohy (tj. včetně nadmořské výšky) je pak nutný současný příjem signálu ze čtyř družic.

Důležitým parametrem GPS přijímačů je čas nutný pro získání polohy. Označuje se zkratkou TTFF (time to first fix). Rozlišuje se několik případů TTFF.

- Studený start (cold start)
- Teplý start (warm start)
- Horký start (hot start)

Jednotlivé případy se liší informacemi, které má přijímač k dispozici z předchozího použití. Konkrétní časy závisí na aktuálních podmínkách. Studený start trvá řádově minuty, zbylé dva případy zpravidla několik jednotek až desítek sekund. Pro zrychlení těchto časů se používá technologie A-GPS (Assisted GPS, také aGPS). Funguje pomocí získávání užitečných informací, v prvé řadě o poloze a dráze družic, z internetových serverů. Zvláště v případě studeného startu může použití A-GPS přinést znatelné zrychlení.

Podpora systému GPS je v dnešních chytrých telefonech standardem. Některé modely podporují v kombinaci s GPS také ruský globální družicový systém GLONASS, což dále zvyšuje přesnost přijímače a snižuje časy TTFF. S využitím technologií pro přesné určení polohy je tedy při návrhu aplikace možné počítat.

### 2.4.4 Mapové podklady

S určováním polohy úzce souvisí dostupnost mapových podkladů. Mapy mají v mobilních zařízeních již poměrně dlouhou historii – sahají až do dob monochromatických displejů. V dnešní době jsou v mobilní zařízeních nejčastěji využívány jako součást navigačních aplikací.

Všechny společnosti, stojící za významnými mobilními operačními systémy dnešní doby, nabízejí vlastní mapové podklady[6] a aplikace pro jejich prohlížení. Z pohledu této práce je ale důležitější, že nabízejí také vývojové nástroje, které programátorům umožňují mapy integrovat do mobilních aplikací.

[5] Jedná se o proces výpočtu pozice ze vzdáleností od známých bodů. V případě systému GPS se počítá jako průnik povrchů koulí.

[6] Google Maps, Apple Maps, Bing Maps.

# Kapitola 3

# Mobilní platformy a multiplatformní vývoj

I přesto, že mají nativní aplikace pro různé mobilní operační systémy v zásadě stejné možnosti, ekosystémy pro distribuci a vývojové nastroje jsou často velmi odlišné. Zpravidla se liší i v použitém programovacím jazyce. V posledních letech proto vznikají nástroje pro multiplatformní vývoj mobilních aplikací. Ty vývojářům umožňují přenos kódu mezi platformami a tedy možnost současného vývoje aplikací pro co největší část trhu.

Tato kapitola se zabývá nejrozšířenějšími mobilními platformami v době vzniku práce[1], shrnuje možnosti multiplatformního vývoje a blíže popisuje nástroje, které byly použity pro vývoj zadané mobilní aplikace.

## 3.1 Nejrozšířenější mobilní platformy

Jedním z nejdůležitějších faktorů při výběru cílové platformy pro mobilní aplikaci je tržní podíl. Z tabulky 3.1 vyplývá, že světově nejprodávanější mobilní platformy jsou:

- Google Android
- Apple iOS
- Microsoft Windows Phone

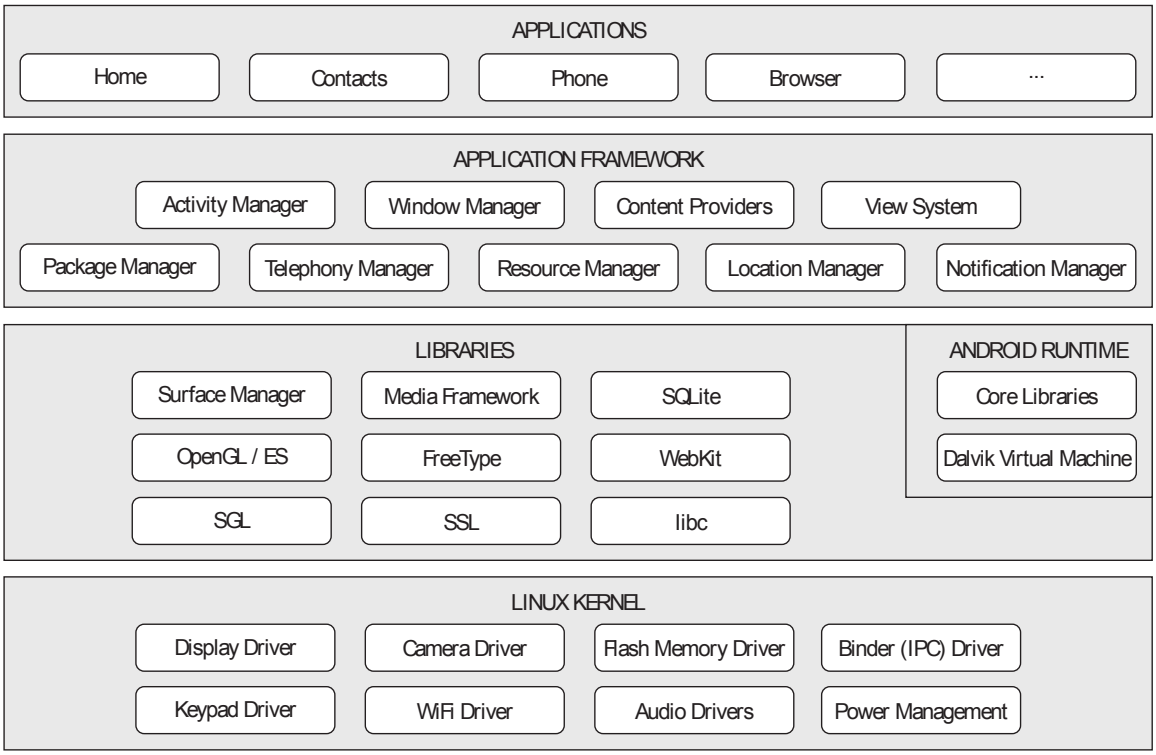
### 3.1.1 Android

Tato část převážně vychází z [11].

[1] Většina textu této kapitoly vnikla v rámci semestrálního projektu v lednu roku 2014.

| **OS** | **Podíl na trhu** [%] |
|---|---:|
| Android | 80,7 |
| iOS | 15,4 |
| Windows Phone | 2,8 |
| Ostatní | 1,1 |

Tabulka 3.1: Rozložení globálního trhu mobilních zařízení v roce 2014. Zdroj: [9]




Obrázek 3.1: Architektura operačního systému Android. Zdroj: [11]

Android je softwarová platforma pro mobilní zařízení, postavená na modifikované verzi operačního systému linux. Je primárně určena pro chytré mobilní telefony a tablety, ale může se uplatnit v mnohem širší škále zařízení. Platformu tvoří operační systém, middleware[2] a základní aplikace. Android byl původně vyvíjen americkou společností stejného jména, kterou v roce 2005 odkoupila společnost Google. Většina zdrojových kódu Androidu pak byla vypuštěna pod open-source licencí Apache[3]. Android se tak stal pro výrobce hardwaru, zvláště mobilních telefonů a tabletů, velmi atraktivním. Chce-li výrobce do svého zařízení nainstalovat Android, stačí mu stáhnout plný zdrojový kód a provést žádané úpravy. Zpravidla výrobci přidávají vlastní grafickou nástavbu.

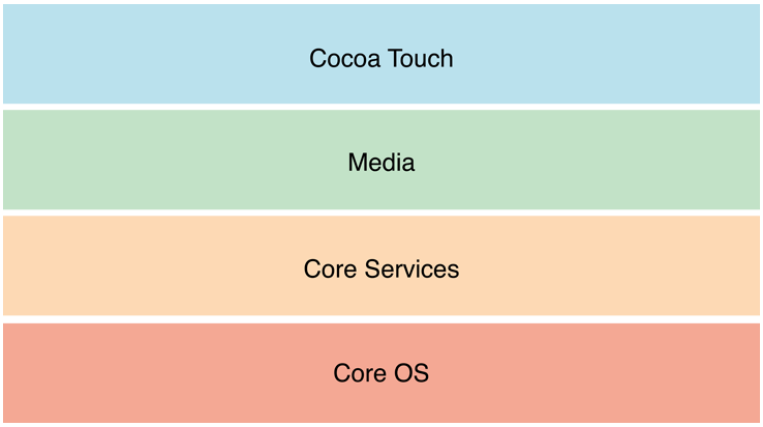
První zařízení, běžící na operačním systému Android, se začalo prodávat v říjnu roku 2008[4]. Android pak zaznamenal velmi rychlý vzestup. Z celosvětového podílu na trhu mobilních operačních systémů 2,8% ve 2. čtvrtletí roku 2009 se do 4. čtvrtletí roku 2010 stal světově nejprodávanější mobilní platformou. Z tabulku 3.1 vyplývá, že jeho celosvětový podíl v roce 2014 je 80,7%.

Obrázek 3.1 zachycuje architekturu operačního systému Android. Z pohledu vývojáře je nejdůležitější část Android Runtime. Jedná se o knihovny umožňující běh aplikací. Součástí vrstvy je také Dalvik VM, speciální virtuální stroj pro exekuci bytového kódu programovacího jazyka Java, navržený pro běh na přenosných zařízeních s omezenými zdroji. Pro každou spuštěnou aplikaci je spuštěna instance VM Dalvik. Vykonávání kódu aplikace je tak izolováno od ostatních běžících aplikací.

[2] Software, spojující operační systém a aplikace.

[3] Licence pro open-source software, která umožňuje libovolné úpravy a distribuci softwaru, který pod ní spadá.

[4] Jednalo se o telefon HTC Dream. Do prodeje byl vypuštěn 22.10.2008. [17]




Obrázek 3.2: Základní vrstvy systému iOS. Zdroj: [2]

#### Vývoj nativních aplikací pro Android

Nativní aplikace pro Android jsou programovány v jazyce Java[5]. Primárním nástrojem je Android SDK, pomocí kterého je možné vyvíjet na všech hlavních počítačových operačních systémech, tedy Windows, Mac OS X a Unixových systémech. To je ve kontextu ostatních mobilních platforem spíše výjimkou. Součástí SDK je upravená verze programovacího prostředí Eclipse. K vývoji lze ale použít velké množství neoficiálních nástrojů. Další důležitou součástí SDK je emulátor. I ten je možné nahradit neoficiálními alternativami[6], jejichž výhodou je často rychlejší běh. Samotný vývoj aplikací pro Android je bezplatný. Licence, umožňující zveřejňování aplikací v obchodě Google Play, je ale zpoplatněna jednorázovým poplatkem ve výší 25 USD.

Při programování aplikací pro Android si programátor musí být vědom existence velkého množství verzí platformy a faktu, že většina z nich má stále své uživatele. Při zacílení co největšího počtu uživatelů tak někdy není možné využívat nejnovější prvky API.

### 3.1.2 iOS

Informace v této části textu byly čerpány z [2, 23, 29].

Operační systém iOS byl vyvinut americkou společností Apple. Je postaven na počítačovém operačním systému Mac OS X, se kterým sdílí některé vlastnosti. Apple umožňuje využívání iOS výhradně na svých vlastních zařízeních. Konkrétně se jedna o mobilní telefony, tablety a multimediální přehrávače z modelových řad iPhone, iPad a iPod. Z toho, zvláště oproti platformě Android, plynou výhody, jako například vysoká optimalizace a relativně malá roztříštěnost platformy.

První zařízení, běžící na operačním systému iOS (původně pojmenovaném iPhone OS), bylo uvedeno na trh v létě roku 2007[7]. Jeho uvedení zásadně změnilo mobilní trh a nastavilo jeho nový směr. Jednalo se o první chytrý telefon s dotykovou obrazovkou, který byl určen pro širokou veřejnost. Zásadními novinkami bylo použití dotykového displeje s kapacitní technologií, ovládání pomocí vícedotykových gest a důraz na kvalitu grafického rozhraní a uživatelskou zkušenost. I přes omezenou funkční výbavu ve srovnání s tehdejšími chytrými

[5] Android také nabízí nástroj NDK (Native Development Kit), který umožňuje pomocí rozhraní JNI (java native interface) vkládat do aplikací komponenty programované v nativním jazyce OS (C/C++).

[6] Příkladem je emulátor Genymotion.

[7] Jednalo se o mobilní telefon iPhone první generace, uvolněný 29.6. 2007.

telefony a z počátku i absencí vývojových nástrojů pro tvorbu nativních aplikací se první generace telefonu iPhone dočkala okamžitého úspěchu. Uvolnění vývojářských nástrojů pro iOS na jaře roku 2008 pak umožnilo vznik dnes standardního ekosystému mobilních platforem, založeném na vývoji a centralizované distribuci aplikací.

Obrázek 3.2 zachycuje základní vrstvy operačního systému iOS. Nejvyšší vrstva, Cocoa Touch, se stará o běh aplikací a multitasking. Její součástí je framework, který vývojářům umožňuje přístup k funkcím operačního systému a samotnému zařízení. Stejně jako nativní aplikace je vrstva Cocoa Touch naprogramována v jazyce Objective-C a využívá API Cocoa, použité v operačním systému pro stolní počítače, Apple Mac OS X.

#### Vývoj nativních aplikací pro iOS

Jak již bylo naznačeno výše, aplikace pro iOS jsou programovány v jazyce Objective-C. Ten vznikl v 80. letech minulého století jako nástavba nad jazykem C. Části syntaxe tohoto jazyka jsou převzaty z objektového jazyka Smalltalk. Jediným oficiálním nástrojem pro tvorbu aplikací, běžících na iOS, je XCode. Jedná se o sadu vývojových nástrojů, jejíž nejdůležitějšími prvky jsou stejnojmenné vývojové prostředí a emulátor mobilních zařízení. Na rozdíl od vývojových nástrojů pro platformu Android je XCode možné používat exkluzivně na operačním systému Mac OS X.

Oficiální distribuce aplikací probíhá přes portál iTunes, který stejně jako portál Google Play, sdružuje distribuci aplikací, hudby, elektronických knih, filmů a dalšího obsahu. Již pro instalaci aplikace do cílového zařízení je nutné zakoupit vývojářskou licenci. Existují dvě základní verze licence:

**iOS Developer Program** – pro vývoj aplikací distribuovaných výhradně pomocí portálu iTunes.

**iOS Developer Enterprise Program** – umožňuje vývoj proprietárních aplikací, distribuovaných mimo iTunes.

Tyto licence je možné pořídit za 99 USD, respektive 299 USD[8] na rok. Aplikace distribuované přes portál iTunes procházejí schvalovacím procesem. Tento fakt, zejména v kontrastu s Google Play, přináší výhody v podobě ochrany uživatelů před nebezpečnými aplikacemi a útoky.

### 3.1.3 Windows Phone

Tato část vychází převážně z [10, 25].

Mobilní operační systém Windows Phone (také WP), za kterým stojí americká firma Microsoft, je jeden z nejmladších na trhu. Vznikl, aby nahradil předchozí operační systém stejné společnosti, tedy Microsoft Windows Mobile[9]. K uvolnění první verze systému došlo ve 3. čtvrtletí roku 2010. Ve stejné době se začaly prodávat i první mobilní telefony s tímto systémem[10]. První verze systému nazvaná Windows Phone 7 se ve srovnání s konkurencí vyznačovala menšími nároky na HW zařízení, ale také menší funkční výbavou. Dalším nedostatkem WP, způsobeným mimo jiné nekompatibilitou aplikací pro Windows Mobile, byl

[8] Zdroj: [3]

[9] Systém Windows Mobile byl v době jeho nahrazení ve verzi 6.5. WP (Windows Phone) není nová verze systému Windows Mobile. Jedná se o zcela nový produkt. Nativní aplikace, vytvořené pro Windows Mobile, nejsou kompatibilní s platformou WP.

[10] Jednalo se o několik modelů značek HTC, LG a Samsung.

také daleko menší počet dostupných aplikací. Tyto nedostatky OS byly postupně odstraňovány v dalších verzích systému. V únoru roku 2011 Microsoft navázal strategické partnerství s výrobcem mobilních telefonů Nokia, které v září roku 2013 vyústilo v akvizici společnosti. Verze systému Windows Phone 8 (WP8) přinesla poměrně radikální změny v architektuře a vývoji pro platformu. Je postavena na Windows NT[11]. WP8 tak sdílí některé prvky s OS Microsoft Windows 8, určeným pro počítače. To usnadňuje přenos aplikací mezi těmito systémy. Nejnovější verze systému v době vzniku této práce je Windows Phone 8.1.

