Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Komparativní implementace trackovacího systému UAI PřF JU

Diplomová práce

Tomáš Honner

Školitel: Ing. František Drdák, CSc.

České Budějovice 2016

Bibliografické údaje:

Honner T., Komparativní implementace trackovacího systému UAI PřF JU [Comparative implementation of UAI PřF JU tracking system. Mgr. Thesis, in Czech.] 50 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Práce se zabývá analýzou stávajícího řešení trackovacího systému na Přírodovědecké fakultě Jihočeské univerzity, návrhem nového řešení tohoto systému a jeho implementací a porovnáním těchto řešení. Součástí práce je návrh metodiky porovnávání a provedení experimentů s cílem získání dat z komparace obou implementací. Následuje vyhodnocení těchto výsledků a vyvození závěru.

Klíčová slova

Trackovací systém, UAI Přf JU, komparativní implementace, Grails

Annotation

The master's thesis deals with analysis of current solution of car tracking system on Faculty of Science at South Bohemia University and new solution proposal of this system and its implementation and comparison of this solutions. One part of the thesis is proposal of comparison methodology and an experiment with the goal of gaining data from comparison of both implementation. Furthermore, to evaluate results of these comparison and to create conclusion.

Key words

Tracking system, UAI Přf JU, comparative implementation, Grails

Prohlášení

Prohlašuji, že svoji diplomovou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své diplomové práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 12.12.2016

Podpis…….....

Poděkování

Rád bych poděkoval panu Ing. Františku Drdákovi, CSc. za vedení mé diplomové práce a pomoc při jejím psaní.

Dále členům Ústavu aplikované informatiky za předané znalosti a odborné konzultace.

Rodičům Pavlovi a Marii Honnerovým za psychickou, morální a hmotnou podporu při studiu.

Obsah

1 Úvod

Tato diplomová práce se zabývá komparací dvou trackovacích aplikací, které jsou obě zaměřené na trackování služebních aut. První aplikace již existuje a je nasazena v provozu. Jmenuje se MotoTracker a provozuje jí Přírodovědecká fakulta Jihočeské univerzity v Českých Budějovicích, kde byla taktéž vyvinuta pracovníky a studenty ÚAI PřF. Tato aplikace slouží k získávání a zpracování dat ohledně jízd, realizovaných služebními vozy, patřícími téže fakultě. Dále slouží i jako kniha jízd a rezervační systém s funkcí schvalování rezervací, potažmo jízd. Druhá aplikace byla vyvíjena od analýzy, přes návrh, až k samotné implementaci, jako komparativní řešení se zaměřením na moderní JAVA technologie. Nejde o kompletní enterprise řešení, ale spíše o proof-of-concept nového přístupu. Po realizaci nové aplikace provedl autor srovnání těchto aplikací na základě metodiky, kterou si sám určil. Zároveň autor provedl i několik experimentů s oběma aplikacemi, vyhodnotil data z těchto experimentů a posoudil jejich relevantnost. Po provedení všech těchto kroků vyhodnotil autor výsledky porovnání obou aplikací a stanovil výsledný závěr.

Téma diplomové práce si autor vybral na základě toho, že jde z části o teoretickou a z části o praktickou diplomovou práci. Dalším hlediskem při posuzování tématu byla znalost a zkušenosti autora v oblasti programování vzhledem k tomu, že se mu věnoval ve zvýšeném rozsahu během studia univerzity.

Další autorovou motivací pro tuto diplomovou práci je snaha zdokonalit se v oblasti vývoje software a seznámit se s novými technologiemi, s nimiž se autor ještě nesetkal.

2 Cíle práce

1) Implementovat nově stávající systém pro sledování pohybu služebních aut, vyvinutý na UAI PřF JU, formou webové aplikace s využitím technologií Java Enterprise Edition.

2) Navrhnout metodiku porovnání stávající a nové implementace systému.

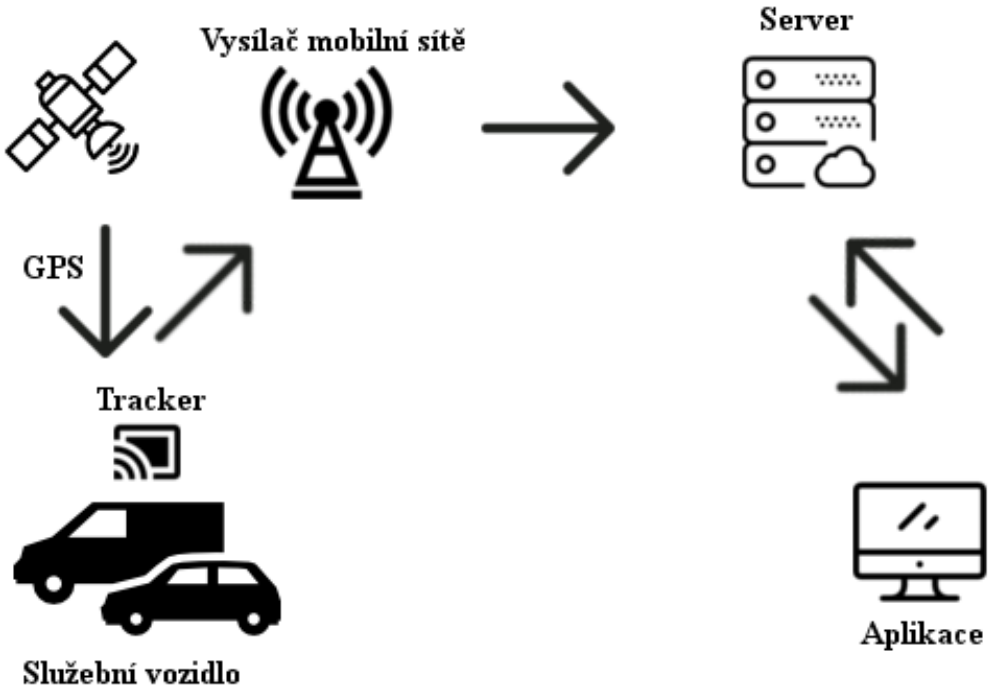
3) Provést příslušné experimenty s cílem získat data k porovnání obou implementací dle bodu 2.

4) Zhodnotit výsledky porovnání obou implementací, vyplývající ze zjištěných skutečností.

3 Základní pojmy

3.1 Trackovací systém

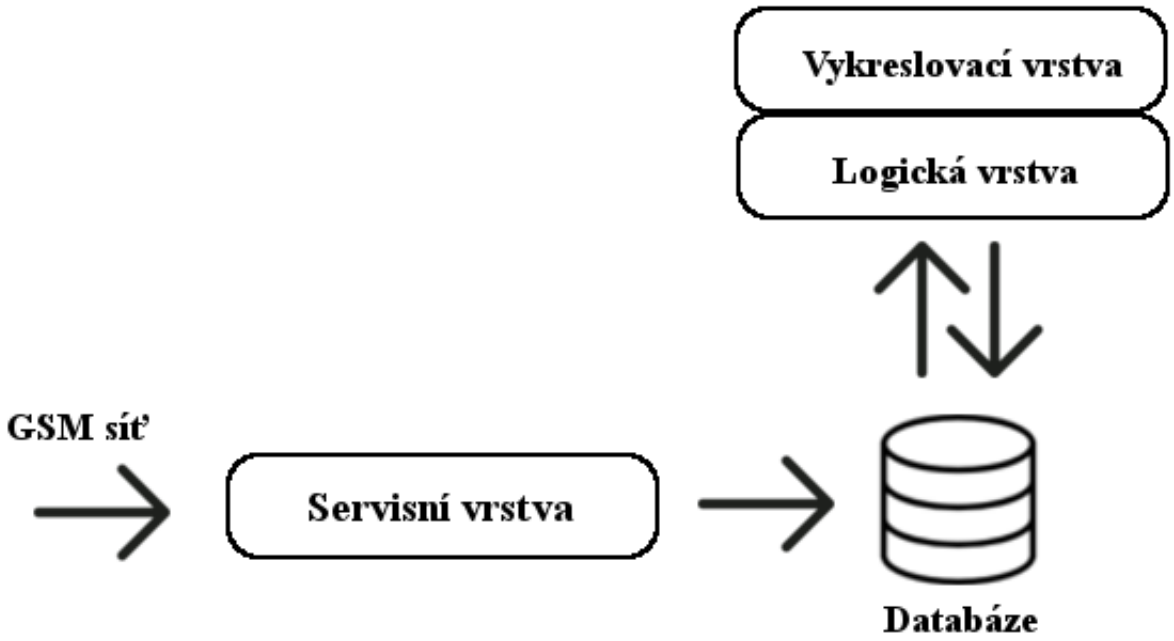
Jedná se o komplexní systém, řešící zjišťování geografické polohy sledovaných objektů a její archivaci. Tento systém zahrnuje hardware, takzvané trackery, které poskytují vstupní data pro další jeho část, a to programové vybavení trackovacího systému. Data jsou dále ukládána a zpracovávána v prostředí tohoto Systému. Výstupem ze systému jsou pak data o pohybu sledovaných objektů nebo konkrétně knihy jízd, či podklady pro účetnictví.



Obrázek 1 Všeobecný trackovací systém

3.2 Programové vybavení trackovacího systému

Programové vybavení trackovacího systému zpracovává data, která zasílají sledované objekty. Skládá se většinou z robustní databáze, schopné zpracovávat velké množství dat najednou. Dále z mezivrstvy, která provádí automatické operace nad daty a vrstvy vyobrazující nějakým způsobem zpracovaná data.



Obrázek 2 Všeobecné programové vybavení trackovacího systému

3.3 Tracker

Je hardwarové zařízení, které za pomocí signálu z GPS družice a triangulace zjišťuje svou polohu na povrchu planety Země. Toto zařízení dále v určitém časovém intervalu tuto polohu nějakým způsobem, nejčastěji pomocí sítě GSM, odesílá do trackovacího systému, kde dochází ke zpracování těchto dat.

Obrázek 3 Hardwarový tracker zdroj: http://www.nairaland.com/2016145/promo-gps-gsm-car-tracker-now

3.4 MotoTracker

Je stávající aplikace pro trackovací systém Přírodovědecké fakulty Jihočeské univerzity. Tato aplikace již běží v ostrém provozu a je využívána fakultou. Aplikace je plně funkční. Do aplikace jsou i nadále vyvíjeny dodatečné funkce a probíhá u ní aktualizace verzí.

S touto aplikací bude provedena komparace nově vyvinutého řešení v podobě aplikace RoadTitan.

Výrazem MotoTracker je v práci myšlena buď přímo aplikace MotoTracker, nebo celý trackovací systém, a to podle kontextu.

3.5 RoadTitan

Je nově vyvinuté řešení trackovacího systému. Tato aplikace je součástí této diplomové práce. Aplikace funguje pouze ve vývojovém a testovacím režimu. Zatím není připravena na produkční nasazení. Její porovnání se stávající aplikací MotoTracker je taktéž součástí překládané diplomové práce.

Výrazem RoadTitan je v práci myšlena buď přímo aplikace RoadTitan, nebo celý trackovací systém, a to podle kontextu.

3.6 Geoservice modul

Je samostatně stojící modul aplikace RoadTitan, který má za úkol sběr dat z trackerů, umístěných ve služebních vozech. Následně tato data vkládá do struktury datového modelu aplikace RoadTitan a tuto strukturu ukládá do hlavní databáze, se kterou již pracuje aplikace RoadTitan. Tento modul není implementován v aplikaci RoadTitan vzhledem k tomu, že samotný sběr dat není předmětem diplomové práce. V diplomové práci je proveden pouze jeho návrh.

4 Analýza stávajícího řešení (aplikace MotoTracker) trackovacího systému JU

4.1 Současné řešení

V současné době používaný systém trackování aut provozuje Přírodovědecká fakulta Jihočeské univerzity. V tomto systému jsou čtyři automobily, patřící této fakultě. Celý systém spravuje Ústav aplikované informatiky na téže fakultě, jmenovitě Ing. Jan Fesl a Mgr. Lukáš Valenta.

