Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Systém pre sledovanie a evidenciu pohybu motorových vozidiel II

Diplomová práca

2013 **Ladislav Szalai**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Systém pre sledovanie a evidenciu pohybu motorových vozidiel II

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	9.2.1 Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Miroslav Michalko, PhD.
Konzultant:	Ing. Miroslav Michalko, PhD.

Košice 2013 **Ladislav Szalai**

Erráta

Systém pre sledovanie a evidenciu pohybu motorových vozidiel II

Ladislav Szalai

Košice 2013

Abstrakt v SJ

Táto diplomová práca sa zaoberá analýzou, návrhom a implementáciou konkrétnych segmentov monitorovacieho systému pre pohyb vozidiel. Analyzuje vybrané mobilné operačné systémy, ktoré na základe zvolených kritérií opisuje a porovnáva. V ďalších kapitolách využíva priestor pre opis návrhu celkovej architektúry zameranej na služby. Jadro práce prezentuje rozbor statickej časti prostredníctvom pojmu aplikačného rámca a kategorizuje ich do základných typov. Analýza mobilnej časti je venovaná problematike fungovania globálneho systému určenia polohy. Algoritmu výpočtu sférickej vzdialenosti dvoch bodov je venovaná osobitná kapitola, súčasťou ktorej je deskripcia Haversinusovej rovnice. Praktická časť opisuje implementáciu základných pilierov vo webovej, ale aj mobilnej aplikácii podľa charakterizovaných technológií, pričom demonštruje použitie niektorých návrhových vzorov s ukážkami na príkladoch zdrojových textov. Záverečná časť je venovaná verifikácii funkčnosti implementovaného riešenia na základe porovnania kvalitatívnych experimentálne odvodených výsledkov z konkurenčných systémov.

Kľúčové slová

GPS, Monitorovanie pohybu, SOA, Webové služby, Android, Vaadin

Abstrakt v AJ

This diploma thesis deals with analysis, design and implementation of specific segments of vehicle movement monitoring system. Thesis is dedicated to analysis and description of prevalent mobile operating systems, that is based on selected criteria resulting to comprehensive comparison of features and capabilities of individual representatives. In following sections is addict itself to design the service oriented architecture of application. Part of this work specializes in static part of involved information system in terms of application framework and another one is directional to mobile client recording global positioning system. It includes a description of Haversine formula as a consequential equation, giving great-circle distances between two points on a sphere. Practical section exemplifies the fundamental blocks of static and mobile application with demonstration of the software architecture patterns used in implementation. The final section is devoted to verify the functionality of the implemented solutions based on the comparison of experimental results from competitive systems.

Kľúčové slová v AJ

GPS, Movement tracking, SOA, Web services, Android, Vaadin

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
Katedra počítačov a informatiky

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **9.2.1 Informatika**
Študijný program: **Informatika**

Názov práce:

Systém pre sledovanie a evidenciu pohybu motorových vozidiel II.
Car tracking system II.

Študent (tituly, meno, priezvisko): **Bc. Ladislav Szalai**
Školiteľ (tituly, meno, priezvisko): **Ing. Miroslav Michalko, PhD.**
Školiace pracovisko: **Katedra počítačov a informatiky**
Konzultant práce (tituly, meno, priezvisko):
Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:

1. Analyzujte vybrané dostupné mobilné operačné systémy a zvoľte vhodné riešenie pre potreby lokalizácie motorových vozidiel.
2. Navrhnite vhodnú architektúru riešenia lokalizácie vozidiel z pohľadu aplikačnej vrstvy.
3. Na základe definovanej architektúry implementujte navrhnuté riešenie
4. Verifikujte správnosť návrhu a funkčnosť aplikácie.
5. Spracujte zadanie podľa katedrového štandardu.

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 26.04.2013
Dátum zadania diplomovej práce: 31.10.2012

prof. Ing. Ján Kollár, CSc.
vedúci garantujúceho pracoviska

TECHNICKÁ UNIVERZITA V KOŠICIACH FAKULTA ELEKTROTECHNIKY A INFORMATIKY

prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som diplomovú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice 15. 4. 2013 .....

Vlastnoručný podpis

Poďakovanie

Na tomto mieste by som chcel vyjadriť vďaku môjmu vedúcemu diplomovej práce Ing. Miroslavovi Michalkovi, PhD. za odbornú a metodickú pomoc a cenné pripomienky, ktoré mi boli nápomocné pri vypracovaní diplomovej práce.

Predhovor

Súčasný stav spoločnosti smeruje jednoznačným spôsobom k využitiu všetkých moderných zdrojov vo svoj prospech. Organizácie sa snažia používať informačné technológie v každom odvetví a postupne sa znižuje potreba ľudskej pracovnej sily pre základné účely. Tento stav je ľahko vysvetliteľný aj ľudským charakterom, ktorý postupom času rapídne mení svoje hodnoty. Vo všeobecnosti, ľudia sa bez ostychu prikláňajú k rôznym pokryteckým spôsobom, ktorých hlavným cieľom je vyťažiť pre svoj úžitok aj za cenu porušenia predpisov. Zamestnávateľ sa nenechá okrádať svojimi zamestnancami, a tak využíva modernú dobu pre automatizovanie kontrolných činností. Každému je dobre známa situácia ohľadom využívania služobných vozidiel. Niekoľko rokov dozadu boli zamestnanci povinní vypĺňať mesačné výkazy o využívaní vozidiel vo forme knihy jázd. Dnes už sú vo firmách s lepším zázemím vybavené vozidlá GPS modulmi a spravované formou informačných systémov.

Cieľom prezentovanej diplomovej práce je analýza, návrh a implementácia systému pre rozšírenie možností monitorovania vozidiel pre súkromné aj biznis účely. Hlavným cieľom je znížiť náklady pre prevádzkovanie, preto je kladený dôraz na používanie GPS modulov, ktoré sú súčasťou mobilných telefónov. Súčasťou práce je analýza celkového problému pri návrhu systémov podobného charakteru, implementácia statickej serverovej časti v podobne webovej aplikácie spolupracujúcej s mobilnou aplikáciou zabezpečujúcou zber dát a tiež krátky výskum v oblasti technológií pre optimalizovanie implementácie riešení.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam symbolov a skratiek

δ	Delta, písmeno gréckej abecedy označujúce smerodajnú odchýlku
λ	Lambda, písmeno gréckej abecedy v práci označujúce zemepisnú dĺžku
μ	Mí, písmeno gréckej abecedy označujúce aritmetický priemer
ϕ	Fí, písmeno gréckej abecedy v práci označujúce zemepisnú šírku
θ	Théta, písmeno gréckej abecedy v práci označujúce premennú funkcie
\$	Americký dolár, oficiálna mena Spojených štátov amerických
%	Percento, vyjadruje stotinu z celku
km	Kilometer, jednotka dĺžky zodpovedajúca 10^3 metrov

Slovník termínov

ADT (Android Development Tools) - rozšírenie vývojového nástroja Eclipse pre vývoj na platforme Android

AJAX (Asynchronous JavaScript + XML) - kombinácia technológií pre vývoj interaktívnych webových aplikácií

AOP (Aspect-oriented programming) - vývoj aplikácií v podobe menších častí so špecifickou logikou - aspektov

API (Application programming interface) - rozhranie pre programovanie aplikácií

AWT (Abstract Window Toolkit) - nástroj pre vývoj platformovo závislých aplikácií v jazyku Java

CRUD (Create,Read, Update, Delete) - základné operácie perzistentnej vrstvy

DAO (Data Access Object) - objekt vytvárajúci abstraktné rozhranie

EJB (Enterprise JavaBeans) - riadené serverové komponenty pre modulárny vývoj webových aplikácií v jazyku Java

EXIF (Exchangeable Image File Format) - špecifikácia pre formát meta údajov

GNU (GNU's Not Unix) - slobodný operačný systém vo vývoji

GPRS (General Packet Radio Service) - univerzálna paketová rádiová služba

GPS (Global Positioning System) - globálny lokalizačný systém

GWT (Google Web Toolkit) - webový aplikačný framework od spoločnosti Google

HTML (HyperText Markup Language) - štandardizovaný značkovací jazyk pre vytváranie webových stránok pre prostredie internetu

HTTP (Hypertext Transfer Protocol) - internetový protokol pre výmenu údajov

IOC (Inversion of Control) - návrhový vzor pre obrátené riadenie

JavaScript - multiplatformový interpretovaný skriptovací programovací jazyk

JDBC (Java DataBase Connectivity) - rozhranie pre prístup k relačnej databáze

JSON (JavaScript Object Notation) - platformovo nezávislý formát pre zápis dát

Middleware - počítačový softvér prepájajúci komponenty medzi sebou

MVC (Model View Controller) - návrhový vzor pre viac-vrstvové aplikácie

ORM (Object-relational mapping) - konverzia dát z údajov relačnej databázy na objekty programovacieho jazyka

PDA (Personal Digital Assistant) - malý výkonný vreckový počítač, často vybavený dotykovou obrazovkou a sadou užitočných funkcií

PID (Process IDentifier) - unikátny identifikátor procesov v systéme

REST (Representational State Transfer) - architektúra rozhraní navrhnutá pre distribuované prostredia

SIM (Subscriber Identity Module) - identifikačná karta pre použitie v mobilných telefónnych sieťach

Smartfón - inteligentný mobilný telefón s pokročilým operačným systémom

SOA (Service Oriented Architecture) - architektúra založená na nezávislých komponentoch poskytujúcich služby

SOAP (Simple Object Access Protocol) - protokol pre výmenu správ založených na XML formátoch

Startup - novo založená firma alebo nádejný projekt podporený investorom spojený s komercializáciou produktov vyvinutých v predchádzajúcej fáze

SWT (Standard Widget Toolkit) - knižnica zabezpečujúca nástroj so sadou grafických prvkov pre vývoj v jazyku Java

XAML (Extensible Application Markup Language) - jazyk pre popis grafických rozhraní v aplikáciách platformy .NET

XML (eXtensible Markup Language) - štandardizovaný značkovací jazyk

URI (Uniform Resource Identifier) - jednotný identifikátor zdrojov

Úvod

Dnešná spoločnosť sa vyznačuje prudkým rozmachom z hľadiska informačných technológií. Mnoho pracovných pozícií si vyžaduje agilný prístup zamestnancov, ktorým práve počítače a iné výdobytky modernej technológie pomáhajú zvyšovať efektivitu práce. Takýto progres samozrejme nezabezpečuje len samotný hardvér. Podstatnú úlohu v ňom zohráva programové vybavenie, ktoré je schopné dané technické vybavenie efektívne využívať.

Samotný softvér rozlišujeme na systémový a pre nás viac podstatný aplikačný. Z kategórie aplikačného softvéru sa zameriame na webové aplikácie, ktoré predstavujú základ dnešnej internetovej spoločnosti. Takéto interaktívne aplikácie spolu s webovými službami sú doplnené o moderné a intuitívne grafické rozhrania a predstavujú základ komplexných systémov existujúcich v prostredí internetu. Súčasný trend vývoja aplikácií sa orientuje na webové koncepcie, ktoré postupne poukazujú na desktopové systémy ako na zastaralé a neefektívne. Ako hlavnú nevýhodu označujú platformovú závislosť. Pre túto prácu sú druhou kategóriou mobilné aplikácie. K aktuálnej dobe evidujeme niekoľko významných predstaviteľov mobilných operačných systémov. Každý z nich sa vyznačuje špecifickými vlastnosťami a snažia sa o jedinečnosť v každom smere. Treba však povedať, že v poslednej dobe ide skôr o kopírovanie nápadov a využívanie patentových sporov v konkurenčnom boji, ako o implementáciu nových sofistikovaných prvkov.

Zámerom práce je využitie týchto softvérových kategórií pre návrh aplikácie, ktorá bude schopná monitorovať pohyb vozidiel. Kľúčovým prvkom je určovanie polohy. Najpopulárnejším spôsobom pre monitorovanie polohy je využívanie satelitnej navigácie, ktorá je implementovaná v podobe globálneho pozičného systému. Je zaujímavá rovnako z pohľadu vývojára, ako aj z pohľadu koncového používateľa, najmä kvôli relatívne nízkej finančnej náročnosti.

Prvá kapitola práce stručne popisuje formuláciu zadanej úlohy, demonštruje ciele, približuje tému a abstraktne popisuje podstatu práce. Súčasťou kapitoly je postupnosť krokov, pomocou ktorých bola daná práca realizovaná.

Úvod druhej kapitoly predstavuje stručnú charakteristiku mobilných technológií. V ďalších krokoch charakterizuje vybrané mobilné operačné systémy z dôležitých hľadísk pre ich použitie vo vývoji vlastných aplikácii na danej platforme. Kapitola je zameraná hlavne na ich architektonickú skladbu a prostriedky potrebné pre samotný vývoj.

Tretia kapitola obsahuje samotný návrh informačného systému. Návrh je dekomponovaný na serverovú časť, t.j. statického klienta aplikácie a mobilnú časť informačného systému. V prvej časti je navrhnutá architektúra orientovaná na služby a stručne opísaná problematika a prostriedky takejto architektúry v spolupráci s aplikačnými rámcami. Druhá časť je venovaná problematike určovania polohy prostredníctvom GPS a spôsobom lokalizácie.

Štvrtá kapitola zhŕňa implementáciu podľa navrhnutej architektúry v predchádzajúcej kapitole. V úvodnej časti stručne charakterizuje technológie, ktoré boli v rámci implementácie použité. V nasledujúcich sekciách názorne opisujú implementáciu vybraných segmentov informačného systému na príkladoch zdrojových textov.

Piata kapitola je venovaná overeniu implementovaného riešenia. V tejto časti je verifikovaný konkrétny algoritmus pre výpočet vzdialenosti na základe výsledkov z konkurenčných systémov. Kapitola obsahuje aj komplexné overenie funkcionality mobilnej aplikácie.

Poslednú časť tvorí kapitola, ktorá sumarizuje prácu. Jej obsahom je zhrnutie vyplývajúce z predošlých kapitol, implementačné nedostatky a návrhy v podobe možných rozšírení.

1 Formulácia úlohy

Cieľom diplomovej práce je analyzovať vybrané dostupné operačné systémy pre inteligentné mobilné telefóny a na základe kvalitatívneho porovnania a nadobudnutých poznatkov zvoliť konkrétnu platformu, prostredníctvom ktorej je možné navrhnúť a implementovať vhodné riešenie zaoberajúce sa monitorovaním polohy vozidiel. Výsledkom má byť informačný systém, zahrňujúci webovú aplikáciu s vhodne navrhnutou architektúrou, ktorá bude prostredníctvom preddefinovaných komunikačných protokolov schopná spolupracovať s mobilnou aplikáciou, predstavujúcou tenkého klienta diskutovaného riešenia. Podstatu riešenia predstavuje zber údajov pomocou globálneho systému pre určenie polohy. Tieto údaje sa lokálne ukladajú v mobilnej aplikácii a v ďalšom procese spracovania sa vyhodnocujú pre potreby získania štatistických údajov ako súčasť webovej aplikácie, kde sú následne vykreslené ako trasa na mapovom podklade.
Pre dosiahnutie definovaného cieľa je potrebné dodržať postupnosť krokov:

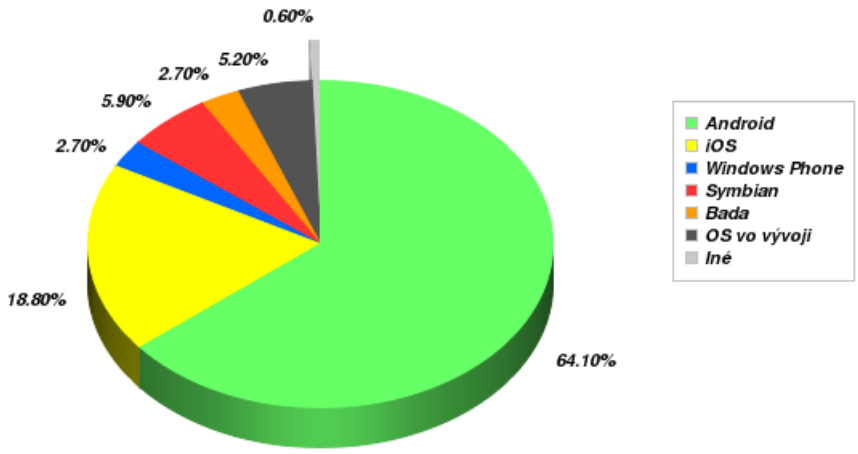
- Analyzovať vybrané mobilné operačné systémy a zamerať sa na segmenty potrebné pri implementácii monitorovania pohybu.
- Z nadobudnutých vedomostí ohľadom mobilných operačných systémov a technológií zvoliť vhodného predstaviteľa.
- Navrhnúť vhodnú architektúru orientovanú na služby, ktoré budú využívané mobilným zariadením.
- Z pohľadu aplikačnej vrstvy navrhnúť aplikačný rámec, ktorý bude reprezentovať funkcionalitu deklarovanú vo webovej aplikácii.
- Naštudovať problematiku týkajúcu sa určenia polohy prostredníctvom GPS.
- Verifikovať riešenie formou dostupných systémov zaoberajúcich sa monitorovaním polohy.

2 Mobilné platformy

Súčasná spoločnosť zaznamenáva progresívny rozvoj mobilných technológií. Aktuálnym trendom sú predovšetkým prenosné zariadenia, ktoré sú postupom času deklarované pre čoraz konkrétnejšie funkcionality. Tieto zariadenia predstavujú bránu do sveta nových možností.

Viaceré smery priemyslu, obchodu či služieb to vnímajú ako ďalší spôsob reprezentácie svojho produktu pred zákazníkom. Ľudia používajú smartfóny ako neodmysliteľnú súčasť svojho života. Elektronická pošta, bankové transakcie, pripojenie na internet, to všetko je možné.

Výkon zariadení je každým dňom posunutý o nejakú úroveň a tento jav je spôsobený jednak vývojom nových hardvérových komponentov, ale taktiež príslušným softvérom. Podobne ako výpočtové jednotky, aj tieto zariadenia obsahujú vo svojej softvérovej výbave operačný systém s príslušným balíkom kompatibilných aplikácií. Práve tento smer vyprodukoval aktuálne najväčší konkurenčný boj v informačných technológiách. Podiel jednotlivých mobilných operačných systémov je znázornený na obrázku2 – 1



Obr. 2 – 1 Grafické zobrazenie podielu mobilných OS vo svete[1]

Táto kapitola je venovaná porovnaniu vybraných mobilných operačných systémov z viacerých hľadísk. Primárne je zameraná na charakteristické črty pre vývoj aplikácií.

2.1 Android

Systém Android vznikol ako startup projekt vyvíjaný niekoľkými predstaviteľmi pôvodnej Kalifornskej spoločnosti Android Inc. Táto spoločnosť bola neskôr kúpená gigantom v oblasti informačných technológií, firmou Google. Od nich sa očakávalo vydanie prvého smartfónu a jednou z možností bolo prefinancovať sľubne rozbehnutý program, z ktorého sa prostredníctvom združenia vedúcich mobilných operátorov a výrobcov mobilných zariadení vedúceho k neziskovej organizácii Open Handset Alliance, vyvinula open source platforma Android[2].

Tento podnet pri vzniku bol zdá sa kľúčovým faktorom pre dosiahnutie prvenstva v zastúpení mobilných zariadení s operačným systémom Android. Samotný systém je uvoľnený s licenciou predstavujúcou slobodný softvér, konkrétne Apache Software License 2. Tejto licencii podlieha modul jadra pre operačným systém, ktorého pôvod je v základnom Linuxovom jadre strážiac si licenciu GNU, programové rozhrania, základné knižnice a niektoré zdrojové aplikácie. Tento jav zvyšuje použiteľnosť tohoto systému a jeho portabilitu pre rôzne zariadenia ako PDA, navigačné prístroje, satelity alebo televízory. Z toho plynú výhody jednoduchej synchronizácie viacerých zariadení, či už v domácnosti alebo rozvetvených spoločnostiach v prospech tejto platformy.

Každé pre však má svoje proti a výnimkou nie je ani analyzovaný systém. K súčasnému stavu evidujeme množstvo verzií zdokumentovaných v tabuľke 2 – 1. Tento jav si vyžaduje zvýšenú potrebu optimalizácie jednotlivých verzií pre konkrétne zariadenia. Väčšina výrobcov preto vydáva rôzne nadstavby základného systému, čo

je v súčasnosti viac na škodu ako k úžitku.

Tabuľka 2 – 1 Prehľad verzií systému Android[3]

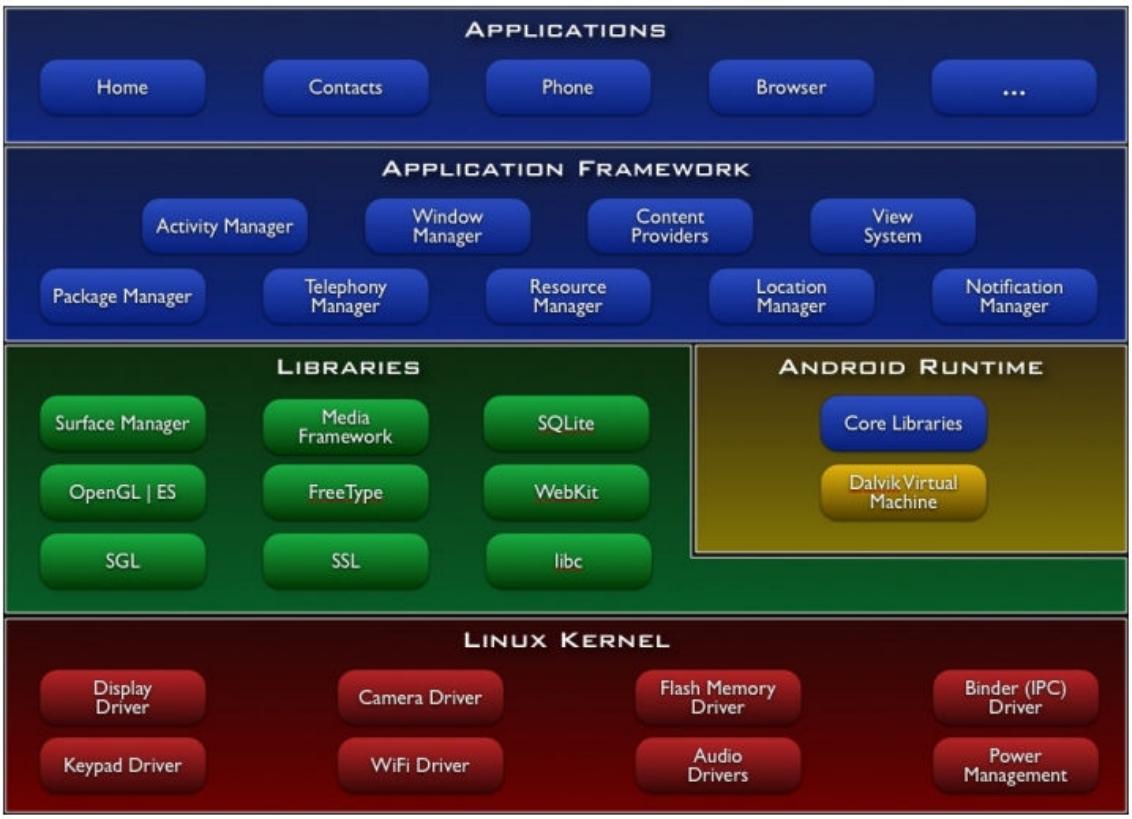
Verzia	Názov	Dátum vydania	Verzia API
Android 1.0	Base	23.september 2008	1
Android 1.1	Petit Four	9.február 2009	2
Android 1.5	CupCake	30.apríl 2009	3
Android 1.6	Donut	15.september 2009	4
Android 2.0	Eclair	26.október 2009	5
Android 2.01	Eclair	3.december 2009	6
Android 2.1	Eclair	12.január 2010	7
Android 2.2.x	Froyo	20.máj 2010	8
Android 2.3-2.3.2	GingerBread	6.december 2010	9
Android 2.3.3-2.3.7	GingerBread	9.február 2011	10
Android 3.0	Honeycomb	22.február 2011	11
Android 3.1	Honeycomb	10.máj 2011	12
Android 3.2	Honeycomb	15.júl 2011	13
Android 4.0-4.0.2	Ice Cream Sandwich	19.október 2011	14
Android 4.0.3-4.0.4	Ice Cream Sandwich	16.december 2011	15
Android 4.1	Jelly Bean	27.jún 2012	16
Android 4.2	Jelly Bean	29.október 2012	17

2.1.1 Architektúra

Android samotný v skutočnosti tvorí operačný systém, založený hlavne nad linuxovým jadrom, middleware-om zabezpečujúci interoperabilitu v rámci poskytovaných služieb cielene zameraných na distribuované architektúry a sadou aplikácií nevyhnutnými pre uspokojenie základných potrieb používateľa. V prístupe k architektúre sa nerozlišuje medzi základnými vstavanými aplikáciami a aplikáciami tretích strán. Zo

skutočnosti vyplýva, že platforma je otvorená používateľovým požiadavkám v každom smere.