#### Vývoj nativních aplikací pro Windows Phone

Primárním podporovaným nástrojem pro vývoj nativních aplikací pro WP je Windows Phone SDK. Ten je možné používat výhradně na počítači s operačním systémem Windows verze 7 a vyšší. Hlavní součástí Windows Phone SDK je vývojové prostředí Visual Studio Express. Jedná se o zjednodušenou verzi rozšířeného komerčního vývojového prostředí stejného jména. SDK také obsahuje emulátor mobilních zařízení. Hlavním programovacím jazykem pro vývoj pomocí W8 SDK je C#. Je ale možné programovat i v jiných jazycích, jako je Visual Basic a částečně i C/C++[12]. I přes rozdílnou architekturu WP8.1 oproti starším verzím, je s určitými omezeními možné pomocí Windows Phone SDK současně vyvíjet i pro starší verze WP.

Distribuci mobilních aplikací pro WP oficiálně zajišťuje portál Windows Phone Store. Základní vývojářský účet, umožňující zveřejňování aplikací do Windows Phone Store a Windows Store[13], je možné pořídit za 19 USD na rok. Firemní účet, umožňující distribuci aplikací mimo dva zmíněné portály, je možné získat za roční poplatek 99 USD. Podobně, jako v případě aplikací pro iOS, prochází aplikace pro WP před publikací na portálu WP Store certifikačním procesem, jehož součástí je i manuální testování aplikace. Tento fakt může mít kromě výhod pro uživatele také nevýhody z pohledu vývojářů, a to zejména v podobě prodlev při publikaci aplikace. [14]

## 3.2 Multiplatformní vývoj

Tato část práce čerpá zejména z [21] a [22].

Se zvyšováním výkonu mobilních zařízení v posledních letech nabývaly mobilní aplikace na významu. Z informací uvedených v předchozích částech tohoto textu vyplývá, že je trh s mobilními aplikacemi nehomogenní a dynamický. Z toho plynou poměrně omezené možnosti předvídání jeho vývoje. Z roztříštěnosti trhu vzniká potřeba vývoje aplikací pro více platforem.

Oddělený vývoj pro jednotlivé cílové platformy může tvorbu aplikace značně prodloužit a tak i zdražit, a to až tolikanásobně, kolik platforem je zacíleno. Další nevýhodou tohoto přístupu je fakt, že vyžaduje znalosti potřebné pro vývoj na všech cílových platformách nebo oddělené vývojové týmy v případě firemního vývoje.

Typický životní cyklus mobilní aplikace je podobný životnímu cyklu jiných softwarových produktů. Můžeme jej rozdělit do následujících fází:

- Návrh

[11] Starší verze systému měly základ ve Windows CE.

[12] Je např. možné využít existujících knihoven v C/C++. Další využití jsou programování aplikací, které využívají Direct3D nebo zrychlení běhu výpočetně náročných částí aplikací.

[13] Portál pro prodej HW a aplikací pro operační systémy of firmy Microsoft, určené pro počítače.

- Vývoj
- Publikování
- Údržba

Jedinou fází, kterou lze při odděleném vývoji aplikací provádět současně, je návrh. A to navíc jen částečně[14]. Je zřejmé, že zvýšení nákladů a časových nároků, spojené s cílením více platforem, se neomezuje pouze na fázi vývoje. Jako východisko se nabízí multiplatformní vývoj aplikací.

Na tomto místě je vhodné poznamenat, že samostatný vývoj pro více platforem má také své výhody. A to především v rozsáhlejších možnostech optimalizace aplikace, které plynou z přímého přístupu k jednotlivým API v možnosti použití specializovaných vývojových nástrojů.

Z důvodů, uvedených v předchozích odstavcích, začaly v posledních letech vznikat nástroje pro multiplatformní vývoj. Většina existujících multiplatformních řešení produkuje aplikace založené na webových technologiích. Využívají společného rysu mobilních operačních systémů, integrovaného webového prohlížeče. Uživatelské rozhraní takto vzniklých aplikací je zpravidla postaveno na technologiích HTML5, Javascript a CSS3. V některých případech tyto aplikace nejsou z pohledu uživatele rozeznatelné od aplikací, vzniklých nativním vývojem. Vývojové nástroje z této kategorie také často umožňují přístup k omezené množině funkcí nativních API jednotlivých platforem. Právě omezená funkcionalita a často nižší výkon aplikací jsou nejvýraznějšími nevýhodami použití přístupů, založených na webových technologiích. V tomto kontextu stojí za zmínku také tzv. aplikační továrny (Application factory). Ty umožňují vývoj jednoduchých aplikací bez znalostí programovacích jazyků.

Další skupinou multiplatformních řešení jsou nástroje, které umožňují vývoj plnohodnotných nativních aplikací. Zpravidla nabízejí vlastní SDK. Mohou obsahovat vlastní runtime, ve kterém je běh aplikace zajištěn pomocí virtualizace nebo interpretu. Další nastroje přímo generují strojový kód (nebo bytový kód) pro cílovou platformu. Představitelem kombinace těchto přístupů je nástroj Xamarin, popsaný v následující části textu.

## 3.3 Xamarin

Xamarin je sada nástrojů pro multiplatformní vývoj, kterou vyvíjí stejnojmenná americká společnost. Je založená na open-source projektu Mono, který byl původně vyvíjen společností Novell. Projekt Mono je implementací specifikace Common Language Infrastructure (CLI)[15]. Jeho hlavním cílem je umožnit běh aplikací vytvořených v prostředí Microsoft .NET Framework na dalších platformách.

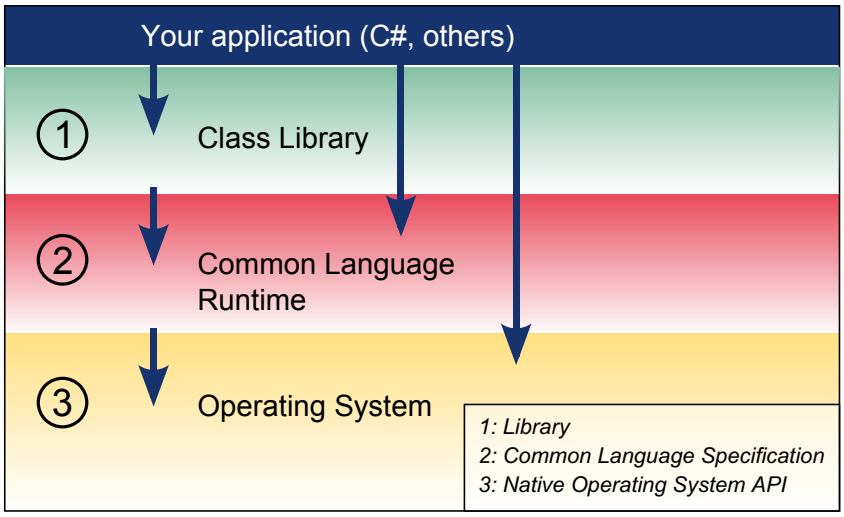
Základními stavebními prvky nástroje Xamarin jsou Xamarin.iOS[16], a Xamarin.Android[17]. Jedná se o implementace projektu Mono pro dva nejvýznamnější mobilní operační systémy, Android a iOS. Při programování pomocí Xamarin.iOS a Xamarin.Android má programátor k dispozici plnohodnotná API odpovídajících platforem v jazyce C# a navíc knihovny

[14] Návrh UI je nutné pro jednotlivé platformy odlišovat v závislosti na jejich rozdílných konceptech.

[15] Jedná se o otevřenou specifikaci vyvinutou firmou Microsoft, která definuje vlastnosti spustitelného kódu a prostředí pro jejich běh. Implementace CLI je základním stavebním prvkem architektury frameworku Microsoft .NET.

[16] Původně pojmenovaný MonoTouch. První verze byla vydána v roce 2009.

[17] Původně pojmenovaný Mono for Android. První verze byla vydána v roce 2011.




Obrázek 3.3: Zjednodušené schéma architektury frameworku .NET. Zdroj: [18]

z Base Class Library (BCL), tedy základní knihovny specifikované v CLI. Roli BCL a CLI v rámci architektury frameworku .NET znázorňuje obrázek 3.3.

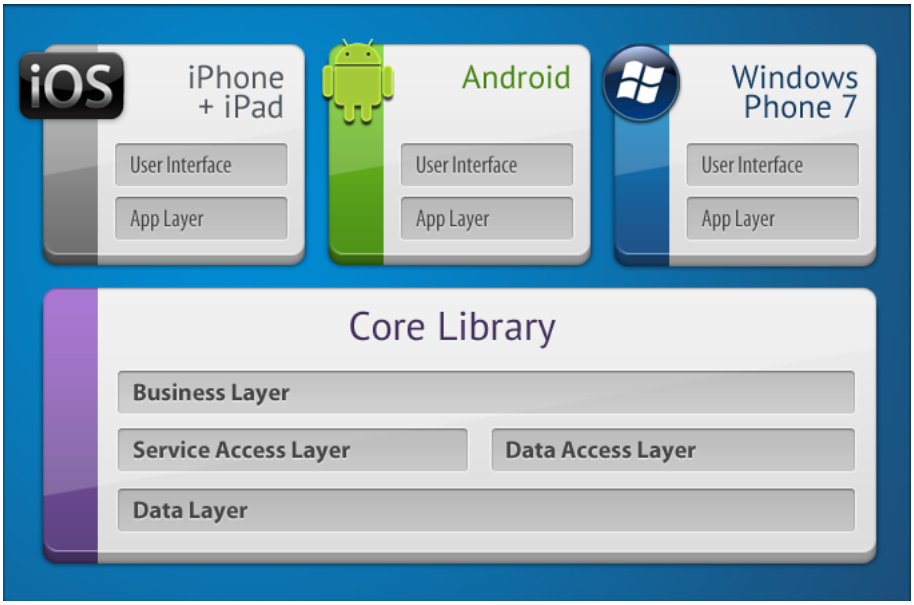
Standardním přístupem v .NET frameworku je kompilace do Common Intermediate Language (CIL)[18] a jeho následná just-in-time (JIT) kompilace do strojového jazyka cílového zařízení za běhu. Tento přístup je využíván u Xamarin.Android. Z důvodu omezení systému iOS není použití tohoto přístupu v případě Xamarin.iOS možné. Aplikace vytvořené v Xamarin.iOS jsou tak předkompilované do nativního spustitelného souboru operačního systému. Pro co možná nejefektivnější využití zdrojů obsahuje Xamarin.iOS také vlastní linker.

Xamarin sice explicitně nepodporuje operační systém Windows Phone, ale jak již bylo uvedeno výše, kód vytvořený v C# a .NET je z touto platformou kompatibilní. Xamarin tak umožňuje tvorbu nativních aplikací pro všechny tři nejrozšířenější platformy v programovacím jazyce C#. To znázorňuje schéma na obrázku 3.4.

Za výhodu nástroje je možné považovat možnost implementace uživatelského rozhraní pro každou cílovou platformu zvlášť. Výsledný vzhled a ovládací prvky aplikace jsou tedy stejné, jako v případě, kdyby by aplikace byla vytvořena oficiálním SDK pro danou platformu. Na tuto vlastnost lze ale také pohlížet jako na nevýhodu. Většina konkurenčních nástrojů používá jednotnou implementaci UI, což může ve srovnání s nástrojem Xamarin přinést úsporu času a nákladů na vývoj. Nespornou výhodou Xamarin.iOS a Xamarin.Android je možnost integrace nativních knihoven, vytvořených pro cílové platformy pomocí jejich oficiálních vývojových nástrojů.

Aplikace, postavené na Xamarin.iOS a Xamarin.Android, je možné vyvíjet v integrovaném vývojovém prostředí Xamarin Studio, nebo pomocí prostředí Microsoft Visual Studio při použití poskytovaného doplňku. Je třeba zmínit, že s vývojem (hlavně s překladem) aplikací pomocí nástroje Xamarin jsou spojena všechna omezení, charakteristická pro vývoj na jednotlivých platformách, zmíněná v sekci 3.1. Pro vytvoření aplikace pro Android,

[18] Jedná se o mezijazyk, do kterého jsou překládány všechny jazyky, podporované v .NET.




Obrázek 3.4: Schéma multiplatformního vývoje pomocí nástroje Xamarin. Zdroj: [32]

iOS a Windows Phone je tak třeba mít k dispozici počítače[19] s operačními systémy Mac OS X a Windows 7 nebo vyšší. Počítač s Mac OS X by v této konfiguraci mohl sloužit pouze pro překlad výsledné aplikace pro platformu iOS. Zatím ale neexistuje nástroj pro tvorbu uživatelského rozhraní pro tuto platformu, který by běžel v OS Windows. Je tak třeba využívat oficiální vývojové prostředí Xcode nebo se spokojit s programováním uživatelského rozhraní výlučně pomocí programovacího jazyka C#.

Xamarin je komerční produkt. Poskytuje však také bezplatné licence, které za omezených podmínek umožňují jeho použití. Nejlevnější komerční licenci, *Indie*, umožňující vývoj plnohodnotných aplikací pomocí Xamarin.iOS nebo Xamarin.Andorid je možné pořídit za 299USD. Tuto částku je tedy pro vývoj na obou podporovaných platformách zaplatit dvakrát. Podporu integrace do prostředí Visual Studio přinášejí až vyšší licence. Ty je možné pořídit za 999 USD, respektive 1899 USD a nabízí oproti základní licenci některé další výhody. V roce 2014 začal Xamarin nabízet bezplatné licence pro studenty a učitele na vysokých školách. [33]

## 3.4 MvvmCross

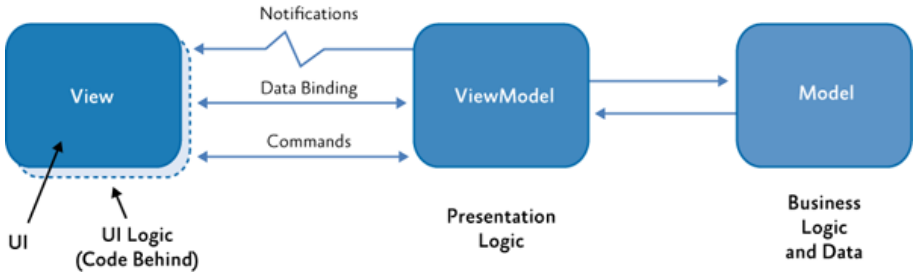
MvvmCross, dále také Mvx, je framework pro vývoj multiplatformních aplikací s pomocí technologií .NET. Jeho základem je architektonický návrhový vzor Model-View-ViewModel.

### 3.4.1 Návrhový vzor Model-View-ViewModel

Tato část vychází především z [15].

Architektura softwarových aplikací je zpravidla tvořena na základě zavedených návrhových vzorů. Jedním z nich je Model-View-ViewModel, dále MVVM. Jedná se o jeden z nejmladších používaných návrhových vzorů architektury aplikace. Je odvozený od starších rozšířených vzorů, zvláště Model-View-Controller (MVC) a Model-View-Presenter (MVP).

[19] Je možné použít nástroje pro virtualizaci.




Obrázek 3.5: Schéma návrhového vzoru Model-View-ViewModel. Zdroj: [15]

Primárním účelem všech těchto vzorů je oddělení datového modelu a aplikační logiky od uživatelského rozhraní. MVVM se toho snaží docílit rozdělením částí aplikace na do tří následujících vrstev:

**Model** – součástí modelové vrstvy je reprezentace dat a veškerá aplikační logika spojená s jejich získáváním a správou. Obsahuje také reprezentaci obchodních procesů. Modelová vrstva a její součásti se v MVVM nijak neliší od modelových vrstev v návrhových vzorech MVC a MVP.

**View (pohled)** – je zde obsažena implementace prvků uživatelského rozhraní a definovány vazby na tyto prvky, zejména vlastnosti (property) a příkazy (command), z vrstvy viewmodel. Tyto vazby jsou realizovány pomocí níže popsaného mechanizmu data binding. Vrstva view je zpravidla reprezentována kódem ve značkovacím jazyce dané platformy, nebo jiným způsobem deklarace grafického uživatelského rozhraní (GUI). Její součástí je také kód pro manipulaci s GUI (code-behind).

**Viewmodel (pohledový model, také VM)** – obsahuje prezentační logiku a data, zobrazovaná vrstvou view. Pohledový model neobsahuje přímou referenci na vrstvu view a nemá informace o její specifické implementaci. Implementuje vlastnosti a příkazy, na které se mohou prvky z pohledové vrstvy pomocí techniky data binding navázat a přijímat notifikace o jejich změně v podobě událostí.

Nedílnými součástmi návrhového vzoru MVVM jsou koncepty data binding (vázání dat) a command (příkaz).

