Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Systém pro zpracování a vizualizaci polohy GPS signálu pro potřeby automaticky generované knihy jízd

Bc. Miloš Falta

Diplomová práce
2009

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra softwarových technologií
Akademický rok: **2008/2009**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Miloš FALTA**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Systém pro zpracování a vizualizaci polohy GPS signálu pro potřeby automaticky generované knihy jízd**

Zásady pro vypracování:

- V úvodní části práce bude představena problematika globálních pozičních systémů.
- Dále bude proveden úvod do problematiky systémové integrace aplikací ve vazbě na konkrétní potřeby výsledné aplikace.
- Výsledkem diplomové práce bude vytvoření hostované aplikace pro společnost RADOM s.r.o., která bude schopna jednoduchým a přehledným způsobem zpracovávat a vizualizovat polohu GPS pro potřeby plně automatického vytváření knihy jízd. Aplikace by měla být schopna výsledné knihy jízd zobrazovat a statisticky zpracovat. Hodnoty pro zpracování knihy jízd bude možné číst z více typů vstupních formátů.
- V diplomové práci bude kladen důraz na řešení problematiky automatického přiřazení popisných informací k získané poloze z GPS.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

- WELLENHOF B., LICHTENNEGGER H., WASLE E. GNSS - Global Navigation Satellite Systems: GPS, GLONASS, Galileo and More. Wien, Springer publishing, 2007, ISBN 978-3211730126

- JUŘEK M., Moderní integrace aplikací Praha, Microsoft, http://download.microsoft.com/download/8/6/c/86c09926-affc-4e14-bec0-3c45cd989436/Moderni_integrace.pdf

- On-line: http://code.google.com/more/products-geo-maps

Vedoucí diplomové práce: **Ing. Lukáš Čegan**
Katedra informačních technologií

Datum zadání diplomové práce: **31. října 2008**

Termín odevzdání diplomové práce: **22. května 2009**

L.S.

doc. Ing. Simeon Karamazov, Dr.
děkan

doc. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 3. listopadu 2008

Prohlašuji

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

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

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně Univerzity Pardubice.

V Pardubicích dne 20. 8. 2009 Miloš Falta

(vlastnoruční podpis)

Poděkování

Rád bych poděkoval vedoucímu diplomové práce Ing. Lukáši Čeganovi za odborné vedení a cenné rady při jejím zpracování. Další poděkování patří Ing. Petrovi Šebestovi za vysvětlení problematiky knihy jízd a seznámení s výpočetními postupy.

Poděkování patří také mým rodičům, kteří mi umožnili studovat tuto školu a všem, kteří mě podporovali během studia.

ANOTACE

Tato práce se zabývá vytvořením Internetové knihy jízd, pro vytváření knihy jízd a přehlednou evidenci záznamů z vozidlových jednotek. Práce charakterizuje práci na projektu Internetová kniha jízd, od počátku, kdy bylo nadefinované zadání, přes jednotlivé kroky a změny v návrhu a zpracování řešení až o současnou verzi, která dle posledních požadavků, bude ještě dále měněna dle přání zákazníků.

KLÍČOVÁ SLOVA

kniha jízd, RACAR¸ Zend, MVC, SXM30, vozidlová jednotka

TITLE

System for processing and visualization of the location of the GPS signal for the purpose of automatically generated book of tours

ANNOTATION

This work deals with the creation of Internet book of tours. Its purpose is creating book of tours and keeping clear evidence of the records from the vehicle units. This work characterizes the work on the project Internet book of tours, from the beginning, when the submission was defined, through the individual steps and changes in the design and processing of solutions to the current version, which according to recent requests, will be further modified according to customer requirements.

KEYWORDS

book of tours, RACAR, Zend, MVC, SXM30, GPS devices

Obsah

Seznam obrázků

Seznam použitých zkratek

Motohodiny - čas, kdy se vozidlo nepohybuje, ale musí mít nastartovaný motor kvůli sekundárním zařízením

GMT - (*Greenwich Mean Time* – greenwichský střední čas) - udává čas platný v časovém pásmu základního poledníku, který je založen na rotaci Země[8]

Know-how - technologické a informační předpoklady a znalosti pro určitou činnost - nejčastěji výrobu

RFID - je další generace identifikátorů navržených (nejen) k identifikaci zboží, navazující na systém čárových kódů

Úvod

Tato práce se zabývá vytvořením Internetové knihy jízd, pro vytváření knihy jízd a přehlednou evidenci záznamů z vozidlových jednotek. Současná desktopová verze je pro významné zákazníky nepoužitelná a proto se společnost RADOM, s.r.o. rozhodla vytvořit internetovou verzi knihy jízd.

Práce charakterizuje práci na projektu Internetová kniha jízd, od počátku, kdy bylo nadefinované zadání, přes jednotlivé kroky a změny v návrhu a zpracování řešení až po současnou verzi, která dle posledních požadavků, bude ještě dále měněna dle přání zákazníků. Důraz této práce je kladen na nutné napojení k systému RACAR, bez nutnosti jej jakkoli měnit. Na základě svých znalostí a zkušeností jsem navrhl a realizoval vlastní řešení problému.

Na počátku práce se zabývám analýzou zadání a popisem informací poskytovaných systémem RACAR. Poté jsem podrobněji zpracoval funkce systému se zaměřením na více zdrojů vstupních dat. Dále popisuji vozidlovou jednotku a vysvětluji fungování stávajícího systému RACAR pro použití z hlediska funkcionalit webové knihy jízd. V další části popisuji jednotlivé části knihy jízd rozdělené do vrstev modelu MVC. Každá z vrstev je složena ze tříd, které jsou v této části detailně popsány.

Dále jsou popsány jednotlivé možnosti webové knihy jízd a popis práce jak s ní efektivně pracovat. Závěr shrnuje dosažené výsledky a nabízí možnost dalšího rozšíření.

1 Analýza zadání

1.1 Požadavky na knihu jízd

Požadavky na webovou knihu jízd byly specifikovány s ohledem na stávající systém RACAR a poznatků z desktopové verze knihy jízd. Webová kniha jízd by měla obsahovat vše, co obsahuje současná verze knihy jízd a tuto funkcionalitu rozšířit o nové možnosti. Jsou nadefinované požadavky ohledně použitých technologií a podmínky pro korektní zobrazení uživatelům. Dále jsou nadefinovány vstupy a základní atributy záznamů, které je nutné vytvořit.

Základní požadavky

- serverová aplikace spustitelná odkudkoli, za použití webového prohlížeče a zařízení připojeného k internetu
- aplikace bude komunikovat s databází MySQL
- uživatelský jazyk: čeština
- webový server Apache ve verzi 2.0 nebo vyšší
- podpora PHP 5.0 nebo vyšší pro použití frameworku Zend
- podporované prohlížeče: IE7, Firefox
- minimální rozlišení 1024x768

Vstupem aplikace budou údaje ze zařízení GPS, které bude obsahovat

- datum a čas
- identifikaci vozidla
- GPS souřadnice
- rychlost
- výšku
- místo výskytu
- informace z přípojných zařízení

Tyto data budou uložena v databázi RACAR nebo v *.red souborech u zákazníka.

Aplikace má více možností provozu

- Aplikace bude nainstalována ve společnosti RADOM a bude využívat databázi RACAR.
- Alternativou pro zákazníky, kteří chtějí mít data u sebe je zde i možnost, že si aplikaci nainstalují na svůj počítač. Tím veškerá data budou mít ve své databázi. V tomto případě bude probíhat vše lokálně, připojení k internetu bude aplikace používat pouze pro vykreslení map.

Aplikace bude počítat atributy jednotlivých jízd, kterými jsou

- datum zahájení a ukončení jízdy
- vzdálenost
- doba jízdy
- motohodiny

Další rozšiřující vlastnosti

- Uživatelsky definovaná pozice, pro detailnější určování polohy.
- Aplikace bude schopna vytvářet výkazy za jednotlivý den nebo měsíc.
- Webová kniha jízd bude umět vytvořit a zobrazit statistické údaje vhodné pro zlepšení plánování a využití vozidel.
- Výpočet diet jednotlivých řidičů.
- Jízdy bude možné ručně editovat, nebo přidávat.
- Aplikace bude schopna jednotlivé jízdy zobrazovat na mapě a výslednou knihu jízd exportovat do souborů *.csv nebo *.txt

1.2 Funkce systému

1. Načtení dat z databáze nebo ze souboru. Takto načtená data se uloží do tabulky Pozice
2. Stisk tlačítka zpracuj, ověří rozmezí datumu a zjistí, zda jsou jízdy již vytvořeny, nebo zda se má nějaká jízda vytvořit nebo doplnit z nově zadaných dat. Pokud jsou nová data nalezena, otestují se, zda nespadají do uživatelsky definovaných lokalit a postupují k dalšímu zpracování. Záznamy se rozdělí podle identifikace vozidla. Zjistí se datum posledního záznamu. Záznamy, které jsou starší než toto datum, by měly být kompletní. Kompletní nebudou v případech, že jde o první jízdu, protože systém nemusí mít záznam o zahájení jízdy a stav kdy jízda z předchozího dne pořád pokračuje. Pokud se bude jednat o neukončenou jízdu, nebude pro výpočet použita. Započítána bude až po dodání zbylých údajů. Z takto vybraných bodů setříděných podle data nyní můžeme spočítat potřebné údaje pro uložení jízd do databáze. Jako první budu hledat stav zapnutí zapalování. Od této doby začíná jízda a pokračuje do doby, než je zapalování vypnuto nebo znova zapnuto. Z této množiny bodů se spočítají položky:
3. datum a čas zahájení a ukončení jízdy
   - odkud a kam
   - vzdálenost
   - doba jízdy
   - motohodiny
4. Tyto údaje budou následně uloženy do databáze Roads jako nový záznam se všemi doplňujícími atributy jako jsou řidič, účel jízdy, idVozidla, vytvoření a stav.
5. Rychlostní průměr se bude počítat až při výpisu.
6. Po uložení do databáze se aktualizují položky v tabulce Diety, kde se řidiči počítá čas a ujetá vzdálenost na daný kalendářní měsíc. Tyto údaje jsou potřebné pro výpočet diet – do budoucna jistě rozšířit výpočet o taxy za diety v jednotlivých státech.

7. Definice uživatelských poloh bude spočívat v ověření, zda daná poloha spadá do čtverce, který stanovují hraniční body uložené jako atributy la/lo min/max. Pokud ano, bude se pak podrobněji testovat, zda zvolený bod spadá do oblasti vytyčené uživatelem nebo nikoli. Vymezení lokality by mělo být uživatelsky přívětivé a to takové, že by si uživatel zobrazil mapu na libovolném místě s libovolným přiblížením. Poté by si myší v daném módu aplikace navolil požadované body okolo své lokality. Tyto body budou uloženy do databáze a očíslovány, jelikož u nich záleží na pořadí, v jakém byly zadávány.
8. Kontrola překročení rychlosti bude implementována tak, že si uživatel zvolí, jakou rychlost ještě toleruje a záznamy, kde bude zaznamenaná vyšší rychlost, tento záznam zvýrazní.
9. Detailní informace o jízdě, místo jízd se zobrazí seznam jednotlivých poloh pro danou jízdu, a jízda se zakreslí do mapy jako spojnice bodů, kudy vozidlo projelo.
10. Dalším požadavkem je třídění nejenom podle vozidla ale také podle řidiče.
11. Ruční editace jízd vytvoří nebo změní jízdu, bez změny vstupních poloh. To znamená, že při výpisu detailů jízdy, bude tato jízda zobrazena dle vstupních dat. Při změně jízdy se bude jízda zobrazovat nezměněná a jízda, která byla kompletně vytvořena uživatelem, nebude disponovat zobrazením v mapě, jelikož tato jízda nebude mít žádné polohy.
12. Aplikace bude obsahovat výpis za jednotlivý den, ve kterém bude zobrazovat, zda byl zapnut nějaký ze vstupů.
13. Webová kniha jízd bude mít jednotlivá zobrazovaná slovní spojení v externím souboru, pro jednodušší překlad do jiných jazyků.

1.3 Analýza vstupních dat

Vstup z databáze

Pro vstup z databáze mi postačí 4 tabulky ze současného systému a databáze vlastního systému, kde si budu udržovat své vlastní informace. Jelikož aplikace nesmí ovlivňovat stávající systém, budou data v této databázi sloužit pouze pro čtení. Jedná se především o přístup k tabulce uživatelů, abych mohl porovnávat přihlašovací údaje shodně se systémem RACAR. Dále o tabulku vozidel a vazební tabulku mezi uživateli a vozidly. Jako poslední je zde tabulka, kam jsou ukládány záznamy.

Vstup ze souborů

Alternativa pro zákazníky, kteří si chtějí data zpravovat sami, je zde možnost používat webovou knihu jízd lokálně. Zákazník pak ale nemá přístup do databáze systému RACAR. Pro tuto možnost jsou veškeré tabulky uloženy i lokálně. Tabulka se záznamy je v tomto případě nahrazena lokální tabulkou, a ukládají se do ní data z binárních souborů, poskytnutých společností RADOM s.r.o.

1.4 Představa o funkci knihy jízd

Představa o funkci je taková, že načtená data se seřadí podle času vzniku. Pokud není prvním záznamem zapnutí zapalování, jedná se o předchozí jízdu, a tak systém vyhodnotí první záznam jako zapnutí zapalování. Poté následují data a jízda je ukončena záznamem o vypnutí zapalování nebo zapnutí zapalování (prvotní oprava dat). Nyní jsem ze skupiny záznamů vyfiltroval pouze ty, které tvoří jízdu. Podle časového záznamu první a poslední položky je dáno časové rozmezí jízdy. Z GPS údajů se mezi každým záznamem spočítá ujetá vzdálenost a ta se sumarizuje pro celou cestu. Podobně se vypočítají i zbylé údaje jako například spotřeba paliva.

1.5 Kam, co a jak exportovat

Export dat je dán výběrem, který si uživatel zvolí ve filtru na stránce. Zvolí si vozidlo a časové údaje na filtru položek a nechá si zobrazit výsledek. Pokud se mu bude tento výsledek líbit, může jej exportovat. Export do csv by měl odpovídat normě RFC 4180.

2 Použité technologie

K vytvoření webové knihy jízd bylo zapotřebí použití skriptovacího jazyka. Z důvodu kompatibility jsem zvolil jazyk PHP pro zpracování skriptů na straně serveru a JavaScript ke zpracování skriptů na straně klienta. Následně bylo nutné data ukládat do relační databáze. Jako databázový server jsem zvolil MySQL, který je na serveru již nainstalován.

2.1 Skriptovací jazyk PHP

PHP je v současnosti velmi rozšířená technologie umožňující snadné programování na straně serveru (server-side programming). Toho lze využít k tvorbě různých interaktivních webových stránek. Stručně lze říci, že skript napsaný v PHP je proveden na serveru podle zadaných kritérií a výsledek je odeslán volajícímu počítači stejným způsobem, jakým se odesílají běžné statické (XHTML) stránky. Jakmile je však stránka načtena klientem, pomocí PHP ji již není možné dále měnit.

Změnu stránky bez nutnosti jejího opětovného načtení provádíme skripty na straně klienta - například pomocí JavaScriptu. Výhoda těchto programovacích jazyků spočívá tedy v tom, že není třeba pro změnu stránky neustále obnovovat obsah stránky. Jejich veliké omezení však spočívá nejen v možnostech, ale také v jejich (ne)bezpečnosti. Protože skripty se vykonávají přímo na počítači uživatele, mohl by potenciální útočník bez problémů vykonat nebezpečný

kód. Z tohoto důvodu jsou například v JavaScriptu vypnuty funkce, které by přímo pracovaly se soubory na harddisku. Díky tomu se JavaScript využívá spíše pro okamžitou úpravu CSS kódu či jako "doplněk" při vytváření webové grafiky. Pro "opravdové" funkce internetových stránek se ale používají skripty na straně serveru.

