Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Moderní uživatelské rozhraní projektu Metrocar

Vojtěch Knaisl

Vedoucí práce: Ing. Martin Komárek

11. května 2015

Poděkování

Děkuji panu Ing. Martinu Komárkovi za cennou zpětnou vazbu, kterou mi během tvorby této práce poskytoval, a také za vedení mé práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 11. května 2015 .....

Odkaz na tuto práci

Knaisl, Vojtěch. *Moderní uživatelské rozhraní projektu Metrocar*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato práce navazuje na již existující projekt webové aplikace Metrocar, která slouží k jednodušší a pohodlnější správě carsharingové společnosti. Práce stručně shrnuje, co již bylo na projektu vytvořeno. Dále se zabývá důvody, proč bylo třeba vytvořit novou klientskou aplikaci a proč byla aplikace vytvořena dle Single page application přístupu. V práci je též řešen návrh nového REST API, které má komunikovat s klientskou aplikací. Zvolené řešení pak analyzuje a implementuje. Výsledkem práce je fungující klientská aplikace, které spolupracuje se serverem pomocí nově vytvořeného REST API.

Klíčová slova Carsharing, Single page aplikace, Ember.js, REST API, Django

Abstract

This thesis continues on development of an information system for carsharing company - Metrocar. Text briefly summarizes a history of project and discusses the reasons why it had to be done the new client application and why as a Single page application. This thesis also deals with purpouse of new REST API which will be communicated with client application. Then the chosen solution is analyzed and implemented. Result of this thesis is fully functional

client application which communicates with server through new created REST API.

Keywords Carsharing, Single page application, Ember.js, REST API, Django

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Jako student nevlastnící auto stojím někdy před problémem, že potřebuji na hodinu či dvě půjčit auto. Zpravidla jsem pak nucený požádat kamaráda či rodiče, aby mi auto na danou chvíli zapůjčili. Myšlenka carsharingu mezi kamarády či v rámci nějaké komerční společnosti mi tak je velmi blízká, vyřešila by totiž mé problémy. Po krátkém průzkumu, jak na tom český trh je, jsem zjistil, že u nás tato služba není příliš vyvinuta. Prací na této bakalářské práci chci tedy přispět k rozvinutí již existující platformy, a přispět tak k jejímu možnému nasazení v budoucnu.

Díky nefunkčnosti a značné zastaralosti klientské aplikace je cílem práce implementace nové, moderní aplikace, která by komunikovala se zákazníky. K napojení na stávající řešení pak bude v serverové aplikace vytvořeno nové REST API.

Práce začíná stručným popisem, co přesně carsharing je, a shrnuje historii projektu Metrocar, jehož součástí tato práce je. Dále autor popisuje zkušenosti s aplikací a s objevením chyb, které musely být opraveny před dalším vývojem aplikace. V další kapitole je pak shrnut aktuální stav a jsou zde vytyčeny cíle práce, tedy vytvoření nového REST API a klientské aplikace. Práce dále diskutuje technologie, které by bylo vhodné použít, a zvolené řešení porovnává s možnými alternativami. Pátá kapitola je pak věnována analýze. Šestá kapitola popisuje návrh aplikace, její strukturu a rozbor jejích dílčích částí, jako například popis procesu validaci dat, autentizaci a autorizaci uživatelů, či lokalizace aplikace. V posledních dvou kapitolách se pak autor věnuje nasazení a testování a zmiňuje možné návrhy řešení, jak tento proces automatizovat.

Kapitola 1

Carsharing, historie projektu

1.1 Carsharing

Carsharing[1] je služba, která má za cíl nabídnout alternativu lidem, kterým se nevyplatí vlastnit auto, ale přesto by jeho služeb rádi někdy využili. Staví na principu sdílení aut mezi více lidmi, a tedy i lepší vytíženosti auta. Nabízí také možnost vybírat z různých modelů aut dle aktuálních potřeb.

Výhody tohoto řešení jsou vcelku jasné. Díky vetší efektivitě využití auta je možno rozpočítat fixní náklady na provoz auta (např. povinné ručení, pravidelná údržba...) na více lidí. Tato myšlenka částečně nabízí řešení i pro problém s parkováním aut ve velkých městech. Vzhledem k tomu, že se o auto dělí více lidí, nemusí ve městě být tolik parkovacích míst. Výhodou pro zákazníka může být i to, že se nemusí starat o papírování související se správou vlastního auta (např. registrace auta, zařízení povinného ručení...).

1.2 Historie projektu

Projekt Metrocar už má za sebou několik let vývoje. Celé to začalo v roce 2009, kdy byla na uvedené téma napsána první bakalářská práce. Ondřej Nebeský[2] se zde zabýval zejména návrhem webového rozhraní pro správu rezervací aut. Celý projekt byl napsaný v jazyce PHP[3] a na platformě Drupal[4]. Toto řešení se ale nakonec neukázalo jako optimální. Pro pokračování byla tedy zvolena vývojová platforma Python[5] a webový framework Django[6]. Tato kombinace se ukázala jako více než vhodná pro tento projekt a je použita až dodnes. Transformaci provedl Filip Vařecha[7] ve své diplomové práci.

Dále v projektu pokračoval Jan Wágner[8], který ve své práci dokončil modul pro rezervaci a modul pro správu uživatelů. Připravoval také aplikaci na reálné nasazení. Modul pro lepší zpracování geografických dat pak ve své práci obsáhl Petr Pokorný[9]. Ten také do projektu vnesl pořádek. Významného vylepšení se dostalo dokumentaci a byly také aktualizované závislosti projektu. Poslední, kdo se systémem Metrocar v závěrečné práci zabýval, byl

Jakub Ječmínek[10]. Ten zkoumal možnosti napojení aplikace na účetní systém Flexibee[11].

Kapitola 2

Seznámení s projektem, bugfixing

2.1 První zkušenosti

Když jsem se připojil k projektu Metrocar, první, co mě čekalo, bylo ozkoušení aplikace. Ze začátku jsem se zaměřil na prozkoumání aplikace, která už byla nasazená na ostrém serveru. Pro hlubší pochopení, co již bylo na projektu uděláno, jsem dále přečetl závěrečné práce mých předchůdců. Následně jsem se snažil rozchodit instanci aplikace u sebe na lokálním počítači.

2.2 Bugfixing

2.2.1 Neaktuální balíčky

Zde se vyskytly první problémy. Jelikož byl projekt rok a půl nevyvíjen, tak zastaral. Stalo se, že některé požadované verze balíčků pro chod projektu už na centrálním úložišti pythonovského správce balíčků nebyly k dispozici. Jelikož na neaktuálnost balíčků upozorňoval již Petr Pokorný[9] ve své diplomové práci, rozhodlo se, že bude proveden upgrade. Největší problém byl v povýšení verze frameworku Django. Některé metody, funkce a třídy totiž byly v nové verzi odebrány, a tak se musel řešit první zásah do kódu. Nakonec se však vše podařilo vyřešit, opravit a aplikaci spustit.

2.2.2 Nefunkční migrace databáze

Další problém se ale naskytl hned záhy. Nešla totiž postavit databáze. Po bližším zkoumání se zjistilo, že mezi jednotlivými migracemi existuje cyklická závislost. Přesněji existovaly dvě iniciální migrace pro moduly Reservations a Cars, kde každá potřebovala navzájem tu druhou. Rozhodlo se tedy, že iniciální migrace v modulu Reservations bude rozdělena na dvě migrace. Dále bylo

také potřeba nastavit vzájemné závislosti migracím, aby byly vykonávané v správném pořadí.

Kapitola 3

Cíl práce

3.1 REST API

3.1.1 Aktuální stav

Po seznámení se s aplikací a opravení chyb následovalo hlubší prozkoumání struktury projektu. Vzhledem k tomu, že cílem bakalářské práce je vytvořit novou klientskou aplikaci odpovídající Single page application přístupu, první, co bylo podrobeno zkoumání, bylo REST API, které aplikace poskytuje. Při bližším sledování se zjistilo, že aktuálně by měla serverová aplikace poskytovat API dalším dvěma aplikací.

První aplikace, která těchto služeb využívá, je aplikace pro operační systém Android. Ta slouží k odesílání sbíraných dat získaných z palubní jednotky automobilu na server. Konkrétní služby, které aplikace využívá jsou:

- zápis dat získaných z palubní jednotky,
- čtení aktuálních rezervací pro daný automobil.

Implementaci těchto služeb vytvořil Petr Pokorný[9] ve své diplomové práci. Na tomto API nebyly potřeba žádné změny.

Druhá aplikace, která měla REST rozhraní serverové aplikace využívat, byla klientská aplikace. S ohledem na to, že uvedené rozhraní nebylo zcela dodělané, nebylo ve výsledku vůbec použito. Jeho hlavní nevýhodou bylo, že povětšinou poskytovalo jen služby pro čtení, ale už ne pro modifikaci dat.

Vzhledem k tomu, že rozhraní pro novou klientskou aplikaci bylo vytvořeno v jiné době než rozhraní pro mobilní aplikaci, bylo každé napsáno za použití jiných nástrojů. REST rozhraní pro klientskou aplikaci bylo napsáno s pomocí knihovny Django Piston[12], která už v době tvorby bakalářské práce nebyla vyvíjena. Rozhraní pro mobilní aplikaci pak bylo napsáno čistě s použitím nástrojů dostupných v použitém webovém frameworku Djangu.

3.1.2 Plán

Vzhledem k nutnosti mít plně funkční REST rozhraní pro klientskou aplikaci se nabízely dvě možnosti řešení této situace. První byla doplnit funkčnost do stávajícího řešení. Toto řešení ale bylo záhy zamítnuto. Důvodem byla již zmíněná, delší dobu nevyvíjená knihovna Django Piston, která zde byla použita. Druhou možností bylo celé toto API přepsat od základu. Toto řešení bylo nakonec vybráno jako finální. První úkol tedy byl:

1. vybrat novou knihovna, která zde bude použita,
2. navrhnout možnosti, jak bude staré REST API transformováno do nového řešení,
3. toto řešení následně implementovat a otestovat.

3.2 Klientská aplikace

3.2.1 Aktuální řešení

Jak už bylo řečeno v předešlých kapitolách, aktuální stav klientské aplikace nebyl ideální. Klientská aplikace obsahovala spoustu chyb, které byly zapříčiněny rychlým vývojem této aplikace. Ta měla původně sloužit pouze jako ukázka aktuálních možností a funkcí, které projekt Metrocar nabízí.

Přestože byl projekt Metrocar rozdělen na více aplikací (serverová aplikace poskytující ostatním data, klientská aplikace komunikující se zákazníkem a mobilní aplikace komunikující s palubní jednotkou automobilu), nezabránilo to na straně klientské aplikace duplikaci kódu. Vzhledem k tomu, že klientská aplikace nepoužívala API, co poskytovala serverová aplikace, ale přímo si importovala či kopírovala kusy kódu ze serverové aplikace, došlo zde k tomu, že se serverová aplikace začala obcházet. Celá tato situace vyústila až do absurdního stádia, kdy část funkcionality byla lépe implementována na klientu a vzhledem k „zpětné kompatibilitě“ serverové aplikace si zase serverová aplikace začala přímo importovat některé věci z klientské aplikace. Aplikace si tak na sobě vytvořily kruhovou závislost.

3.2.2 Plán

Toto řešení tedy bylo absolutně nevyhovující pro budoucí rozvoj projektu, a bylo tedy rozhodnuto, že bude přepracováno. Serverová část aplikace bude zbavena závislosti na staré klientské aplikaci, která bude následně odstraněna.