**Data binding** – technika deklarování vztahů mezi prvky vrstvy view a jejich odpovídajícími reprezentacemi ve vrstvě viewmodel. Příkladem může být navázání uživatelského prvku typu textové pole na proměnnou typu řetězec, nacházející se ve vrstvě viewmodel. Vytvořené vztahy mohou být jednosměrné nebo dvousměrné. Při jednosměrném vázání reflektují prvky uživatelského rozhraní jim odpovídající data ve vrstvě viewmodel. Při dvousměrném vázání se změny způsobené interakcí uživatele projevují změnou stavu prvků ve vrstvě viewmodel. Vázaná data mezi vrstvami view a viewmodel mohou být převáděna a upravována pomocí konvertorů (value converter).

**Command** – rozhraní pro reprezentaci a správu akcí a operací, které je možné vyvolat z vrstvy view. Jsou definovány ve vrstvě viewmodel a s prvky vrstvy view jsou propojeny pomocí mechanizmu data binding.

Vztah mezi výše zmíněnými vrstvami a koncepty je patrný z obrázku 3.5.

### 3.4.2 Projekt MvvmCross

Tato část se zabývá frameworkem MvvmCross, který byl vybrán pro implementaci mobilní aplikace, vyvíjené v rámci práce. Většina informací v této sekci pochází z oficiální stránky projektu MvvmCross, [13].

Open-soure projekt MvvmCross umožňuje multiplatformní vývoj aplikací za použití návrhového vzoru MVVM. Je založen na projektu Portable Class Libraries (PCL), který je součástí platformy .NET Framework. Přestože se tento text zabývá zejména platformami pro mobilní telefony, je vhodné na tomto místě vyjmenovat všechny platformy, pro které lze pomocí MvvmCross vyvíjet:

- Xamarin.Android
- Xamarin.iOS
- Windows Universal App (aplikace pro Windows Phone 8.1 a Windows 8.1 Store aplikace)
- WinRT framework pro Windows 8 Store aplikace.
- Silverlight pro Windows Phone 7 a 8
- Xamarin.Mac
- Windows Presentation Foundation

Nativní vývoj aplikací pro Windows Phone (a další moderní platformy ekosystému Microsoft Windows) standardně podporuje využití MVVM přístupu. V případě platforem Android a iOS jsou ale využívány jiné přístupy a architektury[20], které použití MVVM neumožňují. Při využití nástroje Xamarin je sice možné programovat, stejně jako pro platformu Windows Phone, v jazyce C#, ale na architektuře aplikací tento fakt nic nemění. Hlavním přínosem projektu MvvmCross je tedy implementace mechanizmu data binding, který tvoří základ architektury MVVM pro Xamarin.iOS a Xamarin.Android.

Při použití MvvmCross je implementace vrstev architektury MVVM model a viewmodel společná pro všechny platformy. Je tak docíleno maximální přenositelnosti kódu při zachování oddělené implementace nativního uživatelského rozhraní jednotlivých platforem, tj. vrstvy view. Součástí projektu MvvmCross je také množství doplňků, které obsahují implementaci vybraných funkcí, vlastností a technologií pro jednotlivé platformy (např. práce se soubory nebo databází) pomocí společného rozhraní.

Multiplatformní aplikace vytvořená pomocí MvvmCross je zpravidla složena z následujících částí:

- Hlavní, plně přenosný PCL projekt (také „core“ projekt), který obsahuje součásti vrstvy model a viewmodel.
- Několik nativních (také „UI“ projekt) projektů, jejichž počet odpovídá počtu podporovaných platforem, které obsahují především vrstvu view.
- Doplňky (anglicky plugin), z nichž každý se skládá z částí v UI projektech (nativní implementace dané funkcionality) a v core projektu (abstraktní vrstva pro přístup k nativním částem doplňku).

[20] V případě iOS se jedná o vzor MVC. U platformy Android jde architekturu podobnou MVC, která ale nesplňuje některé vlastnosti tohoto vzoru.

```
public class MyClass {
    private readonly IDatabaseService db;
    public MyClass(IDatabaseService database) {
        this.db = database;
    }
    /* ... */
    public static Bitmap TakePicture() {
        var camera = Mvx.Resolve<ICameraService>();
        return camera.TakePicture();
    }
}
```

Výpis 3.1: Příklad využití IoC v MvvmCross.

Následující sekce se zabývají hlavními součástmi a přístupy, použitými v MvvmCross s ukázkami jejich využití.

### Inversion of control v MvvmCross

MvvmCross, stejně jako většina moderních frameworků, využívá přístupu Inversion of Control (obrácené řízení, také IoC). Na tomto místě je vhodné vysvětlit samotný pojem IoC a některé další související přístupy, které MvvmCross využívá:

**Inversion of control** – mechanismus, který přenáší odpovědnost za výběr konkrétní implementace, konstrukci a provázání objektů z programátora na framework.

**Dependency injection (DI)** – návrhový vzor, který implementuje IoC. Dle konkrétní implementace může podporovat několik různých způsobů, jakými programátor může definovat závislosti mezi objekty (např. pomocí konstruktoru nebo setteru).

**Service locator** – další návrhový vzor, který úzce souvisí s IoC. Zapouzdřuje proces spojený se získáváním služeb (závislostí) pomocí abstraktní vrstvy. Využívá k tomu globální registr služeb.

Využití těchto přístupů je zvláště výhodné v multiplatformních aplikacích, kde za jejich pomoci mohou být abstrahovány jednotlivé nativní implementace daných závislostí. Toho je využito například v doplňcích. Za nevýhodu lze považovat fakt, že k implementaci těchto přístupů je v Mvx ve velké míře využito reflexe. Některé chyby pak nemohou být odhaleny při překladu a způsobují chybové ukončení aplikace. Výpis 3.1 obsahuje zjednodušenou ukázku využití IoC v MvvmCross.

### Start a inicializace aplikace

Platformy podporované frameworkem MvvmCross se v mnohém velmi zásadně liší. Framework má za úkol tyto rozdíly abstrahovat. Při startu MvvmCross aplikace jsou nejdříve vytvořeny nativní objekty (např. objekt `AppDelegate` na platformě Xamarin.iOS), ve které je vytvořen objekt `Setup`. Ten obsahuje nastavení pro danou platformu a má za úkol vytvoření a inicializaci potomka třídy `MvxAplication`, který reprezentuje aplikaci a spravuje její životní cyklus.

```
public class Application : MvxApplication {
    public override async void Initialize() {
        this.InitializeIoC();
        this.CreatableTypes()
            .EndingWith("Service")
            .AsInterfaces()
            .RegisterAsLazySingleton();
        this.RegisterAppStart<HomeViewModel>();
    }
}
```

Výpis 3.2: Příklad inicialize MvvCross aplikace v Core projektu.

Během inicializace aplikace (metoda `Initialize()`) jsou zpravidla registrovány služby, které jsou pak v aplikaci využity pomocí výše zmíněných mechanizmů IoC, DI a Service location. Dále je zde vytvořen objekt typu `Start`, který definuje první zobrazený viewmodel. Výpis 3.2 obsahuje typickou podobu této třídy.

### ViewModel

Zdrojový kód MvvmCross aplikace obsahuje jeden nebo více pohledových modelů. Nejdůležitější úkolem VM je reprezentace dat zobrazených vrstvou view. To zahrnuje rozesílání událostí o jejich změně a naslouchání změnových událostí způsobených interakcí uživatele s vrstvou view, tedy realizace mechanizmu data binding. V Mvx je viewmodel vytvořen jako potomek abstraktní třídy `MvxViewModel`, která implementuje zmíněnou funkčnost.

Zdrojový kód ve výpisu 3.3 ukazuje příklad implementace VM s vlastnostmi (property) a příkazem (command), kterých je možné využít z vrstvy view pomocí mechanizmu data binding.

Navigaci mezi jednotlivými VM zajišťuje zejména metoda `StartViewModel<T>()`. Jejím voláním se vytvoří objekt třídy `ViewModelRequest`. O výběr konkrétního typu VM a jeho konstrukci a inicializaci na základě předané instance `ViewModelRequest` se starají třídy `MvxViewModelLocator` a `MvxViewModelLoader`, jejichž základní implementaci framework poskytuje. Programátor si však může zvolit jinou funkčnost vytvořením vlastních tříd, které implementují příslušná rozhraní a jejich registrací při inicializaci aplikace.

Konstrukce a inicializace VM má při použití výchozí implementace třídy `MvxViewModelLoader` čtyři základní fáze:

**Konstrukce** – vytvoření instance a předání jejích závislostí pomocí Dependency Injection.

**Zpracování navigačních parametrů** – jsou definovaný pomocí argumentů jedné nebo více implementací metody `Init()`, které jsou volány pomocí reflexe.

**Obnovení uloženého stavu** – realizováno pomocí přetížení metody `ReloadState()`.

**Dokončení inicializace** – reprezentováno voláním metody `Start()`, ke kterému dochází po dokončení dvou předchozích bodů. Tuto metodu je vhodné přetížit jako asynchronní a načítat v ní perzistentní data.

```
public class LoginViewModel : MvxViewModel {
    /* ... */
    private string _loginString;
    public string LoginString {
        get { return this._loginString; }
        set {
            this._loginString = value;
            this.RaisePropertyChanged(() => this.LoginString);
        }
    }
    private string _passwordString;
    /* ... */
    private MvxCommand _loginCommand;
    public ICommand LoginCommand {
        get {
            if (this._loginCommand == null) {
                this._loginCommand = new MvxCommand(() => {
                    if (this.TryAuthenticate()) {
                        this.ShowViewModel<HomeViewModel>();
                    }
                }
            }
            return this._loginCommand;
        }
    }
}
```

Výpis 3.3: Příklad implementace pohledového modelu.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://..." moreparams="...">
    <EditText
        android:hint="Uživatelské jméno"
        local:MvxBind="Text Password"/>
    <EditText
        android:hint="Heslo"
        android:password="true"
        local:MvxBind="Text LoginString"/>
    <Button
        android:text="Přihlásit"
        local:MvxBind="Click LoginCommand, Enabled=LoginButtonEnabled" />
</LinearLayout>
```

Výpis 3.4: Ukázka definice UI v jazyce XML v MvvmCross pro platformu Xamarin.Android.