Štruktúra architektúry platformy je rozdelená na 5 samostatných, ale úzko súvisiacich vrstiev. Každá táto vrstva má jednoznačne deklarovanú svoju funkcionalitu. Sú identifikovateľné na obrázku2 – 2, ktorý okrem iného znázorňuje nástroje a komponenty jednotlivých vrstiev. Nasledujúci odstavec podrobnejšie charakterizuje jednotlivé vrstvy.



Obr. 2 – 2 Grafická notácia architektúry Android platformy[4]

Jadro - Báza celého systému je postavená na linuxovom jadre verzie 2.6 vrátane licencie GNU, ktorej podlieha. Toto jadro predstavuje vrstvu abstraktnej povahy medzi hlavnými hardvérovými prostriedkami a prostriedkami zabezpečujúcimi softvérové požiadavky. Ich cieľom je teda zabezpečovať základné systémové služby, ktoré zahrňujú administráciu procesov, pamäťový manažment,

komunikáciu sieťových prostriedkov a v neposlednom rade dodržiavanie princípov bezpečnosti. Jednotlivé aplikácie ale pristupujú k funkciám poskytovaným týmto jadrom prostredníctvom rozhrania Android API.

Keďže vychádzame z linuxovej základnej bázy, každý proces, ktorý je spravovaný týmto jadrom musí byť jednoznačne identifikovaný svojim PID identifikátorom. Tieto identifikátory sú jednotlivým procesom prideľované taktiež najnižšou vrstvou architektúry a zabezpečujú paralelné vykonávanie viacerých procesov bez toho, aby boli ovplyvňované, čím dodržujú základný princíp bezpečného systému. Súčasťou jadra sú tiež všetky potrebné ovládače jednotlivých modulov pre využívanie hardvérových prvkov zariadení.

Knižnice - Nasledujúcu vrstvu predstavuje sada systémových knižníc, ktoré sú prevažne napísané v programovacom jazyku C/C++. K týmto knižniciam aplikácie rovnako pristupujú prostredníctvom aplikačného rozhrania a môžu zabezpečovať prístup ku ktorémukoľvek komponentu v rámci architektúry. Základné systémové knižnice podľa obrázku2 – 2 predstavujú:

- Surface Manager - zabezpečuje prístup k obrazovému podsystému a vytvára skladbu grafiky z viacerých častí pre dvojrozmernú aj trojrozmernú reprezentáciu.
- OpenGL ES - knižnica, ktorá využíva hardvérovú alebo softvérovú akceleráciu s vysokou optimalizáciou pre rezprezentáciu trojrozmernej grafickej podoby.
- SGL - základný riadiaci prostriedok dvojrozmernej grafiky.
- MediaFramework - knižnica poskytujúca rozhranie pre multimédia. Zahŕňa nahrávanie a prehrávanie štandardných multimediálnych zvukových aj video formátov. Využíva najmä OpenCore a PocketVideo.

- FreeType - rozhranie pre vykresľovanie vektorových a bitmapových písmen.
- SSL - zachováva princípy bezpečnosti v komunikačných prostriedkoch vhodnými šifrovacími algoritmami.
- SQLite - jednoduchým spôsobom implementovaná podpora relačnej databázy, optimalizovaná pre prenosné zariadenia na princípe vytvárania súborov štrukturovaného dotazovacieho jazyka.
- Webkit - jadro pre moderný prehliadač na báze typu WebKit.
- libc - štandardná systémová knižnica jazyku C s upravenou implementáciou pre prenosné zariadenia.

Android Runtime - Základ tejto vrstvy tvorí virtuálny stroj Dalvik so sadou potrebných knižníc, s funkcionalitou obsiahnutou v základných Java knižniciach, pre jeho korektné fungovanie. Ako už bolo spomenuté, každá aplikácia má pridelený proces jadrom a taktiež vlastnou inštanciou pre virtuálny stroj Dalvik. Jedným zo základných princípov systému Android je nezávislosť aplikácií na hardvérovom vybavení a taktiež na sade inštrukcií pre konkrétny procesor zariadenia. Práve táto vlastnosť je dosahovaná prostredníctvom virtuálneho stroja Dalvik.

Predstavuje virtualizovaný procesor so špeciálne navrhnutými sadami inštrukčného toku. Princíp fungovania je podobný prekladu zdrojových kódov vyšších programovacích jazykov do strojových inštrukcií, pričom tu sa prekladajú programy do inštrukcií virtuálneho stroja, ktorým sú potom pri behu programu interpretované[5]. Tento princíp odlišuje Android v hlavnej miere od ostatných platforiem pre mobilné zariadenia. Dalvik je navyše navrhnutý tak, aby bol čo najviac optimalizovaný pre nízkoúrovňové aktivity zahrňujúce napríklad vlákna a správu pamäte.

Aplikačné rozhranie - Nad vrstvou obsahujúcou virtuálny stroj s dynamicky odkazovanými knižnicami sa nachádza aplikačný rámec celého systému Android. Framework je napísaný v programovacom jazyku Java a z väčšej miery obsahuje rozhrania aplikované na konkrétne triedy vytvárajúce základ pre tvorbu aplikácií vyššej úrovne abstrakcie. Súčasťou rozhrania sú v niektorých prípadoch len zapuzdrené natívne volania knižníc nižších vrstiev. Prvky tejto vrstvy sú rovnako prekladané do inštrukcií spracovateľných virtuálnym strojom Dalvik[6].

Vrstva aplikačného rozhrania je nastavená metódou, ktorá umožňuje jednoduchým spôsobom znovupoužitie všetkých dostupných komponentov v rámci architektúry vo viacerých paralelných vláknach behom vykonávania, pri dodržiavaní všetkých bezpečnostných pravidiel[7]. Pri vývoji umožňuje programátorovi prístup k zdrojovým súborom a do určitej miery aj prekrývanie jednotlivých metód pri využití prostriedkov dedičnosti. Aplikačný rámec zahŕňa komponenty ako:

- Views - predstavuje jednotlivé subkomponenty, ktoré sú súčasťou View system komponenty a umožňujú vytvárať a spravovať grafické používateľské rozhrania aplikácií.
- Content provider - zaisťuje prístup k dátam, či už vlastným alebo k externým z iných aplikácií.
- Resource Manager - je zaopatrený funkciou obstarávania zdrojov z externých konfiguračných súborov, čo v sebe zahŕňa grafické podklady, slovníkové výrazy alebo rôzne konštanty.
- Activity Manager - zo sady manažérov v tomto rozhraní zaisťuje najdôležitejšiu funkciu, čím je správa životného cyklu celej aplikácie. V rámci jednej aplikácie môže byť použitých viacero takýchto manažérov špecifi-

kovaných pre konkrétne činnosti.

- Notification Manager - vyvolávanie a správa upozornení z celého systému.

Aplikácie - Treba si uvedomiť, že Android nepredstavuje len operačný systém pre hardvér. Je distribuovaný súčasne so sadou aplikácií, ktoré zabezpečujú základné funkcie pre používanie konkrétneho zariadenia. Tieto aplikácie zabezpečujú napríklad manažovanie kontaktov a správ, funkcionalitu kalendára, rôzne prehrávače multimédií, prehliadač alebo navigáciu. Predvolené aplikácie je možné neskôr podľa používateľových potrieb nahradiť, ale vo väčšine prípadov ich výrobcovia zariadení neumožňujú dodatočne vymazať. Je to dané tým, že systém poskytuje obnovenie továrenských nastavení zariadenia a pri vymazaní by došlo k narušeniu referencií pre predvolené aplikácie.

2.1.2 Vývojové prostredie

Vývoj aplikácií pre platformu Android nie je striktne obmedzený predvoleným vývojovým nástrojom. Existuje niekoľko variant s tým, že sú dostupné prostredia pre všetky existujúce platformy, a teda neobmedzujú vývojára konkrétnym operačným systémom.

Organizácia stojaca za vznikom operačného systému Android odporúča použitie voľne dostupného vývojového prostredia Eclipse v spolupráci s rozšírením ADT, ktoré je potrebné do vývojového prostredia nainštalovať. Toto rozšírenie je prezentované ako oficiálne vyvíjaný a podporovaný produkt korporáciou Google. Samozrejme existuje viacero vývojových nástrojov ako vhodná alternatíva. Z tých známejších to je vývojármi veľmi obľúbené prostredie Netbeans s neoficiálne podporovaným rozšírením NBAndroid. Ďalšou možnosťou je integrované vývojové prostredie IntelliJ Idea, za ktorým stojí spoločnosť JetBrains. V tomto prípade sa už ale nejedná o voľne dostupný softvér v prípade, že chceme využiť plnú funkcionalitu, ktorú poskytuje.

Tá však obsahuje integráciu vývojových prvkov pre Android už po nainštalovaní a navyše na vysokej úrovni. Možnosť predstavuje aj vývoj prostredníctvom akéhokoľvek textového editora a následný preklad zdrojových kódov prostredníctvom príkazového riadku, využitím nástrojov popísaných v nasledujúcom odseku.

Základ vývoja aplikácií pre Android ale spočíva v balíku vývojárskych nástrojov nazývaných tiež Android Software Development Kit. Je rovnako dostupný pre všetky platformy a je potrebné ho nainštalovať a integrovať do vyššie opísaných vývojárskych prostredí v spolupráci s vhodnými zásuvnými modulmi. Tento balík pozostáva zo základného prostredia pre vývoj, nástroja pre ladenie programov, emulátora prostredia, dokumentácie, sady základných systémových knižníc a tiež vzorových zdrojových kódov.

2.2 iOS

Operačný systém iOS je jedným z pilierov úspešnosti medzinárodnej spoločnosti Apple. Pôvod tohoto operačného systému vychádza z jeho predchodcu nazývaného iPhone OS, ktorého prvá verzia bola zverejnená v roku 2007 príchodom prvých zariadení z kategórie inteligentných telefónov a prenosných zariadení od spoločnosti Apple.

Pomenovanie iOS vzniklo až vydaním 4. verzie operačného systému. Cieľom bolo zachovať organizačnú politiku spoločnosti, ktorá všetky svoje produkty z danej kategórie pomenovala rovnakým prefixom. Tento operačný systém bol primárne určený pre mobilné telefóny, no neskôr bol distribuovaný aj na ďalšie mobilné zariadenia, konkrétne tablety, hudobné prehrávače a podľa posledných zdrojov sa používa aj v televízoroch. Nie je tajomstvom, že spoločnosť Apple nepovoľuje distribúciu svojich produktov pre iných výrobcov. Rovnakou politikou sa riadia aj v prípade softvéru, ktorý môže byť použitý len na špecifickom licencovanom hardvéri svojej

výroby. Toto je možné považovať za hlavný znak odlišnosti oproti konkurencii, ktorú tvoria najmä spoločnosť Google so svojim systémom Android a Microsoft s vývojom Windows Phone.

Na stranu iOS prináša výhodu fakt, že softvér je prispôsobený hardvéru, čo ich úplne odlišuje od konkurencie, ktorá sa skôr zameriava na optimalizovanie hardvérových prvkov pre hotový operačný systém. Keďže licenčne vyhovujú len zariadenia od spoločnosti Apple, môžu optimalizáciu posunúť na vyššiu úroveň. Majú k dispozícii presné hardvérové špecifikácie, na ktoré vyvíjajú daný systém, čo vedie k stabilnejšiemu a sofistikovanejšiemu softvérovému produktu. Navyše iOS predstavuje uzatvorený operačný systém, preto nemusia riešiť kompatibilitu aplikácií 3.strán. Z toho vyplýva, že tieto zariadenia sú po finančnej stránke drahšie ako konkurenčné výrobky.

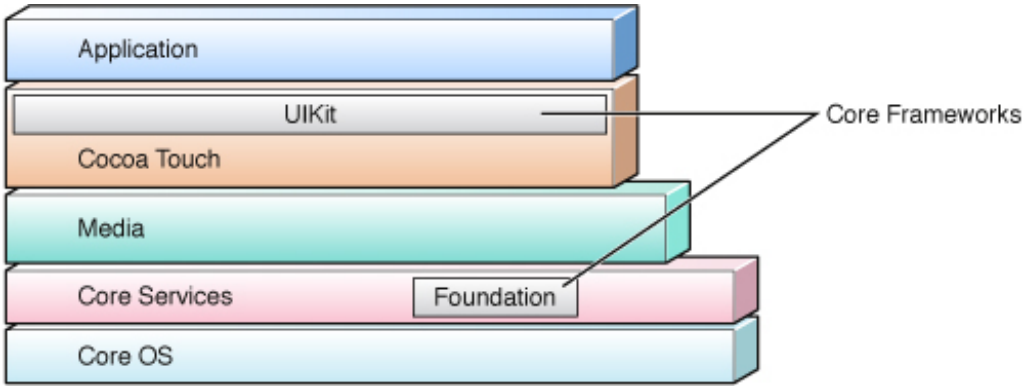
Prehľad verzií operačného systému iOS používaných v súčasnosti v spolupráci s konkrétnymi zariadeniami, je znázornený v tabuľke 2 – 2.

Tabuľka 2 – 2 Prehľad súčasných verzií systému iOS [8]

Verzia	**Dátum vydania**	**Zariadenia**	**Stav vývoja**
iOS 3.1.3	2.február 2010	iPhone, iPod Touch (1.gen)	prerušený
iOS 4.2.1	22.november 2012	iPhone 3G, iPod Touch (2.gen)	prerušený
iOS 5.1.1	7.máj 2012	iPod Touch (3.gen), iPad	prerušený
iOS 6.1	28.január 2013	iPhone 3GS, iPhone 4, iPhone 5, iPod Touch (4.a 5.gen), iPad 2, iPad 3, iPad 4, iPad Mini	aktuálny
iOS 6.1.1	11.február2013	iPhone 4S	aktuálny

2.2.1 Architektúra

Architektúra operačného systému iOS je zložená podobne ako v prípade konkurencie z niekoľkých špecifických vrstiev. Hierarchia týchto vrstiev je zobrazená na obrázku 2 – 3. Samotná architektúra predstavuje modifikovanú a zjednodušenú verziu operačného systému MacOS, ktorý spoločnosť Apple používa pre svoje stolové počítače. Z dôvodu, že cieľovým produktom sú zariadenia odlišného charakteru, boli vynechané niektoré jeho podstatné prvky. Zároveň bol iOS doplnený ovládacími prvkami pre podporu dotykového ovládania alebo tiež o akcelerometer, ktorý sa v stolových počítačoch nevyužíva. Samotné jadro je odvodené z unixového jadra Mach.



Obr. 2 – 3 Grafická notácia architektúry iOS[9]

Vrstva jadra operačného systému - predstavuje nosnú vrstvu celej hierarchie. Poskytuje funkcie nízko-úrovňovej abstrakcie, využívané ostatnými vrstvami. Súčasťou tejto vrstvy sú prvky ako jadro, súborový systém, sieťová infraštruktúra, bezpečnostné prvky, správa napájania a niekoľko ovládačov. Špecifickým prvkom tejto vrstvy je knižnica libSystem, ktorá podporuje a zahŕňa špecifikácie pre jednotlivé úrovne rozhraní pre programovanie aplikácií.

Vrstva služieb poskytovaných jadrom - jedná sa o fundamentálne služby, poskytujúce aplikácie v rámci systému a aplikačné rámce, slúžiace okrem iného ako základné služby pre prácu s prvkami systému iOS. Niektoré časti systému

ich nevyužívajú priamo, ale v skutočnosti sú na nich podstatne závislé. Ako príklad kľúčových služieb je možné uviesť automatické počítadlo referencií, ochranu dát, podporu pre zdieľanie súborov, platobný systém v rámci aplikácií, databázovú podporu prevádzkovanú formou knižnice SQLite alebo podporu pre prácu so štrukturovanými súbormi typu XML, prostredníctvom triedy NSXMLParser, využívajúc pri tom prvky knižnice libXML2. Medzi najnovšie rozšírenia týchto služieb patrí integrácia pre iCloud, čo predstavuje poskytovanie služieb a dátových úložísk prostredníctvom internetu.

Táto vrstva zároveň poskytuje aplikačné rámce napríklad pre správu účtov používateľa, telefónny zoznam, lokalizačné rozhranie, rozpoznávanie pohybov, systémovú konfiguráciu a mnoho ďalších.

Vrstva médií - vrstva je silne závislá na predošlej, konkrétne vrstve služieb, ktoré poskytuje samotné jadro. Poskytuje grafické a multimediálne služby pre vyššiu vrstvu Cocoa Touch layer, ktorá ich sprostredkúva. Vrstva médií teda poskytuje priestor pre vytváranie zaujímavých grafických a zvukových riešení. Táto vrstva je dekomponovaná na:

Grafické technológie

- Core Graphics - podporuje grafické dvoj-dimenzionálne zobrazovanie.
- Core Text - rozhranie, ktoré zabezpečuje zobrazovacie prostriedky pre vykresľovanie a umiestňovanie písmen a reťazcov.
- OpenGL ES - rozhranie postavené na výkonných grafických knižniciach v jazyku C, slúžiace pre vykresľovanie 2D a 3D objektov.
- Core animation - je súčasťou rozhrania Quartz a jeho úlohou je správa pokročilých animačných prvkov v multimediálnom prostredí.

Zvukové technológie

- Media player - predstavuje jednoduchý prístup ku zložke medií iTunes a zároveň poskytuje prehrávač pre podporované médiá.
- AV foundation - rozhranie pre prehrávanie a nahrávanie audio formátov.
- Core audio - kompletný balík sofistikovaných rozhraní pre manažovanie všetkých zvukových prostriedkov.

Video technológie Rozhranie umožňuje nahrávanie a prehrávanie video formátov a ich následné použitie v rámci aplikácií. iOS poskytuje niekoľko technológií pre prácu s obsahom tohoto formátu, ktoré sú závislé od príslušného hardvérového vybavenia. Základ tejto vrstvy je rozhranie Core media.

Vrstva Cocoa Touch - vrstva produkujúca kľúčové aplikačné rámce pre vývoj aplikácií pre operačný systém iOS. Definuje infraštruktúru pre implementáciu grafického rozhrania aplikácie a interakcie s používateľom. Poskytuje vysokoúrovňové systémové služby. Pôvod vrstvy vychádza z vrstvy Cocoa, ktorá bola zdedená z operačného systému MacOS a predstavuje najsofistikovanejšiu aplikačnú vrstvu vôbec. Postavená je na interakcii prostredníctvom klávesnice a myši, preto bolo potrebné abstrahovať novú vrstvu a implementovať v nej ovládanie dotykom. Medzi jej základné služby patria:

- Multitasking - podpora behu viacerých programových prostriedkov, ktoré sú súčasťou aplikácií.
- Ochrana dát - údaje, ktoré sú ukladané jednotlivými aplikáciami za behu, sa označujú ako chránené a pred uložením sú zašifrované.
- Notifikácie - rôzne typy upozornení. Môže sa jednať o ”push”notifikácie, ktoré upozorňujú používateľa krátkymi správami o nových informáciách alebo stave jednotlivých aplikácií. Druhým typom sú lokálne notifikácie,

ktoré podporujú len novšie verzie iOS. Tieto sa ukladajú lokálne a môžu bežať v aplikáciách na pozadí alebo môžu predstavovať notifikácie uložené v systéme, ktoré sú naplánované na určitý čas.

- Identifikácia gest - v nižších verziách bolo potrebné rozpoznávanie riešiť manuálne, ale novšie verzie to robia systémovo a automaticky ich predávajú konkrétnym aplikáciám.

- Zdieľanie súborov - je možné prostredníctvom rozhrania iTunes pre definované typy. Vyriešená je aj podpora zdieľania protokolom peer-to-peer prostredníctvom bluetooth pripojenia.

- Zobrazenia na externom zariadení - je možné len prostredníctvom prenosového káblu.

2.2.2 Vývojové prostredie

Operačný systém iOS predstavuje najviac uzavretú platformu pre mobilné zariadenia. Ako bolo spomenuté, je možné ho používať len na mobilných zariadeniach vyrobených spoločnosťou Apple. Inštalácia aplikácií do týchto zariadení je možná len prostredníctvom ich vlastného obchodu, pričom každá aplikácia musí prejsť zložitým schvaľovacím procesom po jej distribuovaní.

Vývoj pre túto platformu a následné testovanie na konkrétnych zariadeniach je spoplatnený členským príspevkom $99 na rok. Príspevok zahrňuje pre používateľa vývojársky účet potrebný pre následný vývoj aplikácií na opisovanej platforme. Samotný balík vývojárskych nástrojov je spoločnosťou Apple poskytovaný zdarma. Je obmedzené ho používať len na operačnom systéme MacOS. Balík obsahuje emulátory jednotlivých zariadení a tiež potrebné knižnice pre vývoj. K vývojárskemu účtu je poskytované prostredie pre vývoj XCode, podporované výrobcom.[10] Existuje niekoľko alternatívnych prostredí, no v žiadnom z nich nie je možné vyvíjať bez

použitia primárneho prostredia XCode, kvôli kompilátoru. Programovacím jazykom pre túto platformu je primárne jazyk ObjectiveC, ale niektoré pôvodné nosné časti sú programované v jazyku C.

2.3 Windows Phone

Windows Phone predstavuje jednu z najmladších mobilných platforiem súčasnosti. Za jeho vývojom stojí ďalší gigant informačných technológií, spoločnosť Microsoft. Windows Phone bol uvedený na trh v roku 2010 na svetovom kongrese mobilných zariadení v Barcelone pre používanie v Európe, Amerike a Austrálii. O rok neskôr sa ho dočkali aj v Ázii.[11]

Cieľom vývoja operačného systému bolo naviazať na niekdajší úspech jeho predchodcu vyvíjaného Microsoftom, čím je operačný systém Windows Mobile. Platforma Windows Mobile predstavovala v dobe príchodu prvých inteligentných telefónov najrozšírenejšie spojenie hardvéru novej technológie s príslušným softvérovým vybavením. Napriek tomu sa výrobca k novej technológii postavil úplne odlišne ako v prípade predchodcu, či už po stránke technologickej alebo marketingovej. Kým Windows Mobile bol zameraný hlavne pre zariadenia reprezentujúce vyššiu úroveň zamestnancov v spoločnostiach, Windows Phone je predstaviteľom komerčnej sféry, čo ale nijak neznižuje jeho schopnosti, práve naopak. Windows Phone predstavuje skutočnú revolúciu, pričom sa nenechal ovplyvniť žiadnym iným systémom[12]. Nevýhodou novej verzie je spätná nekompatibilita, keďže aplikácie zo starších verzií sú nepoužiteľné.