Vzhledem k aktuálním trendům bylo zvoleno, že klientská aplikace bude napsána ve stylu odpovídající Single page application přístupu. Pro komunikaci se serverovou aplikací bude sloužit přepracované REST rozhraní.

Kapitola 4

Použité technologie

4.1 Knihovna Django REST Framework

Jako nástroj usnadňující vytvoření REST API byl zvolen Django REST Framework[13]. Tato knihovna se po dobu své existence stala nepsaným standardem pro tvoření REST API nad webovým frameworkem Django.

Knihovna umožňuje jednoduché vytvoření množiny pohledů pro model, která reprezentují operace, jež lze přes API s modelem dělat. Množina pohledů je zde napevno svázána s modelem a se serializerem. Vše je tvořeno podle paradigmatu Convention over configuration[14], takže pro základní konfigurace a zprovoznění stačí vytvořit příslušnou třídu, která zaštiťuje množinu pohledů, a v ní definovat jméno modelu a serialezeru. Webový framework Django se ale dal opačnou cestou (Configuration over convention[15]), a tak je nutné tuto třídu s množinou pohledů ještě zaregistrovat v URL konfiguraci Django aplikace.

Tento nástroj v sobě obsahuje také vestavěnou podporu pro autentizaci, autorizaci a pro řízení přístupu k datům pomocí oprávnění. Sám framework pár základních a hojně používaných definuje, ale není problém si napsat vlastní.

Knihovna také umožňuje uživateli filtrovat či měnit standardní kolekci objektů, která se běžně získává z databáze. To se může hodit například, když chcete na zdroji rezervace vrátit uživateli jen rezervace týkající se jeho samotného a ne rezervace pro všechny zákazníky.

Pro potřeby validace dat přicházejících z požadavků na server lze definovat validátory. Avšak vzhledem k možné duplikaci kódu s validacemi na modelu je jejich využití značně omezené. S validacemi souvisí též vyhazování výjimek. Pro jejich zachycení a transformaci lze jednoduše nastavit příslušný handler.

Jak je vidět, tato knihovna obsahuje spoustu nástrojů pro usnadnění tvorby API dle architektury REST. K tomu přidává ještě nástroje pro testování, a tvoří tak celek, který většině vývojářů bohatě dostačuje. Vzhledem k oblíbenosti také nelze vyloučit, že projekt skončí podobně jako projekt pro migrace v Djangu - South[16]. Tedy jeho začleněním do standardní Django distribuce.

4.2 Single page aplikace

4.2.1 Definice dle Mikowskiho a Powella

Single page aplikace je definovaná podle Mikovskiho a Powella jako aplikace spuštěná v prohlížeči, která se během svého používání pokaždé znovu nenačítá. Single page aplikaci si tedy lze představit jako tlustého klienta, který je načten z web serveru.[17](vlastní překlad)

4.2.2 Historie

Single page aplikace jsou zde s námi již řadu let. Technologie, které se k tomu dříve používaly, jsou Adobe Flash[18] a Java Applet[19]. K jejich spuštění ale bylo potřeba nainstalovat do prohlížeče speciální pluginy, abychom mohli tyto aplikace spustit. Nevýhodou těchto řešení také byl vznikající overhead, neboť aplikace se musela spouštět v pluginu prohlížeče a ne přímo nativně v prohlížeči. S tím souvisí také zvýšení hrozby, že kromě chyby v prohlížeči může bezpečnost naší aplikace ještě ohrozit chyba v překladači třetí strany či v již zmíněném pluginu.

V posledních pár letech ale nastal rozmach psaní Single page aplikací v javascriptu. Postupně začaly vznikat celé frameworky, které psaní těchto aplikací dosti usnadňují. Z těch nejvíc populárních lze zmínit například Googlem tvořený Angular[20]. Z trošku jiného úhlu pohledu se k tomu dnes populárnímu proudu připojil i Facebook, který vytvořil React[21]. Z těch nejznámějších zbývá ještě vyjmenovat komunitou vyvíjený Ember[22] a Backbone[23].

4.2.3 Výhody Single page aplikace napsané v javascriptu

- Prohlížeč je jeden z nejvíce používaných prostředí pro spuštění vaší aplikace.
- Nasazení takovéto aplikace je jednoduché, stačí aplikaci umístit na webový server.
- „Jedna aplikace pro všechny platformy a operační systémy. “ Vzhledem k tomu, že prohlížeč s podporou javascriptu je dnes, dá se říct, standardem, je možné aplikaci spustit jak na desktopovém počítači, tak třeba na tabletu či mobilu. Díky dostupnosti prohlížeče ve všech známých operačních systémech pak nezáleží, zda máte na počítači nainstalované Windows, Linux či OS X. Standardně lze aplikaci spustit i v tabletu či mobilu. Zde se nabízí i možnost portace aplikace do samostatně spustitelné aplikace. To umožňuje větší integraci aplikace do systému - distribuce přes obchod aplikací, používání systémových notifikací atd.
- Javascript se stal také v posledních letech díky soupeření Googlu, Mozzilly, Microsoftu a dalších velmi rychlým jazykem. Jeho implementace

v prohlížeči tak má k dispozici velmi pokročilé techniky, jako například spouštění více vláken v aplikaci, předpovídání skoků nebo třeba JIT kompilace do nativního kódu.

4.2.4 Nevýhody Single page aplikace napsané v javascriptu

- Vzhledem k tomu, že aplikace se spouští na straně klienta, narostou nám tím nároky na klientský počítač. To je úměrné zaměření aplikace. Pokud například vytvoříme aplikaci, která před odesláním dat na server data zašifruje, je jasné, že část výkonu počítače nám to dozajista sebere. S tím souvisí také fakt, že pokud jsme na notebooku, tabletu či mobilu, tak nám může při používání takovéto aplikace klesnout výdrž baterie.

- Nevýhodou také někdy může být skutečnost, máme-li aplikaci rozdělenou na klient-server a se serverem komunikujeme přes API, které je vystaveno na internetu. Problémem se totiž mohou stát validace. Ty totiž musíme zduplikovat - resp. mít je jak v klientské aplikaci, tak na serveru. Samozřejmě pokud se rozhodneme, že jediné ukládání dat bude na serveru, tak na klientu všechny validace implementovat nemusíme, ale pokud chceme v klientské aplikaci používat navíc například databázi (př. IndexedDB[24]), duplikaci validací se bohužel nevyhneme.

4.2.5 Výběr vhodného frameworku

Jak jsem psal výše, aktuálně existují na trhu čtyři javascriptové frameworky, které pomáhají s tvorbou Single page aplikace - Angular, Ember, React a Backbone.

Prvním měřítkem výběru byla oblíbenost v komunitě a celkový počet doplňků pro daný framework.

Tabulka 4.1: Porovnání javascriptových frameworků pro tvorbu Single page aplikací podle oblíbenosti (stav k 17.4.2015)

Framework	**Oblíbenost na githubu (počet hvězdiček)**	**Počet doplňků**
Angular	37 543	1 353
Ember	13 414	898
Bacbone	21 431	249
React	20 394	Neexistuje centrální repositář

Jak je vidět v tabulce, nejoblíbenější framework mezi vývojáři je Angular, na který existuje i nejvíce doplňků. Velkou nevýhodu v použití má pak

React, na který neexistuje jednotný systém instalování doplňků přes centrální repositář.

Dalším měřítkem pro vybrání vhodného frameworku bylo, zda obsahuje kompletní podporu od zobrazovací vrstvy až po modelovou. Zde nám odpadl React, který je zaměřený spíše na tvorbu webových komponent než na tvorbu ucelených složitějších aplikací. Chybí mu tak například podpora pro správu routování.

Při výběru byl také kladen důraz na pohodlí psaní aplikace, a tak po zjištění, že Backbone neobsahuje podporu pro tzv. Two way binding[25] a v základu v sobě nemá šablonovací systém, byl tento framework vyřazen. Ve výsledku nám tedy zbyl výběr pouze mezi dvěma frameworky - Angularem a Emberem. Po bližším zkoumání ale bylo zjištěno, že mezi těmito dvěma platformami není prakticky žádný podstatný rozdíl. Ember sice nabízí některé pokročilejší funkce, které ale vyvažuje Angular množstvím doplňků a velkou komunitou vývojářů. Rozhodnutí nakonec ovlivnil zejména fakt, že projekt je spíše dlouhodobého rázu. Vzhledem k tomu, že Angular bude nahrazen v roce 2016 novou verzí, která bude se starou zpětně nekompatibilní, je jasné, že komunita kolem starší verze bude postupně upadat. Jako finální řešení tedy byl vybrán nakonec Ember.

4.2.6 Ember

Ember nabízí vše, co by měl framework pro Single page aplikace obsahovat. Architektura je postavená na osvědčeném modelu MVC (Model-view-controller). Pro snazší tvorbu aplikací byl pak vytvořen Ember CLI[26], který nabízí sadu nástrojů pro založení projektu, správu závislostí projektu, vestavěný webserver pro vývoj a také nástroj pro sestavení projektu. Umožňuje vám také psát kód s pomocí nového standardu Ecmascript 6. Vzhledem ale k zpětné kompatibilitě se staršími prohlížeči je tento kód překládán do standardu Ecmascript 5.1.

Do projektu byl také integrován šablonovací systém Handlebars[27]. Ten usnadňuje vývojáři práci díky sadě helperů, například pro hromadný výpis dat pomocí `each` helperu. Data zobrazovaná v šabloně mohou být navázána na model, a lze tedy jednoduše tvořit formuláře a následně jejich data ukládat prostřednictvím adapterů na vzdálený server. O správu ukládání dat se starají v aplikaci Ember Data. Ta jsou vytvořené podle návrhového vzoru Active Record.

Vzhledem k tomu, že se v Emberu dají tvořit i opravdu velké a složité aplikace, byla přidána podpora pro Dependency Injection. Lze tedy jednoduše spravovat závislosti mezi třídami. Ty jsou také jedním z vylepšeních, které byly do Emberu přidány. Jedná se spíš ale o syntaktické pozlátko, které umožňuje uživateli, který je zvyklý z jazyků obsahující třídní systém, si na používání javascriptu lépe zvyknout.

4.2.7 Coffeescript

Javascript je sám o sobě mocný jazyk. Díky své syntaxi ale bohužel přehlednost kódu při některých jazykových konstrukcích značně trpí. Proto tedy vznikají různé nadstavby, které vývojářům dávají k dispozici různé syntaktické cukříky. Nejznámějším zástupce těchto vylepšení je CoffeeScript[28]. Jeho syntax se značně podobá Pythonu. Nenutí vás ohraničovat bloky závorkami, stačí jen odsadit blok pomocí tabulátoru. Zaměřuje se také na psaní kódu tak, aby byl čitelný jako věta v klasickém jazyce. Díky tomu a dalším mnoha vylepšením se dostal do hledáčku společností, které mají značnou část frontendu napsaného v Javascriptu. Jako příklad společností, které přepsaly své aplikace z Javascriptu do CoffeeScriptu lze jmenovat například Github nebo třeba Dropbox[29].

Listing 4.1 : Porovnání syntaxe CoffeeScriptu a Javascriptu

```
# CoffeeScript

show picture if checkbox is on

# Javascript

if (checkbox == true) {
    show(picture)
}
```

4.2.8 Emblem

Jako CoffeeScript ulehčuje práci se psaním javascriptového kódu, tak Emblem[30] pomáhá zrychlit a zjednodušit psaní šablon v Handlebars. Opět se snaží redukovat množství napsaného textu, a tak odstraňuje menšítka a většítka, která obalují HTML tag. Opět jako například v CoffeeScriptu nebo v Python odsazujete obsah pomocí tabulátorů. Nemusíte tedy psát ani ukončovací HTML tagy.