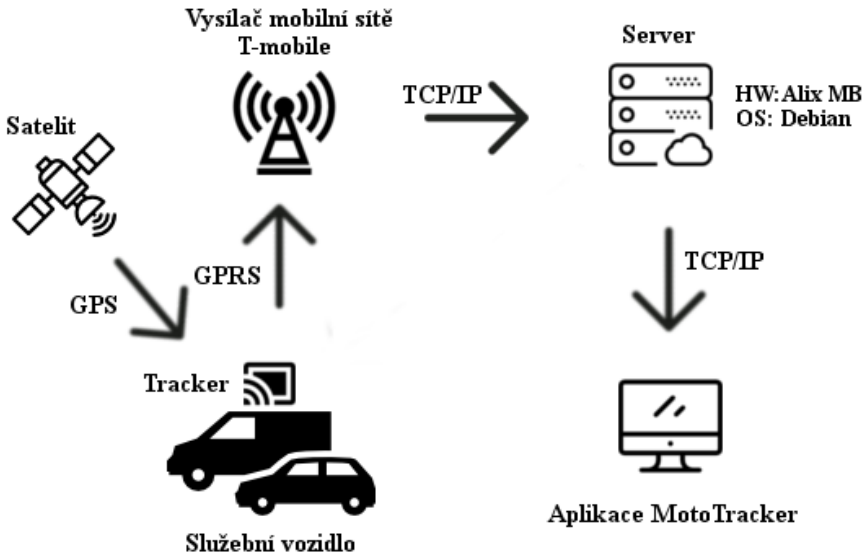
Z důvodu, že autorovi nebyly k dispozici detailní informace o modelech trackerů nebo konfiguraci serveru, se analýza stávajícího řešení zabývá spíše principy fungování systému.

4.1.1 Hardwarová část

Každý automobil má v sobě umístěný tracker. Tento tracker v pravidelných intervalech ze signálu vysílaného družicovým systémem přepočítává GPS souřadnice. Pokud se souřadnice nemění, GSM modul trackeru je neaktivní. Jakmile se pozice změní, GSM modul se aktivuje a přes mobilní síť, v našem případě T-mobile, odešle informace o své poloze. Tyto informace zasílá v pravidelných intervalech 20 sekund, dokud automobil opět nezastaví. Spolu se souřadnicemi a časovým razítkem jsou ještě odesílány informace o rychlosti, stavu nabití baterie v trackeru, o síle GPS signálu a IMEI jako unikátním identifikátoru trackeru.

Jako server funguje základní deska ALIX.2D13 s operačním systémem Debian. V tomto systému je ještě navíc spuštěný TCP server.

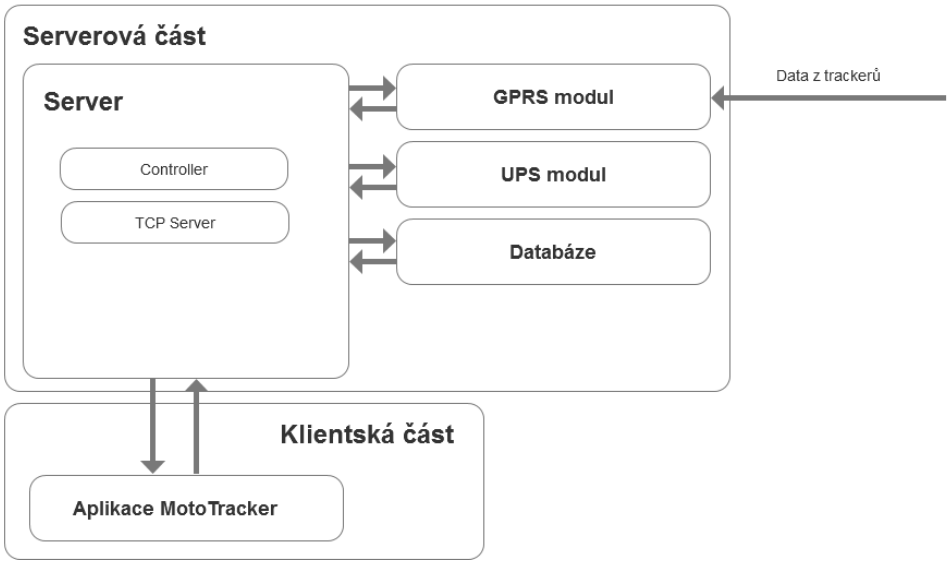
V předchozí verzi implementace se komunikace mezi trackerem a serverem zajišťovala za pomoci SMS zpráv. Nyní komunikace běží na úrovni sockets v síťovém provozu za použití technologie GPRS.



Obrázek 4 Schéma programu MotoTracker [1]

4.1.2 Softwarová část

Softwarovou část zajišťuje servisní vrstva na serveru a program MotoTracker, sloužící pro vyobrazení dat a konfiguraci, vyvinutý právě pro systém trackování aut Přírodovědecké fakulty Jihočeské univerzity. Tento program je popsán dále.



Obrázek 5 Schéma softwarové části [1]

4.1.3 Popis modulů softwarové části

GPRS modul

Je modul, který zajišťuje komunikaci s trackery, umístěnými ve služebních vozidlech. Tato komunikace se hlavně týká dat, odesílaných trackery o pohybu vozidel.

UPS modul

Jedná se o apcupsd klienta, který komunikuje s UPS a v případě potřeby server řádně vypne. Momentálně není používaný.

4.2 Analýza základních vlastností programu MotoTracker

4.2.1 Analýza základních vlastností programu všeobecně

Program MotoTracker je implementovaný v jazyce C++. Využívá návrhového vzoru MVC a komunikačního protokolu remoteAdmin ke komunikaci se serverem. V případě aplikace MotoTracker se jedná o architekturu klient-server, kdy dochází k „real-time“ zpracování požadavků a dat.

4.2.2 Analýza základních vlastností programu dle uživatele

Poloha vozidel

Program Mototracker umožňuje zobrazit současnou polohu vozidla nebo historii poloh, podle zadaných parametrů vyhledávání. Tuto polohu zobrazuje přes knihovnu GMLib a Google masp API na mapě. Uživatel si zároveň může zvolit druh mapy, či dokonce street view. Historii pohybu lze zobrazit buď graficky na mapě nebo rovněž textově v tabulce.

Kniha jízd

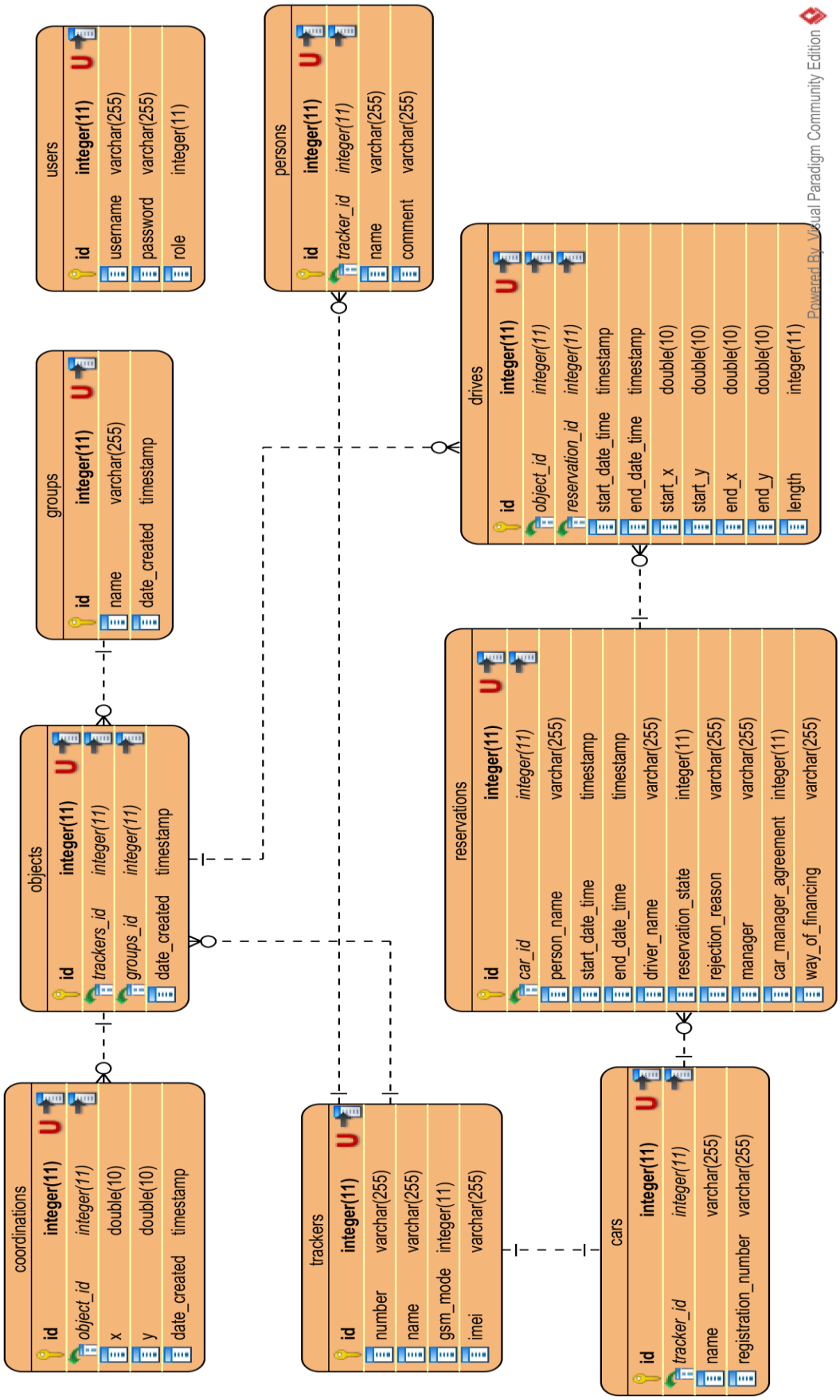
Slouží jako evidence cest používaných vozidel. Lze v ní vyhledávat podle zadaných parametrů. U každé jízdy jde zobrazit detail s podrobným výpisem.

Rezervační systém

Rezervační systém není součástí aplikace MotoTracker, ale je samostatně stojící webovou aplikací. Avšak je s programem MotoTracker používána.

Rezervační systém vozidel slouží k rezervaci vozidla na určitou jízdu. Tento záznam obsahuje vozidlo, řidiče, destinaci a další. Rezervaci musí nejdříve schválit nadřízený. V tomto konkrétním případě tajemnice fakulty.

4.2.3 Entitně-relační model databáze aplikace MotoTracker



5 Návrh nového řešení softwarové části trackovacího systému

5.1 Návrh aplikace obecně

Vzhledem k tomu, že aplikace vznikla jako komparativní řešení, se autor rozhodl zachovat funkčnosti ze stávající implementace, tj. aplikace MotoTracker, i v nové implementaci, tj. v aplikaci RoadTitan. Nová implementace obsahuje zobrazení polohy na mapě, knihu jízd a rezervační systém. Dále aplikace zachovává i dodatečnou funkčnost, jako zobrazení jednotlivé jízdy nebo zobrazení aktuální polohy na mapě. V neposlední řadě v aplikaci je možnost, tak jako ve stávajícím řešení (MotoTracker), vytvoření auta, trackeru nebo uživatele aplikace.

5.2 Navrhované změny

Základní změnou oproti původnímu řešení je přesunutí celé aplikace na aplikační server a vytvoření tak čistě webovského řešení. Aplikace už není rozdělena na klientskou část, někde v osobním počítači a na serverovou část, ale existuje pouze jako jeden celek, provozovaný aplikačním serverem. To přináší řadu výhod, jako je snadnější údržba, nenáročnost na výpočetní výkon klienta nebo z hlediska bezpečnosti plná kontrola nad aplikací.

Další změnou oproti původnímu řešení je víceuživatelský systém v RoadTitan. Předchozí řešení MotoTracker má jednoduchý uživatelský systém, kdy jsou uživatelé aplikace rozlišeni pouze právy (rolí). V aplikace RoadTitan jsou uživatelé navíc rozděleni do skupin, umožňujících oddělenou obsluhu více klientských společností. Skupiny se mezi sebou nevidí a členové jedné skupiny nemají žádná práva v jiné skupině, tzn. že mají práva pouze ve své skupině. Pouze osoba s právy SuperAdmin má práva mimo svou skupinu, resp. má práva vytvářet a mazat skupiny.

Další změnou oproti původnímu řešení je použití ORM v aplikaci RoadTitan oproti klasickému přístupu ukládání surových dat v entitně-relační databázi. Tento způsob

z vývojářského hlediska usnadňuje vývoj aplikace. Hlavně se to týká oblasti návrhu databáze, kdy vývojáři pouze stačí navrhnout třídní model, ze kterého se generuje databáze a nemusí navrhovat podobu databáze v podobě entitně-relačního modelu.

Další změnou oproti původnímu řešení je použití profesionálního frameworku pro tvorbu webových aplikací. Tím, že byl použit profesionální framework, bylo docíleno zkvalitnění vývoje. To se týká hlavně bezpečnosti, validace a dodatečných funkcionalit aplikace. Framework měl tyto funkcionality v sobě již zakomponované a plně jich využívá.

