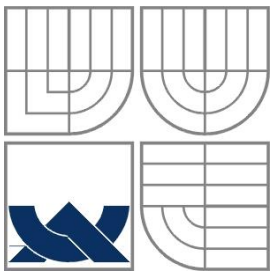
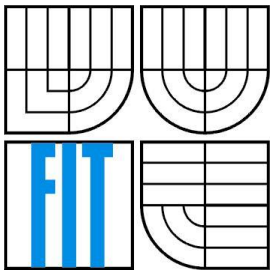


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MULTIPLATFORMNÍ MOBILNÍ APLIKACE

MULTI-PLATFORM MOBILE APPLICATIONS

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE
AUTHOR

BC. JIŘÍ LANG

VEDOUCÍ PRÁCE
SUPERVISOR

ING. RADEK BURGET, Ph.D.

BRNO 2014

Abstrakt

Tato práce popisuje problematiku vývoje aplikací na mobilní zařízení, se zaměřením na vývoj napříč platformami. Seznamuje s nejpoužívanějšími nástroji pro multiplatformní vývoj a jejich principy. Dále se zabývá návrhem a implementací mobilní aplikace pro systémy iOS a Android. Tato aplikace slouží pro organizaci času stráveného na různých projektech a úkolech pomocí přehledných grafů a statistik. Poskytuje také nástroje pro lepší řízení týmových projektů pro malé firmy a podnikatele.

Abstract

This thesis describes the development for mobile devices, with a focus on cross platform development. It introduces the most used tools for multiplatform development and their principles. It also deals with design and implementation of mobile application for iOS and Android operating systems. This application serves for management of time spent on various projects and tasks using graphs and statistics. It also provides tools for better management of team projects for small businesses and entrepreneurs.

Klíčová slova

multiplatformní mobilní aplikace, organizace času, iOS, Android, PhoneGap, Titanium, Angular

Keywords

multiplatform mobile applications, time management, iOS, Android, PhoneGap, Titanium, Angular

Citace

Lang Jiří: Multiplatformní mobilní aplikace, diplomová práce, Brno, FIT VUT v Brně, 2014

Multiplatformní mobilní aplikace

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Radka Burgeta, Ph.D. Další informace mi poskytl Mgr. Tomáš Hnilica. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jiří Lang
27.5.2014

Poděkování

Rád bych poděkoval vedoucímu diplomové práce Ing. Radku Burgetovi, Ph.D. za odborné vedení a připomínky při tvorbě této práce. Dále bych rád poděkoval Mgr. Tomáši Hnilicovi za cenné konzultace.

© Jiří Lang, 2014
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod

Tato práce má za cíl popsat problematiku vývoje multiplatformních mobilních aplikací a dostupných vývojových nástrojů. Na základě zjištěných informací a na základě požadavků zadavatele je dále cílem vytvořit mobilní aplikaci pro systémy iOS a Android. Aplikace má sloužit k podpoře řízení projektů a lidských zdrojů skrze sledování odpracovaného času a ujetých cest. Vytvořená aplikace musí být otestována na požadovaných platformách. Práce je tematicky rozdělena do kapitol, z nichž první kapitolou je tento úvod.

Druhá kapitola seznamuje s principy tvorby nativních a webových aplikací pro mobilní zařízení. Představuje související technologie a nástroje pro vývoj na jednotlivých platformách a srovnává podíl na trhu mezi jednotlivými platformami.

Třetí kapitola popisuje dostupné frameworky pro multiplatformní vývoj a navrhuje nejvhodnější nástroj pro vyvíjenou aplikaci.

Čtvrtá kapitola obsahuje návrh aplikace, je zde zmíněna zejména motivace na vytvoření aplikace, její použití, návrh architektury a uživatelská specifikace.

V páté kapitole jsou vypsány technologie použité pro implementaci aplikace, včetně jejich klíčových vlastností a vývojových principů.

Šestá kapitola popisuje samotnou implementaci aplikace. Zmiňuje zajímavé části implementace, problémy způsobené multiplatformním vývojem a jejich řešení.

V sedmé kapitole je popsána funkcionalita výsledné aplikace spolu se stručným popisem jednotlivých aplikačních obrazovek.

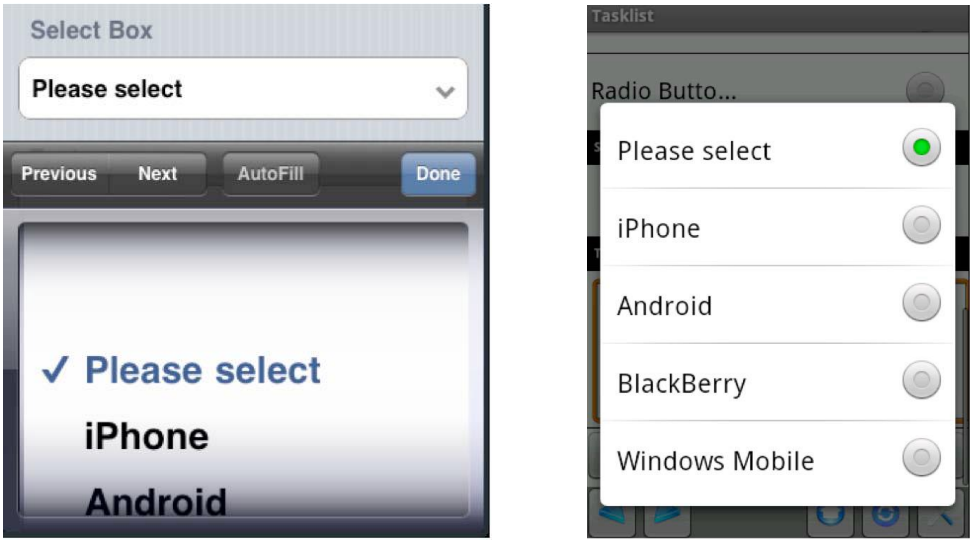
Osmá kapitola obsahuje popis testování aplikace a nasazení na reálná zařízení. Jsou zde vyjmenovány různé přístupy jak docílit překladu aplikace na různé platformy.

Tato diplomová práce navazuje na stejnojmenný semestrální projekt, v jehož rámci byla řešena zejména teoretická část práce. Kapitoly 2, 3, 4 a částečně i 5 tak přímo využívají zjištění a výsledků plynoucích z řešení semestrálního projektu.

2 Nativní a webové mobilní aplikace

Existují dva přístupy k vývoji aplikací na mobilní zařízení. Prvním a zároveň nejčastějším přístupem je vytvoření nativní aplikaci na míru cílovému operačnímu systému. Největší výhodou nativních aplikací je optimalizace pro konkrétní hardware a OS, díky čemuž jsou rychlejší a plynulejší. Vyžaduje to však oddělený vývoj pro každou platformu a v konečném důsledku větší nároky na čas a prostředky. Z toho důvodu se v současné době se stále častěji používá také druhý způsob, a to vývoj webové aplikace, která se spouští v mobilním prohlížeči. Taková aplikace přináší výhodu přenositelnosti mezi platformami, kdy je jediným omezením podporovaná funkcionalita prohlížeče. Vývoj webové aplikace probíhá současně na všechny platformy, což ušetří čas i peníze. Navíc při použití responzivního designu je možné použít stejný kód aplikace pro desktopovou i mobilní verzi. Největší nevýhodou webových aplikací je, že nemohou využívat všech funkcí mobilního zařízení jako je geolokace nebo fotoaparát. Díky spouštění v mobilním prohlížeči jsou také pomalejší a závislé na připojení k internetu.

Každá platforma má svůj specifický look and feel, uživatel tak předpokládá stejný vzhled a chování základních ovládacích prvků. Na obr. 1 je vidět rozdíl mezi ovládacím prvkem rozbalovací nabídka na systémech iOS a Android. Také některé akce, například dlouhý stisk tlačítka, mají na různých platformách různou interpretaci. Specifického vzhledu a chování lze dosáhnout i při vývoji multiplatformní webové aplikace pomocí detekce operačního systému a přizpůsobení stylů a skriptů. Oproti nativnímu vývoji na jedné platformě je však potřeba větší úsilí vývojáře pro dosažení správného vzhledu a chování.



Obr. 1 : Srovnání vzhledu rozbalovací nabídky v systému iOS (vlevo) a Android (vpravo)

Nativní aplikace mají jedinečný a efektivní způsob distribuce. Vývojář vytvořenou aplikaci nahraje na online obchod dané platformy a ten za něj obstará marketing a distribuci koncovým uživatelům. Nativní aplikace mohou být také snadno monetizovány, vývojář nastaví poplatek za stažení aplikace nebo za rozšiřující obsah a peněžní transakce za něj opět řeší obchod. U webových aplikací je situace jiná, jsou sice snadno přístupné odkudkoliv díky internetu, ale uživatel mobilního zařízení není zvyklý vyhledávat aplikace na internetu a spouštět je v prohlížeči skrze URL.

Tyto a další nevýhody vývoje webových aplikaci na mobilní zařízení řeší multiplatformní frameworky, které umožňují webovou aplikaci zabalit do nativní podoby a tu pak vystavit v klasickém online obchodě s nativními aplikacemi. Díky tomu došlo k vyřešení problému s distribucí a monetizací. Další funkcí frameworků je poskytnutí rozhraní pro přístup k hardwaru mobilního zařízení, webová aplikace tak může používat GPS senzor nebo fotoaparát.

V současnosti je možné webové aplikace zkompilovat do podoby nativní aplikace bez nutnosti instalace SDK. Webové služby, jako například Adobe PhoneGap Build, umožňují nahrát adresáře se zdrojovými soubory webové aplikace na server, kde dojde ke zkompilování na zvolenou platformu. Můžeme tak říci, že vývoj webových aplikací se tak stal plnohodnotnou alternativou k vývoji aplikací nativních a v mnoha ohledech je dokonce převyšuje.

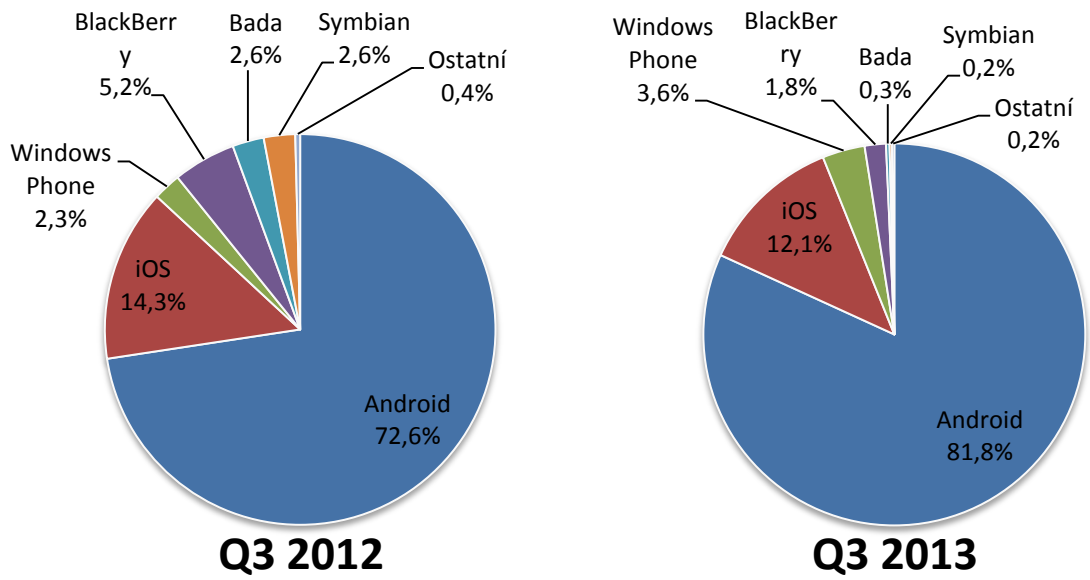
2.1 Mobilní platformy

V současnosti je na trhu k dispozici široká škála mobilních zařízení s různými operačními systémy. Každá platforma je specifická jak programovacím jazykem pro vyvíjené aplikace, tak i vývojovým prostředím, ve kterém lze aplikace vyvíjet a testovat. V tabulce 1 je přehled mobilních platforem spolu s výčtem potřebných nástrojů pro vývoj.

Cílový OS	**OS pro vývoj**	**Vývojové prostředí/Software**	**Programovací jazyk**
iOS	pouze Mac	Xcode	Objective C
Android	Windows/Mac/Linux	Eclipse/Java/Android Development Tool (ADT)	Java
BlackBerry	Windows	Eclipse/JDE, Java	Java
Windows Phone	zejména Windows	Visual Studio	C#, .NET, Silverlight nebo WPF
WebOS	Windows/Mac/Linux	Eclipse/WebOS plugin	HTML/JavaScript/C++

Tab. 1: Přehled požadavků pro vývoj na mobilních platformách [2]

Tato práce nemá za cíl vyčerpávající popis všech platforem, a tak zde budou popsány pouze dvě nejrozšířenější platformy Android a iOS, které však podle výzkumu agentury Gartner, Inc. dohromady pohání 94% celosvětově prodaných smartphonů za třetí čtvrtletí roku 2013. Podrobnější údaje jsou uvedeny v tabulce 1. a v souvisejícím grafu na obr. 2.



Obr. 2: Grafické znázornění podílu OS v celosvětovém prodeji smartphonů v letech 2012 a 2013 [8]

Operační systém	**Prodaných telefonů v Q3 2013**	**Podíl na trhu v Q3 2013 (%)**	**Prodaných telefonů v Q3 2012**	**Podíl na trhu v Q3 2012 (%)**
Android	205 022,7	81,9	124,552.3	72,6
iOS	30 330,0	12,1	24,620.3	14,3
Windows Phone	8 912,3	3,6	3,993.6	2,3
BlackBerry	4 400,7	1,8	8,946.8	5,2
Bada	633,3	0,3	4,454.7	2,6
Symbian	457,5	0,2	4,401.3	2,6
Ostatní	475,2	0,2	683.7	0,4
Celkem	**250 231,7**	**100,0**	**171,652.7**	**100,0**

Tab. 2: Celosvětový prodej smartphonů v tisících podle operačního systému ve třetím čtvrtletí roku 2012 a 2013 [8]

2.1.1 Android

Operační systém Android je open-source projekt vyvíjený skupinou Open Handset Alliance (OHA), jejímž zakladatelem je společnost Google. První mobilní telefon s tímto operačním systémem se objevil v říjnu 2008 a byl jím T-Mobile G1 (označován také jako HTC Dream). Dnes se jedná o nejrozšířenější mobilní operační systém, běžící na mobilních telefonech a tabletech od široké škály výrobců.

Android je postaven na linuxovém jádře verze 2.6. Podporuje 2D a 3D grafiku podle specifikace OpenGL ES 2.0. Obsahuje vestavěný webový prohlížeč, který běží na jádře WebKit a obsahuje V8 Chrome JavaScript runtime pro rychlejší vykonávání skriptů. Systém podporuje technologii multitouch, která však nemusí být dostupná na všech mobilních zařízeních.

Běžící aplikace v systému jsou strukturovány jako tzv. aktivity. Aktivita má obvykle jeden účel, jako třeba focení fotografií. Komplexní aplikace mohou implementovat více aktivit. Android podporuje multitasking, v jeden okamžik je tak možné mít spuštěno více aplikací.

Nativní aplikace pro Android jsou typicky vyvíjeny v jazyce Java, jelikož však systém neobsahuje Java Virtual Machine, jsou třídy rekompilovány do bytekódu Dalvik a spuštěny na Dalvik virtual machine. Dalvik je speciálně navržen pro mobilní zařízení za účelem snížení spotřeby baterie a menších nároků na paměť a CPU. Díky Android NDK (Native Development Kit) je možné také psát aplikace v C nebo C++, čímž lze v některých případech dosáhnout rychlejšího běhu. Pro vývoj je nejčastěji používáno prostředí Eclipse s nainstalovanými komponenty Android SDK a Android Development Tools (ADT) plugin, které umožňují testování aplikací v emulátoru.

Pro distribuci aplikací na platformě Android slouží online obchod Google Play, který k červenci 2013 obsahoval 1 milión aplikací ke stažení. Registrace pro vývojáře v současnosti stojí 25 $ a aplikace musí být digitálně podepsána soukromým klíčem, ten však může vygenerovat sám vývojář [1].

2.1.2 iOS

Jedná se o v současnosti druhý nejrozšířenější operační systém, běžící pouze na výrobcích firmy Apple. Poprvé byl představen v červnu 2007 na mobilním telefonu iPhone. Dnes kromě iPhone, pohání také tablet iPad, hudební přehrávač iPod touch nebo chytrou televizi Apple TV.

Pro vývoj aplikací na iOS je nutné nainstalovat iPhone SDK, obsahující mimo jiné vývojové prostředí Xcode a iPhone simulátor. iPhone SDK je však možné nainstalovat pouze na operačním systému OS X. Aplikace na iPhone jsou typicky vyvíjeny v jazyce Objective-C s využitím Model-View-Controller architektury. Model je definován jako podtřída NSObject, Controller dědí z UIViewController nebo UITableViewController. View soubory pro grafické rozhraní je možno vytvořit pomocí WYSIWYG nástroje Interface Builder nebo je možné je navrhnout programově.