Listing 4.2 : Porovnání syntaxe Handlebars a Emblemu

```
# Emblem

div#content-frame: div.container: div.row:
  div.span4: render "sidebar"
  div.span8: render "main"

# Handlebars

<div id="content-frame">
  <div class="container">
    <div class="row">
      <div class="span4">
        {{render "sidebar"}}
      </div>
      <div class="span8">
        {{render "main"}}
      </div>
    </div>
  </div>
</div>
```

Opět se jedná pouze o nadstavbu. Stačí tedy při kompilaci šablon přidat navíc kompilaci z Emblemu do Handlebars.

4.2.9 Stojí tato vylepšení opravdu za to?

Je jasné, že všechna tato vylepšení nám nepomohou v psaní lepších a rychlejších aplikací. Někdy by se dalo i polemizovat, zda nejsou aplikace dokonce pomalejší. Nemáme totiž kontrolu nad automaticky generovaným kódem, tudíž ho nemáme jak optimalizovat. Trochu problém může také nastat u ladění, kde už se bohužel bez studování vygenerovaného javascriptového kódu neobejdeme.

Kapitola 5

Analýza

5.1 Úvod

Ještě než začneme programovat, je dobré provést analýzu. Vzhledem k tomu ale, že k tématu projektu bylo vypracováno několik bakalářských a diplomových prací (viz kapitola 1.2), není potřeba vše dělat úplně od začátku. Například identifikování domén a vytvoření diagramu tříd bylo uděláno již mými předchůdci na projektu. Pro serverovou aplikaci pak už byly vyhledány i případy užití a vytvořen model požadavků.

5.1.1 Chybějící části dokumentace

Jeden z důvodů proč tvořit dokumentaci je, že díky její existenci se dá usnadnit proces adaptace nově příchozích lidí do projektu. Zde tomu nebylo výjimkou. Bohužel pro pochopení celé problematiky autorovi této práce chyběly některé diagramy. Například proces registrace uživatele není úplně triviální. Roli v něm hraje totiž kromě samotného uživatele i manažer. Proto je určitě dobré mít tento proces řádně zdokumentovaný.

Už při autorově připojení k projektu obsahoval projekt Metrocar čtyři aplikace - samotnou serverovou aplikaci Metrocar, geolokační aplikaci Geotrack, mobilní aplikaci pro palubní jednotku auta a klientskou aplikaci. Ty mezi sebou komunikují přes různá rozhraní a dohromady tvoří již vskutku komplexní systém, který by si určitě zasloužil zvláštní popis (diagram).

Z diagramu týkající se klientské aplikace bylo vzhledem ke kompletnímu přepracovaní nutné předělat model požadavků. Bylo třeba také pozměnit a doplnit diagramy užití.

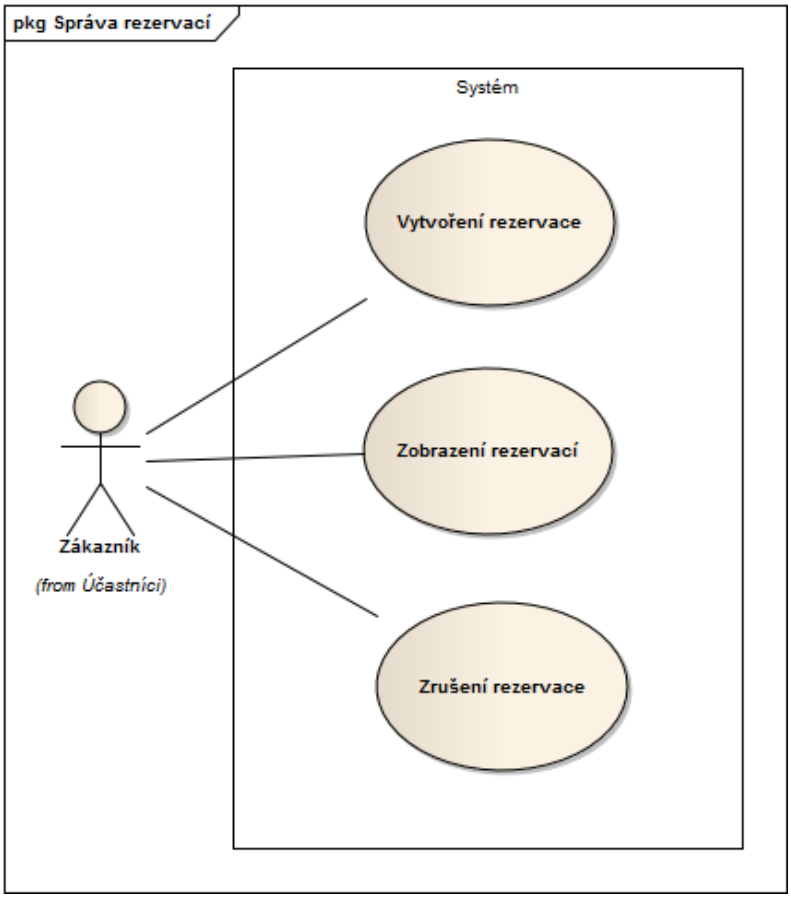
5.2 Případy užití

Vzhledem k tomu, že současná klientská aplikace vznikla jen jako ukázka části funkcí, které již byly implementovány v serverové aplikaci, nikdy nebyla po-

řádně provedena její analýza. Při tvoření diagramu užití se tedy začínalo skoro od začátku. Při analyzování toho, co by měla nová klientská aplikace zákazníkovi nabízet, se určily tři logické celky, do kterých se dají vyhledané případy užití rozdělit - správa rezervací, správa účtu a správa uživatelů. Jako aktéři byli identifikováni zákazník a manažer.

5.2.1 Správa rezervací

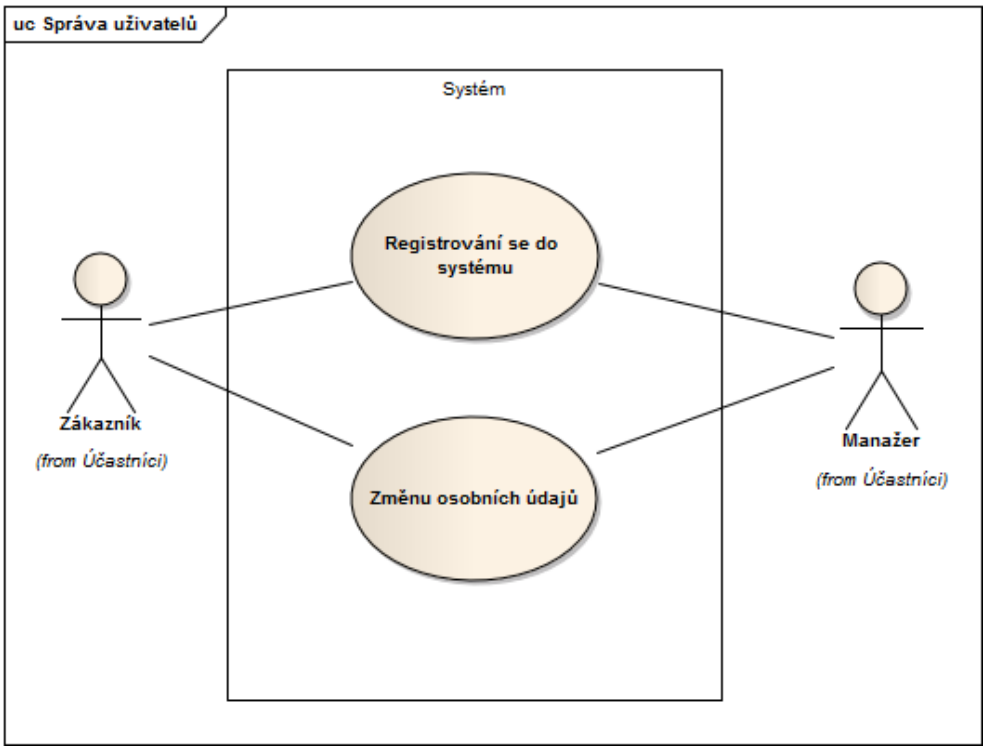
Pokud jde o možnosti, co by klientská aplikace měla nabízet, tak vše se točí hlavně kolem rezervací. Jejich problematiku a implementaci na straně serveru řešil Jan Wágner ve své bakalářské práci[8]. Z hlediska zákazníka se pak identifikovaly tři případy užití - vytvoření rezervace, zobrazení rezervace a možnost rezervaci zrušit. V žádném z těch případů nehraje roli manažer. Díky větší komplexivitě u vytvoření rezervace byl tento případ užití podrobněji rozepsán pomocí diagramu aktivit (viz kapitola 5.3.4)



Obrázek 5.1: Diagram případů užití - Správa rezeravací

5.2.2 Správa uživatelů

U správy uživatelů byly identifikovány dva případy užití - registrování do systému a požádání o změnu osobních údajů. Oba případy mají společné, že se v nich objevuje také role manažera. Ten je zapojen zejména do fáze schvalování. Jeho úkolem je ověřit zejména pravost a relevantnost údajů zadaných uživatelem.



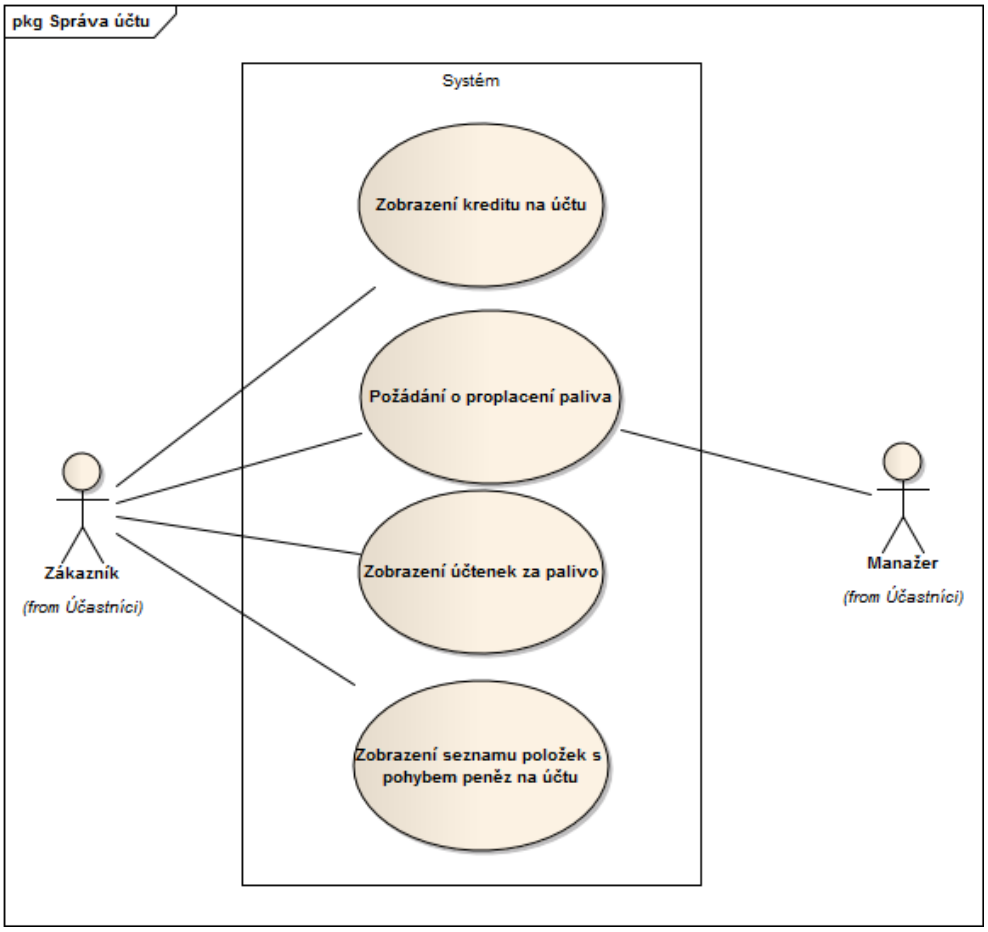
Obrázek 5.2: Diagram případů užití - Správa uživatelů

5.2.3 Správa účtu

Správa účtu byla, co se týká počtu identifikovaných případů, bohatší. Zákazník by měl mít možnost zobrazit aktuální kredit, který mu zbývá na účtu. Měl by mít také možnost podívat se do historie transakcí, které byly na jeho účtu zpracovány. To znamená, že by měl by mít možnost zobrazit si na jedné stránce jak připsané peníze za dobití kreditu a za proplacené žádosti o proplacení, tak odečtené peníze za vytvořené rezervace.

