VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Informační systém evidence nákladů vozidel

Information System for Manage Cost of Vehicles

2011

Vojtěch Vrbka

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava*.

V Ostravě 6. Května 2011 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. Května 2011 .....

Rád bych poděkoval vedoucímu bakalářské práce za cenné rady, které mi zjednodušily moji práci a mým rodičům za jejich podporu

Abstrakt

Tato bakalářská práce se zabývá evidencí nákladů motorových vozidel a jejich dalším zpracováním. Nejprve je v krátkosti rozebrána problematika měření nákladů. Následuje analýza konkurenčních služeb, která popisuje jednotlivé služby a motivaci uživatelů je používat. V třetí části je analýza nově vzniklé aplikace. V analýze jsou přesněji specifikovány požadavky, podle kterých je provedena datová a funkční analýza. Poslední část obsahuje návrh a implementaci. Tato kapitola obsahuje popis technického zázemí, struktury aplikace a konkretní ukázky realizace.

Klíčová slova: Mirah, AppEngine, HTML5, CSS, webová aplikace

Abstract

This thesis deals with the recording of vehicles costs and their further processing. First, it briefly analyzes the problems of measuring costs. Followed by an analysis of competing services, which describes each service and motivate users to use them. The third part is an analysis of created application. The analysis covers details of requirements specification which are base for data and functional analysis. The last section includes the design and implementation. This chapter describes the technical background of the application and implementation of concrete examples.

Keywords: Mirah, AppEngine, HTML5, CSS, web application

Seznam použitých zkratek a symbolů

HTML	–	Extensible HyperText Markup Language
CSS	–	Cascading Style Sheets
API	–	Application Programming Interface
ORM	–	Object-relational mapping
PAAS	–	Platform as a service
MVC	–	Model-view-controller
REST	–	Representational State Transfer

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tato bakalářská práce se zabývá evidencí nákladů motorových vozidel. V dnešní době, kdy cena pohonných hmot (znázorněno na obrázku 5) neustálé stoupá, problém spotřeby aut se stává aktuálnějším. Většina řidičů ví, že faktory jako typ pneumatik, typ cesty či styl jízdy má na spotřebu paliva vliv, ale nemají již v rukou nástroj, který by jim poskytl reálné data. Z těch je možné například zjistit, že u Dieslových motorů má styl jízdy nezanedbatelný vliv na spotřeba i když se traduje opak. [1]

Při evidenci všech nákladů může systém sloužit pro určení vhodného auta pro koupi. Myslím si, že stále mnoho lidí si neuvědomuje, že náklady auta zdaleka nekončí u nákupu pohonných hmot. Přitom ostatní náklady mohou být v některých případech i větší než částky za pohonné hmoty.

Mým cílem je tedy poskytnout řidičům dostatečně flexibilní nástroj pro evidenci těchto nákladů a jejich přehlednou interpretaci. Na trhu je již více aplikací, které se touto problematikou zabývají. Protože se orientuji na fyzické osoby nebudu brát v úvahu produkty zabývající se velkými řešeními pro firmy. Většina aplikací je bohužel zastaralá, neudržovaná a na dnešní dobu poněkud těžkopádná. Analýzu konkurenčních aplikací jsem podrobněji rozepsal v kapitole 4.

Na základě těchto poznatků si myslím, že je na trhu prostor pro moderní řešení se snadnou evidenci vlastních vozidel a jejich nákladů. Tato aplikace bude zároveň sloužit jako virtuální profil auta.

[1] Toto usuzuji z vlastní zkušenosti

2 Stanovení a popis cílů

Cílem práce je vytvoření webové aplikace pro správu nákladu na vozidlech. Do aplikace se bude moct zaregistrovat libovolný počet uživatelů. Každý uživatel bude mít vlastní profil pod kterým budou evidována jejich vozidla u kterých se poté budou evidovat jednotlivé náklady a zobrazovat statistiky.

- Analyzujte konkurenční služby
- Definujte požadavky na informační systém a vypracujte analýzu informačního systému
- Navrhněte a implementujte daný systém v jazyce Mirah na platformě Google AppEngine.

Prvním cílem bude analýza konkurenčních služeb poskytující aplikaci, která se zabývá evidenci nákladů nebo jen výpočtem spotřeby vozidla. Prvním krokem tedy bude vyhledání konkurenčních služeb a jejich zařazení podle platformy aplikace do kategorií. U jednotlivých kategorií se vybere vždy nejvýraznější služba [2] U této aplikace se rozebere podrobněji funkčnost. Nejzajímavější informace je náročnost zadávání dat do systému a jejich množství. Mezi další bych zařadil také motivaci uživatelů danou aplikaci používat.

Po analýze konkurenčních služeb je na řadě definice požadavků. Nejprve analyzuji požadavky na nový systém. Tato analýza obsahuje první návrh nové aplikace. Dále je třeba specifikovat funkční a nefunkční požadavky. Mezi tyto požadavky patří motivace proč právě tento systém vytvářet, určení kdo jsou potenciální uživatelé systému, vstupy a výstupy aplikace. Samotná analýza aplikace je rozdělena do více částí. První je datová analýza, které obsahuje lineární zápis, konceptuální schémata a třídní diagramy. Druhá bude funkční analýza, ta obsahuje kontextové diagramy, use case a diagramy aktivit. V poslední části bude návrh vzhledu uživatelského rozhraní pomocí wireframů.

Poslední cíl je návrh a implementace systému. Implementace bude provedena v jazyku Mirah, který běží na platformě Google AppEngine. Prvním krokem tedy je ovládnutí jazyku Mirah a seznámení se s platformou AppEngine. Po zvládnutí technologie začne samotná realizace systému. První bude třeba vytvořit Modely a Controllery [3] nutné pro ovládání aplikace. Zvláštní pozornost je třeba věnovat také autentizaci uživatelů. Na závěr se implementuje grafický design.

[2] Tím je myšlena aplikace, která má největší počet uživatelů nebo má nejzajímavější funkcionalitu

[3] Modely a Controllery jsou používané v softwarové architektuře MVC, kterou základní framework Mirah používá.

3 Úvod do problematiky měření nákladů

Před samotnou analýzou konkurenčních aplikací bych chtěl shrnout způsoby měření nákladů a jejich evidenci.

3.1 Jak správně měřit náklady na vozidle

V dnešní době hodně řidičů [4] využívá předností palubního počítače. (obrázek 2) Ten ukazuje spotřebu paliva vozidla v Evropě udávanou v jednotkách l/100km. Pro základní informace o spotřebě je tento údaj dostačující. Dá se z něj totiž snadno zjistit cena ujetého km. Údaje z palubního počítače je třeba brát s rezervou, protože často mají odchylku. Přesnost se liší podle výrobce a modelu auta.

Pro přesnější údaje palubní počítač stačit nebude. Pokud chceme přesnější informace za delší časové období [5] je třeba sáhnout po komplexnějším řešením. Nabízí se webové či mobilní aplikace zabývající se evidencí nákladů a spotřeby motorového vozidla. Evidence těchto nákladů je sice značně časově náročnější než prosté sledování palubního počítače, ale odměnou nám budou komplexní přehled nad náklady a zajímavé statistiky, které bychom jinak nezískali.

Obrázek 1: Palubní počítač ve vozech Audi

3.2 Co evidovat

Tankování paliva je od jiných nákladů natolik unikátní, že jsem mu věnoval zvláštní kapitolu. Aplikace pracuje s tankování a ostatními náklady jiným způsobem.

3.2.1 Tankování paliva

Pokud mají statistiky odpovídat realitě, je třeba evidovat všechny tankování paliva. I jedno vynechané tankování může závěrečný výpočet hodně změnit. Pokud náhodou

[4] První palubní počítače se začaly objevovat na konci 90. let. V dnešní době u vozidel střední třídy bývají součástí základní výbavy. U starších aut byly pouze v nadstandardní výbavě. Najde se tedy stále dost řidičů, kteří tuto možnost nemají

[5] Palubní počítače jsou schopny uchovávat údaje za určitý počet km. Např. spotřeba za posledních 500km

uživatel zapomene čerpání zapsat, je třeba tuto skutečnost aplikaci sdělit. Tento problém se řeší více způsoby. První možnost je tankování odhadnout. U odhadovaného tankování se zapíšou informace, které se velmi přibližují reálnému tankování. Druhá možnost je zapsat pouze informace, které máme z tankování k dispozici a označit jej jako špatné. Pokud má řidič ještě účet, je z něj schopen zjistit objem paliva a jeho cenu. Aplikace v tomto případě sama další údaje odhadne. Tyto kroky nám zajistí integritu dat a můžeme se u výsledných statistik spolehnout, že odpovídají realitě.

Pro evidenci spotřeby vozidla je potřeba velmi málo informací, protože výpočet pouhé potřeby není příliš složitý. Potřebujeme znát jen počet ujetých kilometrů od posledního tankování a množství natankovaného paliva. Ale protože z těchto údajů není možné zjistit příliš zajímavé statistiky, do základních údajů bych zařadil i datum, celkovou cenu čerpání a stav tachometru pro pohodlí uživatele. Z těchto údajů se dají udělat již dostatečné statistiky. Opravdu zajímavá data můžeme získat pokud do evidence tankování zahrneme i faktory, které přímo či nepřímo ovlivňují samotnou spotřebu. Řidiči se nabízí možnost podle nově nabytých informací snížit vlastní spotřebu. Mezi tyto faktory patří styl jízdy, typ pneumatik, typ cesty, použití klimatizace a další.