5.3 Platforma aplikace

Aplikace RoadTitan byla implementována jako plně webová aplikace, oproti MotoTrackeru, který funguje jako klient-server aplikace. To znamená, že build aplikace RaidTitan se spouští na serveru (JBoss, Apache Tomcat) a pro práci s aplikací se používá webový prohlížeč. Tím odpadá záležitost prerekvizit, potřebných k chodu aplikace, a naopak se zlepší vlastnosti, jako přenositelnost, škálovatelnost, nebo nezávislost na systému a jednodušší údržba systému.

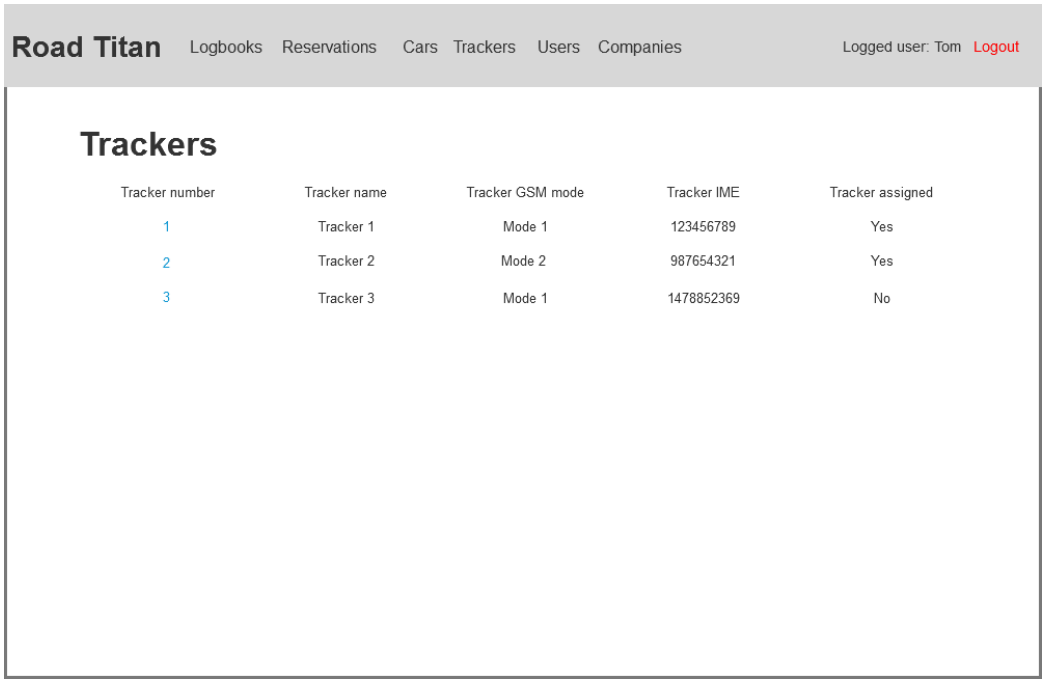
5.4 Návrh databáze

Databáze aplikace RoadTitan vychází z databáze aplikace MotoTracker, a to v tom smyslu, že aplikace RoadTitan má podobnou databázi, jako aplikace MotoTracker. Už teď je ale zřejmé, že databáze naprosto identická být nemůže z důvodu, že aplikace má jinou logiku, jiné zpracování dat a také odlišný způsob ukládání dat, protože aplikace RoadTitan používá ORM. Z toho důvodu byl vytvořen takový datový model, který databázi MotoTrackeru pouze simuluje a navíc ji rozšiřuje o další funkcionality. Podle toho byl vytvořen i model doménových tříd.

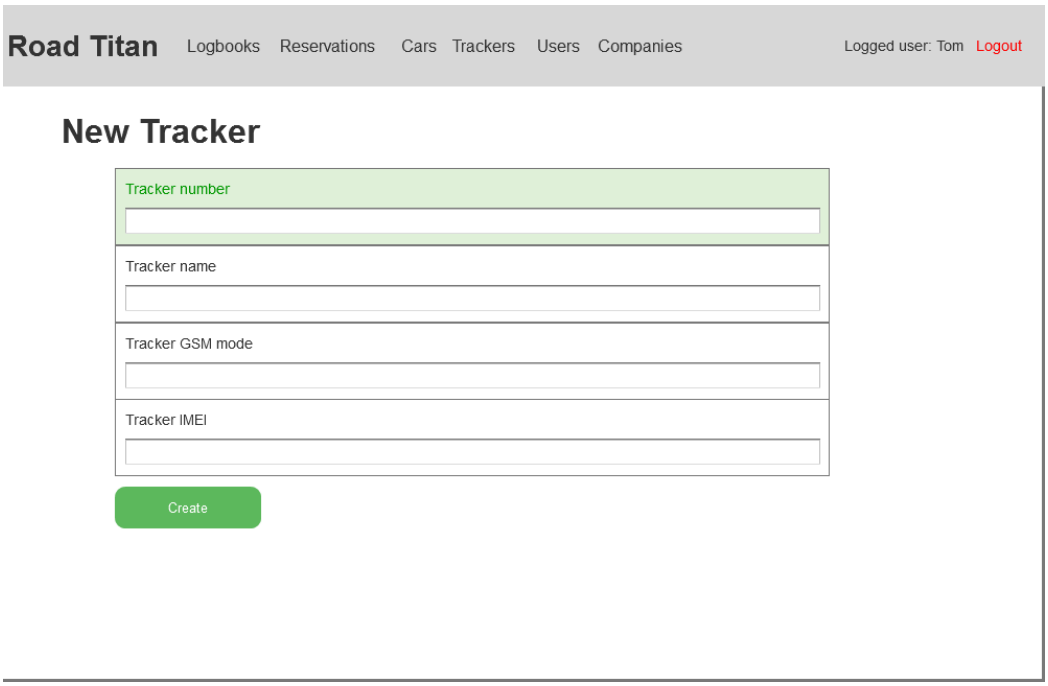
Předmětem implementace není samotný sběr dat z trackerů, ale samotná aplikační nadstavba nad těmito daty. Proto databáze slouží spíše jako vývojový nástroj, než jako plnohodnotná aplikační databáze.

5.5 Návrh grafického rozhraní - wireframes

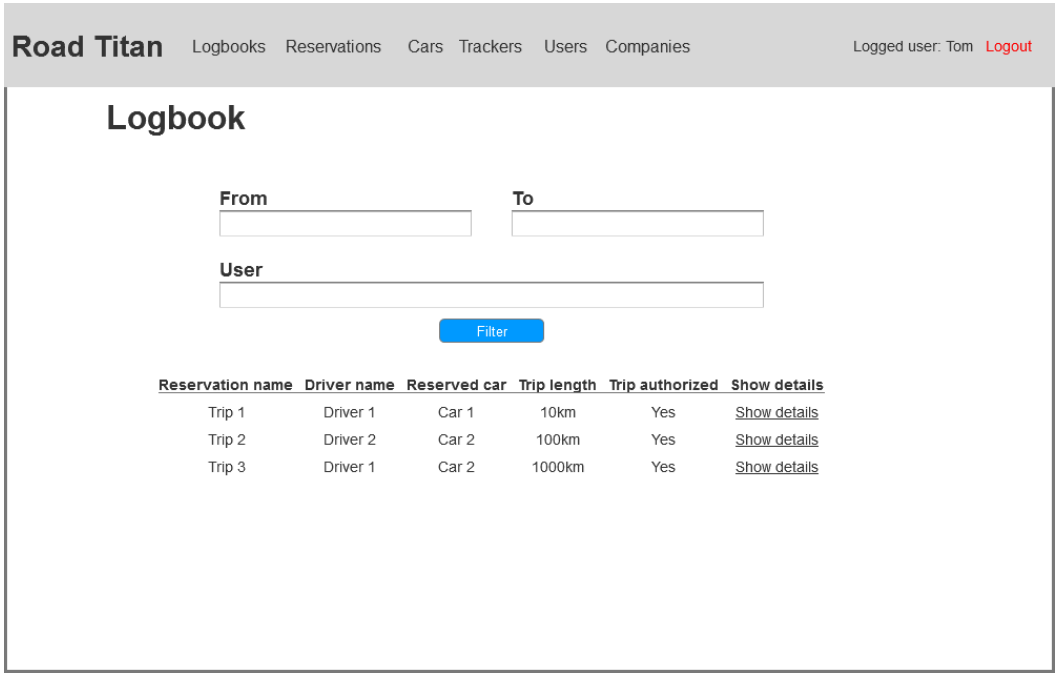
Jelikož má aplikace unifikované grafické rozhraní, tak je předkládán návrh (wireframes) jako příklad, který je stejný pro více obrazovek. Např. uživatelé, auta a trackery mají stejný návrh grafického rozhraní, z toho důvodu je v sekci wireframes uveden pouze návrh pro trackery.



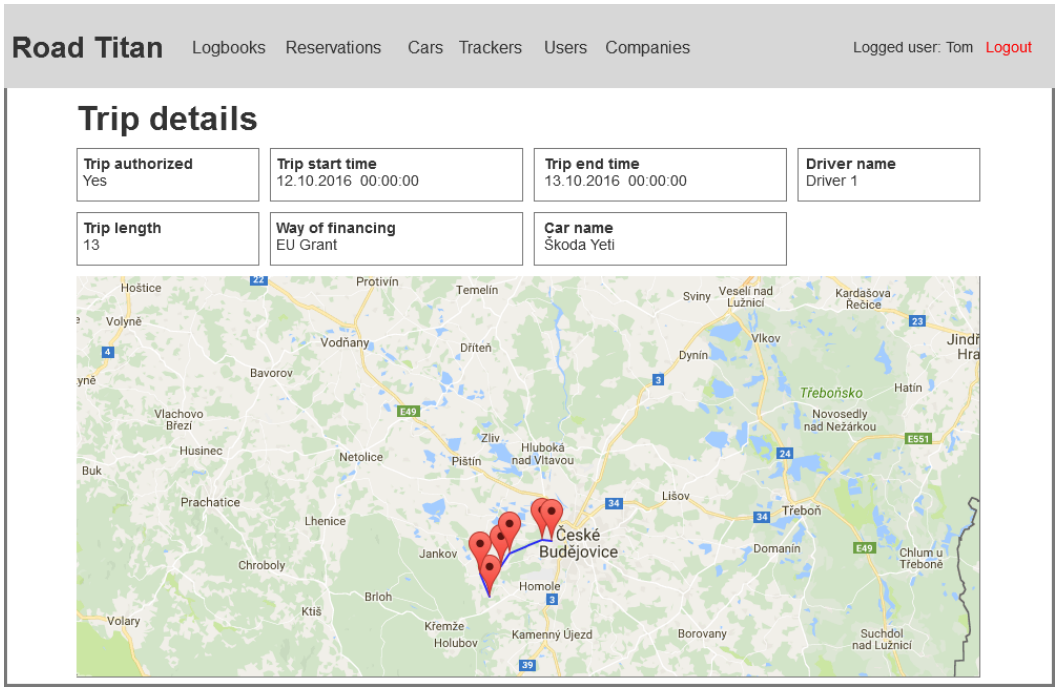
Obrázek 6 Návrh Tracker Index



Obrázek 7 Návrh nový tracker



Obrázek 8 Návrh kniha jízd



Obrázek 9 Návrh detail jízdy

5.6 Návrh definice rolí

5.6.1 SuperAdmin

Osoba s právy SuperAdmin je správcem aplikace. V tomto případě se jedná o správce systému na straně poskytovatele služby. SuperAdmin bude zakládat domény typu společnost a zakládat uživatele s právy ClientAdmin. Rovněž bude tyto společnosti a uživatele spravovat nebo upravovat, popřípadě rušit. Osoba s právy SuperAdmin má rovněž možnost upravovat další možnosti v aplikaci, jako rezervace nebo auta, ale nepočítá se, že bude toho aktivně využívat.

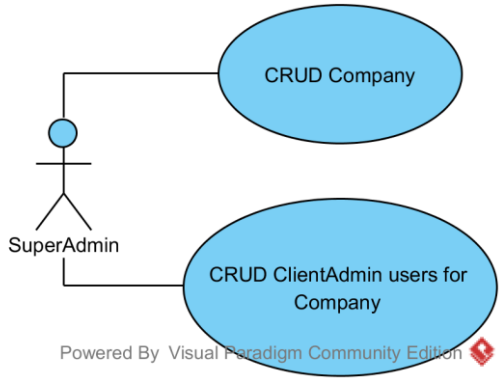
5.6.2 ClientAdmin

Osoba s právy ClientAdmin je správcem aplikace pro danou doménu společnost. Tato osoba bude vytvářet a spravovat auta, potažmo trackery, rezervace, knihu jízd atd. Dále bude moci prohlížet jízdy jednotlivých uživatelů, či aut. Tato osoba bude moci zakládat běžné uživatele s právy user, spravovat je a rušit. Dále bude mít právo potvrzovat nebo zamítat rezervace.