Dalším návrhovým vzorem zhusta používaným ve vývoji na iPhone je Delegace, ta umožňuje komplexním objektům delegovat část své funkcionality na pomocné objekty tzv. helpery. Při pohledu zvenčí to vypadá, že se volá metoda původního objektu, ale ve skutečnosti si objekt zavolá helper pro vykonání požadavku.

Distribuce aplikací na iOS probíhá přes online obchod App Store, který v říjnu roku 2013 (tedy jen o 3 měsíce později než Google Play) dosáhl 1 miliónu aplikací ke stažení. Umístění aplikace do obchodu je oproti Google Play o něco složitější. Z důvodu přísné politiky firmy Apple musí vývojář nejprve požádat o schválení aplikace komisí App Review Board a teprve poté může aplikaci nahrát do obchodu [1].

3 Tvorba multiplatformních aplikací

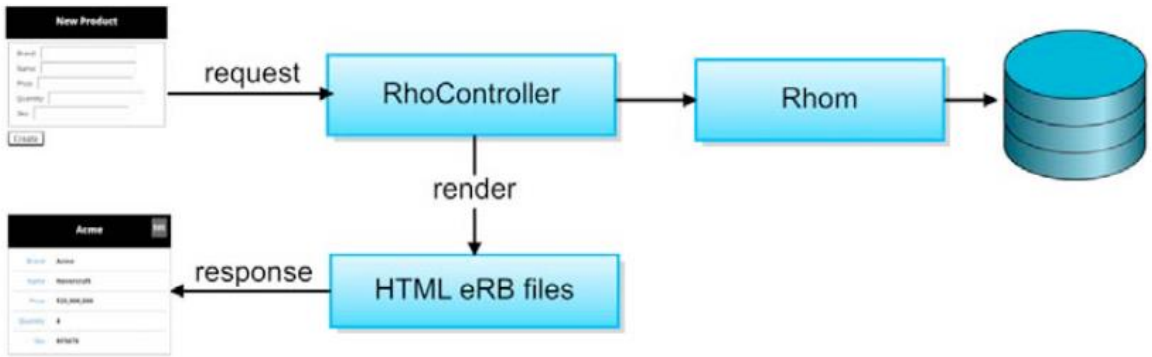
Existuje velké množství frameworků pro vývoj aplikací napříč technologiemi. V této kapitole zmíníme ty nejrozšířenější, z nichž každý používá lehce odlišný způsob pro zajištění přenositelnosti kódu.

3.1 Rhodes

Rhodes je součástí projektu RhoMobile Suite společnosti Motorola. Jedná se o open source framework pro vývoj mobilních aplikací napříč platformami, s využitím HTML, CSS, JavaScriptu a programovacího jazyka Ruby. Jeho primárním určením je vývoj podnikových aplikací. Rhodes zajišťuje objektově relační mapování pomocí nástroje Rhom a synchronizaci lokálního úložiště se vzdáleným pomocí RhoSync. Aplikace jsou zkompilovány do bajtkódu Ruby 1.9 a Rhodes přiloží speciální spouštěč pro spuštění bajtkódu na cílovém zařízení.

Vývoj aplikací v Rhodes vychází z návrhového vzoru Model-View-Controller (MVC). Při vývoji aplikací je použito HTML a CSS pro definování vzhledu uživatelského rozhraní a JavaScript pro interakci s uživatelem. Pomocí Ruby je možno vytvářet také ERB šablony, kde je do značkovacího jazyka HMTL vložen programovací jazyk Ruby (obdobně jako v PHP nebo JSP šablonách). Za běhu aplikace se nejprve vyhodnotí kód v Ruby, poté dojde k vykreslení HTML stránky komponentou vestavěného webového prohlížeče a nakonec se vykoná JavaScript. Aplikační logika je implementována taktéž pomocí Ruby. Controller implementuje akce, které se starají o obsluhu HTTP požadavků, typická akce Controlleru je získání dat z modelu (implementovaného pomocí Rhom) a vykreslení odpovídajícího View (ERB šablona).

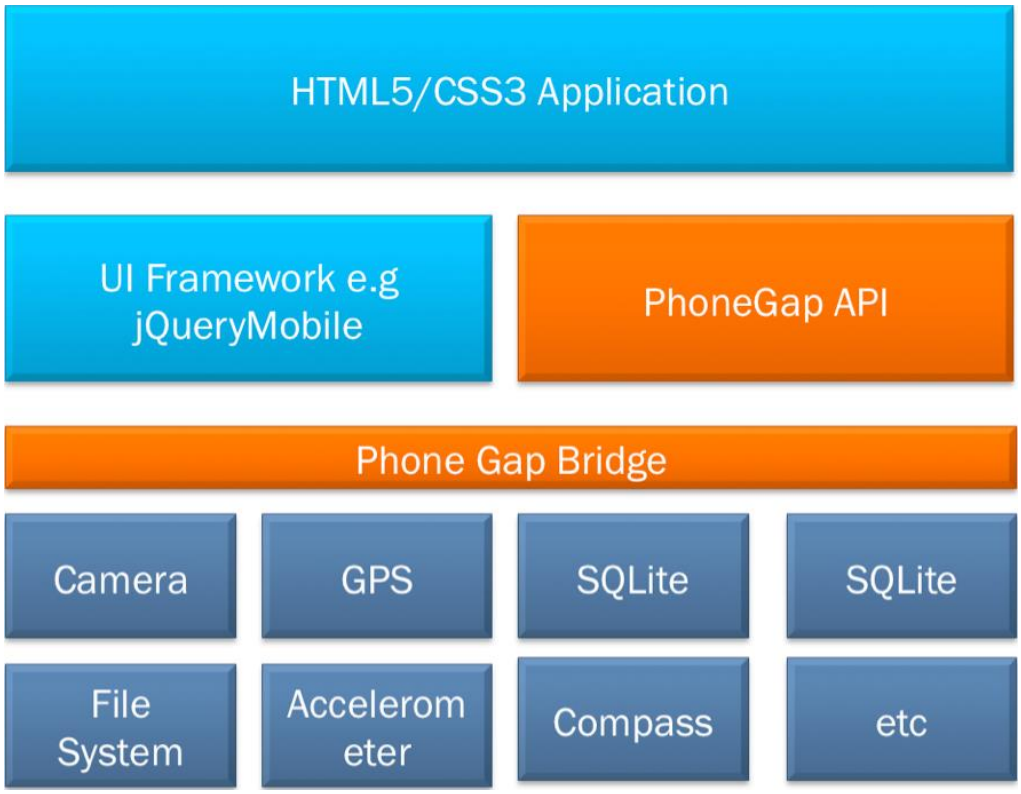
Na obr. 3 je zobrazen princip fungování Rhodes aplikace, kdy uživatel nejprve vyplní formulář s informacemi o novém produktu. Odesláním formuláře se odešle požadavek na vestavěný webový server, který se stará o obsluhu těchto požadavků a akcí Controlleru. V tomto případě se zavolá create metoda Controlleru, v níž jsou atributy nového produktu předány do Modelu produktu, kde dojde k uložení nového produktu do lokální databáze. Poté Controller vykreslí View s výpisem nového produktu uživateli. Celý tento cyklus webového požadavku a odpovědi se odehrává lokálně na mobilním zařízení [1].



Obr. 3: Architektura Rhodes aplikace [1]

3.2 PhoneGap

PhoneGap je HTML5 framework určený pro vývoj nativních aplikací skrze webové technologie. Díky tomuto nástroji není nutné ovládat všechny programovací jazyky na rozličných platformách, ale stačí znalost HTML, CSS a JavaScriptu. S využitím PhoneGap jsou vytvářeny hybridní aplikace – nejedná se čistě o HTML/JavaScript ani o čistě nativní. Části aplikace související s uživatelským rozhraním, aplikační logikou a komunikací se serverem jsou psány v HTML/JavaScriptu. Části komunikující s mobilním zařízením jsou v nativním jazyce platformy. Jak je vidět na obr. 4, PhoneGap poskytuje propojení mezi webovou a nativní částí pomocí svého API. Když vývojář potřebuje přistoupit k hardwaru mobilního zařízení, může to udělat přímo v JavaScriptu voláním metod z PhoneGap API. Díky nim pak může ovládat takové prvky zařízení, jako jsou fotoaparát, GPS nebo akcelerometr bez nutnosti psát nativní kód. Aplikace vytvořená ve PhoneGap se dělí na dvě hlavní části: část JavaScript Business Logic, která se stará o uživatelské rozhraní a jeho funkcionalitu a část JavaScript, která má přístup k mobilnímu zařízení a ovládá jej [2].



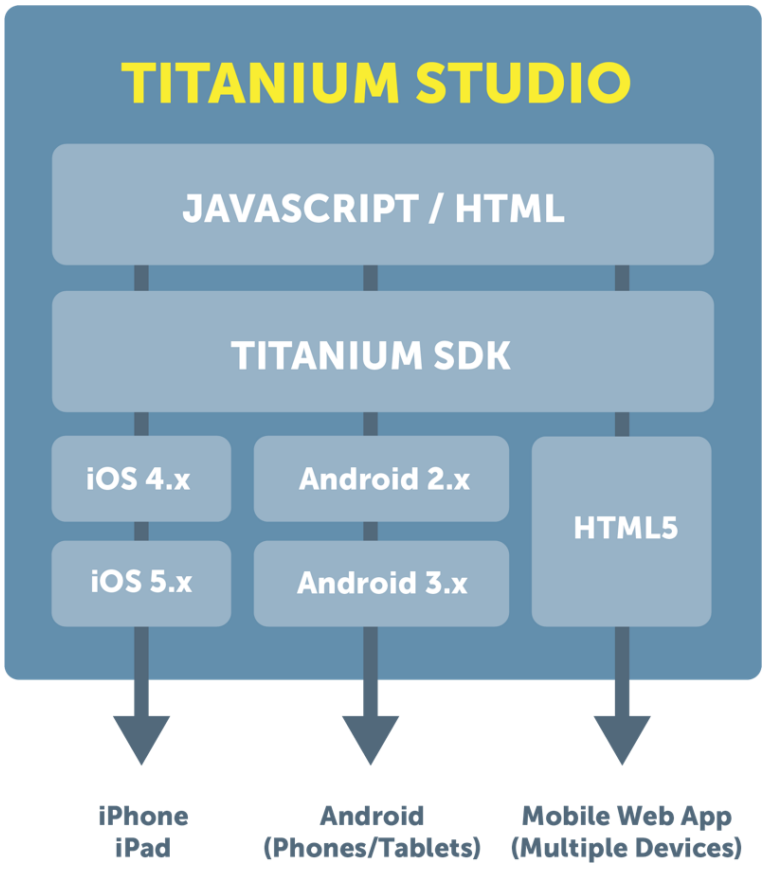
Obr. 4: Architektura aplikace ve PhoneGap [2]

PhoneGap aplikace je v podstatě webová aplikace zabalená do nativního obalu, aby ji bylo možné umístit do obchodu s aplikacemi a snadno nainstalovat. Takové aplikace jsou snadno přenositelné na širokou škálu mobilních platforem jako: iOS, Android, Windows Phone, Blackberry nebo webOS. PhoneGap obsahuje vestavěný webový prohlížeč, ve kterém probíhá spouštění aplikace. Díky tomu není závislý na konkrétních ovládacích prvcích platformy a nehrozí tak nekompatibilita s některou verzí systému. Na druhou stranu z důvodu spouštění v prohlížeči není tento framework vhodný pro vývoj aplikací s vysokými nároky na hardware a na rychlost.

3.3 Appcelerator Titanium

Titanium je produkt společnosti Appcelerator, který umožňuje vytvářet mobilní aplikace v JavaScriptu a poté je zkompilovat do nativní podoby pro iOS, Android a BlackBerry. Titanium tedy stejně jako PhoneGap používá JavaScript, ale na rozdíl od něj nevytváří šablony uživatelského rozhraní pomocí HTML a CSS. Veškerý kód je psán pouze s použitím JavaScriptu a Titanium SDK. Pokud je potřeba změnit například vzhled tlačítka, nelze použít kaskádové styly, ale místo toho se změní parametry tlačítka. Titanium SDK používá nativní ovládací prvky, což vytváří přirozenější vzhled a chování pro každou platformu. Titanium při kompilování aplikace zpracuje JavaScript a vygeneruje z něj nativní projekt pro cílovou platformu. Při vývoji na iOS je vygenerován skutečný Xcode projekt, který je poté zkompilován Apple kompilátorem a výsledkem je nativní .IPA soubor, který může být nahrán přímo do mobilního zařízení, nebo do App Storu. Stejný postup platí i pro Android, kdy je vygenerována nativní Java aplikace a zkompilována Android kompilátorem. Výsledná aplikace je tak vždy 100% nativní a používá 100% nativní ovládací prvky. Díky tomu je Titanium vhodné pro vývoj aplikací, kde je důraz kladen na rychlost a optimální využití hardwaru [3].

Na obr. 5 je zobrazena architektura aplikace ve frameworku Titanium. Mezi cílovým operačním systémem a aplikací v JavaScriptu je vrstva Titanium SDK s poskytovaným API. Vývojář se pak v JavaScriptu odkazuje na toto API za účelem provedení akcí jako vykreslení tlačítek, otevření okna, zobrazení fotoaparátu apod. Část SDK zvaná Kroll potom tyto API volání přeloží do nativní podoby. Kroll také předává parametry a události směrem z JavaScriptu do nativní části a naopak [9].



Obr. 5: Architektura aplikace v Titanium [9]

3.4 Tabris

Tabris je framework umožňující vývoj multiplatformních mobilních aplikací v jazyce Java. Tabris aplikace je v podstatě webová aplikace v Javě, která běží na serveru, jednotliví klienti (mobilní zařízení) se pak k tomuto serveru připojují a komunikují prostřednictvím JSON požadavků. Server zasílá informace o uživatelském rozhraní a to je pak na klientovi vykresleno pomocí nativních prvků. Tento framework je v současnosti zdarma pouze pro open source a vzdělávací projekty, jinak je placený. Taktéž díky své architektuře vyžaduje neustálé připojení k internetu, což je pro navrhovanou aplikaci nevyhovující [10].

3.5 Zhodnocení frameworků

Framework Rhodes má výhodu v dodávaném nástroji RhoSync pro synchronizaci dat se vzdáleným úložištěm, nevýhodou však je implementační jazyk Ruby, což by přineslo nutnost přepsání existující serverové částí aplikace psané v jazyce PHP.

Appcelerator Titanium je z hlediska implementačního jazyka vhodnější, díky kompilování uživatelského rozhraní do nativní podoby, však vyžaduje přizpůsobení kódu jednotlivým platformám a není tedy tolik univerzální. Titanium poskytuje rychlejší běh aplikace, protože je veškerý kód zkompilován do nativní podoby, naproti tomu PhoneGap aplikace se spouští skrze vestavěný webový prohlížeč, takže jsou z principu pomalejší. V případě navrhované aplikace nejsou nároky na hardware příliš vysoké a otázka rychlosti není klíčová.

PhoneGap skrze spouštění aplikace ve vestavěném prohlížeči přináší největší přenositelnost mezi platformami a jejich různými verzemi. Díky responsivním HTML šablonám se přizpůsobí rozměrům obrazovky na tabletu nebo mobilním telefonu. Nevýhodou je stejný vzhled na všech platformách, což ale lze upravit pomocí kaskádových stylů oddělených zvlášť pro každou platformu. Z výše uvedených důvodů jsem tedy pro vyvíjenou aplikaci zvolil framework PhoneGap.

4 Návrh aplikace

Mobilní aplikace bude vycházet z webové služby Estimo.cz pro sledování projektů, zdrojů a pracovníků. Z důvodu rozdílné použitelnosti aplikace na mobilním zařízení a požadavku na běh i bez internetového připojení se nebude jednat o pouhé zabalení existující webové aplikace do nativní podoby a spuštění na mobilu. Je třeba vyvinout samostatnou mobilní aplikaci, která však bude s webovou službou komunikovat pro zajištění synchronizace dat.

Při návrhu mobilní aplikace k existující webové službě je třeba mít na paměti rozdíly mezi těmito dvěma platformami. Mobilní zařízení má výrazně menší displej než je obrazovka počítače, proto tomu musí být grafické uživatelské rozhraní přizpůsobeno. Oproti webové verzi by měla mobilní verze obsahovat obrazovky s menším množstvím obsahu. Uživatelé webu jsou zvyklí mít vše přehledně viditelné na jednom místě, nejlépe s co nejmenší úrovní hierarchie stránek, naproti tomu na mobilním zařízení uživatel předpokládá nutnost hlubšího zanoření v hierarchii obrazovek pro nalezení požadované akce. Ruku v ruce s tímto principem je však nutné definovat nejčastější a klíčové funkce aplikace a k nim zajistit co nejsnazší a nejrychlejší přístup. Mobilní aplikace také ze své podstaty nenabízí veškerou funkcionalitu svého webového předobrazu.