3.2.2 Ostatní náklady

Provoz každého vozidla není jen pouhé čerpání paliva, ale častější menší či větší náklady. První a největší investice je koupě vozidla. Tato cena se ve statistikách rozloží na určité užitné období, které si zvolí sám uživatel. Další náklady bych rozdělil na pravidelné náklady (po určité době nebo počtu km) to zahrnuje pojištění vozidla (povinné ručení, havarijní pojištění), nákup a výměna pneumatik, pravidelný servis, STK a další. Druhá kategorie jsou nepravidelné náklady mezi které patří havárie, některá část vozidla vypoví službu a další. Všechny tyto náklady musí být zahrnuty do výsledných statistik. Pokud si řidič vše poctivě eviduje je schopen získat reálný pohled na cenu provozu vozidla.

4 Konkurenční služby

Protože se bakalářská práce zabývá vývojem webové aplikace budu analyzovat pouze webové a mobilní aplikace. Tento typ aplikací je oblíbenější a více používaný.

4.1 Webové služby

Trend posledních let všechny aplikace přesouvat na internet se nevyhnul ani tomuto odvětví. Hlavní výhoda aplikací, které běží na webu je, že uživatel má svoje data přístupná na kterémkoli počítači nebo mobilním přístroji. U tohoto druhu aplikací jsou data všech uživatelů v jedné centrální databázi. Je tedy možná vytvářet i globální statistiky nebo statistiky jednotlivých automobilových značek. Po registraci si uživatel přidá vlastní vozidla, u kterých zadá základní informace. Pro globální statistiky je důležité znát výrobce, model auta, jeho motor a výkon. Pro samotný výpočet spotřeby je třeba znát jednotky palubního počítače. K těmto vozidlům již může přidávat jednotlivé tankování a další náklady.

Konkurenční služby bych rozdělil na dvě odvětví. Jedny obsahují pouze základní informace o tankování (např. fuelfrog, fuelly) a druhé obsahují více informací (např. spritmonitor, spotřeby). Konkretně u spritmonitoru je zadávací formulář tankování rozdělen na tři části: Základní data, Jízdní podmínky a Informace o čerpací stanici. V základních datech je třeba zadat zejména datum čerpání, zda se jedná o čerpání "do plné nádrže", stav tachometru, objem a cena pohonné hmoty. Sekce jízdní podmínky je volitelná, ale pro další statistiky se vyplatí ji vyplňovat. Obsahuje typ pneumatik, styl jízdy, typy cest a další věci, které mohou ovlivnit samotnou spotřebu. Třetí část, jak již název napovídá, obsahuje informace o konkretní čerpací stanici. Po zadání určitého počtu čerpání či nákladů již může uživatel sledovat vlastní statistiky. Pokud bude službu používat určitou dobu bude později zařazen i do globálních statistik aut. Vlastní statistiky je možné porovnávat s jinými vozidly. Statistiky jsou velmi strohé. Vzhledem k počtu uživatelů v systému to považuji za nevyužitý potenciál získaných dat.

4.1.1 Motivace uživatelů

Zaměřím se na jednu z největších služeb a to konkretně na spritmonitor.de Tato služba eviduje celkem 274 752 vozidel u kterých je evidováno 3 569 699 215 Km. To je průměrně přibližně 13 000 Km na jedno vozidlo. Průměrný dojezd vozidla na plnou nádrž může být v rozmezí 500-12000 Km. Tedy každý uživatel zapsal průměrně 11-26 tankování. Myslím, že tyto čísla již svědčí o velké úspěšnosti zmíněné aplikace. [6]

Uživatelé těchto aplikací bych rozdělil do dvou kategorií. Do první patří šetřivý uživatel, který si zaznamenává každou korunu, kterou do auta vložil, snaží se jezdit s malou spotřebou a pečlivě sleduje statistiky. V druhé kategorii jsou uživatelé, kteří jsou téměř přesný opak. Jedná se o velké fanoušky aut. Tito uživatelé pravidelně navštěvují automobilová fóra, účastní se srazů a také si vlastní auto upravují k obrazu svému. Nemyslím si, že jim jde tolik o to všechny náklady mít zapsané, aby mohli ušetřit. Nejspíše vnímají aplikaci jako svou virtuální garáž.

Obrázek 2: Spritmonitor.de - služba pro měření nákladů na vozidle [6]

4.1.2 Srovnání

Název služby	**spritmonitor**[6]	**spotřeby**[7]	**mpgtune**[8]	**fuelfrog**[9]	**fuelly**[10]
Podrobné tankování	●	●	●	○	○
Jiné náklady	●	●	●	○	●
Mobilní verze	●	○	○	○	●
API	○	○	○	●	○
Globální statistiky	●	○	●	○	○
Jazyky	en, de	en, sk, cz	en	en	en
Počet uživatelů	216 560	6 151	5300	-	49 599
Ev. kilometrů	3 567 M	51 M	-	-	585 M

Tabulka 1: Porovnání webových aplikací

4.2 Mobilní aplikace

U těchto aplikací zadává uživatel data přímo do mobilní aplikace. To je pro uživatele velice pohodlné, protože může přímo na čerpací stanici si tankování zapsat či přímo v servise si zapsat opravu do systému. Zmenšuje se tedy pravděpodobnost, že uživatel na zapsání zapomene. Daň za toto pohodlí jsou většinou výrazně jednoduší aplikace, které nedosahují možností webových aplikací. Některé dokonce neumožňují evidenci jiných nákladů než tankování. Statistiky bývají často omezeny jen na základní údaje.

Obrázek 3: RoadTrip - iPhone aplikace pro měření spotřeby [11]

Mobilní aplikace mohou být řešeny následujícími způsoby:

4.2.1 Offline aplikace

Tento typ aplikací si ukládají data o tankování do vnitřní paměti telefonu. Výhoda je, že nemusí být aktivní připojení k internetu pro zapsání nákladu. Tím, ale veškeré výhody končí a nastupuje velké množství nevýhod. Pokud jsou náklady pouze ve vnitřní paměti můžeme je zadávat pouze na jednom telefonu a je zde vyšší riziko ztráty všech pracně zadávaných údajů. Osobně tyto aplikace nedoporučuji.

4.2.2 Online aplikace

Tyto aplikace si ukládají data na internet. Uživatel má tedy účet, pod kterým se v mobilní aplikaci přihlásí a data se pak ukládají přímo na internet. U této varianty je velká výhoda při nainstalování aplikace na jiný telefon, že lze snadno načíst již zadávaná data.

4.2.3 Online aplikace doplňující webovou aplikaci

Rozhodně nejlepší varianta je mobilní aplikace doplňující její webovou variantu. Zadávat data je tedy možné dvěma kanály, buď to mobilní verzí nebo přímo přes web. V tomto případě ani nevadí, že mobilní verze nezobrazuje kompletní statistiky, protože se na ně můžeme podívat přes web.

5 Analýza

5.1 Analýza požadavků

Našim cílem je vytvoření systému pro evidencí nákladů na vozidlech. V prvé řadě zde budou uživatelé, takže je třeba vytvořit systém pro registraci uživatelů. Uživatelé se poté budou moct přihlašovat a odhlašovat. Přihlášení uživatelé budou mít větší pravomoci a budou mít možnost přidávat vlastní vozidla.

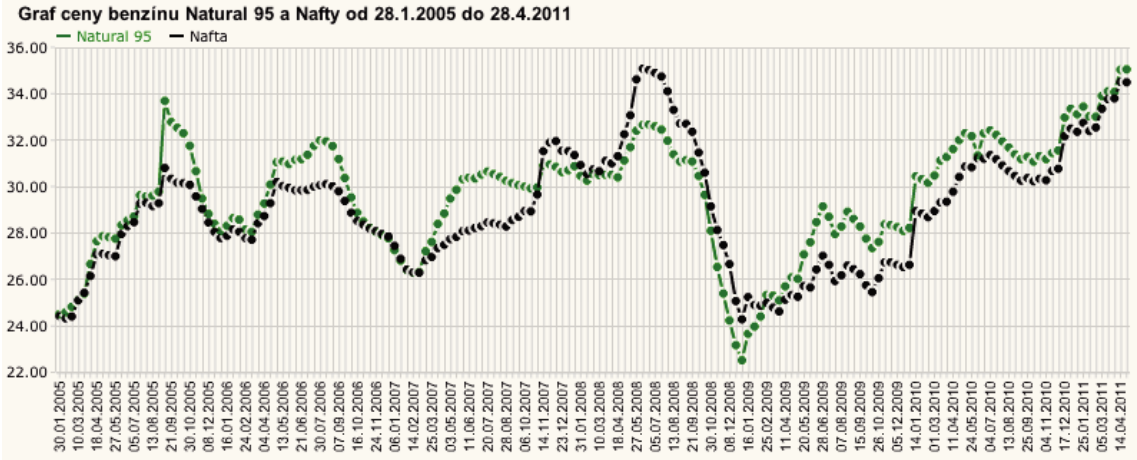
Každé vozidlo bude obsahovat povinné a volitelné údaje. Mezi povinné údaje patří veškeré údaje, které jsou nutné pro zařazení vozidla do systému a jeho jednoznačnou identifikaci. K vozidlu bude moct jeho majitel zaznamenávat čepování pohonných hmot a jiné náklady. Tyto náklady bude moct také editovat či mazat.

Po zadání určitého počtu čepování paliva se u vozidla zobrazí grafické statistiky. Tyto statistiky půjde omezit jen na určitý časový úsek. Budou rozděleny na dvě části. První bude zobrazovat spotřebu vozidla za daný časový úsek. Druhá část bude zobrazovat veškeré náklady za daný časový úsek.