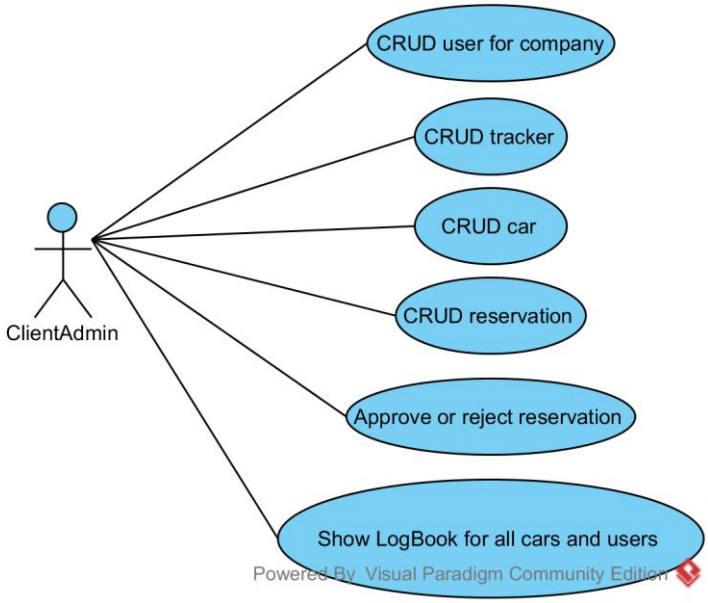
5.6.3 User

Osoba s právy user je základním uživatelem systému. Osoba s těmito právy bude moci vytvářet rezervace, upravovat je, popřípadě mazat. Dále bude moci vidět seznam všech jízd, vykonaných danou osobou.

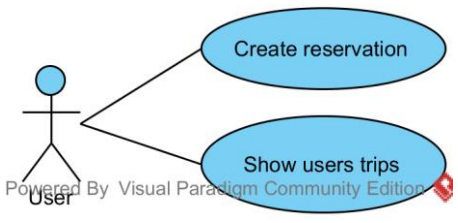
5.7 Use cases



Obrázek 10 Use case diagram pro uživatele SuperAdmin



Obrázek 11 Use case diagram pro uživatele ClientAdmin



Obrázek 12 Use case diagram pro běžného uživatele

6 Přehled technologií a standardů pro řešení

Pro implementaci nového řešení byl po přezkoumání vybrán framework Grails pro Java Enterprise. Hlavní důvody pro výběr tohoto frameworku byly, že striktně dodržuje architekturu MVC, která je v návrhu implementace. Dále má v sobě komponenty typu zabezpečení, autentifikace a autorizace, validace atd., čímž usnadňuje vývoj aplikace. Právě díky celé filozofii frameworku, kterou je co nejrychlejší a nejefektivnější vývoj, je tento framework skvělým kandidátem.

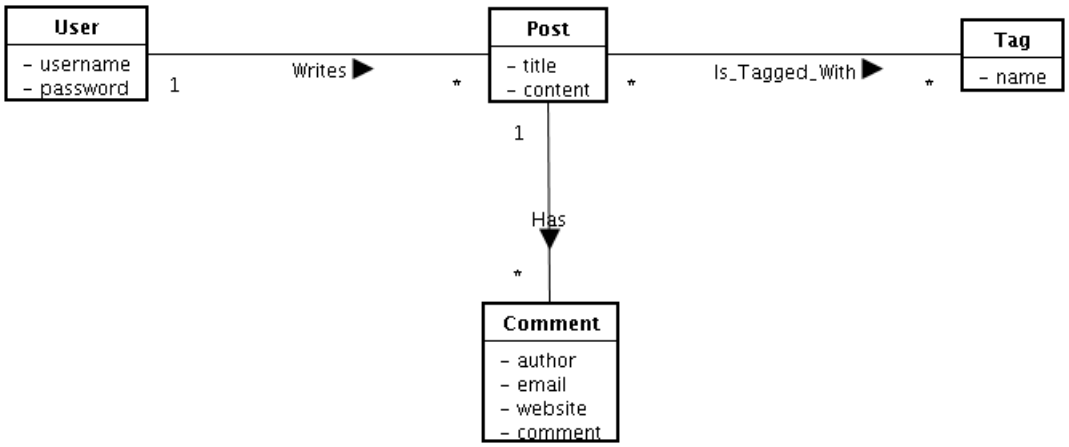
6.1 Grails framework

Grails je open source framework pro vytváření webových aplikací. Grails používá programovací jazyk Apache Groovy s kombinací Ruby on Rails (Grails). Tento framework je volně šiřitelný pod licencí Apache. Je zaměřen na vysokou produktivitu a celý je postaven na paradigmatu „Convention-over-Configuration“. Užití tohoto paradigmatu v reálu znamená snížení počtu rozhodnutí programátora bez ztráty flexibility frameworku. To má za následek, že se vývojář může více zaměřit na vývoj softwaru tím, že se sníží počet nutných konfigurací frameworku, které je nutné provést, aby software fungoval.

Grails framework implementuje takzvaný doménový model. Což je v softwarovém inženýrství konceptuální model domén, který zahrnuje jak data, tak jejich sémantický význam. Jinými slovy to je systém abstrakce, který popisuje vybrané aspekty z oblasti znalostí, vlivu nebo aktivity. V podstatě to znamená, že aplikace je rozdělena na mapu (síť) domén a těmito doménami a jejich strukturou popisuje něco z reálného světa. [2]

Součástí frameworku Grails je framework Spring a Hibernate. Všechny tyto frameworky jsou vyvíjeny firmou SpringSource.

Framework Grails byl použit ve verzi 2.5.5.



Obrázek 13 Příklad doménového modelu pro blog
Zdroj: http://cakebaker.42dh.com/2006/10/09/how-to-understand-your-problem-domain/

6.2 Spring framework

Spring Framework je populární open-source aplikační rámec neboli framework pro vývoj J2EE aplikací. První verze byla napsána Rodem Johnsonem, který ji vydal v rámci publikace své knihy Expert One-on-One J2EE Design and Development v říjnu 2002. Rod Johnson se ve své knize zabývá vývojem J2EE aplikací a věnuje pozornost problémům, se kterými se programátoři setkávají. V knize je průběžně prezentován kód frameworku, který se nazývá Interface21, a měl by vývoj J2EE aplikací usnadnit. Za pomoci Juergena Hoellera je později framework rozšířen a pod názvem Spring Framework uvolněn, jako open-source. Framework byl poprvé uvolněn pod licencí Apache 2.0 v červnu 2003. První verze 1.0 byla vydána v březnu 2004 a další verze potom v září 2004 a březnu 2005. Spring Framework 1.2.6 získal v roce 2006 ocenění Jolt productivity award a JAX Innovation Award. Současná verze je 3.0.0. Spring Framework může být použit libovolnou Java aplikací. Spring se stal populární v Java komunitě, jako alternativa k Enterprise Java Beans (EJB) nebo jako jeho nadstavba.[6]

6.3 Hibernate framework

Hibernate je framework napsaný v jazyce Java, který umožňuje tzv. objektově-relační mapování (ORM). Usnadňuje řešení otázky zachování dat objektů i po ukončení běhu aplikace. Je jednou z implementací Java Persistence API (JPA). Projekt vytvořil Gavin King, který později spolu s Hibernate přešel pod křídla firmy JBoss. Ta byla v roce 2006 převzata firmou Red Hat, která nadále pokračuje ve vývoji frameworku Hibernate. Gavin King spolu s JBoss také přešel do RedHatu. Hibernate poskytuje způsob, pomocí něhož je možné zachovat

stav objektů mezi dvěma spuštěními aplikace. Říkáme tedy, že udržuje data v persistentním stavu. Dosahuje toho pomocí ORM, což znamená, že mapuje Javovské objekty na entity v relační databázi. K tomu používá tzv. mapovací soubory, ve kterých je popsáno, jakým způsobem se mají data z objektu transformovat do databáze, a naopak a jakým způsobem se z databázových tabulek mají vytvořit objekty. Druhým způsobem, jak mapovat objekty, je použít anotace místo mapovacích souborů. V Hibernate se tedy pracuje s běžnými business objekty, přičemž mohou být sloupce tabulky spojeny přímo s atributy objektu, nebo mohou být připojeny skrze metody get/set a metody hashCode() a equals(). Nutno podotknout, že nelze použít EJB (viz Java Bean), ale pouze klasické objekty - tzv. POJO (Plain Old Java Object). Poté co jsou objekty uložené v databázi se na ně lze dotazovat jazykem HQL (Hibernate Query Language), který je odvozen z SQL, a je mu tedy velice podobný.[5]

6.4 Java EE

Java Enterprise Edition je součástí platformy Java, určené pro vývoj a provoz podnikových aplikací a informačních systémů. Základem pro platformu Java EE je platforma Java SE, nad ní jsou definovány součásti tvořící Java EE.

6.5 PostgreSQL

Jedná se o open source, entitně relační databázi, distribuovanou pod licencí MIT. Tato databáze se primárně vyvíjí pro linux, ale existuje i verze pro windows. Důvod pro její použití v projektu je, že její podpora je implementovaná přímo ve frameworku Grails a oproti MySql nabízí pokročilé funkce (např. při práci s datumy) a nástroje.[3]

Server PostgreSQL byl použit ve verzi 9.5

6.6 IntelliJ Idea 2016 (vývojové prostředí)

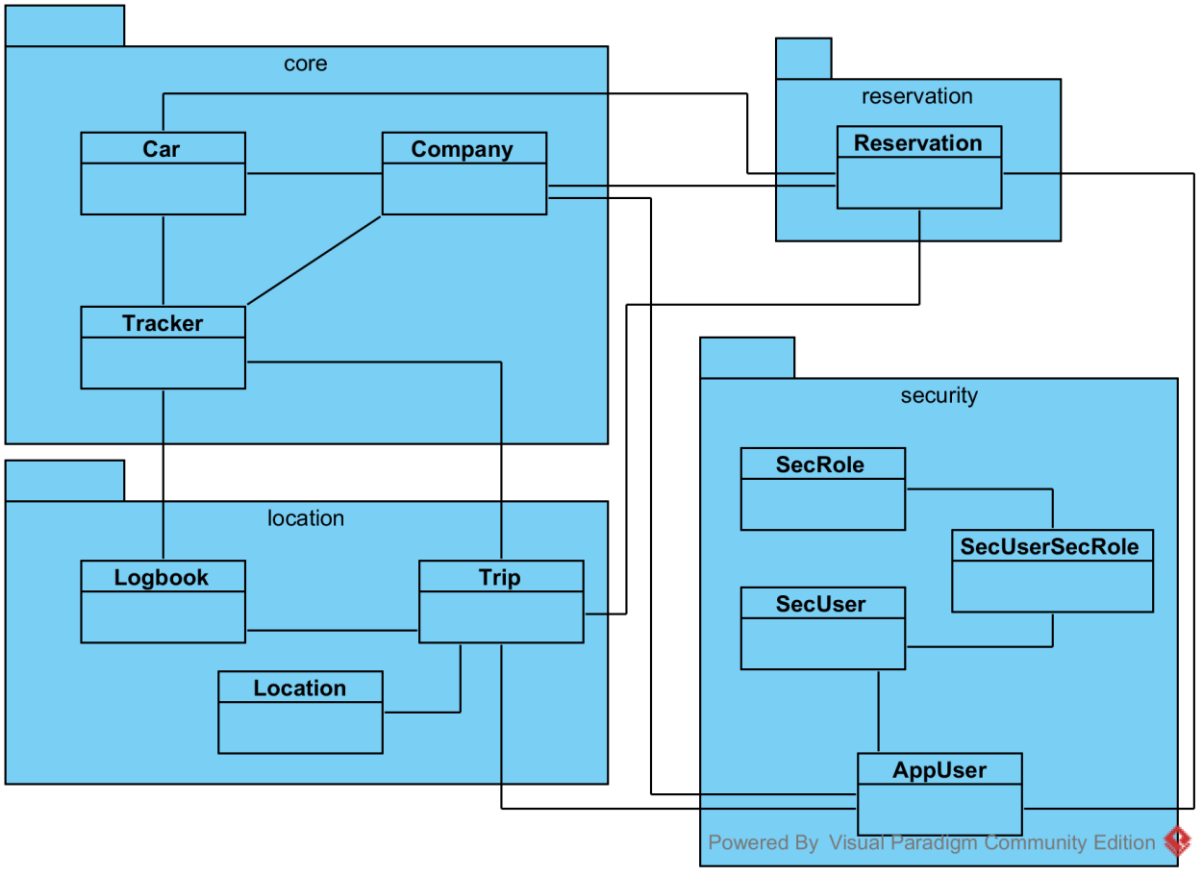
IntelliJ IDEA je komerčním vývojovým prostředím pro programování v jazycích Java, Groovy a dalších. První verze IntelliJ IDEA byla vydána v lednu 2001 a brzy si získala rozsáhlou uživatelskou základnu. Jako jedno z prvních IDE pro Javu nabídlo integrovanou sadu nástrojů pro refaktoring kódu.