Oproti Android platforme je podstatný rozdiel aj v spôsobe distribúcie samotnej platformy. Android je uvoľnený pod open-source licenciou, zatiaľ čo Windows Phone predstavuje uzatvorený systém do najvyššej možnej miery, čo sa z určitého hľadiska môže považovať za nevýhodu. Ak pri príchode novej verzie v Androide dôjde k obja-

veniu neočakávaného správania zo strany systému, často je to diskutované v rôznych komunitách zaoberajúcich sa modifikáciami systému. Takýmto spôsobom je chyba rýchlo opravená a používateľ si ju môže opraviť softvérom tretích strán alebo počká na oficiálnu záplatu od výrobcu, ktorá veľakrát predstavuje práve použitie implementácie zdrojových kódov zo spomínaných komunít. Avšak pre Windows Phone je realizácia podobných úprav nemožná z dôvodu neprístupných zdrojových kódov systému. Jediným možným riešením je vyčkať na oficiálnu záplatu.

V prospech tohoto systému hovorí relatívne nízky počet verzií, s čím si zachováva určitú integritu a konzistenciu voči zariadeniam s hardvérom definovaným pre túto platformu. V tabuľke 2 – 3 sú zdokumentované vydané verzie platformy Windows Phone.

Tabuľka 2 – 3 Prehľad verzií systému Windows Phone[13]

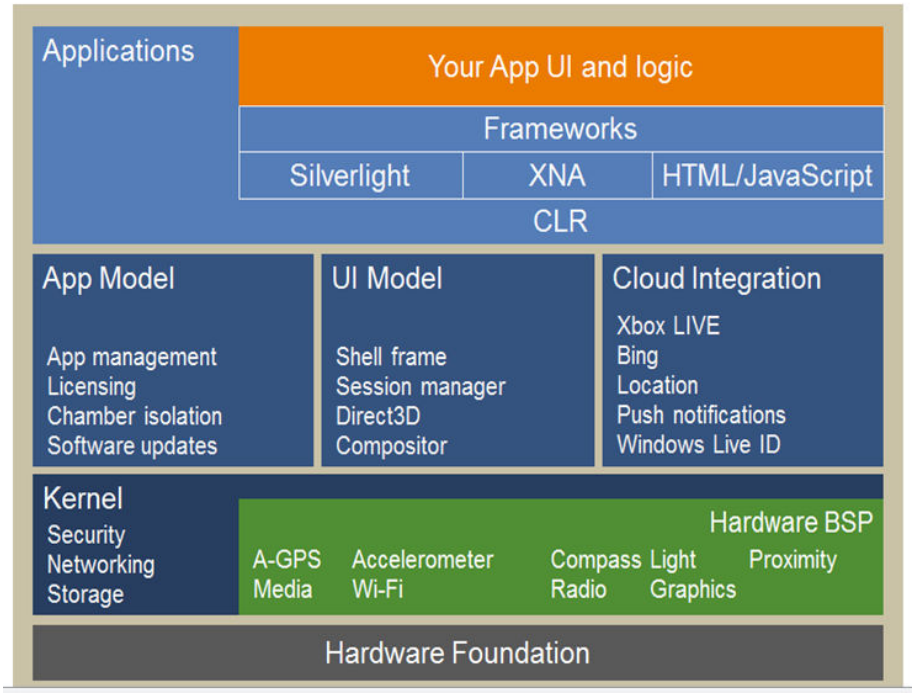
Verzia	**Názov**	**Dátum vydania**	**Stav**
Windows Phone 7		21.október 2010	zastaralý
Windows Phone 7	NoDo	20.marec 2011	zastaralý
Windows Phone 7.5	Mango	24.máj 2011	zastaralý
Windows Phone 7.5	Refresh	15.jún 2011	zastaralý
Windows Phone 7.8		31.január 2013	pripravuje sa
Windows Phone 8		29.október 2012	aktuálny

2.3.1 Architektúra

Spoločnosť Microsoft oficiálne nikdy neuverejnila celkovú architektúru na ktorej je postavený ich mobilný operačný systém Windows Phone. Z viacerých zdrojov však došlo k zjednoteniu názorov, predpokladajúc architektúru z väčšej časti podobnej tej, ktorá je zobrazená na obrázku 2 – 4.

Veľkú výhodu z pohľadu architektúry na svoju stranu sťahuje Windows Phone

tým, že presne definuje hardvérové vybavenie zariadení, ktoré budú pod týmto operačným systémom prevádzkované. Nemôže teda nastať stav nekompatibility z pohľadu rôznych verzií operačných systémov ako v prípade Androidu, pretože zariadenia, ktoré nedefinujú dohodnutú hardvérovú konfiguráciu, nie sú oficiálne licencované. Výrobca vzhľadom na architektúru zakazuje vytváranie nadstavieb pre používateľské rozhranie, čo predstavuje úplne odlišný prístup ako v prípade Androidu, kde každý výrobca využívajúci ich operačný systém vyvíja vlastné rozhranie. Navyše predstavitelia spoločnosti Microsoft, pod ktorý tento systém patrí, uzatvorili dohodu so spoločnosťou Nokia, zahŕňajúca klauzulu, že Windows Phone bude primárne určený pre mobilné zariadenia tejto značky a dôjde k softvérovým zjednoteniam a integráciám poskytovaných služieb[14]. Zariadenia s operačným systémom Windows Phone sú teda zo softvérového pohľadu málo diferencované napriek odlišným výrobcom.



Obr. 2 – 4 Grafická notácia architektúry Windows Phone[15]

Z obrázku2 – 4 je zrejmé, že architektúra operačného systému pozostáva okrem hardvérových prvkov z 3 základných úrovní.

Jadro - je založené na jadre predchodcu tohoto operačného systému, konkrétne Windows Embedded CE verzie 6.0. Tento systém predstavoval systém reálneho času s hybridným jadrom, ktorý bol pôvodne určený pre malé počítače. Je obohatený o funkciu stránkovania pamäte, úložný priestor pre potrebné výpočty a aplikácie a tiež radou sieťových prvkov. Súčasťou tejto vrstvy sú tiež podporné hardvérové prostriedky pre využitie základných prvkov, vrátané rôznych senzorov na špecifickej platforme.

Midlleware - úroveň, ktorá predstavuje prepojenie medzi jadrom systému a samotnými prvkami aplikácií, pričom je dekomponovaná na ďalšie prvky:

- Aplikačný model - obsahuje aplikáciu vrátane zdrojov, manifestu a knižníc v komprimovanom formáte, prostredníctvom ktorého je potom táto aplikácia distribuovaná a licencovaná pre verejnosť. Zároveň sú prostredníctvom tejto časti zabezpečované aktualizácie prostredníctvom obchodu s aplikáciami pre platformu Windows Phone.

- Model používateľského rozhrania - rozdeľuje funkcionalitu aplikácií na menšie časti, ktoré adekvátne stránkuje pre konkrétne sekcie. Takýmto spôsobom sa rieši napríklad navigácia v histórii prehliadača. V tejto vrstve je potom možné vhodným oddelením a využitím prvkov grafického procesora zobrazovať animácie trojrozmernej podoby.

- Integrácia služby Cloud - Windows Phone neponúka synchronizáciu zariadenia s počítačom ako v prípade predošlých verzií operačných systémov od spoločnosti Microsoft, ale ponúka synchronizáciu prostredníctvom služby Cloud, čo predstavuje poskytovanie služieb prostredníctvom internetu. Tieto služby predstavujú spojenie vzdialeným spôsobom s funkciami platformy pre Microsoft.

Aplikačná úroveň - táto vrstva podlieha najviac platforme .NET a je napísaná v programovacom jazyku C#. Obsahuje základné knižnice, sústredené najmä pre zabezpečenie behu aplikácií. Treba však vyzdvihnúť najmä knižnice Silverlight a XNA, ktoré v spolupráci s príslušným aplikačným rámcom a prvkov HTML a Javascript zabezpečujú vytvorenie a chod grafického používateľského rozhrania. Táto úroveň zároveň definuje aplikačnú logiku.

2.3.2 Vývojové prostredie

Pre vývoj aplikácií na platforme Windows Phone je rovnako ako v prípade Androidu potrebná sada vývojárskych nástrojov. V tomto prípade sa jedná o balík Windows Phone Software Development Kit. Tento balík obsahuje aj samotné vývojové prostredie Microsoft Visual Studio funkčne obmedzenej verzie Express. Samotná platforma umožňuje pre vývoj používať len toto jediné prostredie, ktoré je navyše závislé od operačného systému. Inštalácia balíku je vyriešená takým spôsobom, že v prípade detekcie nainštalovaného vývojového prostredia Visual Studio sa už samotné prostredie neinštaluje, iba sa doplní rozšírením pre vývoj na mobilnej platforme. Express edícia neobsahuje ani kompilátor, keďže vývoj prebieha v jazyku C#, ktorý je interpretovaný, ale za to je doplnená o emulátor mobilného zariadenia.

Pri samotnom vývoji sa používateľ rozhodne, či zvolí cestu prostredníctvom technológie Microsoft Silverlight alebo jeho alternatívu, predstavujúcu Microsoft XNA[16]. Obe sú určené pre tvorbu grafických používateľských rozhraní, ale pre odlišné typy aplikácií. Väčšina aplikácii využíva Silverlight v spolupráci so značkovacím jazykom XAML, typickým pre platformu .NET. Microsoft XNA sa používa hlavne na vytváranie mobilných aplikácií s vysokým grafickým výkonom pri 2D a 3D zobrazovaní. Primárne je teda určený pre vývoj hier, avšak jeho prvky môžu byť použité aj ako súčasť komponentov technológie Silverlight.

3 Návrh informačného systému

Webová aplikácia bude určená pre prostredie internetu a základnou požiadavkou pri jej realizácii je zabezpečenie jej fungovania s minimálnymi potrebami pre údržbu. Cieľom je vytvoriť viacvrstvovú architektúru, ktorá by prinášala výhody z pohľadu programátora, kvôli prípadnej migrácii jednotlivých modulov aplikácie.

Témou informačného systému je monitorovanie pohybu vozidiel prostredníctvom GPS modulu, ktorý bude súčasťou mobilného klienta tejto platformy. Aplikácia bude schopná spracovať údaje o konkrétnych pozíciách v tvare súradníc, tvorených dvojicou údajov o zemepisnej šírke a dĺžke a k nim príslušným údajom o rýchlostiach, vzdialenostiach a nadmorskej výške. Následne bude tenký klient komunikovať so statickou časťou informačného systému, kde budú dáta z procesu monitorovania spracované, vyhodnotené a použité v ďalších krokoch. Statickú časť bude poskytovať webová aplikácia, ktorá bude spôsobilá tieto údaje v ďalších krokoch vykresliť ako perspektívu na mapový podklad. Rozšírenou funkciou bude vytváranie štatistických výsledkov v podobe tabuliek a grafov nad výsledkami jednotlivých monitorovacích činností.

Doména systémov pre monitorovanie pohybu pomocou GPS modulov začína byť presýtená. V dnešnej dobe je označovaný ako jeden z najrýchlejšie rozvíjajúcich sa odborov na trhu pre aplikácie využívajúce modul pre lokalizáciu. Rozmach spôsobujú najmä organizácie, ktoré chcú mať pod kontrolou svojich zamestnancov. Existuje niekoľko riešení tohoto problému na rôznych úrovniach. Všetky systémy obsahujú GPS prijímač a softvér pre vyhodnotenie výsledkov. Rozdiely medzi softvérmi sú zásadné, a preto je veľmi ťažké zhodnotiť, ktorý poskytovateľ ponúka softvér s najlepším riešením.

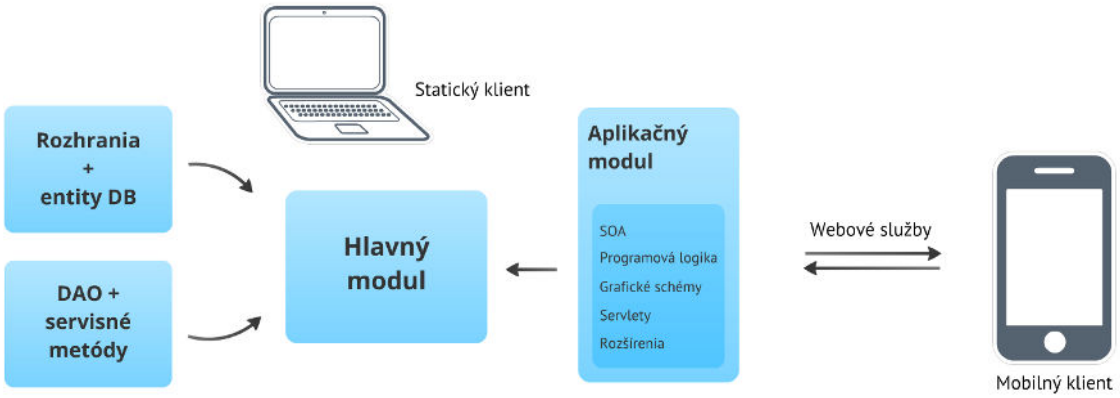
3.1 Architektúra serverovej časti

3.1.1 Kompozícia informačného systému

Návrh architektúry celého systému vychádza zo základnej požiadavky vytvoriť viacvrstvovú webovú aplikáciu, ktorá bude plniť úlohu statickej serverovej časti monitorovacej platformy.

Webová aplikácia je navrhnutá ako 4 samostatné jednotky, z ktorej každá predstavuje špecifický modul. Schéma navrhnutej architektúry je zobrazená na obrázku3 – 1 ako statický klient.

Ďalšiu časť diskutovaného systému bude tvoriť mobilná aplikácia. Tá bude implementovaná nezávisle od statickej časti, pričom musí byť schopná zabezpečiť životný cyklus aplikácie predstavujúcej mobilného klienta. S aplikačným modulom webovej aplikácie budú podliehať servisne orientovanej architektúre, v ktorej sa údaje vymieňajú prostredníctvom odovzdávania správ využitím webových služieb.



Obr. 3 – 1 Návrh celkovej architektúry pre informačný systém

V nasledujúcich sekciách sú analyzované dostupné techniky a prostriedky pre dosiahnutie navrhnutej architektúry a zároveň sú z nich zvolené tie, ktoré budú použité v implementačných postupoch.

3.1.2 SOA - Architektúra orientovaná na služby

Architektúra orientovaná na služby je koncept, ktorý sa stal neodmysliteľnou súčasťou moderných programovacích princípov. Do veľkej miery rozšíril objektovo-orientované programovanie s novým pohľadom na vytváranie väzieb medzi časťami rozsiahlych systémov. Jedným z hlavných cieľov takejto architektúry je zvýšenie prehľadnosti informačných systémov a produktivity vývojárskych tímov pri znižovaní nákladov, k čomu mu pomáhajú agilné princípy.

Spôsob, akým tieto princípy dodržuje spočíva v tom, že na dosiahnutie cieľa využíva budovanie jednotlivých častí systému ako distribuované autonómne služby[17]. V rámci aplikácie služby vytvárajú sieť, ktorej veľkosť je priamo úmerná veľkosti výsledného produktu. SOA nepredstavuje presnú definíciu, ale spôsob spolupráce a prepojenosť väzieb medzi samostatnými celkami.

Pri správnom implementovaní takejto architektúry sa znižuje platformová závislosť, keďže jednotlivé služby medzi sebou komunikujú najčastejšie formou odovzdávania správ. V takomto prípade je možné kedykoľvek a ktorúkoľvek zainteresovanú službu odpojiť a nahradiť ju inou alebo jednotlivé služby rôzne kombinovať. Systém sa stáva stabilnejším, znižuje sa závislosť na konkrétnych prvkoch a je možné rozložiť záťaž na systém rovnomerne.

Webové služby - jednou z najznámejších implementácií SOA sú webové služby. Niektoré definície nesprávne uvádzajú fakt, že aplikácia využívajúca webové služby je servisne orientovaná. Podľa [17] sa s týmto tvrdením nedá súhlasiť, pretože môžu existovať aplikácie používajúce webové služby a pritom nespĺňajú základné princípy pre SOA.

Webové služby predstavujú platformovo nezávislý štandard, ktorým jednotlivé aplikácie alebo časti aplikácií komunikujú medzi sebou v rámci definovanej siete.

V skutočnosti je možné podotknúť, že webové služby predstavujú aplikačné rozhranie prístupné cez sieť. Pre komunikačné účely používajú ako základ komunikačný protokol HTTP. Každá služba je predstaviteľom jednej z 2 hlavných kategórií.

Big web service - používa pre účely odovzdávania správ štrukturované súbory XML vo formáte SOAP, čo predstavuje odľahčený protokol, ktorý je nadstavbou nad HTTP a umožňuje výmenu štrukturovaných informácií v decentralizovanom a distribuovanom prostredí. Poskytované služby sú pre ich konzumentov popisované príslušným jazykom WSDL rovnako využívajúc XML. Cieľom je popis a lokalizácia webových služieb, teda operácie a ich umiestnenie. Je považovaný za štandard pre rozhrania, ktorým sú vytvárané webové služby[18].

RESTful web service - predstavuje novšiu a v poslednej dobe veľmi obľúbenú implementáciu webových služieb. REST predstavuje architektonický štýl, vyvíjaný súčasne s protokolom HTTP, z ktorého bol odvodený na základe obmedzení a je narozdiel od SOAP orientovaný údajovo, nie procedurálne. Pristupuje k údajom, spravidla definovanými základnými metódami protokolu HTTP, konkrétne: POST, GET, PUT a DELETE. Vychádzajúc z obmedzení, podľa ktorých REST vznikol predstavuje komunikáciu typu klient-server, pričom táto komunikácia je bez-stavová, využívajúca vyrovnávaciu pamäť, aby znížila interakcie so serverom, pre získanie údajov na definovaných URI odkazoch[19]. Webové rozhranie preto v súčasnosti kladie dôraz na odklon od používania SOAP protokolu a smeruje k vývoju prostredníctvom REST webových služieb, umožňujúci kombináciu viacerých typov služieb do jedného sofistikovanejšieho riešenia. Prostredníctvom protokolu HTTP používa komunikáciu vo forme XML alebo vo forme modernejšieho variantu JSON.

V navrhovanom informačnom systéme bude využitá komunikácia prostredníctvom RESTful webových služieb a jednotlivé údajové pakety pri komunikácii budú serializované do objektov typu JSON.

3.1.3 Aplikačný rámec

Pre analýzu z pohľadu aplikačnej vrstvy je najdôležitejším krokom výber vhodného rozhrania. Pre potreby praktickej časti prezentovanej práce bol zvolený programovací jazyk Java, z čoho vyplýva, že výber bude zúžený na moderné objektovo-orientované rámce zvoleného jazyka. Tento bod analýzy sa týka najmä serverovej časti diskutovaného systému. Jedná sa o hrubého klienta, ktorý bude sprostredkovaný webovou aplikáciou.

V prvom rade je vhodné vysvetliť, čo predstavuje pojem aplikačný rámec - framework. Existuje niekoľko interpretácií z rôznych zdrojov. Najrozšírenejšou je definícia podľa[20], ktorá uvádza, že sa jedná o model s vysokou úrovňou abstrakcie, konkrétne špecifikovaný pre určitú doménu, prípadne jednotlivé subdomény, ktorý umožňuje využívať rovnaké komponenty z pohľadu aplikačnej vrstvy pri implementácii a vývoji softvérových projektov.

3.1.4 Typy aplikačných rámcov

Frameworky riadené požiadavkami - predstavujú viacvrstvovú architektúru s použitím návrhového vzoru Model View Controller. Pracujú na princípe spracovania HTTP požiadavky na server prostredníctvom handleru, ktorý usmerňuje následný životný cyklus na konkrétne implementácie tried, pre následné vyhodnotenie požiadavky. Je náročný na konfiguráciu kvôli zložitým mapovacím procesom, základ ktorých tvorí striktné oddelenie logickej časti od grafických prvkov. Ako príklad je uvedený Stripes, Spring MVC alebo Struts.

Komponentovo orientované frameworky - podstatou tohoto typu je zachovanie vysokej miery abstrakcie, ktorú zabezpečuje dekompozícia celku na jednotlivé komponenty. Využitím rozhraní komponentov je možné vytvoriť viacvrstvovú architektúru, ktorá vďaka svojim vlastnostiam je schopná automati-

zovať viaceré procesy v prospech programátora. Sú využiteľné hlavne na jednoduchšie výkonovo nenáročné webové aplikácie v prostrediach intranetu s množstvom ovládacích prvkov, najmä kvôli nízkej miere škálovateľnosti. Predstavitelia tejto kategórie sú Wicket, Java Server Faces alebo Tapestry.

RIA frameworky - v preklade "Rich Internet Application"predstavuje frameworky, ktorých charakteristickým znakom je vytvorenie aplikácií v prostredí Internetu s typickými vlastnosťami pre "desktopové" aplikácie. V skutočnosti sa jedná o komplexné riešenia z pohľadu grafických rozhraní s využitím prvkov pre zvýšenie používateľskej kompatibility prostredníctvom najmodernejších webových štandardov. V prípade niektorých predstaviteľov je potreba použitia externých doplnkov pre ich korektné fungovanie. Do tejto kategórie spadajú frameworky ako Vaadin pre platformu Java a v prípade konkurenčnej platformy .NET je to Silverlight.

3.2 Analýza tenkého klienta

3.2.1 Globálny systém určenia polohy

Globálny systém určenia polohy tvorí podstatu pre všetky navigačné prístroje. Samotný systém je tvorený konšteláciou 27 satelitov pohybujúcich sa využitím solárnej energie mimo zemského povrchu. Každý jeden zo satelitov má svoju vlastnú obežnú dráhu, pričom sústava týchto dráh je navrhnutá takým spôsobom, aby bolo možné z akéhokoľvek bodu na Zemi získať spojenie potrebné pre lokalizáciu. V súčasnosti evidujeme 24 funkčných satelitov a 3 rezervné, ktoré sú schopné okamžite vykonávať svoju činnosť po zlyhaní niektorého z aktuálne činných družíc.

Princíp fungovania týchto satelitov je zabezpečený formou vysielania elektromagnetických signálov aktívnym prijímačom. Pre získanie presnej polohy musí prijímač kumulovať signály aspoň zo 4 satelitov. V skutočnosti je poloha určená pomocou

troch z nich a štvrtý potvrdzuje na základe svojich dát jej korektnosť. Podľa počtu naviazaných satelitov v komunikácii a ich jednotlivých vzdialeností od prijímača sa určuje presnosť v metroch vzhľadom na získanú polohu. Proces trilaterácie pracuje na základe vytvárania imaginárnych kružníc z troch rôznych bodov[21]. Výsledkom je samotná poloha, ktorá tvorí priesečník kružníc o polomere vzdialenosti jednotlivých satelitov od prijímača. Zvyšné satelity v komunikácii slúžia na overovanie týchto výsledkov a v prípade nezrovnalostí sa vytvárajú nové kombinácie pre vytváranie kružníc. Na základe časových odoziev pri posielaní signálov je možné okrem presnej polohy podľa kružníc určiť aj nadmorskú výšku a rýchlosť pohybujúceho sa objektu.

Celkovo je systém globálnej navigácie tvorený 3 hlavnými zložkami:

Kozmická - tvoria ju už spomínané satelity na obežných dráhach vysielajúce signály. Nachádzajú sa približne vo výške 20 200 km nad zemským povrchom. Tieto satelity majú prijímací a odosielací modul a súpravu spravujúcu solárny pohybový aparát.