```
[Activity (Title = "Přihlášení")]
public class LoginView : MvxActivity{
    protected override void OnCreate(Bundle bundle) {
        base.OnCreate(bundle);
        SetContentView(Resource.Layout.login_screen);
    }
}
```

Výpis 3.5: Ukázka implementace pohledové třídy.

### View

Pohledem (view) se v Mvx rozumí platformně závislá třída, která reprezentuje grafické uživatelské rozhraní zobrazené na obrazovce nebo její části. Zpravidla vychází z tříd nativních API, které mají stejný účel (např. `Activity` a `Fragment` pro platformu Xamarin.Android nebo `UIViewController` pro Xamarin.iOS) a jsou obohaceny o funkčnost potřebnou pro MvvmCross, především podporu pro data binding.

Výpisy 3.4 a 3.5 obsahují příklad implementace pohledu na platformě Xamarin.Android.

### Presenter

Jak již bylo zmíněno výše, navigace je v Mvx aplikaci řízena z core projektu, zpravidla pomocí volání metody VM, `ShowViewModel<T>()`. Tím je vytvořen požadavek na zobrazení VM, `ViewModelRequest`, který však neurčuje, jakým způsobem budou data VM prezentována uživateli. Pro různé platformy existují různé návrhové vzory pro zobrazení obsahu aplikace. V MvvmCross je možné jejich využití pomocí presenterů. Jedná se tak o další nástroj, který pomáhá nabídnout uživateli komfort nativní aplikace.

Každý UI projekt Mvx aplikace musí poskytovat platformově závislý presenter, implementaci rozhraní `IMvxViewPresenter`. Jeho úkoly jsou výběr konkrétní pohledové třídy pro přijatý požadavek (`ViewModelRequest`) a změna prezentace (zobrazení pohledu). Podobně

jako u ostatních součástí frameworku, Mvx poskytuje základní implementaci presenteru pro všechny podporované platformy.

# Kapitola 4

# Návrh

Tato kapitola se zabývá návrhem výsledného řešení dle požadavků specifikovaných v kapitole 2. Je rozdělena na čtyři části. Postupně popisuje návrh společných prvků, mobilní aplikace, webové služby a webové aplikace.

Na tomto místě je vhodné zopakovat, že výsledná aplikace má sloužit jako technologický prototyp. Nejedná se o plně funkční systém, který je připraven ke komerčnímu využití. Z tohoto důvodu jsou některé části navrhovaného systému zjednodušeny. Návrh se zaměřuje zejména na hlavní oblasti zadaní této práce, tj. sběr dat v terénu a jejich slučovaní s hlavní databází.

## 4.1 Doména systému

V aplikační doméně navrhovaného systému se vyskytuje několik základních entit, jejichž vztah je patrný z diagramu doménových tříd na obrázku 4.1.

**Contract** – zakázka. Obsahuje základní data, především zadavatele, datum dodání, identifikátor zakázky a stav, ve kterém se zakázka nachází (*nová*, *předána*, *v reklamačním řízení* apod.). Zpravidla obsahuje několik úkolů, které je pro zhotovení zakázky nutné vykonat.

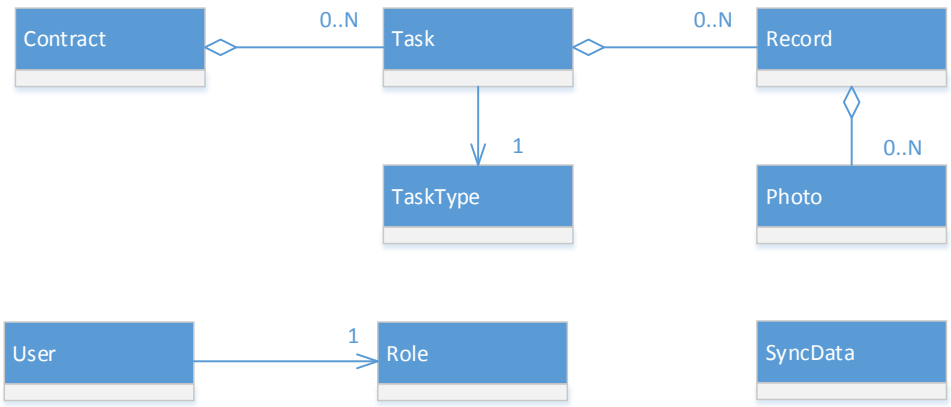
**Task** – úkol zakázky. Obsahuje zejména popis plánovaných prací, jejich typ a pozici. Pro úkol může existovat několik záznamů, které dokumentují průběh prací.

**TaskType** – typ úkolu. Určuje činnost, vykonávanou v rámci úkolu. Může být bodový nebo úsekový. Typy úkolu je možné vytvářet a upravovat ve webové aplikaci a jsou synchronizovány pomocí webové služby na jednotlivé klienty.

**Record** – záznam pořízený v terénu. Obsahuje jednu nebo více fotografií. Pro jeden úkol může existovat několik záznamů, které zachycují například jednotlivé fáze prací. Záznam může být pořízen i nezávisle na úkolu, může tak například sloužit k přípravě přihlášky na výběrové řízení.

**Photo** – pořízená fotografie. Entita uchovává mimo jiné cestu k souboru a čas pořízení. Fotografie musí mít vhodně nastavenou kompresi a rozlišení tak, aby byla její kvalita dostatečná s ohledem na dokumentaci, ale zároveň aby soubor nebyl příliš velký s ohledem na přenos přes internet a úložiště mobilního zařízeni.

**User** – uživatel v systému. Jeho autentizace probíhá pomocí uživatelského jména a hesla.




Obrázek 4.1: Doménové třídy v systému.

**Role** – uživatelská role. Určuje úlohu uživatele v systému a jeho přístupová práva. Uživateli může být přiřazena jedna[1] ze tří rolí: *stavbyvedoucí* (uživatel mobilní aplikace), *manažer zakázek* a *administrátor*, tak jak jsou popsány v kapitole 2.

**SyncData** – záznam o změně. Entitu tohoto typu lze chápat jako návrh na změnu hlavní databáze. Tuto změnu je možné přijmout nebo odmítnout. Synchronizační data jsou vytvářena klienty (mobilní aplikací) a poté odesílány na server. Jejich začlenění do hlavní databáze je prováděno pomocí webové aplikace. Entita obsahuje zejména typ změny (*odstraněno*, *vytvořeno* nebo *upraveno*) a typ entity, Photo, Record nebo Task. Je-li typ změny *vytvořeno* nebo *upraveno*, obsahuje také serializovanou entitu.

## 4.2 Mobilní aplikace

Tato část textu diskutuje části návrhu týkající se mobilní aplikace RoCoManager (*Road Construction Manager*).

### 4.2.1 Případy užití

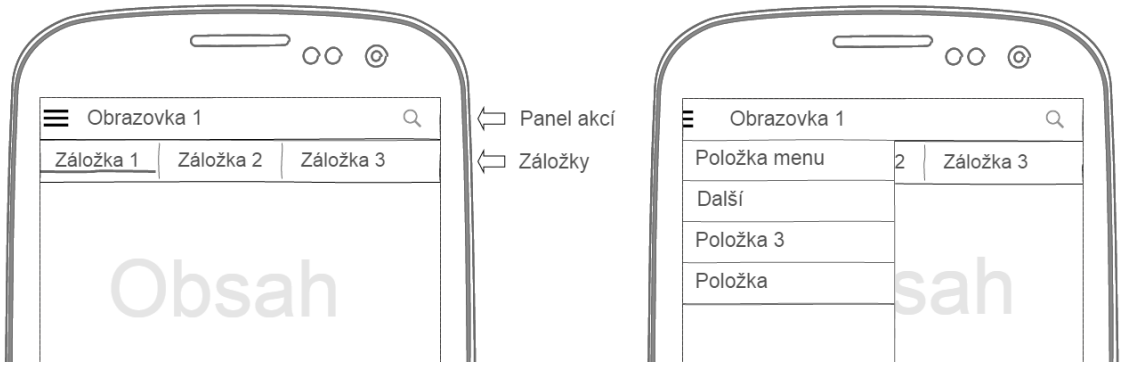
Jak již bylo zmíněno v kapitole 2, mobilní aplikace se zaměřuje na sběr dat v terénu a v menší míře také na řízení prací. Na tomto místě je vhodné přesněji vyjmenovat případy užití aplikace RoCoManager[2]:

1. Zobrazit a změnit nastavení připojení a přihlašovací údaje
2. Stáhnout systémová nastavení ze serveru (především typy úkolů)
3. Stáhnout a zobrazit seznam otevřených[3] zakázek

[1] Protože množina oprávnění každé role je podmnožinou práv následující vyšší role, není nutné přiřazovat uživatelům více než jednu roli.

[2] Pozn.: Protože se jedná o případy užití pro jediný typ aktéra, není na tomto vhodné vkládat do textu diagram případů užití.

[3] Zakázka, pro kterou je možné sbírat data.




Obrázek 4.2: Použité koncepty pro prezentaci obsahu aplikace (vysouvací nabídka, záložky a panel akcí).

4. Zobrazit detail zakázky
5. Synchronizovat (stáhnout) zakázku
6. Zobrazit seznam synchronizovaných zakázek
7. Aktivovat/Deaktivovat vybranou zakázku
8. Spravovat[4] úkoly pro aktivní zakázku
9. Spravovat záznamy pro vybraný úkol
10. Spravovat záznamy pro aktivní zakázku
11. Spravovat záznamy, které nepatří do žádné zakázky
12. Odeslat změny na server

Za nejdůležitější případy užití lze považovat ty, které patří do posloupnosti akcí, jejíž výsledkem je vytvořen a odeslán na server nový záznam pro zakázku. Jedná se o případy užití 1 (při prvním spuštění aplikace), 3, 10 nebo 9 a 12 z výše uvedeného seznamu. Tyto hlavní případy užití je nutné zohlednit při návrhu dalších částí aplikace, obzvláště uživatelského rozhraní.

### 4.2.2 Uživatelské rozhraní

Uživatelské rozhraní musí umožnit komfortní provádění zejména primárních případů užití. Jedním z kritických faktorů, které ovlivňují efektivitu práce s aplikací, je zvolený způsob navigace a prezentace obsahu. Pro navigaci a prezentaci obsahu v mobilních aplikacích existuje mnoho návrhových vzorů. Některé z nich jsou popsány v [12].

Pro aplikaci RoCoManager bylo zvoleno několik následujících konceptů a návrhových vzorů:

[4]Rozumí se zobrazit seznam položek, zobrazit detail, vytvořit novou položku, změnit nebo smazat existující položku.

**Vysouvací nabídka**[5] – představuje hlavní navigační prvek aplikace a umožňuje rychlý přechod mezi hlavními částmi aplikace. Jeho nejdůležitějším znakem je jednoduchý přístup do hlavního menu ze všech obrazovek aplikace a efektivní využití prostoru, jak je možné vidět na obrázku 4.2. V aplikaci RoCoManager se položky menu mění podle toho, zda je aktivní zakázka.

**Záložky (tabs)** – umožňují jednoduchý přechod mezi souvisejícími částmi obsahu. Mimo jiné jsou použity pro přechod mezi obrazovkou s detailem entity a seznamem jejích asociovaných entit (například detail zakázky a seznam úkolů pro zakázku).

**Panel akcí**[6] – tento prvek se typicky nachází v horní části obrazovky. Obsahuje tlačítko pro ovládání vysouvací nabídky, titulek pro aktuálně zobrazený obsah a tlačítka pro kontextové akce, zpravidla ve formě ikon. Zejména v případě akcí, které jsou pro uživatele intuitivně snadno rozpoznatelné podle ikony, je použití panelu akcí velmi efektivní.

Výše uvedené navigační prvky jsou zobrazeny na obrázku 4.2. Všechny jsou s drobnými odlišnostmi implementovatelné (tak aby odpovídali směrnicím pro návrh UI pro jednotlivé platformy) na všech třech nejrozšířenějších mobilních platformách, které byly představeny v části 3.1.

### 4.2.3 Architektura

Hlavní architektura aplikace je z velké míry dána použitím frameworku MvvmCross. Ten především určuje použití návrhového vzoru Model-View-ViewModel. Principy a využití frameworku MvvmCross, a návrhového vzoru MVVM se podrobně zabývá kapitola 3.4. Role MVVM vrstev view a view model jsou zřejmé – dvojice tříd view a viewmodel reprezentuje jednu obrazovku aplikace. Následující text se proto věnuje hlavně vrstvě model, jejíž architekturu je možné dále dělit.

#### Služby

Službami (service) se nejen v kontextu aplikace RoCoManager rozumí třídy, které tvoří mezivrstvu pro komunikaci dalších částí vrstvy model a pohledových modelů. Zapouzdřují interní modelové třídy a poskytují rozhraní pro jejich využití.

Speciální případ služeb tvoří v aplikaci RoCoManager služby pro přístup k perzistentním datům (data service). Jejich hlavními úkoly jsou vytváření mezipaměti perzistentních entit pro efektivnější přístup a vyvářeení záznamů o změnách.

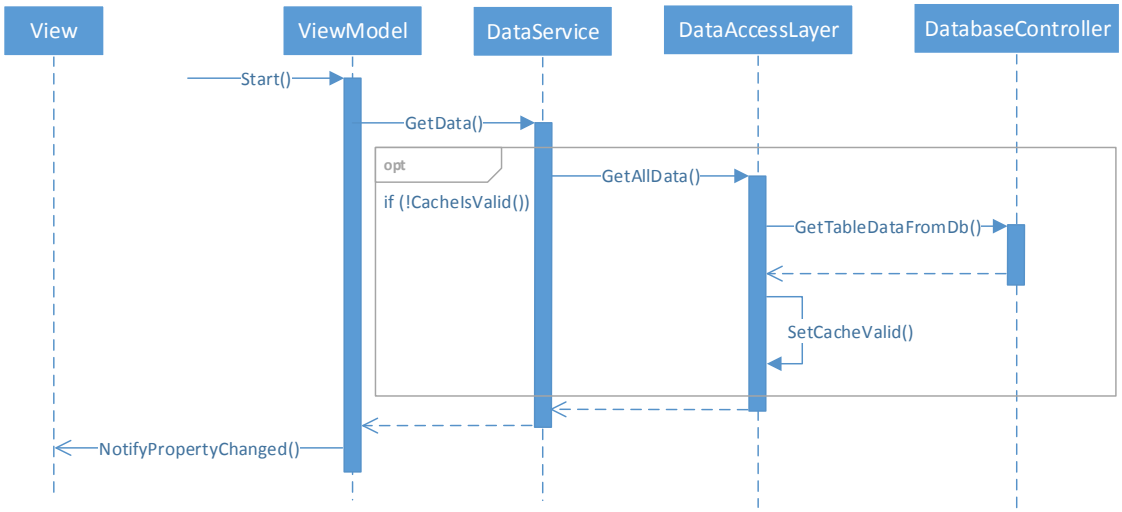
#### Doménové třídy

Další důležitou součást modelové vrstvy tvoří třídy reprezentující doménu systému. Pro každou doménovou entitu mohou v aplikaci RoCoManager existovat až čtyři třídy:

**Modelová** – reprezentuje doménovou entitu.

[5] Na oficiálních stránkách pro vývojáře platformy Android [1] je tento návrhový vzor označován jako Navigation Drawer. V jiných zdrojích také jako Flyout Navigation nebo Side Navigation.

[6] Na platformě android se označuje jako Action Bar, na plaformě iOS jako Navigation Bar, na platformě Windows Phone pak jako App bar.




Obrázek 4.3: Sekvenční diagram přístupu k perzistentním datům aplikace.

**Pohledová** – existuje v případě, že modelovou třídu je nutné obohatit o funkčnost, která je využita při zobrazení jejich dat. Zpravidla vznikne rozšířením modelové třídy nebo vytvořením obalovací třídy (wrapper class).

**Transportní** – vhodně serializovatelná třída, která je použita pro přenos dat mezi aplikací a serverem.

**Perzistentní** – třída pro uložení entity do databáze pomocí ORM[7].

### 4.2.4 Perzistentní data

Protože aplikace RoCoManager umožňuje práci v režimu bez připojení, musí mezi starty uchovávat některá data. Jedná se hlavně o data zakázek, stažená ze serveru, a jejich neodeslané úpravy. Všechny moderní mobilní platformy poskytují programátorům API pro práci s perzistentními daty. Zpravidla se jedná o databázi SQLite[8]. Tyto API zpravidla implementují ORM. Perzistentní data je tak možné při návrhu reprezentovat pomocí tříd.

Přístup k perzistentním datům je realizován pomocí několika vrstev a je zachycen v diagramu sekvence na obrázku 4.3.

### 4.2.5 Žurnál změn

Protože jedním z hlavních úkolů aplikace v systému je odesílání změněných dat na server, musí implementovat mechanismus pro žurnálování změn. Ze sekce 4.2.1 vyplývá, že aplikace musí uchovávat záznamy o změnách doménových entit `Task`, `Record` a `Photo` (Entity typu

[7] Jedná se o techniku pro automatický převod dat z relační databáze do objektů použitého programovacího jazyka.

[8] Jedná se o relační databázový systém, založený na SQL.