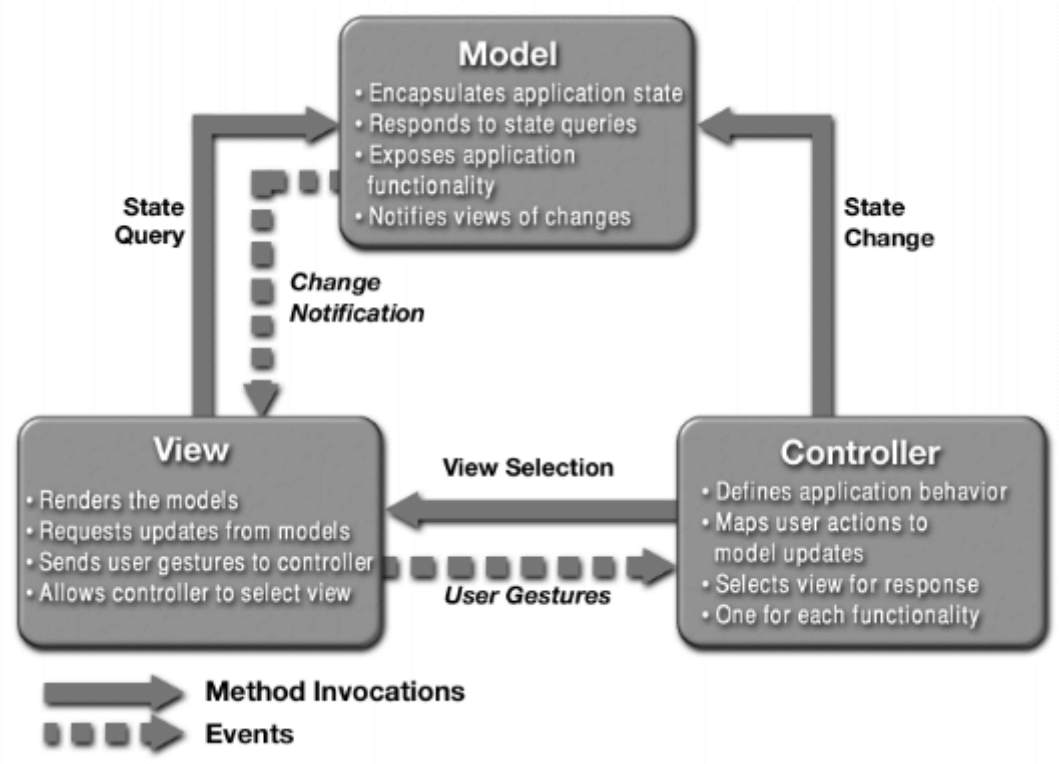
PHP je programovací jazyk umožňující procedurální i objektově orientované programování. Znalost objektově orientovaného programování (OOP) tedy může být při práci v PHP výhodou, není však nutnou podmínkou. PHP také patří mezi jazyky, kde například není nutné předem definovat typ proměnných, navíc jakákoli proměnná může

kdykoli změnit svůj typ. Jednoduše řečeno, co se týče psaní kódu, z PHP při psaní skriptů "sálá" určitá volnost a neomezenost. Na druhé straně záleží plně na programátorovi, jaký si bude v kódu udržovat pořádek.

[1]

2.2 Využití modelu MVC pomocí frameworku Zend

Podstata architektonického návrhového vzoru MVC spočívá v oddělení vzhledu od aplikační logiky a od dat. Tento model tyto tři vrstvy odděluje a kód je tak přehlednější. Jednotlivé vrstvy by o sobě navzájem neměly vědět. To je zajištěno tak, že vrstva aplikační může komunikovat s okolními vrstvami. A ty tak o sobě navzájem nevědí ani nepotřebují vědět.



Obrázek 1 - MVC [8]

V Zend frameworku je pro vrstvu aplikační logiky používán název Controller a pro oblast vzhledu název View. Vrstvě View je jedno kde Controller data vzal, podstatné je pro ni pouze to, že data dostane. Databázi je jedno, jak budou data reprezentována, pouze zpracuje požadavek z Controlleru a vrátí data zpět. Získaná data se upraví a zašlou View vrstvě, která už jen data vezme a zobrazí je v požadovaném formátu.

Popis jednotlivých vrstev:

- Controller se používá ke zpracování akce uživatele.
- Modelem jsou myšleny takové třídy, které nám nabízí přístup k datům, nebo nějaké vlastní funkcionality systému.
- View pracuje s daty, které ji předal Controller, ovšem i sama může využít možnosti modelu ke změně vypisovaných dat.

2.3 Databáze MySQL

MySQL je švédský databázový server založený na jazyce SQL. Je k dispozici jako open source, tedy program šířený zdarma. K dalším výhodám MySQL patří podpora všech hlavních platforem, vysoký výkon i rychlost a vynikající kompatibilita s jinými systémy, zejména se serverovým programem Apache a skriptováním PHP (dohromady tvoří tzv. **triádu**, trojici programů nejčastěji instalovanou k vytváření databázových aplikací). MySQL se také díky své relativní jednoduchosti poměrně snadno učí. Díky těmto vlastnostem se databázový server MySQL prosadil jako univerzální řešení používané na většině internetových projektů a je automaticky dostupná téměř na všech typech webhostingu.

Nevýhody MySQL pramení z jejích výhod. Nepodporuje složitější programátorské konstrukce (někdy je možné je obcházet skriptováním) a nemá dostatečný výkon v opravdu náročných (zatěžovaných) webových aplikacích. Tehdy se používají konkurenční databáze, například PostgreSQL nebo Oracle. Přesto je však třeba říci, že MySQL vyhoví ve většině případů.

2.4 JavaScript

JavaScript *je multiplatformní, objektově orientovaný skriptovací jazyk, jehož autorem je Brendan Eich z tehdejší společnosti Netscape.*

Nyní se zpravidla používá jako interpretovaný programovací jazyk pro WWW stránky, často vkládaný přímo do HTML kódu stránky. Jsou jím obvykle ovládány různé interaktivní prvky GUI (tlačítka, textová políčka) nebo tvořeny animace a efekty obrázků.

Jeho syntaxe patří do rodiny jazyků C/C++/Java. Slovo Java je však součástí jeho názvu pouze z marketingových důvodů a s programovacím jazykem Java jej vedle názvu spojuje jen podobná syntaxe. JavaScript byl v červenci 1997 standardizován asociací ECMA (Europen Computer Manufacturers Association) a v srpnu 1998 ISO (International Organization for Standardization). Standardizovaná verze JavaScriptu je pojmenována jako ECMAScript a z ní byly odvozeny i další implementace, jako je například ActionScript.

JavaScript byl původně obchodní název implementace společnosti Netscape, kde byl vyvíjen nejprve pod názvem Mocha, později LiveScript, ohlášen byl společně se společností Sun Microsystems v prosinci 1995 jako doplněk k jazykům HTML a Java. Pro verzi firmy Microsoft je použit název ***JScript****. JScript je podporován platformou .NET.*

Program v JavaScriptu se obvykle spouští až po stažení WWW stránky z Internetu (tzv. na straně klienta), na rozdíl od ostatních jiných interpretovaných programovacích jazyků (např. PHP a ASP), které se spouštějí na straně serveru ještě před stažením z Internetu. Z toho plynou jistá bezpečností omezení, JavaScript např. nemůže pracovat se soubory, aby tím neohrozil soukromí uživatele.

[4]

3 Popis stávajícího fungování systému

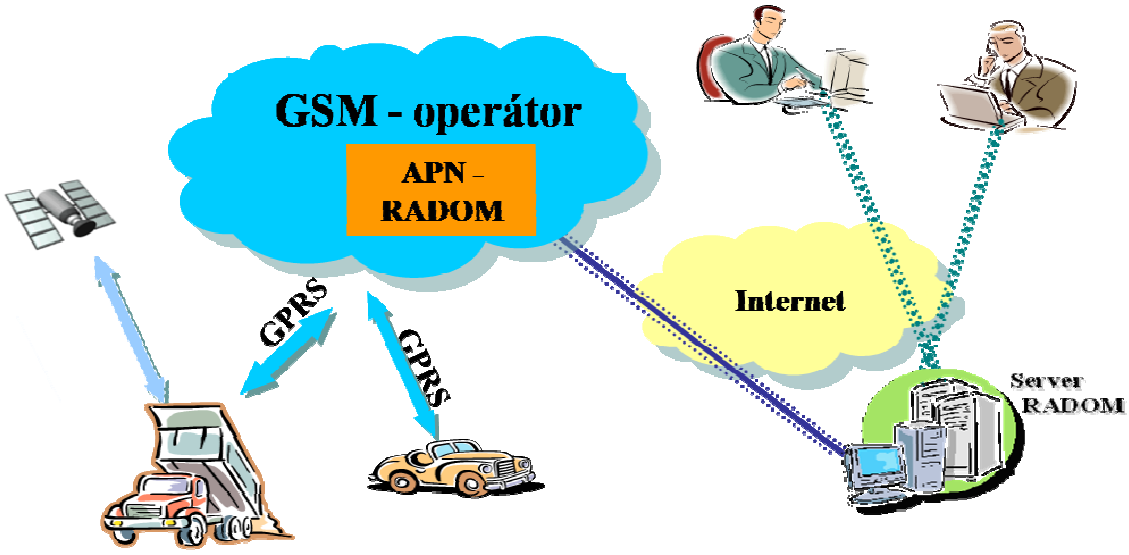
3.1 Globální pohled na systém

Aby mohlo být vozidlo sledováno, je nutné ho vybavit zařízením, které bude umět přijímat GPS signály a odesílat je na server, kde budou data zpracována a poskytnuta k dalšímu zpracování. Ze získaných dat je možné jednotlivé záznamy zobrazovat na mapě matematicky a statisticky zpracovávat.

Pro získání dat je ve vozidlech instalována vozidlová jednotka. Vozidlová jednotka je složena z vozidlové soupravy typového označení SXM30-D. Vozidlová jednotka se skládá z GPS modulu, GSM modulu, baterky, 1MB flash paměti a konektorů.

Veškerá vozidla jsou pomocí SIM karet českého mobilního operátora napojena na server společnosti RADOM, s.r.o. Po předchozí autorizaci mají uživatelé zajištěn přístup k záznamům. Omezení v počtu sledovaných vozidel zde neexistují.

Na serveru běží aplikace komunikující s připojeným GSM modemem. Aplikace přijme zprávy ze zařízení, zpráva je následně dle specifického protokolu převedena na záznam s patřičnými parametry a tento záznam je uložen do databáze pro další zpracování a snadnou archivaci záznamů. Pro zákazníky vlastnící deskopovou knihu jízd jsou ještě generovány binární soubory se záznamy.



Obrázek 2 - Fungování systému RACAR

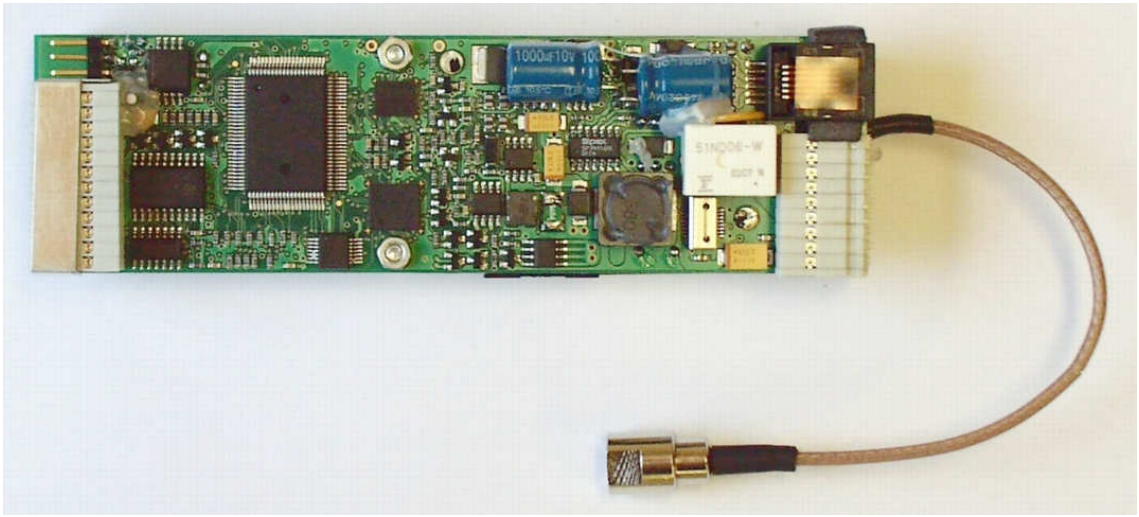
Z těchto záznamů následně vychází monitorovací systém RACAR. Systém RACAR je konečným systémem umožňujícím sledovaní aktuální polohy vozů. Je to finální část celého systému. K systému se poté připojují uživatelé, aby viděli, kde se nacházejí jejich vozidla. Připojení k internetové aplikaci, umožňující sledovat provoz vozidel, nevyžaduje instalaci dalšího software. Tím je zajištěno, že přihlášení k aplikaci je možné z kteréhokoliv PC připojeného k internetové síti.

Po přihlášení je možné sledovat jak on-line provoz vozidel, tak zobrazovat pohyb vozidel v definovaných časových úsecích z archivu. V on-line zobrazení jsou na mapě viditelná všechna vozidla přiřazená účtu uživatele, přičemž přiblížením mapy lze zobrazení zúžit na náhled na konkrétní vůz. Kromě základních informací je možné po otevření příslušného okna zobrazit detailní informace o vozidle a jeho stavu.

Vstupy vozidlové jednotky umožňují monitorovat a přenášet stavy zařízení či vybavení vozidla. Takto lze získat přehled např. o otevření dveří nákladového prostoru, použití či době běhu nástaveb vozidla a dalších.

3.2 Vozidlová jednotka

Vozidlová jednotka posílá pravidelně hlášení o své poloze. Doba hlášení je nastavena jak pro vozidlo, které se nepohybuje, tak pro vozidlo aktuálně jedoucí a speciální případ alarmového stavu. Nejčastěji se používá denní interval pro vozidlo stojící. Jedoucí vozidlo má nejčastěji interval každou druhou minutu a v alarmovém stavu je hlášení posíláno každých 10 vteřin. Mimo tyto intervaly je ještě možné vyžádat si od jednotky poslání aktuální polohy.



Obrázek 3 - Plošný spoj vozidlové jednotky

Vysíláním lokalizačních SMS dotazů lze kdykoliv ověřit aktuální polohu vozidla bez ohledu na nastavenou četnost automatických hlášení vozidla. Současně je tato funkce nezbytná pro případné dohledání vozidla.

Blokování zařízení vozu se provádí jejich ovládáním prostřednictvím výstupů vozidlové jednotky. Cílem je zamezení použití vozidla neoprávněným uživatelem. Takto lze například odpojit elektroinstalaci od napájení, přerušit dodávku paliva do motoru atd.

Obrázek 4 - Vozidlová jednotka SMR30-D

Předání informace o napadení vozidla je zajištěno neprodleným vysláním alarmových SMS zpráv na předem určená telefonní čísla. Tímto způsobem je možné zapojit vozidlo do systému střežení pultů centralizované ochrany. Alarmové SMS mohou být vysílány jednorázově nebo v periodických intervalech. K informování o napadení vozu je možno dále použít zařízení vozidla, která jsou ovládána vozidlovou jednotkou, např. výstražná světla, sirény, autoalarm …

3.2.1 Technické parametry vozidlové jednotky

- Napájecí napětí 10,8 až 31 VDC
- Záložní akumulátor olověný 6VDC/1,3Ah
- binárních vstupů
- 3 výstupy, max. proudová zatížitelnost 6A
- Integrované moduly GPS, GSM
- Rozhraní pro připojení antény čtečky ID čipů
- Konfigurační rozhraní RS 422, RS 232
- Provozní teplota -40 až +85 °C
- Rozměry 125x61x27mm
- Hmotnost 0,28kg
- Stupeň krytí IP40
- Certifikace dle E8, č. homologace 97 RA-01 3837

3.2.2 Přípojná zařízení

Přípojná zařízení jsou zařízení, které mohou spolupracovat s vozidlovou jednotkou. V současné době se používají přípojná zařízení:

- Hladinový palivoměr
- Průtokový palivoměr
- RFID čtečka karet řidiče

Jednotlivé komponenty jsou připojeny na vstupy vozidlové jednotky, která následně informace z těchto zařízení posílá současně s daty o poloze. Proto je ve specifikaci komunikace vozidlové jednotky se serverem proměnná délka bloku dat, v závislosti na počtu připojených zařízení.

3.3 Proč se vlastně tvoří kniha jízd

Zpracovávat knihu jízd jsou povinni všichni, kteří mají vozy v obchodním majetku, vypůjčené, v nájmu a pořízené na leasing. Konkrétně se jedná o zákon č. 111/1994 Sb. a vyhláška č. 187/1994 Sb.

V současné době je ve firemním vozidle papír, kam se každý řidič zapíše, napíše start a cíl cesty, kilometry z tachometru na začátku jízdy společně s datem a časem. Po dojezdu na místo doplní aktuální stav kilometrů z tachometru, čas dojezdu a podpis, kterým stvrzuje správnost údajů.