Vzhledem k tomu, že zákazník si auto může vyzvednout hned po jiném zákazníkovi, může se stát, že v autě nebude dost paliva na další jízdu. Carsharingová společnost proto svým klientům nabízí, že v této situaci můžou natankovat do auta benzín a následně požádat o proplacení zaplacené částky. Zákazník by měl tedy mít možnost v systému tuto žádost o proplacení paliva

vytvořit. Stav těchto žádostí a jejich historie by měla být pro zákazníka též dostupná.



Obrázek 5.3: Diagram případů užití - Správa účtu

5.3 Diagram aktivit

Díky netriviálnosti některých procesů se nabízí k jejich lepšímu pochopení vytvořit diagram aktivit.

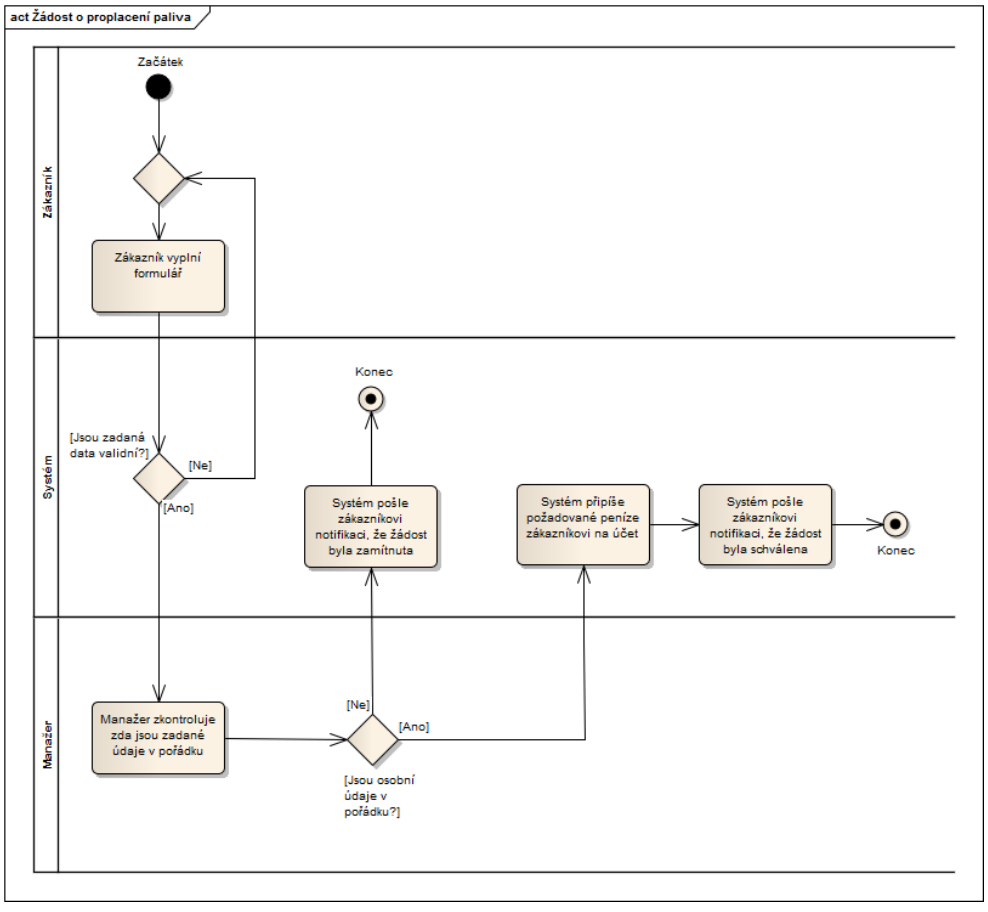
5.3.1 Žádost o proplacení paliva

Cílem toho procesu je vytvoření žádosti o proplacení paliva. Proces může skončit buď schválením žádosti, nebo její zamítnutím.

Proces začíná tím, že uživatel vyplní formulář s žádostí. Povinná políčka zde jsou:

- místo,
- typ paliva,
- počet litrů natankového paliva,
- cena,
- auto,
- fotka účtenky.

Systém dále zvaliduje, zda jsou data platná. Poté se přesune žádost k manažerovi. Ten zkontroluje, zda bylo palivo opravdu natankováno do zadaného automobilu a zda zadaná částka souhlasí s požadovanou částkou. Pokud je vše v pořádku, žádost schválí. Systém pak pošle uživateli notifikaci a připíše žadateli příslušnou částku na účet. Pokud ne, žádost je zamítnuta.



Obrázek 5.4: Diagram aktivity - Žádot o proplacení paliva

5.3.2 Registrace

Carsharingové společnosti si musejí být jisté, že pokud někomu půjčují auto, tak tento člověk skutečně existuje. Je to hlavně kvůli právní vymahatelnosti v případě problémů, jako například při dopravní nehodě.

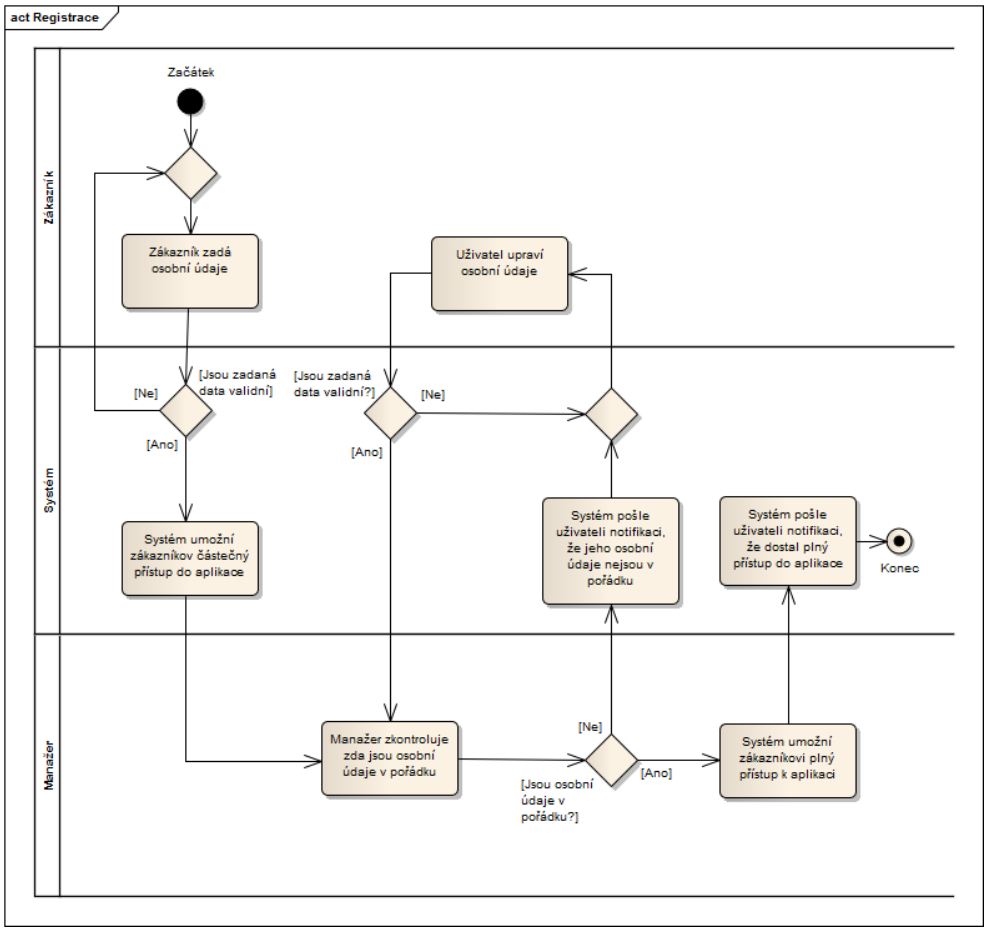
Pokud se chce zákazník registrovat, musí vyplnit formulář. Ten obsahuje tato povinná pole:

- jméno,
- příjmení,
- číslo občanského průkazu,
- číslo řidičského průkazu,
- telefonní číslo,
- email,
- datum narození,
- uživatelské jméno,
- heslo,
- fotka občanského průkazu,
- fotka řidičského průkazu.
- adresa,
- číslo popisné,
- PSČ,
- město.

Pokud je vše opět správně vyplněno, je žádost o registraci postoupena k manažerovi. Tímto krokem také získává zákazník částečný přístup do aplikace. Jediné, co je mu ale umožněno v aplikaci dělat, je změna svých osobních údajů.

Dále je na manažerovi, aby zkontroloval, zda souhlasí vyplněné údaje s údaji v občanském a řidičském průkazu. Pokud ano, systém pošle zákazníkovi notifikaci, že jeho registrace byla úspěšně schválena a že získal plný přístup do aplikace.

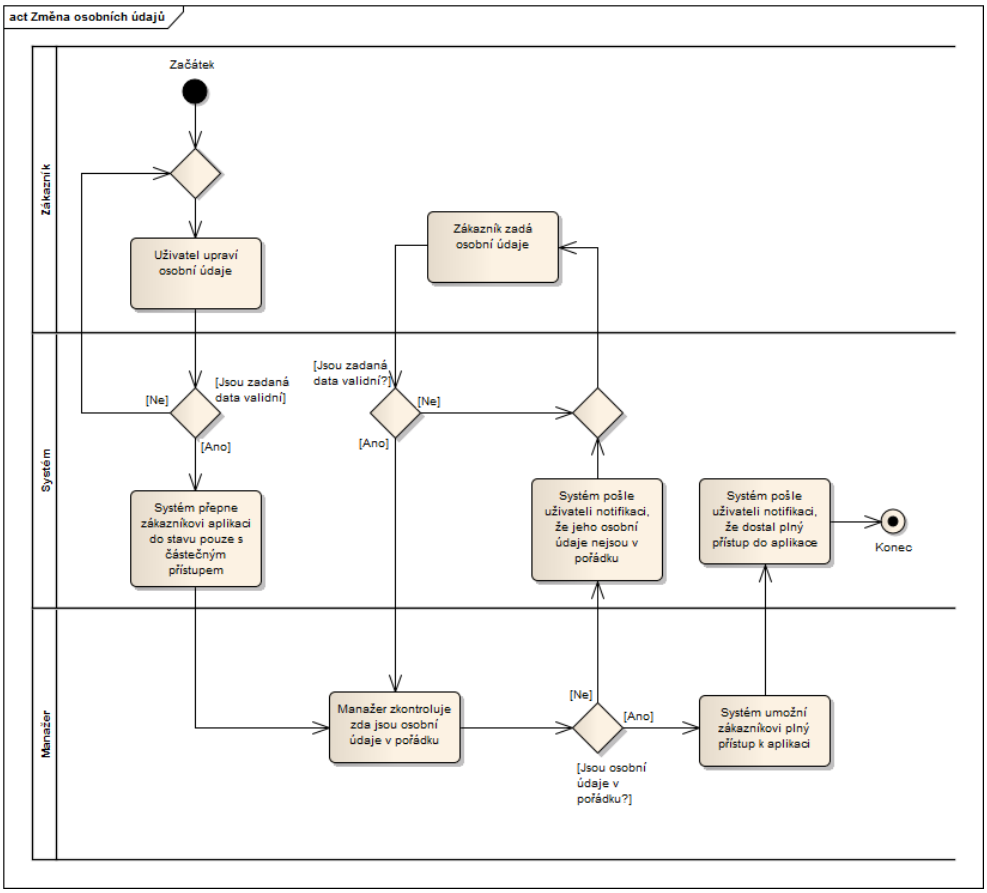
Pokud údaje podle manažera nejsou v pořádku, systém pošle uživateli notifikaci, aby své zadané osobní údaje opravil. Zákazník pak po opravení pošle opět žádost o registraci manažerovi. Ten ji může buď opět zamítnout, nebo schválit. To se opakuje tak dlouho, dokud manažer žádost zákazníkovi neschválí.