7 Implementace

Samotná implementace byla rozdělena na tři etapy - iterace. Výsledkem první iterace bylo seznámení se s frameworkem, ve kterém bude aplikace implementována a vytvoření kostry aplikace. V druhé iteraci byl vytvořen model, za pomoci doménových tříd, a vytvořena základní funkčnost aplikace. Ve třetí iteraci byla vytvořena dodatečná funkčnost a aplikace byla testována. Každá ze tří iterací trvala přibližně 15 pracovních dnů.

7.1 Doménové třídy (model)

Doménové třídy slouží ve frameworku Grails k definici modelu (z architektury MVC) aplikace. Model se tak definuje kódem programovacího jazyka Java. Ve většině případů platí, že jedna doménová třída se rovná jedné tabulce v entitně-relační databázi. Přes kód se definují i vztahy mezi tabulkami (asociace), primární klíče, cizí klíče atd.



Obrázek 14 Zjednodušené schéma doménových tříd aplikace RoadTitan

7.1.1 Balíček core

Balíček core sdružuje základní doménové třídy v aplikaci.

Company

- Funguje jako hlavní doména celé aplikace.
- Rozděluje data z ostatních domén do jednotlivých celků.
- Prakticky funguje tak, že odděluje uživatele, auta, trackery, rezervace, knihu jízd jednoho „hlavního uživatele“ od jiného tak, aby se mezi sebou neviděli a nemohli se ovlivňovat.
- Instance domény Company může být založena pouze adminem.

Car

- Doména auto patří pod doménu společnost ve vztahu 1:N.
- Obsahuje název, typ, SPZ atd.
- Instance domény Car vytváří supervisor, může být vytvořena i adminen.

Tracker

- Doména Tracker patří pod doménu Company a zároveň patří doméně car ve vztahu 1:1.
- Obsahuje název trackeru, IMEI, GSM mód atd.
- Instance domény Tracker vytváří supervisor, může být vytvořena i adminen.

7.1.2 Balíček location

Balíček location sdružuje domény, které souvisejí s geolokací nebo s ní nějakým způsobem nakládají.

Logbook

- Doména Logbook patří doméně car ve vztahu 1:1.
- Obsahuje instance domény Trip.

Trip

- Doména Trip patří doméně Logbook ve vztahu 1:N a popřípadě rezervaci ve vztahu 1:1.
- Obsahuje jednotlivé lokace (Location) cesty, čas začátku, čas konce, délku atd.
- Instance domény Trip vytváří GeoService modul.

Location

- Doména Location patří doméně Trip ve vztahu 1:N.
- Obsahuje zeměpisnou šířku a délku a časové razítko.
- Instance domény Location vytváří GeoService modul.

7.1.3 Balíček reservation

Reservation

- Doména reservation slouží k ukládání rezervací .
- Tato doména patří pod doménu Company, AppUser a Car.
- Obsahuje datum rezervace, start, cíl, atd.
- Instance domény Resevation vytváří user nebo supervisor.

7.1.4 Balíček security

Balíček security sdružuje doménové třídy, spadající pod zabezpečení aplikace nebo pod management uživatelů. Většina domén v tomto balíčku patří pluginu Spring-security, která je ve frameworku Grails implementován.

AppUser

- Tato doména rozšiřuje doménu SecUser o další informace o uživateli.

SecUser

- Doména patří Spring-security.
- Jsou v definováni jednotliví uživatelé aplikace.

SecRole

- Doména patří Spring-security.
- Jsou v ní definovány role.

SecUserSecRole

- Doména patří Spring-security.
- Přiřazuje jednotlivým uživatelům jejich role.

```
class Tracker {

    Integer trackerNumber
    String trackerName
    GsmMode trackerGsmMode
    String trackerImei
    boolean trackerAssigned
    static belongsTo = [company: Company]
    static hasOne = [logBook: LogBook]

    DateTime dateCreated
    DateTime lastUpdated

    static constraints = {
        trackerNumber nullable: true, unique: true
        trackerName nullable: true
        trackerGsmMode nullable: true
        trackerImei nullable: true, unique: true
        trackerAssigned defaultValue: false
        company nullable: true
        logBook nullable: true

        dateCreated nullable: true
        lastUpdated nullable: true
    }

    static mapping = {
        table 'trackers'
        trackerNumber column: "tracker_number"
        trackerName column: "tracker_name"
        trackerGsmMode column: "tracker_gsm_mode"
        trackerImei column: "tracker_imei"
        trackerAssigned column: "tracker_assigned"

        dateCreated type: PersistentDateTime
        lastUpdated type: PersistentDateTime
    }
```

Obrázek 15 Ukázka doménové třídy Tracker

7.2 Kontrolery a servisy (Controllers, Services)

Ve frameworku Grails platí, že každá doménová třída, která se používá na front-endu, musí mít kontroler. Ten se stará o komunikace mezi front-endem a back-endem. Kontroler implicitně obsahuje metody CRUD, nadefinované uživatelem nebo defaultní nadefinované frameworkem. Dále by měl kontroler obsahovat metody, které něco vykreslují, respektive posílají nějakou odezvu na front-end. Tím se liší od servis, které by měly naopak obsahovat kód, který pouze zpracovává data. Například viz. Obrázek 16 kde kontroler se zeptá na trackery patřící společnosti a servisa mu dodá konkrétní společnost, ve které uživatel je. Kontroler ale obsahuje i bezpečnostní prvky, jako které uživatelské role mají k němu přístup nebo které HTTP metody je možné v kontroleru používat.

```groovy
@Secured(["ROLE_ADMIN","ROLE_SUPERVISOR"])
@Transactional(readOnly = true)
class TrackerController {

    def secService

    static allowedMethods = [save: "POST", update: "PUT", delete: "DELETE"]

    def index(Integer max) {
        params.max = Math.min(max ?: 10, 100)
        respond Tracker.findAllByCompany(secService.currentCompany()), model:[trackerInstanceCount: Tracker.count()]
    }

    def show(Tracker trackerInstance) {
        respond trackerInstance
    }

    def create() {
        respond new Tracker(params)
    }


    @Transactional
    def save(Tracker trackerInstance) {
        if (trackerInstance == null) {
            notFound()
            return
        }

        if (trackerInstance.hasErrors()) {
            respond trackerInstance.errors, view:'create'
            return
        }

        trackerInstance.setCompany(secService.currentCompany())
        trackerInstance.save flush:true
        LogBook book = new LogBook(tracker: trackerInstance).save(failOnError: true)
        trackerInstance.setLogBook(book)
```

Obrázek 16 Ukázka controlleru pro třídu Tracker

7.3 Pohledy (view)

Vytváří se v nich konečná podoba aplikace a to tak, že slouží jako skript pro tvorbu finálního HTML kódu, který se posílá do prohlížeče. Ve frameworku grails se používají pohledy typu .gsp . Což znamená Grails JSP. Jsou to v podstatě JSP skripty, rozšířené o Grails Tag knihovnu. Pohled se v nich vytváří velice podobným způsobem, jako v JSP, až na několik řídících tagů, které se nevolají z knihoven JSP, ale z knihoven Grails.

V aplikaci se používají v kombinaci s Javascriptem, konkrétně s JQuery a AJAX. Když není potřeba renderovat nový pohled, ale akorát poslat data pro pohled vygenerovaný

dříve, používá se právě AJAX request. V celé aplikaci se v kombinaci s AJAXem používají pouze requesty typu GET. Pro jakokoli úpravu dat se už standardně používají metody příslušných kontrolerů.

Pro vylepšení grafické úpravy webové aplikace je zároveň v pohledech naimportována grafické knihovna Twitter Bootstrap ve verzi 3 a knihovna pro použití Google Maps API Maplace.js.

```
<body>
    <div class="nav" role="navigation">
        <ul class="nav nav-tabs">
            <li><g:link class="create" action="create"><g:message code="tracker.crud.new" /></g:link></li>
        </ul>
    </div>
    <div>
        <h1 class="h1"><g:message code="tracker.title2" /></h1>
        <g:if test="${flash.message}">
            <div class="message" role="status">${flash.message}</div>
        </g:if>
        <table class="table table-hover table-condensed">
        <thead>
                <tr>
                    <g:sortableColumn property="trackerNumber" title="${message(code: 'tracker.number')}" />
                    <g:sortableColumn property="trackerName" title="${message(code: 'tracker.name')}" />
                    <g:sortableColumn property="trackerGsmMode" title="${message(code: 'tracker.gsmMode')}" />
                    <g:sortableColumn property="trackerImei" title="${message(code: 'tracker.imei')}" />
                    <g:sortableColumn property="trackerAssigned" title="${message(code: 'tracker.assigned')}" />
                </tr>
            </thead>
            <tbody>
            <g:each in="${trackerInstanceList}" status="i" var="trackerInstance">
                <tr>
                    <td><g:link action="show" id="${trackerInstance.id}">${fieldValue(bean: trackerInstance, field: "trackerNumber")}</g:link></td>
                    <td>${fieldValue(bean: trackerInstance, field: "trackerName")}</td>
                    <td>${fieldValue(bean: trackerInstance, field: "trackerGsmMode")}</td>
                    <td>${fieldValue(bean: trackerInstance, field: "trackerImei")}</td>
                    <td>${fieldValue(bean: trackerInstance, field: "trackerAssigned")}</td>
                </tr>
            </g:each>
            </tbody>
        </table>
        <div class="pagination">
            <g:paginate total="${trackerInstanceCount ?: 0}" />
        </div>
    </div>
</body>
```

Obrázek 17 Ukázka pohledu pro controller Tracker

7.4 Konfigurace aplikace

Následuje ukázka konfiguračních souborů aplikace RoadTitan. Konfigurační soubory ve frameworku grails mají podobu tříd .groovy, viz. Obrázek 18 a 19

```
// environment specific settings
environments {
    development {
        dataSource {
            dbCreate = "create-drop" // one of 'create', 'create-drop', 'update', 'validate', ''
            //url = "jdbc:h2:mem:devDb;MVCC=TRUE;LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE"
            driverClassName = "org.postgresql.Driver"
            url = "jdbc:postgresql://localhost:5432/road_titan_final"
            username = "road_titan"
            password = "road"
        }
    }
    test {
        dataSource {
            dbCreate = "update" // one of 'create', 'create-drop', 'update', 'validate', ''
            //url = "jdbc:h2:mem:devDb;MVCC=TRUE;LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE"
            driverClassName = "org.postgresql.Driver"
            url = "jdbc:postgresql://localhost:5432/road_titan_final"
            username = "road_titan"
            password = "road"
        }
    }
```

Obrázek 18 Konfigurace databáze soubor DataSource.groovy

```
class BootStrap {

    def init = { servletContext ->

        // Company
        Company company1 = new Company(companyName: "Toms Company", companyCode: "123", companyAddress: "Silicon Valley").save(failOnError: true)
        Company company2 = new Company(companyName: "Ondras Company", companyCode: "321", companyAddress: "Silicon Valley").save(failOnError: true)
        Company company3 = new Company(companyName: "Bobs Company", companyCode: "789", companyAddress: "Silicon Valley").save(failOnError: true)

        if (!AppUser.count) {
            // Define of Security roles
            def userRole = SecRole.findByAuthority('ROLE_USER') ?: new SecRole(authority: 'ROLE_USER').save(failOnError: true)
            def adminRole = SecRole.findByAuthority('ROLE_ADMIN') ?: new SecRole(authority: 'ROLE_ADMIN').save(failOnError: true)
            def supervisorRole = SecRole.findByAuthority('ROLE_SUPERVISOR') ?: new SecRole(authority: 'ROLE_SUPERVISOR').save(failOnError: true)

            // User with admin privileges
            def adminUser = AppUser.findByAppUserName('ADMIN') ?: new AppUser(
                    appUserName: 'ADMIN',
                    appUserEmail: 'admin@admin.com',
                    appUserAddress: 'Silicon Valley',
                    appUserPassword: 'admin',
                    secUser: new SecUser(
                            realName: 'Tomas Honner',
                            username: 'admin',
                            password: 'admin',
                            enabled: true).save(failOnError: true),
                    company: new Company(companyName: "ADMIN COMPANY").save(failOnError: true)
            ).save(failOnError: true)

            if (!adminUser.secUser.authorities.contains(adminRole)) {
                SecUserSecRole.create adminUser.secUser, adminRole
            }
```

Obrázek 19 Konfigurace a zavedení uživatele Admin soubor Bootstrap.groovy

8 GeoService modul

8.1 Důvod zavedení

Při implementaci aplikace RoadTitan vznikl zásadní problém. Na vzniku tohoto problému měly hlavní podíl dva fakty. Prvním z nich byl striktně objektové orientace programu RoadTitan. Druhým byl návrh databáze programu Mototracker, který se ukázal jako nevhodný pro RoadTitan a musel být dodatečně upraven.