Riadiaca - je zodpovedná za zaistenie bezproblémového chodu celej štruktúry systému. Tvoria ju 4 pozemné monitorovacie stanice rozmiestnené v rôznych častiach sveta a hlavné riadiace centrum v Spojených štátoch amerických. Monitorovacie stanice sledujú priebeh činností družíc a predávajú tieto údaje riadiacemu centru, pomocou ktorých majú prehľad o ich polohe a stave.

Užívateľská - zložka je tvorená používateľmi s prijímačmi signálov, ktoré posielané údaje spracúvajú. Údaje sa zbierajú prostredníctvom jedného alebo viacerých zberných kanálov a sú vyhodnocované výpočtovými zariadeniami podľa typu.

3.2.2 Koncepcie lokalizácie

V súčasnej dobe rozlišujeme mobilné monitorovacie zariadenia využívajúce GPS modul podľa spôsobu, akým údaje zaznamenávajú a následne synchronizujú s vyhodnocovacím softvérom na 3 základné kategórie:

Dátové zapisovače - Predstavuje kategóriu pasívnych monitorovacích systémov, ktoré získané dáta ukladajú do vnútorných pamätí zariadení, prípadne na externé média pripojené k daným zariadeniam. Produkujú súbory špecifických formátov, ktoré sa potom z daných zariadení získavajú rôznymi spôsobmi. Súbory sú spracovávané hrubými klientmi systémov. Tento spôsob sa využíva aj v prípade podpisovania fotografií, ku ktorým je možné pripojiť EXIF metaúdaje, v danom prípade pozíciu, kde bola fotografia zhotovená.

Dátové zavádzače - V súčasnosti najpoužívanejší typ zariadení pre personálne aj produktívne účely. Pracujú na princípe, ktorý v pravidelných časových intervaloch posiela údaje zo zariadenia pre spracovanie na serverovú časť. Pri komunikácii so serverom môže pre posielanie údajov využívať internet alebo v prípade zariadení vybavených SIM kartou, paketovú rádiovú službu GPRS. Túto kategóriu zvyčajne charakterizujú zariadenia napevno namontované vo vozidlách so špecifickým hardvérom, ktorý slúži ako monitorovacia jednotka v spolupráci so softvérovou serverovou časťou. V dnešnej dobe vývoj mobilných telefónov pokročil natoľko, že môžu plniť funkciu zapisovačov aj údajových zavádzačov súčasne.

Dátové obstarávače - Poslednú kategóriu tvoria údajové obstarávače, niekedy nazývané transpondéry alebo sprostredkovatele. Sú presným protikladom v porovnaní s predošlou kategóriou. Takéto zariadenia sa využívajú v prípade, ak sa chceme ojedinele dotazovať na ich pozíciu. Fungujú na princípe, že sú zapnuté nepretržite a prostredníctvom dopytu zo strany servera sa vyžiada odoslanie

údaju o ich aktuálnej polohe. Príkladom môže byť zabezpečenie zariadení so zdrojom energie proti krádeži.

3.2.3 Samotná aplikácia

Pre potreby implementácie je primárnym zámerom integrovanie údajových zapisovačov a zavádzačov a vytvorenie komplexného systému pre monitorovanie v spolupráci s analyzovanou webovou aplikáciou v kapitole 3.1.1. Zariadenie pre prenos dát bude zabezpečovať mobilné zariadenie s GPS modulom na platforme Android, ktoré poskytuje otvorené rozhranie pre prácu s modulom poskytujúcim prijímanie družicových signálov v zariadeniach.

Platformu bola zvolená hlavne z dôvodu prístupnej synchronizácie webovej a mobilnej aplikácie prostredníctvom prihláseného užívateľa cez Google účet, keďže hlavnou identitou v zariadení s operačným systémom Android je práve táto paradigma. Navyše sa pre vývoj a testovanie vzniknutej aplikácie nevyžadujú žiadne finančné príspevky ani registrácie pre obstaranie vývojových prostriedkov v porovnaní s konkurenčnými platformami.

4 Implementácia systému CarTrackingSystem

4.1 Použité technológie

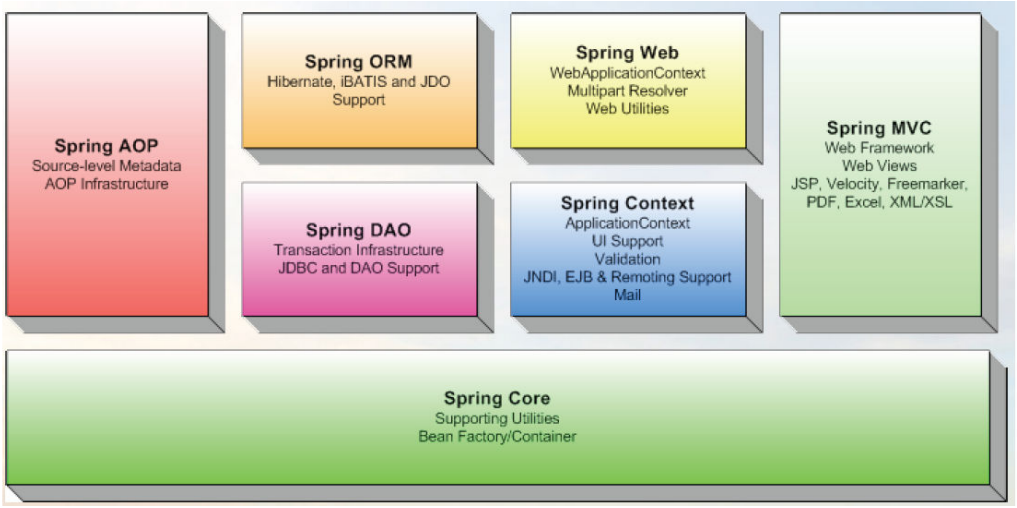
Na základe porovnania základných vlastností v kapitole 3.1.4 sa pri výbere aplikačného rámca rozhodlo pre použitie kombinácie Spring - Vaadin. Prvý menovaný bude riešiť špecifickú logiku na pozadí, ktorá bude zahŕňať najmä interakciu prostredníctvom nástroja objektovo-relačného mapovania s databázou typu MySql. Vaadin bude vykonávať aplikačnú logiku, prostredníctvom ktorej budú výsledky z dátovej vrstvy spracované touto logikou transformovať do prezentačnej vrstvy pre používateľa.

4.1.1 Spring

Spring sa môže charakterizovať ako jeden z najpopulárnejších aplikačných rámcov medzi vývojármi. Vyznačuje sa robustnosťou a integruje v sebe rozsiahlu škálu prostriedkov pre vývoj. Spring pôvodne vznikol ako náhrada za populárne EJB, ktoré striktne dodržovali zásady oddelenia jednotlivých aplikačných vrstiev.

Základnou charakteristickou vlastnosťou je teda modulárnosť, čo bol hlavný podnet pre rozhodnutie ho použiť. Toto aplikačné rozhranie obsahuje množstvo modulov, ktoré je možné použiť samostatne pre špecifické potreby a nevyžaduje ich aplikovanie ako celku. Architektúru Springu graficky znázorňuje obrázok4 – 1. V tomto prípade bolo niekoľko modulov vybratých pre potreby implementácie informačného systému. V nasledujúcej časti práce sú priblížené:

Spring CORE - je definovaný ako hlavný modul aplikačného rámca. Implementuje jeho spôsob fungovania dvomi hlavnými princípmi. Prvým princípom je návrhový vzor IOC, čo v praxi znamená obrátené riadenie životného cyklu(Nevolaj ma, ja ťa budem kontaktovať, ak budem mať záujem). Kód frameworku spúšťa aplikačný kód a koordinuje celý vykonávací tok kódu, resp.



Obr. 4 – 1 Architektúra aplikačného rozhrania Spring [22]

zdrojový text aplikácie neaktivuje priamo implementáciu frameworku. Tento spôsob zahrňuje spôsob použitia rozhraní, ktorých cieľom je alokácia a uvoľňovanie zdrojov.

Druhý princíp predstavuje návrhový vzor ”Dependency injection”. Jedná sa o konštrukčne založený mechanizmus jazyku Java, ktorý neberie do úvahy špecifické rozhrania pre framework. Aplikačný kód nevyužíva programové rozhranie pre riešenie závislostí v konfiguračných parametroch a kolaburujúcich objektoch[24]. Aplikačné triedy vystavia svoje závislosti cez konštruktory, ktoré sú volané samotným frameworkom na základe konfigurácie. Predstavuje teda princíp vkladania závislostí do aplikácie počas behu programu. V diskutovanom informačnom systéme bol tento návrhový vzor použitý v spolupráci s nástrojom Maven, ktorý je priblížený v kapitole 4.2.1. Súčasťou informačného systému je tiež konfigurácia prostredníctvom rozhrania ApplicationContext, ktorý združuje použitie opísaných nástrojov aplikačného rámca Spring.

Spring AOP - predstavuje popri module core jednu z kľúčových častí tohoto frameworku. Charakterizuje sa ako aspektovo-orientovaný rámec Springu, ktorý je založený na princípe proxy. Je zástupcom konceptu “interceptor chain”,

čo predstavuje akúsi reťaz narúšania. Aktivácia je týmto spôsobom presmerovaná na cieľový objekt - bod vstupu, pričom interceptor vykonáva logické akcie. Môže dôjsť k trom rôznym fázam aktivácie, kde je funkcionalita akcie vsunutá pred aktiváciu, nasleduje aktivácia metódy a v ďalšom kroku sa vsúva funkcionalita po vykonaní metódy. Tieto fázy vykonania sú voliteľné. Jeho výhodou je, že umožňuje vkladať v určitých bodoch programu bez toho, aby sa menil zdrojový kód základného modulu. V našom prípade je tento modul použitý pre zapuzdrenie servisných metód a DAO objektov.

Spring DAO - hlavnou funkciou tohoto modulu je oddeliť kód perzistencie od biznis logiky. Tá teda nebude závislá od technológie perzistencie a prístup k dátovým zdrojom je štandardizovaný. DAO predstavuje objekt s vysokou mierou abstrakcie, ktorý reprezentuje rozhranie k entitám uloženým v databáze. Poskytuje základné CRUD metódy ku konkrétnej entite bez toho, aby k svojej existencii potreboval základné vlastnosti databázy. V spolupráci s modulom pre objektovo-relačné mapovanie poskytuje základ perzistentnej vrstvy a špecifikuje komunikáciu medzi perzistentnou a aplikačnou vrstvou aplikácií. Spôsob použitia týchto modulov v informačnom systéme je podrobnejšie popísaný v jednotlivých moduloch aplikácie popísaných v kapitole 4.2.2.

4.1.2 Hibernate

Hibernate vytvára komplexnú architektúru s prepracovanou funkcionalitou poskytujúcou spôsob objektovo-relačného mapovania objektov tried zameraných prevažne na objektovo-orientované jazyky a na reprezentáciu údajov v tabuľkách relačnej databázy. Hibernate tiež sprístupňuje nástroje využívajúce spracovanie dát prostredníctvom transakcií, polymorfnú úroveň pre perzistentné objekty, požiadavky zamerané a prísne dodržujúce objektovo-orientované princípy a nebráni sa ani implementácii vlastných dátových typov ako súčasť komplexnej dátovej vrstvy. Hibernate podpo-

ruje množstvo databázových typov a jeho prístup je v tomto smere rovnaký s jedinou požiadavkou pre nakonfigurovanie správneho pripájača pre konkrétny typ. Pri korektnej konfigurácii Hibernatu v projektovej štruktúre je možné premigrovať na ľubovoľný databázový typ.

Je vo vysokej miere škálovateľný, čiže ľahko použiteľný v clustrových prostrediach. Účinným spôsobom využíva prvky dedičnosti a polymorfizmu a s aplikovaním generických tried predstavuje efektívny nástroj, schopný redukovať čas vývojárov vzhľadom na perzistentnú vrstvu.

Samotné mapovanie môže byť implementované formou xml konfiguračného súboru, obľúbenejším spôsobom je však použitie anotácií (Hibernate annotations). V prípade, že je model databázovej vrstvy navrhnutý korektne a do budúcnosti bez požiadaviek na úpravy je vhodnejšie využívať anotácie. V prípade, ak sa jedná o dynamický model neustále podliehajúci zmenám, odporúča sa použiť mapovanie prostredníctvom xml.

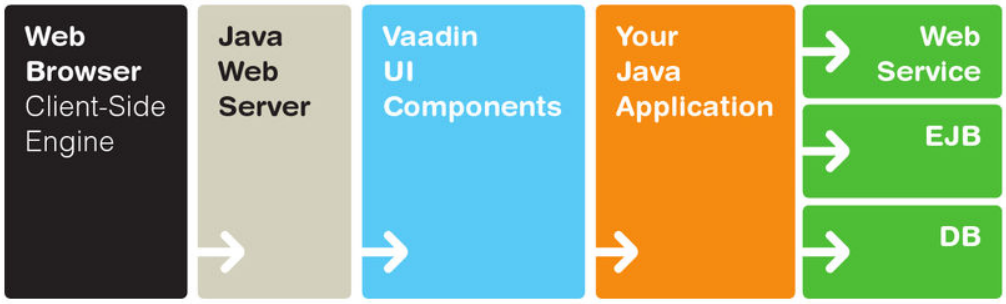
Výber tejto technológie uľahčil fakt, že je priamo integrovateľná s vyššie spomenutým modulom Spring ORM rozhrania Spring a zároveň predstavuje celosvetovo populárnu náhradu za nepraktické statické riešenie cez JDBC prístup.

4.1.3 Vaadin

Vaadin je webový aplikačný framework pre RIA aplikácie. Na rozdiel od Javascriptu a jeho využívaniu na strane klienta (v prehliadači) sa vyznačuje robustnou architektúrou na strane servera a poskytuje serverom riadený model, čo invokuje k úplne odlišnému programovaniu webových aplikácií ako pri iných Java webovo-orientovaných frameworkoch. Vývoj skôr pripomína programovanie starších AWT a SWT aplikácií. Základ tohto aplikačného rámca môžeme hľadať za dobre známym rozhraním GWT. Architektúra Vaadin-u je prevažne postavená na 2 základných

častiach.

- framework na strane servera – beží ako inštancia java servletu na serverovej časti,
- vaadin engine na strane klienta – ktorý beží v prehliadači ako java-scriptový program pre vzbudenie bohatšieho a dokonalejšieho dojmu.



Obr. 4 – 2 Architektúra Vaadin aplikačného frameworku na strane serveru [23]

Vaadin engine na strane klienta nepotrebuje pre svoju funkčnosť inštalovanie žiadneho z externých rozšírení. V rámci aplikačnej logiky vo veľkej miere využíva JavaScript, čo je hlavným znakom, ktorý podlieha dedičnosti z jeho predchodcu GWT.

Ďalšou črtou, ktorou sa Vaadin ako RIA framework vyznačuje je používanie AJAX. Prostriedky na strane klienta teda kompilujú aplikačnú vrstvu napísanú v jazyku Java do JavaScriptu, čo vysvetľuje rýchle a stabilné odozvy v rámci aplikácie.

Vaadin obsahuje veľké množstvo komponentov na tvorbu používateľského rozhrania pre aplikáciu ako tlačítka, tabuľky, stromy a schémy pre usporiadanie. Tieto komponenty využívajú rôzne udalosti, prijímače a údajové spojenia na komunikáciu medzi sebou a hlavnou logikou aplikácie. Komponentovo založená architektúra Vaadinu s programovacím jazykom Java a funkciami zabezpečujúcimi dátové spojenia, pomáha vytvárať jednoducho modifikovateľné aplikácie. Vaddin štandardne posky-

tuje jadro vzhľadu, striktne oddelené od logiky aplikácie a šablóny v ňom nazývané ako témy. V rámci implementácie frameworku je poskytovaných niekoľko základných tém pre vzhľad aplikácií, ktorých implementácia je prekvapivo na vysokej úrovni. Ak však je potreba tieto témy meniť, jednoduchým spôsobom prekryjeme importovaný kaskádový štýl základných tém. Prístup ku konkrétnym HTML šablónam stránok je relatívne obmedzený.

Framework je ľahko integrovateľný s IDE SpringSourceTools a rozšírením určeným pre Eclipse vývojové prostredie. Poskytuje podporu rôznych nástrojov vrátane nástroja na tvorbu dizajnu a rozloženia jednotlivých stránok. Uľahčuje vytvárať používateľské rozhrania vo veľmi krátkom čase. Voľba padla na toto aplikačné rozhranie aj z dôvodu jej podpory pre integráciu s vyššie spomenutými nástrojmi.

Vaadin je špecifický framework určený pre konkrétne úlohy a typy webových aplikácií. V prípade, že je potrebné implementovať niečo podľa striktných pravidiel, prípadne prispôsobiť si niektoré časti aplikácií svojim potrebám, vývojár narazí na vysokú mieru obmedzenosti. Pri každom špecifickom kroku je potrebné upraviť množstvo konfiguračných súborov a spätná kompatibilita pre jednotlivé rozšírenia je na slabej úrovni.

4.1.4 Android annotations

Android annotations predstavuje framework otvoreného zdroju. Jeho zámerom je zrýchliť a optimalizovať vývoj na platforme Android. Jeho podstatou je písať jednoduchší zdrojový kód, čo uľahčuje jeho následnú údržbu.

Základom tohoto rámca je odbremeniť vývojára od písania rovnakých šablón a signatúr jednotlivých metód, čo mu umožňuje koncentrovať sa na dôležité aspekty programu. Výsledkom používania takýchto anotácií je v konečnom dôsledku rovnaký zdrojový kód ako ten, ktorý by napísal vývojár, avšak pri použití vhodných

anotačných procesov je približne 30% automaticky vygenerovaných na pozadí. Vývojári naznačia svoj zámer a implementácia frameworku vygeneruje potrebný zdrojový text v čase kompilácie. V informačnom systéme bol tento projekt použitý ako externá referencia v podobe knižnice. Ako príklad použitia je uvedená anotovaná trieda fragmentu *HomeFragment.class*:

```
@EFragment(R.layout.home_fragment_layout)
public class HomeFragment extends Fragment {

  @ViewById(R.id.carSelect)
  Spinner carSelect;

  @Click(R.id.startTrackButton)
  public void startBtnClick() {
  ...
  }
}
```

Pre porovnanie je uvedený aj zdrojový text s rovnakou funkcionalitou bez použitia anotácií:

```
public class HomeFragment extends Fragment {

  private Spinner carSelect;
  private Button startBtn;

  @Override
  public View onCreateView(LayoutInflater inflater,
                           ViewGroup container,
                           Bundle savedInstanceState) {

    view = inflater.inflate(R.layout.home_fragment_layout, container, false);

    carSelect = (Spinner) view.findViewById(R.id.carSelect);
    startBtn = (Button) view.findViewById(R.id.startTrackButton);

    startBtn.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View v) {
        ...
      }
    });
    return view;
  }
}
```

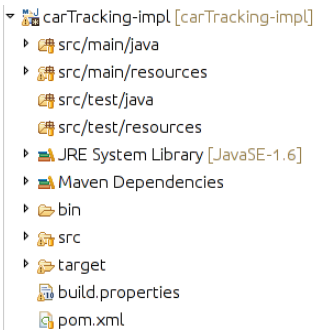
V názornej ukážke je možné vidieť podstatný rozdiel v prehľadnosti a úspore textu. Android annotations implementuje množstvo ďalších anotácií, použitie niektorých je bližšie opísané v kapitole 4.3.4. Jedinou požiadavkou pri programovaní týmto spôsobom je neuvádzať modifikátory pre premenné kvôli zachovaniu prístupu pre anotačný procesor frameworku.

4.2 Webová aplikácia

Architektúra aplikácie vychádza z predstaveného návrhu v kapitole3.1.1. Statický klient informačného systému pozostáva zo 4 modulov. Každý z nich je určený pre konkrétne účely, pričom podstata tohoto riešenia vychádza zo základného princípu, podľa ktorého by malo byť aplikáciu alebo jej časti možné jednoduchým spôsobom v prípade potreby migrovať.

4.2.1 Maven projektová štruktúra

Súčasťou každého vývojového procesu je zostavenie a preklad jednotlivých zdrojových textov alebo ich častí do výsledného produktu. V opísanom riešení bol tento proces automatizovaný pomocou nástroja pre uľahčenie vytvárania projektových zostáv Maven, ktorý poskytuje možnosti nielen pre programátorov, ale aj pre analytikov a testerov najmä z pohľadu správy projektu. V prípade implementovaného systému podliehajú všetky moduly takejto projektovej štruktúre, zdokumentovanej na obrázku4 – 3.



Obr. 4 – 3 Projektová štruktúra vybraného modulu aplikácie

Výhodou tohoto nástroja je platformová nezávislosť a tiež je možné ho použiť ako súčasť všetkých dostupných vývojových nástrojov s vhodným rozšírením alebo ho spúšťať priamo z príkazového riadku. Pri prvom vytváraní projektovej zostavy vyžaduje prístup k internetu, prostredníctvom ktorého získava potrebné závislosti

v podobe knižníc a verziovaných konfiguračných súborov vrátane tranzitívnych závislostí z repozitárov, ktoré ukladá lokálne do pamäte zariadenia a vytvára z nich lokálny repozitár dotazovaný pri ďalších spusteniach.

Základným prvkom Maven projektovej štruktúry je súbor *pom.xml* - Project Object Model. Prostredníctvom tohoto súboru sú definované vlastnosti samotného projektu v 4 základných skupinách, kde narozdiel od ostatných nástrojov neobsahuje konkrétne príkazy pre vykonanie, ale atribúty spracované nástrojom. Najsilnejšou stránkou Maven-u je správa závislostí, ktorá vychádza z návrhového vzoru Dependency injection. Závislosťou sú v tomto prípade externé artefakty, ktoré sú využívané inými artefaktmi pri kompilácii. Sú jednoznačne definované skupinou, názvom, verziou a prípadne mierou pôsobnosti. Ukážkou takejto závislosti je použitie nástroja pre zobrazovanie mapových podkladov v informačnom systéme.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.google.gwt</groupId>
      <artifactId>gwt-maps-widget</artifactId>
      <version>3.8.0</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

4.2.2 Moduly aplikácie

Návrhový vzor dependency injection v spolupráci s nástrojom Maven je použitý v každom jednom module webovej aplikácie. V tejto časti práce je opísaný ich obsah a funkcionalita:

- **carTracking-api** - modul poskytuje všetky rozhrania pre spracovanie DAO objektov a servisných tried v perzistentnej vrstve. Okrem toho súčasťou sú aj objekty predstavujúce jednotlivé entity databázového modelu, ktoré sú využívane pri transakciách objektovo-relačného mapovania prostredníctvom nástroju Hibernate. Ten využíva mapovacie súbory vo forme XML pre definova-

nie, akým spôsobom sa majú tieto entitné modely a jeho vlastnosti transformovať do databázy. Alternatívou k mapovacím súborom je použitie anotácií. Pre názornosť je možné uviesť ukážku objektu Track:

```
@Entity
@Table(name = "Track", catalog = "cartrackingsystem")
public class Track implements java.io.Serializable {

  private long idTrack;
  private Car car;

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  @Column(name = "id_track", unique = true, nullable = false)
  public long getIdTrack() {
    return this.idTrack;
  }

  @ManyToOne(fetch = FetchType.EAGER)
  @JoinColumn(name = "id_car", nullable = false)
  public Car getCar() {
    return this.car;
  }
}
```

V tomto prípade anotácia *@Entity* určuje, že ide o entitnú triedu, ktorá bude mapovaná na konkrétnu tabuľku v databáze. Tabuľka je špecifikovaná ďalšou anotáciou *@Table*, ktorej parametrami sú jej názov a použitá schéma. Vlastnosť idTrack je anotovaná prostredníctvom @Id, ktorá zabezpečí, že bude predstavovať primárny kľúč, ktorý je zároveň špecifikovaný anotáciou @GeneratedValue s parametrom, ktorý udáva automatické inkrementovanie. *@Column* označuje stĺpec v tabuľke, na ktorý sa daná vlastnosť mapuje. V prípade vnorenej entity sa pomocou *@ManyToOne* nastaví vzťah medzi entitami s mapovaním na stĺpec cez *@JoinColumn*.