```
vytvorZaznam(typEntity, typZmeny, entita) {
    existujiciZaznam = zurnal->ziskejNeodeslanyZaznamProEntitu(typEntity,
     entita->Id);
    if (existujiciZaznam == null) {
        zurnal->vytvorAUlozZaznam(typEntity, typZmeny, entita);
    }
    else if (typZmeny == "smazano"){
        if (existujiciZaznam.TypZmeny == "upraveno") {
            zurnal->vytvorAUlozZaznam(typEntity, typZmeny, entita);
        }
        zurnal->smaz(existujiciZaznam);
    }
    else if (typZmeny == "upraveno"){
        existujiciZaznam->aktualizuj(entita);
    }
}
```

Výpis 4.1: Pseudokód algoritmu pro vytváření žurnálových záznamů

`Contract` v aplikaci měnit nelze.). Jak již bylo zmíněno výše, vytváření záznamů žurnálu změn mají na starosti služby pro přístup k perzistentním datům (data service).

Položku žurnálu je možné vnímat jako perzistentní entitu obsahující především následující vlastnosti:

- Typ entity (*Task*, *Record*, *Photo*)
- Typ změny (*vytvořeno*, *upraveno*, *smazáno*)
- Datum změny
- Příznak odeslání na server
- Stav změny (*čeká*, *přijato*, *odmítnuto*)
- Serializovaná entita před provedenou změnou
- Serializovaná entita po provedené změně

Při vytváření žurnálových záznamů je nutné dbát na to, aby byly validní z pohledu serveru. Například vznikne-li záznam o vytvoření nové entity a ještě před odesláním změn na server je stejná entita upravena, žurnál by měl obsahovat pouze záznam o vytvoření entity. Tato logika je zachycena v algoritmus ve výpisu 4.1.

## 4.3 Webová služba

Webovou službou se obecně rozumí systém pro interakci síťových prvků podle definovaného protokolu. Funguje bezstavově na bázi požadavek-odpověď. Samotný přenos dat probíhá zpravidla pomocí protokolu HTTP a data jsou přenášena ve formátů XML nebo JSON.

Jak již bylo zmíněno výše, webová služba je v kontextu této práce použita pro komunikaci klientů a serveru, obsahujícího hlavní databázi. Návrh protokolu a rozhraní služby primárně vychází z případů užití mobilní aplikace specifikovaných v části 4.2.1. Služba tedy musí podporovat minimálně následující operace:

- Odeslání seznamu otevřených zakázek
- Odeslání všech dat zakázky
- Přijetí záznamu o změně entit

Protože je server, na kterém webová služba běží, veřejně přístupný z internetu, je nutné, aby byly ve webové službě implementovány mechanizmy pro autentizaci a autorizaci uživatelů. Vzhledem k tomu, že je webová služba bezstavová, je nutné aby byly přihlašovací údaje nebo jiný způsob autentizace zahrnuty v každém požadavku.

Jelikož služba běží na stejném serveru jako webová aplikace, je vhodné aby s ní sdílela implementaci některých svých částí, zejména doménovou logiku a přístup ke zdrojům (např. přístup k databázi).

Rozhraní webové služby je dáno návrhem API. V dnešní době jsou pro návrh webových API ve velké míře využívány principy architektonického stylu REST (Representational State Transfer). V architektuře REST je rozhraní definováno pomocí zdrojů, které jsou identifikovány URI (Uniform Resource Identifier, *jednotný identifikátor zdroje*) a je s nimi manipulováno pomocí metod protokolu HTTP. V navrhované webové službě zdroje odpovídají perzistentním doménovým entitám. Nejdůležitější HTTP metody, využívané v RESTful službách jsou:

**GET** – požadavek na získání zdroje.

**POST** – požádá server o vytvoření nového zdroje, jehož typ je dán URI a jehož data se nachází v těle odeslaného požadavku.

**PUT** – požádá server o uložení zdroje identifikovaného URI. Často se využívá pro úpravu existujících zdrojů.

**DELETE** – požádá server o smazání zdroje identifikovaného URI.

### 4.3.1 Návrh API

Pro splnění navrhované funkčnosti, za použití výše popsaného principu REST, musí webová služba implementovat následující rozhraní.

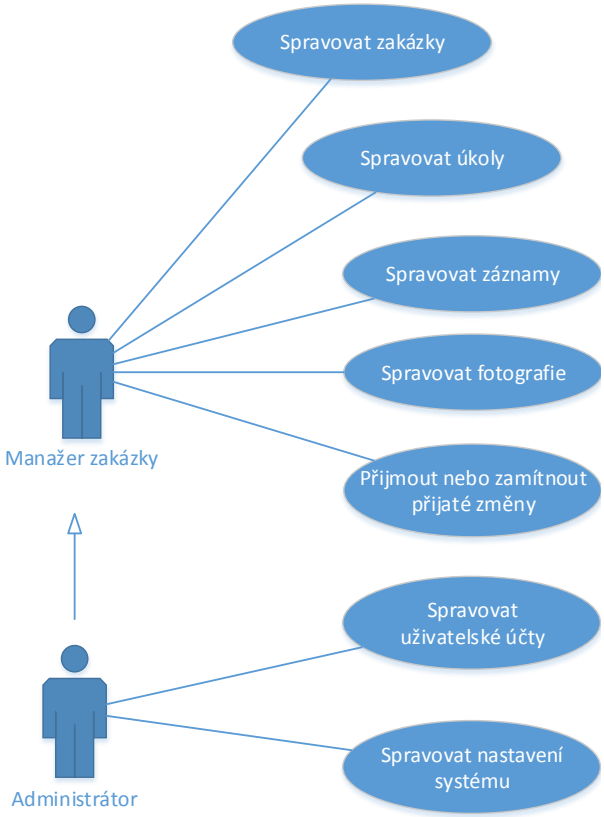
`GET taskTypes` – vrátí seznam typů úkolů.

`GET contracts` – vrátí seznam otevřených zakázek.

`GET contract/<id>` – vrátí všechna data zakázky (tj. včetně úkolů, záznamů a fotografií) podle id zakázky v parametru.

`POST <resource>` – pomocí přijatých dat vytvoří na serveru novou entitu typu daného parametrem `resource`.

`PUT <resource>/<id>` – pomocí přijatých dat aktualizuje na serveru entitu identifikovanou dvojicí parametrů `resource` a `id`.




Obrázek 4.4: Diagram případů užití webové aplikace.[9]

`DELETE <resource>/<id>` – smaže na serveru entitu, která je identifikována dvojicí parametrů `resource` a `id`.

Jak již bylo uvedeno výše, data, která klient odešle na server, lze považovat pouze za návrhy na změnu hlavní databáze. Je tedy vhodné uvažovat další typ zdroje, který odpovídá žurnálovému záznamu popisovanému v sekci 4.2.5. Jeho využitím je možné nahradit poslední tři položky výše definovaného rozhraní (`POST<resource>`, `PUT<resource>/<id>` a `DELETE<resource>/<id>`) jedinou, `POST journalItem`, která v databázi serveru vytvoří záznam o navrhované změně.

## 4.4 Webová aplikace

Stejně jako návrh mobilní aplikace RoCoManager, je návrh webové aplikace ovlivněn použitým programovacím jazykem a frameworkem. Výběrem těchto nástrojů se blíže zabývá následující kapitola 5.2. Na tomto místě je nicméně vhodné zmínit, že pro implementaci webové aplikace byl zvolen programovací jazyk PHP a framework Nette.

Následující část textu se zabývá důležitými aspekty návrhu webové aplikace.

### 4.4.1 Uživatelé a případy užití

[9] Případy užití *Spravovat* se rozumí operace (zobrazit, vytvořit, upravit a smazat.)

Případy užití webové aplikace vycházejí z kapitoly 2 a jsou znázorněny na obrázku 4.4. Uživatelé webové aplikace mohou mít jednu ze dvou rolí, Stavbyvedoucí nebo Administrátor, které jsou popsány v kapitole 2.3.

### 4.4.2 Architektura

V úvodu této kapitoly byl zmíněn výběr frameworku Nette pro implementaci serverové části řešení (webová aplikace a webová služba). Jeho použití do značné míry ovlivňuje návrh aplikace. Především určuje použití architektury Model-View-Presenter (MVP), kterou je vhodné na tomto místě alespoň stručně popsat.

#### MVP

Architektonický vzor MVP vychází ze staršího vzoru Model-View-Controller (MVC). Jeho hlavním cílem je, stejně jako u jiných návrhových vzorů této kategorie, oddělení prezentace od doménové logiky. Jak jeho název napovídá, rozděluje aplikaci do tří vrstev:

**Model** – obsahuje doménovou logiku a práci s perzistentními daty (databází, souborovým systémem). Je nezávislý na zbývajících vrstvách.

**View** – pohledová vrstva. Lze ji vnímat jako množinu šablon, které definují způsob, jakým jsou uživateli zobrazena data.

**Presenter** – spojuje vrstvy model a view. Zpracovává požadavky od uživatele a na jejich základě využívá aplikační logiku implementovanou vrstvou model. Výsledná data předá k vykreslení vrstvě view.

Z pohledu návrhu serverové části této práce jsou velmi významné možnosti sdílení kódu mezi webovou aplikací a webovou službou, které MVP nepochybně umožňuje.

### 4.4.3 Databázová struktura

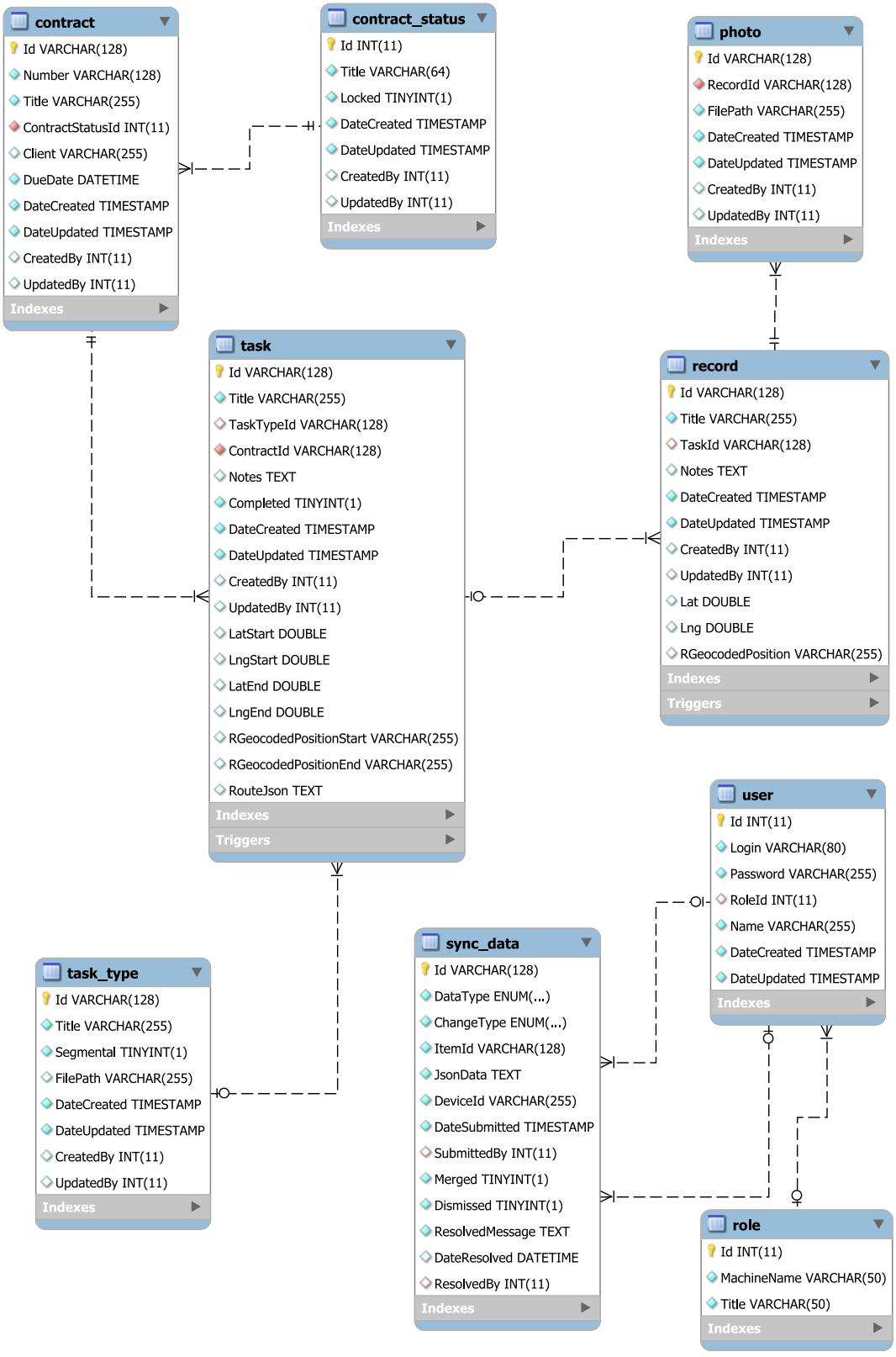
Jak již bylo uvedeno výše v tomto textu, na serveru se nachází hlavní databáze. Tato část se zabývá jejím návrhem. Je zřejmé, že v databázi musí existovat tabulky pro uchování všech doménových entit tak, jak jsou popsány v části 4.1. Další důležitou funkcí databáze je ukládání dat změn, odeslaných klienty. Výsledný návrh databáze je znázorněn v ER diagramu na obrázku 4.5.

#### Univerzální identifikace entit

Některé doménové entity mohou vznikat jak na serveru, tak v klientské mobilní aplikaci. Je nutné zajistit jejich unikátní identifikaci v systému. Jednou z možností, jak toho docílit, je vytvářet dočasné identifikátory na straně klientů a přiřazovat permanentní identifikátory serverem.

V návrhu byl využit jiný přístup – univerzální jedinečné identifikátory (UUID, Universally unique identifier). Jedná se o standard identifikace entit, který zaručuje zanedbatelnou šanci vzniku kolizí. Zpravidla se jedná o posloupnost 128 bitů. Pro vytvoření UUID existují ve většině programovacích jazyků a databázových systémů zabudované funkce. UUID je často reprezentován textovým řetězcem složeným z hexadecimálních číslic a pomlček. V tomto formátu je také ukládán do databáze.

[10] Diagram je v některých ohledech zjednodušen pro zvýšení přehlednosti. Mimo jiné jsou vynechány vazby (cizí klíče) všech entit na tabulku `user`, pro sloupce `CreatedBy` a `UpdatedBy`.




Obrázek 4.5: ER diagram návrhu databáze.[10]

### Uložení změn v databázi

Pro ukládání změnových dat je v návrhu databázového schématu možné použít jeden z následujících přístupů:

**Zrcadlení tabulek**
: Jednou z možností je pro každou entitu, kterou je možné měnit v mobilní aplikaci, vytvořit další tabulku, ve které budou ukládány změny čekající na schválení. Tento přístup není z pohledu implementace příliš efektivní, protože v důsledku jeho využití by v databázi existovaly dvě tabulky s identickou nebo velmi podobnou strukturou.

**Použití příznaku**
: Hlavní nevýhodu prvního přístupu by bylo možné obejít použitím jediné tabulky pro data entit i změny čekající na schválení a jejich rozlišením pomocí příznaku. Za hlavní nevýhodu tohoto přístupu je možné považovat velkou náchylnost k chybám při implementaci (opomenutí příznaku).

**Serializace změn**
: Použití relačních databází je obecně velmi výhodné z pohledu manipulace s daty. Operace, jako jsou vyhledávání, řazení, úpravy apod., jsou nad relačními daty velmi efektivní. Tyto operace jsou však z pohledu dat změn čekajících na schválení nepodstatné. Proto je možné ukládat změnová data do databáze pomocí serializace. Pro ukládání odeslaných změn pak stačí jediná tabulka, která kromě serializované entity obsahuje další informace o navrhované změně, například typ změny, autora nebo stav přijetí.

Jak je patrné z ER diagramu na obrázku 4.5, při návrhu databáze serveru byl pro ukládání dat použit třetí z uvedených přístupů. Pro ukládaní změn slouží tabulka `sync_data`. Protože se jedná o jednu z kritických částí celé aplikace, je vhodné blíže popsat její strukturu.

`Id` – univerzální unikátní identifikátor záznamu o změně.

`DataType` – typ změněné entity (`photo`, `record` nebo `task`).

`ChangeType` – typ provedené změny (`created`, `updated` nebo `deleted`).

`ItemId` – identifikátor změněné entity (je uložen i v serializované entitě i ve speciálním sloupci pro efektivnější vyhledání existující entity v databázi).

`JsonData` – serializovaná entita.

`DeviceId` – identifikátor zařízení, ze kterého byla odeslána změna.

`DateSubmitted` – datum přijetí změny.

`SubmittedBy` – id uživatele, který odeslal změnu.

`Merged` – příznak přijetí změny.

`Dismissed` – příznak zamítnutí změny.

`ResolvedMessage` – komentář k přijetí nebo zamítnutí (zpravidla použit pro uvedení důvodu pro zamítnutí změny).

`DateResolved` – datum přijetí nebo zamítnutí.

`ResolvedBy` – id uživatele, který přijal nebo zamítl změnu.

# Kapitola 5

# Implementace

Tato kapitola se zabývá implementací řešení navrženého v předchozí kapitole 4. Nejedná se o vyčerpávající popis implementace všech částí aplikace, ale zaměřuje spíše na kritické prvky, použité přístupy a zajímavá řešení. Postupně jsou diskutovány části implementace mobilní aplikace RoCoManager a poté serverové části.

Popis implementace serverové části není na rozdíl od jejího návrhu v kapitole 4 rozdělen na dvě části, protože webová služba a webová aplikace byly implementovány v jediném projektu.