Problémem byly vazby mezi objekty v programu RoadTitan. De facto pro vytvoření objektu (např. Trip) bylo potřeba dalších objektů, které v daný okamžik nebyly k dispozici. Díky těmto vazbám bylo nutné vymyslet jiné řešení, než použití stávající databáze programu MotoTracker.

Problém bylo možno řešit několika způsoby. Prvním z nich bylo ukládání bezkontextových dat z trackerů do sekundární databáze a posléze jejich zpracování aplikací tím, že jim bude dodán context. Druhým způsobem, který byl nakonec zvolen, je modul aplikace, který bude zastupovat servisní vrstvu trackovacího systému PřF JU.

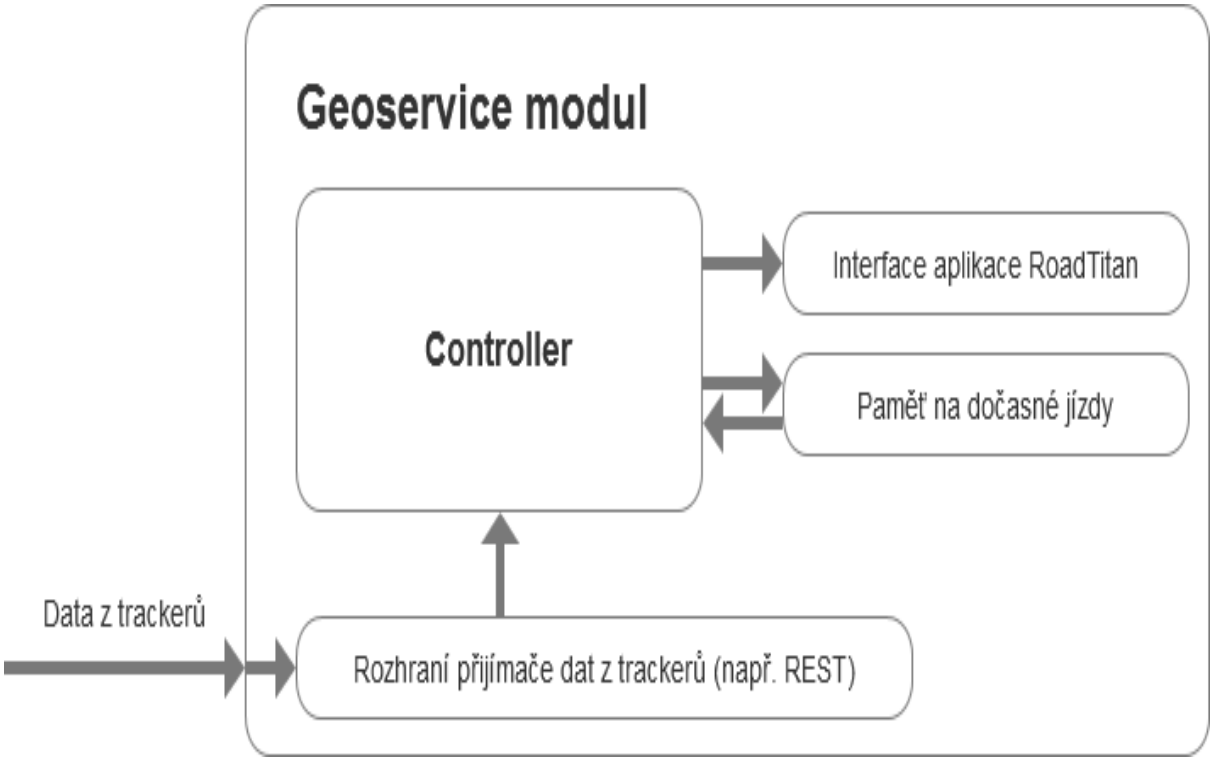
8.2 Princip fungování modulu

Tento modul aplikace funguje jako příjmač dat z trackerů a to tak, že je v něm naimplementované RESTové rozhraní, přes které trackery komunikují s aplikací. Data z trackerů ale neukládá do databáze, ale nechává v paměti do doby, než z nich sestaví objekty, které přepošle hlavní programu a ten už je bez obtíží zpracuje.

V praxi modul funguje tak, že naslouchá na rozhraní přijímače dat. Po přijetí záznamu z trackeru v podobě nové lokace a času tracker jednoznačně identifikuje pomocí databáze hlavní aplikace. Poté se podívá do paměti, jestli je tracker v pohybu. Pokud ano, přiřadí lokaci k aktivní cestě. Pokud ne, založí novou cestu a to tak, že se podívá do rezervací, jestli je na určený čas nějaká rezervace v rezervačním systému a podle toho cestu vyhodnotí jako autorizovanou nebo neautorizovanou a založí ji v systému. V každém případě, po nějakém daném časovém úseku, pokud z trackeru nepřijde další záznam, tzn. tracker se nepohybuje, cestu uzavře, připojí k ní další potřebná data a přepošle hlavní aplikaci, která cestu uloží do

knihy jízd. Tím, že toto nedělá aplikace, ale její modul, dochází k vyřešení problému s bezkontextovými daty.

8.3 Schéma modulu



Obrázek 20 Schéma GeoService modulu

8.4 Implementace modulu

Po dohodě s vedoucím diplomové práce byl autorem Geoservice modul pouze navržen a nikoliv již implementován z těchto důvodů:

1. Předmětem diplomové práce není vytvořit kompletní enterprise řešení trackovacího systému aut, ale pouze navrhnout alternativu aktuální verze a porovnat je.
2. Vytváření vlastní servisní vrstvy pro trackery ve formě modulu by bylo vzhledem k předmětu práce nelogické a neúměrně časově náročné.
3. Tento modul nelze vytvořit univerzálně, naopak se musí vytvořit na míru hardwaru trackerů. Proto je racionálnější řešení tento modul programovat až s konkrétními trackery, než jen do aplikace, která slouží jako proof-of-concept.

9 Testování

Aplikace byla průběžně testována při vývoji. Testování bylo prováděno na dvou úrovních. Na první úrovni byla testována funkčnost za pomoci JUnit testů. Na druhé úrovni byla testována stabilita tím, že se testovalo chování aplikace za uměle vytvořených podmínek, které měly simulovat nestandardní události např. výstup v jiném formátu.

Bylo prováděno i autorské testování klasickou formou. Toto testování zahrnovalo, že se vývojář pokoušel najít a opravit co nejvíce chyb v kódu. Také zahrnovalo testování validací vstupů a to jak na frontendu, tak i na backendu.

9.1 Testování zabezpečení

Testování bezpečnosti aplikace RoadTitan bylo provedeno ve dvou fázích. V první fázi byl program podroben testování implementace uživatelů a jejich práv tak, že bylo testováno, jestli uživatel může provádět operace, ke kterým nemá práva. Ve druhé fázi bylo testováno komunikační rozhraní aplikace. Testy se skládaly z posílání různých HTTP metod na RESTové rozhraní aplikace. Vyhodnocovalo se chování bezpečnostního modulu aplikace, tudíž to, zda se metoda provedla, nebo ne. Tímto byly zjišťovány nedostatky v zabezpečení aplikace, které byly následně odstraňovány.

9.2 Vyhodnocení bezpečnostních testů

Na základě výsledků bezpečnostních testů aplikace a toho, že frontend aplikace s backendem aplikace komunikuje za pomoci RESTového rozhraní, byla konfigurace aplikace změněna tak, aby aplikace komunikovala výhradně přes protokol HTTPS a tím došlo k významnému zlepšení zabezpečení proti útokům mířeným právě na RESTové rozhraní. Použitím protokolu HTTPS byla zároveň zvýšena úroveň celkového zabezpečení aplikace.

10 Komparace stávajícího a nově implementovaného řešení

10.1 Z hlediska funkcionality

Aplikace RoadTitan je vyvíjena jako obecnější řešení oproti aplikaci MotoTracker. Ta úzce souvisí se stávajícím systémem trackování služebních aut na PřF JU. Aplikace RoadTitan nabízí díky GeoService modulu napojení na jakýkoliv trackovací systém, nehledě na hardware tohoto systému úpravou stávajícího GeoService modulu, nebo implementací nového. Taktéž aplikace RoadTitan má naimplementován více uživatelský přístup ve smyslu, že ho může najednou používat nekonečně mnoho institucí nebo firem. MotoTracker toto neumožňuje. Další rozdílem je rezervační systém. V případě MotoTrackeru jde o samostatně stojící aplikaci. U RoadTitan je rezervační systém součástí hlavní aplikace.

10.2 Z hlediska architektury

Co se týká architektury, hlavní změna byla provedena navržením architektury aplikace RoadTitan jako plně webové aplikace oproti aplikaci MotoTracker, která používá architekturu klient-server.

Obě implementace používají architekturu MVC. Stávající aplikace MotoTracker používá architekturu MVC volně, oproti tomu nová aplikace RoadTitan ji dodržuje naprosto striktně. To je i dáno použitím frameworku, který použití architektury MVC striktně vyžaduje. Podrobněji viz. schémata softwarové a hardwarové části programů MotoTracker a RoadTitan v analytické části práce.

10.3 Z hlediska použitých technologií

Aplikace MotoTracker je implementována pomocí technologie C++, ze které používá pseudo objekty. Tato implementace není postavená na žádném frameworku a vychází pouze z best practices.

Aplikace RoadTitan je implementována v jazyce JAVA ve frameworku Grails, který stojí na jazyce Apache Groovy. To z ní dělá plně objektovou aplikaci. Krom toho jsou

ve frameworku nainstalovány pluginy pro bezpečnost, validaci, práci s objekty typu čas a designové pluginy.

10.4 Z hlediska databáze

Aplikace Mototracker používá klasickou entitně-relační databázi, konktrétně MySql. Do této databáze se ukládají holá data, a to ze dvou zdrojů. Prvním zdrojem je servisní vrstva, starající se o obsluhu trackerů. Druhým zdrojem je samotná aplikace MotoTracker, která ukládá data o uživatelích, autech, trackerech, rezervacích atd.

Aplikace RoadTitan používá entitně-relační databázi PostgreSql. Tuto databázi využívá poměrně odlišným způsobem, než MotoTracker. Na serveru PostgeSQL existuje pouze prázdná databáze bez tabulek. Framework Grails dynamicky generuje databázi z tzv. doménových tříd a obsahu, který se do databáze ukládá přes framework Hibernate.

10.5 Z hlediska škálovatelnosti

Program Mototracker, jelikož se jedná o klient-server aplikaci, je sám o sobě špatně škálovatelný. Škálovatelná je pouze jeho serverová část ve formě databáze. Tam lze při zvýšení zátěže navýšit zdroje výkonu, nebo zavést clusterizaci.

Program RoadTitan, vzhledem k tomu, že se jedná o plně webovou aplikaci, je maximálně škálovatelný. Při zvýšené zátěži nebo nárůstu komunikace, stačí navýšit výkon serveru, popřípadě virtuálního stroje. Pokud tento postup bude neefektivní nebo příliš finančně náročný, může program běžet na více serverech a požadavky je možné mezi ně rozdělit. Navíc lze i samotnou aplikaci rozdělit na části, které samostatně poběží na různých serverech. Takto jde aplikace škálovat bez omezení až na hranici únosnosti finančních nákladů.