- **carTracking-impl** - tento modul spolu s modulom carTracking-api tvorí základ perzistentnej vrstvy. Obsahuje triedy typu Data Access Object k jednotlivým entitám. Okrem toho obsahuje generický objekt, ktorý implementuje základné CRUD operácie pre entitu. Súčasťou takýchto tried sú metódy vykonávajúce transakcie nad databázou podľa definovaných kritérií metódou objektovo-relačného mapovania. Uvedieme príklad takejto triedy pre entitu Track:

```
public class TrackDaoImpl extends GenericDaoImpl<Track, Long> implements
    TrackDao {

  @Override
  public List<Track> getTracksToCar(Car car, Person person) {
    Criteria reservationCrit = getSession().createCriteria(Track.class);
    reservationCrit.add(Restrictions.eq("person", person));
    reservationCrit.add(Restrictions.eq("car", car));
    reservationCrit.addOrder(Order.desc("startTime"));

    List<Track> res = reservationCrit.list();
    return (res.size() > 0) ? res : null;
  }
}
```

Rodičom je trieda *GenericDaoImpl*, ktorá zabezpečí implementáciu CRUD operácií a trieda implementuje rozhranie *TrackDao*, obsahujúce signatúry metód. Príkladom je metóda, zabezpečujúca transakciu nad databázou a vrátenie všetkých entít *Track* ku špecifikovanému vozidlu použitého daným používateľom v parametri. Princíp spočíva vo vytvorení objektu typu *Criteria* nad konkrétnou entitou, ku ktorej pridávame obmedzenia podľa názvu vlastnosti v entite. Jednou z možností je obmedzenie zoradenia podľa konkrétnej vlastnosti. Návratovou hodnotu je zoznam výsledkov podľa entity.

- **carTracking-parent** - predstavuje hlavný modul celej aplikácie. Neobsahuje žiadnu konkrétnu logiku pri spracovaní. Jeho úlohou je združenie ostatných modulov aplikácie a zabezpečenie modularity. Jeho súčasťou je jediný súbor typu Project Object Model, ktorý definuje základné vlastnosti a závislosti tohoto systému. Spôsob akým túto funkcionalitu vykonáva je uvedený v priloženom zdrojovom texte.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://maven.apache.org/POM/4.0.0">

  <modelVersion>4.0.0</modelVersion>
  <groupId>sk.tuke.carTrackingSystem</groupId>
  <artifactId>carTracking-parent</artifactId>
  <packaging>pom</packaging>
  <version>1.0.1-SNAPSHOT</version>
  <name>carTracking-parent</name>

  <modules>
    <module>../carTracking-api</module>
    <module>../carTracking-impl</module>
    <module>../carTracking-web</module>
  </modules>
```

```
    <dependencyManagement>
      <dependencies>
        ...
      <dependencies>
    </dependencyManagement>

  </project>
```

Zároveň je vrstva modulu základom pre vytváranie zostáv, preklad a spustenie. Na tieto účely boli využité nástroje maven:goal, konkrétne *clean, package, install*. Samotné spúšťanie bolo zabezpečené prostredníctvom aplikačného serveru jetty príkazom *jetty:run* ako súčasť maven:goals.

- **carTracking-web** - modul predstavuje hlavnú časť aplikačnej a prezentačnej vrstvy informačného systému. Jeho súčasťou je celá aplikačná logika, ktorá sa v aplikácii vykonáva. Implementácia modulu je riešená prostredníctvom webového aplikačného rámca Vaadin.

  Ako príklad je uvedený postup pre vytvorenie registračného formulára prostredníctvom priradenia entity, ktorá mapuje tabuľku z databázy, pre objekt typu *BeanItem*. Tento objekt vytvorí generátor formulárových komponentov, pre ktorý budú vstupné hodnoty a ich dátové typy odvodené podľa vlastností entity. Aplikačný rámec je schopný podľa týchto vlastností vrátane špecifických mapovaní vytvoriť formulár. V prípade, že chceme dodefinovať konkrétnu funkcionalitu, je možné prekryť generovanie konkrétneho komponentu. V prvom kroku vytvoríme triedu, ktorá bude dediť od objektu *DefaultFieldFactory*. V nej pridáme komponentu, ktorej vlasnosť bude mať názov *name* validačné prvky nasledovne.

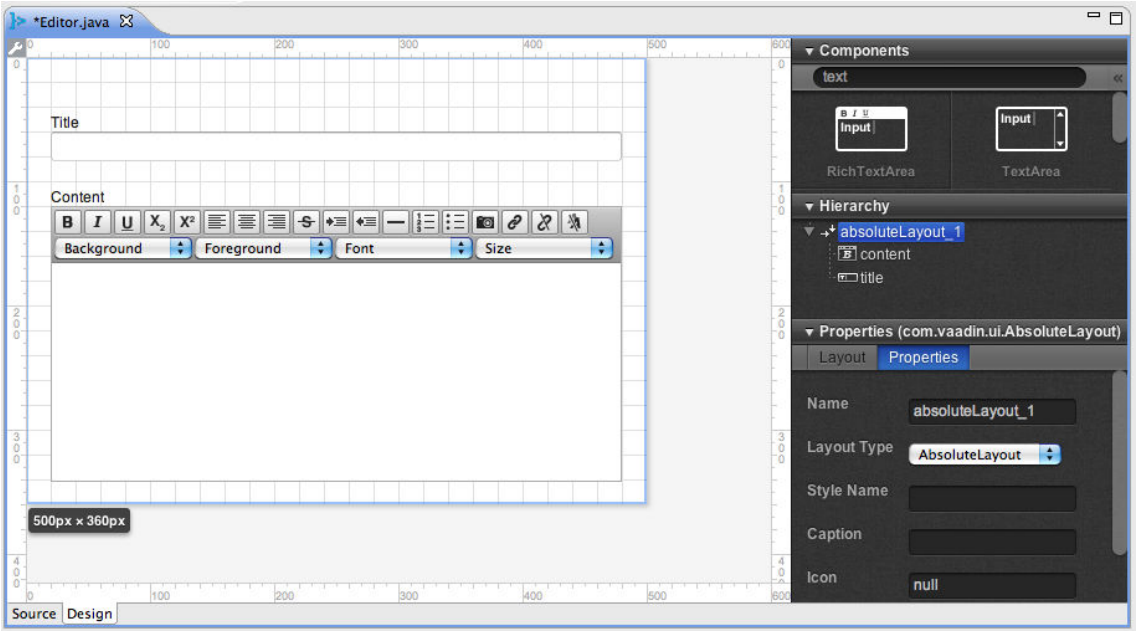
```
private class PersonFieldFactory extends DefaultFieldFactory {

  @Override
  public Field createField(Item item, Object propertyId,  Component
      uiContext) {
    Field f;

    f = super.createField(item, propertyId, uiContext);

    if ("name".equals(propertyId)) {
      TextField tf = (TextField) f;
      tf.setNullRepresentation("");
```

```
            tf.setRequired(true);
            tf.setImmediate(true);
            tf.setRequiredError("Please enter a First Name");
            tf.setWidth(COMMON_FIELD_WIDTH);
            tf.addValidator(new StringLengthValidator("
              First Name must be 3-25 characters", 3, 50, false));
          }
        }
      }
```

Tento objekt je použitý pre vytvorenie komponentov na stránke a je priradený spolu s objektom *BeanItem*, ktorého vstupným parametrom bude entita konkrétnemu formulárovému komponentu typu *Form*. Implementovaný registračný formulár bude napĺňať tabuľku *Person*, čiže bude pracovať s entitou na tabuľku mapovanou v module carTracking-api. Metódou *setVisibleItemProperties()* sa uvádzajú vlastnosti, ktoré pre priradenú entity budú zohľadnené.

```
public class RegistrationPage extends VerticalLayout {

  private Form form;
  private BeanItem<Person> personItem;
  private Person person;

  public RegistrationPage(String caption) {
    person = new Person();
    personItem = new BeanItem<Person>(person);

    form = new Form();
    personForm.setFormFieldFactory(new PersonFieldFactory());
    personForm.setItemDataSource(personItem);
    addComponent(form);

    setVisibleItemProperties(Arrays.asList(new String[] {
                             "name", "surname", "email", "password" }));
  }
}
```

Vytvorenú triedu pre generovanie komponentov z vlastností priradíme formulárovému komponentu v metóde *setFormFieldFactory()*. Rovnakému komponentu musíme nastaviť aj objekt s našou entitou v objekte typu *BeanItem* cez metódu *setItemDataSource()*. V aplikačnom rámci Vaadin je možné celkový návrh stránok vytvárať prostredníctvom dizajnérskeho prvku rozšírenia Vaadin pre vývojárske prostredie Eclipse. Je možné ho vytvárať aj manuálne, keďže sa jedná o klasické rozmiestnenie prvkov v schémach, ktoré je dobre známe z iných nástrojov alebo rámcov.

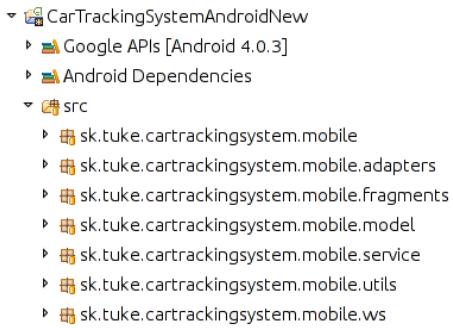


Obr. 4 – 4 Nástroj pre návrh stránok v aplikačnom rámci Vaadin

4.3 Mobilná aplikácia

4.3.1 Štruktúra aplikácie

Projekt je vytvorený vo vývojovom prostredí Eclipse verzie Indigo s oficiálne podporovaným rozšírením ADT pre Android. Jedná sa o klasickú *Android project* konfiguráciu s podporou pre operačný systém Android verzie 4.1 a vyššej. Logika a prvky aplikácie sú rozdelené do niekoľkých balíčkov, ktoré sú znázornené na obrázku 4 – 5.



Obr. 4 – 5 Projektová štruktúra mobilného klienta

Stručný opis funkcionality tried zahrnutých v balíčkoch je nasledujúci:

- ***.mobile** - balíček predstavuje hlavnú triedu objektu *Activity*, ktorá má na starosť životný cyklus aplikácie, vrátane základného rozvrhnutia jednotlivých fragmentov.
- ***.adapters** - súčasťou balíčka sú nastaviteľné adaptéry pre komponenty obsahujúce vstupné údaje zo zoznamov, ktoré sú vo väčšine prípadov prepojené s lokálnou databázou.
- ***.fragments** - balíček obsahuje triedy typu *Fragment*, ktoré zabezpečujú 3 hlavné obrazovky v aplikácii vo forme kariet. Ich nadradeným komponentom je panel akcií, kontrolujúci jednotlivé inštancie týchto fragmentov.
- ***.model** - zložky balíčka mapujú tabuľky v databáze na entity. Okrem toho obsahuje aj odvodené entity pre špecifické prípady a tiež triedy, ktoré sa serializujú do objektov posielaných pri komunikácii prostredníctvom webových služieb.
- ***.service** - jediná trieda objektu *Service* využívaná pri procese monitorovania prostredníctvom GPS modulu v zariadení, kvôli zachovaniu integrity aplikácie aj v prípade vloženia vykonávania aplikácie na pozadie.
- ***.utils** - balíček obsahuje nástroje využívané ostatnými triedami. Konkrétne sa jedná o konektor k databáze, ktorý spravuje všetky lokálne transakcie, syntaktický analyzátor pre potreby získania údajov zo špecifických formátov a v neposlednom rade nástroj pre exportovanie údajov do formátu spracovateľného službami pre mapy od spoločnosti Google.
- ***.ws** - balíček predstavuje klienta, ktorý riadi komunikáciu so serverovou časťou informačného systému prostredníctvom webových služieb a metód v ňom definovaných v rámci architektúry orientovanej na služby .

4.3.2 Správa lokalizácie

Základná myšlienka informačného systému spočíva v získavaní koordinácií pre potreby zabezpečenia lokalizácie v rámci monitorovania pohybu. Koordinácie sú získavané prostredníctvom družicových satelitov, ktoré tvoria globálny systém určenia polohy. V prípade diskutovaného informačného systému predstavuje zariadenie komunikujúce na tejto paradigme mobilný telefón s GPS modulom a operačným systémom Android.

V Androide je spravovaný tento modul prostredníctvom triedy *LocationManager*. Základnou požiadavkou je zabezpečiť, aby bol GPS modul po spustení prístupný a aktívny aj v prípade, že sa používateľ rozhodne vykonávať inú činnosť v rámci operačného systému jeho zariadenia a vlákno s procesom monitorovania sa presunie na pozadie. Pre tento účel bolo potrebné implementovať už spomínanú triedu objektu *Service*. Vo fragmente, ktorý slúži pre aktiváciu procesu lokalizácie je vytvorená inštancia takéhoto objektu a zároveň musí byť naviazaná na konkrétne spojenie.

```
private CarTrackingService carTrackingService;

public ServiceConnection carTrackingServiceConnection = new ServiceConnection() {

  public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
    carTrackingService = ((CarTrackingService.CarTrackingBinder) iBinder)
                         .getCarTrackingService();
    }

  public void onServiceDisconnected(ComponentName componentName)  {
    carTrackingService = null;
    }
  };
```

Samotné spustenie takejto služby je priradené tlačidlu, ktoré získa impulz od používateľa a na stlačenie reaguje volaním nasledujúcej časti zdrojového textu. V tejto metóde sa inicializuje objekt typu *Intent*, ktorý je následne spojený s danou triedou. Vykonávanie je teda presunuté do triedy *CarTrackingService.class*, kde je posunutá inštancia objektu *Activity*, predstavujúca hlavnú triedu životného cyklu aplikácií na platforme Android. Táto logika je implementovaná ako súčasť objektu *Fragment*, preto platí obmedzenie, podľa ktorého nie je možné pristupovať priamo

k *Activity* triede kľúčovým slovom *this*. Preto je potrebné volanie v tvare *getActivity().getApplication()*.

```
private Intent carTrackingServiceIntent;

@Click(R.id.startTrackButton) {
public void startBtnClick() {

  carTrackingServiceIntent = new Intent(getActivity().getApplication(),
                                        CarTrackingService.class);

  getActivity().getApplication().startService(carTrackingServiceIntent);
  getActivity().getApplication().bindService(
        new Intent(getActivity().getApplication(),CarTrackingService.class),
        carTrackingServiceConnection,
        Context.BIND_AUTO_CREATE);
  }
```

Základným krokom opísanej služby je deklarácia a inicializácia objektu typu *LocationManager* pre potreby lokalizácie. Tento objekt má prístup k hardvérovému GPS modulu zariadenia podľa deklarovaných povolení v hlavnom konfiguračnom súbore projektovej štruktúry *AndroidManifest.xml*. Následne je potrebné vytvoriť triedu, ktorá implementuje rozhranie, prostredníctvom ktorého bude schopná čítať koordináty v priradenej službe. Táto trieda je napojená na konkrétnu udalosť vyvolanú používateľom.

```
public LocationManager locationManager;
private CarTrackingLocationListener locationListener;

@Override
public void onCreate() {

  locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
  locationListener =  new CarTrackingLocationListener();

  locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,1,1,
                                         locationListener);

   }
```

Samotná trieda s príslušným rozhraním, ktorá rieši logiku v prípade získania novej pozície vyzerá nasledovne. Potrebujeme prekryť metódu *onLocationChanged* a parameter metódy reprezentuje objekt typu *Location*, z ktorého je možné sa dotazovať na všetky poskytované informácie. V tele metódy sa implementuje vlastná logiku vývojára. V systéme CarTrackingSystem sa vytvorí objekt typu *MapPoint*, ktorému sú následne nastavené a zapísané hodnoty do databázy.

```
private class CarTrackingLocationListener implements LocationListener {

  MapPoint currentMapPoint;
  DatabaseHanler dh = new DatabaseHandler(CarTrackingService.this);

  public void onLocationChanged(Location location) {

    if (location != null) {
        currentMapPoint = new MapPoint(df.format(location.getLatitude()),
                                       df.format(location.getLongitude()),
                                       String.valueOf(location.getSpeed()),
                                       trackId);

        dh.addMapPoint(currentMapPoint);
        }
    }
}
```

4.3.3 Haversinusova rovnica

Prostredníctvom mobilnej aplikácie informačného systému sú zaznamenávané body, ktoré sú jednoznačne identifikovateľné dvojicou údajov. Túto dvojicu tvorí pre každý bod zemepisná šírka a dĺžka. GPS modul jednotlivých zariadení vracia získané údaje ako desatinné číslo s presnosťou 10^{-5}.

Problém spočíva v tom, že samotný GPS modul je schopný pre konkrétny bod poskytnúť len údaje ako zemepisná šírka a dĺžka, nadmorská výška a rýchlosť pohybu v danom bode. Pre prípad monitorovania pohybu je potrebné vypočítať vzdialenosť, ktorú monitorovaná trasa zobrazuje. Haversinusova rovnica predstavuje algoritmus, využívaný pri navigácii s ohľadom na zemský povrch. Jej podstatou je výpočet najkratšej vzdialenosti medzi dvoma ľubovoľnými bodmi určenými zemepisnou šírkou a dĺžkou na povrchu Zeme. Opisuje vyberaný prípad sférického trigonometrického vzorca, problémom ktorého sú uhly a strany podobných trojuholníkov[25]. Meno metódy je odvodené z funkcie, ktorá sa pri jeho výpočte používa:

$$\operatorname{haversin}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1-\cos(\theta)}{2}$$

Funkcia je využívaná v samotnej rovnici, ktorá je použitá pre výpočet vzdialenosti

dvoch bodov na sférickej ploche:

$$\text{haversin}\left(\frac{d}{R}\right) = \text{haversin}\left(\phi_2 - \phi_1\right) + \cos\left(\phi_1\right)\cos\left(\phi_2\right)\text{haversin}\left(\lambda_2 - \lambda_1\right)$$

kde jednotlivé vstupy do rovnice predstavujú:

d - počítaná vzdialenosť dvoch bodov

R - predstavuje vzdialenosť na sférickom povrchu, v našom prípade polomer Zeme

ϕ_1 - zemepisná šírka bodu 1

ϕ_2 - zemepisná šírka bodu 2

λ_1 - zemepisná dĺžka bodu 1

λ_2 - zemepisná dĺžka bodu 2

Podmienkou pri použití Haversinusovej rovnice je premeniť jednotky vstupných bodov na radiány z dôvodu zachovaniu integrity údajov pri goniometrických výpočtoch. Nasledujúci zdrojový text dokumentuje implementáciu Haversinovej rovnice v jazyku Java.

```
public Double distance(Double latitudeA, Double latitudeB,
                       Double longitudeA, Double longitudeB) {

  double radius = 3958.75;
  double dlat = ToRadians(Double.parseDouble(String.valueOf(latitudeB)) -
                          Double.parseDouble(String.valueOf(latitudeA)));

  double dlon = ToRadians(Double.parseDouble(String.valueOf(longitudeB)) -
                          Double.parseDouble(String.valueOf(longitudeA)));

  double a = Math.sin(dlat / 2)
           * Math.sin(dlat / 2)
           + Math.cos(ToRadians(Double.parseDouble(String.valueOf(latitudeA))))
           * Math.cos(ToRadians(Double.parseDouble(String.valueOf(latitudeB))))
           * Math.sin(dlon / 2)
           * Math.sin(dlon / 2);

  double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
  Double d = radius * c;
  double meterConversion = 1609.00;

  return d * meterConversion;
}

private static double ToRadians(double degrees) {
  return degrees * 3.1415926535897932385 / 180;
}
```

Volanie tejto metódy prebieha spôsobom implementovaným v nasledujúcej úkažke zdrojového textu, kde výsledná vzdialenosť je uložená v premennej result.

```
List<MapPoint> mapPoints = dh.getMapPointsToTrack(trackId);

for(int i=0; i<(mapPoints.size()-1); i++) {

  Double latA = (Double) df.parse(mapPoints.get(i).getLatitude());
  Double lonA = (Double) df.parse(mapPoints.get(i).getLongitude());

  Double latB = (Double) df.parse(mapPoints.get(i+1).getLatitude());
  Double lonB = (Double) df.parse(mapPoints.get(i+1).getLongitude());

  tempDis = distance(latA, latB, lonA, lonB);

  result += tempDis;

  }
```

4.3.4 Komunikácia so serverom

V kapitole 3.1 pre návrh architektúry už bolo spomenuté, že bude orientovaná na služby. Základným komunikačným kanálom je protokol HTTP a pri výbere návrhu konkrétnej implementácie v spomínanej kapitole sme sa rozhodli pre použitie webových služieb v distribuovanom prostredí REST z niekoľkých dôvodov. Jedná sa o modernú alternatívu k staršiemu spôsobu SOAP, z čoho vyplýva, že je navrhnutá jednoduchšie. V dnešnej dobe to je kľúčovou vlastnosťou. Okrem toho REST služby sú prispôsobené väčšiemu využitiu bez zbytočného množstva rôznych štandardov ako v prípade SOAP. Podstatným rozdielom je prípad, ktorý sa len ťažko dá nasimulovať v prípade SOAP, a tým je odpoveď, ktorá má v sebe zapísané odkazy na iné údaje alebo dotazy vo forme URL.

Android poskytuje natívny spôsob pre použitie webových služieb. V tomto prípade je potrebné vhodne navrhnúť spracovanie formou asynchrónnych úloh a správne implementovať manažment jednotlivých vlákien v rámci životného cyklu aplikácií. Po krátkom prieskume domény ohľadom implementácie webových služieb v mobilných zariadeniach bola objavená technológia *Android Annotations*. Táto technológia posúva implementáciu prostredníctvom anotácií na novú úroveň a to vo viacerých

smeroch. Jedným z nich sú práve spomínané RESTful webové služby.

Základným krokom je vytvorenie rozhrania pre klienta spravujúce metódy webových služieb na strane Androidu. Ako príklad je uvedená metóda, ktorá bude využívať službu pre kontrolu, či je daný používateľ registrovaný v systéme.

```
@Rest(rootUrl = "http://cartrackingsystem.bitnamiapp.com/CarTrackingSystem/rest",
                converters = { MappingJackson2HttpMessageConverter.class})
public interface WSClient {

  @Get("/users/checkUser/{user_email}")
  @Accept(MediaType.APPLICATION_JSON)
  UserMessage checkUser(String user_email);
```

Anotáciou *@Rest* a jej parametrami nastavíme základné vlastnosti pre dotazovanie služby, akými sú URL adresa umiestnenia webovej služby a trieda, ktorá zabezpečí dodefinovanie šablóny pre serializáciu posielaného objektu. Anotáciou *@Get* definujeme typ operácie z HTTP protokolu a príslušný sufix URL adresy. V našom prípade je použitý aj vstupný parameter - emailová adresa prihláseného používateľa. Anotáciou *@Accept* filtrujeme typy objektov, ktoré môžu byť súčasťou komunikácie. V uvedenom príklade teda bude objekt typu *VehicleContainer* serializovaný do súboru formátu JSON v metóde *vehicles*.

Následné volanie tejto metódy je potrebné podľa pravidiel vývoju pre platformu Android použiť mimo hlavného vlákna. Výhodou použitej knižnice *Android annotations* je, že nie je potrebné vytvárať asynchrónnu úlohu pre samotné volanie, ale stačí anotovať metódu, v ktorej je webová služba použitá anotáciou *@Background*. Samotný klient komunikujúci s webovou službou je inicializovaný anotáciou zobrazenou v nasledujúcom príklade.