V dnešní době digitalizace je možné vytvářet knihu jízd elektronicky. Systémy pro tvorbu knihy jízd prozkoumají záznamy a vygenerují všechny požadované položky. Uživateli tak poté stačí zadat důvod jízdy. Tato položka musí být vyplněna, aby byl dokument uznán jako kniha jízd.

Výhodou ručně psané knihy jízd je fakt, že na začátku nového měsíce stačí obejít vozidla a vyplněné papíry za minulý měsíc sebrat a je hotovo. Elektronická kniha jízd je vytvořena automaticky a stačí pouze doplnit údaje, které knize jízd nejsou známé, jako důvody jednotlivých jízd apod. Výhodou elektronické knihy jízd jsou přípojná zařízení, pomocí kterých lze zjistit odčerpávání pohonných hmot z firemního vozidla. Další výhodou je odhalení řidiče, jenž si bude dělat nadměrné množství přestávek, aby mohl prokazovat delší dobu na služební cestě. Obdobně lze kontrolovat, kudy řidič jezdil, zda nejezdí zbytečně delší trasy než by musel.

Výhody internetové knihy jízd

Pro přístup ke knize jízd stačí počítač s připojením k Internetu bez nutnosti investic do speciálních softwarů a v případě vykreslování jízd ani nákladů spojených s pořízením mapových podkladů. Jednodušší je i delegování přístupu dalším subjektům, kterým stačí předat přihlašovací informace. Není nutné na jejich počítačích cokoli instalovat a konfigurovat pro bezproblémový provoz. Podstatnou výhodou je přístup odkudkoli, jelikož jsou záznamy na serveru a nikoli na lokálním počítači. Neustálá aktualizace je prováděna na serveru, čímž zákazník ušetří za náklady spojené s aktualizací a spravováním lokální verze.

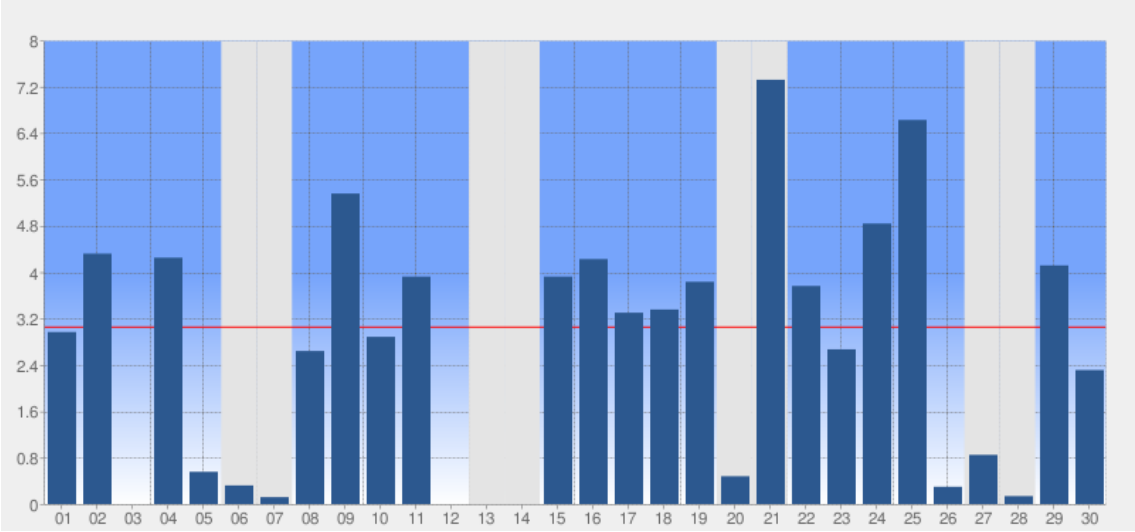
3.4 Účely knihy jízd

Primární

Primárním účelem knihy jízd je vytvářet záznamy o provozu vozidla, v kolik a kde se vozidlo pohybovalo. Z těchto údajů lze exportovat výpisy pro finanční úřad, jako náhradu za klasickou knihu jízd. Následně lze také kontrolovat zaměstnance. Z archivu záznamů lze získat informace dokazující prohřešek řidiče.

Sekundární

Účelem sekundárním se postupem času staly veškeré statistické analýzy ohledně nákladů a využitelnosti vozů. Rozšiřující vlastností internetové knihy jízd je tvorba grafů, ze kterých je na první pohled patrné využití vozu v jednotlivých dnech, nebo jednotlivých částech dne. V denních statistikách je možné zobrazit jednotlivé vstupy a výstupy vozidlové jednotky. V měsíčních statistikách jsou zobrazeny průměrné hodnoty a hodnoty využitelnosti vozu, ujetých kilometrů, spotřeby paliva apod. v jednotlivých dnech.



Obrázek 5 - Graf měsíčního záznamu

Dalším neocenitelným údajem je využití vozu v rámci dne. U každého záznamu je uveden čas, v kolik hodin vozidlo v daný den poprvé vyjelo a kdy skončilo svou poslední jízdu společně s časem nevyužitelnosti vozidla mezi těmito časy. Z těchto údajů lze následně optimalizovat vozový park a využitelnost jednotlivých vozidel.

3.5 Propojení knihy jízd se systémy třetích stran

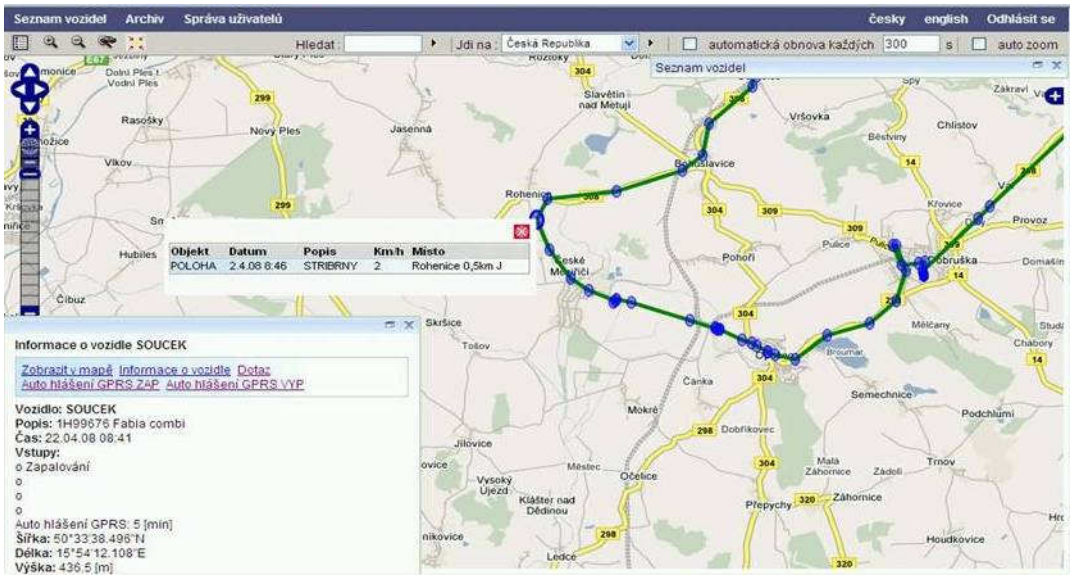
Kniha jízd není samostatně fungující systém. Už pro získání dat se využívá okolí knihy jízd. Kniha jízd používá zdrojová data umístěna v databázi serveru nebo v binárním souboru. Oba tyto zdroje dat jsou tvořeny na serveru společnosti RADOM. Prvním externím systémem se tedy stává systém RACAR pro zobrazení tras v mapě. Druhým systémem jsou data od společnosti CCS. Ty měl původně systém sám stahovat, ale společnost CCS soubory nedovoluje automaticky stahovat vzhledem ke konkurenční výhodě jejich monitorovacího systému. Proto záznamy o placení u benzínových pump musí uživatelé ručně do knihy jízd vkládat.

Třetím systémem je server google.com, který je využíván pro vykreslování grafů. Poslední vliv na systém mají jednotlivé druhy zařízení, do kterých se vozidlové jednotky instalují a jejich specifické požadavky. Specifickým zákazníkem je mobilní míchačka, kde zákazník chce sledovat, zda vůz během jízdy opravdu míchá, nebo nikoli. Další zákazník naopak potřebuje vypočítat dobu, kdy zásahové vozidlo přijede na místo zásahu a po dojezdu musí nechat nastartovaný motor z provozních důvodů. Zákazníka by zajímal čas, který vozidlo strávilo jízdou a jak dlouho zasahovalo. Nebo taxikář potřebuje odlišit jízdy, kdy veze nějakého zákazníka, od jízd, kdy jezdí mezi taxikářskými nástupišti.

3.6 Analýza stávajícího systému RACAR

3.6.1 Základní funkčnost systému RACAR

Začal jsem spuštěním a prozkoumáním systému RACAR na webových stránkách http://www.racar.cz. Jako první je nutné se přihlásit, přes přihlašovací formulář. Data jsou odeslána pomocí http nikoli zabezpečeně pomocí https. Https je síťová nadstavba protokolu http, kde jsou přenášená data šifrována pomocí SSL nebo TSL na port 443. Při bližším zkoumání jsem zjistil, že jsou data kódována jiným způsobem.



Obrázek 6 - Systém RACAR

Po zobrazení úvodní stránky je vidět menu obsahující systémové logo. Následuje výpis údajů přihlášeného uživatele, jako je jeho celé jméno a počet vozidel, se kterými může v systému pracovat. Dalším funkčním blokem v hlavní liště jsou tlačítka pro jednotlivé části systému. Jsou to jednotlivé dílčí části systému. Jedná se o seznam vozidel, správu archivu, správu uživatelů. Posledním tlačítkem je legenda ke značkám vozidel, u kterých se rozlišují následující stavy:

- vozidlo má zapnuté zapalování
- vozidlo má vypnuté zapalování
- vozidlo je v alarmovém stavu
- vozidlo má nízké napětí na akumulátoru

Pod těmito tlačítky jsou funkční tlačítka pro pohyb a práci s mapovými podklady. Dále systém umožňuje hledání jednotlivých aut a automatickou obnovu, která zajistí, že vozidlo bude po tomto čase zobrazeno na aktuálnější pozici. Posledními ovládacími prvky v hlavní liště je změna jazyka a odhlášení ze systému.

Ovládací prvky nad mapovým podkladem umožňují přibližování a posun po mapě a výběr mapového podkladu. Dříve byl použit mapový podklad serveru Google Maps, který změnil licenční podmínky a tento mapový server již není pro komerční použití bezplatný. Proto se firma rozhodla vybudovat vlastní mapový server, který souběžně umožňuje vyšší komfort pro zákazníka. Uživatel si může sám zvolit, kterou z map chce použít. Nabízeny jsou mapy Evropy, České republiky, Prahy, Brna a podrobná mapa. Na tomto mapovém podkladu lze ještě zvolit, které informace zobrazovat. Dále bych se chtěl zabývat jednotlivými složkami systému a jejich případného využití pro knihu jízd. Pokud dovolíte, začal bych seznamem vozidel.

3.6.2 Moduly systému RACAR

Seznam vozidel

V seznamu vozidel je uveden seznam všech vozidel, se kterými může daný uživatel disponovat a současně s tímto údajem se zobrazuje i poslední známá pozice vozu s časovým údajem a rychlostí.

Zobrazení záznamu z archivu

Tato možnost je zde, pro zobrazení pohybu vozidla za určité období. Maximálně však období jednoho měsíce. V okně si lze jednoduše zvolit, za jaké období se mají záznamy hledat. Nutnou podmínkou úspěšného hledání je volba, pro které(á) vozidlo(a) záznamy hledám. Po odeslání se načtou data o jednotlivých nahlášených pozicích a zobrazí se v mapě. V okně archivu se zobrazí pozice s údaji spojenými s daným hlášením. Je zobrazen čas, kdy se záznam odeslal, o jakou událost se jedná, aktuální rychlost, místo, ze kterého byla informace odeslána a stavy vstupů vozidlové jednotky.

Správa uživatelů

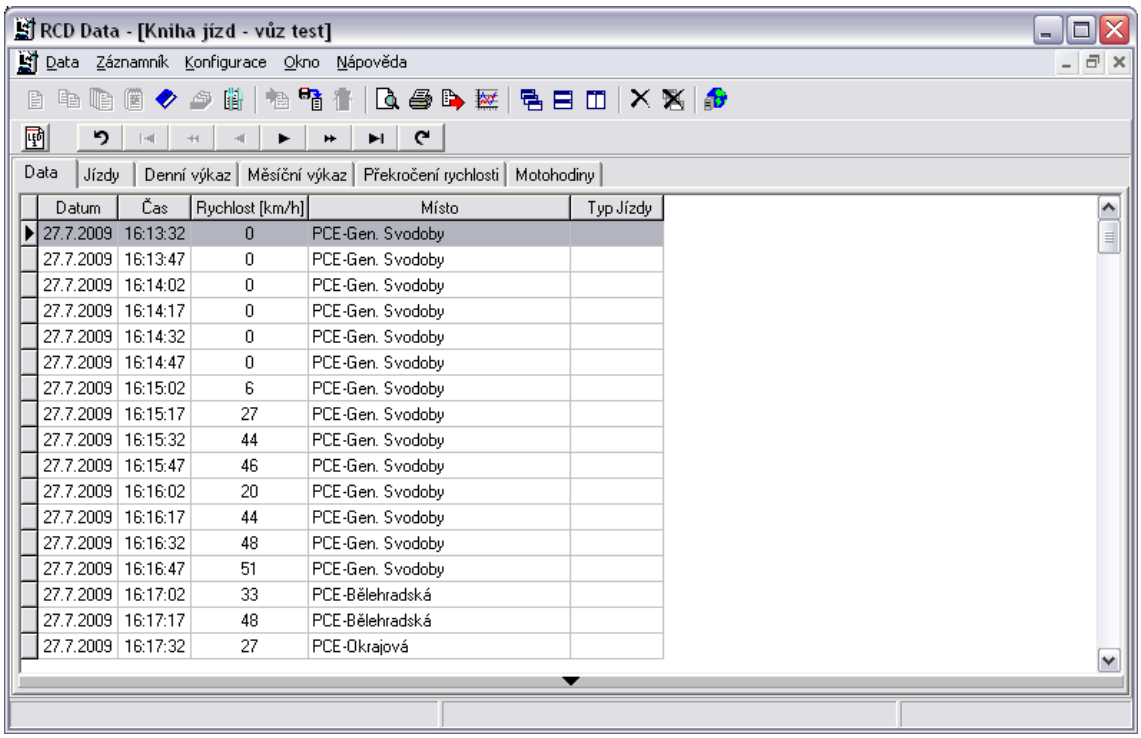
Při stisku tohoto tlačítka se zobrazí tabulka uživatelů. Je zde aktuálně přihlášený uživatel a dále všichni uživatelé, kteří jsou v hierarchii uživatelů pod tímto uživatelem. Aktuální uživatel nemá právo se smazat a z editací je mu dovolena pouze změna vlastního hesla. Další funkcí je zadávání nových uživatelů, které jsou automaticky přiřazeny k tomuto účtu. Je možné jim zvolit další parametry, kterými rodičovský účet disponuje. Restrikci oprávnění lze provést na vytváření nových uživatelů, nastavení limitů SMS dotazů, přístup do archivu a další. Pro lepší kontrolu nad účty lze zvolit i rozmezí platnosti účtu.

Každému vytvořenému účtu lze přidělit vozidla, ke kterým má přístup přihlášený uživatel. Filosofie je taková, že administrátor má oprávnění na všechna vozidla a každý jím vytvořený uživatel má přístup ke stejným vozidlům, nebo jen k jejich části. Lze tedy říci, že s rostoucím počtem hierarchií je počet přiřazených vozidel menší, popřípadě stejný.

S přidělením map je situace úplně stejná. Lze dále přidělovat pouze mapy, které mám již přiděleny. A tento uživatel pak uvidí vozidla pouze na mapě, kterou měl přidělenou. Nastane-li situace, že nemá žádnou, vozidla neuvidí.

3.7 Stávající desktopová kniha jízd

Stávající desktopová kniha jízd je nainstalována u klienta a stahuje si soubory ze serveru RACAR nebo je uživatel zkopíruje do zvoleného uložiště, ze kterého jsou data následně zpracována. Data se zpracovávají po spuštění desktopové knihy jízd. Po jejich zpracování je možné data zobrazit, procházet je, vytvářet z nich grafy a tato data exportovat.