5.2 Specifikace funkčních požadavků

5.2.1 Proč tento systém ?

Tento systém jsem si vybral, protože momentálně neexistuje dostatečně flexibilní aplikace u které je zadávání dat zábava a přitom je schopen ze zadaných dat vytvořit dostatečně zajímavé statistiky. Při neustálém zvyšování ceny pohonných hmot se problematika spotřeby paliva a faktorů, které na ní mají vliv dostává čím dál více do popředí, proto si myslím, že tato aplikace má smysl. Vývoj ceny benzínu za poslední roky je znázorněn na obrázku 5.



Obrázek 4: Vývoj ceny benzínu a nafty za posledních 5 let [12]

5.2.2 Kdo bude pracovat se systémem ?

Se systémem budou pracovat běžní uživatelé a administrátor. Pro běžné prohlížení aut, jejich porovnání a statistiky nebude třeba registrace. Pokud uživatelé budou chtít vytvořit si vlastní garáž bude třeba, aby se zaregistrovali. Po registraci může uživatel vkládat a upravovat vlastní auta a k těm poté přidávat náklady. Administrátor se bude starat o chod a správu aplikace.

5.2.3 Vstupy

Po uživatelích budeme vyžadovat následující data:

Vozidlo Aby bylo možné jednotlivé auta srovnávat je třeba znát výrobce auta, model, jeho konkretní označení a rok výroby. Pro porovnání spotřeby je velmi zajímavý i motor auta, tedy jeho objem a výkon a typ převodovky. Mezi volitelné data se pak řadí velikost nádrže a jednotky tachometru.

Protože aplikace má zároveň sloužit jako takový internetový profil vozidla, může si uživatel k vozidlu přidat fotografie a napsat k němu vlastní popis.

Čerpání paliva Většina statistik bude vycházet právě z tohoto údaje. Je třeba zjistit o jednotlivých čerpáních co nejvíce, ale zároveň nenutit uživatele vyplňovat příliš mnoho údajů. To se dá zařídit tím, že při každém vyplňování budou volitelné údaje již předvyplněné podle posledního čerpání. Uživatel poté změní jen podmínky, které se při tomto čerpání změnily. Například pneumatiky se přezouvají 2x ročně, tedy je zbytečné při každém čerpání vyplňovat, že má právě obuté zimní/letní pneumatiky.

Na základě těchto poznatků údaje o čerpání rozdělíme do dvou částí. Základní data a podmínky. V základních datech uživatel zadá datum čerpání, typ čerpání (plná nádrž, část nádrže), stav tachometru, množství paliva a jeho cenu. Mezi volitelnými je konkretní typ paliva. V druhé části budou údaje o pneumatikách, řidičském stylu, typu cest a další.

Jiné náklady U těchto nákladů je největší problém přesvědčit uživatele zadávat opravdu všechny údaje, protože ve výsledném porovnání aut právě tyto náklady budou dělat největší rozdíly. Může se stát, že se subjektivně bude zdát výhodnější určité auto, jen protože u něj uživatel tyto náklady nezadával a tedy vypadá jeho provoz levnější, což nutně nemusí reflektovat realitu. Hlavní cíl tedy zejména bude ukázat uživatelům, že tyto údaje jsou stejně důležité jako zadávání informací o čerpání paliva.

U tohoto typu nepotřebujeme znát tolik údajů jako u čerpání paliva, zadávání tedy bude o dost jednodušší. Samotný záznam tedy obsahuje datum, stav tachometru (nepovinný), typ, cena a poznámka. Údaje má uživatel většinou ještě na faktuře, takže není nutné data zadávat ihned. Vyplatí se tedy jednou za čas uživateli připomenout zda na něco náhodou nezapomněl.

5.2.4 Výstupy

Každý uživatel má možnost se podívat na statistiky svého vozidla. Statistiku si může zobrazit pouze za určité období, protože pro něj často může být zajímavý například jen poslední půlrok.

Jako první údaje se mu zobrazí cena za měsíc a cena za kilometr. Tyto údaje jsou pak graficky znázorněny, kolik z dané částky je čerpání paliva a ostatní náklady, které jsou rozepsány podle typu. Pod těmito údaji jsou další velmi zajímavé grafy. Graf zobrazující spotřebu paliva za jednotlivé měsíce a další graf zobrazující měsíční náklady na vozidlo ve kterém je znázorněno, která část je spotřeba paliva a která obsahuje jiné náklady. Z těchto grafů je již možné si udělat základní představu o reálných nákladech.

5.2.5 Funkce

Běžný uživatel

- Prohlíží vozidla
- Vyhledává vozidla
- Sleduje statistiky
- Registruje se

Registrovaný uživatel

- Přihlásí se
- Prohlíží vozidla
- Vyhledává vozidla
- Přidává vozidlo do garáže
- Upravuje vlastní vozidla
- Přidává tankování paliva
- Přidává jiné náklady
- Odhlásí se

API - Cizí aplikace

- Autorizuje se
- Prohlíží náklady
- Přidává tankování paliva
- Přidává jiné náklady

V tabulce 4 se můžete podívat na příklad událostí a reakcí systému na danou událost.

Udalost	Reakce systému	Aktér
Registruje se	Přidej nového uživatele	Běžný uživatel
Vyhledává vozidla	Vyber vozidla	Běžný uživatel
Přihlásí se	Vyber uživatele a přihlaš jej	Registrovaný uživatel
Přidává vozidlo do garáže	Přidej vozidlo	Registrovaný uživatel

Tabulka 2: Příklad jednotlivých událostí a reakcí systému

5.2.6 Okolí

Nejlepší nástroj pro zobrazení okolí systému je kontextový diagram. Se systémem budou pracovat tři objekty. Běžný uživatel, registrovaný uživatel a administrátor systému. Kontextový diagram je na obrázku 8.

5.3 Specifikace nefunkčních požadavků

Aplikace pro správu nákladech na vozidlech požaduje:

- Pro rychlou aplikaci i při velkém množství uživatelů a vysoké návštěvnosti se jako technické řešení použije platforma Google AppEngine
- Zajištění testovacích dat pro odzkoušení funkčnosti aplikace

5.4 Datová analýza

V této části si popíšeme datovou analýzu, které bude obsahovat lineární zápis a konceptuální schéma. Datový slovník vynechám, protože projekt nepoužívá klasickou databázi a tedy není třeba tyto údaje určovat.

5.4.1 Lineární zápis

Lineární zápis je v tabulce 3. Entity jsou uloženy v databázi BigTable na Google Appengine. Podrobnější popis technologie je v kapitole 6.1.2. V tabulce jsou primární klíče vyznačeny tučně a cizí klíče kurzivou. Protože se nejedná o relační databázi cizí klíče neobsahují reálnou referenci na jiný primární klíč. Pouze je na nich index.

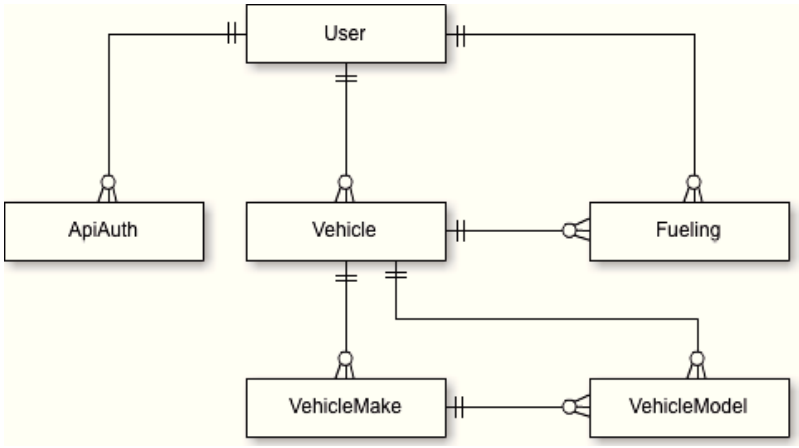
User	(**id**, name,email,password_hash,salt)
Vehicle	(**id**, *user_id* ,*make_id*, *model_id*, model_exact, fuel_type, gearing, year, engine_power, odometer_unit, tank_capacity, license_number, note, deleted)
VehicleMake	(**id**, name)
VehicleModel	(**id**, *make_id*, name)
Fueling	(**id**, *user_id*, *vehicle_id*, date, type, fueling_type, cost_type, odometer, trip, fuelsort, quantity, quantity_unit, price, price_currency, note, tires, driving, route_motorway, route_city, route_country_roads, ac, trailer)
ApiAuth	(**id**, *user_id*, last_used, session)

Tabulka 3: Lineární zápis

Databáze BigTable není relační a jednotlivé entity v databázi nemají žádné schéma. Schéma mají pouze jednotlivé záznamy. Dotazy na databázi se provádí odlišně, než v klasické relační databázi (MS SQL, Oracle, MySQL). Není možné jednotlivé entity spojovat na úrovni databáze, ale je nutno tyto operace provádět v aplikaci. [6] Kvůli této vlastnosti se při návrhu nesnažíme databázi uvést do určité normální formy. Naopak nejefektivnější dotazy budou, které vyberou v jednom dotazu všechny data, které právě potřebujeme.

5.4.2 ER-Diagram

ER-Diagram popisující vztah mezi entitami znázorněný na obrázku 6 ukazuje propojení entit v databázi.