Obrázek 5.5: Diagram aktivity - Registrace

5.3.3 Změna osobních údajů

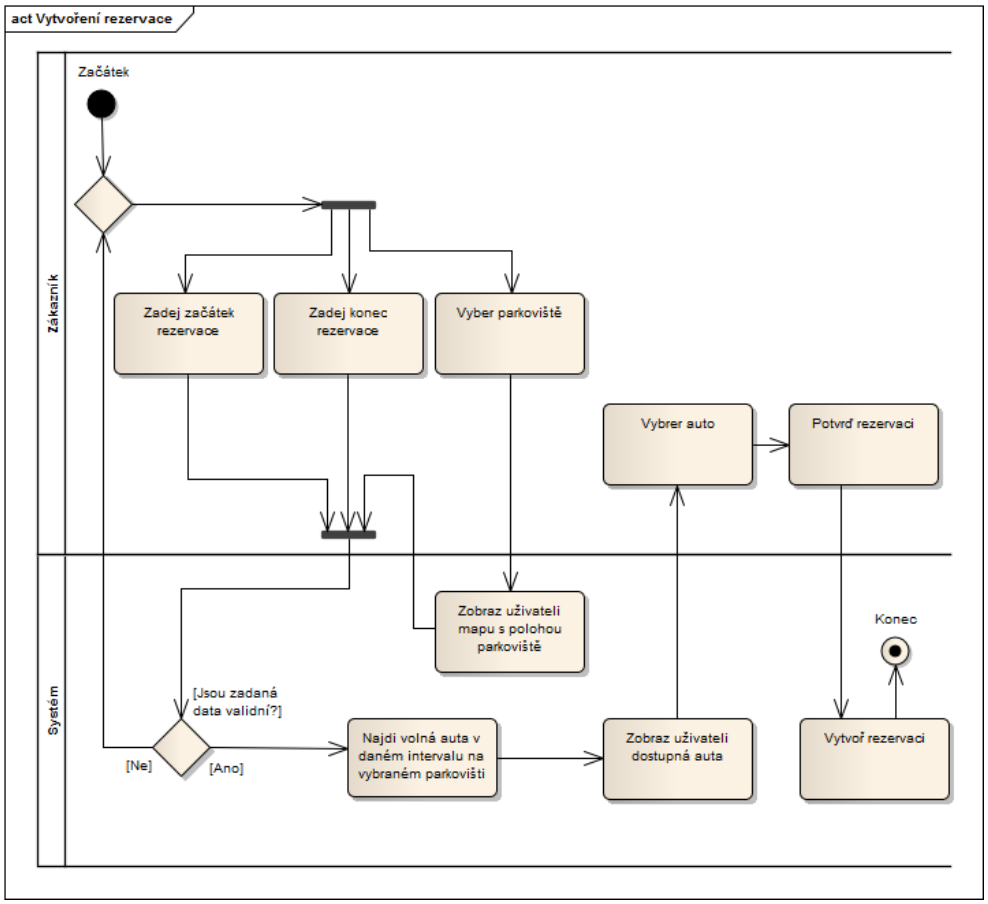
Změna osobních údajů je podobný proces jako registrace. Rozdíl je akorát v tom, že uživatel je již úspěšně registrovaný, a v systému měl tedy platná data. Díky tomu měl až dosud plný přístup k aplikaci. Vzhledem k tomu, že ale údaje změnil, tento plný přístup k aplikaci ztrácí. Dostává opět pouze omezený přístup, který mu umožňuje pouze měnit své údaje. Situace trvá až do té doby, než manažer zákazníkovi nové osobní údaje schválí.



Obrázek 5.6: Diagram aktivity - Změna osobních údajů

5.3.4 Vytvoření rezervace

Tento diagram aktivity slouží především k správnému pochopení průběhu vytvoření rezervace. Zákazník během toho vyplňuje formulář, který se ale postupně mění. Nejdřív uživatel vyplní časy registrace, mezi kterými si chce auto půjčit, a vybere parkoviště, kde si chce auto půjčit. Systém mu následně ukáže pro kontrolu mapu s vyznačeným parkovištěm a zobrazí mu aktuálně dostupná auta. Zákazník poté vybere auto a potvrdí rezervaci, která se následně zadá do systému.



Obrázek 5.7: Diagram aktivity - Vytvoření rezervace

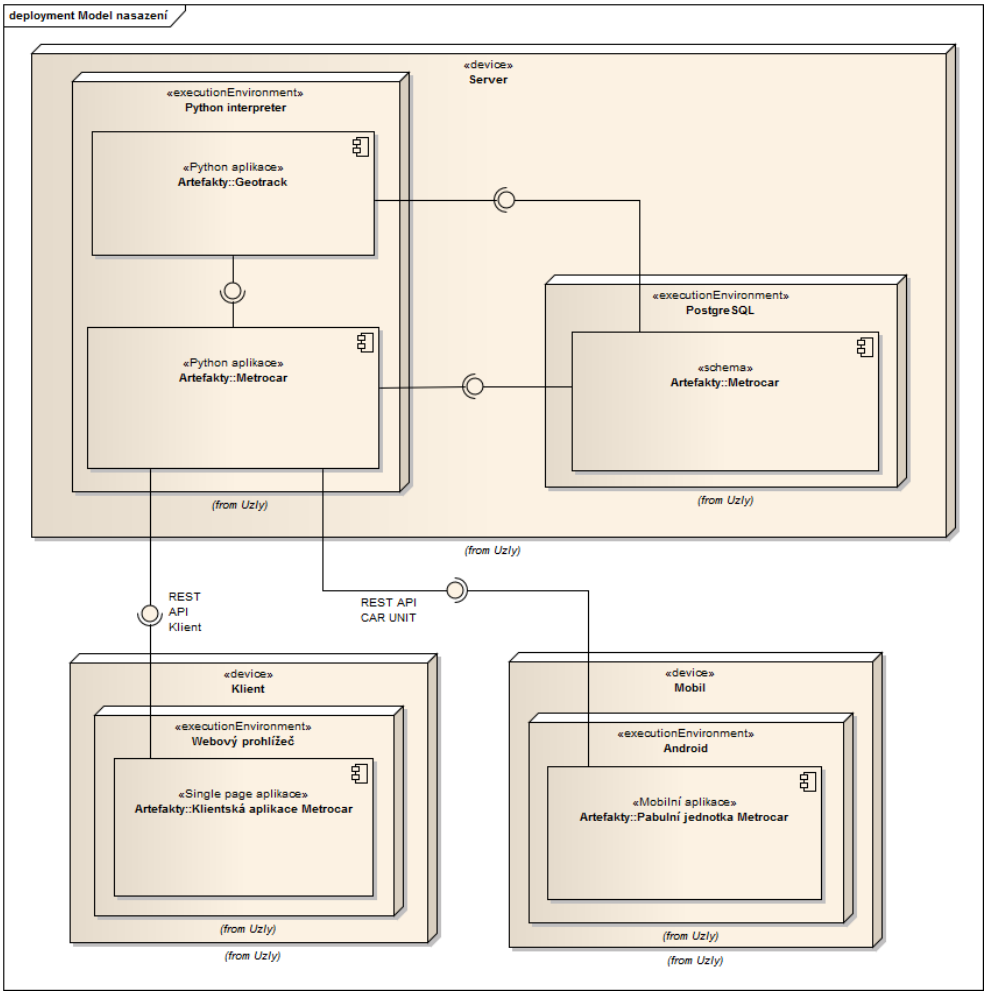
5.4 Model nasazení

Diagram, který v dokumentaci k projektu chybí a podle autora by zde měl určitě být, je, diagram nasazení. Čtyři aplikace, které projekt aktuálně obsahuje, a jejich vzájemná komunikace přes různá rozhraní, je již vcelku složitý systém, který určitě stojí za to detailněji popsat.

Středobodem tohoto modelu je serverová aplikace Metrocar. Ta vystavuje do okolí dvě RESTová rozhraní. Jedno je určeno primárně pro použití s mobilní aplikací, která zajišťuje komunikaci s palubní jednotkou. Toto rozhraní je již hotové a není třeba na něm aktuálně nic měnit. Druhé API pak slouží ke komunikaci s klientskou aplikací. Návrh a implementace tohoto rozhraní je součástí této bakalářské práce. Druhou aplikací, která je umístěna na serveru, je Geotrack. Tato aplikace slouží jako prostředník mezi serverovou, aplikací Metrocar a databází PostgreSQL[31]. Stará se zejména o zpracování a správné uložení geolokačních dat. API, přes které komunikuje serverová aplikace Metrocar s Geotrackem, sice není uděláno podle žádného standardu, ale je plně

zdokumentováno a popsáno v diplomové práci[9] Petra Pokorného.

Třetí aplikace, která se v projektu nachází, je mobilní aplikace pro komunikaci s palubní jednotkou. Ta komunikuje se serverovou aplikací pomocí webových služeb vytvořených dle architektury REST. Poslední aplikací je pak klientská aplikace, která komunikuje se serverem také pomocí webových služeb, ale ty jsou v serverové aplikaci vyčleněny zvlášť. Jde o to, že implementace webových služeb pro mobilní aplikaci byla vytvořena bez použití knihovny či frameworku. To zde sice nevadí, neboť toto API je tak malé, že v tu dobu to byla spíše výhoda nepřidávat do projektu další závislost. Ale použití tohoto řešení pro vytvoření API pro klientskou aplikaci, které je velmi obsáhlé, by nebylo příliš vhodné. Bylo tedy rozhodnuto, že zde bude použita knihovna Django REST framework.



Obrázek 5.8: Diagram aktivity - Vytvoření rezervace

5.5 Funkční a nefunkční požadavky

Vzhledem k tomu, že funkční požadavky byly již v projektu vypracovány a prakticky jsou pokryty případy užití, rozhodlo se po konzultaci s vedoucím práce, že je zbytečné je uvádět znovu. Nefunkční požadavky byly pak identifikovány tyto:

- Klientská aplikace je napsána za pomoci frameworku Ember.js.
- Serverová aplikace poskytuje REST API.
- Klientská aplikace běží pouze v prohlížečích s podporou Ecmascript 5.1.
- Serverová aplikace se musí instalovat přes balíčkovací systém pro Python pip.
- Klientská a serverová aplikace musí běžet na stejné doméně.