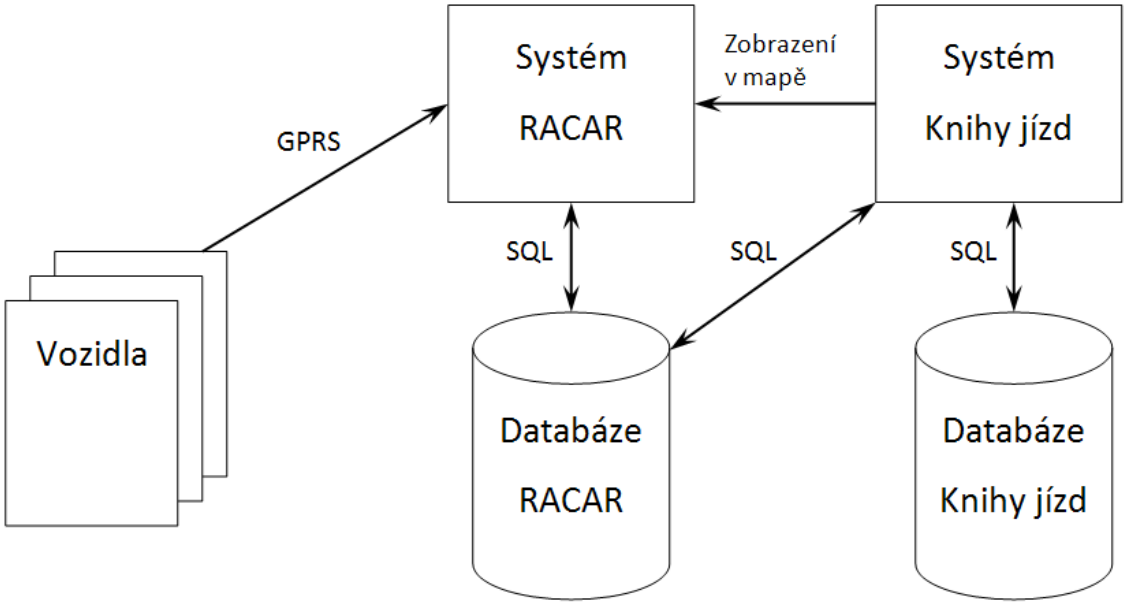


Obrázek 7 - Desktopová kniha jízd

Desktopová kniha jízd má lepší možnosti exportování a zobrazení grafů oproti webové knize jízd. Desktopová verze knihy jízd může libovolně pracovat se soubory na disku a tím zaručen i export dat. Ve webové aplikaci je možné obdobně soubory vytvořit na serveru, ale je problém takové množství souborů doručit ke klientovi. Pokud chce uživatel export z jednoho vozidla, není problém mu soubor zpřístupnit, aby si jej stáhl. Problém nastává, pokud uživatel chce exportovat například 100 vozidel. Pokud soubory na serveru zabalím a poté odešlu zákazníkovi, nastává problém s rozbalením archivu a zobrazit uživateli 100 odkazu ke stažení mi také nepřijde rozumné. Z tohoto důvodu je tento problém zatím nevyřešen.

4 Návrh architektury webové knihy jízd

Návrh webové knihy jízd je od počátku stavěn tak, aby neovlivnil funkčnost systému RACAR. Proto kniha jízd využívá systém RACAR pouze pro získání dat a popřípadě vykreslení jízdy jak zobrazuje Obrázek 8. Dále se v návrhu zabývám propojení tabulek mezi jednotlivými databázemi, protože musí být na sobě nezávislé.



Obrázek 8 - Zapojení knihy jízd k systému RACAR

4.1 Databázový pohled

Jako základ systému jsou použity tabulky uživatelů a vozidel, spojené vazební tabulkou, aby bylo možné vytvořit vazby m:n. Protože je nutné zajistit, aby více osob mohlo sledovat jedno vozidlo, a zároveň jedna osoba může mít dohled nad více vozidly. Dále je zde tabulka dat z vozidlových jednotek, která má sloupek identifikující o jaké vozidlo se jedná. Unikátnost je dána autoinkrementačním sloupkem, ale jedinečnost je dána spojením čísla vozidla, datumu záznamu a typu záznamu.

To je vše k datovému modelu systému RACAR. Nyní se začnu zabývat tím, jak tyto informace využít, aby nedocházelo k duplikování záznamů a uživatelé mohli používat shodné přihlašovací údaje apod.

V začátku mi je jasné, že tabulka vozidel, se bude muset rozšířit o údaje, které nejsou potřebné pro stávající systém. Jde o položky tachometru, spotřeby paliva a podobné. Webová kniha jízd bude používat shodné přihlašovací údaje a tudíž i shodnou tabulku. Rozšíření vlastností uživatele zatím není nutné, a proto tuto tabulku nebudu rozšiřovat. Jen pro upřesnění, webová kniha jízd je oddělitelná od stávajícího systému a nesmí do něj jakkoli zasahovat. Ani databázový stroj pro knihu jízd nemusí běžet na stejném stroji, a proto jsou tabulky v jiné databázi, aby bylo jasné jejich oddělení. Z tohoto důvodu jsem se rozhodl rozšíření tabulky vozidel vyřešit další tabulkou, která bude mít stejný primární klíč s původní tabulkou. Dále je nutné nadefinovat číselníky a jejich propojení. Jako číselníky je nutné evidovat řidiče a druhy jízd. Nejlepší řešení, které mě napadlo, bylo použít skupiny. Ty jsem pracovně pojmenoval jako společnosti, s tím, že pro každou společnost bude několik uživatelských účtů, několik vozidel, několik řidičů a několik druhů cest. Všechny položky budou tímto svázány a zamezí se tak přístupu k datům jiného subjektu.

Dále je nutné vytvořit tabulky přímo pro záznamy knihy jízd, kde jsou data uložena v tabulce pro pozice, jízdy, denní záznamy a měsíční záznamy. Poslední dvě tabulky jsou shodné a v měsíčních záznamech jsou pouze sumáře za dané období. Zvolil jsem tabulku místo možnosti data neustále vypočítávat a to z toho důvodu, že tyto informace jsou podstatné a jsou odrazovým můstkem pro jakoukoli další práci v knize jízd. Pro tak časté použití mi přišlo vhodnější mít data uložena v databázi. Databázový model je v příloze číslo 1.

Jedna z podmínek je samostatnost aplikace. To v případě, že by kniha jízd byla nainstalována bez okolních systémů, a po zadání vstupních dat ze souborů, je dokázala zpracovat a korektně vyhodnotit a zobrazit uživateli výsledky. Vše bez použití přihlašovacích údajů z jiného systému a podobně. Z tohoto důvodu jsou v databázi také tabulky pro uchování důležitých dat pro tento samostatný chod webové knihy jízd.

4.2 Načítání souborů jako zdroj dat

Soubory s daty potřebnými pro knihu jízd jsou uloženy v binárním souboru, který se skládá z hlavičky a bloků dat. Každý blok dat reprezentuje jeden záznam. Bloky mají různou délku, v závislosti na použitých zařízeních spojených s vozidlovou jednotkou a na délce hodnot v záznamu uložených. Například nejdříve je v dvoubajtové proměnné uložena délka řetězce jako proměnná typu integer následovány bajty daného řetězce.

Soubor byl uložen aplikací psanou v prostředí Delfi a přejímá i její nastavení pro ukládání binárních dat. Vyčítání dat byl prvotní úkol, jelikož jsem mohl pouze tušit, jaká data nakonec dostanu.

4.2.1 Problémy při vyčítání dat.

Prvním problémem, který mě potkal, bylo hned vyčítání hlavičky. Z názvu souboru mi bylo číslo vozu známo, ale nemohl jsem ho dohledat nikde v binárních datech. Problém byl v tom, že každé 2 bajty se při ukládání prohodí.

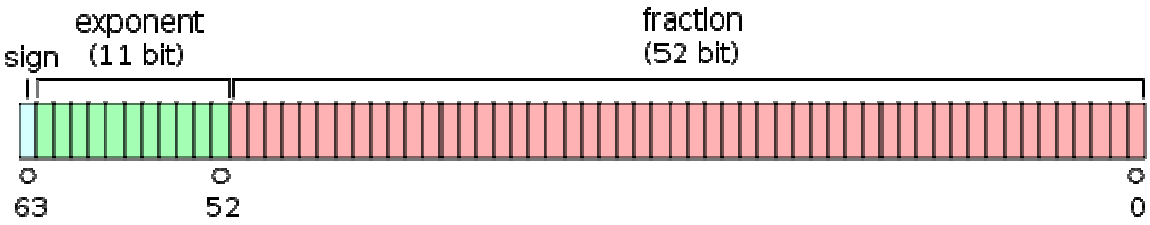
Pokud jsem tedy hledal sekvenci „11809235“ nalezl jsem ji pod sekvencí „35928011“. Pro příjemnější práci používám v programu hexadecimální soustavu. Tímto zádrhelem jsem vyřešil načítání celých čísel. Problém opět nastává u desetinných čísel. Jelikož desetinné číslo může v programovacím jazyce Delfi být typu Double nebo Float. Podle toho se taky do souboru uložila, ale při vyčítání souboru v programovacím jazyce PHP žádný typ double není, je pouze float. Proto jsem si musel vytvořit vlastní metodu, která vrátí hodnotu typu float, ze zadaného hexadecimálního tvaru.

Popis vytvoření desetinné hodnoty

Pro zjištění desetinné hodnoty z binárních dat, jsem se pro lepší práci rozhodl pracovat v hexadecimální podobě. Desetinné číslo má 8 bajtů, což znamená 16 hexadecimálních čísel, které toto číslo reprezentují. Při čtení je nutné nejprve popřehazovat jednotlivá čísla do správného formátu, hexadecimální čísla se musí otočit po dvojicích. Takto upravenou hodnotu můžeme dále zpracovat následujícím postupem.

Nejprve vezmu první hexadecimální číslo a zjistím jeho nejvyšší bit. Pokud je nastaven na jedničku, jedná se o záporné číslo, pokud ne, vše je beze změn. Následně vezmu první 3 hexadecimální čísla a převedu je na desítkové číslo, od kterého odečtu

číslo 1023. Toto číslo určuje, kolik bitů z následujících tvoří celou část desetinného čísla. Následně těchto prvních 12 bitů nahradíme jedničkou a přidáme k nim zbytek celého čísla. Tuto binární hodnotu převedeme na desetinné číslo a máme část finálního čísla.



Obrázek 9 - Bitová reprezentace typu Double [7]

Zbytek bitů je část desetinná, která je vypočítána stejným převodem z binární soustavy na desítkovou, pouze s uplatněním převrácené hodnoty násobku. Neboli jednotlivé bity reprezentují 1/2, 1/4, 1/8, … Spojením desetinné a celé složky získáme požadované desetinné číslo. Před ukončením je ještě nutné vynásobit číslo hodnotou -1 pokud první bit byl nastaven na jedničku.

Převod datumového formátu

Dalším problémem vzniklým použitím dvou různých programovacích jazyků byl čas. Tedy spíše datum s časem. Jazyk PHP používá k zaznamenání daného data a času počet vteřin od data 1.1.1970 00:00:00. Delfi zase používá desetinné číslo typu Double, kde celá část představuje počet dnů od roku 30. 12. 1989. Desetinná část představuje čas, uložený jako zlomek dne. Pro tento převod existuje vzorec:

timestampUNIX = (timestampDelfi - 25569) * 86400

Poslední překážkou k dokončení vyčítání binárních souborů bylo načtení všech dostupných zařízení, včetně jejich parametrů. Webová kniha jízd byla zpočátku napsána pro dva možné rozšiřující moduly. V průběhu času se ukázalo, že je potřeba používat více modulů, a proto jich současná verze aplikace rozpozná pět.

Jelikož webová aplikace je víceuživatelský systém, bylo nutné zabezpečit, aby více uživatelů nemohlo spustit generování dat najednou a tím vytvořit duplicitní záznamy. Tento problém jsem vyřešil tak, že se pro každého uživatele, který chce načítat data, vytvoří složka. Přesunou se do ní soubory pro vozidla, na která má daný uživatel oprávnění. Tím je dána jedinečnost, jelikož nelze přesunout soubor vícekrát,

soubor se bude nacházet jen u jednoho uživatele, kde se následně zpracuje. Zpracovaná data vidí již všichni uživatelé.

Vytvoření datumu

Tento problém mi dosti zamotal hlavu, protože jsem webovou knihu jízd začal programovat v listopadu roku 2008. Tehdy jsem napsal metody pro vyčítání dat ze souborů společně s generováním dalších záznamů, jako jsou jízdy, denní a měsíční záznamy. Problém nastal při přechodu na letní čas. Kdy se všechny generované záznamy o hodinu rozcházely. Problém jsem hledal v generování. Později se však ukázalo, že stejné datum dává Framework i funkce „date()" v PHP. Oba postupy při vytváření textové podoby datumu porovnávají podle časového pásma a přihlíží i k posunu na letní respektive zimní čas. Proto všude tam, kde je potřeba převést hodnotu datumu na textový řetězec je nutné nastavit časovou zónu na GMT pomocí příkazu date_default_timezone_set('GMT'); Poté k hodnotě datumu nebude připočítán žádný rozdíl, ovlivněn pásmem a lokálním časem.

Druhý problém se vyskytl až v pokročilé fázi vývoje a týkal se chyby ve frameworku Zend. Tato chyba způsobuje, že k datumu mezi 18. 12. 2008 až 31. 12. 2008 framework připočítá jeden rok, čímž pro datum 22. 12. 2008 získáme datum 22. 12. 2009. V aktuální verzi frameworku je tato chyba již opravena. Pro současnou verzi jsem problém vyřešil kontrolou datumu vygenerovaného frameworkem se statickou metodou „Zend_Date::YEAR", která dává správné datum. Hodnota roku v textové podobě se porovná, a pokud nesouhlasí, zamění se za rok, který vrátila statická metoda třídy.

4.2.2 Archivace souborů

Po zpracování kompletního souboru se zdrojovými daty je soubor přesunut do složky „Processed", kde je soubor připraven pro archivaci, kdyby v budoucnu bylo nutné soubor znovu vyčíst. Znovunačtení se použije pravděpodobně tam, kde je velké množství vozidel a omezená databáze pro uchování všech dat a proto se data z databáze postupně odmazávají. Budeme-li se chtít vrátit k nějakému historickému datu, v databázi data již nebudou a z toho důvodu se i zdrojové soubory archivují. Výhodou proti archivaci databáze je to, že ze jména souboru se dá usoudit, jaká data soubor

obsahuje. Není tedy nutné obnovovat několik záloh databáze, než naleznu tu, kde jsou požadované údaje.

4.3 Vytváření jízd

Vytváření jízd je podstatnou záležitostí celého systému. Jelikož kniha jízd musí pracovat ve dvou režimech, jsou její podstatné části zdvojeny. Každý režim použití disponuje jedním zpracováním informací, jelikož se zdrojová data od sebe liší natolik, že je nelze seskupit. Druhý důvod je také v umístění dat. Pro možnost lokální instalace jsou veškerá data uložena v databázi systému. Při použití možnosti instalace na firemním serveru jsou data, generovaná knihou jízd, uložena v databázi knihy jízd, ale zdrojová data jsou vyčtena z existující databáze systému RACAR, který disponuje potřebnými informacemi pro správné fungování knihy jízd.

4.3.1 Vytváření jízd z dat systému RACAR

Nejprve je nutné zjistit, kterými vozidly uživatel disponuje. Pro tento účel mám vytvořenou metodu, která vrátí textovou podobu identifikačních údajů vozidel oddělenými čárkou, aby se tento tvar dal použít bez jakýchkoli úprav do SQL dotazů do databáze. Pro tyto vozidla si zjistím veškeré informace, jak základní ze zdrojového systému, tak rozšiřující pro knihu jízd. Nyní už mohu procházet vozidla a pro každé vytvořit záznamy jízd. Pro každé vozidlo si zjistím poslední ukončovací záznam o jízdě. Tento údaj je podstatný pro rychlé získání dat. Pro každé vozidlo je zaznamenáno datum posledního zpracovaného záznamu. Pokud vyčtené datum je vyšší jak poslední zpracované, jsou v databázi nová data a může se pokračovat ve zpracovávání. Jinak pokračuji na další vozidlo.