Mobilní aplikace, ale přináší hned několik významných výhod. První je možnost zjištění pozice uživatele pomocí GPS souřadnic. Toho jde s úspěchem využít například pro počítadlo kilometrů na služebních cestách. Další výhodou je možnost zasílat uživateli notifikace. Tímto způsobem může aplikace proaktivně upozornit uživatele na novinky, aniž by ji musel několikrát spouštět a kontrolovat vše ručně.

Na mobilním zařízení obvykle nemáme k dispozici hardwarovou klávesnici (pokud pomineme telefony BlackBerry), ale pouze softwarovou. Z toho důvodu nemůže aplikace po uživateli chtít zadávání dlouhého textu, který je také obtížné formátovat. Absence hardwarové klávesnice je na druhou stranu kompenzována vestavěným mikrofonem, což v kombinaci s nástrojem pro rozpoznáváním řeči, umožní uživateli hlasové zadávání krátkých poznámek.

4.1 Motivace

Motivací pro vytvoření aplikace je usnadnění řízení projektů a pracovníků malým až středně velkým firmám. Aplikace bude spolupracovat s existující webovou službou, ale díky nasazení na mobilní zařízení umožní efektivněji sledovat čas pracovníků v terénu. Aplikace stejně jako webová služba nesměřuje na firmy podnikající v oblasti vývoje softwaru, neboť v této oblasti je podobných řešení dostatek a poskytují komplexní řešení pro projektové řízení včetně sdílení kódu, report chyb apod. Na druhou stranu řada firem a uživatelů, zejména mimo obor informačních technologií, tyto aplikace nevyužívá, protože jim přijdou moc složité a řadu funkcí vůbec nevyužijí. Právě na tyto firmy cílí vyvíjená aplikace, která chce nabídnout uživateli jednoduché rozhraní s důrazem na snadnou ovladatelnost a pouze s funkcemi, které opravdu potřebuje. Typickým uživatelem aplikace může být účetní, pracující pro několik firem, nebo obchodní zástupce či technik objíždějící klienty a domácnosti po celé republice.

4.2 Požadavky na aplikaci

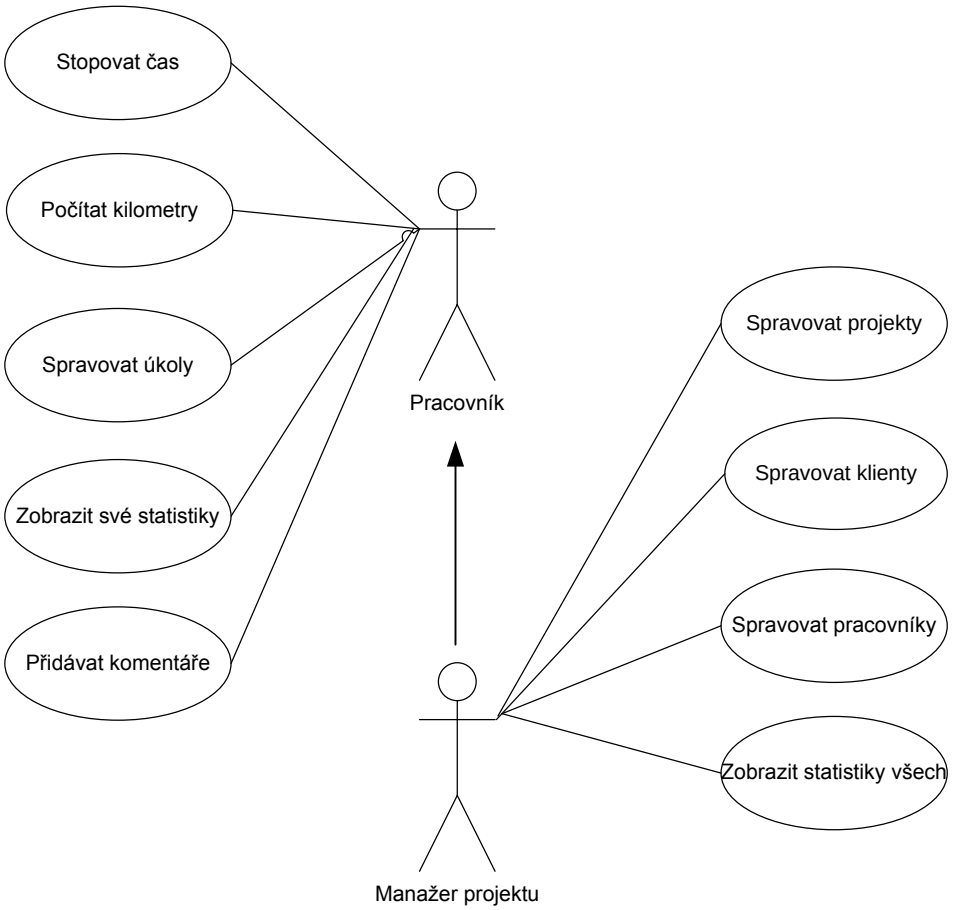
- Aplikace umožní vytvářet hierarchickou strukturu projektů a úkolů a k nim přiřazovat lidské zdroje.
- Manažer projektu bude moci sledovat průběh prací na jednotlivých projektech.

- Aplikace bude poskytovat přehledné statistiky využití času a nákladů.
- Jednotlivý pracovníci budou moci stopovat svůj čas strávený na projektech.
- Aplikace bude umožňovat vyúčtování cest pracovníka prostřednictvím GPS souřadnic.
- K jednotlivým projektům a úkolům bude možné nahrát poznámky prostřednictvím hlasu.
- Aplikace bude pracovat i v offline režimu, po připojení k internetu umožní synchronizaci s webovou službou.
- Aplikace musí být multijazyčná a podporovat minimálně češtinu a angličtinu.

4.3 Použití aplikace

V aplikaci vystupují dvě role uživatelů: pracovník a manažer. Pracovník může stopovat čas strávený na jednotlivých úkolech a počítat ujeté kilometry. Dále má přístup k výpisu svých přiřazených úkolů a zobrazení přehledných statistik svého času stráveného na jednotlivých úkolech. Pod projekty, ke kterým je přiřazen, může vytvářet nové úkoly. K přiřazeným úkolům může přidávat také komentáře se souborovými přílohami.

Uživatel s rolí manažer má navíc možnost vytvářet nové klienty a projekty a přiřazovat k nim pracovníky. Má také možnost zobrazit statistiky rozšířené o další pohledy, jako zobrazení pracovního vytížení členů týmu napříč projekty nebo kalkulaci aktuální ceny projektu. Na obr. 7 jsou případy použití zakresleny do diagramu.



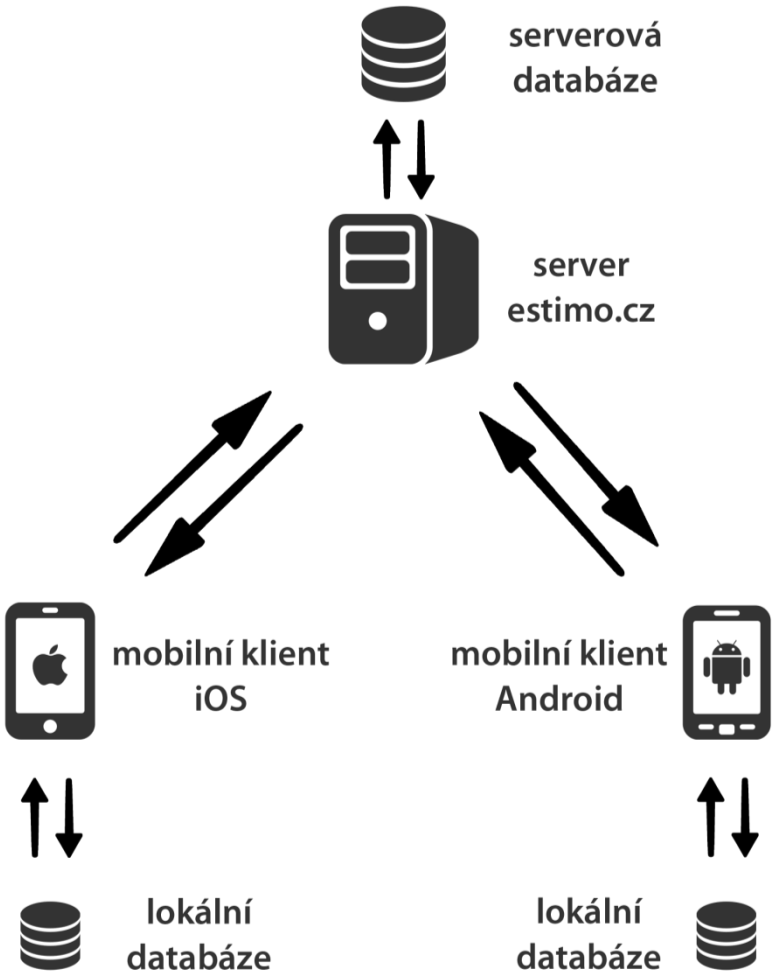
Obr. 6: Diagram případů použití navrhované aplikace.

4.4 Architektura

Při návrhu bylo třeba specifikovat jak architekturu celého systému, ve kterém spolu komunikují webová služba a jednotliví klienti, tak architekturu klienta a jeho databáze.

4.4.1 Server a více klientů

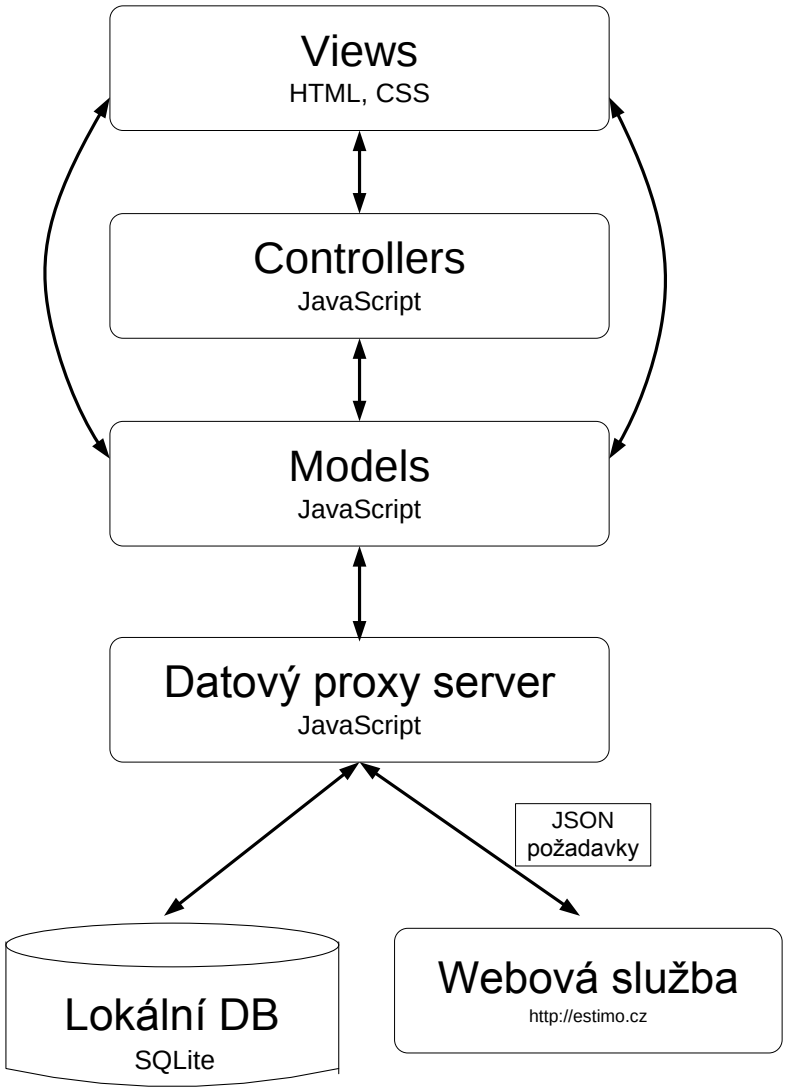
Architektura celého systému odpovídá vzoru klient-server, kde se předpokládá větší množství klientů na různých platformách. Toto s sebou přináší dva problémy, které bude nutné vyřešit. První jsou rozdílné platformy na klientech, což bude vyřešeno použitím multiplatformního frameworku PhoneGap, který zajistí stejné chování klientské aplikace na různých platformách. Protože klientské aplikace musí být schopné omezeně pracovat i v případě výpadku internetového připojení, jsou za tímto účelem vybaveny lokální databází. Z tohoto faktu vyvstává druhý problém, a to zajištění synchronizace databází mezi klientem a serverem a mezi klienty navzájem. Tato synchronizace je podrobněji popsána v kapitole 4.4.3. Níže uvedený obrázek demonstruje architekturu aplikace a její napojení na existující webovou službu.



Obr. 7: Architektura aplikace
Zdroj: vlastní zpracování s využitím ikon ze zdroje: http://www.visualpharm.com/

4.4.2 Architektura klienta

Architektura klientské (mobilní) aplikace bude odpovídat návrhovému vzoru Model-View-Controller (MVC) s využitím JavaScriptového frameworku Angular.js. Přenositelnost aplikace zajistí framework PhoneGap, který spustí aplikaci ve vestavěném prohlížeči. Soubory View sloužící pro zobrazování dat uživateli a načítání uživatelských vstupů budou implementovány pomocí HTML a CSS. Angular.js zajistí pomocí JavaScriptu propojení webových šablon s metodami modelu a controlleru. Mezi modelem a databází bude vrstva zajišťující obsluhu dotazů a synchronizaci dat.



Obr. 8: Architektura navrhované klientské aplikace

4.4.3 Synchronizace dat

Aplikace bude fungovat i v offline režimu, a proto je nutné zajistit synchronizaci dat. Aplikace bude uchovávat data v lokální databázi a po obnovení připojení k internetu je odešle webové službě, kde dojde ke sloučení s existujícími daty od všech ostatních klientů. Komunikace mezi mobilní aplikací a webovou službou bude probíhat přes rozhraní REST zasíláním požadavků ve formátu JSON.

Pro zajištění synchronizace bude využita open-source knihovna WebSqlSync. Knihovna přidá na straně klienta do databáze dvě tabulky: tabulku `new_elem`, která uchovává všechny nové nebo

změněné záznamy a tabulku sync_info, která ukládá datum poslední synchronizace. Dále knihovna vytvoří SQLite spouštěče, které po provedení INSERT nebo UPDATE automaticky uloží nový nebo modifikovaný záznam do tabulky new_elem. Na straně serveru je třeba přidat ke všem tabulkám určeným k synchronizaci atribut last_sync_date, který udává datum poslední synchronizace. Synchronizaci iniciuje vždy klient, pokud například detekuje nové záznamy v tabulce new_elem, odešle na server JSON požadavek obsahující nové záznamy, identifikaci klienta a datum poslední synchronizace. Jakmile server obdrží požadavek, uloží nové záznamy do databáze a zkontroluje, zda nejsou k dispozici aktuální data pro klienta porovnáním přijatého a lokálního last_sync_date. Jestliže je datum na serveru novější, server odešle nové záznamy klientovi.

Každá tabulka v databázi obsahuje časová razítka, která určují validitu záznamu. Při dotazu nad daty v aplikaci datový proxy server načte nejprve data z lokální databáze a ověří jejich aktuálnost pomocí časových razítek. Pokud jsou data zastaralá a je k dispozici připojení k internetu, proxy přepošle dotaz na webovou službu, odkud stáhne data aktuální a synchronizuje je s lokální databází. Lokální tabulky musí obsahovat také atribut server_id, který určuje id odpovídajícího záznamu na serveru. Server v odpovědi uvádí syncDate, čímž potvrdí nové záznamy a ty se už v dalším požadavku od klienta neposílají.

4.4.4 Databáze

Struktura databáze je zobrazena na obr. 7. Základní logickou jednotkou aplikace je tracker, reprezentovaný tabulkou trackers. Tracker může být tří typů, z nichž každý typ má specifické vlastnosti. Client_tracker představuje klienta nebo firmu, obsahuje proto kontaktní a fakturační údaje. Dalším typem je project_tracker, který reprezentuje projekt a eviduje informace o uzávěrce a rozpočtu projektu. Posledním typem je tracker pro úkol (task_tracker) s informacemi o uzávěrce, očekávané pracnosti v hodinách a o tom, kolik procent úkolu je již hotovo. Trackery lze uspořádat do stromové hierarchie, kdy je na první úrovni klient (typicky firma, pro niž je práce vykonávána), pod klienta jsou řazeny jednotlivé projekty, které mohou být dále děleny na jednotlivé úkoly.