## 5.1 Mobilní aplikace

V kapitole 3 bylo uvedeno, že pro implementaci aplikace RoCoManager byl využit framework MvvmCross. Jednou z jeho vlastností, která se dá dle úhlu pohledu považovat za výhodu nebo nevýhodu, je nutnost implementace UI projektu pro každou podporovanou platformu zvlášť. Jinými slovy umožňuje pro každou z cílových platforem implementovat nativní uživatelské rozhraní, které ale není přenositelné. Vzhledem k tomu, že výsledné řešení nemá být komerční produkt ale spíše prototyp komerčního produktu, je dostatečné implementovat UI projekt[1] pouze pro jedinou platformu. Pro implementaci byla vybrána platforma Android. Jako důvody tohoto výběru lze uvést největší podíl na trhu, dostupnost vývojových nástrojů[2] nebo dostupnost testovacích zařízení.

V této části textu jsou diskutovány použité vývojové nástroje a pak významné části implementace aplikace. Často jsou využívány pojmy související s frameworkem MvvmCross, uvedené v části 3.4.2.

### 5.1.1 Použité vývojové nástroje

#### Microsoft Visual Studio

Pro vývoj aplikací pomocí nástroje Xamarin, a tedy i MvvmCross, ma programátor dvě možnosti výběru integrovaného vývojového prostředí (IDE). První z nich je Xamarin Studio, prostředí dodávané se sadou nástrojů Xamarin. Jedná se však o poměrně mladý produkt, který nedosahuje kvalit a efektivity práce jiných zavedených integrovaných prostředí.

[1] Termín UI projekt je popsán v části 3.4.

[2] Jako další možnost se jevila platforma iOS. Pro vývoj pomocí Xamarin.iOS je však nutné využívat nástroj Xcode na počítači s Mac OS, který řešitel neměl k dispozici.

Druhou možností je použití IDE Microsoft Visual Studio a doplňku Xamarin for Visual Studio, který společnost Xamarin vyvíjí. Vzhledem k tomu, že Microsoft ve spolupráci s fakultou informačních technologií nabízí licenci pro Visual Studio studentům zdarma a že se v současné době jedná o vyspělejší nástroj než Xamarin Studio, bylo Visual Studio zvoleno pro vývoj aplikace RoCoManager. Konkrétně byla použita verze VS 2013 Ultimate.

Doplněk Xamarin for Visual Studio přidává do Visual Studia veškerá potřebná nastavení a ovládací prvky pro vývoj Xamarin.Android aplikací, jako například integraci ADB[3], grafické rozhraní pro editaci projektového souboru `Manifest.xml` aj.

Jako další důležité rozšíření Visual Studia lze zmínit doplněk Nuget. Umožňuje snadné přidání komponent a knihoven třetích stran do projektu. Pomocí nástroje Nuget byl do projektu RoCoManger přidán na příklad framework MvvmCross a jeho rozšíření (plugin).

### Resharper

Pro vývojové prostředí Visual Studio existuje celá řada doplňků, které zvyšují efektivitu vývoje. Jedním z nich je produkt Resharper, vyvíjený společností JetBrains. Jako některé nejdůležitější funkce, využité při implementaci aplikace RoCoManager lze jmenovat:

**Analýza kvality kódu** – Resharper v reálném čase analyzuje kvalitu zdrojového kódu, nabízí efektivnější zápisy a upozorňuje na nedodržení nastavených konvencí.

**Efektivní navigace v projektu** – Díky inteligentnímu vyhledávání, je navigace obzvláště ve větším projektu[4] značně zjednodušena. Další zjednodušení navigace přináší kontextová nabídka, která umožňuje rychlý přechod na implementaci rozhraní, odvozené typy apod.

**Generování kódu** – Resharper nabízí generování kódu dle aktuálního kontextu. Například vlastností, podpůrných proměnných nebo zděděných konstruktorů a metod.

**Dekompilátor kódu** – Součástí nástroje Resharper je také dekompilátor, pomocí kterého je možné nahlédnout do implementace kompilovaných knihoven.

**Správa referencí** – Resharper v reálném čase vyhodnocuje připojené knihovny a případně programátora upozorňuje, pokud nejsou nadále potřebné. Při psaní kódu naopak nabízí přidání chybějících referencí.

Hodí se doplnit, že společnost JetBrains nabízí u většiny svých produktů bezplatnou licenci pro studenty a učitele vysokých škol. Nástroj Resharper není výjimkou.

### Genymotion

Důležitým nástrojem při vývoji mobilních aplikací jsou virtuální zařízení, emulátory. Umožňují testovat vyvíjenou aplikaci bez fyzického testovacího zařízení. To programátorovi přináší hned několik výhod, zejména možnost testovat aplikaci na více verzích platformy a zpravidla také rychlejší testování.

Součástí oficiální sady nástrojů Android SDK je emulátor Android Virtual Device (AVD). Jeho funkčnost je ale poměrně omezená. Jako jeho nedostatky lze mimo jiné zmínit

[3] Android Debug Bridge je nástroj sloužící pro připojení testovacích zařízení k počítači a následné ladění vyvíjených aplikací.

[4] RoCoManager obsahuje okolo 200 vlastních tříd.

nemožnost emulace některých senzorů nebo velmi komplikovanou práci s emulovanou kamerou. V AVD také chybí podpora Google Play Services, bez které například není možné testovat aplikace využívající mapy Google. Za největší nevýhodu AVD lze považovat jeho rychlost. Emulovaná zařízení běží i na moderních počítačích velmi pomalu. Není tedy divu, že vývojáři pro emulaci zařízení s platformou Android hledají alternativy.

Jednou z těchto alternativ je nástroj Genymotion společnosti Genymobile. Je postaven na pokročilém nástroji pro počítačovou virtualizaci, VirtualBox od společnosti Sun, což mimo jiné umožňuje běh na většině dnes rozšířených operačních systémech a distribucích. Oproti AVD nabízí hlavně nižší hardwarové nároky a tím vyšší plynulost emulace. Mezi další jeho výhody patří rozsáhlé možnosti emulace senzorů nebo možnost instalace Google Play Services[5]. S ohledem na vývoj aplikace RoCoManager se jako další výhody Genymotion nabízí zmínit snadnou emulaci GPS (pomocí ručně zadaných souřadnic nebo kliknutím na mapu) nebo emulaci fotoaparátu pomocí webové kamery počítače.

Genymotion nabízí prostřednictvím zabudovaného manažeru desítky virtuálních obrazů zařízení na platformě Android a dává tak vývojáři možnost testovat vyvíjené aplikace na různých druzích zařízení, verzích platformy a velikostech a rozlišeních obrazovek. Za nevýhodu Genymotion je možné považovat fakt, že je zdarma pouze pro nekomerční účely.

### 5.1.2 Struktura projektu

Pro popsání struktury projektu je nutné vysvětlit pojmy, které souvisejí s vývojem v ekosystému .NET [16]:

**Solution** (*řešení*) – definuje všechny součásti nutné pro překlad aplikace. Obsahuje jeden nebo více projektů a pomocná metadata. V prostředí MS Visual Studio jsou data řešení uložena v souborech s koncovkou `.sln`.

**Project** (*projekt*) – projekty jsou jednotlivé části řešení. Jejich výstupem (po překladu) jsou zpravidla spustitelné soubory (`.exe`) nebo dynamické knihovny (`.dll`).

Zdrojové kódy aplikace RoCoManager tvoří řešení[6], které se skládá ze dvou projektů:

`RoCoManager.Core` – knihovna, ve které je obsažena přenositelná část aplikace, tedy třídy spadající do vrstev model a viewmodel architektury MVVM. V dalším textu je pro ni používán termín core projekt.

`RoCoManager.Droid` – UI projekt pro platformu android. Z pohledu architektury MVVM obsahuje pouze vrstvu view.

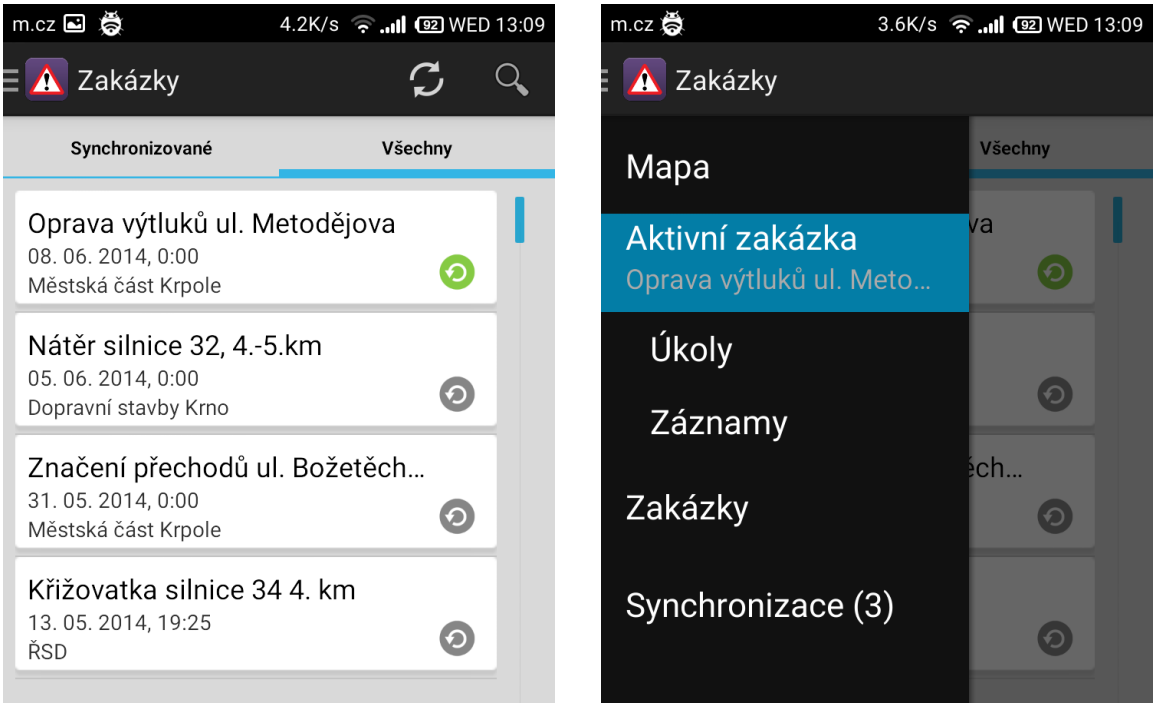
### 5.1.3 Vysouvací nabídka a záložky

V části 4.2.2 byly představeny použité přístupy pro prezentaci obsahu uživateli, především vysouvací nabídka a záložky. Jejich implementace ve frameworku MvvmCross vychází zejména z [19] a [5] a zahrnuje množství úprav.

Položky menu jsou definovány ve třídě `HomeViewModel` v core projektu a mění se v závislosti na tom, zda je aktivována zakázka. Na straně UI projektu existuje třída `HomeView`, která definuje pohledové třídy pro jednotlivé položky a stará se o jejich vykreslení. V

[5] V současné době již služby a API Google Play Services nejsou na virtuálních obrazech Genymotion předinstalované, nicméně je možná snadná instalace neoficiální cestou.

[6] V předchozích částech textu bylo řešení RoCoManager označováno za projekt, protože se jedná o intuitivnější pojem.




Obrázek 5.1: Ukázka navigace v aplikaci RoCoManager.

původní implementaci nebyla animace výměny pohledu plynulá. Záseky byly patrné hlavně při zasouvání nabídky, během kterého probíhá načítání nového pohledu. Byly odstraněny vytvořením jednoduchého pohledového objektu, zobrazujícího indikací načítání (třída `PlaceholderFragment`), který je zobrazen během asynchronní konstrukce cílového pohledu. Pro další zvýšení plynulosti byly přidány animace `FadeIn` a `FadeOut`.

O integraci vysouvací nabídky s frameworkem MvvmCross se stará třída `NavigationDrawerPresenter`, která zachytává požadavky na zobrazení pohledů (`ViewModelRequest`) asociovaných s položkami menu a zobrazuje je ve spolupráci s `HomeView`. Ostatní požadavky deleguje na výchozí presenter.

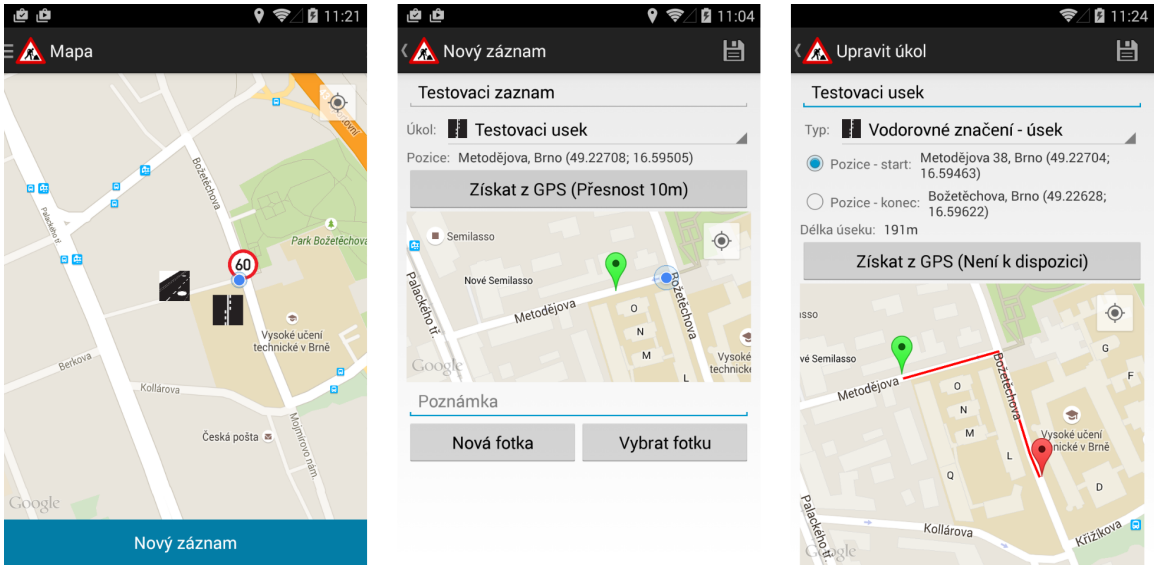
Záložky, sloužící jako druhá úroveň navigace, jsou implementovány pomocí knihovny ViewPageIndicator, převzaté z [5]. Jejich implementace je složena z několika dvojic pohled - pohledový model. Jedna z těchto dvojic reprezentuje hlavní pohled – obrazovku bez obsahu se záložkami zobrazenými v její horní části. Tento hlavní pohled má na starosti vytvoření podřazených pohledů. Na straně vrstvy VM k tomu využívá třídu `ViewModelLoader` a na straně UI projektu statickou metodu `Fragment::Instantiate`.

Výslednou implementaci záložek a vysouvací nabídky je možné vidět na obrázku 5.1.

### 5.1.4 Mapy

Zobrazováni map je v aplikaci implementováno pomocí Google Maps Android API. Mapa je v aplikaci součástí pěti pohledů, v každém z nich má jinou funkci:

**Úvodní obrazovka** – mapa na úvodní obrazovce má za úkol poskytnout uživateli přehled o aktivní zakázce. Pomocí ikon zobrazuje rozmístění jednotlivých úkolů. Ikony jsou určeny typem úkolu.




Obrázek 5.2: Mapové komponenty v aplikaci RoCoManager (mapa aktivní zakázky, editace záznamu a editace úkolu).

**Editace záznamu** – zobrazuje polohu záznamu na mapě. Umožňuje získání aktuální polohy z GPS. Polohu je možné měnit pomocí kliknutí na mapu a přesunu ukazatele.

**Editace úkolu** – zobrazuje polohu úkolu, kterou je stejně jako u záznamu možné získat z GPS. Pokud se jedná o úsekový úkol, je poloha složena ze dvou bodů a je zobrazena spojovací křivka.

**Detail záznamu a detail úkolu** – zobrazují stejné prvky jako odpovídající pohledy pro změnu dat, ale neimplementují interakce s uživatelem.

První tři z mapových komponent je možné vidět na obrázku 5.2.

### Ukazatele

Ukazatelé polohy jsou v mapách Google implementovány pomocí třídy `Marker`. Pro tu ale MvvmCross neobsahuje výchozí implementaci mechanizmu data binding. Pro tento účel byla v UI projektu vytvořena třída MarkerWrapper, která spouští událost `PositionChanged` na základě interakcí uživatele s mapou. Core projekt a UI projekt používají jiné třídy pro reprezentaci pozice. Bylo tedy nutné implementovat vhodný konvertor.

### Spojovací křivka

Umístění úsekových úkolů je dáno dvěma body. Získání křivky a její délky je implementováno na straně serveru pomocí Google Directions API a je popisováno v části 5.2.5. Data křivky jsou získávána pomocí webové služby. Při změně pozice jednoho z bodů se VM (implementováno v `BaseEditTaskViewModel`) pokusí pomocí třídy `MapInfoApiService` získat ze serveru spojovací křivku a uloží ji do vlastnosti `Route`. Odpovídající pohledová třída (`BaseEditTaskView`) pak křivku vykreslí do mapy pomocí třídy `Polyline` z knihovny Google Maps API.

#### Ikony úkolů zakázky

Na straně UI projektu je zobrazení ikon úkolů realizováno třídou `TaskMarkerSet`, která implementuje rozhraní pro mechanizmus data binding mezi kolekcí úkolů a kolekcí ukazatelů. Samotné zobrazení ikony je implementováno pomocí třídy `Marker`. Velikost zobrazené ikony je vypočítána podle rozlišení obrazovky zařízení.