V případě vyhodnocení datumů se zjistí, že jsou data ke zpracování, vyčtou se veškerá data pro knihu jízd od data posledního zpracování. Data jsou seřazena již z databáze a je tedy možné ihned přistoupit ke zpracovávání. Zpracování dat spočívá v rozdělení záznamů na jednotlivé jízdy. K tomuto úkolu nám pomáhají typy záznamů. Každý záznam obsahuje typ, zda se jedná o počátek, konec nebo průběžná data. Jednoduše procházím data a při označení záznamu začátkem cesty začínám data kopírovat do pomocné proměnné. Prvkem ukončení jízdy kopírování ukončím a tento balík dat pošlu k dalšímu zpracování. Touto problematikou se budu zabývat později.

Zde zbývá ještě dodat, že takto algoritmus pracuje s ideálními daty, ale reálný provoz mi ukázal, jak daleko má ideální stav k tomu reálnému. Prvním překvapením byla absence některých záznamů. V případě, že chybějící záznam je z průběhu jízdy, problém není, pouze vypočtená vzdálenost nedosahuje takové přesnosti. V momentě, kdy je chybějící záznam z počátku nebo konce jízdy, problém vzniká, protože jízda buďto nezačne, nebo neskončí. Z tohoto důvodu jsou do algoritmu doplněny mechanismy, které si s touto absencí poradí.

Nedílnou součástí problémů s tímto mechanismem byla i duplicita záznamů. Pokud do algoritmu vstoupila data oznamující počátek cesty, začala se data kopírovat. Příchod dalšího počátku cesty nebyl korektní. Algoritmus na to reagoval tak, že předchozí jízdu ukončil a začal novou. Postup ale nebyl vyhovující. Proto jsem musel přijít s další změnou algoritmu tentokrát na detekci duplicitních záznamů a jejich odstranění ze vstupních dat. Tento problém by se dal řešit na úrovni databáze, kdy jsem mohl specifikovat dotaz do databáze, že nechci shodná data. Toto řešení má problém v tom, že čas na zpracování takto tvořeného dotazu je několikanásobně delší. Z tohoto důvodu jsem se tento stav rozhodl řešit přímo v kódu. Tento stav byl vyhovující několik týdnů.

Změna nastala v situaci, když došlo k duplikování záznamů, které vznikly ve stejný čas. Pro příklad při odesílání pravidelného hlášení vozidlové jednotky došlo současně k ukončení jízdy. S touto možností jsem počítal a systém ji byl schopen zpracovat. Nepočítal jsem ovšem s duplikací těchto záznamů. Po příchodu události o ukončení jízdy přijde zpráva pravidelného záznamu a další ukončovací událost. Pro tento stav byla přepracována detekce duplicitních záznamů.

Nejpodstatnější změnou prošlo získání a zpracování dat poslední změnou, která spočívá v absenci některých parametrů dříve využívaných pro správné řazení událostí za sebou, v pořadí v jakém by měla být. Tato změna způsobila chaos v údajích. Jelikož databáze měla data rovnat dle atributu, který v současnosti neexistuje. Z tohoto důvodu bylo nutné od algoritmu ještě na začátku připojit řazení, upravující data dle požadovaného formátu. Funkce, kterou jsem napsal pro řazení záznamů, nebyla dostatečně optimalizovaná a zpracování dat s 50 000 záznamy trvalo 40 vteřin. Tento čas byl nepřijatelný zejména proto, že jde o čas srovnání záznamů pro jedno vozidlo. Při představě vlastnění sta vozů by řazení trvalo přes hodinu. Proto jsem věnoval čas optimalizaci a dostal se pro tento počet záznamů na čas pod 10 vteřin. Tento čas je již

uspokojivý, jelikož se nepředpokládá generování dat s takovýmto počtem záznamu. V průměru je tento počet záznamů za 3 měsíční provoz vozidla.

Zpracování dat pro jednu jízdu

Zde popíši postup vytváření jízd. Datum počátku a ukončení jízdy udává první a poslední záznam. Pozici startu a cíle lze také zjistit z okrajových záznamů. Doba jízdy je dána rozdílem časových údajů počátku a konce jízdy. Poté procházím jednotlivé záznamy a určuji vzdálenost bodů mezi sebou z GPS souřadnic.

Generování souřadnic je prováděno přepočtem souřadnic na radiány. Z těchto hodnot je možné vypočítat šířku v radiánech. Z šířky je spočítán poloměr země. Za pomocí těchto proměnných se Pythagorovou větou zjistí vzdálenost mezi těmito body, s přihlédnutím k nadmořské výšce.

Sečtením těchto dílčích vzdáleností jsem získal trojrozměrnou vzdálenost jednotlivých bodů. Z doby a vzdálenosti je možné vypočítat průměrnou rychlost. Maximální rychlost udává maximální rychlost během jízdy. Jako speciální parametr eviduji ještě prostátý čas. Jedná se o čas, kdy se vozidlo pohybuje rychlostí menší než 3 km/h. Což je čas stání na křižovatkách, v koloně a podobných situacích. Zajímavostí je ještě výpočet spotřeby paliva, který se standardně počítá klasicky z ujeté vzdálenosti vydělením číslem 100 a vynásobením normovanou spotřebou definovanou pro vozidlo. Nebo rozdílem impulsů z palivoměru v nádrži, či průtokoměrem v palivové hadičce. Při těchto doplňkových systémech se místo normované hodnoty udává objem pohonné hmoty na jeden impulz.

Zpracování dat pro jeden den

Pro zpracování denního záznamu pro konkrétní den, je nutné vybrat z databáze jednotlivé jízdy za daný den, a pokud se jedná o jízdu protínající více dnů, použít data za daný konkrétní den. Toho je docíleno speciálním atributem pro tento účel. Pokud je jízda ve více dnech, uchovává se jízda jako celek a dále jsou do databáze uloženy jednotlivé části pro každý den. Proto vyberu z databáze jízdy pro aktuální den a ty jsou následně zpracovány.

Zpracování začíná zjištěním, zda záznam pro daný den již v databázi existuje, nebo zda se bude vytvářet. Pokud záznam neexistuje, uloží se do něj hodnoty dané

jízdy, nastaví se, zda jde o čas motohodiny, nebo o běžnou jízdu a záznam se uloží. V případě, že záznam již existuje, ke stávajícím hodnotám se přičtou hodnoty dané jízdy, s ohledem na specifické parametry jako jsou motohodiny a soukromí nebo firemní čas strávený jízdou. Na přání zákazníka, jsem ještě výpočet doplňoval o sumarizované údaje. Původně se čas rozděloval pouze na firemní a soukromé. Ale zákazník měl asi problém při práci s těmito časy a tak se aplikace upravovala, aby bylo možné vypsat sumarizované časy i vzdálenosti včetně procentuálního využití za daný den.

Později se ukázalo, že by si zákazník rád vytvářel grafy i s těmito parametry. Ale tato funkcionalita byla odložena na později, jelikož její využití by bylo velice specifické. Zato parametry týkající se využitelnosti vozidel jsem musel do systému již integrovat. Jedná se o údaj, v kolik hodin začala první jízda a kolik hodin vozidlo stálo od poslední jízdy. Dále bylo nutné počítat dobu mezi jízdami. Tuto funkcionalitu využívají například dopravci balíků. Sledováním několika vozů bylo zjištěno, že průměrný čas na doručení balíku danému zákazníkovi trvá 5 minut. Což je 5 minut pro zákazníka, pro 20 zákazníků je to čas 100 minut, které by měl řidič stát s vozidlem a komunikovat se zákazníkem. Pokud se ale řidič vrátí na stanoviště a měl například jen 30 minut času mezi jízdami a tvrdí, že se pokoušel dozvonit a domluvit se všemi zákazníky evidentně není něco v pořádku.

Jiným zákazníkem bylo zjištěno, že lidé ve služebních vozech jezdí v pracovní době volněji a klidněji než ve svém volnu. Proto jsme se rozhodli do statistik zahrnout i údaje o průměrných a maximálních rychlostech. Maximální rychlost je zrádná, protože stačí jednou za den předjíždět pomalejší vozidlo, rychlost stoupne a už se ve statistikách projeví. Z mého pohledu je zajímavější rychlost průměrná, která eliminuje podobné chvilkové odchylky od běžného stavu používání.

4.3.2 Vytváření jízd z načtených souborů

Hlavní odlišností je množství informací ukrývajících se v jednotlivých záznamech. Záznamy načítané ze souborů mají pouze informace nutné pro knihu jízd na rozdíl od informací v systému RACAR. Jelikož je správa záznamů kompletně v režii knihy jízd, načtená data jsou uložena v tabulce „positions“ a zpracovaná v tabulce „positionsprocessed“. Vzhledem k rozvržení dat a faktu, že nezpracované události jsou v jedné tabulce a zpracované v druhé, nemusím si pamatovat, jaký poslední záznam jsem zpracoval a podobné údaje. Zde mám vždy jen data, která je nutno zpracovat. Proto si mohu vyčíst pouze identifikační čísla vozidel nacházející se v této tabulce a pro každé si vyčíst jeho záznamy. Postup rozpoznání jízdy je stejný jako v předešlém případě, protože události mají stejné typy, ale zbylé údaje se liší. Toto popíši v části o zpracování jednotlivých jízd.

Zpracování dat pro jednu jízdu

Zde je algoritmus obdobný, pouze se liší atributy záznamů. Například čas je v záznamech vyčtených ze souborů ve dvou formátech, jako čas lokální a jako čas GTM. Já pro výpočet používám čas lokální. Další změnou jsou údaje dat z čtečky karet řidičů. Zatím co v systému RACAR jsou v databázi identifikační čísla řidičů z databáze řidičů, v záznamech je uloženo přímo číslo karty řidiče. Proto před jeho uložením se musí zjistit identifikační číslo daného řidiče.

Podstatnou změnou jsou údaje o vstupních a výstupních údajích vozidlové jednotky. Zatím co v systému RACAR jsou data komprimována, zde má každý vstup i výstup svůj bitový atribut. Není tedy nutné pro vyhodnocení stavů hodnoty nijak přepočítávat či upravovat. Po vytvoření jízd je poté další zpracování shodné a k odlišnostem nedochází.

4.4 Základní modely aplikace

Modely jsou v pojetí frameworku Zend brány jako třídy, které mají své atributy a vlastnosti. Výhoda spočívá ve využití třídy pro přístup k metodám a při změně jej stačí opravit na jednom místě. Třídy jsem pojmenovával dle jejich využití. Pokud třída zastupuje databázovou tabulku, je potomkem třídy „Zend_Db_Table“. Pokud se v budoucnu změní název tabulky. Postačí mi opravit název tabulky a atributu této třídy a aplikace bude dále fungovat.

Ukázka databázové třídy:

```
class Car extends Zend_Db_Table
{
  protected $_name = 'car';
  //Nastavení jména tabulky v databázi.
  public function GetIDs()
  {
    $tmp = $this->select();
    $adapter = $tmp->getTable()->getAdapter();
    //Nastavení instance, která umí zpracovat konkrétní sql dotaz.
    $sql = 'SELECT car_id FROM car';
    //sql dotaz
    $result = $adapter->fetchAll($sql);
    //Provedení dotazu databází.
    if ($result != null) return $result; else return null;
    //Pokud je výsledek z databáze vrátím jej, jinak vracím null
  }
}
```

Shodným stylem jsou psány ostatní třídy. Pokud se nejedná o databázovou třídu, ale o mnou vytvořené třídy, nemají žádného rodiče. Jedná se tedy o třídy typově nezávislé, obsahující až na výjimky pouze metody. Tyto třídy následně popíši jednu po druhé.

4.4.1 Třída CompleteStructure

CompleteStrukture je třída zodpovídající za zjištění kompletních údajů o objektu, ke kterému máme jednoznačný identifikátor. Seznam metod:

- IdToCar($tmp_id) – Vrátí všechny atributy tabulky Car pro dané vozidlo
- IdToCar_modem_registration($tmp_id) – Vrátí pouze sloupce modem_id a registrační značku vozidla
- IdToCar_modem_registration_FromCarsId($tmp_id) – Vrátí pouze dva sloupce, tentokrát pro více vozidel
- IdToUser($tmp_id) – Vrátí kompletní záznam o uživateli
- IdToPorpuse($tmp_id) – Vrátí kompletní záznam o druhu jízdy
- IdToCompany($tmp_id) – Vrátí kompletní záznamy o společnosti
- IdToCarSecond($tmp_id) – Vrátí kompletní údaje o rozšířených vlastnostech vozidla
- IdsToCarSeconds($tmp_id) – Vrátí podstatné rozšiřující informace o vozidle, pro skupinu vozidel
- IdToDriver($tmp_id) – Vrátí kompletní záznamy o řidiči
- CardIdToDriver($tmp_id) – Vrátí kompletní záznamy o řidiči po zadání jeho kódu z identifikační karty
- ModemIdToCar($tmp_modemID) – Vrátí kompletní záznamy vozidla z jeho identifikačního čísla.

Použití těchto metod je nutné všude tam, kde znám jeden identifikátor, většinou číslo uživatele, číslo vozidla a další. Tyto údaje mi však nestačí a potřebuji například zjistit celé jméno uživatele, nebo SPZ vozidla, ke kterému mám jen jeho číslo.

4.4.2 Třída GenerateHTML

Je třída určená ke generování HTML kódu do stránek. Je použita především ke generování záznamů, jako jsou výpisy záznamů jízd, denních a měsíčních výkazů a podobně. Metoda dostane v parametru data a potřebné informace, jako jsou zvýrazněná období a podobně a vrátí HTML kód, který je ihned umístěn do stránky.

- GetHTMLCodePositions(…) – Vrátí HTML kód pro zobrazení požadovaných informací. Jako parametr vstupují všechna data, pole určující, které sloupky se mají zobrazit při jakém oprávnění uživatele, zvýraznění času, zvýraznění vozidla, mód knihy jízd, identifikátor řádku, který má být vybrán.
- GetHTMLCodeRoads(…) – Vrátí HTML kód jízdy.
- GetHTMLCodeDailyRecord(…) – Vrátí HTML kód denního výkazu
- GetHTMLCodeMonthlyRecord(…) – Vrátí HTML kód měsíčního výkazu
- GetNameMonthFromMonthNumber($num = ‘00’, $big = true) Vrátí řetězec pro daný měsíc z lokalizačního souboru. Druhý parametr rozhoduje, zda počáteční písmeno je velké nebo malé.
- GetURLfromRacar(…) – Vrátí řetězec pro přístup k systému RACAR a načtení požadovaného archivního záznamu.

4.4.3 Třída InitHeadTable

HeadTable je proměnná typu pole – pole. Pole v PHP je bráno jako klíč a jeho atribut. V tomto případě je jako klíč jedinečný identifikátor každého sloupce tabulky pro lokalizační soubor. Jako atribut je použito pole, které má dvě položky. První položkou je autorita, která udává, při jakém uživatelském oprávnění se zobrazí. Druhou položkou je zobrazení, která určuje, jestli bude tento sloupec tabulky v základním nastavení vidět nebo nikoli. Takto vytvořené pole se uloží do pole hodnot „Session“, aby byly dostupné během celé práce v systému. Dále je vytvořeno klasické pole, kde jako atributy jsou uloženy klíče první tabulky, pro rychlejší vytváření tabulek. Seznam metod:

- InitHeadTablePositions()
- InitHeadTableRoads()
- InitHeadTableDailyRecords()
- InitHeadTableMonthlyRecords()
- InitHeadTableCarSeconds()

4.4.4 Třída InitSettingsFilters

Třída „InitSettingsFilters“ má na starosti inicializaci filtrů pro jednotlivé výpisy. Jde o identifikaci vozidla, datumu a času, typu jízdy a popřípadě omezení počtu zobrazovaných záznamů. Všechny tyto atributy jsou atributy třídy SearchParameters, která reprezentuje nastavení jednotlivých filtrů. Filtry pro každé zobrazení jsou dva. První je klasicky zobrazený, který se dá modifikovat ručně a druhý je daný systémem, při přechodu mezi jednotlivými typy záznamů.

- InitPositionRecords() – Nastaví třídu na předem definované hodnoty a rozmezí datumu je 7 dní. Což znamená, že v základním zobrazení se zobrazí pozice za posledních 7 dní.
- InitRoadRecords() – Nastaví časové rozmezí na posledních 7 dní.
- InitDailyRecords() – Nastaví časové rozmezí od prvního dne v měsíci do aktuálního, pokud se jedná o prvního, zobrazí se poslední měsíc.
- InitMonthlyRecords() – Nastaví časové omezení tak, aby byly na první stránce zobrazeny nejaktuálnější záznamy, pokud se tam vejdou. Pokud bude mít uživatel jedno vozidlo, zobrazí se posledních 10 dní. Pokud bude mít 12 vozidel, zobrazí se pouze nejaktuálnější měsíc pro všechna vozidla. Algoritmus je založen na zjištění počtu měsíčních záznamů. Nejprve posunuji datum nazpět, dokud nenaleznu alespoň jeden záznam. Poté posunuji datum o měsíc zpět a zjišťuji, zda je počet událostí menší nebo je roven 10. Pokud ano, upravím aktuální datum a postupuji dále, dokud nenajdu více záznamů, nebo dokud nedosáhnu konce cyklu, který byl zvolen na 1. 1. 2000.