Kapitola 6

Návrh a implementace

6.1 REST API

Dříve než se začne navrhovat a psát nová klientská aplikace, bylo by dobré se zamyslet nad specifikací nového API, které bude nabízet server. Již v zadání se rozhodlo, že API bude vytvořeno dle architektury REST a to z důvodu jednodušší implementace klienta. Frameworky pro tvorbu klienta zpravidla již v základu obsahují REST adaptér pro komunikaci se serverem, a není tak potřeba tuto část aplikace psát.

6.1.1 Popis ověření uživatele

Pro ověření byl zvolen způsob autorizace pomocí tokenu. Tento výběr je opět veden snahou ulehčit si práci při napojování klientské aplikace na serverovou, ale také proto, že tento způsob se dnes již považuje za nepsaný standard. Získání tokenu je možné posláním přihlašovacích údajů (uživatelské jméno a heslo) metodou `POST` na endpoint `auth-token`. Ten vrací příslušný token a id právě ověřeného uživatele. Vše je vráceno ve formátu JSON.

6.1.2 Odchytávání chyb

Vzhledem k tomu, že není dobré, aby server vracel při vyhození výjimky odpověď se stavovým kódem `500`, bylo vhodné zde implementovat handler, který příslušné výjimky zachytí a pokusí se je vyřešit.

Výjimky, které lze v tomto případě řešit, jsou zejména výjimky vyhozené při validaci. Ty se transformují do odpovědi se stavovým kódem `400`. Tělo odpovědi je pak asociativní pole, jehož klíče jsou jednotlivé vlastnosti daného modelu, které neprošly validací. Hodnoty tohoto asociativního pole jsou pak dále tvořeny dalšími poli, které obsahují seznam chybových zpráv, které vznikly během validace.

Další výjimka, kterou lze úspěšně řešit, je, pokud uživatele nejde ověřit. Tedy jeho požadavek neobsahuje v hlavičce jeho autorizační token. V tomto případě je pak vrácena odpověď se stavovým kódem `401`. Tato situace může nastat v celém API s výjimkou endpointu pro získání tokenu a endpointu pro odeslání žádosti o registraci.

S tímto problémem souvisí i řešení problému, zda má uživatel oprávnění danou akci provést. Pro ověření, zda opravdu uživatel příslušné povolení má, zde slouží tzv. `Permissions`. To jsou třídy, které mají implementované metody, jež na základě kontextu rozhodnou, zda uživatel tuto akci může vykonat či ne. Podle toho vrací hodnotu pravda/nepravda. Podle toho se též rozhodne, zda se má do odpovědi nastavit stavový kód `403` či kód jiný.

Seznam oprávnění, která jsou definována:

- `IsAuthenticated`,
  - Zkontroluje, zda je uživatel přihlášený.
- `IsAdminOrReadOnly`,
  - Pokud je aktuální uživatel administrátor, dovolí mu vykonat všechny metody, které jsou na dané množině pohledů definovány. Pokud uživatel nepatří do skupiny administrátorů, povolí vykonat pouze metody, které nikterak nemění data.
- `IsAdminUserOrOwner`,
  - Akci povolí tehdy jen pokud uživatel patří do skupiny administrátorů nebo je daný uživatel vlastníkem daného objektu. To znamená, že daný objekt má vazbu na aktuálního uživatele.
- `IsAccountOwner`
  - Povolí akci jen těm uživatelům, kteří mají na daný objekt nepřímou vazbu přes uživatelský účet.
- `IsUserFullyActive`
  - Zabraňuje uživatelům, kteří nemají plný přístup k aplikaci, aby vytvářeli, měnili či mazali data.

6.1.3 Přenášení geolokačních dat

Díky potřebě zobrazit na mapě polohu parkovišť byla i na API přidána podpora pro přenos geolokačních dat. Ta jsou zabalená uvnitř klíčového slova `POLYGON`, které označuje, že se jedná o ohraničené území a ne jen o samostatný bod. V závorce jsou pak uvedeny po dvojicích souřadnice bodů, které jsou vrcholy víceúhelníku stanovující hranice parkoviště.

Listing 6.1 : Ukázka serializace geolokačních dat

```
POLYGON ((
14.4041007737154345 50.0681585661877833,
14.4053506831159410 50.0683479467924215,
14.4053721407868256 50.0680414941669412,
14.4041544178972245 50.0678555556647069,
14.4041007737154345 50.0681585661877833
))
```

6.1.4 Specifikace REST API

- **zdroj:** `accountactivities`
  - **Popis:** Vrací seznam obsahující seznam aktivit pro aktuálního uživatele
  - **Povolené operace:**
    - `GET` /
  - **Serializovaný model obsahuje tyto položky:**
    - `id`,
    - `datetime` - čas vytvoření aktivity,
    - `money_amount` - částka,
    - `activity_type` - typ aktivity (kredit, účet za palivo nebo peníze za rezervaci).
- **zdroj:** `auth-token`
  - **Popis:** Na základě poslaného jména a hesla vrací token a id uživatele
  - **Povolené operace:**
    - `POST` /
- **zdroj:** `carcolors`
  - **Popis:** Nabízí klasické operace pro listování mezi barvami aut a pro vytvoření, editace, získání detailu a smazání barvy auta.
  - **Povolené operace:**
    - `GET` /
    - `POST` /
    - `GET` /{pk}
    - `PUT` /{pk}

        * `DELETE` /{pk}
    - **Serializovaný model obsahuje tyto položky:**
        * `id`,
        * `color` - název barvy.

- **zdroj:** `carmodels`
    - **Popis:** Nabízí klasické operace pro listování mezi modely aut a pro vytvoření, editace, získání detailu a smazání modelu auta.
    - **Povolené operace:**
        * `GET` /
        * `POST` /
        * `GET` /{pk}
        * `PUT` /{pk}
        * `DELETE` /{pk}
    - **Serializovaný model obsahuje tyto položky:**
        * `id`,
        * `name` - jméno modelu auta,
        * `manufacturer` - jméno výrobce auta,
        * `type` - typ auta,
        * `engine` - popis motoru auta,
        * `seats_count` - počet míst v autě,
        * `storage_capacity` - objem kufru auta,
        * `main_fuel` - typ paliva auta.

- **zdroj:** `cars`
    - **Popis:** Nabízí klasické operace pro listování auty a pro vytvoření, editace, získání detailu a smazání auta.
    - **Povolené operace:**
        * `GET` /
        * `POST` /
        * `GET` /{pk}
        * `PUT` /{pk}
        * `DELETE` /{pk}
    - **Serializovaný model obsahuje tyto položky:**
        * `id`,
        * `active` - flag, zda je auto k dispozici pro rezervace,
        * `manufacture_date` - datum vyrobení auta,

    * `registration_number` - registrační značka,
    * `model` - cizí klíč na model auta,
    * `color` - cizí klíč na barvu auta,
    * `owner` - vlastník,
    * `home_subsidiary` - domovská pobočka,
    * `last_echo` - poslední známá pozice,
    * `parking` - parkoviště, kde je auto umístěno,
    * `car_name` - jméno auta dopočítané z názvu modelu auta a barvy auta.

- **zdroj:** `fuelbills`
  - **Popis:** Nabízí klasické operace pro listování mezi účty za palivo a pro vytvoření, editace, získání detailu a smazání účtu za palivo.
  - **Povolené operace:**
    * `GET` /
    * `POST` /
    * `GET` /{pk}
    * `PUT` /{pk}
    * `DELETE` /{pk}
  - **Serializovaný model obsahuje tyto položky:**
    * `id`,
    * `datetime` - čas vytvoření účtenky paliva,
    * `account` - účet zákazníka,
    * `money_amount` - částka za palivo,
    * `car` - cizí klíč na auto,
    * `fuel` - palivo,
    * `liter_count` - počet litrů natankovaného paliva,
    * `place` - místo tankování,
    * `image` - obrázek účtenky,
    * `approved` - flag, zda byla účtenka schválena manažerem.

- **zdroj:** `journey`
  - **Popis:** Vrací všechny cesty, které se vztahují k aktuálnímu uživateli. Uživateli také umožňuje vytvořit novou cestu, měnit jeho aktuální cesty a popřípadě je i mazat.
  - **Povolené operace:**
    * `GET` /
    * `POST` /

        * `GET` /{pk}
        * `PUT` /{pk}
        * `DELETE` /{pk}
    - **Serializovaný model obsahuje tyto položky:**
        * `id`,
        * `comment` - komentář k cestě,
        * `start_datetime` - startovní čas cesty,
        * `end_datetime` - koncový čas cesty,
        * `length` - délka cesty,
        * `total_price` - celková cena za cestu,
        * `type` - typ cesty,
        * `reservation` - cizí klíč na rezervaci,
        * `car` - cizí klíč na auto,
        * `user` - cizí klíč na uživatele,
        * `speedometer_start` - počáteční stav tachometru,
        * `speedometer_end` - konečný stav tachometru.

- **zdroj:** `parkings`
    - **Popis:** Nabízí klasické operace pro listování mezi parkovišti aut a dále umožňuje vytvoření, editace, získání detailu a smazání parkovišť.
    - **Povolené operace:**
        * `GET` /
        * `POST` /
        * `GET` /{pk}
        * `PUT` /{pk}
        * `DELETE` /{pk}
    - **Serializovaný model obsahuje tyto položky:**
        * `id`,
        * `name` - jméno parkoviště,
        * `places_count` - počet míst k parkování,
        * `street` - ulice, kde je parkoviště,
        * `land_registry_number` - číslo popisné,
        * `city` - město,
        * `polygon` - souřadnice popisující parkoviště na mapě.

- **zdroj:** `registrations`

  - **Popis:** Umožňuje pouze operaci `POST`, která vytvoří na serveru uživatele s částečným přístupem do aplikace, vytvoří pro něj žádost a také pro něj vytvoří token pro autorizaci se serverem.
  - **Povolené operace:**
    * `POST` /
  - **Serializovaný model obsahuje tyto položky:**
    * `id`,
    * `username` - uživatelské jméno,
    * `first_name` - křestní jméno,
    * `last_name` - příjmení,
    * `active` - flag, zda má uživatel plný přístup k aplikaci,
    * `email` - emailová adresa,
    * `primary_phone` - telefonní číslo,
    * `date_of_birth` - datum narození,
    * `identity_card_number` - číslo občanského průkazu,
    * `identity_card_image` - fotka občanského průkazu,
    * `drivers_licence_number` - číslo řidičského průkazu,
    * `drivers_licence_image` - fotka řidičského průkazu,
    * `street` - ulice,
    * `land_registry_number` - číslo popisné,
    * `zip_code` - poštovní směrovací číslo,
    * `city` - město.

- **zdroj:** `reservations`
  - **Popis:** Vrací všechny rezervace, které se vztahují k aktuálnímu uživateli. Uživateli také umožňuje vytvořit rezervaci, měnit jeho aktuální rezervace a popřípadě je i mazat.
  - **Povolené operace:**
    * `GET` /
    * `POST` /
    * `GET` /{pk}
    * `PUT` /{pk}
    * `DELETE` /{pk}
  - **Serializovaný model obsahuje tyto položky:**
    * `id`,
    * `cancelled` - flag, zda byla rezervace zrušená,
    * `comment` - komentář k rezervaci,

    * `created` - čas vytvoření rezervace,
    * `ended` - čas uzavření rezervace,
    * `finished` - flag, zda byla rezervace úspěšně ukončena,
    * `modified` - časová známka poslední modifikace,
    * `price` - cena za rezervaci,
    * `reserved_from` - čas začátku rezervace,
    * `reserved_until` - čas konce rezervace,
    * `started` - flag, zda začala rezervace,
    * `user` - cizí klíč na uživatele,
    * `car` - cizí klíč na auto.