10.6 Z hlediska aktualizace programu

Oba programy mají v sobě nějakou podobu aktualizace programu. V případě MotoTrackeru to je klasická aktualizace, která se stahuje ze serveru a posléze se nainstaluje. Oproti tomu RoadTitan, jelikož se jedná o webovou aplikaci otevíranou ve webovém prohlížeči, se aktualizuje nahráním nové verze na server. Tím budou mít všichni uživatelé bez

jakékoliv interakce nejnovější verzi. Navíc z toho vyplývá další benefit, kterým je jednodušší údržba.

10.7 Z hlediska přenositelnosti

Aplikace MotoTracker je částečně závislá na platformě, jelikož se jedná o klient-server aplikaci a klientská část je naprogramována v jazyce C++. Proto je spustitelná pouze na systému, který má interpreta tohoto jazyka, popřípadě umožňuje jeho kompilaci.

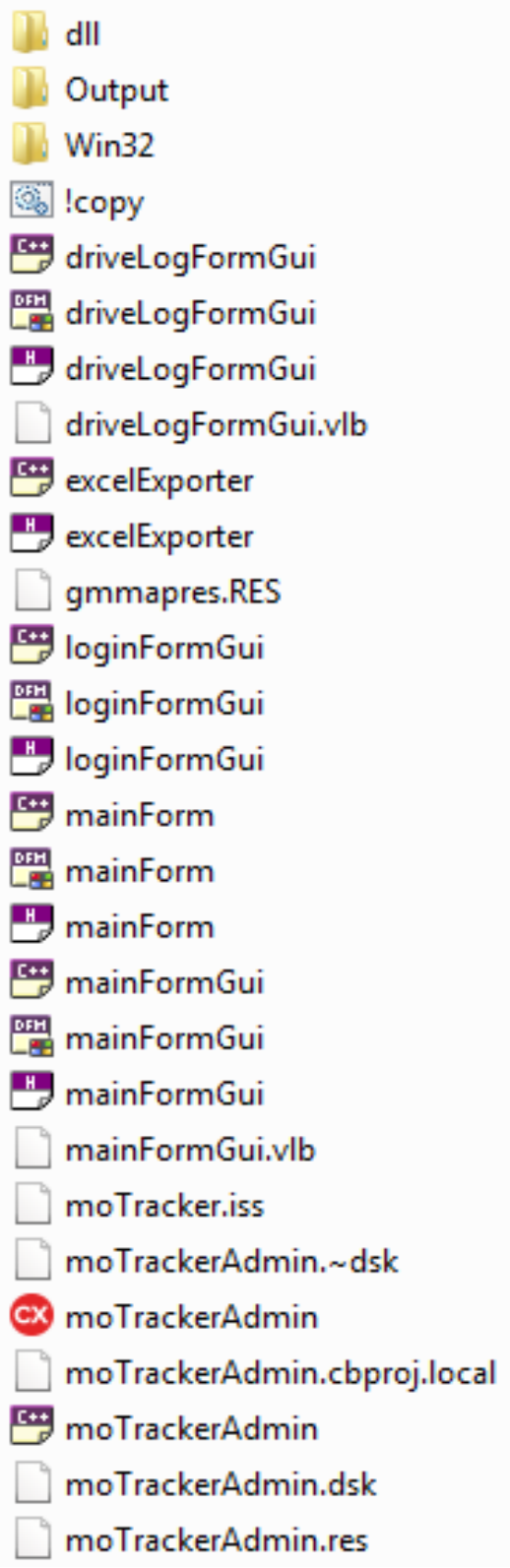
Aplikace RoadTitan je snadno přenositelná. Tato implementace je nezávislá na platformě, systému, interpretech, dokonce i zařízeních. Jediné, co je potřeba pro její plnou funkčnost je webový prohlížeč.

10.8 Z hlediska náročnosti implementace pro programátora

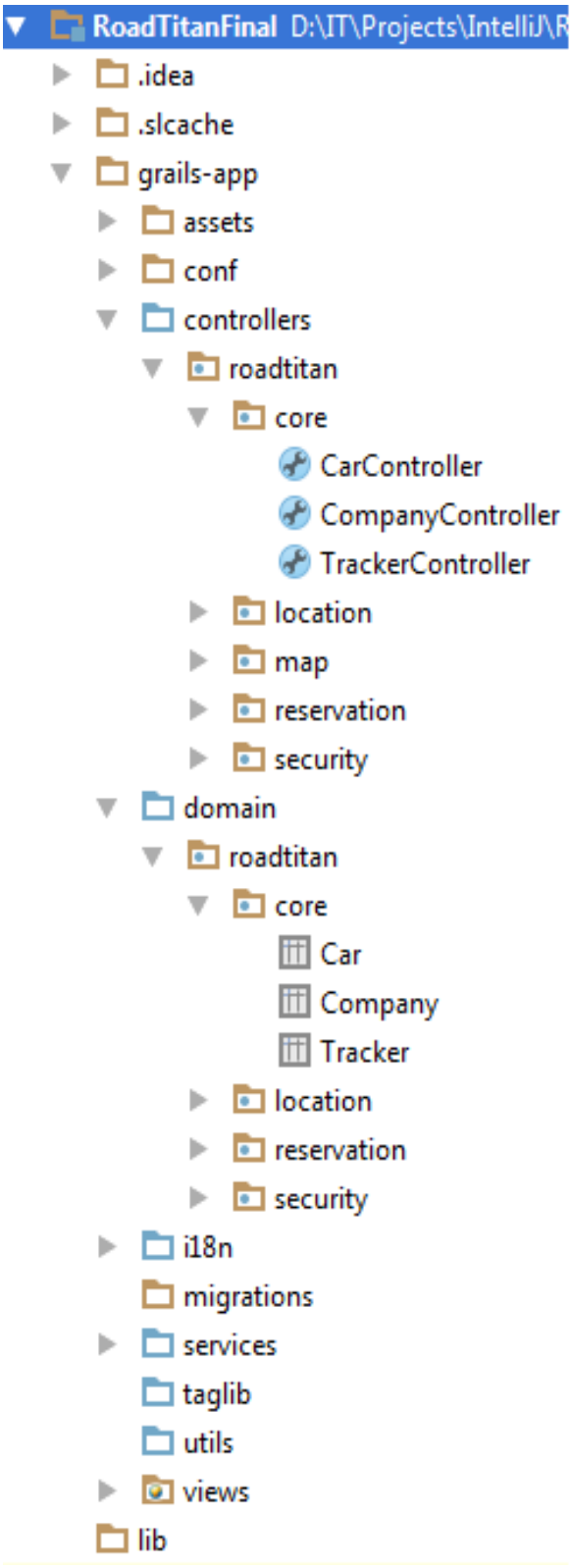
U aplikace MotoTracker byla implementace objektivně náročnější. Jde hlavně o to, že program RoadTitan byl implementován ve frameworku, oproti MotoTrackeru, který byl implementován v prakticky čistém C++ s použitím několika knihoven. Framework Grails do RoadTitanu sám implementoval řadu dodatečných funkčností, jako stránkování, validaci, bezpečnostní prvky a toto všechno muselo být v aplikaci MotoTracker doděláno programátorem.

10.9 Z hlediska adresářové struktury

Aplikace MotoTracker má jednoduchou adresářovou strukturu, kde jsou všechny zdrojové kódy uložené v kořenu adresáře. Knihovny a buildy se ukládají do samostatných podadresářů. Naopak aplikace RoadTitan má striktně nadefinovanou adresářovou strukturu, která je daná frameworkem Grails a musí být dodržena pro správnou funkčnost aplikace. To znamená, že doménové třídy musí být v adresáři domain, kontrolery v adresáři controllers a konfigurační soubory v adresáři conf atd.



Obrázek 21 MotoTracker adresářová struktura



Obrázek 22 RoadTitan adresářová struktura

11 Experimenty

Experimenty byly průběžně prováděny během testovací fáze vývoje aplikace. Z převážné části se jednalo o pokusy s výkonem aplikace. Při provádění těchto experimentů autor došel k závěru, že porovnávání aplikací RoadTitan a MotoTracker v oblasti výkonu a rychlosti je neracionální a výsledky tohoto porovnání by byly neprůkazné. A to z těchto důvodů.

Aplikace, co se týká rychlosti a výkonu, jsou jen těžko porovnatelné vzhledem k tomu, že obě pracují v jiných prostředích. Výkon a rychlost aplikace MotoTracker, jelikož se jedná o klient-server aplikaci, závisí jak na hardwarových prostředcích serveru, tak na hardwarových prostředcích klienta. Oproti tomu aplikace RoadTitan je plně závislá jenom na hardwarových prostředcích serveru. Z toho vyplývá, že aplikace RoadTitan, pokud bude mít dostatek serverových HW prostředků, bude vždy výkonná a k tomuto výkonu i přiměřeně rychlá (míněna doba odezvy). Aplikace MotoTracker je závislá i na výkonu klienta, který může, při použití nedostatečného hardwaru, aplikaci zpomalit.

Dalším faktorem, který ovlivnil pokusy s aplikacemi bylo to, že aplikace MotoTracker v sobě neobsahuje žádné nástroje pro výkonnostní testování a tyto nástroje by do ní musely být doimplementovány. Jednalo by se například o metody měřící rychlost provedení dotazu, či funkce, které aplikace RoadTitan již obsahuje.

Díky tomu, že obě aplikace pro svoji funkčnost potřebují různá prostředí a experimenty na nich se nedají provádět v unifikovaném prostředí, je prakticky nemožné výsledky těchto experimentů považovat za průkazné nebo něco z nich dále vyvozovat.

11.1 Experiment náročnosti na systémové prostředky

11.1.1Konfigurace testovací sestavy

Počítač:	HP Probook 470 G1
Procesor:	Intel Core i5
RAM:	8GB
HDD	250 GB SSD
Operační systém	MS Windows 7 64bit

Tabulka 1 Konfigurace testovací sestavy

Tabulka ukazuje základní konfiguraci počítačové sestavy, na které byly testovány obě aplikace.

11.1.2Náročnost aplikace RoadTitan na systémové prostředky

	Aplikace	**Databáze**
Vytížení procesoru	30-80% podle zatížení	0.5%
Vytížení RAM	1500MB	40MB
Velikost samotné aplikace	35MB	-

Tabulka 2 Náročnost aplikace RoadTitan na systémové prostředky

Tabulka ukazuje, kolik systémových prostředků si vyžádala aplikace RoadTitan pro svůj chod v prostředí operačního systému MS Windows 7.

11.1.3Rychlost operací v aplikaci RoadTitan

	Aplikace (ms)	**Databáze (ms)**
Založení auta	1305	11
Založení trackeru	947	9
Založení rezervace	1970	15
Zobrazení jízd	1864	85
Detail jízdy	2938	25

Tabulka 3 Rychlost operací v aplikaci RoadTitan

Tabulka ukazuje odezvu v řádu milisekund na jednotlivé akce v aplikaci, a to z pohledu odezvy databáze a odezvy aplikace, myšleno až do vykreslení události v prohlížeči.

Při provádění experimentů byla zjištěna vyšší náročnost aplikace na hardwarové prostředky stroje, na kterém je aplikace provozována. To vyplývá ze skutečnosti, že se samotnou aplikací se spouští zároveň aplikační server, databázový server a vývojové a testovací prostředí. Vyšší náročnost na HW prostředky serveru je také dána tím, že aplikace byla vyvíjena jako business řešení trackovacího systému, oproti MotoTrackeru, který byl od začátku vyvíjen jako nízkonákladové řešení.

12 Vyhodnocení

Na základě provedeného srovnání aplikací RoadTitan a MotoTracker byly odvozeny výhody a nevýhody obou aplikací, viz. níže.

12.1 Výhody vs. nevýhody

MotoTracker

Výhody	Nevýhody
Konkrétní řešení pro konkrétní systém	Konkrétní řešení pro konkrétní systém
Nižší náročnost na HW než RoadTitan	Škálovatelná pouze serverová část
Nižší náklady na pořízení a provoz	Aplikace je závislá na platformě, v tomto případě musí mít interpreta jazyka C++

Tabulka 4 Výhody a nevýhody aplikace MotoTracker

RoadTitan

Výhody	Nevýhody
Obecnější řešení	Obecnější řešení
Kompletně škálovatelné, a to na více úrovních	Vyšší náročnost na výpočetní výkon
Aplikace není rozdělena do částí, ale je celá na serveru	Vyšší náklady na pořízení a provoz
Přístup z jakéhokoliv zařízení s webovým prohlížečem	
Nezávislá na platformě	

Tabulka 5 Výhody a nevýhody aplikace RoadTitan

12.2 Návrhy na vylepšení aplikace RoadTitan

Aplikaci RoadTitan by bylo možné vylepšit co se týče výkonu, a to zejména v oblasti databáze. Ta zatím nepočítá s využitím většího objemu dat z trackerů. Toto by šlo řešit například vytvořením pohledů, které by odpovídaly určitým časovým intervalům, např. jízdy pro konkrétní auta za poslední měsíc. Při velkém objemu dat by se mohla databáze rozdělit do clusterů s tím, že každý cluster by byl spuštěn na jiném serveru z hlediska hardwaru.