4.4.5 Třída MyAuth

Jedná se o jedinou třídu odvozenou od frameworku Zend, kterou jsem musel přepsat tak, aby bylo možné použít stávající systém RACAR. Jelikož v současném systému jsou uživatelská jména i hesla uložena v databázi nekódovaně. Po odeslání přihlašovacího formuláře, se heslo zašifruje a odchází na server zaheslované, tak i heslo z databáze se musí zakódovat a porovnat takto zakódované údaje. Pokud bych kódoval heslo přímo, v PHP kódu by nebyl žádný problém, ale já jsem chtěl využít komponenty Zend_Auth a proto jsem musel poupravit kontrolu hesla v této komponentě. Nalezl jsem metodu, ve které komponenta komunikuje s databází. Funkci jsem poupravil tak, aby hesla načítaná z databáze byla kódována dle správného šifrovacího algoritmu.

```
class MyAuth extends Zend_Auth_Adapter_DbTable
{
  protected function _authenticateCreateSelect(){…}
}
```

4.4.6 Třída ProcessingRecord

Tato třída je napsána pro čtení z binárního souboru a následného zpracování takto získaných dat.

- MFGetDecFromHex($hex, $postion, $length) – Vrátí číslo v desítkové soustavě, pro hexadecimální čísla o délce 2, 4, 8, 16 znaků.
- MFGetRevertHexFromHex($hex, $postion, $length) – Vrátí hexadecimální tvar s přeházenými bajty do správného pořadí pro další konverzi. Funkční pro všechny sudé délky hexadecimálního řetězce.
- MFRevertHexDouble($tmp) – Vrátí otočený hexadecimální tvar speciálně určený pro double hodnotu.

- MFGetTimestampFromHex($hex, $postion, $length) – Vrátí unixový timestamp z hexadecimálního tvaru. Hexadecimální tvar je dán hexadecimálními znaky, počáteční pozicí a délkou čteného řetězce. Tento řetězec je následně upraven metodami MFRevertHexDouble a MFGetDoubleFromHex čímž získáme desetinné vyjádření datumu a času dle jazyka Delfi. Delfi má nastaveno timestamp 0 na 31.12.1899 00:00:00. Oproti PHP které má timestamp 0 na 1.1.1970 00:00:00. Z tohoto důvodu je nutný přepočet, který od libovolného datumu odečte rozdíl dní jednotlivých formátů, což v našem případě je 25569. Dalším krokem jsou jednotky, jelikož v PHP celá čísla určují vteřiny a v delfi dny, musí se výsledná hodnota ještě vynásobit číslem 86400. Číslo je počet sekund za jeden den, neboli 3600*24.
- MFDoubleFromHex($hex) – Vrátí desetinné číslo z hexadecimálního tvaru na vstupu. Postup získání je následující. Nejprve zjistím, zda je první bit nastaven. Pokud ano jedná se o záporné číslo. Dále vezmu první 3 bajty a odečtu od nich konstantu 1023, výsledné číslo určí exponent. Binární číslo vznikne tak, že k číslu jedna přidáváme jednotlivé hexadecimální znaky převedené do binární podoby vždy o délce 4 znaků. Z tohoto binárního formátu vezmeme prvních x čísel. Číslo x vypočítáme jako exponent+1. Tím jsme získali celou část desetinného čísla. Zbylé byty procházíme a přičítáme jednotlivé desetinné složky, pokud je bit aktivní (1/2, ¼, 1/8, …). Nakonec sečteme jak desetinnou tak celou část a máme výsledné desetinné číslo. Pokud byl první byt aktivní, musíme nakonec vynásobit výsledné číslo hodnotou -1.
- ReadFromFile($file) – Vrátí binární hodnotu, zda byl soubor zpracován korektně nebo nikoli. Metoda vyčte celý soubor v hexadecimální podobě. Odstraním hlavičku souboru a pokračuji vyčítáním datových bloků, dokud nejsou všechny vyčteny. Pro maximální modularitu jsem zvolil následující postup. Pro vyčítání jednotlivých položek nejprve zvolím délku, která určuje, kolik bajtů proměnná obsahuje, poté hodnotu vyčtu a posunu index neboli počátek čtení o vyčítanou délku. Po vyčtení datového bloku zkrátím data o vyčtenou délku.

4.4.7 Třída ProcessingRecordNext

ProcessingRecordNext navazuje na předchozí třídu, která umí načíst pouze jeden zadaný soubor. Tato třída je nadřazená a generuje záznamy ze souborů, jízdy a jednotlivé výkazy. Třídu bych pro vysvětlení rozdělil ještě do bloků, podle toho jaké záznamy se generují.

Načtení souborů

Pro načítání souborů, je v tomto bloku pouze jedna metoda. Jako vstup je zadána cesta, kde se nachází zdrojové soubory. Poté metoda ReadFromFilesystem vrací proměnnou typu bool, zda byla data načtena správně.

Metoda zjistí všechny soubory v definované složce a seřadí je podle datumu od nejstaršího po nejnovější. Dále si každá instance přesune soubory do své složky, aby nemohlo docházet k několikanásobnému zpracování údajů. Po přesunutí do pracovního adresáře Temp jsou soubory po jednom vybírány a zadávány metodě ReadFromFile ke zpracování. Metoda ReadFromFile je součástí třídy ProcessingRecord.

Generování jízd

Generování jízd bere za zdroj dat vyčtené záznamy ze souborů nebo data uložená v databázi systému RACAR. Zdroje dat se liší jak ve struktuře, tak v hodnotách definujících jednotlivé stavy. Například vstupy jsou pro jeden vstup jako bit, pro druhou variantu jako bajt, kde jednotlivé bity představují vstupy v bitové masce. Generování má na starost funkce Generate, která se stará o zpracování jízd. Funkcionalita metody spočívá v rozdělení záznamů do jednotlivých bloků dat, kde každý blok reprezentuje jednu jízdu. Metoda Generate poté předá tento blok dat jiné metodě, která blok zpracuje a vytvoří jízdu. Metoda Generate může využívat metody:

- Processed – Zpracování bloku dat, pokud se jedná o lokální instalaci.
- ProcessedRacar – pracování bloku dat, pokud se jedná o instalaci na firemním serveru a data se čerpají z RACARu.
- Move – Přesunutí dat pro lokální instalaci z tabulky positions do positionsprocessed.

Metody pro zpracování souboru dat ke své funkci využívají metodu GetDistanceFromWGS. Tato metoda vrací vzdálenost dvou bodů v metrech zadanými zeměpisnou šířkou a délkou v milisekundách a nadmořské výšky zadané v centimetrech. Přepočet je postaven na přepočtu na bod v kartézské soustavě v metrech pro osy x,y,z. Přepočet je postaven na přepočtu WGS souřadnic do úhlů vůči zemským osám v radiánech. Následně je vypočítán poloměr zeměkoule pro dané místo a z něj už Pythagorovou větou vypočtena vzdálenost dvou bodů od sebe pro 3D obrazce, aby byl zachován vliv nadmořské výšky. Více o výpočtu nelze popsat vzhledem k know-how firmy RADOM s.r.o.

Generování denních výkazů

Denní výkazy jsou generovány funkcí CreateDailyRecords, jenž zpracuje vytvořené jízdy, které ještě nebyly zpracovány, a započítá hodnoty jednotlivých jízd do odpovídajících kolonek pro denní výkazy. Postup práce metody je následující.

Na počátku se připojím k databázi a vytáhnu z ní všechny čísla vozů, která se budou zpracovávat. Ta pak procházím a ze záznamů si filtruji pouze ta, která skončila ve stejný den, co začala. Pro jednotlivou jízdu pak procházím následujícími kroky. Zjistím, zda záznam pro toto vozidlo v daný den již existuje, pokud ano hodnoty jízdy se pouze připočítají. Pokud záznam neexistuje, bude vytvořen s hodnotami jízdy. Nejpodstatnější částí je určení do jakých kolonek hodnoty zapisovat. Kam se hodnota uloží, určuje druh jízdy. Nejpodstatnější položkou je druh jízdy, určující zda jsou kilometry a časy započítávány do správných položek. K tomuto rozdělení je ještě nutné zdůraznit motohodiny, prostátý čas a nevyužitý čas. Motohodiny jsou započítávány, v době kdy má jednotka zásah, nebo během nějaké opravy či práce. Jako příklad bych uvedl hasiče, kteří během jízdy k zásahu mají klasický stav a během hašení požáru, kdy motor vozidla musí běžet a napájet další zařízení je tento čas brán jako motohodiny. Prostátý čas, je čas, během kterého se vozidlo pohybuje rychlostí menší než 3 km/h. Tento údaj je podstatný z pohledu stání v kolonách, na křižovatkách apod. Posledním důležitým údajem je nevyužitý čas. Jedná se o čas mezi jízdami za jednotlivý den, kdy vozidlo nebylo využíváno. Z tohoto parametru se dá dále získat využitelnost jednotlivých vozů. Pro zachování vazeb, je po zpracování údajů do zdrojové jízdy uložen index denního záznamu, pro lepší přechod mezi sekcemi.

Generování měsíčních výkazů

Generování měsíčních záznamů je obdobné jako tomu je u denních záznamů, s tím rozdílem, že u měsíčních záznamů jsem musel pro jednotnost zvolit den, ve kterém bude datum ukládán. K velkému překvapení jsem si vybral den první. Takže například únor 2009 bude v databázi uložen jako 2009-02-01. Tato metoda jako již předchozí řídící metody má za úkol data rozdělit do skupin a takto vytvořenou skupinu nechat zpracovat jinou metodou. Skupina se v tomto případě určuje zjištěním měsíce, ve kterém je první záznam. Další záznamy se testují, zda mají shodný měsíc, pokud ano data se překopírují, pokud nikoli data se pošlou ke zpracování metodě ProcessedDays, která vytvoří požadovaný záznam a data se začnou ukládat jako nový blok.

Metoda ProcessedDays ze zadaných dat zjistí počátek a konec období, ve kterém se mají záznamy hledat společně s číslem vozidla. Poté je odeslán do databáze odkaz, který vrátí požadované údaje v sumarizované podobě. Následují podmínky, pro generování dalších záznamů. Například průměrná hodnota se vypočítá pouze v případě, že součet doby jízdy pro firemní i soukromé účely je větší jak nula. Dále je zde kontrola, aby průměrná rychlost nebyla větší jak maximální.

4.4.8 Třída SearchParameters

SearchParameters je třída, která reprezentuje nastavení filtrů v jednotlivých sekcích. Ke všem atributům třídy jsou metody get a set. Ve filtru se uchovávají tyto informace:

- from_date – určující od kterého data budou záznamy hledány
- from_time – určující od kolika hodin budou záznamy hledány
- to_date – určují do kolikátého data budou záznamy hledány
- to_time – určují do kolika hodin budou záznamy hledány
- priv – zda mají být hledány soukromé jízdy
- serv – zda mají být hledány firemní jízdy
- car_id – určuje vozidlo, pro které se mají vozidla hledat, při variantě -1 bude systém hledat ve všech vozech
- count – určuje maximální počet záznamů, které jsou požadovány
- offset – určuje posunutí v záznamech, v současné době všude nastaven na 0

4.4.9 Třída ViewFunctions

Tato třída má dva atributy translate a session. Atributy se inicializují v konstruktoru třídy a jsou využívány v jejích metodách. Metoda YesNo10 pracuje s čísly 0 a 1. Pokud je na vstupu hodnota jedna, vrací zaškrtnutý čtvereček. Pokud vstupní hodnota je nula vrátí prázdný znak. Tato metoda je hojně využívána v tabulkovém výpisu, kde je nutno uživateli zobrazit například oprávnění k některému z účtů. Metoda ViewMessages má za úkol informovat uživatele o zprávě z probíhajícího procesu. Zprávy mají informační charakter, jako jsou informace o úspěšné změně apod. Až po informaci, že není možné zobrazit vybraný záznam, protože nebyl vygenerován a je nutné nejdříve vyplnit inicializační záznam a teprve následně vygenerovat zbylé údaje. Pro tento účel se vyskytl požadavek hodnoty zvýraznit, takže nejspíše v budoucnu bude metoda rozšířena o atribut, který určí, zda má být uživateli více zvýrazněn, nebo zda má být zobrazen jen jako informace informativního charakteru.

4.4.10 Lokalizace

Lokalizace aplikace knihy jízd je uložena v textovém souboru nacházejícího se ve složce aplication/languages. Zde musí být minimálně jedna lokalizace a případně další. V kódu je poté cesta k licenčnímu souboru vytvářena dynamicky, jako spojení cesty a názvu souboru pro konkrétní jazyk, do kterého je Kniha jízd přeložena. Dynamičnost je zajištěna proměnnou v session, kterou lze kdykoli při běhu aplikace změnit a všechny následující texty budou přeloženy do aktuálního jazyka. Lokalizační soubor je ve formátu *.ini a texty v nm jsou uloženy jako:

Jedinečný_identifikátor = „Překlad“

4.4.11 JavaSkriptové funkce

Javaskriptové funkce jsou jako vše ostatní rozčleněny do souborů, podle jejich činnosti. Tyto soubory jsou uloženy ve složce aplications/js. Výčet souborů:

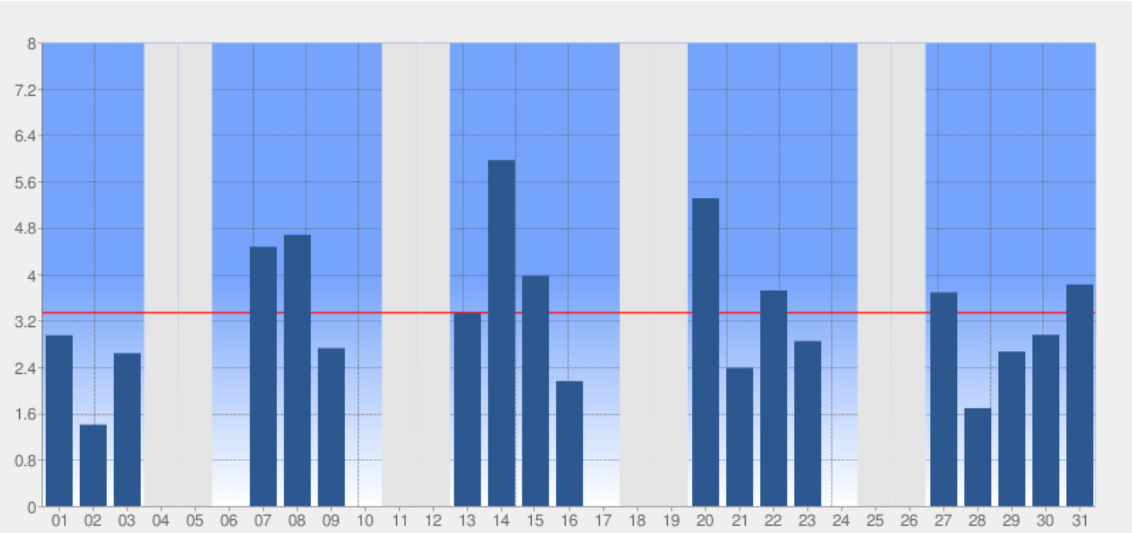
- calendar – vše co se týká funkcionalit kalendáříku
- carseconds – kontrola formuláře pro práci s rozšířenými vlastnostmi vozidel
- daily – kontrola formuláře pro práci s denními výkazy
- fuction – metody celého systému, jako aktualizace času, dotazování před smazáním, aktivace a deaktivace funkčních tlačítek, apod.
- monthly – kontrola formuláře pro práci s měsíčními záznamy
- road – kotrola formuláře pro práci s jízdami
- script_table – funkce pro fungování tabulky s výpisy – řazení, zvýrazňování, …
- user – kontrola formuláře pro práci s uživateli
- usersettings – funkce pro uživatelské nastavení formuláře