- **zdroj:** `userbalances`
  - **Popis:** Vrací aktuální zůstatek na účtu zákazníka.
  - **Povolené operace:**
    * `GET` /{pk}
  - **Serializovaný model obsahuje tyto položky:**
    * `id`,
    * `balance` - aktuální zůstatek na účtu zákazníka,
    * `user` - cizí klíč na uživatele účtu.

- **zdroj:** `users`
  - **Popis:** Vrací pouze detail uživatele.
  - **Povolené operace:**
    * `GET` /{pk}
  - **Serializovaný model obsahuje tyto položky:**
    * `id`,
    * `username` - uživatelské jméno,
    * `first_name` - křestní jméno,
    * `last_name` - příjmení,
    * `active` - flag, zda má uživatel plný přístup k aplikaci,
    * `email` - emailová adresa,
    * `primary_phone` - telefonní číslo,
    * `date_of_birth` - datum narození,
    * `identity_card_number` - číslo občanského průkazu,
    * `identity_card_image` - fotka občanského průkazu,
    * `drivers_licence_number` - číslo řidičského průkazu,
    * `drivers_licence_image` - fotka řidičského průkazu,

* `street` - ulice,
* `land_registry_number` - číslo popisné,
* `zip_code` - poštovní směrovací číslo,
* `city` - město.

6.2 Klientská aplikace

6.2.1 Návrh aplikace

Klientská aplikace bude vytvořena podle osvědčené architektury Model-view-controller (MVC), která nám umožňuje oddělit jednotlivé vrsty aplikace do smysluplných celků.

6.2.1.1 View

Díky použité technologii pro tvorbu aplikace (Ember) musela být ale zobrazovací vrstva (view) rozdělena ještě na dvě části. První část jsou samotné šablony. Ty jsou napsány za pomoci šablonovacího systému Emblem a mají za úkol uživateli pouze zobrazovat již spočtená data. Výjimkou je řízení toho, co se má zobrazit pomocí podmínek. Ty ale berou jako argument pouze proměnou typu boolean. Dopočtení hodnot a případnou manipulaci s daty zajišťují třídy, které jsou v základu odvozeny od třídy `Ember.View`. Ty umožňují reagovat na události vzniklé v šabloně, sledují proměnné a dopočítávají podle nich jiné proměnné. Data mění pouze ale směrem od modelu k uživateli, ne naopak.

6.2.1.2 Controller

Řadiče (controllery) zde klasicky zastávají úlohu řídící jednotky, která distribuuje události vzniklé typicky od uživatele k ostatním jednotkám systému (například k modelu). Řídí také, jaké pohledy se mají uživateli zobrazit, a naplňuje šablony daty.

6.2.1.3 Model

Model je zde v aplikaci zajišťován za pomoci knihovny Ember Data. Ta v základu nabízí již předpřipravené třídy pro práci s daty. Vychází přitom z návrhového vzoru Active Record. K dispozice je třída `DS.Model`, z které je možné dědit a vytvořit model popisující data přicházející z API. Tento model je nezávislý na podobě příchozích dat. O jejich serializaci a deserializaci se stará potomek třídy `Serializer`. Díky důslednému oddělení je možné jednoduše přijímat data v různých formátech (json, xml, ...), které jsou převedeny vždy do samého modelu.

Serializéry pro běžné formáty jsou v knihovně již připraveny. Vzhledem k tomu, že námi vytvořené API na serveru poskytovalo trochu jiný formát

odpovědi, než uměl JSON serializér z knihovny, musel se pro potřeby aplikace JSON serializér trochu poupravit. Šlo konkrétně o to, že serializér z knihovny očekával, že objekt či pole bude zaobaleno ještě do objektu, jehož klíč bude odpovídat názvu modelu.

Listing 6.2 : Porovnání odpovědi přicházející ze serveru a serializérem očekávané odpovědi

```
# Odpověď ze serveru
{
  "id": 1,
  "name": "Josef Mika"
}

# Serializérem očekávaná odpověď
{
  "user": {
    "id": 1,
    "name": "Josef Mika"
  }
}
```

Vzhledem k tomu, že někdy potřebujeme změnit strukturu či formát dat přišlých ze serveru, byly do knihovny Ember Data přidány transformery. Ty mají metody pro normalizaci a denormalizaci hodnot. Využití můžeme nalézt například u transformace data. V této bakalářské práci tohoto modelu pak využijeme při transformaci přišlých geolokačních dat, které transformujeme z jednoduchého řetězce do pole obsahujícího souřadnice zadaného místa.

Poslední část knihovny, kterou lze využít, jsou pak předpřipravené adaptéry. Ty umožňují napojení naší aplikace na poskytované API. V základu je nabízen i REST adaptér, ze kterého se v klientské aplikaci vychází. Adaptér definovaný v klientské aplikaci pak obsahuje navíc i možnost přidat token pro autorizaci do hlavičky požadavku na server.

6.2.2 Použité návrhové vzory

Je jistě dobré řídit se během návrhu aplikace standardními postupy a uplatňovat návrhové vzory. Umožníte tak snazší čtení kódu vašim nástupcům či kolegům. A dozajista se též vyhnete stavu, že vaše vlastní řešení má konsekvence, které jste z počátku neviděli. Například že aplikace se dá hůře testovat.

6.2.2.1 Active Record

Návrhový vzor Active Record patří do skupiny architektonických návrhových vzorů. Martin Fowler ho ve své knize Patterns of Enterprise Application

Architecture[32] definuje jako objekt, který obaluje řádek v databázové tabulce nebo pohledu, zapouzdřuje přístup k databázi a přidává doménovou logiku (vlastní překlad). V klientské aplikaci byl tento návrhový vzor použit pro definování přístupu k persistentním datům.

6.2.2.2 Adapter

Jako most mezi API a modelem v klientské aplikaci byl pak použit návrhový vzor Adapter. Ten patří do skupiny strukturálních vzorů. Jeho podstatou je, že umožňuje spolupracovat dvěma třídám, které nemají kompatibilní rozhraní. Slouží tedy pro převádění jednoho rozhraní na druhé. V naší aplikaci je tento návrhový vzor použit pro spojení rozhraní pro perzistenci dat poskytovaných modelovou vrstvou s RESTovým rozhraním.

6.2.2.3 Observer

Jedna z výhod Single page aplikací je, že bývají povětšinou velmi interaktivní. Jsou zde běžné věci, kdy se vám například s vyplněním textu ve formuláři zároveň mění obsah na stránce. Tento případ je ukázkovým příkladem použití návrhového vzoru observer, který patří do skupiny vzorů chování. Použití tohoto vzoru je zejména tehdy, máte-li více objektů závislých na dalším objektu, a vy potřebujete dát vědět, že se na tomto dalším objektu něco změnilo.

6.2.2.4 Proxy

Kvůli neustále měnícím se obsahu potřebujeme udržovat kolekce dat synchronizované s tím, co vidí uživatel zobrazený v šablonách. Na to se hodí kombinace dvou vzorů. Nejdřív obalíme klasické pole pomocí proxy a k datům v poli přistupujeme pouze přes tento objekt. Tato proxy pak implementuje návrhový vzor observer. Výsledkem bude skutečnost, že pokud se změní data v kolekci, například přidáme prvek či prvek smažeme, projeví se nám tato změna i v GUI.

6.2.3 Návrh obrazovek

Poslední část, kterou zbývá promyslet, je návrh obrazovek. Při analyzování případů užití a funkčních požadavků se zjistilo, že je potřeba pro první verzi aplikace implementovat devět obrazovek a tři komponenty.

Seznam obrazovek:

- registrace,
- přihlašovací obrazovka,
- seznam rezervací,

- vytvoření rezervace,
- seznam žádostí o proplacení paliva,
- vytvoření žádosti o proplacení paliva,
- historie pohybů na účtu,
- uživatelský profil,
- změna hesla.

Seznam komponent:

- modální okno,
- komponenta pro zobrazení mapy,
- komponenta pro zobrazení avatara.

6.2.4 Struktura aplikace

Struktura klientské aplikace je vytvořena podle doporučení daného projektem Ember CLI. Ten přináší do frameworku Ember mimo jiné i dva návrhy možné složkové struktury aplikace. První, který se více hodí pro menší projekty, sdružuje vždy do jedné složky stejně orientovaný obsah. Existuje tak například jedna složka pro modely, jedna složka pro šablony atd. Druhý návrh, který je použit i zde v klientské aplikaci, je rozdělení aplikace do modulů. To činí aplikaci většího rozsahu přehlednější, neboť je na první pohled jasné, co všechno například patří k tématu rezervace.

Listing 6.3 : Struktura klientské aplikace

```
/app    - složka obsahující samotnou aplikaci
  /styles - složka se styly
  /helpers - složka s helpery určenými pro snazší tvorbu
        šablon
  /initializers - složka se všemi initializéry, které se
     pouští po startu aplikace
  /mixins - složka obsahující mixiny určené pro celou aplikace
  /locales - složka se soubory s lokalizacemi
  /pods - složka s moduly aplikace
  /app.coffee - vstupní bod aplikace
  /index.html - jediná HTML stránka v aplikaci, která
     obsahuje věci pro nastartování aplikaci
  /router.coffee        - soubor mapující url na routovací
     třídy
```

```
/Brocfile.js    - soubor popisující sestavení aplikace
/bower.json - soubor obsahující seznam bower závislostí
/bower_components - složka s bower závislostmi
/package.json - soubor obsahující seznam node.js závislostí
/node_modules - složka s node.js závislostmi
/config - složka obsahující soubor enviroment.js, který
   popisuje konfiguraci aplikace pro různá prostředí
/public - složka, jejíž obsah je zkopírován do kořene
   sestavené aplikace. Obsahuje tradičně obrázky a další
   assety
/dist - složka obsahující sestavenou aplikaci
/testem.json - soubor s nastaveními určenými pro spouštění
   testů
/tests - složka s testy
```

6.2.5 Autentizace a autorizace

Pro usnadnění řešení problému autentizace a autorizace uživatele byla použita knihovna Ember Simple Auth[33]. Její použití je jednoduché. Stačí přidat na příslušnou definici rout mixin říkající, že zde je přístup až po úspěšné autentizaci. Dále stačí nastavit v konfiguraci, jaký typ autentizace a autorizace se má použít. V nabídce je opět několik předpřipravených možností. Pro naše potřeby ale bylo potřeba naimplementovat vlastní autentizátor, který po úspěšném ověření uživatelského jména a hesla na serveru uloží do Local storage prohlížeče token a id uživatele a při odhlášení ho naopak z Local storage vymaže. Autorizace uživatele je pak řešena jednoduše zkontrolováním, zda v Local storage daný token existuje.

Pro potřeby naší aplikace byl přimixován ke všem routám mixin vyžadující přihlášeného uživatele, s výjimkou routy umožňující uživateli přihlásit se či se registrovat. Pokud se ale přesto uživatel snaží do této zabezpečené části aplikace dostat, je přesměrován na přihlašovací obrazovku. Při úspěšné autentizaci je pak uživatel přesměrován naopak na seznam všech jeho rezervací.