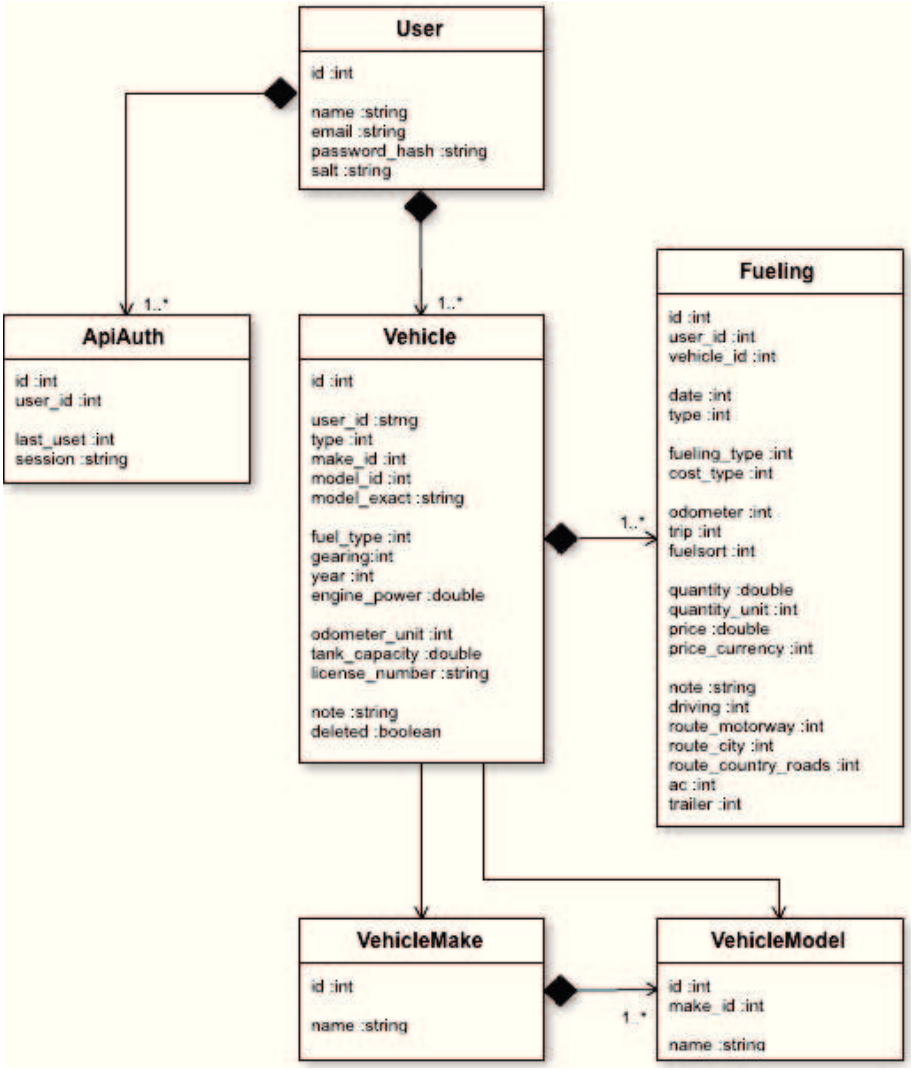


Obrázek 5: ER Diagram

[6] Nelze v jednom dotazu vybírat data z více entit

5.4.3 Třídní diagram

Na ER diagram navazuje Třídní diagram, který znázorňuje propojení tříd. Protože použitý programovací jazyk Mirah (popis v kapitole 6.1.3) používá ORM, které automaticky načítá objekty z databáze. Třídní diagram je na obrázku 7.

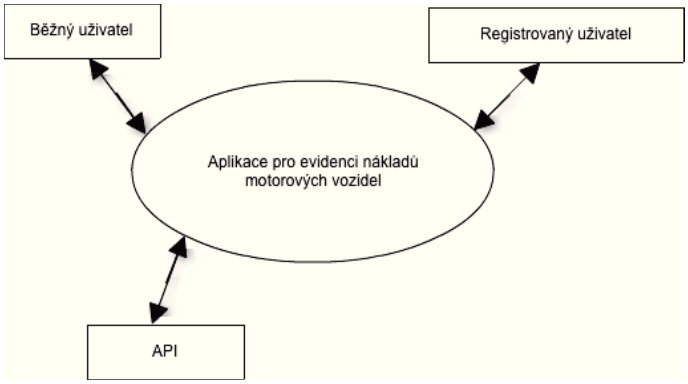


Obrázek 6: Třídní diagram - modelové třídy

5.5 Funkční analýza

5.5.1 Kontextový diagram

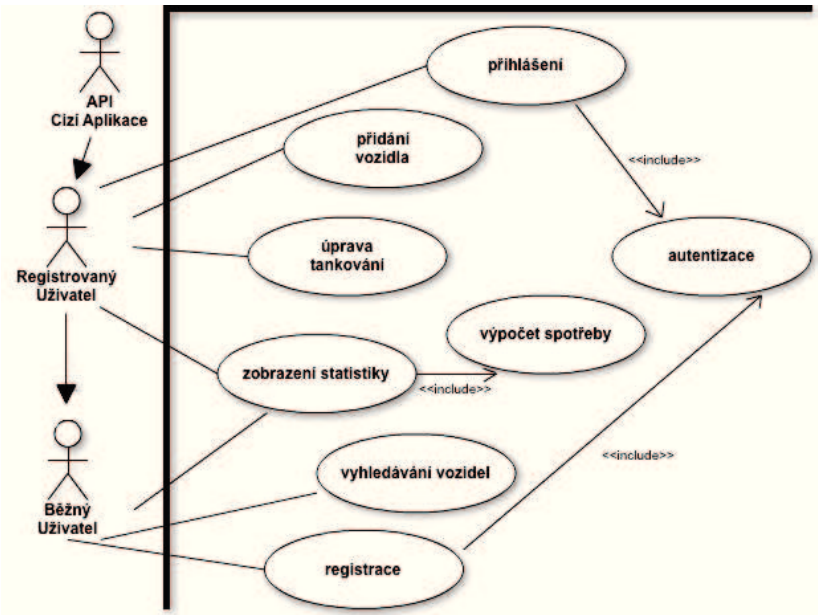
Kontextový diagram nám zobrazuje komplexní pohled na strukturu systému a nalezneme jej na obrázku 8.



Obrázek 7: Kontextový diagram

5.5.2 Use Case

Diagram Use Case se používá pro znázornění dynamické struktury systému z pohledu uživatele. Jsou zde tři aktéři Obyčejný uživatel, Přihlášený uživatel a API. Jejich role znázorňuje obrázek 9. Kompletní diagram je v příloze.



Obrázek 8: Use Case

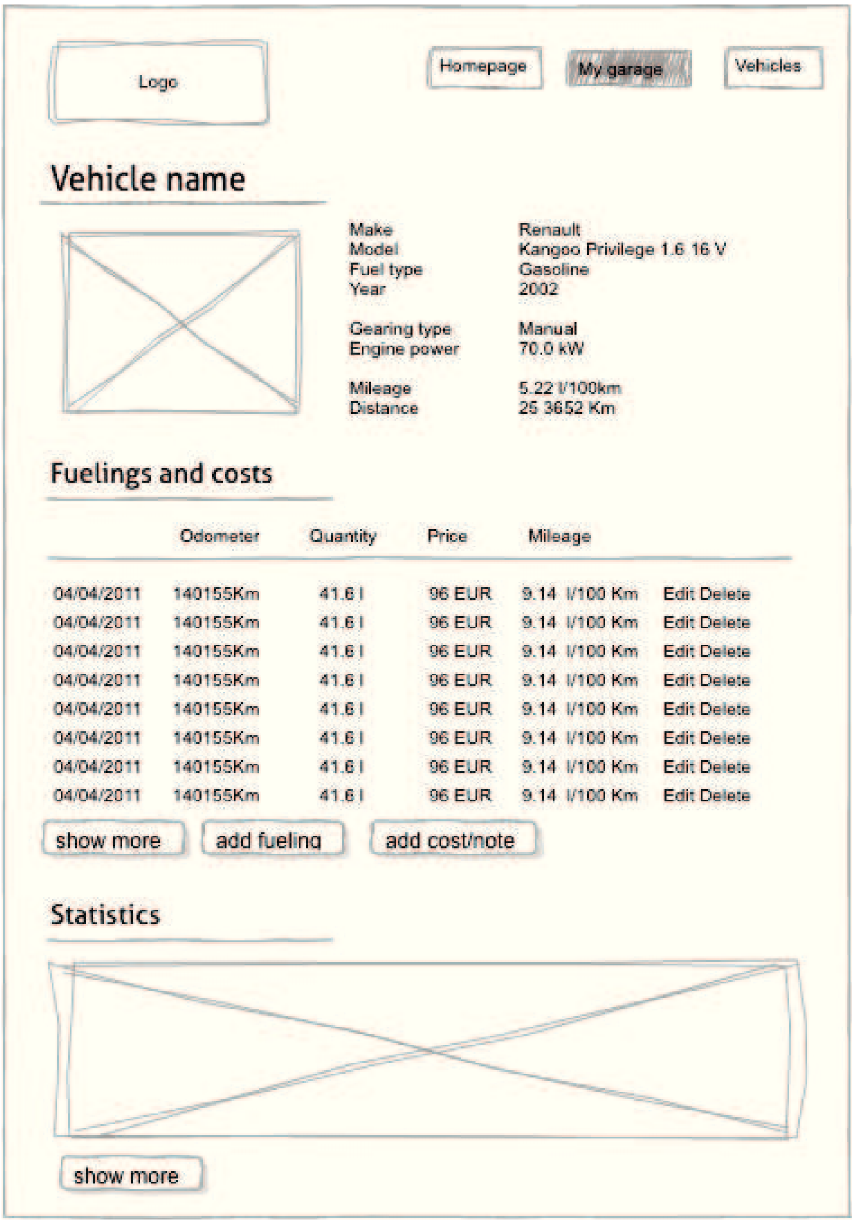
5.6 Grafický návrh

Při návrhu aplikace není možné podceňovat ani samotný grafický vzhled. Ten se navrhuje pomocí Wireframe, které se pak předají grafikovi a ten z nich udělá grafický návrh. Na obrázku 10 můžeme vidět wireframe úvodní stránky aplikace.