4.4.12 Vykreslení grafů

K vykreslování grafů je využívána služba serveru www.google.com, který po zadání přesně formulovaného dotazu vrátí obrázek grafu, dle specifik. Aby bylo možné vykreslovat více grafů, rozhodl jsem se napsat univerzální postup pro vytvoření požadovaného dotazu nezávislém na konkrétní proměnné. Algoritmus pracuje pouze s předanými hodnotami, které zpracuje a matematicky analyzuje. Vstupem jsou data, která mají strukturu pole – pole. První pole je pole prvků a každý prvek se skládá ze dvou složek, datumu a hodnoty. Na počátku rozdělím data na jednotlivá pole. Poté stanovím maximum jako maximální hodnota z pole plus 2% aby hodnoty nebyly až k okraji grafu. Maximum je dále upraveno na násobek blízkého velkého čísla. Dále vytvořím popisek pro osu y. Jedná se o nulu a deset popisků, kde maximum rozdělím na 10 dílů. Dále nastavím okraje okolo grafu tak, aby zde bylo místo a graf nebyl až do okraje. A spočítám šířku sloupku jako celé číslo z výpočtu. Od okrajového rozměru grafu, jenž je 800 px odečtu okraje vpravo a vlevo. Tuto hodnotu vynásobím procentem z poměru mezi grafem a popisky a nakonec hodnotu vydělím počtem hodnot na ose x. Nezbývá než procházet jednotlivé dny v měsíci a kontrolovat zda pro daný den jsou nějaké záznamy, pokud ano hodnota se zaokrouhlí na 3 desetinná místa a zpracuje se, pokud nikoli nastaví se příslušná hodnota na -1, což je značka pro prázdnou hodnotu. Dále je potřeba zjistit, zda daný den je v oblasti víkendu. Pokud ano, zvýrazní se šedým sloupcem po celé délce grafu. Poslední specifickou hodnotou je průměr, který je získán v úvodní tabulce se všemi průměrnými hodnotami. Pro vybraný graf se tato průměrná hodnota uloží a následně vloží do grafu. Nyní stačí už jen vygenerovat http adresu ve formátu:

```
$addr = 'http://chart.apis.google.com/chart?';
$addr = $addr.'chs=800x375&';//rozmery grafu
$addr = $addr.'chds=0,'.$max.'&';//rozmezi vstupnich hodnot
$addr = $addr.'chd=t:'.$values;//hodnoty
```

Takto vytvořený dotaz automaticky vrátí požadovaná data do položky <img>. Příkaz pro zobrazení grafu vypadá takto: <img src=”<?php echo $addr ?>” alt=”Graf”/>



Obrázek 10 - Graf měsíčního záznamu

4.4.13 Export do csv

Export do *.csv souboru je dán aktuálním výběrem. Jinými slovy to co mám zobrazeno na obrazovce, lze exportovat do souboru. Při stisku tlačítka exportovat se zobrazí nabídka, které údaje chce uživatel zpracovat a v jakém formátu se rozhodl soubor uložit. Formáty jsou zpracovány dva. První je dle definice normy RFC 4180 a druhý kvůli správnému zobrazení v Microsoft Excelu. Po vybrání požadovaných údajů lze stisknout tlačítko Připravit data, které data vytáhne z databáze a uloží do *.csv souboru na serveru. Při stisku tlačítka Zobrazit se uživateli objeví okno, jak chce se souborem naložit, zda ho uložit, nebo přímo otevřít.

Soubor je tvořen pro jednotnost shodně jako tabulka s výpisem. Dle struktury headTable jsou dány sloupce v exportovaném souboru. Pro každý záznam je vygenerovaný totožný HTML kód jako při výpisu, který je následně upraven do předem zvoleného formátu. Tagy HTML kódu jsou nahrazeny uvozovkami nebo přímo ignorovány. Následuje konverze desetinného formátu. Ten funguje tak, že hledá počet teček mezi uvozovkami. Pokud se jedná o jednu, jde o číslo a musí se nahradit čárkou. Pokud jsou dvě, jedná se o datum a tečky musí zůstat. Poté už stačí konvertovat záznam z UTF-8 na Windows-1250 při ukládání pro Excel a data uložit. Takto uložený soubor lze otevřít klasickým doujklikem a není nutná žádná konverze z textového souboru.

4.5 Návrh vrstvy zobrazení

4.5.1 Základní rozvržení stránek

Základní skladba stránek je tvořena horní bílou lištou, ve které je název aplikace, logo, přihlášený uživatel, datum a čas, lokalizace a možnost přihlášení nebo odhlášení. Následuje menu zobrazené jako lišta s tmavým podkladem. Na ní jsou umístěny prvky jednotlivých sekcí knihy jízd dle uživatelského oprávnění. Běžný uživatel má zpřístupněny dva řádky v menu. První dělí sekce, jak již bylo řečeno a druhý je určen k zobrazování jednotlivých vybraných údajů. Lze tu procházet mezi jednotlivými záznamy a jsou brány jako další sekce a to sekce záznamů. Tento obsah stránky je pořád stejný a mění se zbytek stránky. Zde je většinou filtr pro zpřesnění hledaných záznamů.

Ve filtru je možné vybrat všechna nebo jedno specifické vozidlo, dále období, ve kterém chce uživatel záznamy hledat. Při zobrazení jízd je zde ještě rozdělení jízd na soukromé a služební a všude je jako doplňkový parametr omezení počtu zobrazovaných hodnot.

Pod filtrem jsou umístěna funkční tlačítka, která se vztahují k zobrazené tabulce, nebo ke konkrétnímu záznamu. Pokud jsou tlačítka aktivní při zobrazení stránky bez vybrání konkrétního záznamu, jedná se o funkce pro celou tabulku. Jedná se o funkce nastavení tabulky a export. Pro konkrétní záznamy se poté zpřístupní tlačítka pro pohyb mezi jednotlivými výkazy, zobrazení daného záznamu na mapě systému RACAR a v měsíčních záznamech ještě zobrazení měsíčního výkazu o provozu vozidla. V jízdách jsou zde navíc položky rozšířeny o správu jízd.

V další části stránky je většinou tabulka reprezentující jednotlivé záznamy. Některé sloupky jsou omezeny oprávněním a některé lze zapnout, jen se standardně nezobrazují. Pod touto tabulkou je ovládací lišta skládající se z výběru, kolik záznamů má být na stránce, posunu po stránkách a nakonec aktuální číslo stránky ze všech možných. Jako poslední blok stránky je záhlaví, kde se nachází copyright.

4.5.2 Tabulka pro zobrazení dat a řazení údajů

Tabulka je vytvořena v tuml kódu jako klasická tabulka, ovšem pomocí javaskriptu je rozdělena do stránek a lze nad jejími hodnotami uplatnit řazení daného sloupku. Třída se jmenuje TINY.table.sorter a jedná se o free komponentu, kterou jsem nalezl na internetu. Je mnoho komponent, které bych rád použil, ale bohužel jsou placené nebo placené pro komerční účely. Do třídy se nastavují parametry, které definují název kaskádového stylu, který se má použít pro aktuálně zobrazovanou tabulku.

4.5.3 Komponenty Yahoo user interface

Pro lepší uživatelský komfort jsem se rozhodl využít komponent poskytovaných společností Yahoo pro vývojáře. Z možných modulů jsem si prozatím vybral kalendář, který ulehčí vypisování datumu.

Komponenta kalendáře

Pro vytvoření kalendáře je nutné vytvořit třídu yui-skin-sam ve které jsou vytvořeny kontejnery pro každý kalendář zvlášť, aby je bylo možné jednotlivě doladit pomocí kaskádových stylů, hlavně pozici na které se zobrazí uživateli. Pro každý kalendář je vytvořena nová třída kalendáře, na který je připojena událost, vytvořen handler a nakonec zobrazení kalendáře. Handler přijme událost, z parametrů zjistí datum jako pole hodnot, kde první je rok, následují měsíc a den. Dle jména kalendáře se zjistí, kterému elementu se má datum přiřadit a okno kalendáře se zavře. Po sestavení textového řetězce se uloží do příslušného elementu.

4.6 Návrh řízení uživatelských událostí

Ve všech controlerech se nejdříve testuje, jestli je uživatel přihlášen. Pokud není, okamžitě se aplikace přesune na úvodní obrazovku pro přihlášení. To v případě, pokud by někdo chtěl přeskočit přihlášení a rovnou zobrazit nějakou část knihy jízd. Controlery v knize jízd mají nejčastěji 4 standardní metody. Jedná se o metodu index, add, edit a delete.

4.6.1 Vytváření vozidel v knize jízd

Vytváření vozidel v knize jízd je specifické. Pokud se jedná o knihu jízd u zákazníka, lze v knize jízd vytvořit vozidlo, protože tento systém musí být nezávislý a pracovat samostatně. Proto má v tomto případě kniha jízd právo zapisovat i do tabulky vozidel původně spravovanými systémem RACAR. Tudíž při vytváření vozu se vůz vytvoří jak ve standardní tabulce knihy jízd s rozšířenými vlastnostmi a zároveň se vytvoří záznam v tabulce se základními informacemi.

V případě, že je kniha jízd nainstalovaná společně se systémem RACAR nesmí tomuto systému nikterak zasahovat do správy vozidel. Proto je pouze umožněno čtení z tabulky vozidel. Z tohoto důvodu je limitováno vytvoření vozu, které lze provést pouze a jedině v případě, že je vozidlo již zavedeno v systému RACAR. Poté se do tabulky rozšířených informací o vozidle, údaje uloží a kniha jízd s nimi začíná pracovat. Proto v případě, kdy chce uživatel přidat vozidlo, musí se nejdříve načíst všechna vozidla z RACARu a z knihy jízd a uživateli nabídnout pouze vozidla, které ještě v knize jízd nejsou zahrnuta. Na tom není na první pohled nic zvláštního, vytvořím dotaz do databáze, která mi vrátí rozdílná vozidla. Problém spočívá v tom, že databáze jsou nezávislé. V SQL dotazu se nelze odkazovat na jiný databázový server, a proto jsem nucen data zpracovat na úrovni aplikace v PHP.

4.6.2 Nastavování práv jednotlivým uživatelům

Nastavení práv je dáno stromovou hierarchií. Administrátor systému má přidělena všechna vozidla. Hierarchie se řídí pravidlem, uživatel oprávněný vytvářet další účty může těmto uživatelům přiřadit maximálně všechna vozidla, která má on sám. Vytvořím-li účet a přidělím mu 5 vozidel, uživatel tohoto účtu bude moci přidělit maximálně těchto 5 vozidel. Takto lze docílit stejného principu jako je ve vedení firmy. První účet se všemi firemními vozidly dostane ředitel společnosti. Ten vytvoří další uživatele, dle jednotlivých úseků firmy a každému přidělí vozidla využívající se v daném úseku. Tedy ředitel výrobního úseku vidí své vozy a může je dále dělit mezi další podřízené uživatele, ale nikdy se nedostane k vozům jiného úseku a vozu ředitele.

Pro aplikaci tato hierarchie představuje jistou rekurzi. V případě, že přidělím vozidlo někomu z podřízených, musím ho automaticky přiřadit všem nadřízeným. Tedy procházet všechny nadřazené uživatele a přiřadit jim vybrané vozidlo. Stejný případ

nastane, odebere-li někdo vozidlo uživateli nadřazenému, musí se projít všechny jeho podřízení a odebrat jim právo na toto vozidlo.

4.6.3 Algoritmus proměnného období

Algoritmus proměnného období spočívá v určení datumu tak, aby na první stránce se objevily nejnovější údaje. Algoritmus je použit u měsíčních výkazů, kde bylo zapotřebí optimálně rozvrhnout čas tak, aby tam uživatelé s více vozidly neměli příliš údajů, které je nezajímají a naopak, aby uživatel vlastnící jedno vozidlo, neměl jediný záznam. Tento stav by univerzálně řešil stav, kdy by každý uživatel měl své nadefinované hodnoty. Zatím jsem rozhodl, že se tomuto stavu budu snažit vyhnout, jelikož by to vedlo k nepochopení u zákazníků a rozhořčení, že kniha jízd nezobrazuje staré hodnoty a podobné problémy, jak tomu je ve stávajících aplikacích. Proto jsem zvolil výpočetní metodu, která datum určí na základě aktuálně zpracovaných údajů, které lze zobrazit.

Celý výpočet spočívá v počtu záznamu za určité období. Jako první nastavím datum na poslední měsíc. Zjistím počet záznamů. Pokud je roven nebo větší 10ti jsem u konce a mám výsledné datum. Pokud ne, zkusím se vrátit ještě o měsíc zpět a opět zjistím počet záznamů. Takto postupuji, dokud počet nepřekročí hodnotu 10 nebo dokud nenarazím na rok 2000.

4.6.4 Zobrazení trasy v mapě

Zobrazení trasy v mapě je prováděno specifickým dotazem na mapový server systému RACAR, který umožní se přihlásit jako uživatel přihlášený v knize jízd a zobrazit část archivu definovanou výběrem období z knihy jízd. Zobrazení je pak znázorněno v poskytnutých mapách systémem RACAR pro uživatelův účet. Trasa bude vypadat stejně, jako by se uživatel přihlásil do RACARu a v archivu si jednotlivé informace ručně nadefinoval.

5 Popis instalace webové knihy jízd

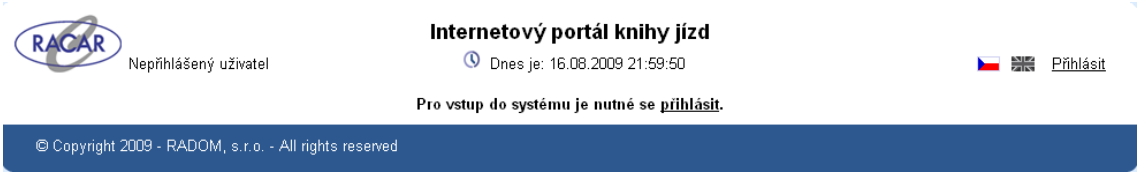
Možnosti instalace jsou myšleny tak, aby byla kniha jízd schopna pracovat jak samostatně, tak se systémem RACAR. V případě, kdy je kniha jízd instalována bez systému RACAR, musí kniha jízd převzít správu nad údaji spravovanými RACARem. V základu se jedná o správu uživatelů a vozidel. Události jsou začleněny přímo do knihy jízd, jelikož pro jejich zpracování je výhodnější odlišné zpracování od stávajícího systému. Podstatnou výhodou je oddělení zpracovaných a nezpracovaných záznamů, jenž urychlí vyhledávání záznamu při generování jednotlivých záznamů.

Proti tomu je zde symbióza se systémem RACAR, kde kniha jízd některá data přijímá z tohoto systému. Výhodou propojení je zobrazení map, jednotné účty, které již existují a nemusí se znovu vytvářet, jednotná evidence všech záznamů. S jednotností záznamů je spjata i nevýhoda, kterou jsou různé možnosti a nároky na evidenci objektů v systému. RACAR používá pro uživatele například vlastnosti, zda uživatel může vytvářet další uživatele a další omezení. Kniha jízd potřebuje podobné informace. Ty podstatné pro vozidla, jsou již zaznamenány v tabulce CarSeconds, ale rozšiřující tabulku pro uživatele není zatím v plánu vytvářet, jelikož ještě nejsou přesně definovány jednotlivé možnosti variability systému.

6 Zjednodušená uživatelská příručka

6.1 Úvodní obrazovka

Na úvodní obrazovce je pouze nejvyšší lišta, která informuje, o jakou aplikaci se jedná. Dále obsahuje logo, informace o přihlášeném uživateli, aktuální časové údaje, možnost přepínání jazyků a tlačítko pro přihlášení.



Obrázek 11 - Před přihlášením

Ve spodní části je copyright a název společnosti vlastnických práv. Pro přihlášení je nutné kliknout na tlačítko přihlásit a vyplnit přihlašovací údaje. Při špatném vyplnění je uživatel informován o chybných údajích. Při úspěšném přihlášení se uživatel dostane na základní obrazovku.