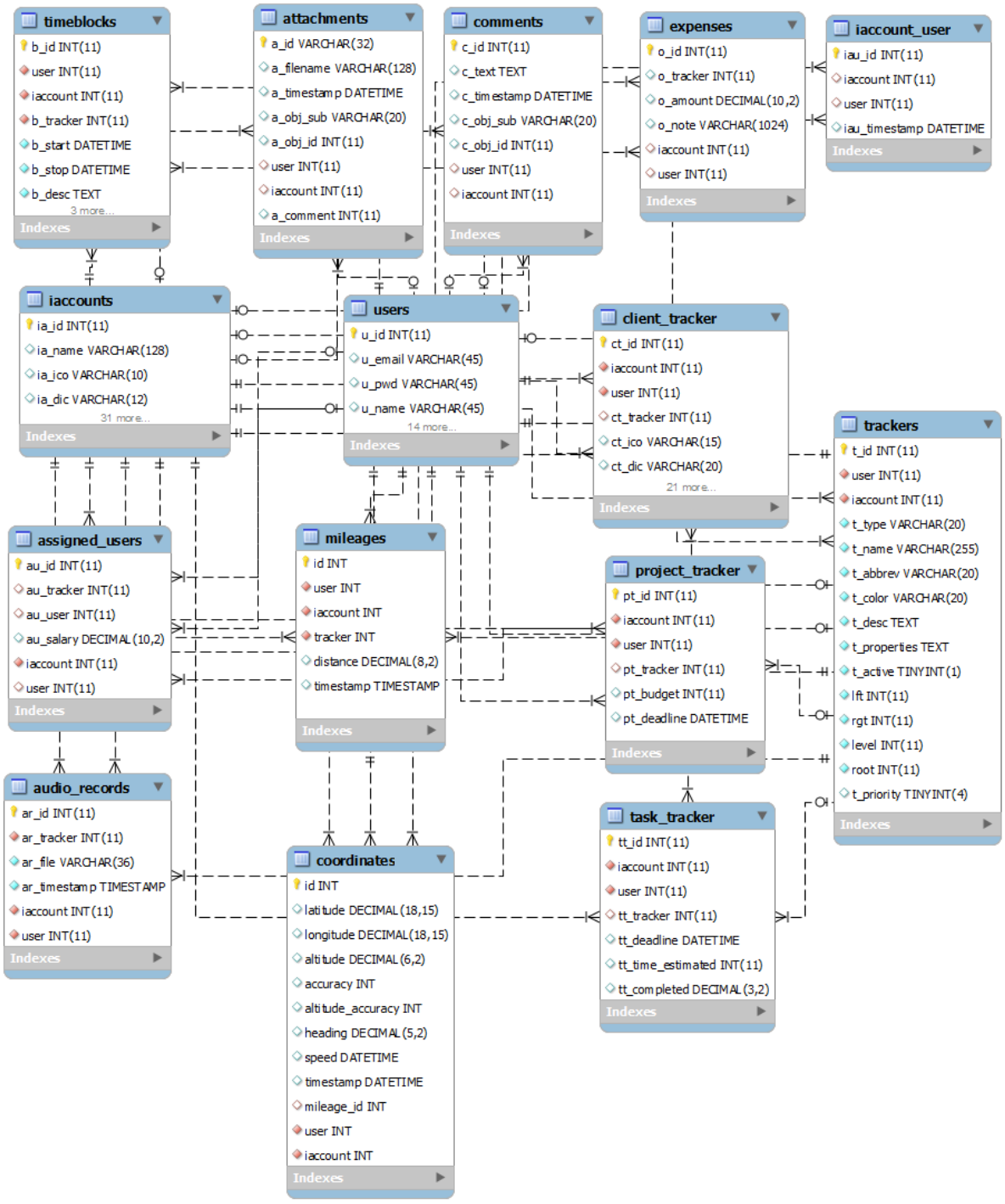
Ke každému projektu nebo úkolu lze přiřadit tým pracovníků zodpovědných za jeho dokončení, tato informace je uložena v tabulce assigned_users, která propojuje tabulky trackers a users. Protože mzda za práci na různých projektech (úkolech) se může lišit, je třeba uchovávat také informaci o hodinové sazbě přiřazeného pracovníka na konkrétním projektu (úkolu). K projektům lze také přiřazovat výdajové položky uchovávané v tabulce expenses.

Dále je možno stopovat čas strávený prací na jednotlivých trackerech. Odpracovaný čas je ukládán v podobě záznamů do tabulky timeblocks, kde nejdůležitější údaje jsou začátek a konec práce a tracker, na kterém práce probíhala.

Tabulka iaccounts představuje jednotlivé účty registrované v aplikaci, přičemž jeden účet může sdružovat více uživatelů, z nichž každý má vlastní přihlašovací údaje. Jednotliví uživatelé jsou uloženi v tabulce users. Účet je v podstatě jednou licencí na používání aplikace. Při zakoupení licence na aplikaci (ve formě měsíčního předplatného) se vytvoří nový účet s uživatelem správce, ten potom může přidávat další uživatele pod tento účet. Ostatní tabulky v databázi pak mají pole iaccount a user, která určují ke kterému uživateli a účtu záznam patří. Díky tomu lze pomocí předdefinovaných globálních databázových dotazů dosáhnout filtrování záznamů pro přihlášeného uživatele.

Uživatel může k jednotlivým trackerům připisovat také komentáře, které jsou uloženy v tabulce comments, a ke komentářům připojovat soubory, uložené v tabulce attachments. K trackeru je dále možné nahrát popis přes mikrofon a nástroj pro rozpoznání řeči, který převede řeč

na psaný text. Zvukové záznamy jsou uchovávány na serveru a informace o nich jsou v tabulce `audio_records`.



Obr. 9: Struktura databáze navrhované aplikace

4.4.5 Počítadlo kilometrů

Pro výpočet kilometráže pomocí GPS souřadnic jsou v databázi tabulky `coordinates` a `mileages`. Tabulka `coordinates` obsahuje tyto atributy:

- **id** – jednoznačný identifikátor záznamu

- **latitude** - zeměpisná šířka ve stupních.
- **longitude** - zeměpisná délka ve stupních.
- **altitude** - nadmořská výška
- **accuracy** – přesnost zeměpisné šířky a délky v metrech
- **altitudeAccuracy** – přesnost nadmořské výšky v metrech
- **heading** – směr cesty, specifikovaný ve stupních vzhledem k severu po směru hodinových ručiček
- **speed** – aktuální rychlost zařízení v m/s
- **timestamp** – časové razítko kdy byly koordináty zaznamenány
- **mileage_id** – odkaz na související záznam o vzdálenosti

Vypočítané vzdálenosti mezi projetými body se budou ukládat do tabulky `mileages`, která bude obsahovat tyto atributy:

- **id** - jednoznačný identifikátor záznamu
- **user** – uživatel, který cestu zaznamenal
- **account** – účet, pod který uživatel patří
- **tracker** – ke kterému trackeru se cesta vztahuje
- **distance** – ujeté kilometry
- **timestamp** – čas zaznamenání cesty

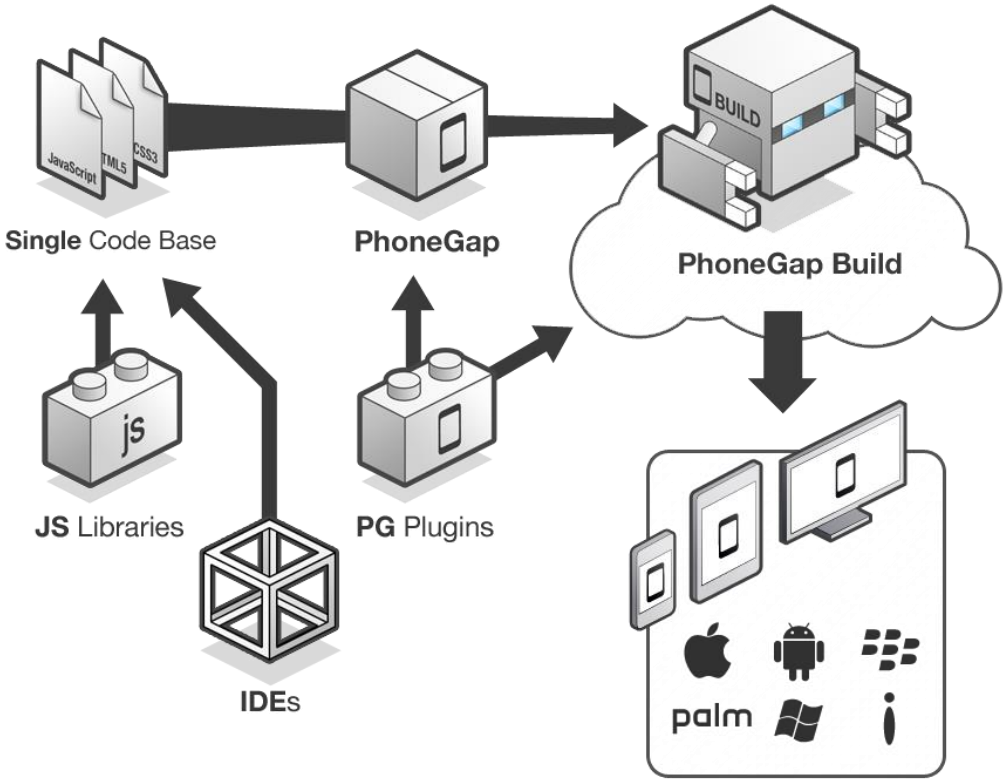
Nové souřadnice se budou získávat pomocí metody `watchPosition` z PhoneGap API, která běží asynchronně a kontroluje změny v aktuální pozici zařízení. Jakmile metoda detekuje změnu, vrací informace o aktuální poloze ve formátu objektu typu `Position`. Pro vypočítání ujeté vzdálenosti a zobrazení trasy na mapě bude použito Google Maps API.

5 Použité technologie

Hlavním požadavkem na aplikaci je její přenositelnost mezi platformami. V kapitole 3 byly uvedeny některé dostupné nástroje pro multiplatformní vývoj a z nich byl vybrán framework PhoneGap. Zdrojové kódy aplikace jsou psány v jazyce JavaScript, a proto byl vybrán vhodný framework pracující s tímto jazykem, konkrétně Angular.js.

5.1 PhoneGap

Přenositelnost kódu bude zajištěna použitím frameworku PhoneGap. Aplikace bude obsahovat soubory v jazyce JavaScript, šablony HTML a definice stylů v souborech CSS. Framework umožní kompilaci jednoho zdrojového kódu na různé platformy, odpadá tak nutnost vytvářet zvláštní projekt na každou platformu. Níže uvedený obrázek demonstruje workflow vývoje aplikace s frameworkem PhoneGap.



Obr. 10: Princip vývoje s frameworkem PhoneGap [27]

Framework rovněž zajišťuje přístup k hardwaru na mobilní zařízení pomocí vestavěných pluginů. V našem případě se bude jednat o plugin poskytující přístup k GPS přijímači pro získání souřadnic o aktuální poloze uživatele a plugin pro správu dat uložených v lokální SQLite databázi zařízení.

5.2 Angular.js

Mobilní aplikace bude vytvářena v jazyce JavaScript, aby bylo dosaženo lepší struktury a modularity kódu, zvolil jsem pro implementaci framework Angular.js. Jedná se o MVC framework pro JavaScript vyvinutý firmou Google pro vývoj lépe strukturovaných, modifikovatelných a testovatelných webových aplikací. Krom definování logické kostry aplikace přináší framework také řadu vestavěných modulů pro usnadnění vývoje. Následuje výčet základních rysů tohoto frameworku.

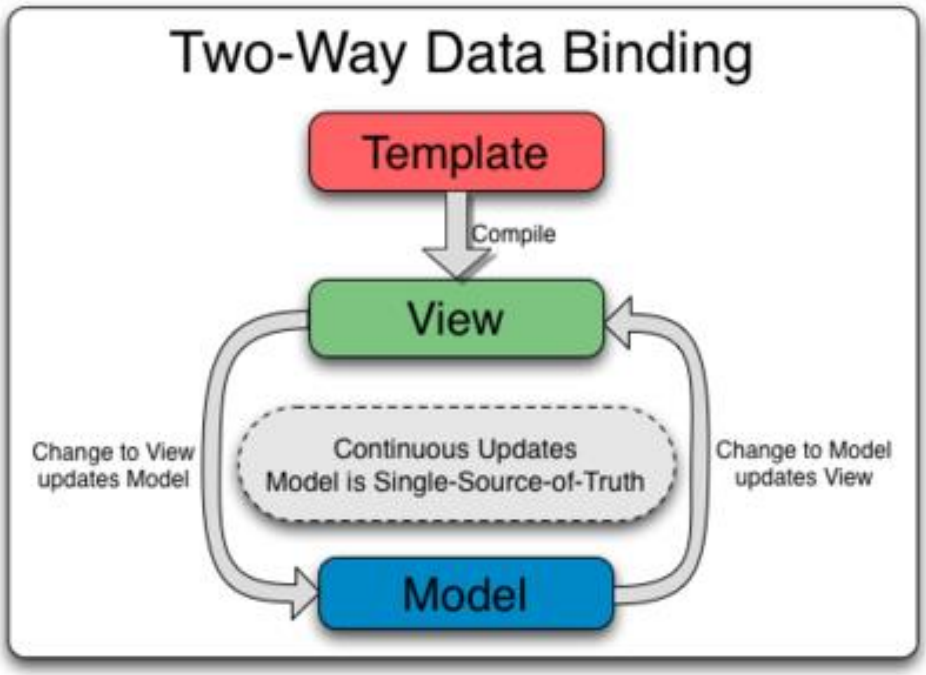
5.2.1 Two-Way Data Binding

Data binding je princip propojení polí ve view s atributy modelu a jejich synchronizace. Díky tomu dochází k omezení množství kódu, jak v modelech, tak ve views. Synchronizace je obousměrná, když uživatel napíše text do textového pole ve view, tento text se odešle do modelu, a poté se aktuální hodnota přepíše do ostatních částí view. Na obr.8 je vidět praktický příklad použití data binding v Angular.js.

```
Enter name: <input type="text" ng-model="name"><br>
Hello <span ng-bind="name"></span>!
```

Obr. 11: Příklad data binding v Angular.js [6]

Přidáním atributu `ng-model` definujeme, se kterým modelem chceme textové pole synchronizovat. Po zadání textu uživatelem se hodnota pole automaticky odešle do modelu a pomocí atributu `ng-bind` se načte aktuální hodnota z modelu a vypíše do příslušného elementu. Princip two-way data binding je zobrazen na obr. 11.



Obr. 12: Dvoucestná synchronizace dat (Two-way data binding) [28]

5.2.2 Dependency injection

Jedná se o návrhový vzor řešící závislosti mezi jednotlivými komponentami programu. Je založen na principu minimální znalosti (nebo taky Law of Demeter), který říká, že objekt by měl vědět co nejméně o struktuře nebo vlastnostech ostatních objektů. Angular.js obsahuje systém pro řešení dependency injection napříč celou aplikací, kdy pro každou komponentu přesně specifikuje, které jiné komponenty jsou uvnitř používané. Objekt samotný nevytváří nové závislosti, ale zažádá si o ně prostřednictvím parametrů ve svém konstruktoru. Za běhu pak pracuje pouze s těmito předanými parametry, což vede k přehlednějšímu testování celé aplikace.

5.2.3 Direktivy

Jelikož jsou šablony uživatelského rozhraní psány v HTML, Angular.js umožňuje rozšířit syntaxi HTML o tzv. direktivy. Direktiva může představovat nový atribut, nový element, nebo výraz uzavřený ve složených závorkách. Například výše zmíněná direktiva `ng-model`, přidaná ve formě atributu k elementu input, zajišťuje synchronizaci s modelem. Angular obsahuje přednastavené direktivy připravené k použití, stejně tak je ale možné definovat vlastní direktivy.

6 Implementace

V rámci implementace bylo nutné nejprve vytvořit projekt ve frameworku PhoneGap a propojit jej s frameworkem Angular.js. Dále bylo potřeba zprovoznit potřebné pluginy pro komunikaci s hardwarem mobilního zařízení. Součástí implementace bylo také propojení s knihovnami Google Maps pro vykreslování map a Highcharts pro zobrazování grafů. V neposlední řadě bylo potřeba zprovoznit komunikaci mezi klientským a serverovým API.

6.1 Struktura aplikace

K vytvoření kostry aplikace byl použit nástroj PhoneGap CLI (Command-Line Interface). Pro použití CLI je nutné mít nainstalovaný **Node.js**, který obsahuje nástroj pro správu balíčků **npm**. Pokud máme nainstalovaný npm, můžeme stáhnout PhoneGap CLI příkazem:

```
npm install -g phonegap
```

Po úspěšné instalaci, lze vygenerovat kostru projektu příkazem:

```
phonegap create <dir> <app_id> <app_name>
```

Kde povinný parametr <dir> udává nově vytvořenou složku projektu, kde bude vytvořen zejména adresář www s podadresáři css, js, img, tedy stejnou strukturou jako u webové aplikace. Ve složce je také vygenerován konfigurační soubor config.xml, kde lze definovat atributy důležité pro překlad a nasazení na jednotlivé platformy. Z tohoto souboru se při překladu automaticky generují příslušné konfigurační soubory pro jednotlivé platformy (AndroidManifest.xml pro Android a Info.plist pro iOS). Další parametry jsou volitelné. Parametr <app_id> určuje identifikaci aplikaci v reverzně doménovém stylu např. com.example.hello a parametr <app_name> udává zobrazované jméno aplikace. Oba tyto parametry lze později změnit v souboru config.xml.