Obrázek 9: Wireframe úvodní stránky

U úvodní stránky je hlavní cílem rychle návštěvníkovi sdělit účel aplikace a proč by ji měl používat. Grafický vzhled by měl být dostatečně zajímavý, aby uživatel byl ochotný si vůbec přečíst popis. Hlavní část stránky je vlevo obrázek zobrazující uživatelskou část aplikace, která je přístupná, až po registraci. Napravo je v několika krátkých bodech popsána funkčnost aplikace. Abych měl jistotu, že uživatel rychle pochopí funkčnost aplikace, přidal jsem ještě popis "Jak to funguje". Ve třech bodech s krátkým popisem bude naznačena funkčnost aplikace.

Na obrázku 11 je wireframe zobrazující profil vozidla. Profil je stránka, která má uživateli přinést základní informace o každém vozidle. Tato stránka je rozdělena na tři části. První obsahuje základní technické údaje s fotografií. Za ní následuje seznam tankování paliva a jiné náklady. Pokud je uživatel vlastníkem vozidla může náklady ihned upravit nebo přidat nový záznam. Dole je jednoduchý graf zobrazující spotřebu uživatele. Z tohoto grafu se uživatel dostane do kompletních statistik.



Obrázek 10: Wireframe detailu vozidla

5.7 Možnosti monetizace

U každého projektu je potřeba již při návrhu myslet i na budoucí monetizaci. Úspěšnost projektu nutně nezmaná, že je možné na něm vydělat peníze.

5.7.1 Reklama

Nejběžnější, ale taky nejméně účinný způsob monetizace. Jediný efektivní způsob je nasazení velice specifické reklamy, která je úzce zaměřené přímo na cílové uživatele aplikace. Podle uživatelských profilů víme o uživatelích spoustu informací podle kterých můžeme cílit reklamu. Reklama by byla vhodná pro čerpací stanice, výrobce aut, nebo prodejce příslušenství.

5.7.2 Placená mobilní aplikace

Vytvoření mobilní aplikace, která by stála malou částku. Tento model i velice úspěšný, pokud má služba dostatečný počet uživatelů. Aplikace se nabízí za nízkou cenu (1-5 dolarů) a spoléhá se, že ji koupí velké množství lidí. V mobilních aplikací je velká konkurence a proto musí mít aplikace velkou úroveň. Při špatném návrhu aplikace a nedostatečné uživatelské základně aplikace může být aplikace prodělečná.

5.7.3 VIP uživatelé

Jedna z dalších možností je nabídnout rozšířenou funkčnost aplikace za příplatek. Ten může být buď to jednorázový nebo periodický (měsíční, roční). Přidaná hodnota pro VIP uživatelé musí odpovídat částce, kterou po nich požadujeme. Počet těchto uživatelů je pouze několik procent z celkového počtu. Uvažovat o této možnosti tedy vyžaduje již nějakou uživatelskou základnu.

5.7.4 Affilate prodej

Prodej přísluše To může být řešeno buď to přesměrování uživatele na stránky prodejce nebo prodejem přímo na stránkách aplikace s tím, že o samotnou dopravu se bude starat prodejce.

5.7.5 Sjednání povinného ručení

Vzhledem k povinnosti každého řidiče mít na své auto sjednané povinné ručení nabízí se tato možnost sama. Známe informace nutné pro výpočet ceny pojištění z kterých můžeme určit konkrétní cenu. Pokud bude cena pro uživatele výhodná je pravděpodobné, že tuto možnost využije.

6 Návrh a implementace

6.1 Technické zázemí

6.1.1 Platforma Google App Engine

Appengine je služba provozována společností Google od roku 2008. Jedná se o takzvaný PAAS (Platform as a Service) a nabízí kompletní platformu pro provozování webových aplikací. Podporované jazyky jsou Java a Python. Appengine obsahuje spoustu knihoven a API usnadňující tvorbu složitých služeb. Umožňuje snadno škálovat aplikace, které rostou od jednoho do milionu uživatelů bez problémů s infrastrukturou. Při růstu uživatelů nedochází k velkému zatížení aplikace, protože se pouze spouští další instance aplikace. Provozovatel platí jen za to co skutečně používá. U appengine jsou kvóty, které určují množství výkonu, prostoru a provedených operací zdarma. Konkretně jsou spolu s cenami vidět na obrázku 12.

Resource	Budget	Unit Cost	Paid Quota	Free Quota	Total Daily Quota
CPU Time	n/a	$0.10/CPU hour	n/a	6.50	6.50 CPU hours
Bandwidth Out	n/a	$0.12/GByte	n/a	1.00	1.00 GBytes
Bandwidth In	n/a	$0.10/GByte	n/a	1.00	1.00 GBytes
Stored Data	n/a	$0.005/GByte-day	n/a	1.00	1.00 GBytes
Recipients Emailed	n/a	$0.0001/Email	n/a	2,000.00	2,000.00 Emails
Always On	n/a	$0.30/day	n/a	none	
Max Daily Budget:	n/a				

Obrázek 11: AppEngine - Billing [1]

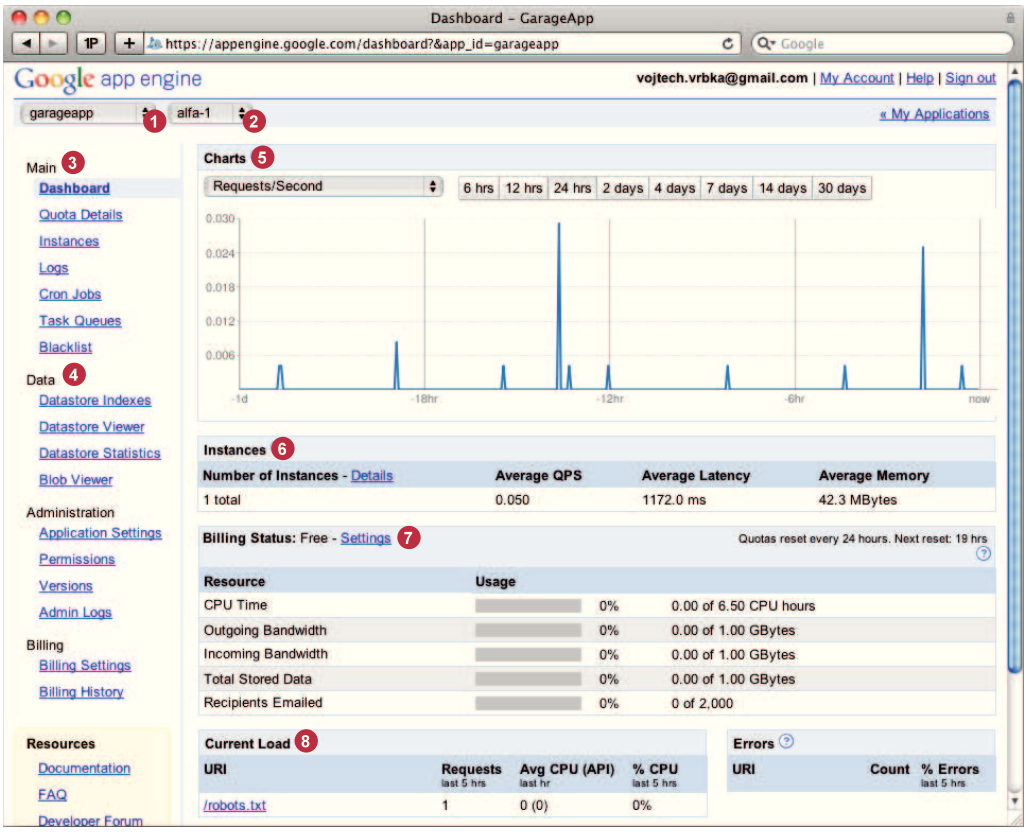
Dashboard Na obrázku 13 je zobrazena úvodní informační stránka správy aplikace takzvaný dashboard. Červenými body jsou pro přehlednost označeny hlavní části administrace. Vlevo nahoře můžeme vidět výběr aplikace (1) a zvolení její verze (2). Nalevo je hlavní menu (3), které je rozděleno do více částí.

První část se zabývá statistikami vytížení aplikace. Lze přehledně vidět, kolik je aktuálně spuštěných instancí a jak si vede aplikace vzhledem ke kvótám.

Druhá část Data (4) se zabývá operacemi nad daty uloženými v datastore. Jsou zde zobrazeny nastavené indexy a také je možnost jednotlivé objekty procházet, měnit či vytvářet nové. Vybírat data je možno jazykem GQL. Mimo jiné je zde i statistika zobrazující statistisku využití prostoru.

Třetí část Administration obsahuje nastavení přístupu k aplikaci. Je možné přidávat a odebírat administrátory. Také je možno nastavit aplikaci vlastní doménu. Mimo jiné tato část obsahuje velmi důležitou funkci a to je verzování. Každá verze má svoji vlastní testovací url na které daná verze běží. Vždy jen jedna verze běží na ostré doméně a je možné je jedním kliknutím přepínat.

Poslední část Billing má na starost nastavení placení za serverový výkon. Programátor si nastaví kolik chce maximálně zaplatit. Po překročení limitu zdarma je mu serverový výkon účtován, až do nastaveného limitu.