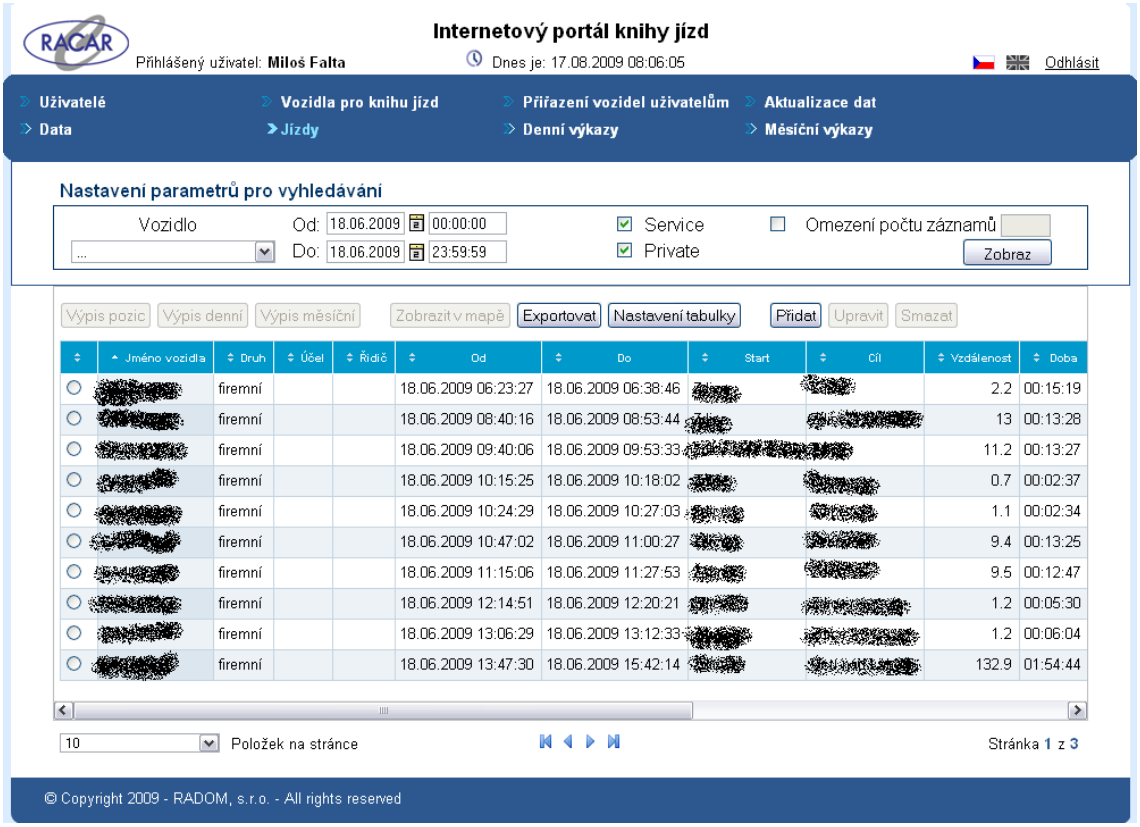
6.2 Základní obrazovka

Na základní obrazovce je již vidět menu, které se skládá ze dvou řádků, první řádek odděluje jednotlivé bloky knihy jízd a druhý řádek dělí záznamy podle jejich doby zpracování. Lze si nechat zobrazit měsíční záznamy nebo záznamy podrobnější. Tím se lze propracovat až na jednotlivé záznamy z vozidlové jednotky.

Obrázek 12 - Úvodní obrazovka

Všechny tyto součásti mají dvojí filtr záznamů. První je aktivní při kliknutí na daný odkaz přes menu a druhý se aktivuje, pokud je zvoleno příslušné tlačítko ve funkční části, vztahující se k jednomu záznamu.

Například si nastavím zobrazení jízd dne 18. 6. 2009 od 00:00 do 23:59. Následně se přesunu na měsíční výkazy a nechám si zobrazit jízdy v červenci vybraného vozidla. Zobrazí se mi pouze vybrané jízdy, díky specificky nastavenému filtru. Při kliknutí zpět na jízdy v horním menu, se vrátím k původnímu výběru.



Obrázek 13 - Denní záznamy za 18. 6. 2009

V případě stisknutí tlačítka Zobrazit, při zobrazení specifického nastavení filtru se toto nastavení uloží i do základního nastavení. Tato funkce je vhodná v případě, chceme-li z tohoto místa dále pokračovat a opět se k němu vracet.

7 Závěr

Na závěr bych chtěl shrnout vše podstatné, co jsem musel během mé práce na diplomové práci vyřešit. Nejprve jsem se několik týdnů zabýval všemi poskytnutými materiály. Převážně šlo o specifikace komunikačních protokolů a datových struktur pro uložení jednotlivých vstupních dat. Následoval seznam požadavků na knihu jízd, které se neustále prohlubovaly a měnily v závislosti na použitých technologiích a komunikacích se zákazníky. Během vstupní analýzy jsem musel nejdříve prostudovat stávající systém RACAR. Zaměřil jsem se na části týkající se přihlašování uživatelů a jejich parametrů, správy uživatelů, vozidel a jejich vzájemné propojení. Poslední společnou částí jsou ještě jednotlivé záznamy tvořené během jízdy.

Jelikož systém RACAR je v režimu online, kde je zobrazena aktuální poloha, respektive se poloha změní vždy s příchodem novější polohy v přesně nadefinovaný čas, např.: každé 2 minuty. Data pro knihu jízd jsou během jízdy ukládána častěji a jsou ukládána do paměti vozidlové jednotky. Paměť se vyčítá většinou 1x denně, v době, kdy server není zatížen a vozidlo je zrovna v pohybu. Proto Kniha jízd pracuje v režimu off-line. Nejčastěji se zpracovávají hodnoty zpětně za poslední den.

Jelikož jako zdroj požadovaných informací nemusí být databáze RACARu, musel jsem se vypořádat ještě s vyčítáním binárních souborů s daty včetně nekompatibility datových typů. Jelikož soubory jsou vytvářeny v aplikaci psané v jazyce Delfi, jež používá jiný formát datumu a času. Datový typ double v PHP vůbec není. Z důvodu nekompatibility jsem napsal vlastní metody pro získání těchto informací. Současně s vyčítáním záznamů a generováním záznamů bylo nutné zabezpečit jednotnost, aby byl soubor zpracován pouze jednou i v případě, že přijde několik požadavků na jeho zpracování. Takto musí být zabezpečené generování všech hodnot, pozicemi počínaje a záznamy o provozu vozidla konče.

Zdrojová data lze získat ze dvou zdrojů, proto části týkající se přímo dat, musí být v systému duplikovány. Zkoušel jsem napsat univerzální nadřazenou metodu, ale její rychlost zpracování byla nízká. Z tohoto důvodu jsem se rozhodl postup zpracování duplikovat. Obě části jsem následně optimalizoval pro jejich specifika. Po načtení souborů do tabulek knihy jízd je práce s těmito daty rychlejší. Zvýhodnění je způsobeno

možností přiřadit k jednotlivým záznamům vlastní identifikátory, pro lepší orientaci mezi záznamy. Této výhody jsme zbaveni při použití stávajícího systému RACAR, ovšem tato možnost poskytuje jinou výhodu. Výhodou je přímý přístup do databáze se záznamy, bez nutnosti zdlouhavě zpracovávat binární soubory generované systémem RACAR.

Další komplikací během práce na Knize jízd je změna mapového serveru. Když systém RACAR před dvěma lety začínal, používal mapové podklady serveru google.com. Ten ke konci roku 2008 změnil licenční politiku svého mapového serveru a nelze jej používat pro komerční účely. Z tohoto důvodu bylo vedením firmy rozhodnuto o vytvoření vlastního mapového serveru.

Mapový server je navržen jako několik vrstev, které se vzájemně překrývají. Jízda je vykreslena jako vrstva s mapovým podkladem, vrstvou zobrazující jednotlivé body a vrstvou zobrazující spojnice mezi těmito body. Tímto způsobem lze zobrazit libovolnou jízdu v mapě. Kniha jízd s těmito informacemi pracuje a při zobrazování jízdy v mapě odešle na server pouze informace o vozidle a období, ve kterém se mají hodnoty zobrazit. Díky těmto údajům není možné jízdu zobrazit v nějakém jiném mapovém serveru. Pokud kniha jízd nebude nainstalována společně se systémem RACAR, bude nutné pro zobrazení jízd v mapě nutné připojení k serveru RACAR na webu, který následně dokáže jízdu zobrazit.

Závažný problém mi dělala i zdrojová data. Na počátku jsem uvažoval, že data budou v databázi uložena korektně a budou se pouze shlukovat do jednotlivých jízd, ze kterých se vytvoří záznam o jízdě. Ale svůj postoj jsem musel přehodnotit, jelikož data v databázi neodpovídají realitě. Domníval jsem se, že aplikace na straně serveru data před ukládáním do databáze ověřuje. Tato domněnka byla špatná. Serverová aplikace přijme pouze data a uloží je do databáze. Z tohoto důvodu jsem musel ještě řešit problémy, jako jsou duplicitní nebo chybějící data. Chybějící data ošetřuje algoritmus zpracování tak, aby výpadek nepozměnil generovaná data. V případě duplikování souborů tomu tak nebylo.

V případě, kdy byla zahájena jízda a přišla další událost s informací o začátku jízdy, algoritmus jízdu ukončil a začal počítat novou. To nebyl stav odpovídající realitě, a proto jsem musel přistoupit k opatřením. Kde před zpracováním záznamu kontroluji, jestli se nejedná o stejný záznam, jenž byl poslední zpracovaný. Tímto způsobem lze zabezpečit ochranu proti duplicitním datům. Takto upravený algoritmus fungoval

spolehlivě asi měsíc. Poté se naskytl další problém s řazením dat. Data byla řazena primárně podle datumu a sekundárně podle čísla klíče, neboli postupně jak byla do databáze vložena.

Problém nastal v případě, kdy ve stejný čas byly vloženy všechny 3 záznamy. V tom případě došlo k náhodnému seřazení při vkládání a následném čtení a záznamy nebyly korektní. Korektní stav je takový, že jako první mají na řadu přijít data z probíhající jízdy, následované záznamem o ukončení jízdy. Jako poslední má přijít na řadu záznam o začátku jízdy. Řazení záznamů podle zmíněných pravidel nemá žádné matematické opodstatnění k číslování typů jednotlivých událostí a z tohoto důvodu bylo nutné napsat vlastní porovnávač, který seřadí hodnoty se shodnými časy v korektním pořadí, tedy jak jednotlivé záznamy vozidlová jednotka odeslala. Dále bylo nutné rozšířit opatření pro duplicitní záznamy, které se z poslední položky rozšířily na poslední 3 položky. Opatření proti duplikování záznamů má za následek zdržení algoritmu a tím celého výpočetního cyklu.

Cíl práce byl splněn mimo jiné i díky využití Zend Frameworku, který je založen na oddělení prezenční, logické a datové vrstvy, tudíž je výsledný informační systém jednoduše modifikovatelný pro pozdější možná rozšíření.

8 Použité zdroje

[1] ROZSYPAL, Petr. *Knihovna PHP : Co je to PHP?* [online]. 2008 , 14.5.2009 [cit. 2009-08-19]. Dostupný z WWW: <http://php.interval.cz/clanky/co-je-to-php>. ISSN 1212-8651.

[2] *Model-view-cotroller* [online]. 2009 [cit. 2009-04-14]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Model-view-controller>.

[3] *MySQL* [online]. 2009 [cit. 2009-08-19]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/MySQL>.

[4] *JavaScript* [online]. 2009 [cit. 2009-08-19]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/JavaScript>.

[5] *HTTPS* [online]. 2009 [cit. 2009-08-19]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Https>.

[6] *RFID* [online]. 2009 [cit. 2009-08-19]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/RFID>.

[7] *Double precision floating-point format* [online]. 2009 [cit. 2009-08-19]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Double_precision_floating-point_format>.

[8] *Coordinated Universal Time* [online]. 2009 [cit. 2009-08-19]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Coordinated_Universal_Time>.

[9] Adaptic, s.r.o.. *Adaptic : Co je to Mysql?* [online]. 2005 [cit. 2009-08-19]. Dostupný z WWW: <http://www.adaptic.cz/znalosti/slovnicek/mysql.htm>.

[10] University of California. *MVC Definition* [online]. 2003 , 24.5.2006 [cit. 2009-08-19]. Dostupný z WWW: <http://foozle.berkeley.edu/projects/streek/talks/mvc/definition.html>.

[11] Zend Technologies. *Zend Framework: Documentation* [online]. 2006 [cit. 2009-08-19]. Dostupný z WWW: <http://framework.zend.com/manual/en/zend.auth.html>.

[12] Zend Technologies. *Zend Framework: Documentation* [online]. 2006 [cit. 2009-08-19]. Dostupný z WWW: <http://framework.zend.com/manual/en/zend.translate.html>.

[13] Zend Technologies. *Zend Framework: Documentation* [online]. 2006 [cit. 2009-08-19]. Dostupný z WWW: <http://framework.zend.com/manual/en/zend.date.html>.

[14] Zend Technologies. *Zend Framework: Documentation* [online]. 2006 [cit. 2009-08-19]. Dostupný z WWW: <http://framework.zend.com/manual/en/zend.db.html>.

[15] Zend Technologies. *Zend Framework: Documentation* [online]. 2006 [cit. 2009-08-19]. Dostupný z WWW: <http://framework.zend.com/manual/en/zend.file.html>.

[16] Zend Technologies. *Zend Framework: Documentation* [online]. 2006 [cit. 2009-08-19]. Dostupný z WWW: <http://framework.zend.com/manual/en/zend.session.html>.

[17] Zend Technologies. *Zend Framework: Documentation* [online]. 2006 [cit. 2009-08-19]. Dostupný z WWW: <http://framework.zend.com/manual/en/zend.config.html>.

[18] Zend Technologies. *Zend Framework: Documentation* [online]. 2006 [cit. 2009-08-19]. Dostupný z WWW: <http://framework.zend.com/manual/en/zend.registry.html>.

[19] BÍLEK, Petr. *Sally - Programování C a C++ : Práce s binárními soubory a pohyb po souborech* [online]. 2003 , 25.9.2008 [cit. 2009-08-19]. Dostupný z WWW: <http://www.sallyx.org/sally/c/c33.php>.

[20] BRÁZA, Jiří. *PHP5: začínáme programovat* [online]. 2005 [cit. 2009-08-19]. Dostupný z WWW: <http://books.google.cz/books?id=9RHzUTsTxqUC&pg=PA162&lpg=PA162&dq=php+%C4%8Dten%C3%AD+bin%C3%A1rn%C3%ADch+dat&source=bl&ots=H4HdlNyUJx&sig=fKZy44uEXpSzLgCqbFBngm-28UE&hl=cs&ei=QnCISpebPMuRsgb-tJTqBw&sa=X&oi=book_result&ct=result&resnum=1#v=onepage&q=&f=false>.

[21] Czech Technical University. *PHP : fopen* [online]. 2001 , 14.8.2009 [cit. 2009-08-19]. Dostupný z WWW: <http://cz2.php.net/manual/en/function.fopen.php>.

[22] Czech Technical University. *PHP : fseek* [online]. 2001 , 14.8.2009 [cit. 2009-08-19]. Dostupný z WWW: <http://cz2.php.net/manual/en/function.fseek.php>.

[23] Czech Technical University. *PHP : usort* [online]. 2001 , 14.8.2009 [cit. 2009-08-19]. Dostupný z WWW: <http://cz2.php.net/manual/en/function.usort.php>.

[24] Czech Technical University. *PHP : isset* [online]. 2001 , 14.8.2009 [cit. 2009-08-19]. Dostupný z WWW: <http://cz2.php.net/manual/en/function.isset.php>.

[25] Czech Technical University. *PHP : hexdec* [online]. 2001 , 14.8.2009 [cit. 2009-08-19]. Dostupný z WWW: <http://cz2.php.net/manual/en/function.hexdec.php>.

[26] Auto-plus. *Business* [online]. 2009 [cit. 2009-08-19]. Dostupný z WWW: <http://www.auto-plus.cz/magazin/jaro06/business.php>.

[27] Výzkumný ústav bezpečnosti práce. *BOZP info : Doprava* [online]. 2002 [cit. 2009-08-19]. Dostupný z WWW: <http://www.bozpinfo.cz/rady/otazky_odpovedi/doprava/kniha_jizd061218.html>. ISSN 1801-0334.

[28] RADOM, s.r.o.. *RADOM, s.r.o. : Mobilní a navigašní souprava SXM30* [online]. 2009 [cit. 2009-08-19]. Dostupný z WWW: <http://www.radom.eu/produkty-a-sluzby/lokalizace-vozidel-a-osob/mobilni-a-navigacni-souprava-sxm30d.htm>.

[29] RADOM, s.r.o.. *RADOM, s.r.o. : RACAR INTERNET* [online]. 2009 [cit. 2009-08-19]. Dostupný z WWW: < RADOM, s.r.o.. *RADOM, s.r.o. : Mobilní a navigašní souprava SXM30* [online]. 2009 [cit. 2009-08-19]. Dostupný z WWW: <http://www.radom.eu/produkty-a-sluzby/lokalizace-vozidel-a-osob/mobilni-a-navigacni-souprava-sxm30d.htm>.