Takto vytvořenou kostru projektu, je již možné spustit ve webovém prohlížeči na počítači s nainstalovaným s nainstalovaným HTTP serverem (např. Apache), pro nasazení na mobilní platformu je nutné ještě vytvořit podsložky v adresáři platforms, toto lze provést příkazem:

```
phonegap platform add <platform_name>
```

V kořenovém adresáři aplikace je vytvořen také adresář merges, který umožňuje přizpůsobit aplikaci jednotlivým platformám. Pokud máme například ve složce `www/css` šablonu stylů `style.css` a chceme některé prvky přizpůsobit platformě android, stačí do adresáře merges/android/css/ umístit nový soubor style.css, který obsahuje požadované změny. Při spuštění na platformě android dojde ke sloučení stylů z obou soborů, přičemž se přednostně aplikují styly z adresáře merges. Na ostatních platformách zůstane vzhled nezměněn.

Po vytvoření kostry a přidání cílových platforem je aplikace připravena k nasazení na mobilním zařízeni. Zdrojový kód aplikace je psán v jazyce JavaScript, pro lepší strukturu kódu, jsem zvolil framework Angular.js. Je tedy nutné propojit Angular a PhoneGap. Do složky www/lib je třeba nakopírovat zdrojové soubory Angularu a vložit na ně odkaz v souboru www/index.html. Základní soubor app.js nakopírujeme do složky www/js a v této složce vytvoříme podadresáře controllers

a services pro ukládání logicky oddělených zdrojových souborů. Ve složce www je třeba dále vytvořit adresář views, kam budou ukládány webové šablony ve formátu HTML. Pokud budeme aplikaci ladit v prohlížeči, je potřeba zkopírovat soubor cordova.js z adresáře platforms/android/platform_www do složky www/.

6.2 Uživatelské rozhraní

Jelikož je mobilní aplikace spouštěná v internetovém prohlížeči na mobilním zařízení, je třeba brát ohled na to, že podpora některých HTML a CSS vlastností se liší v závislosti na operačním systému a verzi mobilního prohlížeče. Z toho důvodu při testování na desktopovém internetovém prohlížeči a po nasazení na mobilní zařízení, vyplynulo v některých směrech odlišné chování aplikace, způsobené právě odlišnou podporou použitých CSS vlastností mezi těmito prohlížeči. Některé mobilní prohlížeče například nepodporují správně vlastnost overflow. Konkrétně při nastavení hodnoty `overflow: auto`, je obsah pouze oříznut, ale nevytvoří se skrolovací lišta a zbytek obsahu tak zůstává nedostupný.

Uživatelské rozhraní aplikace je implementováno s využitím frameworku Mobile Angular UI, který řeší problémy s vlastností overflow použitím polyfillu Overthrow. Polyfill je část kódu (v našem případě v JavaScriptu) poskytující vlastnosti, které nejsou podporovány prohlížečem. Například pokud prohlížeč nepodporuje některé vlastnosti z HTML5 specifikace, vložením polyfillu do aplikace dosáhneme implementace požadovaného chování. V tomto případě se při přetečení textu vytvoří skrolovací lišta i na mobilním zařízení a obsah je tak přístupný v plném rozsahu.

6.3 SQLite

Lokální databáze v mobilní aplikace využívá databázového systému SQLite. Jedná se o softwarovou knihovnu v jazyce C, která implementuje relační databázový systém. Je to v současnosti pravděpodobně nejčastější databázový systém, který běží na mobilních zařízeních, vestavěných systémech a v internetových prohlížečích. Výhodou SQLite je, že nepotřebuje vlastní databázový server, ale knihovnu stačí pouze připojit k aplikaci a přistupovat k ní přes definované rozhraní. Jednotlivé databáze jsou pak uloženy v souboru ve formátu .dbm, nad kterým jsou prováděny SQL dotazy. SQLite je velice nenáročná na výpočetní zdroje a proto ji lze s výhodou použít pro databázi na straně klienta s nízkými výpočetními prostředky, například v mobilních aplikacích. Tento databázový systém naopak není vhodný například pro sdílenou databázi na serveru, do níž přistupuje v jeden okamžik velké množství uživatelů.

Při implementaci je nutné mít na paměti odlišnosti proti běžným databázovým systémům jako například MySQL. SQLite využívá tzv. typovou afinitu, což znamená, že sloupce tabulek nemají pevně daný datový typ. Dále z důvodu maximálního odlehčení knihovny zde nenajdeme plnou podporu Unicode, což má za následek například nemožnost řadit české znaky podle abecedy.

6.4 REST

Pro komunikaci mobilní aplikace s webovou službou byla zvolena architektura REST. Zkratka REST značí Representational State Transfer a představuje popis architektury navržené pro použití v distribuovaném prostředí. Návrh REST architektury se objevil poprvé v roce 2000 v disertační práci Roye Fieldinga s názvem Architectural Styles and the Design of Network-based Software

Architectures. Základní myšlenkou této architektury je pohlížet na data nebo stavy aplikace jako na zdroje (resource) a definovat jednotný přístup k těmto zdrojům. Každý zdroj je identifikován unikátním URI, přes které k němu lze přistoupit. Mezi základní principy REST architektury patří:

- stav aplikace a data jsou abstrahovány do podoby resource, každý resource je určen unikátním identifikátorem (URI, URL)
- Hypermedia jako aplikační stav (HATEOAS = Hypermedia as the Engine of Application State) – URL určuje stav aplikace. Další stavy jsou získány z odkazů, které jsou zaslány v odpovědi od serveru
- přístup k získání a manipulaci s resource je jednotný a definován pomocí čtyř tzv. CRUD operací (Create, Read, Update, Delete)
- klient nepracuje přímo s resource, ale pouze s jeho reprezentací, ta může mít různou podobu (např. XML, HTML, JSON) [15]

Na následující tabulce lze vidět, jak jsou typicky vlastnosti HTTP implementovány v podobě webové služby.

Resource	GET	PUT	POST	DELETE
URI odkazující na kolekci, např. http://example.com/resources	Vrací seznam URI případně další informace o záznamech v kolekci.	Přepsání celé kolekce jinou kolekcí.	Vytvoření nového záznamu v kolekci. URI nového záznamu je vytvořeno automaticky a vráceno v odpovědi.	Odstraní celou kolekci.
URI odkazující na jeden element např. http://example.com/resources/item17	Vrací reprezentaci odkazovaného záznamu v rámci kolekce, vyjádřenou v příslušném formátu.	Vytvoří nebo přepíše odkazovaný záznam v kolekci.	Bere odkazovaný záznam jako kolekci a v rámci ní vytvoří nový záznam.	Odstraní záznam z kolekce.

Tab. 3: Příklad webového API s použitím architektury REST [15]

API webové aplikace estimo.cz vychází z REST architektury a má následující podobu. Všechny požadavky jsou směřovány na adresu http://dev.estimo.cz/api. Každý požadavek musí být autorizován uvedením HTTP hlaviček X_ESTIMO_USERNAME obsahující registrační email uživatele a X_ESTIMO_PASSWORD obsahující API klíč uživatele. Metody pro přístup k resource jsou následující:

- GET /api/{model}/list – vrací seznam všech záznamů třídy určené jménem modelu
- GET /api/{model}/{id} – vrací konkrétní záznam podle daného id
- POST /api/{model} – vytvoří nový záznam, s atributy podle zaslaných dat
- DELETE /api/{model}/{id} – odstraní model s uvedeným id
- PUT /api/{model}/{id} – změna modelu s daným id
- POST /api/{model}/fire&action={action} – spustí akci uvedeného modelu

REST požadavky z mobilní aplikace na server a odpovědi serveru jsou zasílány ve formátu JSON (JavaScript Object Notation). Jedná se o strukturovaný formát dat, který je dobře čitelný i pro člověka a slouží pro popis dat ve formě datových objektů složených z dvojic atribut-hodnota. Používá se zejména pro přenos informací mezi webovou aplikací a serverem. Alternativou k formátu JSON je například formát XML [16].

Následující popis různých struktur ve formátu JSON je převzat z [16]:

JSON je založen na dvou strukturách:

- Kolekce párů název/hodnota. Ta bývá v rozličných jazycích realizována jako objekt, záznam (record), struktura (struct), slovník (dictionary), hash tabulka, klíčový seznam (keyed list) nebo asociativní pole.
- Seřazený seznam hodnot. Ten je ve většině jazyků realizován jako pole, vektor, seznam (list) nebo posloupnost (sequence).

V JSON jsou tyto struktury realizovány s využitím následujících konstrukcí:

Objekt je neuspořádaná množina párů název/hodnota. Objekt je uvozen znakem { (levá složená závorka) a zakončen znakem } (pravá složená závorka). Každý název je následován znakem : (dvojtečka) a páry název/hodnota jsou pak odděleny znakem , (čárka).

Pole je seřazenou kolekcí hodnot. Začíná znakem [(levá hranatá závorka) and končí znakem] (pravá hranatá závorka). Hodnoty jsou odděleny znakem , (čárka).

Hodnotou rozumíme řetězec uzavřený do dvojitých uvozovek, číslo, true, false, null, objekt nebo pole. Tyto struktury mohou být vnořovány.

Řetězcem je nula nebo více znaků kódování Unicode, uzavřených do dvojitých uvozovek a využívající únikových sekvencí (escape sequence) s použitím zpětného lomítka. Znak je reprezentován jako řetězec s jediným znakem. Řetězec je velmi podobný řetězcům z jazyků C nebo Java.

Číslo je podobné číslům z jazyků C a Java. Jedinou výjimkou je, že není používán oktalový ani hexadecimální zápis.

6.5 Cross-Origin Resource Sharing

Tato kapitola vychází ze specifikace uvedené v [17]. Při implementaci mobilní aplikace se objevil problém, při zasílání REST požadavků na server, které byly serverem odmítány. Toto chování vychází z bezpečnostního opatření, kdy není dovoleno přistupovat ke zdroji (resource) z jiné domény, než z domény, na které je zdroj umístěn. Pro povolení sdílení zdrojů mezi doménami se používá mechanismus Cross-Origin Resource Sharing (CORS), jehož princip fungování je následující. Pokud skript požaduje zdroj mimo svojí doménu (např. metodou GET), prohlížeč nejprve zašle na server OPTIONS požadavek, kterým zjistí, zda server podporuje CORS, teprve po potvrzení od serveru je zaslán původní požadavek (v tomto případě GET). Aby server deklaroval podporu protokolu CORS, musí implementovat následující hlavičky:

Povinné hlavičky:

- **Access-Control-Allow-Origin** – udává seznam domén, ze kterých je možné přistupovat k serveru, pro povolení přístupu odkudkoliv se používá znak "*"
- **Access-Control-Allow-Headers** – seznam povolených hlaviček v požadavku oddělený čárkami

- **Access-Control-Allow-Methods** – seznam povolených HTTP metod v požadavku oddělený čárkami, protože odpověď serveru může být cachována, je výhodnější uvádět seznam všech podporovaných metod a ne pouze aktuální metodu

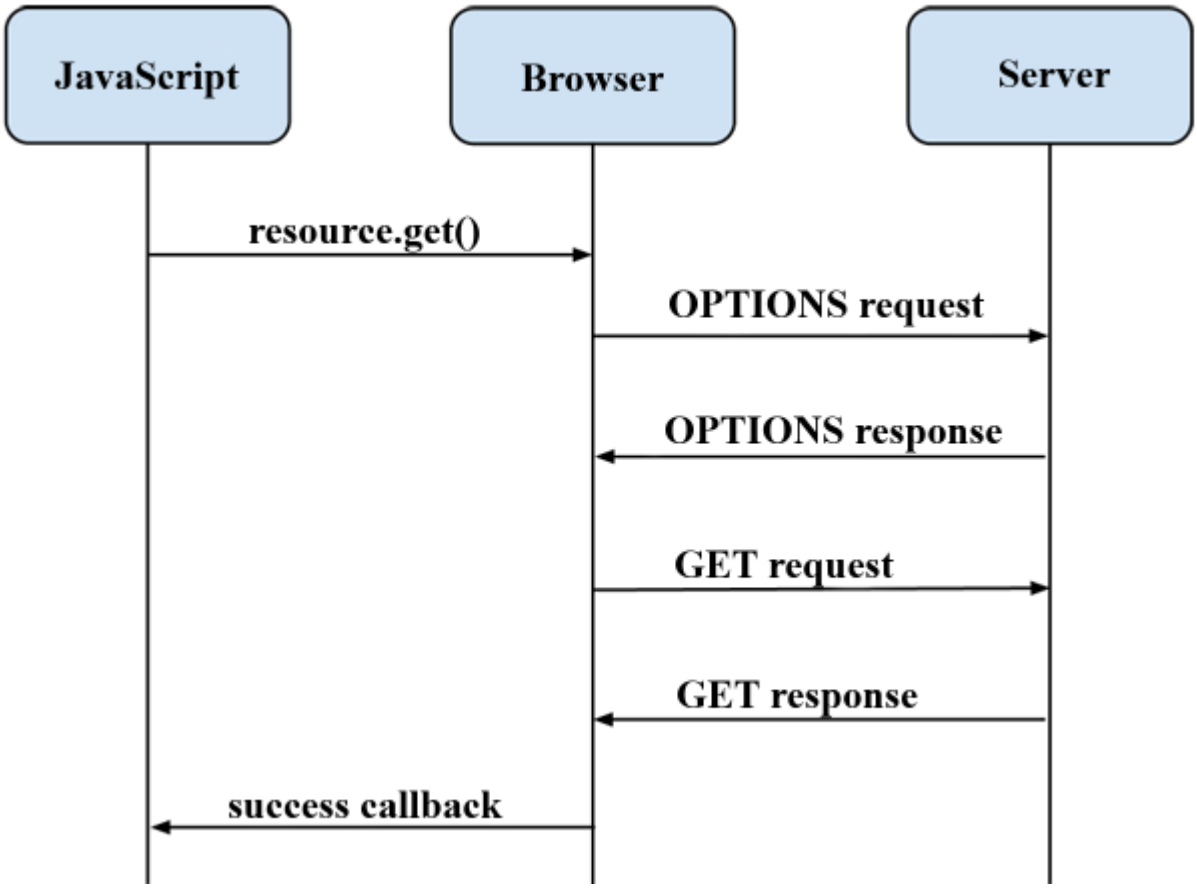
Volitelné hlavičky

- **Access-Control-Allow-Credentials** – hodnota true/false udávající, zda budou spolu s požadavkem zaslány autentizační údaje
- **Access-Control-Expose-Headers** – požadavek v protokolu CORS, může získat hodnotu pouze některých hlaviček z odpovědi (Cache-Control, Content-Language, Content-Type, Expires, Last-Modified a Pragma). Pro povolení přístupu klienta k ostatním hlavičkám, je nutné uvést seznam hlaviček oddělený čárkami.
- **Access-Control-Max-Age** – použitím protokolu CORS se zvýší zátěž serveru, protože se každý požadavek klienta zašle vlastně dvakrát. Pro snížení vytíženosti lze použít tuto hlavičku, kdy její hodnota udává čas v sekundách, po který bude odpověď serveru na CORS požadavek uložena v paměti.

Na straně klienta musí být implementovány následující hlavičky:

- **Origin** – zdrojová doména požadavku, pro povolení požadavku se musí shodovat s doménou uvedenou v odpovědi od serveru v hlavičce Access-Control-Allow-Origin
- **Access-Control-Request-Method** – požadovaná HTTP metoda, musí být povolena na straně serveru v hlavičce Access-Control-Allow-Methods
- **Access-Control-Request-Headers** – seznam speciálních hlaviček v požadavku, musí být povoleny na straně serveru v hlavičce Access-Control-Allow-Headers

Při implementaci mobilní aplikace bylo nutné přidat implementaci protokolu CORS do API webové aplikace. Protože webové API všechny příchozí požadavky autentizuje, bylo nutné vytvořit výjimku a neprovádět autentizaci u požadavků typu OPTIONS. Server místo toho na tyto požadavky odpovídá zprávou se stavem 200 OK a příslušnými CORS hlavičkami. Teprve následující požadavek se autentizuje. Důvodem výjimky je, že na straně klienta definujeme pouze požadavek GET a požadavek OPTIONS je generován automaticky prohlížečem, proto není možné přidat tomuto požadavku přihlašovací údaje uživatele. Pokud by server chtěl odmítnout CORS požadavek, stačí odpovědět zprávou 200 OK, bez nastavených CORS hlaviček, po obdržení takové zprávy prohlížeč další požadavek nezasílá.



Obr. 13: Ukázka komunikace s využitím CORS, upraveno podle [19]

6.6 API webové aplikace

Na straně webové aplikace bylo hotové API pro komunikaci pomocí REST požadavků. Při implementaci mobilní aplikace jsem však narazil na některé problémy, které bylo třeba vyřešit. Prvním problémem byla absence podpory CORS protokolu (viz kapitola 5.3), kterou bylo nutno doimplementovat.