Obrázek 12: AppEngine - Dashboard [1]

6.1.2 Databáze BigTable

BigTable je distribuovaný úložný systém pro správu strukturovaných dat navržený, aby uchoval velmi velké množství (petabajtů) dat tisíců komoditních serverů. Google používá BigTable u vlastních projektů včetně indexace webu, Google Earth a Google Finance. Tyto aplikace mají velmi rozdílné nároky a to jak z hlediska velikosti dat(od adresy URL webových stránek po satelitní snímky) a vyžadované latence požadavku (od hromadného zpracování na pozadí po real-time odpovědi). BigTable poskytuje jednoduchý datový model, který dává vývojářům dynamickou kontrolu nad rozložením dat a jejich formátu. [2]

6.1.3 Programovací jazyk Mirah

Pro tuto bakalářskou práci byl zvolen jazyk Mirah [3], Jedná se o poměrně nový jazyk. Jeho syntaxe se velmi inspiruje jazykem Ruby a jeho zdrojové kódy se kompilují do Java bytekódu. Oproti ostatním jazykům má velkou výhodu v tom, že nevyžaduje žádnou runtime library. Jazyk Mirah byl vyvinut přímo pro nasazení na Google Appengine jako alternativa k Jruby. Jruby je java server realtime implementující ruby. [5]

```
public class Fibonacci {
  public static int fib(int n) {
    if (n < 2) {
      return n;
    } else {
      return fib(n-1) + fib(n-2);
    }
  }
}
```

Výpis 1: Fibonacciho posloupnost v Javě

Na výpisech 1 a 2 je možno porovnat jak se oba jazyky liší. Jazyk Mirah považuje kontrolní strukturu IF za výraz a implicitně vrací jeho hodnotu. Má také podstatně kratší zápis a další drobné vylepšení, které vychází z jazyka Ruby.

```
def fib(a:int)
  if a < 2
    a
  else
    fib(a-1) + fib(a-2)
  end
end
```

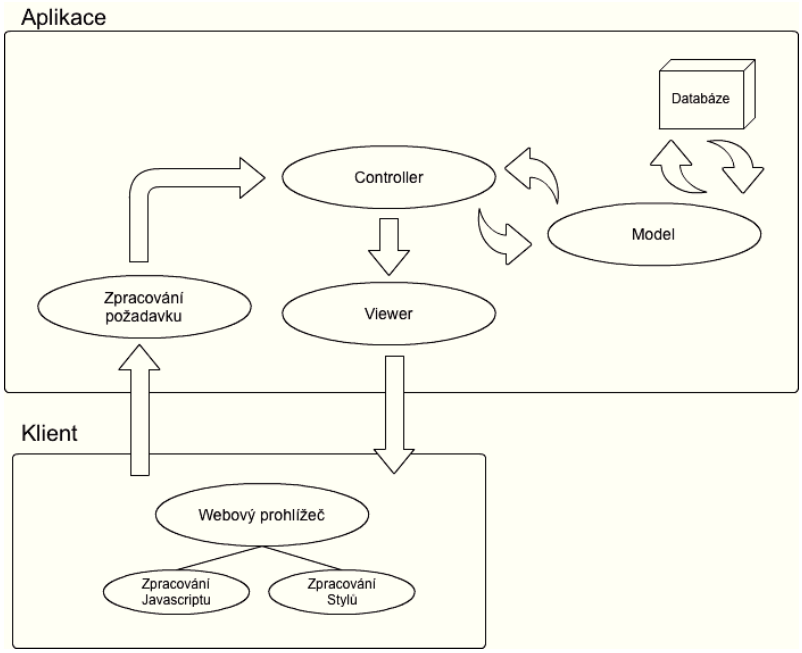
Výpis 2: Fibonacciho posloupnost v Mirah

6.1.4 Framework Dubious

Zároveň s jazykem Mirah je dodáván i framework nazvaný Dubious. Tento framework obsahuje knihovny bez kterých bychom se neobešli. Nejdůležitější část je pravděpodobně ORM (Object-relational mapping), které umožňuje mapování dat uložených v datastore přímo na objekty. Obsahuje také základní controllery, které umí sami již zpracovávat GET/POST requesty. [4]

6.2 Struktura aplikace

Při poslání požadavku na server dojde k rozlišení správného controlleru podle URL. Následně se vytvoří instance zvoleného controlleru. V controlleru se pracuje s modely, které si načítají data z databáze. Modely a další data nutné pro výstup se pošlou do šablony. Po zpracování šablony se výsledek odešle uživateli do prohlížeče. Prohlížeč provede nastylování stránky podle CSS a spustí javascriptové skripty.



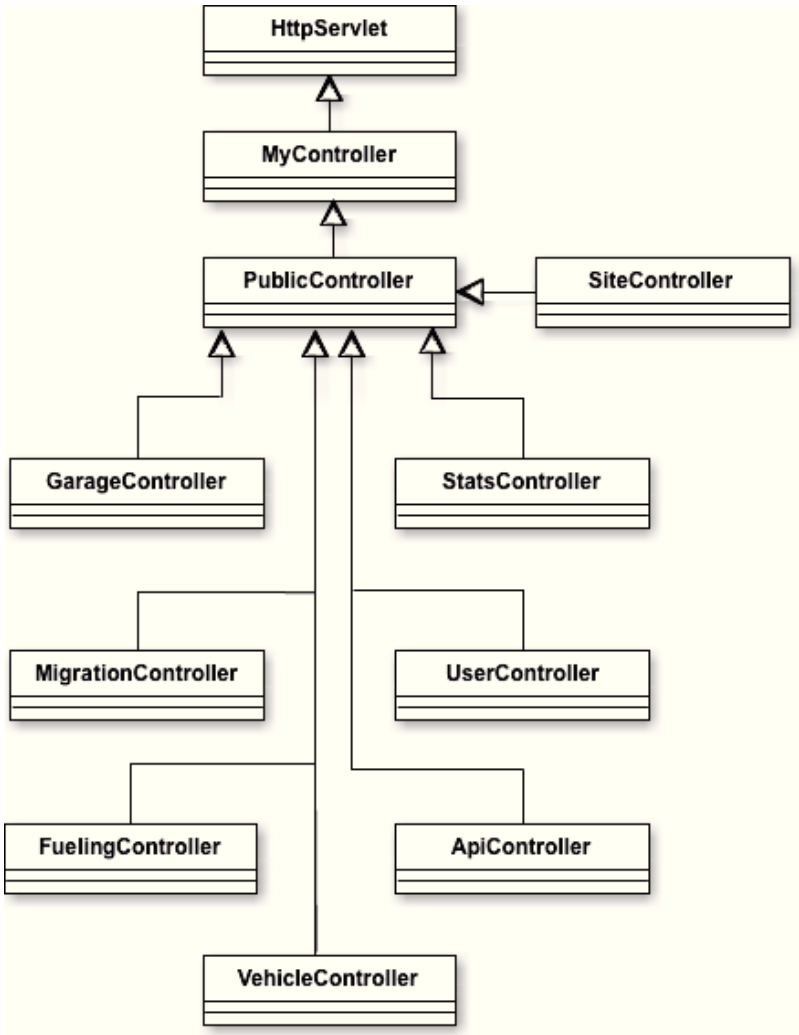
Obrázek 13: Struktura aplikace

6.3 Adresářová struktura

Složka nebo soubor	**Popis**
app/	Obsahuje všechny zdrojové kódy aplikace.
app/controllers/	Controllery
app/ext/	Třídy rozšiřující framework Dubious
app/models/	Modely
app/tasks/	Úlohy pro službu cron
app/widgets/	Moduly pro zobrazení určité části webu
app/views/	Šablony
config/	Konfigurační soubory
public/	Veřejný adresář
public/img/	Obrázky designu
public/js/	Javascripty
public/stylesheets/	CSS styly
WEB-INF/app.yaml	Konfigurační soubor aplikace
WEB-INF/cron.xml	Nastavení služby cron
WEB-INF/ appengine-generated/	Obsahuje lokální databázi a indexy
WEB-INF/classes/	Třídy zkompilované do Java-byte kódu
WEB-INF/lib/	Java jar knihovny.

Tabulka 4: Adresářová struktura aplikace

6.4 Třídní diagram - Controllery



Obrázek 14: Třídní diagram - Controllery

6.5 Autentizace uživatelů

Prvním krokem realizace po vytvoření nového čistého projektu bylo návrh systému pro registraci, přihlášení a odhlášení uživatelů. Tento systém musí být dostatečně robustní, aby i při ukradení cookies nebo nabourání do databáze útočník nezcizil uživatelovu identitu.

První nejjednodušší parametr, který snižuje možnost prolomení hesla je nastavení určité úrovně jeho složitosti. V tomto případě jsem zvolil minimum 8 znaků.

Pro bezpečné uložení hesla v databázi je každému uživateli vygenerován náhodné číslo, které se používá jako salt. Pomocí něj je pak heslo zadané uživatelem převedeno

přes algoritmus SHA-256 na hash a ten je uložen v databázi. Pokud se tedy útočníkovi podaří nějakým způsobem dostat do databáze hesla uživatelů budou stále v bezpečí. Ukázka zdrojového kódu jde vidět ve výpisu 3 a 4.

6.6 Vytvoření Modelů

Nejprve jsem podle analýzy vytvořil navrhlé modely . Těmto modelům jsem přidal parametry. O ukládání do databáze jsem se již nemusel starat, protože Mirah obsahuje ORM. U většiny modelů bylo třeba ještě dopsat pomocné metody a konstanty, které usnadňují práci s modely.