Dalšími vylepšeními by mohla být integrace aplikace RoadTitan s nějakou interní podnikovou aplikací, zlepšení zabezpečení aplikace nebo generování výstupních formulářů pro účetnictví.

13 Závěr

V rámci diplomové práce byla řešena problematika trackovacího systém pro sledování pohybu služebních vozidel. Byla provedena analýza stávajícího trackovacího systému, používaného Přírodovědeckou fakultou Jihočeské univerzity. Tento systém byl analyzován z hlediska hardwarové a softwarové konfigurace. Po této analýze byl vytvořen návrh nového softwarového řešení v podobě aplikace RoadTitan. Návrh obsahuje standardní metodické techniky, jako např. use case diagramy a wireframe aplikace. Poté byly vybrány a popsány technologie, které byly použity při implementaci aplikace. Součástí práce je popis samotné implementace, a to ve všech vrstvách MVC architektury, společně s praktickým příkladem v podobě části kódu aplikace. Diplomová práce dále obsahuje návrh GeoService modulu, jako součásti aplikace RoadTitan. Dále bylo provedeno testování aplikace a provedení experimentů, jako součást testování. Diplomová práce dále obsahuje shrnutí a porovnání vlastností obou řešení a to programu MotoTracker, jako současného řešení trackovacího systému, s programem RoadTitan, jako navrhovaného řešení trackovacího systému. Nakonec byla celá diplomová práce vyhodnocena a byl vyvozen závěr.

Jako vedlejší výsledek se autor seznámil s novými webovými technologiemi a postupy a naučil se je používat a implementovat v projektech.

13.1 Splnění cílů diplomové práce

Cíl	Výsledek
1. Implementovat nově stávající systém pro sledování pohybu služebních aut, vyvinutý na UAI PřF JU formou webové aplikace s využitím technologií Java Enterprise Edition.	**Splněno**
2. Navrhnout metodiku porovnání stávající a nové implementace systému.	**Splněno**
3. Provést příslušné experimenty, s cílem získat data k porovnání obou implementací dle bodu 2.	**Splněno**
4. Zhodnotit výsledky porovnání obou implementací, vyplývající ze zjištěných skutečností.	**Splněno**

Bibliografie

1. **Valenta, Lukáš.** *Řídicí modul pro systém na monitorování polohy mobilních objektů.* České Budějovice : Jihočeská univerzita v Českých Budějovicích, 2016.

2. **Domain model.** *Wikipedia.* [Online] The Wikimedia Foundation, Inc., 2016. https://en.wikipedia.org/wiki/Domain_model.

3. Group, The PostgreSQL Global Development. PostgreSQL documentation. ***PostgreSQl.*** [Online] The PostgreSQL Global Development Group, 2016. https://www.postgresql.org/docs/.

4. Project, The Grails. Grails documnetation. *Grails.* [Online] The Grails Project, 2016. https://grails.org/documentation.html.

5. Hibernate. Hibernate documentation. *Hibernate.* [Online] Hibernate, 2016. http://hibernate.org/orm/documentation/5.2/.

6. Software, Pivotal. Spring documentation. *Spring.* [Online] Pivotal Software, 2016. https://spring.io/docs.

Přílohy

A Aplikace

Aplikace se nachází na přiloženém cd nebo na adrese
https://github.com/TomasHonner/RoadTitanFinal

B Požadavky a postup instalace aplikace

B.I Požadavky

Aplikace pro své fungování potřebuje aplikační server Tomcat nebo JBoss.

Aplikace dále potřebuje databázový server PostgreSQL ve verzi 9.5.

B.II Postup instalace

Založí se databáze roadtitan na databázovém serveru.

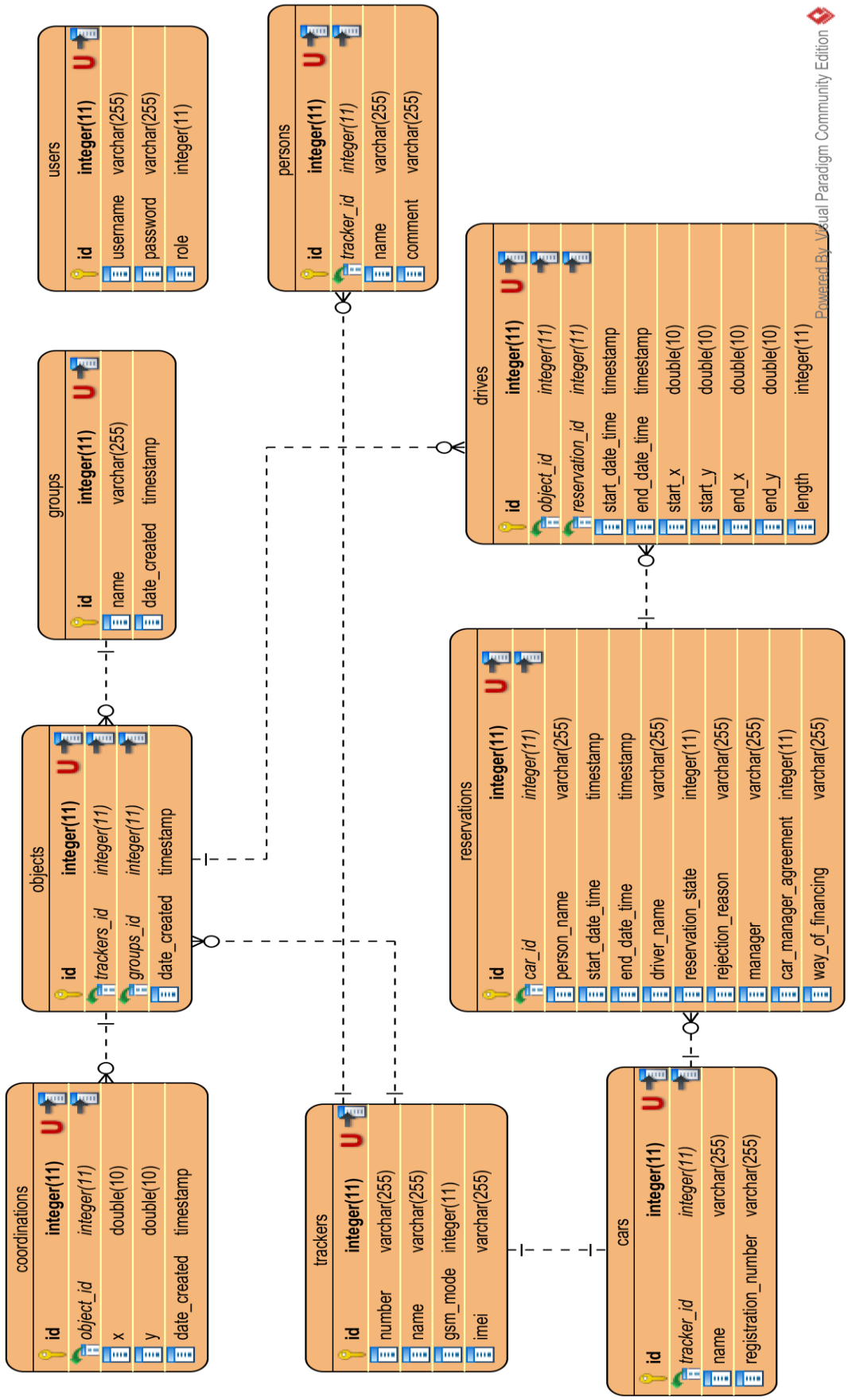
Upraví se konfigurační soubor DataSource.groovy, kde jsou parametry pro připojení k databázi.

Aplikace (soubor .war) se klasickým způsobem nahraje na aplikační server. Server se restartuje a spustí.

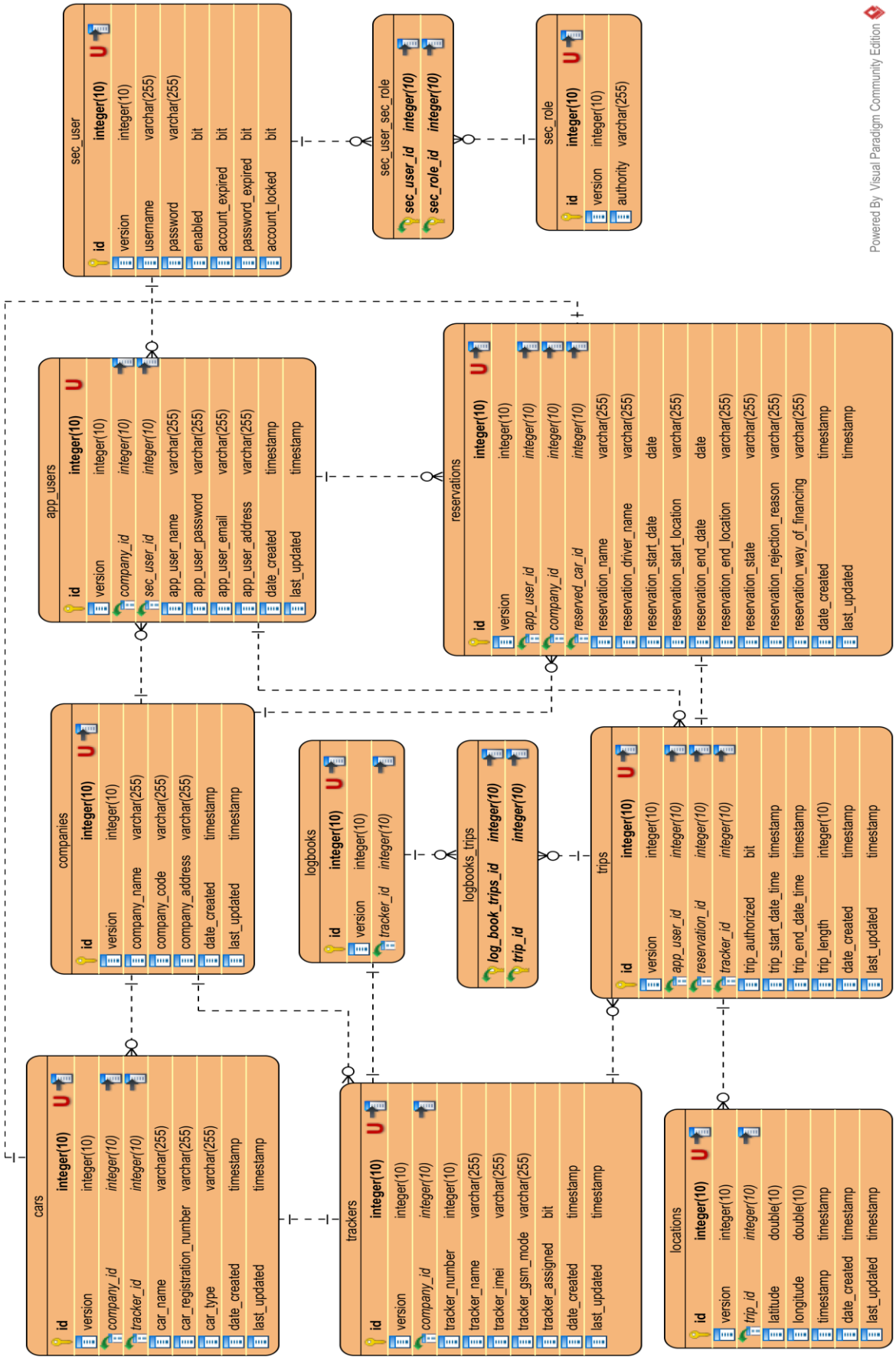
C Seznam pluginů frameworku Grails, použitých v aplikaci

- jadira
- jodatime
- Jquery
- Jquery UI
- Maplace.js
- Spring security core
- Spring security web
- Twitter Bootstrap

D Entitně-relační model databáze MotoTracker



E Entitně-relační model databáze RoadTitan



F Seznam použitých zkratek

CRUD – Create Read Update Delete

GPRS – General Packet Radio Service

GPS – Global Position System

GSM – Global System for Mobile Communications

HTML – HyperText Markup Language

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

HW – Hardware

MIT – Massachusetts Institute of Technology

MS – Microsoft

ORM – Object-relational mapping

SQL – Structured Query Language

SW – Software

G Seznam použitých obrázků

H Seznam použitých tabulek