### 5.1.5 Přístup k perzistentním datům a žurnálování změn

Implementace přístupu k perzistentním datům odpovídá návrhu v části 4.2.4. Pro každý typ doménové entity existuje perzistentní třída (např. `ContractSqlEntity`), datová služba (`ContractDataService`), modelová třída (`Contract`) a případně speciální pohledová třída (`ContractWrapper`). Modelové třídy a perzistentní třídy implementují metody pro vzájemný převod. Práce s databází pro všechny druhy perzistentních entit je implementována ve třídě `DataAccessLayer`. Metody třídy `DataAccessLayer` jsou téměř výhradně volány z datových služeb. Pro samotnou práci s databází nabízí MvvmCross vlastní plugin s podporou ORM.

Datové služby slouží k abstrakci přístupu k perzistentním datům. V aplikaci jsou implementovány jako potomci abstraktní třídy `BaseDataService<TData>`. Jejich instance jsou získávány pomocí mechanizmů dependency injection a service location, představených v části 3.4.2. Kromě vytváření mezipamětí mají na starosti také rozesílání asynchronních zpráv o změně dat. Na základě těchto zpráv se zaregistrované instance tříd, zpravidla pohledové modely, dozví, že data již nejsou validní a požádají datovou službu o data nová. Rozesílání asynchronních zpráv je implementováno pomocí MvvmCross doplňku Messenger. Odeslaná zpráva (třída `DataUpdateMessage`) obsahuje především informace o změněné entitě a typu změny. Metody datových služeb jsou zpravidla volány jako asynchronní úlohy. K tomu je využívána třída `AsyncHelper`.

Žurnálování změn, popsané v části 4.2.5, je implementováno ve třídě `DataAccessLayer` a řízeno z datových služeb. Položka žurnálu je reprezentována třídou `DirtyLogSqlEntity`.

### 5.1.6 Uživatelská nastavení

Speciálním typem perzistentních dat jsou uživatelská nastavení a data relace, která je nutné, stejně jako doménové entity, ukládat do databáze. Přístup a ukládání uživatelských nastavení je řízeno třídou `SettingsService`. Ta spravuje perzistentní instance čtyř následujících tříd:

`MapCamera` – pohled kamery na hlavní obrazovce aplikace, tj. pozice středu zobrazené mapy a její přiblížení.

`ServerSettings` – objekt, reprezentující nastavení připojení k serveru – url adresu a přihlašovací údaje.

`ActiveContract` – identifikátor aktivované zakázky.

`SyncedContracts` – kolekce identifikátorů stažených zakázek a časových razítek jejich stažení.

Protože je od každé třídy ukládána jedna instance a jediné operace, které jsou s těmito daty prováděny je čtení a zápis, jsou tyto objekty do databáze ukládány serializované. Tím je možné využít jedinou společnou databázovou tabulku, která je mapovaná na perzistentní

třídu `SerializedObjectEntity`. Ve třídě `SettingsService` jsou pomocí konstant definovány výchozí hodnoty nastavení.

### 5.1.7 Komunikace se serverem

Přenos doménových entit je řízen synchronizační službou implementovanou ve třídě `SyncService`. Její nejdůležitější metody jsou:

`SyncTaskTypes` – implementuje stažení a uložení typů úkolů.

`SyncContractList` – stažení a uložení seznamu otevřených zakázek.

`SyncContract` – stažení celé zakázky včetně zanořených entit. Při ukládání entit do databáze nepřepisuje neodeslané změny. Po stažení a uložení dat se zakázka označuje jako synchronizovaná.

`DeSyncContract` – vymaže všechny zanořené entity synchronizované zakázky.

`UploadChanges` – metoda pro odeslání žurnálových změn na server. Vhodně seřadí uložené změny, prochází je a jednotlivě odesílá na server.

Před odesláním a po přijmutí dat ze serveru je nutné serializovat, respektive deserializovat entitu. Proto existuje pro každý typ doménové entity transportní třída, (např. `ContractJsonObject`. Serializovaný JSON řetězec je převáděn na transportní objekt a zpět pomocí třídy `JsonConvert` z knihovny Newtonsoft.JSON. Z transportních objektů pak jednoduše vznikají modelové objekty, pomocí odpovídajícího konstruktoru (např.
`public Contract(ContractJsonObject jsonObj)`).

Síťová komunikace je implementována pomocí třídy `SyncRestClient`, která implementuje metody pro odesílání HTTP požadavků a vyhodnocování a dekódování odpovědí. Pro HTTP komunikaci je využita knihovna RestSharp.Portable. Aby bylo možné odeslat jediný požadavek obsahující serializovaný objekt i připojený soubor (při odesílání fotografií), bylo nutné pro odesílání dat metodou POST využít kódování `multipart/form-data`, kterou využívají webové prohlížeče při odesílání formulářů, a zavést konvenci pro jména parametrů.

### 5.1.8 Pozice a reverzní geokódování

Při vytváření a úpravě úkolů a záznamů může uživatel využít svoji aktuální pozicí získanou pomocí GPS. Pohledové modely pro editaci úkolů a záznamů spustí vyhledávání pozice hned v průběhu své inicializace. Na změny aktuální pozice jsou upozorňovány pomocí asynchronních zpráv. Tato funkčnost je implementována třídou `LocationService`. Pro získávání dat z GPS modulu je použit plugin MvvmCross.Plugins.Location.

Reverzní geokódování je proces získání člověkem čitelných informací o místě daném souřadnicemi. Při změně pozice editovaného záznamu nebo úkolu vyšle pohledový model pomocí třídy `MapInfoApiService` požadavek na server. V případě úspěšného získání odpovědi informace (adresu nebo číslo silnice) zobrazí. Implementací reverzního geokódování na straně serveru se zabývá část 5.2.5.

### 5.1.9 Vytváření fotografií

Pro záznam (Record) může uživatel mobilní aplikace přiložit existující fotografie z úložiště telefonu nebo vytvořit nové pomocí integrovaného fotoaparátu. Přesně tuto funkčnost nabízí

doplněk MvvmCross.Plugins.PictureChooser použitý v aplikaci. Pro ukládání fotografií do složek aplikace slouží další plugin, MvvmCross.Plugins.File.

## 5.2 Serverová část

Následující část textu se zabývá implementací serverové části řešení. Jedná se o projekt postavený na frameworku Nette. Postupně jsou popisovány vývojové nástroje a kritické části implementace.

### 5.2.1 Použité vývojové nástroje

#### PhpStorm

Přesto, že i dnes preferuje pro vývoj PHP aplikací nezanedbatelné množství programátorů různé druhy poznámkových bloků, existuje mnoho pokročilých integrovaných vývojových prostředí. Jedním z nich je PhpStorm od společnosti JetBrains, použitý při vývoji serverové části řešení práce. Mezi některé jeho funkce, které se při vývoji aplikace osvědčily patří:

**Integrace s Xdebug** – Xdebug je nástroj pro ladění PHP aplikací. PhpStorm podporuje jeho využití a tím umožňuje programátorovi využít všechny obvyklé ladící prostředky, jako sledování proměnných nebo zarážky.

**Inteligentní navigace** – Podobně jako nástroj Resharper od stejné společnosti, který je popisován v sekci 5.1.1, umožňuje i PhpStorm efektivní navigaci po projektu, zejména pomocí inteligentního vyhledávaní a kontextových nabídek.

**Chování editoru podle kontextu** – Editor dokáže rozeznat použitý jazyk v textových řetězcích jiného jazyka a tomu přizpůsobit zvýrazňování syntaxe.

**Podpora javascriptu** – PhpStorm integruje rozsáhlou podporu jazyka javascript. Především nabízí ladící nástroje a efektivní našeptávač, díky kterému je programátor mnohem méně často nucen procházet dokumentace použitých knihoven.

PhpStorm je stejně jako nástroj Resharper od společnosti zdarma pro studenty a učitelé na vysokých školách.

#### Postman

Při vývoji serverové části práce bylo často nutné testovat webovou službu. K tomu se výborně hodí nástroj Postman - REST Client. Jedná se o doplněk do prohlížeče Google Chrome, který umožňuje ruční vytváření HTTP požadavků a jejich odesílání. Uživatel má možnost nastavit metodu, hlavičky, typ autorizace i případné tělo požadavku. Nástroj pak umí formátovat odpověď podle typu obsahu (pro standarty XML a JSON), procházet hlavičky odpovědi nebo zobrazovat aktuálně nastavené soubory cookie. Odeslané požadavky se ukládají a je možné je třídit do kolekcí, znovu odesílat nebo upravovat.

### 5.2.2 Operace s doménovými daty

Serverová část řešení má typickou strukturu projektu postaveného na frameworku Nette. Pro každý typ doménové entity existuje ve vrstvě model třída pro práci s perzistentními daty (například `Contracs`), presenter (`ContractPresenter`) a několik šablon.

Protože mají presentery pro různé druhy entit stejné metody (např. zobrazení jednotlivých šablon nebo zpracování editačního formuláře), bylo možné většinu funkčnosti přesunout do abstraktní třídy `BaseModelPresenter`. Podobná situace je i u modelových tříd pro práci s perzistentními daty entit. Společná funkcionalita je implementována v abstraktní třídě `BaseModel`.

Pro zobrazení seznamu entit je použita knihovna TwiGrid. Implementuje tabulkový výpis dat z databáze, jejich řazení a stránkování. Pro zobrazení a zpracování formulářů existuje v Nette komponenta Forms.

### 5.2.3 Mapy

Mapy ve webové aplikaci slouží, stejně jako v mobilní aplikaci RoCoManager, k zobrazení a úpravě pozice úkolů a záznamů. Stejně jako v mobilní aplikaci je i na webu využito Google Maps API, tentokrát jeho verze pro javascript.

Veškerá práce s mapami je implementována v souboru `maps.js`, který mimo jiné definuje čtyři objekty pro různá zobrazení mapy:

`RecordDetailMap` – nejjednodušší typ mapy. Do DOM objektu předaného konstruktorem vykreslí mapu a ukazatel pozice. Je použit pro zobrazení detailu záznamu.

`TaskDetailMap` – mapa pro zobrazení detailu úkolu. V případě bodového úkolu má stejnou funkčnost jako `RecordDetailMap`. Pokud je zobrazena pro úsekový úkol, vykreslí kromě dvou ukazatelů pozice také spojovací křivku.

`RecordUpdateMap` – jedná se o objekt, který v konstruktoru dostane DOM uzel zobrazeného formuláře. Ve formuláři nalezne vstupní pole s hodnotami souřadnic pozice a nahradí je mapou s ukazatelem. Při změně pozice ukazatele polohy na mapě pak aktualizuje hodnoty nahrazených formulářových polí a pošle asynchronní požadavek na webovou službu pro získání informací o pozici (reverzní geokódování).

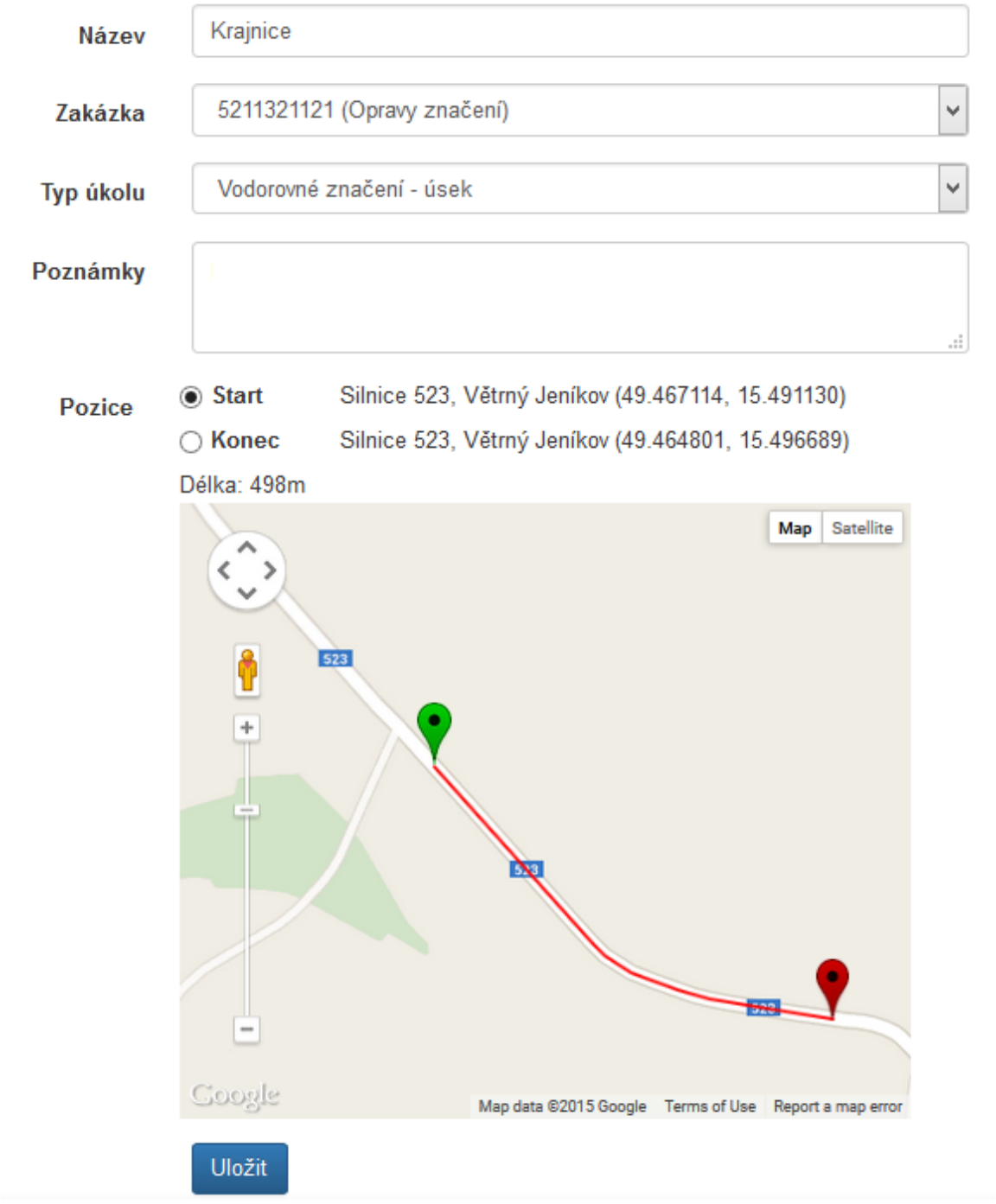
`TaskUpdateMap` – stejně jako objekt `RecordUpdateMap` pracuje s formulářem. Přidává možnost manipulace se dvěma ukazateli (podle zvoleného typu úkolu) a vykreslování spojovací křivky, kterou získá pomocí asynchronního požadavku na webovou službu. Výslednou mapovou komponentu je možné vidět na obrázku 5.3.

### 5.2.4 Zpracování přijatých změn

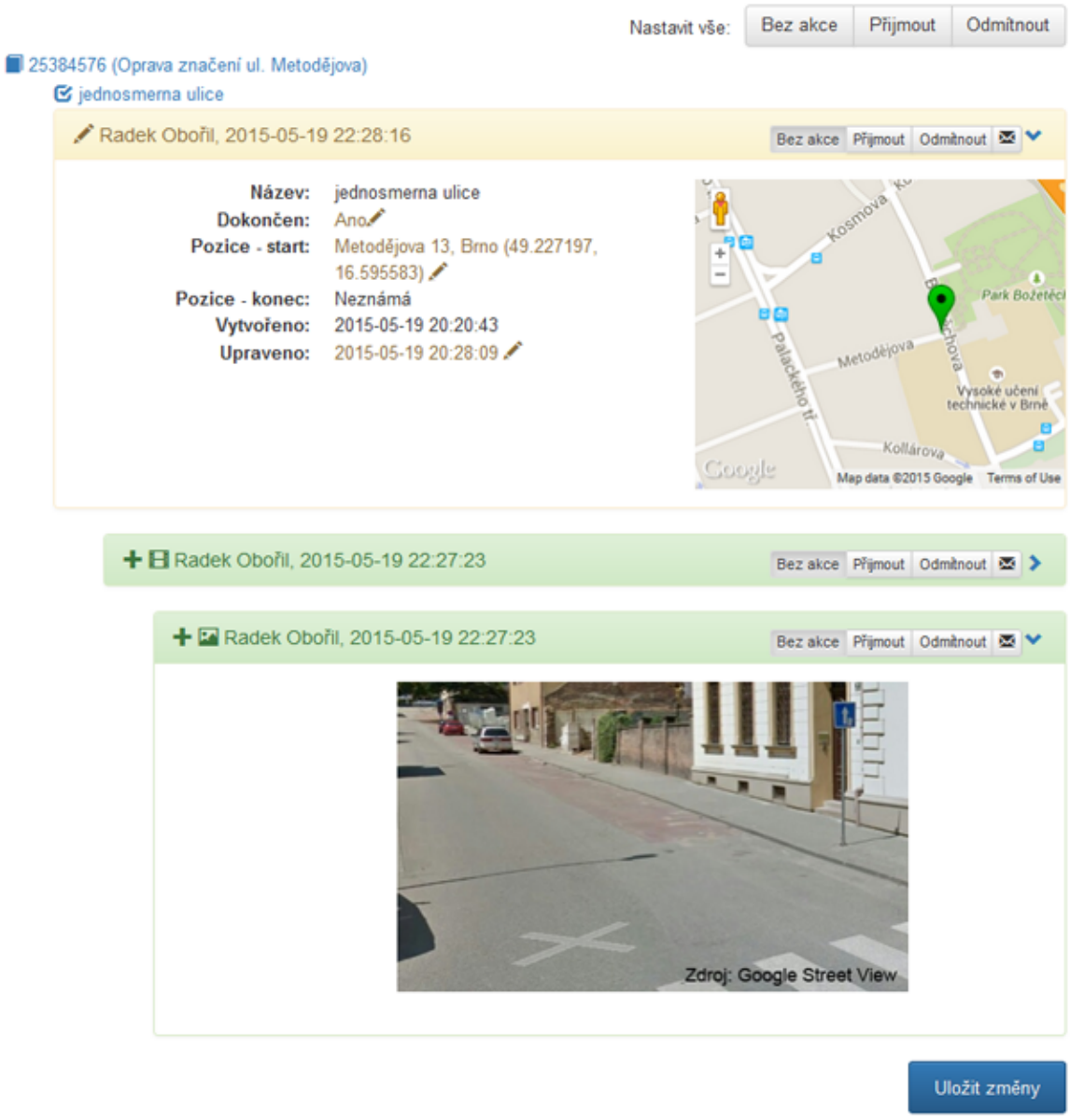
Přijímání a odmítání změn zaslaných klienty je jedna ze stěžejních částí řešení. Musí tedy uživateli nabízet přehledné rozhraní. Jednotlivé změny je třeba zařadit do kontextu zakázky. Proto jsou změny uživateli prezentovány ve formě stromové struktury. Kořeny stromu reprezentují zakázky a jeho uzly představují zanořené entity. Pro jednotlivé uzly jsou vypsány změny čekající na schválení nebo odmítnutí. Uzly stromu reprezentují existující entity v databázi. V některých případech ale nadřazené entity nemusí existovat (například pokud je na zařízení vytvořen úkol a pro něj několik záznamů) a části stromu jsou tak sestaveny pouze na základě změnových dat.

Na straně serveru je sestavení stromu implementováno pomocí tříd `ChangeTree`, `ChangeNode` a `ChangeData`. Presenter pak předá data šabloně. Pro zobrazení stromu je využíváno několik definovaných bloků[7] podle typu změny a typu změněné entity. Pro vyšší přehlednost jsou jednotlivým typům změn přiřazeny barvy a typům entit ikony. Na straně klienta

[7] Šablonový systém Latte frameworku Nette umožňuje definovat bloky, které mohou být použity vícekrát na jedné stránce.




Obrázek 5.3: Editační formulář úkolu s použitím Google Maps API.




Obrázek 5.4: Rozhraní pro zpracování přijatých změn.