Modely:

- **User** - uživatelé aplikace
- **Vehicle** - motorová vozidla
- **VehicleMake** - výrobci vozidel
- **VehicleModel** - modely vozidel
- **Fueling** - čerpání paliva nebo jiné náklady
- **ApiAuth** - autorizace uživatelů pro přístup k api

Ve výpisu 3 můžeme vidět část [7] modelu User obsahující metody pro registraci a přihlášení uživatele. Veškeré modely dědí třídu Model, která obsahuje metody pro práci s databází. Vlastnosti modelu se nastavují jen v této třídě, nikde není třeba určovat strukturu databáze. Mezi vlastnosti se nezapisuje primární klíč (id), ten si databáze určuje sama.

Metoda self.register registruje uživatele. Self v názvu říká, že je statická, můžeme ji tedy volat new_user = User.register(email, pass). Na první pohled vypadá, že pouze ukládá heslo do vlastnosti password. Ve skutečnosti přiřazení volá metodu password=-(pass:String). Rovnítko v názvu znamená, že se tato metoda používá při přiřazení proměnné. Metoda password= vygeneruje salt a provede otisk hesla. Pro uložení musíme ještě zavolat new_user.save

```
class User < Model
  property :name, String
  property :email, String
  property :password_hash, String
  property :salt, String
  property :locked, Boolean

  def self.register(email:String, password:String) # registrace uzivatele
    user = new
    user.email = email
```

[7] Model user obsahuje ještě další metody včetně validace.

```
  user.password = password
  user
end

def password=(pass:String) # prirazeni hesla -> vytvoreni otisku
  @new_password = pass
  self.salt = generate_salt
  self.password_hash = hash(pass, salt)
  null
end

def self.login(email:String, pass:String)    # prihlaseni uzivatele
  if user = all.email(email).first  # kontrola zda uzivatel existuje
    if user.password_hash.equals(user.hash(pass, user.salt)) # kontrola zda se shoduje otisk
        hesla
      user
    else
      raise ClientException.new('Incorrect Email or Password.')
    end
  else
    raise ClientException.new('Incorrect Email or Password.')
  end
end

def generate_salt
  BigInteger.new(256, SecureRandom.new).toString(64) # vygenerovani nahodneho saltu
end

def hash(password:String, salt:String); returns String # vytvoreni otisku hesla
  digest = MessageDigest.getInstance("SHA-256")
  digest.reset()
  digest.update(salt.getBytes())
  return Base64.encodeBase64String(digest.digest(password.getBytes("UTF-8")))
end
```

Výpis 3: Část modelu User s metodami pro registraci a přihlášení

6.7 Vytvoření Controllerů

S modely potřebujeme samozřejmě i pracovat, takže přišly na řadu Controllery. Většina modelů má vlastní controller a přibyly ještě další, které se odvíjely od obsahu webu.

Controllery:

- **UserController** - Stará se o uživatele
- **GarageController** - Garáž uživatele, přidávání/editace/mazání aut v garáži
- **VehicleController** - Vyhledávání a zobrazování vozidel
- **FuelingController** - Přidávání/Editace čerpání paliva a jiných nákladů
- **StatsController** - Výpočet a zobrazení statistik

- **SiteController** - rozhoduje který controller a kdy použít

Ve výpisu 4 můžeme vidět část controlleru UserController, který se stará o práci s uživateli. Metoda login se stará o přihlášení uživatele, logout o jeho odhlášení a signup o registraci.

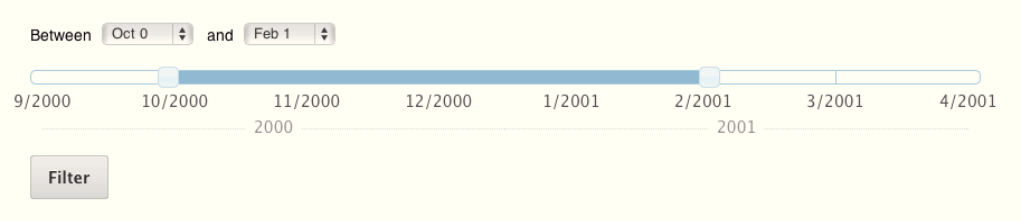
```
class UserController < SiteController

def login
  @error = ""
  if params.has(:email)   # obsahuje request email ?
    begin # pokus o prihlaseni uzivatele
      self.user = User.login(params[:email], params[:password])
      redirect_to '/'
    rescue Exception => ex # odchyceni vyjimky
      @error = ex.getMessage
    end
  end
  layout login_erb # vykreslene login formulare
end

def logout
  session = request.getSession()
  session.setAttribute("user_id", null) # smazani session
  if Application.development? # pro localhost development
    cookie = Cookie.new(:user_id, '')
    cookie.setPath('/')
    params.response.addCookie(cookie)
  end
  redirect_to :index
end

def signup
  @error = ""
  if params.has(:email) # je v parametrech email ?
      email = params['email']
      pass = params['password']
      pass_confirm = params['password_confirm']

      if user = User.all.email(email).first # nema jiz tento email jiny uzivatel ?
        @error = 'That email is already in use, please choose a new one'
      else
        begin
          pass.equals(pass_confirm) || # souhlasi potvrzeni hesla ?
            (raise ClientException.new('Password and Confirm password do not match.'))
          c = User.register( email, pass) # registrace uzivatele
          if c.save # pokud o ulozeni
                self.user = c # prihlaseni uzivatele

          end
          redirect_to '/'
        rescue Exception => ex
          @error = ex.getMessage
        end
```

```
        end
    end
    layout :register_erb # vykresleni registracniho formulare
end
```

Výpis 4: Část controlleru UserController s metodami pro registraci a přihlášení

6.8 Výpočet statistiky

Velmi důležitá část samotné aplikace jsou grafické statistiky zobrazující spotřebu a náklady. Tyto statistiky jde časově omezit (obrázek 16) a jsou rozděleny na dvě části. První část se zabývá pouze spotřebou paliva. Druhá část zobrazuje poměr všech nákladů za dané časové období.



Obrázek 15: Filtr pro vybrání období statistiky

Pro výpočet spotřeby a ceny paliva za měsíc slouží třída Consumption. Výřez z této třídy je ve výpisu 5. U metody get_fuelings jde vidět složitá selekce dat podle času. Je to tím, že v současné době umožňuje AppEngine omezení dotazu na databázi pouze jedním operátorem.

```
def get_fuelings # vraci tankovani v zadanem casovem rozsahu
  q = Fueling.all.vehicle_id(@vehicle.id).type(Fueling.TYPE_FUELING).sort(:date) # vybere
      pouze tankovani

  # apply range
  if @from_ms > 0
    q._query.addFilter(
            :date,
            Query.FilterOperator.GREATER_THAN_OR_EQUAL,
            Long.new(@from_ms) )
  elsif @to_ms > 0 and @from_ms == 0
    q._query.addFilter(
            :date,
            Query.FilterOperator.LESS_THAN_OR_EQUAL,
            Long.new(@to_ms) )
  end

  results = q.run # spusteni dotazu
  fuelings = ArrayList.new
```

```
    results.each do |f|
      if @to_ms == 0 or f.date <= @to_ms
        fuelings.add(f)
      end
    end

    # prepare fuelings for calc
    partly = Fueling.blank # docasna promenna pro neuplne tankovani
    full = ArrayList.new # vytvoreni seznamu pro uplne tankovani

    fuelings.each do |_f|
      f = Fueling(_f)
      if f.fueling_type == Fueling.FUELING_TYPE_FULL
        f.merge(partly) # pridani neuplnych tankovani
        f.consumption = (f.quantity / f.trip)*100 # spocitani spotreby
        f.save
        full.add(f) # pridani do seznamu tankovani
        partly = Fueling.blank # vynulovani docasneho tankovani
      elsif f.fueling_type == Fueling.FUELING_TYPE_PARTLY_FULL
        partly.merge(f) # pripocita do dalsiho tankovani
      elsif f.fueling_type == Fueling.FUELING_TYPE_INVALID
        if f.try_fix # pokus o opravu
          partly.merge(f) # pripocita do dalsiho tankovani
        end
      end
    end # fuelings

    full
  end

  def get_consumption_by_months
    fuelings = get_fuelings # nacte tankovani

    months = ArrayHelper.group(fuelings, :month) # group podle mesicu
    cons_months = MyHashMap.new

    months.keys.each { |k| # prochazi mesice
      sum=0.0; price = 0.0;
      ArrayList(months.get(k)).each { |_f| f = Fueling(_f) # tankovani v mesici
        sum += f.consumption
        price += f.price
      }
      it = MyHashMap.new
      it.put('consumption', sum / ArrayList(months.get(k)).size)
      it.put('price', price)
      cons_months.put(k,it)
    }
    cons_months # vraci pole obsahujici jednotlive mesice a celkovou cenu cepovani v danem
      mesici a spotrebu

  end
```

Výpis 5: Část třídy Consumption počítající průměrnou spotřebu

6.9 API

Aplikace obsahuje API umožňující prohlížení a zadávání dat pro aplikace třetích stran. API funguje jako služba REST, která zadává dotazy pomocí protokolu HTTPS. Všechny odpovědi jsou ve formátu JSON. Pro autorizaci musí cizí aplikace znát jméno a heslo uživatele. Po prvotní autorizaci je uživateli přidělena relace. Kód relace je 64-místné náhodně vygenerované číslo. Pro veškerou další komunikaci je vyžadován tento kód. Veškeré dotazy na API jsou zabezpečené, protože veškerá komunikace včetně prvotního dotazu na kód relace, kdy se zasílá jméno a heslo, probíhá po protokolu HTTPS.