```
@RestService
WSClient wsclient;

@Background
void sync() {
  UserMessage result = wsclient.checkUser(googleAcc.name);
  ...
  }
```

Na strane servera je implementovaná rovnaká metóda, realizujúca danú logiku, pri-

čom výsledok poskytne vo forme služby, na ktorú sa mobilný klient dotazuje.

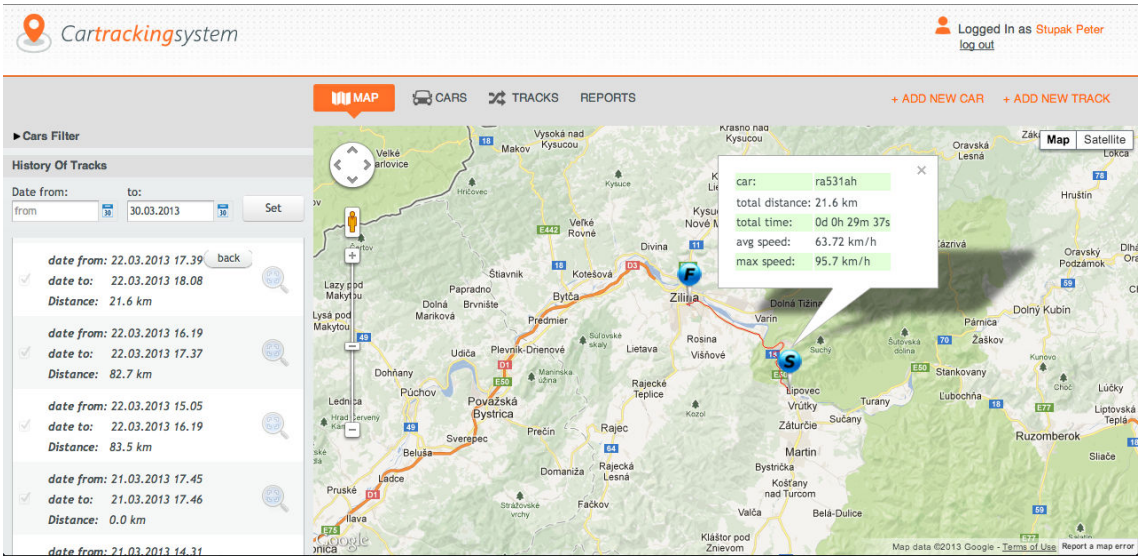
```
@GET
@Path("/checkUser/{email}")
@Produces(MediaType.APPLICATION_JSON)
public  UserMessage checkUser(@PathParam("email") String email) {

  UserMessage userMessage = new UserMessage();
  try {
    Person person = personService.getPersonByEmail(email);
    if(person == null) {
        userMessage.setHasUser(false);
      } else {
        userMessage.setHasUser(true);
      }
  } catch (CarTrackingException e)  {
    userMessage.setHasUser(false);
  }
    return userMessage;
  }
```

5 Verifikácia implementovaného riešenia

5.1 Verifikácia algoritmu pre výpočet vzdialenosti

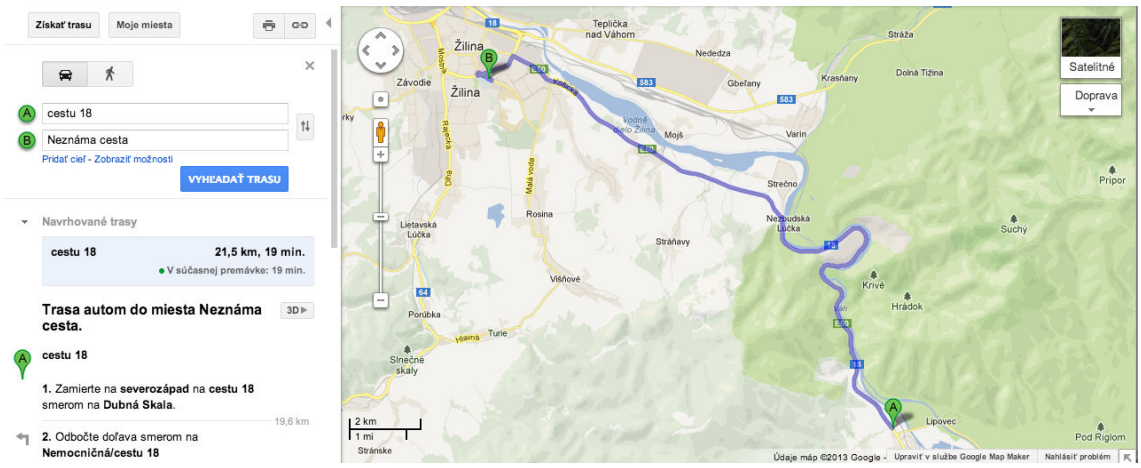
V kapitole 4.3.3 je podrobne analyzovaná a popísaná definícia Haversinusovej rovnice. Na základe danej analýzy bola implementovaná v jazyku Java a použitá v mobilnej aplikácii pre výpočet celkovej vzdialenosti monitorovaných trás. Táto kapitola sa venuje overeniu korektnosti danej implementácie pre výpočet celkovej vzdialenosti pre rovnakú trasu prostredníctvom rozdielnych nástrojov.

Prvý výsledok je odvodený z webovej aplikácie systému CarTrackingSystem a zdokumentovaný na obrázku5 – 1.



Obr. 5 – 1 Príklad trasy monitorovanej aplikáciou CarTrackingSystem

Systém CarTrackingSystem vyhodnotil vzdialenosť medzi bodmi označenými na obrázku o dĺžke 21,6 km. Rovnaká trasa je teraz nasimulovaná prostredníctvom služby maps ponúkanou spoločnosťou Google. Nasledujúci obrázok5 – 2 zobrazuje službu maps s rovnakou trasou, aká bola vyhodnocovaná systémom CarTrackingSystem.



Obr. 5 – 2 Príklad trasy zobrazenej službou `maps.google.com`

Služba `maps.google.com` vyhodnotila približne rovnakú trasu, nakoľko začiatočný a koncový bod bol určený experimentálne, vo vzdialenosti 21,5 km. Z daného merania je možné vypočítať smerodajnú odchýlku:

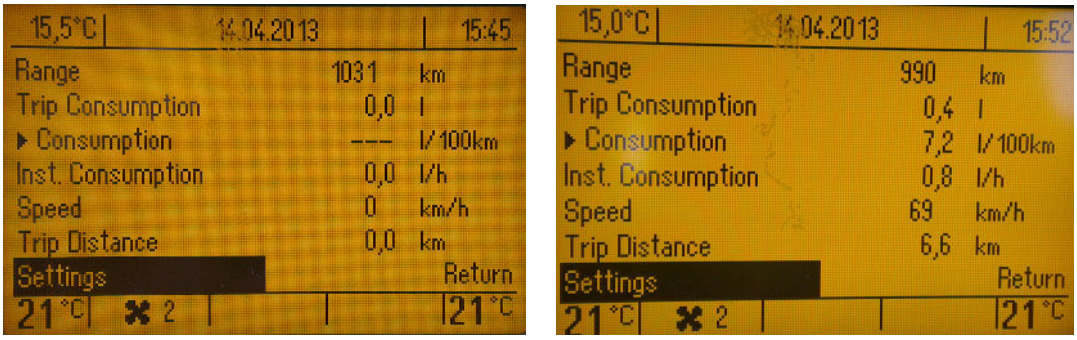
$$\mu = \frac{21,6 + 21,5}{2} = 21,55; \qquad \Delta = \frac{21,6 - 21,5}{21,55} * 100 \doteq \underline{0,46\%}$$

Výsledné riešenie sa v prípade, že za vzor bude považovaná spomínaná služba, líši o 0,46%.

5.2 Verifikácia funkčnosti mobilnej aplikácie

Nasledujúcim krokom verifikácie riešenia je overenie celkovej funkčnosti aplikácie. Pre priblíženie, aplikácia je schopná získavať koordináty bodov prostredníctvom GPS modulu mobilného zariadenia. Každý bod tvorený koordinátmi obsahuje informáciu o aktuálnej rýchlosti v konkrétnom bode. Prostredníctvom zoznamu údajov o rýchlostiach, aplikácia vypočíta priemernú a maximálnu rýchlosť pohybu daného vozidla na konkrétnej trase. Okrem toho, zariadenie počíta celkovú vzdialenosť trasy, ktorá bola prejdená monitorovaným vozidlom. Funkcia je implementovaná prostredníctvom algoritmu opísaného v kapitole 4.3.3 a jej korektnosť implementácie overená predošlou kapitolou.

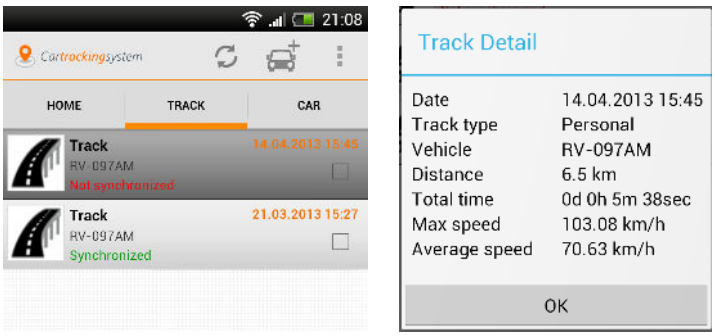
V predchádzajúcej kapitole je riešenie overené vzhľadom na konkurenčnú službu. Táto kapitola bude overovať riešenie prostredníctvom porovnania hodnôt získaných palubným počítačom vozidla, voči hodnotám vypočítaných mobilným zariadením. Pre zachovanie integrity údajov je na obrázku 5 – 3(a) a 5 – 3(b) možné zaznamenať dátum a čas vykonávania testovacieho procesu. Pre získanie čo najkorektnejších údajov bolo potrebné palubný počítač vozidla vynulovať a začať meranie bez akýchkoľvek rušivých elementov.



(a) Začiatok merania (b) Koniec merania

Obr. 5 – 3 Hodnoty získané z palubného počítača

V čase vynulovania údajov palubného počítača bol paralelne aktivovaný proces, ktorý spúšťa monitorovaciu logiku. Tento spôsob zabezpečí rovnaké vstupné údaje pre konkrétne zariadenia.



(a) Výber trasy (b) Zobrazenie detailu

Obr. 5 – 4 Hodnoty získané z mobilnej aplikácie

Po ukončení monitorovacieho procesu v mobilnej aplikácii je prostredníctvom záložky *Track* zobrazený detail konkrétnej trasy. Dlhým podržaním na riadok s trasou je zobrazené kontextové menu, v ktorom je potrebné zvoliť možnosť *Detail*. Následne sa v dialógovom okne vypíšu údaje vypočítané pre monitorovanú trasu.

Pri porovnaní hodnôt na obrázkoch 5 – 3 a 5 – 4 dôjdeme k záveru, že odchýlky sú minimálne.

V prípade celkovej vzdialenosti je smerodajná odchýlka:

$$\mu = \frac{6,6 + 6,5}{2} = 6,55; \qquad \Delta = \frac{6,6 - 6,5}{6,55} * 100 \doteq \underline{0,2\%}$$

Pre priemernú rýchlosť na danej trase platí smerodajná odchýlka:

$$\mu = \frac{70,63 + 69,00}{2} = 69,815; \qquad \Delta = \frac{70,63 - 69,00}{69,815} * 100 \doteq \underline{2,19\%}$$

Treba však podotknúť, že palubný počítač pri testovacom vozidle mal priemernú rýchlosť vyjadrenú bez desatinnej presnosti.

Palubný počítač v testovacom vozidle neposkytoval informáciu o maximálnej rýchlosti pre monitorovaný úsek, preto nebolo možné overiť korektnosť tejto hodnoty získanú z mobilného zariadenia.

Celkový čas bol palubným počítačom reprezentovaný len na minútovú presnosť, preto nie je možné adekvátne dokázať, že sa jedná o rovnaké časové obdobie. Navyše treba prirátať čas, ktorý zodpovedá trvaniu deaktivácie procesu monitorovania a následného zhotovenia fotografie rovnakým zariadením.

6 Záver (zhodnotenie riešenia)

Výsledkom diplomovej práce je informačný systém zaoberajúci sa monitorovaním a evidenciou pohybu vozidiel s využitím prostriedkov globálneho systému pre monitorovanie. Práca predstavuje detailný opis od fázy analýzy, z nej vychádzajúceho návrhu a dokumentovanej implementácie pomocou moderných programovacích technológií použitých na základe výsledkov z krátkeho intenzívneho výskumu. Výsledným produktom je funkčný prototyp webovej aplikácie na platforme Java, na vývoj ktorej bol použitý aplikačný rámec Vaadin a taktiež funkčný prototyp mobilnej aplikácie na platforme Android. S využitím architektúry orientovanej na služby je medzi týmito modulmi zabezpečená komunikácia prostredníctvom webových služieb s implementáciou REST.

Význam prezentovanej práce spočíva v návrhu riešenia, ktoré sa bude v určitom smere diferencovať od existujúcich systémov. Bolo spomenuté, že hlavnou výhodou oproti systémom, ktoré poskytujú pre monitorovacie účely okrem informačného systému aj vlastný hardvér, je použitie technického vybavenia obsiahnutého v inteligentnom mobilnom telefóne, čo vedie predovšetkým k zníženiu investičných nákladov. Treba podotknúť, že poskytnuté riešenie je obmedzené na použitie posledných špecifikácií operačného systému Android, čo nepredstavuje až takú nevýhodu s ohľadom na relatívne časté aktualizácie pre jednotlivé zariadenia.

Testovaním systému bola overená jeho funkčnosť a použiteľnosť na produkčnom prostredí. Experimentálne výsledky získané z aplikácie boli vyhodnotené a v porovnaní s konkurenčnými systémami produkujú zanedbateľnú smerodajnú odchýlku. Riešenie poskytuje niekoľko návrhov pre zlepšenie v ďalších fázach existencie. V prvom rade je potrebné špecifikovať zabezpečenie pre komunikačný kanál webových služieb. Nevyhnutným krokom pre systémy podobného charakteru je poskytnúť monitorovanie reálneho času alebo funkciu odcudzeného vozidla, ktoré neboli implementované kvôli časovému nedostatku.

Literatúra

[1] GARTNER: *Analytická agentúra*. Stamford, USA dostupné na internete: `http://www.gartner.com/newsroom/id/2120015` Aktualizované 2012-14-8, [cit. 2012-12-26]

[2] ROGERS, R. - LOMBARDO, J. - MEDNIEKS, Z. - MEIKE, B.: *Android Application Development.*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2009, ISBN: 978-0-596-52147-9

[3] ANDROID DEVELOPERS COMMUNITY: *Platform Versions Dashboards*. dostupné na internete: `http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels` [cit 2012-12-26]

[4] EMBEDDED LINUX WIKI: *Android Kernel Features*. eLinux. Aktualizované 2011-12. dostupné na internete: `http://elinux.org/Android-kernel-features` [cit. 2010-11-23]

[5] SÁZEL, V.: *Vývoj aplikácií na platforme Android*. Unicorn College, V Kapslovně 2 767/2, Praha 3, 13000

[6] KOMATINENI, S. - HASHIMI, S.: *Pro Android*. Apress, 24. June 2009, 1st edition ISBN-10: 1430215968 — ISBN-13: 978-1430215967

[7] VANDERKA, M.: *Platforma Android*. Posterus, Portál pre odborné publikovanie, 2011 ISSN 1338-0087

[8] WIKIPEDIA: *iOS version history*. Current version. dostupné na internete: `http://en.wikipedia.org/wiki/IOS_version_history#Current_versions` Aktualizované 2010-1-12. [cit. 2013-2-16]

[9] MAC DEVELOPER LIBRARY: *The Cocoa Enviroment*. How Cocoa Fits into OS X. dostupné na internete: `http://developer.apple.com/library/Mac/`

#documentation/Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html Aktualizované 2010-12-13, [cit 2013-2-17]

[10] APPLE DEVELOPER: *iOS developer program*. The fastest path from code to customer.dostupné na internete: https://developer.apple.com/programs/ios/develop.html [cit. 2013-2-17]

[11] HOLLISTER, S.: *Microsoft prepping Windows Phone 7 for an October 21 launch?*. Engadget. AOL. http://www.engadget.com/2010/09/26/microsoft-prepping-windows-phone-7-for-an-october-21st-launch/ Aktualizované 2010-9-26, [cit 2013-2-19]

[12] LACKO, L.: *Vývoj aplikácií pre Windows Phone 7*. MSDN, Brožúry s vývojárskou tematikou, 2011

[13] WARREN, T.: *Windows Phone Update*. Winrumors. dostupné na internete: http://www.winrumors.com/microsoft-rolling-out-windows-phone-update-to-fix-exchange-and-voicemail-bugs/ Aktualizované 2011-11-17, [cit. 2013-2-23]

[14] MICROSOFT NEWS CENTER: *Nokia and Microsoft Announce Plans for a Broad Strategic Partnership to Build a New Global Mobile Ecosystem*. dostupné na internete: http://www.microsoft.com/en-us/news/press/2011/feb11/02-11partnership.aspx Aktualizované 2011-2-10, [cit 2013-2-23]

[15] WINDOWS PHONE INTEROPERABILITY: *Android to WP7*. dostupné na internete:http://windowsphone.interoperabilitybridges.com/articles/android-to-wp7-chapter-1-introducing-windows-phone-platform-to-android-application-developers Aktualizované 2011-6-3, [cit. 2013-2-24]

[16] MILOSHEVKA, B.:*Silverlight for Windows Phone Toolkit in Depth*. Windows

Phone Geek

[17] ERL T.: *Web Service Contract Design and Versioning for SOA*. Prentice Hall, 12. August 2005, ISBN-10: 0131858580 — ISBN-13: 978-0131858589

[18] WORLD WIDE WEB CONSORTIUM: *Messaging Framework*. SOAP Version 1.2 Part 1, second edition. dostupné na internete: `http://www.w3.org/TR/soap12-part1/` Aktualizované 2007-4-27, [cit. 2013-3-14]

[19] RICHARDSON, L.: *RESTful Web Services*. O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2009, 15. Máj 2007, ISBN-10: 0596529260 — ISBN-13: 978-0596529260

[20] RIEHLE, D.: *Framework design*. A Role Modeling Approach. ACM Press, 2000, DOI: 10.3929/ethz-a-003867001

[21] MIO: *What is trilateration*. GPS Explained, dostupné na internete: `http://eu.mio.com/en_gb/global-positioning-system_what-is-trilateration.htm` Aktualizované 2012, [cit. 2013-3-28]

[22] TUTORIALSPOINT: *Spring Framework Architecture*. Spring tutorial in PDF, 2013, dostupné na internete: `http://www.tutorialspoint.com/spring/spring_architecture.htm`

[23] GRÖNROOS, M.: *Book of Vaadin*. Vaadin 7th edition. Vaadin Ltd., 2013, ISBN 978-952-92-6754-5

[24] MOCKO, M.: *Spring core*. Aplikačný framework Spring, 2007

[25] ABRAMOWITZ, M. - STEGUN, I.: *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Courier Dover Publication, New York, 1964, ISBN: 9780486158242

Zoznam príloh

Príloha A CD médium - diplomová práca v elektronickej podobe, prílohy v elektronickej podobe, zdrojové kódy

Príloha B Používateľská príručka

Príloha C Systémová príručka

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Systém pre sledovanie a evidenciu pohybu motorových vozidiel II

Príloha B: Používateľská príručka

2013 Ladislav Szalai

7 Používateľská príručka

Táto kapitola opisuje používateľskú príručku navrhnutého, analyzovaného a implementovaného informačného systému v diplomovej práci. Jej súčasťou je rozbor statického aj mobilného klienta z pohľadu funkcie, požiadaviek a základných prípadov použitia zobrazených na ukážkach z aplikácií.

7.1 Funkcia aplikácie

Informačný systém CarTrackingSystem predstavuje praktický výstup z diskutovaného riešenia, ktoré je súčasťou diplomovej práce. Primárnym cieľom je vytvoriť komplexný systém pre sledovanie a evidenciu pohybu vozidiel. Informačný systém je zložený z dvoch samostatných častí, podliehajúcich architektúre orientovanej na služby. Prvú časť poskytuje webová aplikácia reprezentujúca serverovú časť, ktorá zastáva evidenčnú a vyhodnocovaciu úlohu. Druhou časťou je mobilná aplikácia určená pre platformu Android, predstavujúca nosnú časť tohoto konceptu. Prostredníctvom vstavaného GPS modulu zabezpečuje zber údajov pre vyhodnocovanie polohy vozidla.

Rozšírenú funkcionalitu implementuje najmä webová aplikácia, ktorá poskytuje na základe údajov získaných mobilným telefónom vykreslenie bodov na mapovom podklade, ktoré spája do trás. Každá trasa má uložené údaje, z ktorých je možné vypočítať štatistiky rôznych kategórií. Z týchto údajov je možné generovať výkazy v podobe knihy jázd pre konkrétne vozidlá, prípadne trasy. Rozšírenú funkcionalitu mobilnej aplikácie predstavuje pridávanie tankovacích bodov. Používateľ zaznamená manuálne prostredníctvom rozhrania počet jednotiek načerpaného paliva a jednotkovú cenu pre liter paliva. Aplikácia získa koordináty polohy a ukladá entitu.

7.2 Súpis obsahu dodávky

Táto kapitola obsahuje zoznam súborov, ktoré sú dodávané na vymeniteľnom médiu spolu s prácou. Na základe postupov uvedených v kapitole 7.4 je možné prostredníctvom týchto súborov systém nainštalovať a používať.

- SRC

  CarTrackingSystem.rar - komprimované maven moduly webovej aplikácie

  CarTrackinSystemAndroid.rar - komprimovaný projekt mobilnej aplikácie

- BIN

  CarTrackingSystem.war - komprimovaný súbor webovej aplikácie pre server

  CarTrackingSystem.apk - aplikačný balíček pripravený na inštaláciu mobilnej aplikácie

- DOC

  DiplomováPráca - priečinok obsahujúci prácu v natívnom formáte pre Latex

  DiplomováPráca.pdf - diplomová práca vo formáte .pdf

7.3 Požiadavky aplikácie

Presné požiadavky pre programové a aplikačné vybavenie nie je možné určiť. Pre tieto potreby budeme vychádzať z konfigurácií, na ktorých bol informačný systém vyvíjaný a testovaný. Konkrétne špecifikácie pre webovú aj mobilnú aplikáciu sú opísané v nasledujúcich sekciách.

7.3.1 Hardvérové požiadavky

Webová aplikácia

- Hardvér prispôsobený konfigurácii aplikačného serveru Apache Tomcat
- Internetové pripojenie

Mobilná aplikácia

- Mobilné zariadenie s operačným systémom Android
- Displej - minimálna veľkosť 4.3“ alebo 480x800 pixelov
- GPS modul ako súčasť mobilného zariadenia
- Dostupné internetové pripojenie pre prípad synchronizácie
- Operačná pamäť - minimálne 50mb voľnej pamäte

7.3.2 Softvérové požiadavky

Webová aplikácia

- Aplikačný server Jetty nakonfigurovaný na zariadení serveru
- Internetový prehliadač pre prístup k aplikácii

Mobilná aplikácia

- Operačný systém Google Android verzie 4.1 alebo vyšší
- Povolená podpora pre inštaláciu aplikácií tretích strán

7.4 Inštalácia aplikácie

7.4.1 Webová aplikácia

Webová aplikácia je inštalovaná prostredníctvom konfigurátora aplikačného serveru Apache Tomcat. V priečinku /bin dodaných súborov je možné lokalizovať súbor CarTrackingSystem.war. Súbor je potrebné umiestniť do priečinka /webapps, ktorý sa nachádza v domovskom priečinku Tomcat servera. Po tomto kroku musíme zabezpečiť ešte prepojenie Apache servera s Tomcat serverom, na ktorom bude spustená aplikácia. Toto prepojenie nastavíme pridaním nasledovnej konfigurácie do súboru httpd.conf umiestneného na serveri:

```
<VirtualHost *:80>
    ServerName cartrackingsystem.bitnamiapp.com
    ServerAlias cartrackingsystem.bitnamiapp.com
    <Location />
        ProxyPass ajp://localhost:8009/CarTrackingSystem/
    </Location>
</VirtualHost>
<Location /CarTrackingSystem>
    ProxyPass ajp://localhost:8009/CarTrackingSystem
</Location>
```

Po vykonaní všetkých krokov je potrebné server spustiť alebo reštartovať a aplikácia bude sprístupnená na URL adrese servera.

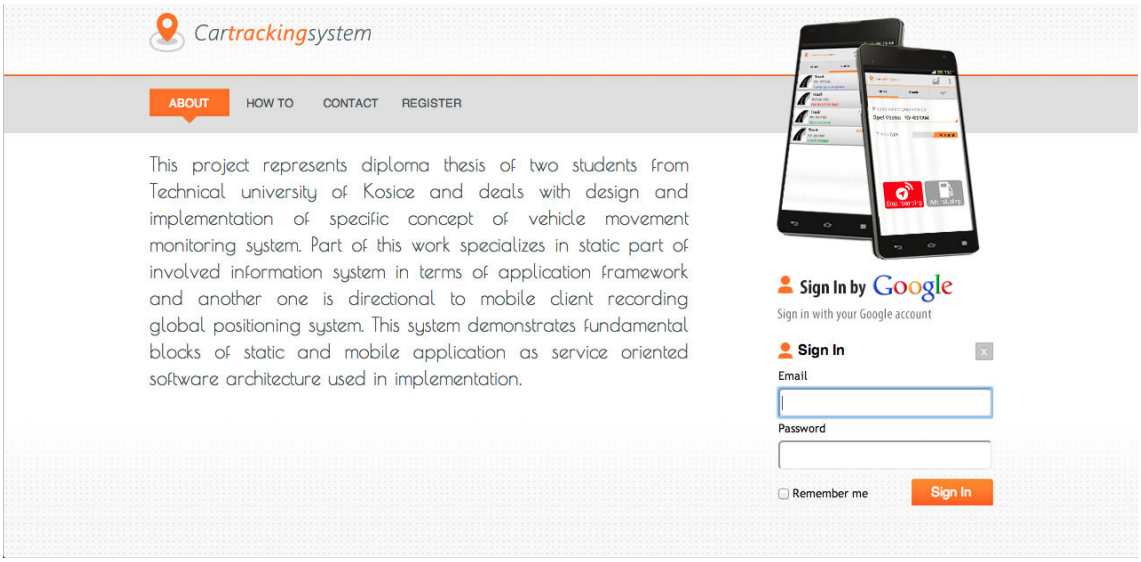
7.4.2 Mobilná aplikácia

Rovnako v priečinku /bin dodaných súborov nájdeme súbor CarTrackingSystem.apk a je nevyhnutné ho nakopírovať prostredníctvom USB rozhrania alebo umiestnením na úložisko dostupné z mobilného zariadenia do pamäte tohoto zariadenia. Za predpokladu, že má systém povolenú inštaláciu softvéru tretích strán, sa otvorením tohoto súboru a následným potvrdením inštalácie aplikácia nainštaluje. V ďalších krokoch sa aplikácia spustí z ponuky.

7.5 Použitie informačného systému

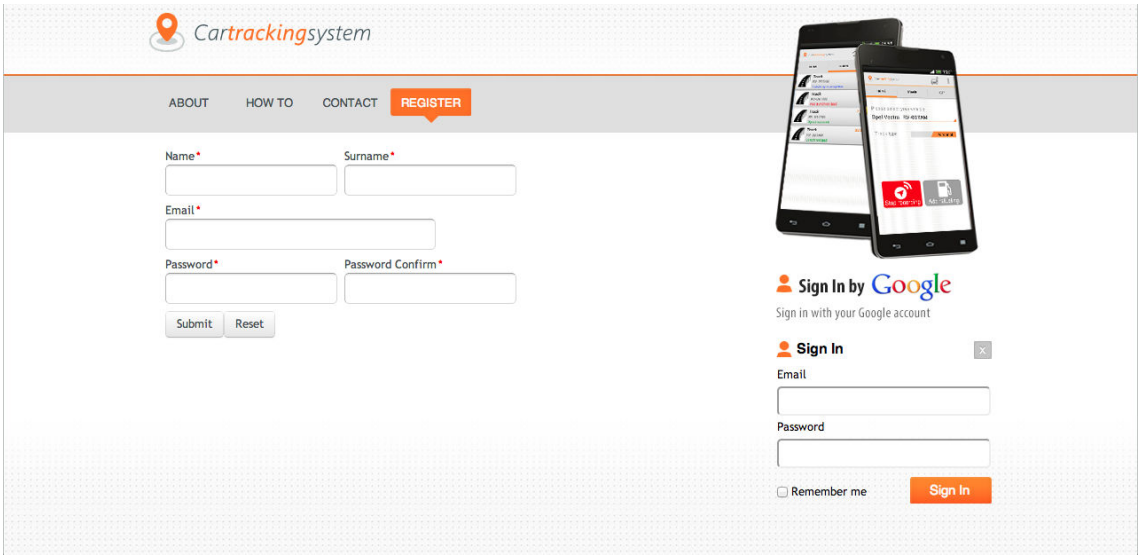
7.5.1 Použitie webovej aplikácie

Po spustení aplikácie v prehliadači na korektnej URL adrese nás privíta úvodná obrazovka(obr. 7 – 1) so základnými údajmi o systéme. Súčasťou stránky je tiež prihlasovací formulár. Prihlásenie je možné prostredníctvom účtu google alebo vyplnením údajov vo formulári, ktorý ale vyžaduje pri prvom spustení registráciu používateľa. Registračný formulár je zobrazený na obrázku 7 – 2.

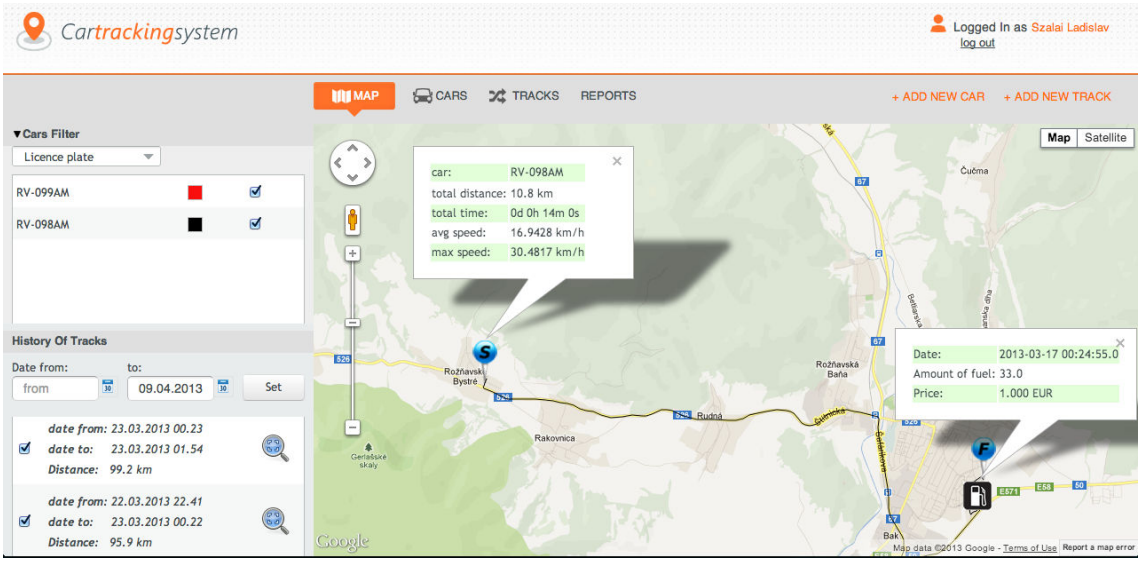


Obr. 7 – 1 Úvodná obrazovka pre prihlásenie do systému

Po úspešnej autentifikácii je používateľ presmerovaný do hlavnej časti aplikácie a je mu zobrazený mapový podklad (obr. 7 – 3), na ktorom má vykreslené trasy z predošlých monitorovaní. Tieto údaje môže filtrovať podľa vozidiel, dátumu alebo aj jednotlivých tratí v ľavom aplikačnom menu. Každá trať je zobrazená štartovacím a koncovým bodom, ktoré zobrazujú najzákladnejšie údaje o trase po kliknutí na ich ikonu. V prípade, že monitorovaná trasa obsahuje tankovací bod, tak je súčasťou vykreslenej trasy v podobe príslušnej ikony. Je možné na neho rovnako kliknúť a zobraziť údaje týkajúce sa tankovania.



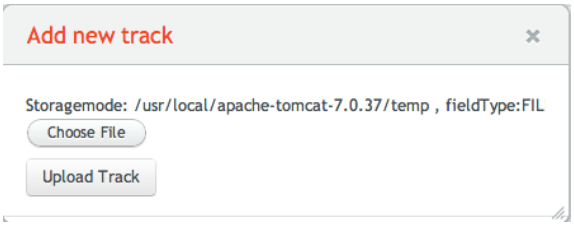
Obr. 7 – 2 Registračný formulár



Obr. 7 – 3 Zobrazenie monitorovaných trás na mapovom podklade

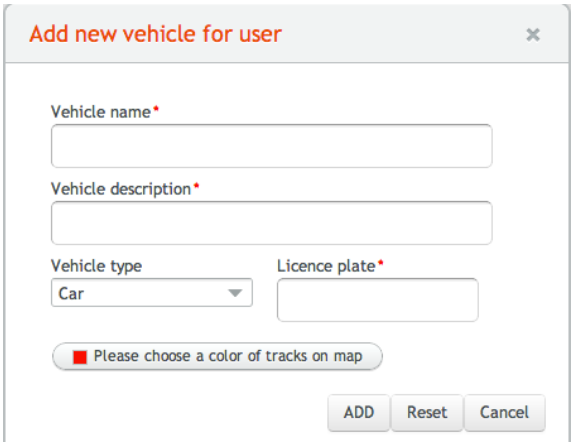
Údaje sa do aplikácie prijímajú dvomi spôsobmi. V prípade, že je používateľ prihlásený prostredníctvom google účtu, môže využiť synchronizačný proces v spolupráci s mobilným zariadením. Tento postup je popísaný obrázkom 7 – 11(a). Druhým spôsobom je manuálne pridanie špecifického súboru, ktorý je generovaný rovnako mobilným zariadením na lokálnu pamäť. Súbor je možné pridať rozhraním, ktoré je

spustené tlačidlom *+ADD NEW TRACK* a zobrazené na obrázku 7 – 4



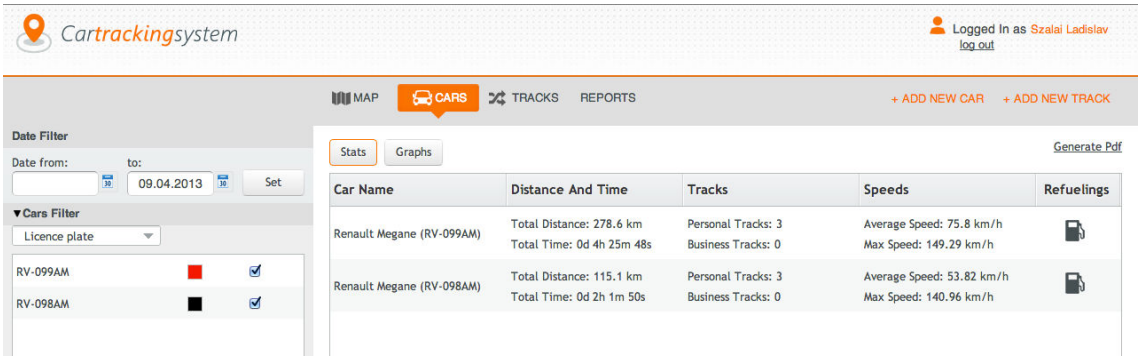
Obr. 7 – 4 Rozhranie pre manuálne pridanie trasy

Rovnakým spôsobom je možné manuálne pridať aj vozidlo, ktoré je v prípade použitia synchronizačného procesu pridávané automaticky. Tento spôsob zobrazuje obrázok 7 – 5



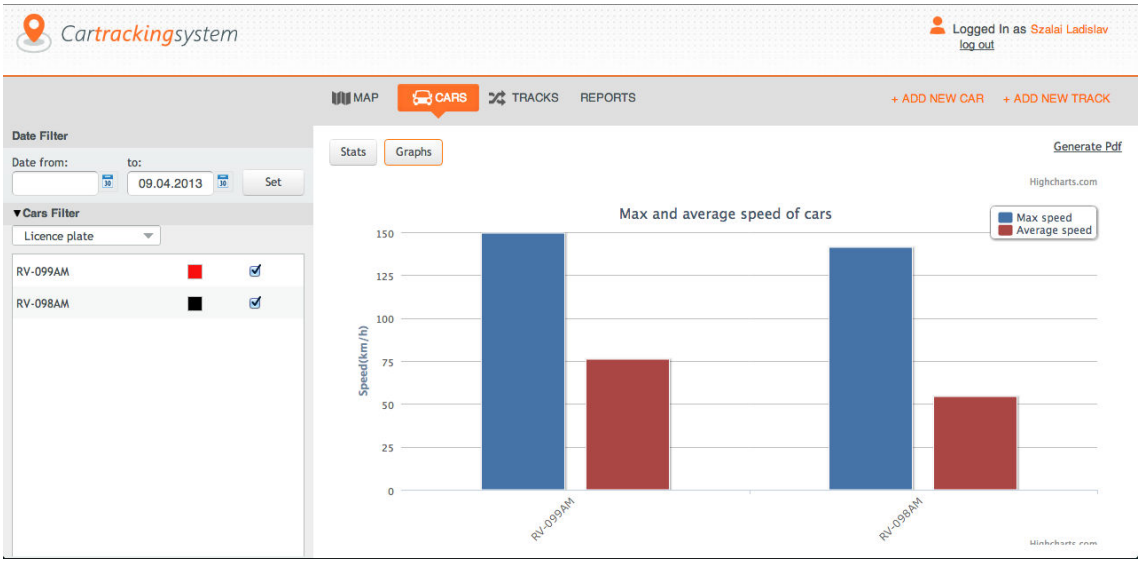
Obr. 7 – 5 Rozhranie pre manuálne pridanie vozidla

Rozšírenou možnosťou pre používateľa je filtrované zobrazenie zoznamu vozidiel, ktoré má registrované v systéme a ich jednotlivé štatistické údaje. Túto možnosť zobrazuje karta *CARS* na obrázku 7 – 6. Zobrazené údaje obsahujú celkovú prejdenú vzdialenosť, celkový čas monitorovania, rozdelenie tratí podľa účelu na súkromné a biznis, výpočet celkovej priemernej a maximálnej rýchlosti a zobrazenie tankovacích bodov v špeciálnom modálnom tabuľkovom rozhraní zobrazenom na obrázku 7 – 8. Aplikácia rovnakým spôsobom zobrazuje štatistiky pre monitorované trasy na karte *TRACKS*

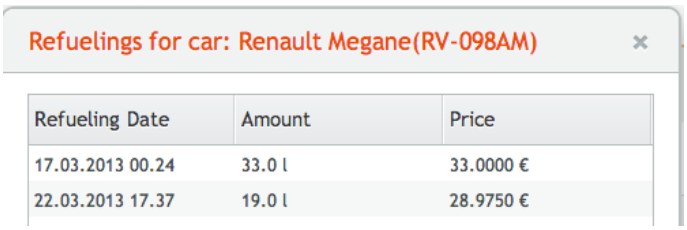


Obr. 7 – 6 Pohľad na štatistiky používateľových vozidiel

Webová aplikácia umožňuje zobrazenie štatistických výsledkov aj vo forme grafického zobrazenia. Túto možnosť deklaruje obrázok 7 – 7



Obr. 7 – 7 Pohľad na grafické zobrazenie štatistík o vozidlách

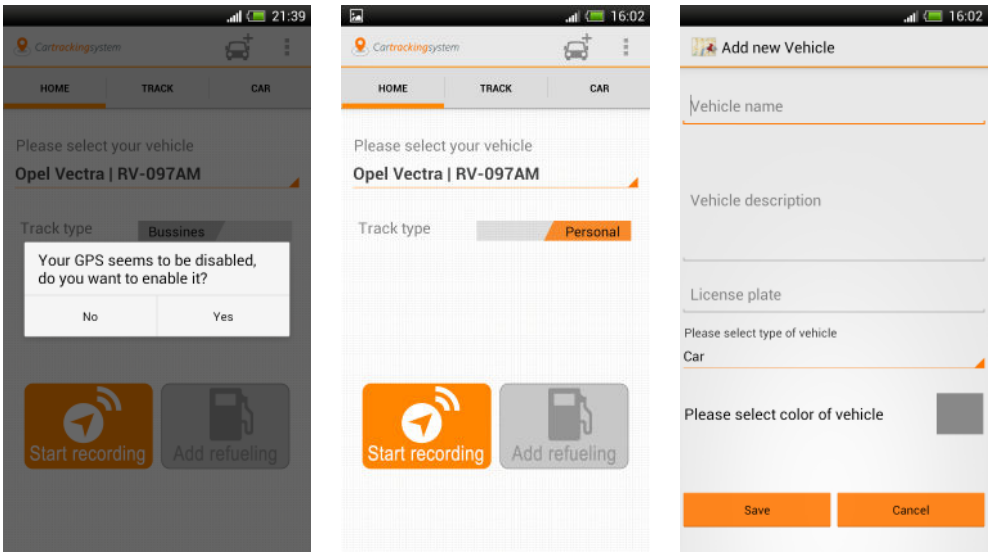


Obr. 7 – 8 Zobrazenie tankovacích bodov pre vozidlo

7.5.2 Použitie mobilnej aplikácie

Po spustení aplikácie sa kontroluje činnosť GPS modulu. V prípade, že je modul deaktivovaný, je používateľ na to upozornený a po potvrdení, že chce tento modul aktivovať, je presmerovaný na systémové nastavenia. Situácia je zobrazená na obrázku 7 – 9(a).

Pri prvom spustení je potrebné vytvoriť aspoň jedno vozidlo. Túto možnosť poskytuje horný panel nástrojov na hlavnej obrazovke 7 – 9(b)), ktorý obsahuje funkciu pridať vozidlo. Po jeho stlačení je používateľ presmerovaný na obrazovku, ktorá obsahuje formulár pre pridanie vozidla 7 – 9(c). Na ňom vyplní základné vlastnosti automobilu, pričom je potrebné dodržať jedinečný identifikátor pre štátnu poznávaciu značku. Následne vyberie farbu, ktorou bude vozidlo vykreslené na mape a uloží entitu.

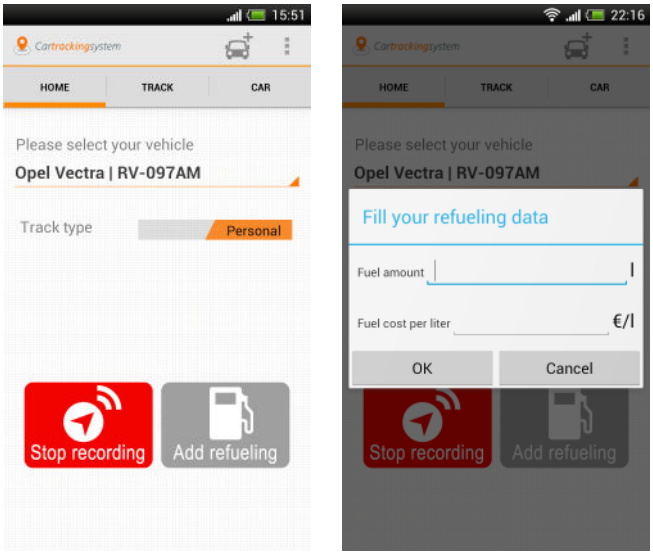


(a) Kontrola GPS modulu (b) Úvodná obrazovka (c) Pridanie vozidla

Obr. 7 – 9 Ukážky mobilnej aplikácie 1

Po uložení je používateľ presmerovaný na poslednú stránku, na ktorej sa nachá-

dzal pred použitím funkcie pridania vozidla. Ak už má používateľ vytvorené vozidlo, môže ho na hlavnej stránke vybrať z ponuky. Nasledujúcou možnosťou je vybrať typ trate, ktorý sa chystá monitorovať. Môže si vybrať z dvoch možností: Trasa pre súkromné alebo biznis účely. Ak sú predošlé podmienky splnené, je možné zahájiť monitorovanie pohybu. Pre túto činnosť je potrebné stlačiť tlačidlo *Start recording*. Po tomto kroku je možné pridávať tankovacie body pre konkrétnu trasu. Dovtedy je táto možnosť zablokovaná. Aplikácia prejde do stavu zobrazeného na obrázku 7 – 10(a).



(a) Spustené monitorovanie

(b) Pridanie tankovacieho bodu

Obr. 7 – 10 Ukážky mobilnej aplikácie 2

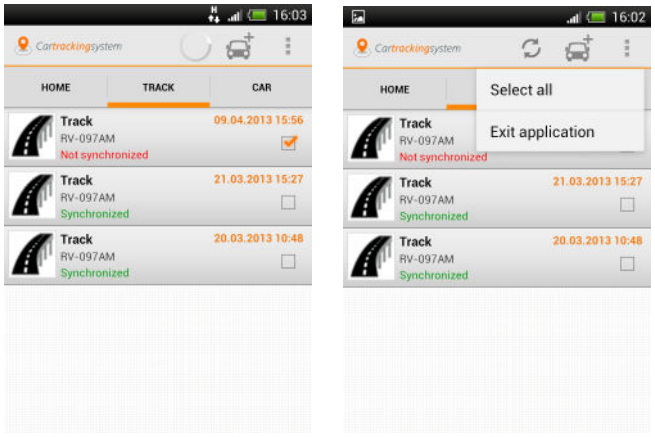
Ďalším prípadom použitia je pridanie tankovacieho bodu tlačidlom *Add refueling*. Po jeho stlačení musí používateľ manuálne vyplniť počet litrov a cenu za tankovaciu jednotku. Aplikácia uloží polohu k danému bodu a identifikátor trate, pre ktorú bol vytvorený. Prípad je znázornený na obrázku 7 – 10(b).

Samotné monitorovanie trasy je ukončené stlačením tlačidla *Stop recording*. Po jeho stlačení aplikácia vytvára v lokálnej databáze trasu, k nej uloží tankovacie body

a zároveň je ešte vygenerovaný v pamäti zariadenia súbor s koncovkou .kml, ktorý slúži pre prípad offline synchronizačného postupu. Zároveň je možné tento súbor načítať aj v službe maps.google.com.

Po vytvorení trasy je možné si ju zobraziť prepnutím aplikácie na kartu *Track* zobrazeného na obrázku 7 – 11(a). Po dlhom dotyku na konkrétnu trasu je možné vyvolať dialógové okno s detailnými údajmi o trase.

V prípade synchronizácie tratí s webovou aplikáciou je potrebné mať aktívne internetové pripojenie akéhokoľvek druhu. Túto funkciu zabezpečuje tlačidlo v hornom paneli nástrojov. Trate majú vlastnosť, ktorá určuje, či už boli synchronizované alebo nie. Synchronizujú sa len trate, ktoré majú vo vlastnostiach výpis *Not synchronized* a ktoré boli zvolené zaškrtnutím políčok. Po stlačení sa údaje posielajú do webovej aplikácie, kde sú trasy zobrazené a vykreslené na mapový podklad. Rovnakým spôsobom funguje synchronizácia vozidiel na karte *Car*.



(a) Zobrazenie a synchronizácia trás

(b) Korektné ukončenie aplikácie

Obr. 7 – 11 Ukážky mobilnej aplikácie 3

Samotná aplikácia sa korektne ukončuje výberom možnosti *Exit application* z hlavného menu aplikácie, ktorá je zobrazená na obrázku 7 – 11(b).

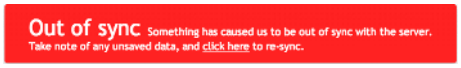
7.6 Chybové hlásenia

Počas interakcie s aplikáciou môže dôjsť v prípade nesprávneho zaobchádzania k výpisu chybových hlášok. Webová aplikácia kontroluje niekoľko takýchto situácií.

- V prípade, že používateľ použije prihlásenie do aplikácie formou registrovaného používateľského konta a vyplní nesprávne používateľské prihlasovacie údaje, tak sa objavuje nasledujúca chybová hláška

Obr. 7 – 12 Chybová hláška pri nesprávnych prihlasovacích údajoch

- Ďalším prípadom chybovej hlášky je nekonzistentnosť údajov v rovnakej relácii. Túto kontrolu zabezpečuje bezpečnostný nástroj rámca Spring, ktorý pri prihlásení vygeneruje bezpečnostný kľúč a v prípade prístupu z rovnakého prehliadača na inej karte je tento kľúč pregenerovaný, čím je narušená integrita údajov.



Obr. 7 – 13 Chybová hláška pri nekonzistentných údajoch

- Posledným prípadom chybových hlášok pre webovú aplikáciu je ukončenie relácie podľa časového nastavenia v konfiguračnom súbore. Pri pokuse o prístup po tejto dobe je vyvolávaná nasledujúca chyba.

Obr. 7 – 14 Chybová hláška pre časové ukončenie relácie

7.7 Obmedzenia

V aktuálnom stave informačného systému je niekoľko obmedzujúcich prípadov použitia. Obmedzenia sa týkajú najmä monitorovacej jednotky, čiže mobilného klienta. Problém predstavuje rozhranie, ktoré neumožňuje získať údaje o spotrebe aktuálne monitorovaného vozidla a tým vytvára priestor pre odchýlky v štatistických údajoch pre knihu jázd.

Nasledujúci prípad obmedzenia predstavuje vypĺňanie formulárových komponentov v rámci mobilnej aplikácie, ktorá nekontroluje číselné vstupy, ale údaje zaznamenáva ako reťazce znakov, ktoré konvertuje pri ich spracovaní. Tento prípad je v budúcnosti možné odstrániť návrhom a použitím vhodnejších databázových typov. V tom prípade bude možné implementovať aj plnohodnotnú validáciu všetkých formulárov.

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Systém pre sledovanie a evidenciu pohybu motorových vozidiel II

Príloha C: Systémová príručka

2013 **Ladislav Szalai**

8 Systémová príručka

Kapitola popisuje systémovú príručku implementovaného informačného systému. Venuje sa hlavne analýze a opisu dosiahnutého riešenia, spôsobu fungovania podstatných tried systému, opisu databázového modelu systému a návodu na preklad zdrojových textov.

8.1 Analýza riešenia

Systém slúži na monitorovanie pohybu vozidiel pomocou mobilnej aplikácie nainštalovanej na mobilnom telefóne s GPS modulom založenom na platforme Android. Mobilná aplikácia využíva na získavanie údajov o konkrétnych pozíciach vozidla GPS modul, ktorý musí byť súčasťou mobilného zariadenia a ukladá ich do internej databázy. Rozšírenou funkcionalitou je exportovanie .kml súboru na externú pamäť zariadenia.

Počas návrhu systému bolo dôležité zohľadniť potrebu vhodnej synchronizácie údajov medzi webovou a mobilnou aplikáciou a vhodného zabezpečenia osobných údajov daných používateľov.

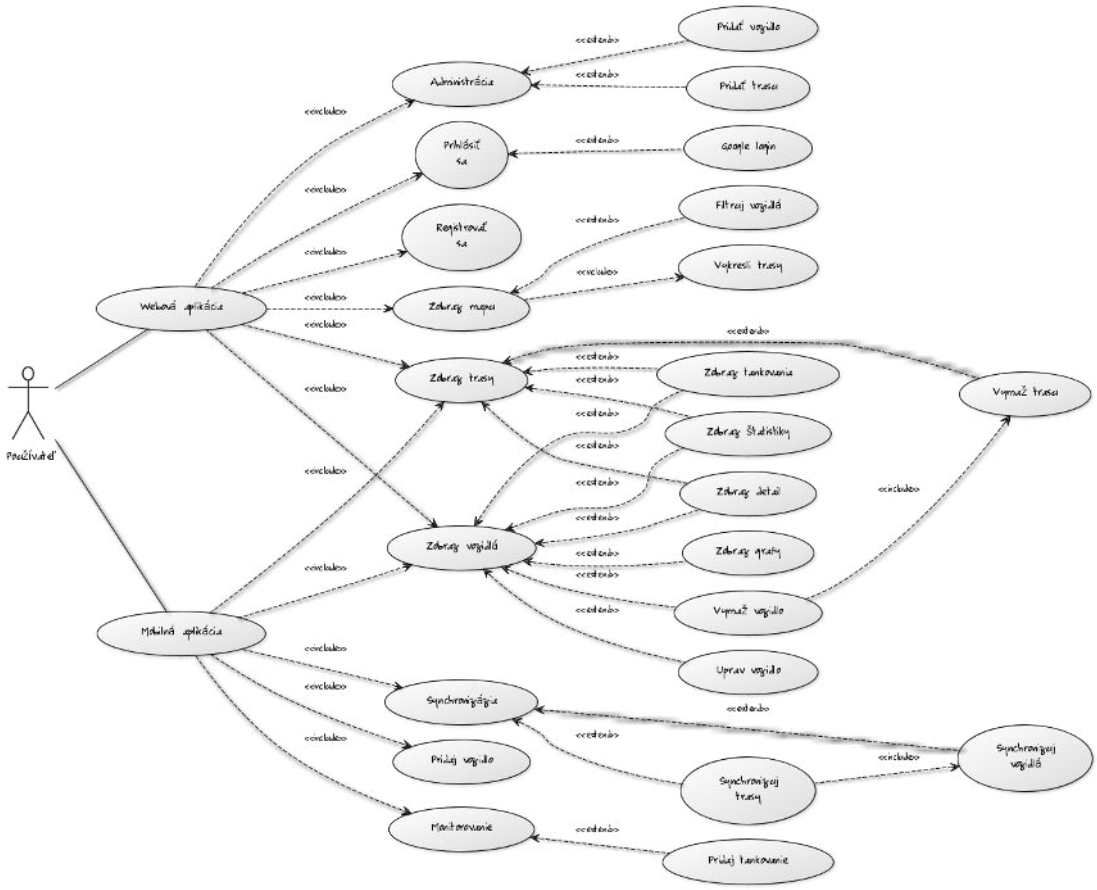
Pre účely sychronizácie údajov boli vytvorené dva módy - online synchronizácia a offline synchronizácia. Online synchronizácia vyžaduje pripojenie mobilného telefónu na internet a tiež synchronizáciu Google účtu medzi danými aplikáciami. Offline synchronizácia sa uskutočňuje prostredníctvom webovej aplikácie, kde po prihlásení bude možné importovať jednotlivé trasy z .kml súborov uložených v pamäti telefónu.

Pre zabezpečenie údajov bolo potrebné využiť framework Spring Security a zabezpečiť autentifikáciu používateľov pri vstupe do systému. Pre tento účel boli vytvorené dva spôsoby prihlasovania - prostredníctvom Google účtu cez OpenID

server a prostredníctvom e-mailovej adresy a hesla po úspešnej registrácii.

Podstatou systému je využívanie webovej aplikácie a mobilnej aplikácie za účelom získania potrebných štatistík, pre zhodnotenie využiteľnosti vozidla daným používateľom ako aj kontroly daných vozidiel na cestách. Pre tieto potreby bolo dôležitým faktorom vytvoriť vhodné grafické používateľské rozhranie ako z pohľadu webovej aplikácie, tak aj z pohľadu mobilnej aplikácie.

8.1.1 Model prípadov použitia



Obr. 8 – 1 Diagram všetkých prípadov použitia v informačnom systéme

8.2 Opis riešenia

V kapitole sú uvedené podrobnosti ohľadom implementácie navrhnutého systému spojené s popisom jeho fungovania z technického pohľadu.

8.2.1 Princíp fungovania

Fungovanie informačného systému je zabezpečené dvomi nosnými časťami. Základ tvorí mobilná aplikácia, ktorej fungovanie je závislé na hardvérovom module GPS. Aplikácia využíva elektromagnetické vlny prenášané prostredníctvom satelitných družíc, ktoré zabezpečujú určovanie polohy vzhľadom na prijímač. Získavanie jednotlivých objektov obsahujúcich údaje o polohe v podobe zemepisných šírok a dĺžok je sprostredkované pomocou objektu triedy *LocationManager*. Jeho funkcie musia mať povolenie pre prístup k danému modulu, preto je potrebné do hlavného konfiguračného súboru *AndroidManifest.xml* zapísať nasledujúce povolenia:

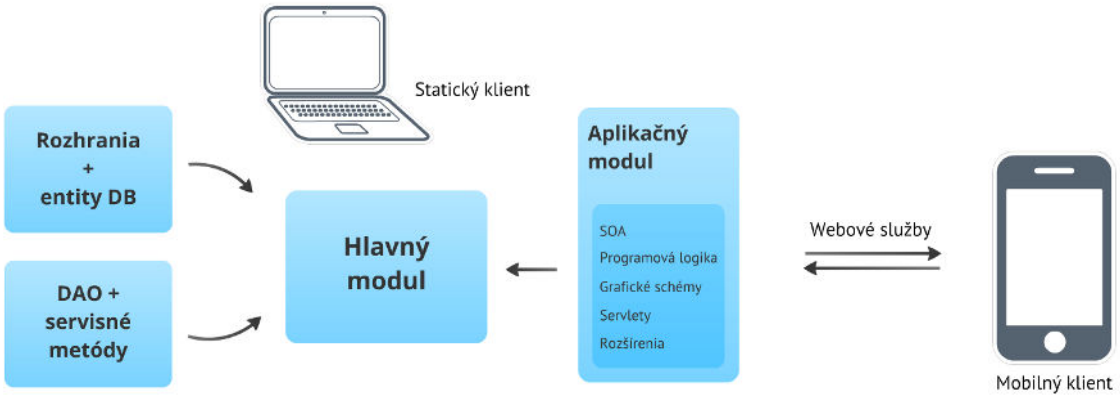
```
<uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Údaje získané prostredníctvom modulu sú uložené perzistentnými metódami na lokálnu databázu.