Dále jsem v serverovém API vytvořil novou metodu pro obsluhu synchronizace s mobilním klientem. Kvůli obsluze synchronizace bylo také nutné ke každé tabulce na serveru určené k synchronizaci přidat atribut `last_sync_date`. Serverové API při přijetí synchronizačního požadavku, nejprve uloží přijatá data a poté zkontroluje, zda existují nové záznamy k odeslání klientovi. Tyto záznamy nalezne porovnáním posledního data synchronizace, které obdržel v klientově požadavku, s datem poslední synchronizace v podobě `last_sync_date` u záznamu v databázi serveru. Pokud nalezne nová data, zašle je v odpovědi klientovi.

Mobilní aplikace přináší novou funkcionalitu v podobě stopování ujetých kilometrů v rámci jednotlivých úkolů. Bylo tedy nutné do databáze na serveru přidat příslušné tabulky `coordinates` a `mileages` a vytvořit k nim modely pro obsluhu požadavků nad daty. Díky tomu je možné přes webovou službu synchronizovat ujeté cesty mezi jednotlivými zařízeními. Zobrazení cest prostřednictvím webové služby však bude teprve implementováno.

Požadavek na synchronizaci obsahuje také informaci o přihlášeném uživateli na mobilním zařízení v podobě jeho id. Podle id se vyhledá záznam o uživateli na serveru a podle jeho role se určí,

jaká data budou zaslána. Správce firemního účtu má typicky přístup k většímu rozsahu dat než běžný uživatel což musí být zohledněno i při synchronizaci.

6.7 API klientské aplikace

Komunikaci mobilní aplikace s webovým systémem zajišťuje služba EstimoResource.js, která implementuje chování datového proxy serveru, jak je popsáno v kapitole Architektura. EstimoResource je založena na modulu $resource z Angular.js, který poskytuje rozhraní pro příjem a zasílání REST požadavků na vzdálené datové zdroje. Modul obsahuje vestavěné metody get, save, query a remove pro zajištění CRUD operací nad zdrojem.

Hlavní význam EstimoResource spočívá v abstrakci datových zdrojů pro zbytek aplikace. Pokud controller zašle požadavek na data, nemusí rozhodovat o tom, jestli mají být získány z lokální databáze nebo ze vzdáleného serveru. Volání metody je stejné pro oba případy a teprve na straně služby se rozhodne, jaký datový zdroj zvolit. Stejně jako modul $resource vrací EstimoResource nejprve odkaz na prázdný objekt, který je naplněn až poté, co jsou úspěšně přijata data. Toho je dosaženo využitím principu tzv. promises.

Angular používá sliby (promises) například pro komunikaci mezi controlerem zajišťujícím logiku aplikace a datovou službou. Promise poskytuje rozhraní pro manipulaci s objektem, který je výsledkem asynchronní akce, která může, ale také nemusí být kdykoliv ukončena. Díky rozhraní Promise, jsme schopni v aplikaci standardizovat obsluhu asynchronních volání. [26]

Pro implementaci promise v mobilní aplikaci jsem použil vestavěný modul $q, který poskytuje tyto tři metody:

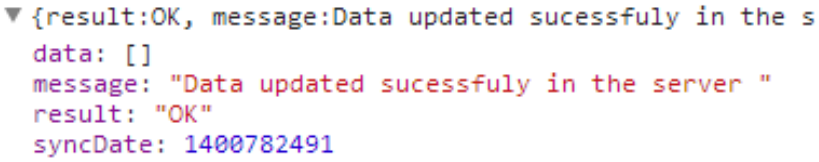
- resolve(value) – splní slib a zašle slibovanou hodnotu (value)
- reject(reason) – odmítne zaslat slibovanou hodnotu a uvede důvod (reason)
- notify(value) – poskytuje informaci o aktuálním stavu vykonávání slibu

V praxi vypadá použití následovně, controler zašle požadavek na data a služba mu “dá slib“ (objekt Promise), že mu data zašle. Controler přijme slib, ale nečeká na jeho splnění a pokračuje dál ve své činnosti. Teprve až jsou data úspěšně získána (ať už ze serveru nebo z lokální databáze), zavolá se metoda Promise.resolve(data), která dá controlleru na vědomí, že data jsou k dispozici a slib je tedy splněn. V případě neúspěchu se zavolá metoda Promise.reject(reason), čímž controller informujeme o tom, že při přijetí dat nastala chyba a slib je odmítnut. Promise se v angularu používá v kombinaci s metodou then(), která přijímá dva parametry: první parametr je funkce, která se vykoná, je-li slib splněn, druhý parametr je funkce vykonaná v případě nesplnění slibu.

Součástí klientského API je také metoda pro synchronizaci databáze. Synchronizaci iniciuje vždy klient, který zašle na server požadavek, který obsahuje JSON objekt ve formátu:

Obr. 14: Příklad struktury synchronizačního požadavku od klienta

kde objekt `data` obsahuje atributy odpovídající názvům tabulek určených k synchronizaci, každý atribut pak obsahuje pole nových záznamů, které mají být uloženy na serveru. Server po přijetí takového požadavku uloží nové záznamy a zkontroluje, zda jsou k dispozici pro klienta nová data porovnáním časového razítka `lastSyncDate` uvedeného v požadavku a atributu `last_sync_date` u záznamů v databázi na serveru. Server poté odpoví zprávou obsahující JSON objekt ve formátu:



Obr. 15: Příklad struktury synchronizační odpovědi od serveru

kde opět objekt `data` obsahuje pole nových záznamu, které mají být uloženy, tentokrát na klientovi.

6.8 Distribuovaná databáze

Z důvodu replikace centrální databáze ze serveru do několika mobilních zařízení, přichází na řadu otázka bezpečnosti dat a zachování soukromí jednotlivých uživatelů systému. Jakmile se uživatel přihlásí na mobilního klienta a synchronizuje se databáze, je třeba zajistit, aby se synchronizovala data přístupná pouze právě přihlášenému uživateli. Tento přístup je rozdílný oproti webové aplikaci s centrální databází, kde jsou data všech uživatelů na jednom místě, ale přihlášený uživatel k databázi přistupuje výhradně přes webovou aplikaci. Jakmile však umístíme databázi na klienta v tomto

případě mobilní zařízení, ztrácíme plnou kontrolu nad řízením přístupů k databázi. Můžeme omezit pouze zobrazení dat v rámci aplikace, ale uživatel může k databázi přistoupit přes externí nástroje pro správu databáze v mobilním zařízení. Z toho důvodu není možné do databáze na klientovi ukládat jakákoliv data, která přihlášený uživatel není oprávněn vidět.

V mobilní aplikaci se jedná zejména o tabulku iaccount, kde jsou obsaženy všechny registrované účty webové aplikace a tabulku users obsahující seznam všech registrovaných uživatelů. Tyto tabulky na klientovi mohou obsahovat pouze záznamy související s aktuálně přihlášeným uživatelem, server tedy nesmí zasílat obsah celé tabulky. Z toho důvodu nelze přihlašovací údaje uživatele ověřovat offline, protože na klientovi nedržíme seznam všech uživatelů aplikace, ale musíme zaslat autentizační požadavek na server pro ověření přihlašovacích údajů v serverové databázi. Ostatní tabulky již obsahují pole udávající vlastníka záznamu a jejich filtrování probíhá automaticky již na serveru, data tedy nejsou vůbec zaslána (viz kapitola 4.4.2).

Přihlášený uživatel smí mít v databázi uložena pouze ta data, ke kterým má přístup. Problém nastane, pokud se uživatel odhlásí a na stejném zařízení se přihlásí jiný uživatel, v tom případě může opět dojít ke kompromitaci dat. Řešením je mazání databáze mezi jednotlivými uživatelskými sezeními. Aby k tomu nedocházelo při opakovaném přihlášení stejného uživatele, ukládá se při odhlášení id posledního uživatele do localStorage. Po dalším přihlášení se zkontroluje, zda se jedná o stejného uživatele jako minule, v tom případě není třeba databázi mazat, nebo jde o nového uživatele a databáze musí být smazána. Před smazáním se provede ještě synchronizace původní databáze na server, aby nedošlo ke ztrátě dat předchozího uživatele.

6.9 Google Maps API

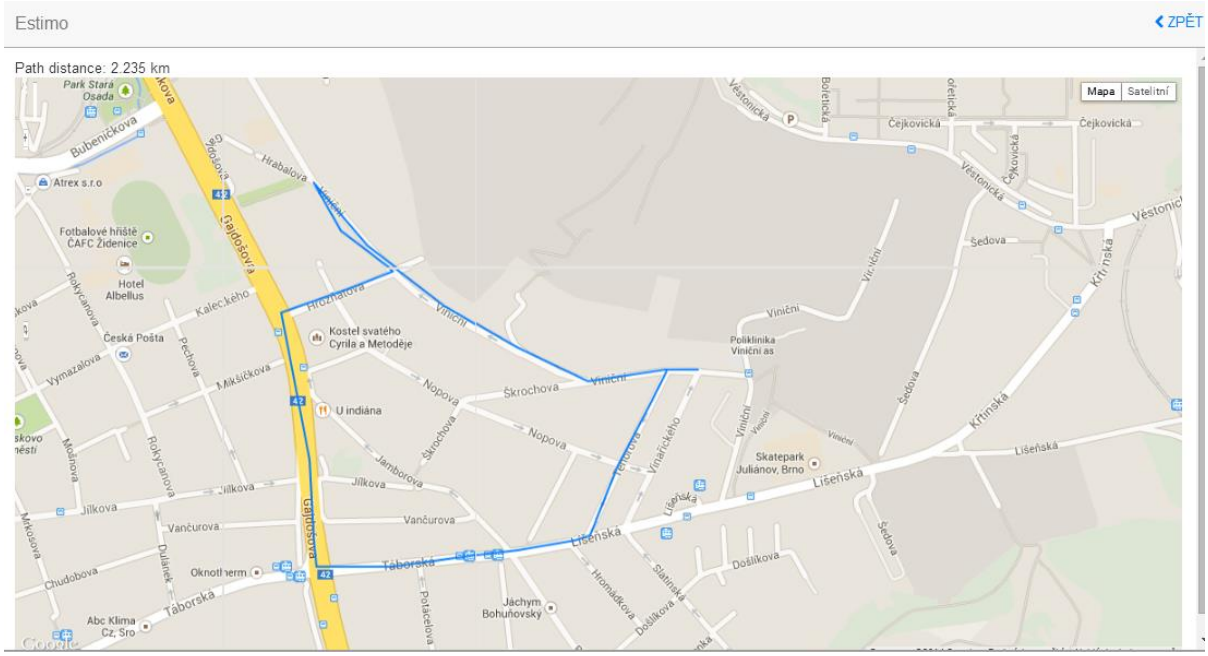
Protože jeden z hlavních požadavků na aplikaci je podpora sledování ujetých cest a jejich vyúčtování, musí aplikace implementovat sledování GPS souřadnic a jejich zobrazení na mapě. Pro práci s mapami je v aplikaci použito Google Maps JavaScript API v3. Toto API umožňuje zobrazit na mapě trasu vypočítanou ze zaznamenaných bodů a určit ujetou vzdálenost. Pro výpočet vzdálenosti je možné využít více metod. Během implementace mobilní aplikace jsem vyzkoušel následující dvě metody: výpočet vzdálenosti pomocí Google Maps API Directions Service nebo s využitím Polyline (lomené čáry).

Služba Directions zasílá požadavek na nalezení cesty mezi dvěma zadanými body. Nalezená trasa v sobě již nese informaci o ujeté vzdálenosti a trasa kopíruje silnici. Nevýhodou je ovšem omezení v počtu průjezdních bodů na 8. Což v našem případě, kdy sledujeme pohyb uživatele a zaznamenáváme desítky GPS souřadnic, rozhodně nestačí. Řešením je cestu rozdělit na menší úseky, pro ty vypočítat trasu a vypočtené trasy poté opět spojit do celkové ujeté trasy. Tato metoda poskytuje dobré výsledky, pokud je vysoká přesnost zaznamenaných GPS bodů. V okamžiku, kdy přesnost klesne, a body se začnou objevovat s větší odchylkou, začne se negativně projevovat snaha umístit trasu na existující silnici. V důsledku toho vzniknou kruhové nebo překrývající se trasy.

Lomená čára (Polyline) představuje na první pohled rychlejší způsob vykreslení cesty, přijímá totiž pole bodů, které poté spojí lomenou čárou. Výhodou je, že počet bodů není explicitně omezen, odpadá tedy nutnost cestu dělit na menší úseky. Nevýhodou je menší počet vestavěných funkcí pro práci s lomenou čárou, například není k dispozici vestavěná funkce pro výpočet délky, lze však použít funkci z knihovny google.maps.geometry. Další nevýhodou může být v některých případech fakt, že čára se vykreslí přesně tam, kde jsou umístěny GPS souřadnice bez ohledu na to, jestli je v okolí cesta nebo ne. Při testování trackování GPS během jízdy v autě však bylo stopování dostatečně přesné a lomená čára tedy vykazovala lepší výsledky, než trasa počítaná pomocí Directions. Pokud se objeví nepřesnost a bod mimo aktuální trasu, lomená čára pouze vykreslí cestu k němu vzdušnou čarou, což

při rozumné míře výskytu nepřesností ve výsledku představuje zanedbatelné zkreslení. Naproti tomu výpočet pomocí Directions, se snaží vždy najít cestu mezi dvěma body po silnici, tudíž při výskytu nepřesnosti může dojít ke zkreslení až o stovky metrů, kvůli "objízdné trase" po nejbližší silnici.

Implementace map na platformě Android proběhla bez problému. Pro korektní načtení Google Maps na platformě iOS bylo však nutné přidat domény google.com, googleapis.com, gstatic.com a googleusercontent.com do seznamu povolených domén (whitelist) v souboru config.xml. Níže je uvedena ukázka obrazovky aplikace s ujetou trasou zobrazenou na mapě.



Obr. 16: Zobrazení ujeté trasy.

6.10 Geolocation plugin

Tato kapitola vychází ze specifikace uvedené v [21]. Pro sledování pozice zařízení používá aplikace Geolocation plugin frameworku PhoneGap. Tento plugin je založen na HTML5 Geolocation API, aplikace tak zaznamenává pozici nejen na mobilním zařízení pomocí GPS přijímače, ale i pokud je spuštěna v prohlížeči s podporou HTML5, pomocí informací ze sítě. Geolocation API poskytuje tři hlavní metody:

- geolocation.getCurrentPosition – vrací GPS souřadnice aktuální pozice zařízení
- geolocation.watchPosition – sleduje pohyb zařízení a při každé změně pozice, vrací aktuální souřadnice
- geolocation.clearWatch – zastavuje sledování pozice spuštěné metodou watchPosition

První dvě metody přijímají jako parametr callback pro úspěšné a neúspěšné získání pozice a také objekt geolocationOptions s atributy:

- enableHighAccuracy – při výchozím nastavení (false), se zařízení snaží zjistit pozici pomocí připojení k síti, nastavením na hodnotu true, sdělíme požadavek na co nejpřesnější pozici, zařízení poté zjišťuje pozici skrze GPS přijímač

- timeout – maximální doba v milisekundách, po kterou bude zařízení čekat na úspěšné zjištění pozice, po vypršení této doby, se zavolá callback pro neúspěšné získání pozice
- maximumAge – doba v milisekundách určující maximální povolené stáří cachovaných pozic

Při použití geolocation API na zařízeních s operačním systémem Android 2.x je nutné nastavit parametr enableHighAccuracy na hodnotu true.

6.11 Běh na pozadí

Pokud uživatel spustí trackování pohybu a pak spustí jinou aplikaci nebo uzamkne telefon, musí trackování pokračovat na pozadí, dokud jej uživatel nevypne. Tohoto chování lze dosáhnout různými způsoby v závislosti na cílové platformě.

Pro záznam GPS souřadnic na pozadí v systému iOS je nutné upravit konfigurační soubor Info.plist. Konkrétně je třeba přidat do pole s názvem "Required background modes" nový řádek s hodnotou "App registers for location updates".