```php
/** @GET contract/<id> */
public function actionGetContract($id) {
    $this->returnResponse(
        $this->contractsModel->getContractWithChildrenArr($id)
    );
}
```

Výpis 5.1: Ukázka využití frameworku Nette společně s knihovnou Drahak\Restful pro implementaci webového API.

(webového prohlížeče) jsou využity knihovny Twitter Bootstrap a jQuery. Prostředí pro zpracování změn je zobrazeno na obrázku 5.4.

### 5.2.5 Webová služba

Framework Nette umožňuje rozdělení souvisejících částí do jednotlivých modulů, které mají svojí vlastní MVP strukturu nebo sdílí některé části kódu. Tímto způsobem je implementována webová služba. Vzhledem k tomu, že sdílí modelovou vrstvu s webovou aplikací a neobsahuje vrstvu pohledů, jedná se pouze o implementaci vrstvy presenter.

API pro manipulaci s doménovými entitami, tak jak je navrženo v části 4.3, je implementováno třídou `ApiPresenter` pomocí knihovny Drahak\Restful. Ta obsahuje funkčnost pro jednoduché odesílání odpovědí ve formátu JSON nebo XML a rozšiřuje možnosti asociace URL adres s aplikačními požadavky (routing). Díky této třídě je možné asociovat URL adresy pomocí anotací přímo v presenteru, tak jak je ukázáno ve výpisu 5.1.

Autentizace uživatele webové služby je implementována pomocí metody basic access authentication. Při jejím použití jsou přihlašovací údaje posílány v hlavičce, zakódované pouze pomocí kódování Base64. Přihlašovací údaje jsou tak jednoduše zachytitelné. V reálném nasazení by bylo možné tuto metodu považovat z pohledu bezpečnosti za použitelnou pouze v kombinaci s využitím protokolu HTTPS.

#### Reverzní geokódování a spojovací křivky

Reverzní geokódování je využíváno jak webovou, tak i mobilní aplikací. Proto je vhodné, aby byla implementace společná. Jako logické řešení se nabízí o tyto funkce rozšířit webovou službu. Výsledné API je implementováno ve třídě `MapServicesApiPresenter` a podporuje dva požadavky:

`GET rgeocode/<latLng>` – získává informace o pozici podle souřadnic pomocí synchronních požadavků na webovou službu Google Geocoding API. Odpověď pak zpracuje (tj. sestaví vhodný textový řetězec) a vrátí.

`GET getRoute/<latLngStart>/<latLngEnd>` – získání spojovací křivky dvou bodů na silnici. Přestože webová služba Google Directions API slouží primárně pro navigaci, je možné ji využít i pro získání spojnic mezi dvěma nedalekými body a její délky. Pro každý požadavek jsou postupně vyslány dva dotazy s opačným pořadím souřadnic začátku a konce úseku. Vrácen je kratší z nich. Tím je možné obejít chybný výsledek na jednosměrných ulicích. Vrácená odpověď obsahuje délku úseku v metrech a zakódovanou spojovací křivku.

Synchronní požadavky jsou odesílány pomocí knihovny Kdyby\Curl, která využívá PHP modul cURL.

# Kapitola 6

# Testování

Vzhledem k tomu, že výsledné řešení má sloužit pouze jako prototyp a nebylo vyvíjeno pro konkrétního zákazníka, nemohlo být předáno k akceptačním testům. Funkčnost byla v průběhu vývoje testována inkrementálně při přidání nových částí. Vzhledem k rozsáhlosti aplikace se často objevovaly regresní chyby. Vznikl proto testovací scénář, při kterém byla ověřena většina funkcí aplikace. Aplikace RoCoManager byla testována především na mobilním telefonu Xiaomi Mi2s a několika virtuálních zařízeních pomocí emulátoru Genymotion, zmíněného v části 5.1.1.

## 6.1 Uživatelské testování

Mobilní aplikace RoCoManager byla podrobena uživatelskému testování. Jeho účelem bylo zhodnotit použitelnost implementovaného uživatelského rozhraní a intuitivnost ovládání aplikace.

Test probíhal podle následujícího scénáře. Uživateli byl stručně představen koncept řešení a vztahy mezi modelovými entitami. Poté dostal několik zadaných úkolů, které vycházely z hlavních případů užití, specifikovaných v části 4.2.1. Každý úkol se nejdříve pokusil splnit samostatně. Pokud uživatel nemohl úkol splnit, nahlédl do přiložené nápovědy. Po ukončení testu vyplnil dotazník. Nakonec následoval neformální rozhovor o zkušenostech s aplikací.

Pomocí dotazníku byly od respondentů testu získány především následující informace:

**Využívání mobilních zařízení** – několik prvních otázek se týkalo znalostí a zkušeností uživatele s mobilními zařízeními a mobilními operačními systémy.

**Intuitivnost aplikace** – bylo zjišťováno kolikrát a při jakých úkolech použil uživatel při plnění testu nápovědu.

**Kvalita uživatelského rozhraní** – dotazník obsahoval několik otázek, které měly za úkol zjistit, zda práce s aplikací přišla uživatelům příjemná a efektivní.

### 6.1.1 Výsledky

Testu se zúčastnilo celkem 8 respondentů, z nichž všichni pravidelně využívají chytré mobilní telefony a mají zkušenosti s mobilní platformou Android nebo iOS. Detailnější pohled na využívání mobilních zařízení respondenty testu nabízí tabulka 6.1.

| **Činnost** | **Respondenti** [%] |
|---|---|
| Psaná komunikace (SMS, komunikační aplikace) | 100 |
| Vyhledávání míst na mapě nebo navigace | 100 |
| Prohlížení internetu | 100 |
| Fotografování | 87,5 |
| Hraní her | 25 |
| Přístup na sociální sítě | 25 |

Tabulka 6.1: Pravidelně využívané funkce mobilních zařízení respondenty testu.

| **Tvrzení** | **Respondenti**[1] [%] |
|---|---|
| V aplikaci jsem se jednoduše zorientoval(a). | 87,5 |
| Vzhled aplikace na mě působil příjemně. | 100 |
| Ovládání aplikace mi přišlo složité. | 25 |
| Ovládání aplikace mi přišlo náchylné k chybám | 12,5 |

Tabulka 6.2: Hodnocení uživatelského rozhraní respondenty testu.

Nápověda byla při testování využita celkem čtyřikrát z možných 56 využití, tedy přibližně v 7% případů. To naznačuje dobrou přehlednost uživatelského rozhraní intuitivnost ovládání, a to i přesto, že respondenty lze považovat za zkušené uživatele mobilních zařízení.

Otázky, ve kterých měli respondenti testu posoudit kvalitu uživatelského rozhraní byly pokládány formou tvrzení s možností odpovědí „Ano“, „Spíše ano“, „Nevím“, „Spíše ne“ a „Ne“. Jak je možné vidět v tabulce 6.2, hodnocení uživatelů bylo převážně kladné.

Na základě zpracovaných výsledku a neformálních rozhovorů s respondenty testu bylo v aplikaci provedeno několik úprav, jejichž cílem bylo dále zvýšit intuitivnost aplikace. Především byly změněny názvy několika příkazů. Například výraz *synchronizovat*, který označoval přenos dat směrem ke klientovi i na server byl nahrazen za výrazy *odeslat* a *stáhnout*. Dále byly přidány a upraveny některé dialogy.

[1] Jsou zahrnuti uživatelé, kteří odpověděli kladně, tedy „Ano“ nebo „Spíše ano“.

# Kapitola 7

# Závěr

Úkolem této práce, k jejímuž vzniku dala prvotní impulz společnost Q2 Interactive, bylo vytvořit mobilní aplikaci pro podporu a řízení dopravních staveb, která bude sloužit jako technologický prototyp pro marketingové účely.

Výsledkem je mobilní aplikace RoCoManager a serverová část řešení. Aplikace RoCoManager umožňuje pomocí integrovaného fotoaparátu, map a GPS dokumentovat průběh dopravních staveb v terénu a nasbíraná data odesílat na server. Je implementována pomocí nástroje Xamarin a frameworku MvvmCross. Jejich kombinace umožňuje vývoj v jazyce C# pro všechny nejpoužívanější mobilní platformy s možností přenositelnosti kódu při využití nativní implementace uživatelského rozhraní. V rámci práce byla implementována verze aplikace pro platformu Android. Serverová část řešení se skládá z webové služby a webové aplikace. Webová služba zajišťuje komunikaci mezi klienty a serverem, především přenos nasbíraných dat. Webová aplikace umožňuje správu hlavní databáze zakázek. Její součástí je také rozhraní pro slučování dat odeslaných klienty z terénu s hlavní databází.

Kapitoly 2 a 3 vznikly převážně v rámci semestrálního projektu a slouží jako teoretický základ práce. V kapitole 2 jsou analyzovány požadavky na výsledné řešení a nejdůležitější technologické vlastnosti mobilních zařízení, které byly při implementaci aplikace RoCoManager využity. Kapitola 3 se zabývá problematikou trhu s mobilními zařízeními a aplikacemi a věnuje se třem nejvýznamnějším současným mobilním platformám. Dále jsou zde popisovány principy multiplatformního vývoje mobilních aplikací a představeny nástroje použité pro tvorbu výsledné aplikace, Xamarin a MvvmCross. Framework MvvmCross je představen podrobněji, protože se jedná o stěžejní komponentu pro implementaci mobilní aplikace RoCoManager.

Návrh aplikace RoCoManager a její serverové části je popsán v kapitole 4. Zejména je zde definována doména systému, identifikovány hlavní případy použití a jsou analyzovány jednotlivé části řešení. Implementací navrženého řešení se zabývá kapitola 5. Aplikace RoCoManager byla podrobena uživatelskému testovaní, jehož metodika a výsledky jsou popsány v kapitole 6.

Výsledná aplikace má mít především úlohu technologického prototypu. V případě zájmu zákazníků může sloužit jako základ pro vývoj komerčního produktu. Možná vylepšení a budoucí vývoj je tak možné rozdělit do dvou kategorií. Vývoj prototypu by mohl pokračovat rozšířením možností dokumentace úkolů (například o možnost připojovat videa nebo vytvářet hlasové poznámky) a podporou různých mapových podkladů. Pokračovaní vývoje směrem ke komerčnímu produktu by spočívalo zejména v implementaci podnikových procesů za úzké spolupráce se zákazníkem. Tím by byly rozšířeny možnosti aplikace v oblasti řízení staveb a podnikového řízení.

# Literatura


[1] Navigation Drawer. developer.android.com, [online, navštíveno 29.04.2015].
URL
https://developer.android.com/design/patterns/navigation-drawer.html

[2] About the iOS Technologies. Apple Inc., [online, navštíveno 08.01.2014].
URL
https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/Introduction/Introduction.html

[3] Which Developer Program is for you? Apple Inc., [online, navštíveno 08.01.2014].
URL https://developer.apple.com/programs/which-program/

[4] Brunclík, A.; Vorel, V.; kolektiv: *Páteřní síť dálnic a rychlostních silnic v ČR*. Agentura Lucie spol. s. r. o., 2009.

[5] Cielecki, T.: Fragments and ViewPager with Mvx. [online, navštíveno 20.11.2014].
URL http://blog.ostebaronen.dk/2013/07/fragments-and-viewpager-with-mvx.html

[6] Silnice a dálnice v České republice | 2013. Ředitelství silnic a dálnic ČR, [online, navštíveno 06.01.2014].
URL http://www.rsd.cz/rsd/rsd.nsf/0/00712811179E3270C1257C08005CD18B/$file/RSD2013cz.pdf

[7] Zpráva o vývoji trhu elektronických komunikací se zaměřením na rok 2012. Český telekomunikační úřad, [online, navštíveno 07.01.2014].
URL http://www.ctu.cz/cs/download/statisticke_udaje/rok_2013/zprava_vyvoj_trhu_ek_2012.pdf

[8] Gartner Says Smartphone Sales Grew 46.5 Percent in Second Quarter of 2013 and Exceeded Feature Phone Sales for First Time. Gartner, [online, navštíveno 01.04.2014].
URL http://www.gartner.com/newsroom/id/2573415

[9] Gartner Says Smartphone Sales Surpassed One Billion Units in 2014. Gartner, [online, navštíveno 01.05.2015].
URL http://www.gartner.com/newsroom/id/2996817

[10] Lee, H.; Chuvyrov, E.: *Beginning Windows Phone App Development*. Apress, 2012, ISBN 978-1-4302-4135-5.

[11] Lee, W.-M.: *Beginning Android Application Development*. Indianapolis: Wiley Publishing, Inc, 2011, ISBN 80-248-0124-8.

[12] Lehtimäki, J.: *Smashing Android UI*. Chichester: John Wiley & Sons, Inc., 2013, ISBN 978-1-118-38728-3.

[13] Lodge, S.: MvvmCross v3. [online, navštíveno 10.01.2014].
URL https://github.com/MvvmCross/MvvmCross

[14] Account types, locations, and fees. Microsoft, [online, navštíveno 10.01.2014].
URL http://msdn.microsoft.com/en-us/library/windows/apps/jj863494.aspx

[15] Implementing the MVVM Pattern. Microsoft, [online, navštíveno 10.01.2014].
URL
http://msdn.microsoft.com/en-us/library/gg405484%28v=pandp.40%29.aspx

[16] Solutions and Projects. [online, navštíveno 2.5.2015].
URL https://msdn.microsoft.com/en-us/library/b142f8e7.aspx

[17] Katalog mobilů. Mobilmania.cz, [online, navštíveno 07.01.2014].
URL http://www.mobilmania.cz/katalog-mobilu/sc-63-c-1/default.aspx

[18] .NET Framework Architecture. Mono Project, [online, navštíveno 10.01.2014].
URL http://www.mono-project.com/.NET_Framework_Architecture

[19] Montemagno, J.: Effective Navigation in Xamarin.Android: Part 1 - Navigation Drawer. [online, navštíveno 20.11.2014].
URL http://motzcod.es/post/60427389481/
effective-navigation-in-xamarin-android-part-1

[20] Rapant, P.: *Družicové polohové systémy*. VŠB-TU Ostrava, 2002, ISBN 80-248-0124-8.

[21] Cross Platform Tool Benchmarking 2013. research2guidance, [online, navštíveno 10.01.2014].
URL http://www.research2guidance.com/shop/index.php/
cross-platform-tool-benchmarking-2013

[22] Shackles, G.: *Mobile Development with C#*. Sebastopol: O'Reilly Media, Inc, 2012, ISBN 978-1-449-32023-2.

[23] Smyth, N.: *iOS 7 App Development Essentials*. eBookFrenzy, 2013, ISBN 978-0-9860273-5-2.

[24] Mapa pokrytí. T-Mobile Czech Republic, [online, navštíveno 07.01.2014].
URL http://www.t-mobile.cz/web/cz/podpora/mapa-pokryti

[25] Tabor, B.; Rutkas, C.; Lieberman, L.: Windows Phone 8 Development for Absolute Beginners. Microsoft, [online, navštíveno 10.01.2014].
URL https://absolutebeginner.codeplex.com/downloads/get/707242

[26] Mapa pokrytí. Telefónica Czech Republic, [online, navštíveno 07.01.2014].
URL http://www.prodej-o2.cz/o2-mobilni-internet.aspx

[27] NAVSTAR GPS User Equipment Introduction. U.S. Coast Guard Navigation Center, [online, navštíveno 08.01.2014].
URL http://www.navcen.uscg.gov/pubs/gps/gpsuser/gpsuser.pdf

[28] Mapa pokrytí. Vodafone Czech Republic, [online, navštíveno 07.01.2014].
URL http://www.vodafone.cz/mapa-pokryti/

[29] Vávrů, J.: *iPhone: vývoj aplikací*. Praha: Grada, 2012, ISBN 978-80-247-4457-5.

[30] General Packet Radio Service. Wikipedia, [online, navštíveno 07.01.2014].
URL http://en.wikipedia.org/wiki/General_Packet_Radio_Service

[31] LTE. Wikipedia, [online, navštíveno 07.01.2014].
URL http://cs.wikipedia.org/wiki/LTE

[32] Introduction to Mobile Development. Xamarin, [online, navštíveno 10.01.2014].
URL http://docs.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/

[33] Store. Xamarin, [online, navštíveno 10.01.2014].
URL https://store.xamarin.com/

# Přílohy

# Seznam příloh

# Příloha A

# Obsah CD

`text` – zdrojové TeX soubory textu práce a z nich vygenerované soubory .pdf

`RoCoManager` – zdrojové kódy a .apk balíček aplikace RoCoManager

`server` – zdrojové kódy serverové aplikace

`plakat` – plakát

`demonstracni_video` – demonstrační video

`uzivatelsky_test` – materiály využité při uživatelském testování aplikace (zadání testu, nápověda a formulář)

`readme.txt` – soubor obsahující další informace o obsahu CD a URL adresu a přístupové údaje pro testování webové části řešení