Využitím prvkov nakonfigurovaných služieb v statickej webovej aplikácii sa lokálne údaje posielajú cez komunikačný protokol v spolupráci s implementáciou REST pre webové služby. Serverová časť systému tieto údaje spracuje a vyhodnotí do rôznych podôb. Vytvoria sa objekty jednotlivých entít a podľa závislostí sú vykresľované do mapy, prípadne spracované ako štatistiky v tabuľkách. V rámci relácie je nastavený časovač, ktorý kontroluje komunikáciu v nastavenom časovom intervale a pri detekcii nových údajov je spustená synchronizácia. Tento proces zabezpečí integritu údajov prihláseného používateľa.

8.2.2 Architektúra aplikácie

Modulovú schému webovej aplikácie a spôsob jej komunikácie s mobilnou aplikáciou môžeme vidieť na obrázku 8 – 2



Obr. 8 – 2 Architektúra systému orientovaná na služby

Webová aplikácia - architektúra webovej aplikácie pozostáva zo 4 samostatných modulov medzi sebou navzájom prepojených, ktoré oddeľujú hlavnú a špecifickú funkcionalitu aplikácie bežiacej na pozadí, od grafického používateľského rozhrania slúžiaceho na zobrazovanie konkrétnych údajov, získaných pomocou mobilnej aplikácie.

Štruktúra webovej aplikácie z pohľadu modulov je nasledovná:

- **carTracking-api** - modul poskytuje rozhrania pre spracovanie DAO objektov a servisných tried v perzistentnej vrstve, ako aj triedy pre vytváranie JSON objektov potrebných pri komunikácii s mobilnou aplikáciou. Súčasťou sú aj objekty predstavujúce jednotlivé entity databázového modelu.

  V tomto module rozlišujeme balíčky ako:

  ***.dao** - balíček obsahuje rozhrania DAO objektov, ktoré slúžia pre prácu s databázou.

***.service** - balíček obsahujúci rozhrania, ktoré vykonávajú operácie nad vytvorenými DAO objektami.

***.exception** - balíček obsahujúci vlastné definované výnimky, ktoré môžu nastať v aplikácii.

***.json** - balíček obsahuje triedy pre vytváranie JSON objektov, ktoré sa používajú pri komunikácii webovej a mobilnej aplikácie.

***.model** - balíček obsahuje triedy reprezentujúce dátové entity v databázovom modeli.

***.restService** - balíček obsahujúci triedy určené pre webové služby, ktoré sa používajú na komunikáciu medzi webovou a mobilnou aplikáciou.

***.support** - balíček obsahuje podporné triedy, ktoré sa používajú v danom module.

- **carTracking-impl** - spolu s modulom carTracking-api tvorí základ perzistentnej vrstvy

  Balíčky modulu sú nasledovné:

  ***.daoImpl** - balíček obsahuje triedy implementujúce DAO rozhrania, ktoré slúžia pre prácu s databázou.

  ***.serviceImpl** - balíček obsahujúci triedy implementujúce rozhrania servisných metód.

  ***.security** - balíček obsahuje rozhranie *CarTrackingSecurityManager*, ktoré obsahuje metódy pre zabezpečenie autentifikácie.

  ***.securityImpl** - balíček obsahuje implementáciu rozhrania CarTrackingSecurityManager.

***.restService** - balíček obsahujúci triedy určené pre webové služby, ktoré sa používajú na komunikáciu medzi webovou a mobilnou aplikáciou.

***.support** - balíček obsahuje podporné triedy, ktoré sa používajú v danom module.

- **carTracking-parent** - predstavuje hlavný modul celej webovej aplikácie. Jeho úlohou je integrovať ostatné moduly a zabezpečiť tým dosiahnutie prvkov modulárnosti.

- **carTracking-web** - modul zabezpečuje hlavnú časť aplikačnej a prezentačnej vrstvy webovej aplikácie. Implementácia modulu je riešená prostredníctvom aplikačného frameworku Vaadin.

  Balíčky modulu sú nasledovné:

  ***.app** - balíček obsahuje hlavnú triedu aplikácie, ktorá sa volá pri spúštaní.

  ***.rest** - balíček obsahujúci triedy, ktoré slúžia na definíciu webových služieb.

  ***.servlet** - balíček obsahuje triedy pre definíciu servletov, ktoré sa používajú v aplikácii.

  ***.support** - balíček obsahuje podporné triedy využívajúce sa v danom module.

  ***.ui** - balíček obsahujúci triedy reprezentujúce úvodnú obrazovku aplikácie s prihlasovacím oknom.

  ***.user** - balíček obsahuje triedy, ktoré reprezentujú hlavné používateľské rozhranie po prihlásení používateľa.

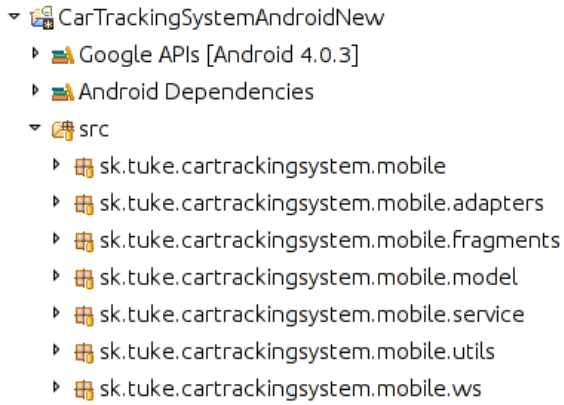
  ***.ui** - balíček obsahujúci triedy reprezentujúce úvodnú obrazovku aplikácie s prihlasovacím oknom.

***.carsInformation** - balíček obsahuje triedy, ktoré reprezentujú rozhranie zobrazujúce štatistiky a grafy pre existujúce vozidlá.

***.googleMapsWidget** - balíček obsahuje triedy, ktoré reprezentujú graficke rozhranie pre zobrazenie jednotlivých trás na mape.

***.tracksInformation** - balíček obsahujúci triedy, ktoré reprezentujú grafické rozhranie pre zobrazenie trás monitorovaných prihláseným používateľom.

Mobilná aplikácia - je postavená na platforme Android a implementovaná s použitím frameworku AndroidAnnotations v podobe externej knižnice. Okrem tohoto rámca sú použité 2 externé projekty v podobe knižníc pridané ako Android referencie. Ich úlohou je poskytnutie špecifického komponentu. Jednotlivé balíčky mobilnej aplikácie sú znázornené na obrázku 8 – 3:



Obr. 8 – 3 Projektová štruktúra mobilnej verzie aplikácie

- ***.mobile** - balíček predstavuje hlavnú triedu objektu *Activity*, ktorá má na starosť životný cyklus aplikácie, vrátane základného rozvrhnutia jednotlivých fragmentov.
- ***.adapters** - súčasťou balíčka sú nastaviteľné adaptéry pre komponenty obsahujúce vstupné údaje zo zoznamov, ktoré sú vo väčšine prípadov prepojené s lokálnou databázou.

- ***.fragments** - balíček obsahuje triedy typu *Fragment*, ktoré zabezpečujú 3 hlavné obrazovky v aplikácii vo forme kariet. Ich nadradeným komponentom je panel akcií, ktorý kontroluje jednotlivé inštancie týchto fragmentov.

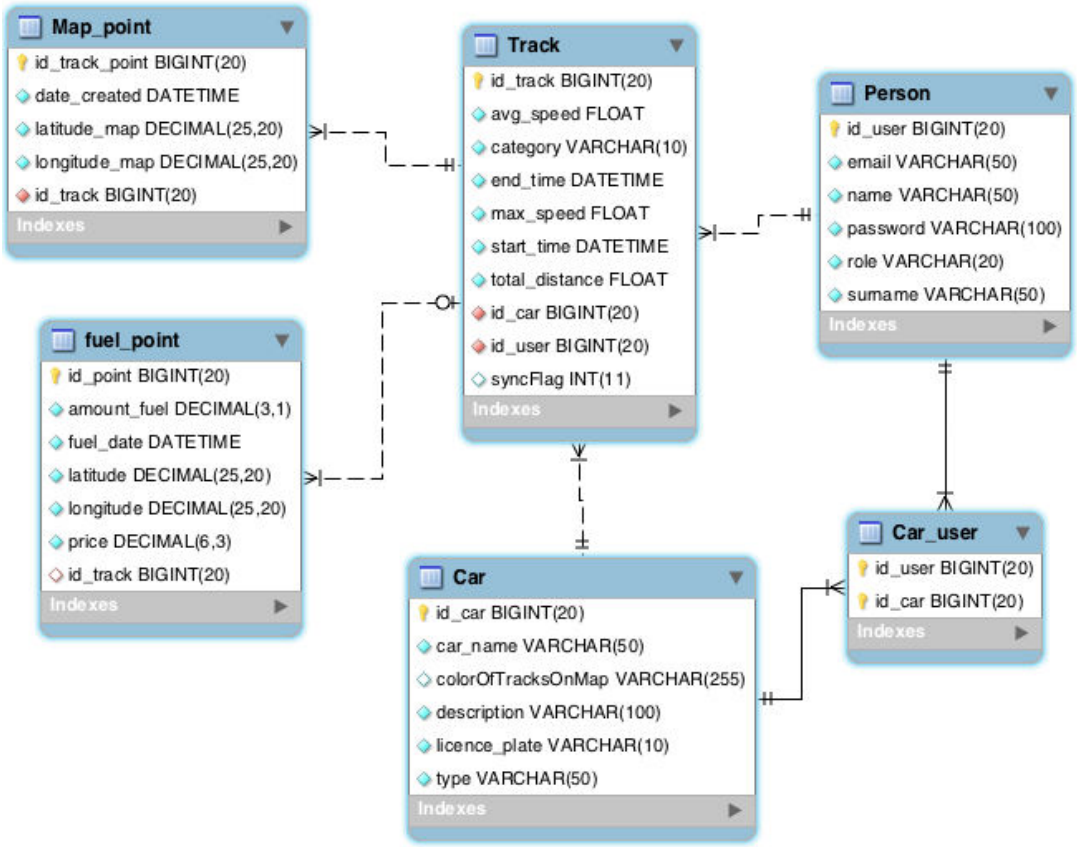
- ***.model** - zložky balíčka mapujú tabuľky v databáze na entity. Okrem toho obsahuje aj odvodené entity pre špecifické prípady a tiež triedy, ktoré sa serializujú do objektov posielaných pri komunikácii prostredníctvom webových služieb.

- ***.service** - jediná trieda objektu *Service* využívaná pri procese monitorovania prostredníctvom GPS modulu v zariadení, kvôli zachovaniu integrity aplikácie aj v prípade vloženia vykonávania aplikácie na pozadie.

- ***.utils** - balíček obsahuje nástroje využívané ostatnými triedami. Konkrétne sa jedná o konektor k databáze, ktorý spravuje všetky lokálne transakcie, syntaktický analyzátor pre potreby získania údajov zo špecifických formátov a v neposlednom rade nástroj pre exportovanie údajov do formátu spracovateľného službami pre mapy od spoločnosti Google.

- ***.ws** - balíček predstavuje klienta, ktorý riadi komunikáciu so serverovou časťou informačného systému prostredníctvom webových služieb a metód v ňom definovaných v rámci architektúry orientovanej na služby .

8.2.3 Perzistentný model



Obr. 8 – 4 Entitno-relačný diagram

8.3 Vývoj informačného systému

8.3.1 Vývojové nástroje

Informačný systém bol vyvíjaný na dvoch platformách. Jedná sa o platformu Macintosh s operačným systémom Mountain Lion 10.8 a platormu Linux s distribúciou Archlinux s verziou jadra 3.0.48-1. Ako vývojový nástroj bol použitý Eclipse IDE verzie Indigo so sadou rozšírení ako ADT, Vaadin, Maven, Sysdeo tomcat. Relačná databáza bola typu MySql a pre jej správu bol použitý nástroj MySql Workbench od spoločnosti Oracle.

8.3.2 Použité knižnice

Pre webovú aplikáciu bol použitý návrhový vzor „dependency injection“, z čoho vyplýva, že knižnice neboli priamo používané. Prostredníctvom nástroja na vytváranie zostáv Maven sa pridali všetky potrebné závislosti a rozšírenia.

V mobilnej aplikácii sme priamo použili z nasledovných dôvodov tieto externé knižnice :

Pre zabezpečenie komunikácie prostredníctvom webových služieb:

- spring-android-auth.jar
- spring-android-core.jar

Pre vytvorenie šablóny serializovaných objektov posielaných webovou službou:

- jackson-core.jar
- jackson-annotations.jar
- jackson-core-asl.jar
- jackson-databinf.jar

Pre implementáciu webových služieb prostredníctvom frameworku:

- androidannotations.jar

Pre použitie komponentu na výber farby zo spektra

- ambilwarna.jar

Pre použitie komponentu prepínača s rozšírenou funkcionalitou

- switchcompatlibrary.jar