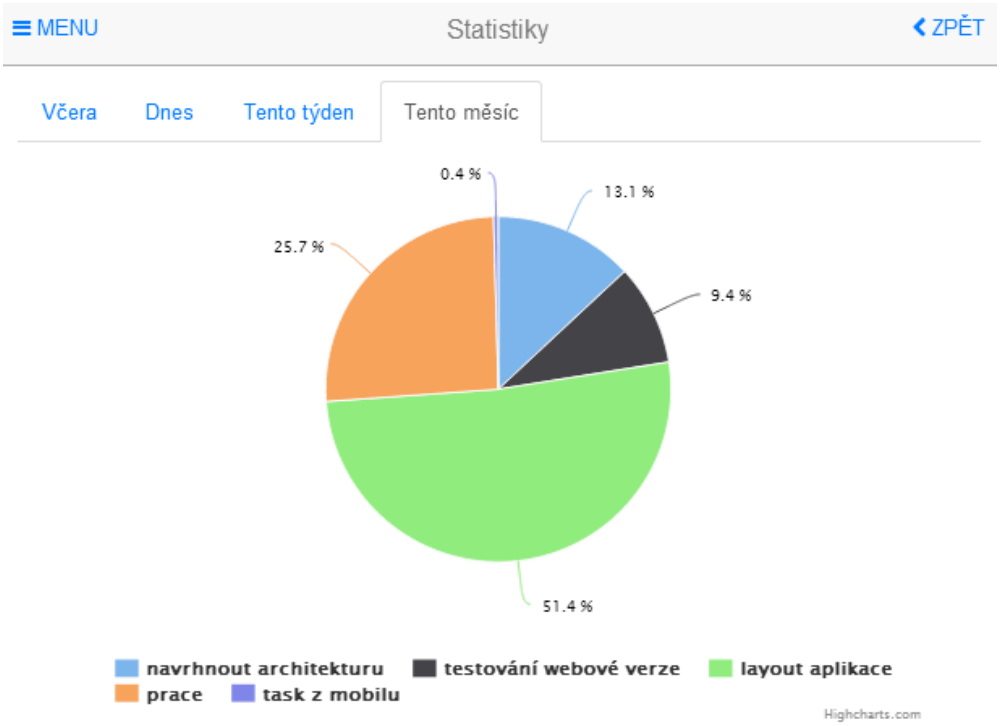
Na platformě Android je trackování na pozadí o poznání složitější. V principu jde o to, že Android neumožňuje vykonávání JavaScriptu na pozadí, proto je nutné napsat vlastní Service v jazyce Java. Tato služba se spustí, jakmile je aplikace přesunuta na pozadí a pokračuje v zaznamenávání souřadnic, přičemž lze službě nastavit, aby při úspěšném nalezení souřadnic spustila callback na straně JavaScriptu. V implementované mobilní aplikaci jsem pro záznam GPS souřadnic na pozadí použil PhoneGap plugin background geolocation (dostupný na: https://github.com/christocracy/cordova-plugin-background-geolocation).

Dalším požadovaným chováním aplikace je, aby se po přesunu na pozadí a opětovnému přivolání do popředí otevřela na poslední obrazovce. Proto je nutné upravit konfigurační soubor config.xml, kde u parametru „launchMode" změníme přednastavenou hodnotu „standard" (při každém spuštění aplikace se vytvoří nová instance) na hodnotu „singleInstance", kdy se při spuštění zkontroluje, zda už aplikace neběží na pozadí a pokud ano obnoví se předchozí instance. V jednom okamžiku je tak v paměti spuštěna maximálně jedna instance aplikace.

6.12 Zobrazení grafů

Denní, týdenní a měsíční výkazy odpracovaných hodin jsou v aplikaci k dispozici ve formě přehledných grafů. Na webové službě jsou grafy implementovány pomocí Google Charts API, které je vykresluje ve formátu SVG (Scalable Vector Graphics). Jedná se o formát pro popis 2D grafických objektů pomocí značkovacího jazyka XML. Na rozdíl od rozšířených formátu pro rastrovou grafiku typu PNG nebo GIF, slouží SVG pro grafiku vektorovou. Formát SVG, ale bohužel není podporován ve webových prohlížečích systému Android ve verzích 2.x. Z toho důvodu jsem zvolil pro grafy v mobilní aplikaci knihovnu Highcharts (dostupná na: http://www.highcharts.com), která podporuje i prohlížeče ve starších verzích systému. Grafy jsou primárně vykreslovány také ve formátu SVG, ale pokud knihovna detekuje chybějící podporu tohoto formátu, vykreslí graf místo toho do elementu <canvas>. HTML5 element <canvas> je určen pro dynamické vykreslování grafických prvků pomocí JavaScriptu. Samotný element slouží pouze jako kontejner, veškeré kreslení je na straně skriptu.

Ve formátu SVG graf po najetí na příslušnou část zobrazí tooltip s názvem úkolu a počtem odpracovaných hodin. Pokud SVG není podporováno, graf neobsahuje interaktivní tooltipy. Na obrázku níže je uveden příklad vzhledu koláčového grafu odpracovaných hodin.



Obr. 17: Přiklad grafu odpracovaných hodin

6.13 Lokalizace

Protože ve specifikaci aplikace byl uveden také požadavek na multijazyčnost, bylo potřeba do aplikace implementovat systém pro překlad textových řetězců. Lokalizace jsem dosáhnul s využitím modulu angular-gettext. Modul detekuje textové řetězce určené k překladu, pomocí atributu translate přidaného k vybraným HTML elementům, například:

```
<a href="#/my-time" translate>My time</a>
```

Takto označené textové řetězce je třeba nějakým způsobem extrahovat a uložit do souboru pro definování překladu, v tomto případě do šablony ve formátu POT. Extrakce je provedena pomocí Gruntu, což je nástroj pro automatizované spouštění naplánovaných úloh v JavaScriptu. Angular-gettext poskytuje dvě automatizované úlohy: první pro extrakci textu do šablony POT a druhý pro konverzi souborů ve formátu PO do JavaScriptu. Vygenerovanou šablonu ve formátu POT je možné otevřít v některém z programů pro překlad, například Transifex, Pootle nebo Poedit. Překlad je tak prováděn vně aplikace a nemusí být vytvářen vývojářem, ale může být zadán externímu překladateli. Z šablony POT je vygenerován soubor PO pro jednotlivé jazyky. Po přeložení textových řetězců do cílového jazyka je opět použit Grunt, který zajistí automatickou konverzi PO soborů do souborů v JavaScriptu. Ve výsledku lze za běhu aplikace dynamicky přepínat mezi češtinou a angličtinou.

6.14 Storage plugin

Pro přístup do lokální databáze zařízení, je použit PhoneGap plugin Storage. Tento plugin je založen na specifikaci W3C Web SQL Database, se kterou je plně kompatibilní, což zajišťuje vzájemnou zaměnitelnost. Pokud je na zařízení detekována podpora výše uvedené specifikace, jsou použity vestavěné funkce. Plugin je pak použit na zařízeních, která specifikaci nepodporují. Specifikace W3C Web SQL Database byla původně vytvořena za účelem poskytnout jednotné API pro přístup k databázi pomocí různých databázových systémů založených na jazyku SQL. W3C později uvedlo, že se specifikace dostala do slepé uličky, protože všichni hlavní implementátoři použili stejný databázový systém (SQLite), čímž došlo k narušení původní myšlenky obecnosti specifikace a W3C ji tak přestalo nadále udržovat. Díky použitému pluginu, však bude mobilní aplikace fungovat i v případě ukončení podpory specifikace Web SQL na zařízeních [24].

Storage API poskytuje asynchronní přístup k databázi. Připojení k databázi probíhá pomocí funkce openDatabase() s parametry: jméno databáze, verze databáze, zobrazované jméno databáze a velikost ukládaných dat v bytech. Pokud databáze uvedeného jména neexistuje, je nejprve vytvořena a poté je vrácen objekt Database pro manipulaci s daty. Database objekt obsahuje metodu transaction() pro vykonání databázové transakce. Metoda přijímá jeden povinný parametr transaction_callback a dva nepovinné parametry error_callback a success_callback. Metoda je asynchronní, po zavolání tedy ihned vrací řízení do aktuální větve a poté spustí asynchronní vykonávání funkce transaction_callback a předá jí jako parametr objekt SQLTransaction. Tento objekt obsahuje metodu executeSql() s povinným parametrem sql_statement, který očekává řetězec obsahující dotaz v jazyce SQL. Dále je možné metodě předat pole parametrů SQL dotazu, success_callback a error_callback. Pole parametrů dotazu slouží k substituci předávaných hodnot v řetězci sql_statement a obraně před SQL injection. Po zavolání executeSql() se opět asynchronně vykoná funkce success_callback nebo error_callback (dle výsledku SQL dotazu) a předá se jí jako parametr objekt SQLResultSet. Tento objekt obsahuje tři parametry: insertId obsahující id nově vytvořeného záznamu (v případě volání SQL INSERT), rowsAffected udávající počet ovlivněných řádků a objekt SQLResultSetList obsahující výsledek SQL dotazu [25].

Kromě přístupu k SQLite databázi v mobilním zařízení, poskytuje Storage API také metody pro přístup k W3C Storage rozhraní. Skrze volání window.localStorage se dostaneme k lokální databázi klíč-hodnota uložené ve webovém prohlížeči. Voláním metod setItem(key, value) a getItem(key) poté manipulujeme nad daty v tomto úložišti. V mobilní aplikaci jsem použil localStorage pro uložení informací o přihlášeném uživateli. Data v localStorage jsou smazána až na požadavek, takže je její použití vhodnější než podobné rozhraní window.sessionStorage, které maže data mezi jednotlivými běhy aplikace.

7 Funkce aplikace

Aplikace je rozdělena na několik obrazovek, mezi kterými lze přepínat pomocí menu. V této kapitole jsou popsány jednotlivé obrazovky a příslušná funkcionalita, kterou poskytují.

7.1 Poslední úkoly

Nejčastější použití aplikace je stopování času. Uživatel tedy musí mít možnost spustit stopování aktuálního úkolu co možná nejrychleji po otevření aplikace. Pro rychlé stopování tak slouží obrazovka Poslední úkoly, kde je zobrazeno deset naposledy stopovaných úkolů. Uživatel díky tomu nemusí procházet seznam všech úkolů nebo se dostávat k úkolu přes nadřazeného klienta a projekt, ale má rychlý přístup k právě aktuálním úkolům a může snadno pokračovat ve stopování práce.

7.2 Správa trackerů

Na této obrazovce má uživatel možnost prohlížet všechny trackery v rámci firemního účtu. Trackery jsou řazeny do kategorií dle typu na klienta, projekt a úkol. Kliknutím na klienta se objeví jeho detail včetně všech projektů, podřazených k tomuto klientovi. Stejně tak je možno zobrazit detail projektu s podřazenými úkoly a nakonec detail úkolu. K úkolu je možné stopovat odpracovaný čas a ujetou vzdálenost, tlačítka pro spuštění stopování jsou umístěna na jeho detailu.

Vytváření nových trackerů je možné pouze v režimu online, kdy se tracker ukládá přímo na server. Po úspěšném uložení klient iniciuje synchronizaci a obdrží od serveru aktuální data, včetně nově vytvořeného trackeru. Stejným způsobem probíhá také mazání, tracker se smaže přímo na serveru a poté se pomocí synchronizace změny promítnou do lokální databáze. Protože jsou trackery ukládány do stromové hierarchie, musí mít každý úkol nadřazený projekt a každý projekt nadřazeného klienta. Z toho důvodu probíhá vytváření nových trackerů na detailu trackeru, čímž se nově vytvořený zařadí pod stávající. Výjimkou jsou klienti, kteří jsou na nejvyšší úrovni, a proto přidání nového klienta probíhá na obrazovce obsahující seznam všech klientů

7.3 Správa času

Jakmile uživatel zahájí stopování času stráveného na úkolu, vytvoří se časový blok s údaji o stopovaném úkolu, začátku a konci stopování a délce trvání. Pro přístup k nastopovaným časovým úsekům slouží obrazovka **Můj čas**, kde jsou časové bloky vypsány v podobě přehledné tabulky. Tento výpis lze filtrovat podle přednastavených časových filtrů na denní, týdenní a měsíční výpis. Dále je možné filtrování podle úkolu. Správce firemního účtu má možnost filtrování také podle uživatelů. V případě, že uživatel nastopoval čas chybně, například zapomněl včas vypnout stopky, je možné časové bloky ve výpisu také upravit. S možností zpětné manipulace s nastopovaným časem vzniká riziko podvodu ze strany uživatele, každý časový blok s sebou proto nese informaci o času poslední změny, která je viditelná pro správce firemního účtu. Ten může zkontrolovat, kdy uživatel naposledy změnil uložený čas a snáze tak odhalit podezřelé manipulace s časovými bloky nashromážděné například s blížícím se měsíčním vyúčtováním.

7.4 Správa cest

Na této obrazovce uživatel vidí přehled svých ujetých cest, zejména datum cesty, ujetou vzdálenost a ke kterému úkolu se cesta vztahuje. Po kliknutí na detail cesty se zobrazí mapa s vyznačenou trasou. Pro vykreslení mapy je použito rozhraní Google Maps, uživatel má tedy k dispozici funkce známé z tohoto prostředí jako přiblížení a oddálení mapy nebo přepínání mezi klasickou a satelitní mapou. Z použití Google Maps, plyne omezení, že pro zobrazení detailu cesty musí být uživatel připojen k internetu.

Správce firemního účtu má možnost zobrazit cesty všech uživatelů pomocí jednoduchého filtru. Aplikace mu tak umožní kontrolu vykázaných cest a sledování pohybu uživatelů.

7.5 Statistiky

Na této obrazovce uživatel nalezne statistiky odpracovaného času ve formě přehledných grafů. Statistiky lze filtrovat na denní, týdenní a měsíční zobrazení. Administrátor firemního účtu má možnost zobrazení statistik pro všechny členy týmu a přepínání mezi nimi pomocí filtru. Knihovna pro vykreslení grafů je uložena lokálně v aplikaci, grafy je tak možné prohlížet i v režimu offline. Výjimkou jsou zařízení se systémem Android 2.x, kde je kvůli chybějící podpoře SVG nutné připojení k internetu pro vykreslení grafu.

7.6 Tým

Na této obrazovce se vypíše seznam členů týmu obsahující email a roli každého člena. Uživatel tak má možnost zjistit, kteří další uživatelé jsou přiděleni pod stejný účet a případně je kontaktovat přes uvedený email.

7.7 Nastavení

Jelikož je aplikace koncipována jako multijazyčná, má uživatel možnost zvolit jazyk aplikace v záložce **Nastavení**. V současnosti je na výběr angličtina nebo čeština.

7.8 Databáze

Na této obrazovce je k dispozici tlačítko pro ruční synchronizaci databáze.

7.9 Přihlašování

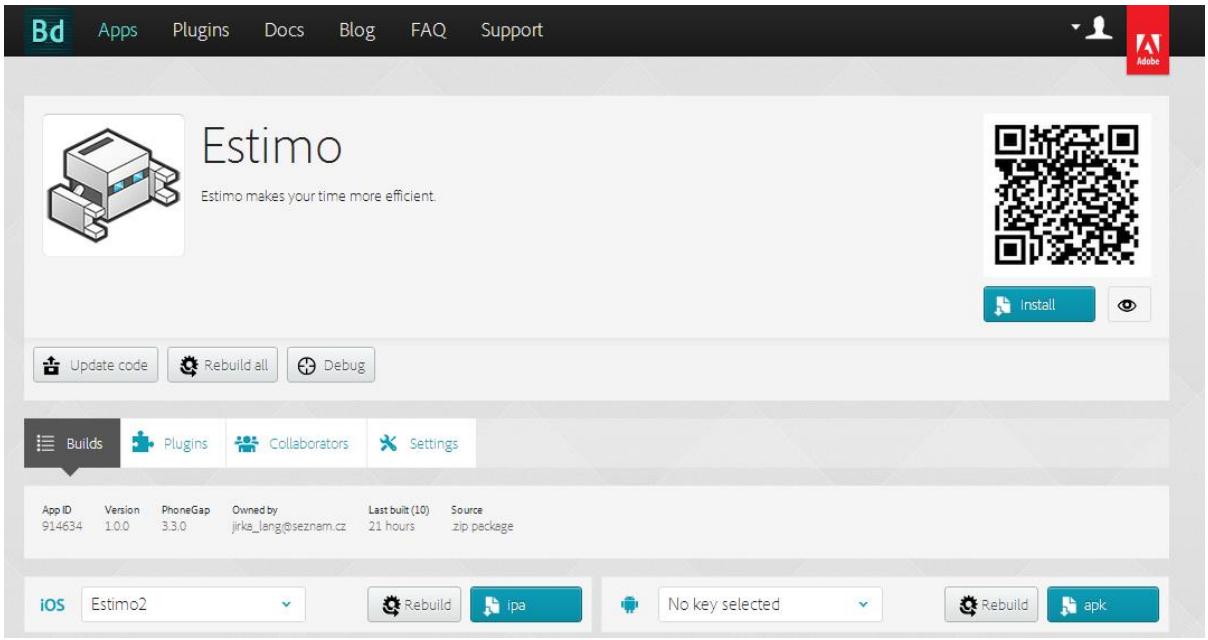
Uživatel se přihlašuje do mobilní aplikace stejným účtem jako na webovou službu. Při prvním spuštění aplikace se objeví přihlašovací formulář, který požaduje zadání emailu a hesla. Pro úspěšné ověření uživatele musí být aplikace připojena k internetu. Pokud uživatel nemá vytvořen žádný účet, musí se nejprve registrovat na webové službě.

7.10 Režim offline

Při výpadku připojení k internetu je aplikace schopna díky přítomnosti lokální databáze nadále fungovat, některé funkce však nejsou v režimu offline dostupné. Uživatel se například nemůže odhlásit a přihlásit pod jiným účtem. Taktéž nemůže vytvářet nové trackery ani mazat existující, může však s nimi pracovat ve smyslu prohlížení seznamu trackerů a zobrazení detailu. Stopování času a stopování cest funguje v režimu offline bez omezení. Také tabulkové výpisy odpracovaných hodin a ujetých cest jsou k dispozici i bez připojení k internetu. V režimu offline ale nelze zobrazit detail cesty na mapě. Grafické statistiky fungují offline na většině mobilních zařízení, kromě systému Android 2.x. Funkci synchronizace databáze z pochopitelných důvodů v režimu offline nelze spustit.