6.10 Počáteční data

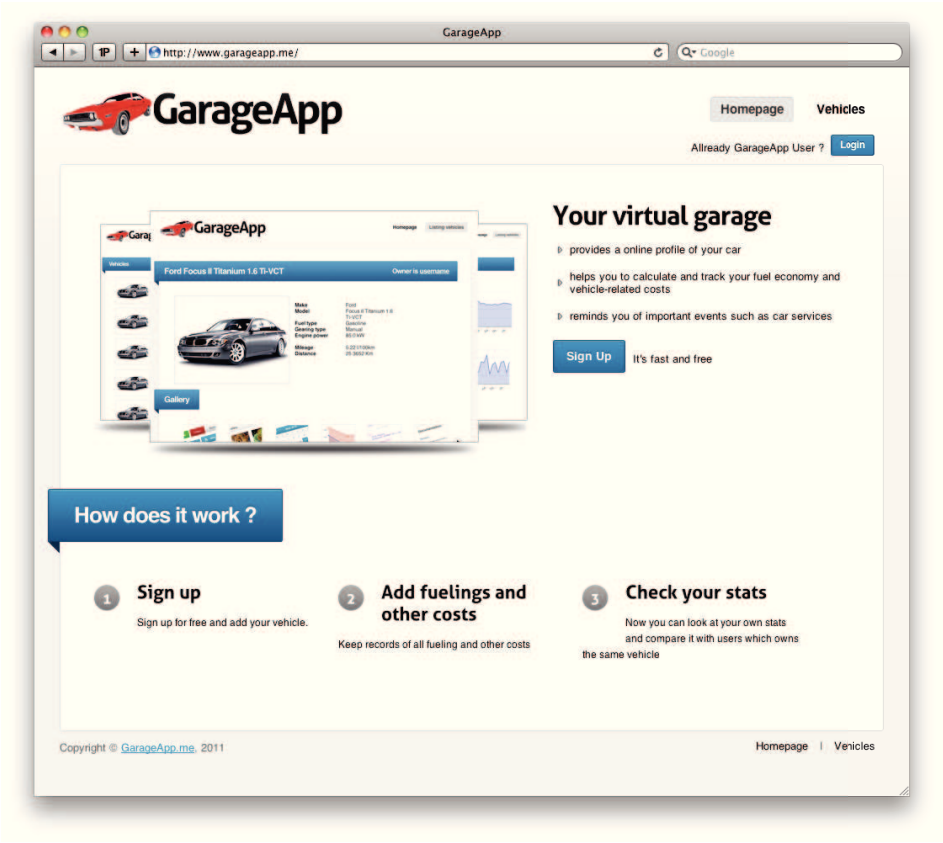
Pro samotné otestování funkčnosti systému je třeba aby systém obsahoval nějaké reálné data. Samozřejmě jsem schopen vygenerovat náhodná data a ty zobrazit v grafech, ty se však mohou od reálných dat velmi lišit. Pro tyto účely jsem vytvořil importovací skript z jedné z největších konkurenčních služeb. Skript je velmi flexibilní a stačí mu říct pouze url nebo případně id profilu vozidla, které chceme importovat a on se postará o všechno ostatní jako je vytvoření vozidla a přidání veškerých nákladů, včetně jejich rozlišení a správné zařazení. Nejvýhodnější tedy je nechat spouštět skript službou Cron, kde je mimo automatické spuštění i delší čas provádění. [8]

6.11 Implementace grafického designu

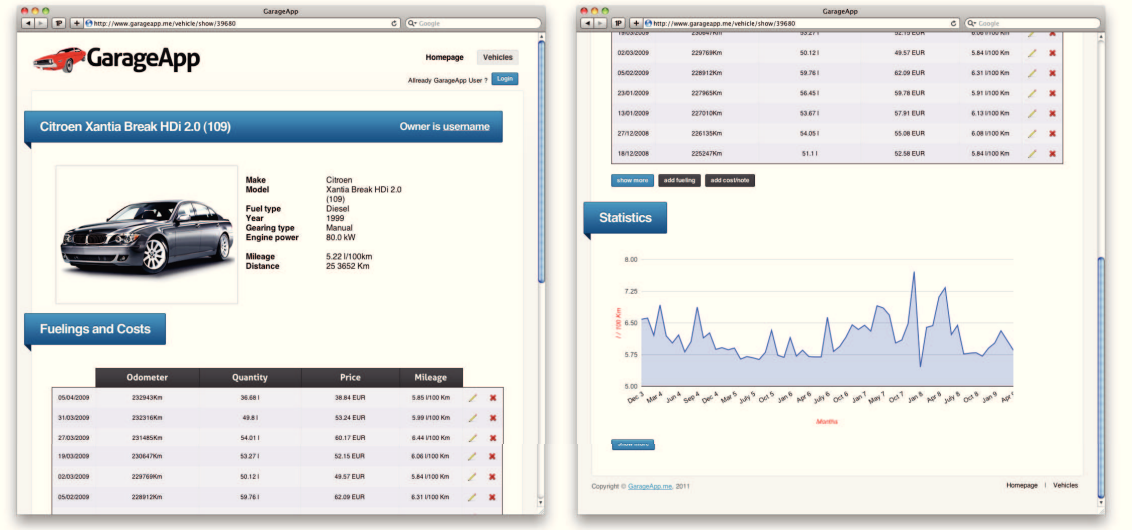
Z wireframe diagramů navrhovaných v analýze grafik nakreslí design. U těchto obrázků poté dojde k nastříhání a umístění do stránky. Celou aplikaci jsem nakódoval podle posledních trendů v HTML5 a CSS3. V prohlížečích, které ještě HTML5 a CSS3 plně nepodporují, se většina funkcí zobrazí v omezenější formě, ale velmi podobně.

Úvodní stránka je na obrázku 17. Na obrázku 18 můžeme vidět profil vozidla a na obrázku 19 statistiky.

[8] Čas provádění (Execution time) je u klasického požadavku (requestu) 30s, u služby Cron to je 90s



Obrázek 16: Úvodní stránka aplikace



Obrázek 17: Profil vozidla

Obrázek 18: Statistiky vozidla

6.12 Ostré nasazení na AppEngine

Není nic snadnějšího než nahrát výsledný projekt na AppEngine. V konfiguračním souboru app.yaml je nutné nastavit id aplikace, které jsme se volí při přidání nového projektu. Projekt poté běží na adrese id_projektu.appspot.com. Ve výpisu 6 je jediný příkaz, který provede po zadání jména a hesla kompletní publikování webu. Pokud později chceme vytvořit novou verzi v app.yaml ji změníme. Po publikování je nová verze dostupná na dočasné doméně, kde je možno si zkusit, zda je opravdu v pořádku. Poté můžeme v administraci nastavit novou verzi jako produkční.

```
cd cesta_k_projektu/projekt
rake publish
```

Výpis 6: Publikování projektu na ostrý server

7 Závěr

Cílem práce bylo vypracovat informační systém, který bude sloužit k evidenci nákladů vozidel. Systém bude umožňovat zadávat spotřebu a jiné náklady na provoz vozidla a vypočítávat statistiky z těchto dat. Nejprve jsem měl provést analýzu konkurenčních služeb a definovat požadavky na nový systém. Poté vypracovat analýzu informačního systému. Na základě analýzy navrhnout a implementovat daný systém v jazyce Mirah na platformě Google AppEngine.

Analyzoval jsem konkurenční služby, kde jsem se zaměřil zejména na motivaci uživatelů služby používat. Jednotlivé služby jsem na závěr porovnal. Podle konkurenčních služeb jsem navrhl vlastní informační systém. Realizaci jsem provedl v jazyku Mirah na platformě Google AppEngine. Na závěr jsem projekt publikoval.

V budoucnu je možné systém rozšířit o další statistiky. Při použití API se mohou dopsat aplikace pro další zařízení, které budou uživatelům zjednodušovat prohlížení a vkládání. Další z možností rozšíření může být aplikaci více socializovat a propojit se známými sociálními službami.

8 Reference

[1] Google, *Google App Engine* [online]. 2008 Dostupné z WWW: http://code.google.com/appengine/

[2] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber *Bigtable: A Distributed Storage System for Structured Data* [online]. 2006 Dostupné z WWW: http://labs.google.com/papers/bigtable.html

[3] Ryan Brown, Charles Oliver Nutter, *Mirah* [online]. 2009 Dostupné z WWW: http://www.mirah.org

[4] Nick Howard, *Dubious for App Engine* [online]. 2009 Dostupné z WWW: https://github.com/mirah/dubious

[5] Charles Oliver Nutter, Thomas Enebo, Nick Sieger, Ola Bini, *JRuby* [online]. 2006 Dostupné z WWW: http://www.jruby.org/

[6] *Spritmonitor* [online]. 2001 Dostupné z WWW: http://www.spritmonitor.de/en

[7] *Spotřeby* [online]. 2008 Dostupné z WWW: http://www.spotreby.cz

[8] *MPG Tune.com* [online]. 2008 Dostupné z WWW: http://mpgtune.com/

[9] *fuelfrog* [online]. 2010 Dostupné z WWW: http://fuelfrog.com/

[10] *Fuelly* [online]. 2008 Dostupné z WWW: http://www.fuelly.com/

[11] *Road Trip for iPhone and iPod touch* [online]. 2008 Dostupné z WWW: http://darrensoft.ca/roadtrip/

[12] *Kurzy.cz - Vývoj cen benzínu a nafty* [online]. 2000 Dostupné z WWW: http://www.kurzy.cz/

9 Přílohy

A. Disk CD

- **application** - Zdrojové kódy aplikace. Struktura popsána v kapitole 6.3.
- **diagramy** - Obsahuje kompletní Use Case diagram.
- **docs** - Bakalářská práce ve formátu pdf.