8 Nasazení a testování

Aplikace byla vyvíjena pod operačním systémem Windows s použitím frameworku PhoneGap a Android SDK. PhoneGap nabízí dvě možnosti kompilace a nasazení aplikace do mobilního zařízení. Pokud je na počítači nainstalováno příslušné SDK cílové platformy, je možné aplikaci zkompilovat a nainstalovat skrze PhoneGap CLI (Command Line Interface). V případě, že SDK není dostupné, je řešením kompilace pomoci služby PhoneGap Build dostupné na serveru https://build.phonegap.com. Protože se jedná v podstatě o webovou aplikaci, je možné testovat základní funkcionalitu i na počítači s nainstalovaným HTTP serverem (např. Apache) prostřednictvím internetového prohlížeče. Pro testování je nejvhodnější prohlížeč Google Chrome, který podporuje W3C Web SQL specifikaci a poskytuje vývojový nástroj pro vizualizaci vytvořené databáze. Pro testování v desktopovém prohlížeči je také vhodný doplněk Ripple Emulator (dostupný na: http://emulate.phonegap.com/), který umožňuje například snadno emulovat odlišné rozměry jednotlivých mobilních zařízení nebo nastavit spouštění událostí mobilního rozhraní.



Obr. 18: Ukázka kompilačního prostředí PhoneGap Build [29]

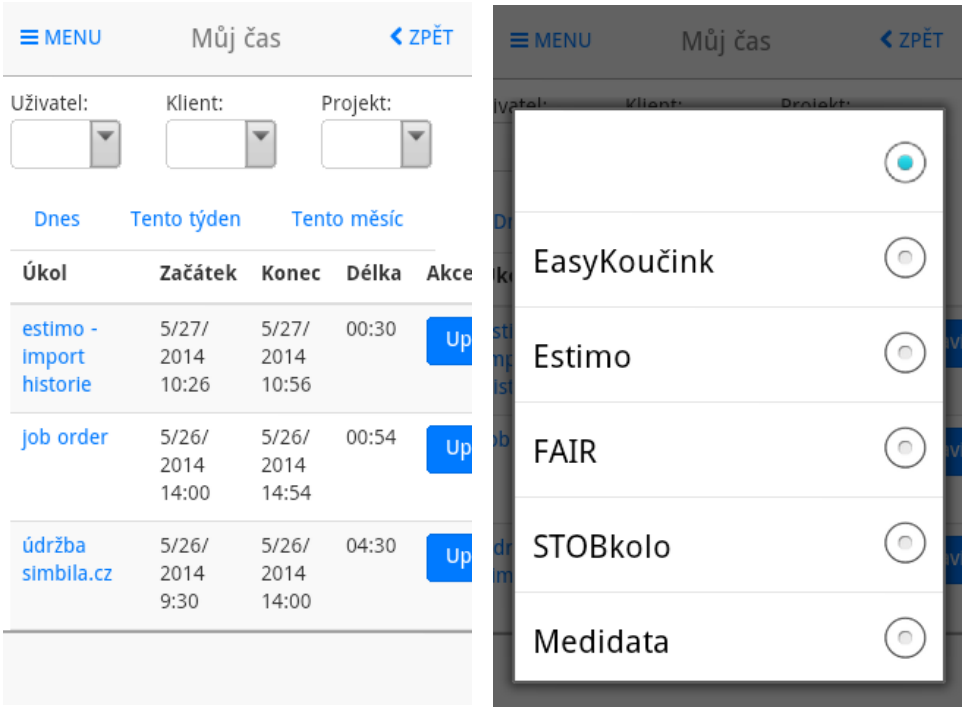
8.1 Android

Díky vývojovému prostředí bylo možné aplikaci kompilovat s použitím PhoneGap CLI, kdy se příkazem `phonegap run android` ve zdrojové složce aplikace, provede jak kompilace, tak instalace aplikace na připojené zařízení či do emulátoru. V případě více připojených zařízení, lze cíl explicitně uvést pomocí parametru `--device`. Ladění aplikace lze poté provádět, například ve vývojovém prostředí Eclipse.

Aplikace byla testována na mobilním telefonu Samsung Galaxy Ace se systémem Android 2.3.7. Tato verze systému je již dnes poměrně zastaralá, nejnovější je verze 4.4. Testování i na starších verzích systému má však smysl, neboť většina výrobců mobilních telefonů nevydává

pravidelné systémové aktualizace a díky tomu jsou starší verze systému Android stále poměrně rozšířené. Aplikaci bylo nutné na mnoha místech upravovat právě pro docílení zpětné kompatibility. Šlo například o získávání GPS souřadnic nebo vykreslování grafů (viz kapitoly 6.10 a 6.12).

Uživatelské rozhraní se na obou platformách vzhledově neliší, výjimku tvoří pouze ovládací prvky typu rozbalovací seznam, které jsou vykresleny v nativním stylu každé platformy (viz obr.19 a obr.20).

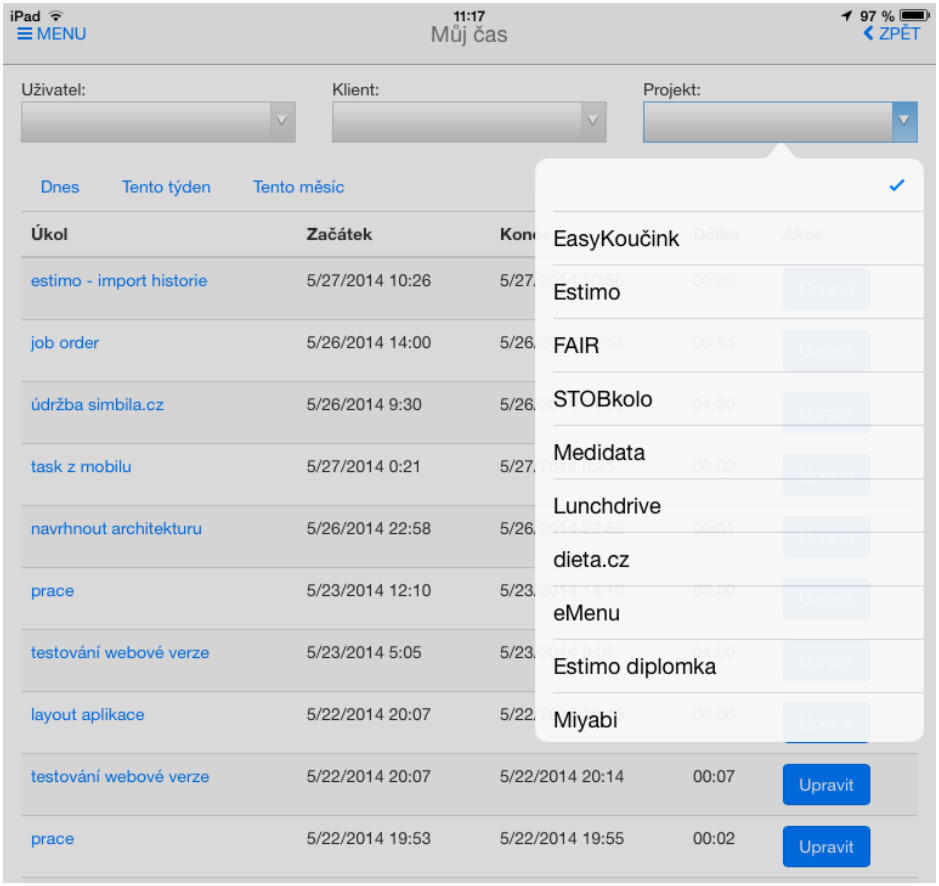


Obr. 19: Ukázka aplikace na platformě Android

8.2 iOS

Instalace iOS SDK je možná pouze na počítače s operačním systémem iOS, z toho důvodu jsem zvolil pro kompilaci aplikace na platformu iOS službu PhoneGap Build. Nejprve je nutné registrovat se jako Apple Developer za poplatek 99 $/rok, poté je potřeba aplikaci vygenerovat certifikát a ten nahrát na server PhoneGap Build. Zdrojový kód lze na server nahrát buď jako ZIP archiv, nebo účet propojit přímo s GIT repozitářem. Přeloženou aplikace je poté možné stáhnout ve formátu IPA a nainstalovat do zařízení přes program iTunes, nebo přímo v zařízení přistoupit na server phonegapu, buď zadáním URL nebo přes vygenerovaný QR kód a odtud stáhnout aplikaci do zařízení. PhoneGap Build umožňuje vzdálené ladění aplikace díky vloženému skriptu ve stránce, který komunikuje se serverem a ten poté podává informace o aktivitě na zařízení. Vzdálené ladění aplikace probíhá přes server http://debug.build.phonegap.com/, kde jsou k dispozici ladící nástroje podobné nástroji firebug v prohlížeči Mozilla Firefox. Ladíce nástroje umožňují procházení a editaci stromu HTML elementů a jejich přiřazených CSS stylů, sledování síťové komunikace, zobrazení logovací konzole a také procházení SQLite databáze a úložiště localStorage přímo v mobilním zařízení.

Aplikace byla testována na tabletu Apple iPad Wifi s operačním systémem iOS 7.1.1. Jedná se o první generaci iPadu, ale jelikož Apple podporuje automatické aktualizace systému i na starších zařízeních, testoval jsem na nejnovější verzi systému iOS. Oproti platformě Android nebylo nutné aplikaci tolik přizpůsobovat pro běh na iOS. Jedinou větší změnou byla konfigurace whitelistu povolených domén pro načtení Google Maps.



Obr. 20: Ukázka aplikace na platformě iOS

9 Závěr

V této práci byly nastíněny možnosti vývoje multiplatformních mobilních aplikací spolu s konkrétními nástroji. Z nich jsem vybral vhodné technologie pro implementaci požadované mobilní aplikace. Dále jsem navrhnul architekturu této aplikace s ohledem na použité technologie a požadavky zadavatele. Podle tohoto návrhu jsem aplikaci implementoval s využitím frameworků PhoneGap a Angular.js a následně ji otestoval na reálných mobilních zařízeních běžících na platformách iOS a Android.

Požadavky zadavatele na aplikaci se v průběhu implementace měnily, v závislosti na výsledcích probíhajícího testování webové služby. Zadavatel se zaměřil na prioritní funkcionalitu aplikace a některé doplňkové služby byly z webové aplikace odstraněny a o jejich konečném zapracování do aplikace se bude rozhodovat až při vývoji dalších verzí. Mezi tyto doplňkové služby patří například vytváření hlasových poznámek. Z důvodu absence této funkcionality na webové službě nebyla nakonec zakomponována ani do mobilní aplikace. Rovněž požadavek na psaní komentářů k úkolům byl vyhodnocen jako neprioritní. Tvorbu hlasových poznámek a vkládání komentářů lze tedy zmínit jako možná rozšíření mobilní aplikace do budoucna.

Aplikace byla otestována na platformách iOS a Android, které byly požadovány zadavatelem jakožto dva v současnosti nejrozšířenější mobilní operační systémy. Pokud by nastal v budoucnu nárůst prodeje mobilních zařízení s jinou platformou, která je podporována použitým frameworkem, bude možné aplikaci po drobných úpravách nasadit i na tuto platformu.

Literatura

[1] ALLEN, Sarah, Vidal GRAUPERA a Lee LUNDRIGAN. *Pro smartphone cross-platform development: iPhone, BlackBerry, Windows Mobile, and Android development and distribution*. New ed. New York, N.Y.: Apress, 2010. ISBN 978-143-0228-684.

[2] GHATOL, Rohit a Yogesh PATEL. *Beginning PhoneGap: mobile web framework for JavaScript and HTML5*. New York: Distributed to the book trade worldwide by Springer Science Business Media, 2012, xii, 332 stran. Books for professionals by professionals. ISBN 14-302-3903-4.

[3] ANDERSON, John. *Appcelerator Titanium: up and running*. First edition. Sebastopol: O'Reilly, 2013, ix, 140 stran. ISBN 14-493-2955-1.

[4] MROZEK, Jakub. Začínáme s AngularJS. In: Zdrojak.cz [online]. 2012 [cit. 2013-12-13]. Dostupné z: http://www.zdrojak.cz/clanky/zaciname-s-angularjs/

[5] GREEN, Brad a Shyam SESHADRI. *AngularJS*. First edition. Sebastopol: O'Reilly Media, 2013. ISBN 978-144-9344-856.

[6] *AngularJS API Docs* [online]. 2010 [cit. 2013-12-13]. Dostupné z: http://docs.angularjs.org/

[7] MAHEMOFF, Michael. HTML5 vs Native: The Mobile App Debate. In: *HTML5 Rocks* [online]. 2011 [cit. 2013-11-24]. Dostupné z: : http://www.html5rocks.com/en/mobile/nativedebate/

[8] VAN DER MEULEN, Rob a Janessa RIVERA. Gartner Says Smartphone Sales Accounted for 55 Percent of Overall Mobile Phone Sales in Third Quarter of 2013. In: *Gartner* [online]. 2013 [cit. 2013-10-21]. Dostupné z: http://www.gartner.com/newsroom/id/2623415

[9] Titanium Platform Overview. In: *Appcelerator Documentation* [online]. 2008 [cit. 2013-12-06]. Dostupné z: http://docs.appcelerator.com/titanium/3.0/?_escaped_fragment_=/guide/Titanium_Platform_Overview

[10] *TABRIS* [online]. 2008 [cit. 2013-12-26]. Dostupné z: http://developer.eclipsesource.com/tabris/

[11] GIFFORD, Matt. *PhoneGap mobile application development cookbook: over 40 recipes to create mobile applications using the PhoneGap API with examples and clear instructions*. New Edition. Birmingham: Packt Publishing, 2012. ISBN 978-184-9518-581.

[12] The Command-line Interface. In: PhoneGap Documentation [online]. 2013 [cit. 2014-04-26]. Dostupné z: http://docs.phonegap.com/en/3.0.0/guide_cli_index.md.html

[13] SHARP, Remy. What is a Polyfill?. In: Remy sharp's b:log [online]. 2010 [cit. 2014-04-22]. Dostupné z: http://remysharp.com/2010/10/08/what-is-a-polyfill/

[14] SQLite [online]. 2004 [cit. 2014-03-15]. Dostupné z: https://sqlite.org/index.html

[15] Representational State Transfer. In: Wikipedia [online]. 2013 [cit. 2014-05-11]. Dostupné z: http://cs.wikipedia.org/wiki/Representational_State_Transfer

[16] Úvod do JSON. JSON [online]. 2013 [cit. 2014-05-11]. Dostupné z: http://www.json.org/json-cz.html

[17] Cross-Origin Resource Sharing. In: W3C [online]. 2014 [cit. 2014-04-20]. Dostupné z: http://www.w3.org/TR/cors/

[19] Using CORS. In: HTML5 Rocks [online]. 2011 [cit. 2014-04-20]. Dostupné z: http://www.html5rocks.com/en/tutorials/cors/

[20] Google Maps Javascript API v3. Google Developers [online]. 2014 [cit. 2014-05-03]. Dostupné z: https://developers.google.com/maps/documentation/javascript/

[21] Geolocation. In: PhoneGap Documentation [online]. 2013 [cit. 2014-05-04]. Dostupné z: http://docs.phonegap.com/en/3.0.0/cordova_geolocation_geolocation.md.html

[22] Introduction to Android. Android Developers [online]. 2012 [cit. 2014-03-19]. Dostupné z: http://developer.android.com/guide/index.html

[23] Web Storage. In: W3C [online]. 2014 [cit. 2014-04-10]. Dostupné z: http://dev.w3.org/html5/webstorage/

[24] Web SQL Database. In: W3C [online]. 2010 [cit. 2014-04-10]. Dostupné z: http://dev.w3.org/html5/webdatabase/

[25] Storage. In: PhoneGap Documentation [online]. 2013 [cit. 2014-04-10]. Dostupné z: http://docs.phonegap.com/en/2.0.0/cordova_storage_storage.md.html

[26] Promises. In: CommonJS [online]. 2014 [cit. 2014-05-20]. Dostupné z: http://wiki.commonjs.org/wiki/Promises

[27] PhoneGap Development. In: InnovationM: Mobile Technologies [online]. 2014 [cit. 2014-05-20]. Dostupné z: http://www.innovationm.com/phonegap.php

[28] Data Binding. In: AngularJS API Docs [online]. 2010 [cit. 2013-12-13]. Dostupné z: https://docs.angularjs.org/guide/databinding

[29] Adobe® PhoneGap™ Build [online]. 2013 [cit. 2014-05-22]. Dostupné z: https://build.phonegap.com

Seznam obrázků

Seznam tabulek

Příloha 1: Obsah CD

Na přiloženém CD jsou k nalezení následující složky a soubory

- /src – adresář obsahující zdrojové soubory aplikace
- /bin – adresář obsahující zkompilovanou aplikaci
- DP.pdf – text diplomové práce ve formátu pdf
- DP.docx – text diplomové práce ve formátu docx
- Readme.pdf – návod na instalaci aplikace