6.2.6 Validace dat

Vzhledem k uplatnění výhod aplikace vytvořené podle Single page application přístupu bylo rozhodnuto řešit validace na dvou místech - jak v serverové aplikaci, tak i v klientské. Pro usnadnění tvorby byla zvolena knihovna Ember Validations[34]. Ta umožňuje definovat na modelu seznam omezení, které jsou kladeny na vlastnosti daného modelu. Validace zde popsané jsou spíše validacemi na formát vstupu. Vyšší logika validací zde není implementována, neboť je již jednou napsána na serveru. Validace dané vlastnosti se provádí

při každé změně hodnoty dané vlastnosti. Uživateli tedy umožňuje zpětnou vazbu, že pole, které vyplnil, je zadáno ve správném formátu. Validaci je též možno přidat manuálně, například při akci na odesláno formuláře.

Pokud validace v klientské aplikaci projde, je odeslána žádost na server. Pokud ten vyhodnotí, že poslané údaje nejsou zcela v pořádku, vrátí pole se seznamem chyb. Ten je následně naparsován a chyby pak přidány do pole chyb k dané vlastnosti modelu. To umožňuje jednoduchou správu chyb, neboť jak validace na klientu, tak na serveru používají stejné pole pro kumulaci chyb.

6.2.7 Lokalizace

Zejména pro budoucí rozvoj aplikace byla přidána podpora pro lokalizaci. Pro usnadnění byla použita knihovna Ember CLI i18n[35]. Použití této knihovny je jednoduché. Stačí vytvořit ve složce `locales` soubor pojmenovaný zkratkou použitého jazyka. V něm následně definovat pomocí zápisu klíč hodnota jednotlivé lokální překlady. Jako ukázka možností tohoto postupu zde byly takto řešeny hlášky, které vzniknou při validaci modelu.

Listing 6.4 : Ukázka části souboru řešící lokalizaci v aplikaci

```
export default {

  errors: {
    empty: "Toto políčko nesmí být prázdné",
    email: "Zadaná hodnota musí být emailová adresa",
    phone: "Zadaná hodnota musí být telefonní číslo",
    password_confirmation: "Zadaná hesla se musí shodovat"
  }

  ...

};
```

Kapitola 7

Testování

7.1 Testování API

Vzhledem k tomu, že serverová aplikace již testy obsahuje, bylo třeba je pouze doplnit o integrační testy na otestování nově vytvořeného API. Celkem bylo přidáno 47 testů. Testy byly zaměřeny například na ověřování správnosti vrácených stavových kódů a na ověřování oprávnění, zda uživatel smí akci provést. Jako příklad lze uvést zakáz vytvoření rezervace uživateli, který nemá plně aktivovaný účet. Dále bylo testováno, zda vrácený obsah odpovídá obsahu, který měl být vrácen. Například, zda se uživateli vrátily pouze jeho rezervace a ne rezervace všech uživatelů.

7.2 Testování klientské aplikace

Pro otestování klientské aplikace byly vytvořeny scénáře, které by měly pokrýt základní funkcionality aplikace. V textu níže se mluví o administraci projektu, kterým je myšlena standardní administrace serverové aplikace, kterou nabízí v základu framework Django.

7.2.1 Registrace a přihlášení

Cíl scénáře

Cílem tohoto scénáře je ověřit, zda vás aplikace bez přihlášení nepustí dále než na obrazovku s přihlášením a registrací. Dále je třeba ověřit, že po registraci má klient pouze omezený přístup do aplikace.

Postup ověření

Při prvním spuštěním by vás aplikace neměla pustit jinam než na obrazovku s přihlášením a na obrazovku s registrací. Zkuste proto změnit URL například na `/reservations/create`. Aplikace by vás měla přesměrovat zpět na úvodní

obrazovku s přihlášením. Dále přejděte na stránku s registrací a vyplňte osobní údaje a nahrajte fotky občanského a řidičského průkazu. Po úspěšném zadání by vás aplikace měla přesměrovat na stránku s poděkováním za registraci a tlačítkem pro přihlášení.

Po přihlášení je nutné zkontrolovat, zda opravdu máte pouze částečný přístup. Pod navigačním panelem by měl být svítit žlutý pruh s nápisem, že nemáte plný přístup k aplikaci. Částečný přístup k aplikaci v praxi znamená, že vás aplikace například nepustí na obrazovku s možností vytvořit rezervaci. Toto lze jednoduše zkontrolovat pomocí změny URL na `/reservations/create`. Aplikace by vás měla automaticky přesměrovat na stránku s textem, že přístup na tuto stránku vám byl odepřen.

7.2.2 Schválení registrace

Cíl scénáře

Cílem toho scénáře je ověřit, zda po schválení žádosti manažerem v administraci, klientská aplikace umožní uživateli plný přístup k aplikaci.

Postup ověření

Pro navození této situace je nutné přihlásit se do administrace projektu a schválit požadavek na registraci. Po aktualizaci klientské aplikace by měl žlutý pruh signalizující omezený přístup zmizet. Též by vám měla přibýt nabídka na vytvoření rezervace, na kterou lze i úspěšně přejít.

7.2.3 Změna osobních údajů

Cíl scénáře

Cílem toho scénáře je ověřit, zda se po změně údajů aplikace přepne do režimu s částečným přístupem.

Postup ověření

Nejprve je nutné přejít na stránku profil. Zde pak kliknout na tlačítko upravit a změnit si například jméno. Poté byste měli být přesměrováni na stránku vysvětlující, že změna údajů čeká na schválení manažerem. Měl by se též ihned opět objevit žlutý pruh indikující částečný přístup a měl by vám být odmítnut přístup například na stránku s vytvořením rezervace.

7.2.4 Změna hesla

Cíl scénáře

Cílem toho jednoduchého scénáře je ověřit funkcionalitu změny hesla.

Postup ověření

Jděte na stránku se změnou hesla. Odkaz na ní se nachází při kliknutí na avatara vpravo nahoře. Zadejte staré a nové heslo a potvrďte tlačítkem Dokončit. Poté se zkuste přihlásit a odhlásit.

7.2.5 Vytvoření rezervace

Cíl scénáře

Cílem tohoto scénáře je ověřit, zda jde úspěšně rezervace vytvořit a zrušit.

Postup ověření

Nejprve zvolíme z levého menu položku pro vytvoření rezervace. Dále vyplníme všechny údaje, zkontrolujeme, že je na mapce zobrazeno opravdu námi vybrané parkoviště a rezervaci odešleme. Měli bychom být aplikací přesměrování na seznam rezervací. Rezervaci pak zkusme zrušit a zkontrolujme, zda se její stav indikovaný ikonou vlevo změnil na křížek. Dále je třeba vyzkoušet, že při úspěšné jízdě se zobrazí zelená fajfka u rezervace a též se zobrazí cena, která je i odečtena z uživatelského účtu. Toto lze nasimulovat v administraci přidáním jízdy v editaci rezervace.

7.2.6 Vytvoření požadavku na proplacení paliva

Cíl scénáře

Cílem tohoto scénáře je požádat prostřednictvím klientské aplikace o proplacení paliva a zkontrolovat, zda se při úspěšném schválení peníze připíší na účet.

Postup ověření

Nejprve je třeba kliknout v levém panelu na položku „Účty za palivo“ a dále na tlačítko „Vytvořit žádost“. Dále je třeba vyplnit formulář a nahrát fotku účtenky. Po úspěšném vyplnění vás aplikace přesměruje zpět na stránku se seznamem žádostí. Dále je třeba jít do administrace projektu a žádost schválit. Po obnovení stránky s klientskou aplikací by pak mělo být vidět, že u žádosti místo modrých hodin svítí zelená fajfka.

Kapitola 8

Nasazení

Nasazení celého projektu se díky jeho rozsáhlosti ukázalo býti vcelku netriviálním problémem. V této kapitole je tedy popsán postup, jak by se aktuálně měl projekt nasazovat.

8.1 Sestavování serverové aplikace

Sestavení serverové aplikace bylo určitě největším problémem. Serverová aplikace totiž obsahuje aktuálně dvě aplikace - samotnou aplikaci Metrocar a aplikaci Geotrack. Pro zjednodušení nasazení bylo potřeba popsat proces vytvoření balíčků tak, aby se následně daly jednoduše generovat a na příslušném serveru nainstalovat či upgradovat. Python nabízí dnes již standardní postup, jak toho docílit. Nabízí též i repozitář (pypi), kam může vývojář popřípadě balíček nahrát, a sdílet tak svojí práci s komunitou.

8.1.1 Tvorba pip balíčků

Vytvořit pip balíček není vůbec těžké. Stačí k tomu vytvořit dva soubory - `setup.py` a `MANIFEST.in`. První soubor popisuje základní informace o balíčku jako název, verzi balíčku, autora, seznam závislostí, krátký popis, co balíček umí, licenci a další. Druhý soubor `MANIFEST.in` není povinný, ale při vytváření balíčku pro Django aplikaci se nám hodí. Umožňuje totiž specifikovat soubory, které se mají do balíčku zahrnout, a soubory, které se mají naopak vynechat. Umožňuje tak přibalení například šablon, obrázků a dalších věcí, které nejsou povahou soubory obsahující python kód. Generovaní samotného balíčku pak probíhá pomocí příkazu `pip sdist`.

8.1.2 Nasazení serverové aplikace

V produkčním prostředí je jistě dobré použít jeden z webových serverů namísto vestavěného serveru v Djangu. S ohledem na jednoduchost nastavení

byl nakonec vybrán Apache HTTP server. Do něho byla ještě potřeba nainstalovat modul mod_wsgi, který umožňuje hostovat Python aplikace s podporou WSGI. Pro samotné nasazení novější verze pak stačí pomocí příkazu `pip install` nainstalovat sestavené balíčky. Dále je potřeba ještě pustit migraci databáze a restartovat webový server.

8.2 Sestavení a nasazení klientské aplikace

Sestavování klientské aplikace probíhá už podstatně jednodušeji. Nejdříve je nutné pomocí příkazů `npm install` a `bower install` nainstalovat všechny závislosti. Pak už stačí pouze příkazem `ember build -env=production` zadat pokyn k sestavení a výslednou aplikaci, které je zkompilovaná v adresáři `dist`, nasadit. Vzhledem k tomu, že výsledná aplikace je pouze kombinace HTML, CSS, Javascriptu, lze jednoduše nastavit například Apache HTTP Server, aby danou aplikaci servíroval klientům.

8.3 Prvky automatizace

Vzhledem k tomu, že příkazů, které je nutno zadat, aby se například při drobné změně aplikace sestavila a nasadila, je dost, bylo rozhodnuto, že alespoň pro vývoj bude experimentováno s možností použití CI serveru. Vzhledem k tomu, že cílem této práce jistě není popsat proces automatizace sestavení a nasazení tohoto projektu, byla hledána co nejjednodušší varianta, jak alespoň nějaké prvky automatizace, které by dozajista zjednodušily práci projektu, zavést. Jako nejschůdnější varianta CI serveru byl nakonec zvolen Jenkins[36]. Ten aktuálně patří k nejznámnějším a nejpoužívanějším CI serverům na světě a navíc je dostupný v plné verzi zcela zdarma. Celý proces sestavení a nasazení byl nakonec rozdělen do 8 kroků, přičemž každý krok odpovídal jednomu build stepu na Jenkinsu. Bylo také nastaveno, že celý tento proces se má spustit pokaždé, kdy nastane ve větvi `master` změna.

Seznam build stepů:

1. stáhnutí nejnovější verze projektu z gitu,
2. nakopírování souboru s lokálním nastavením do adresáře projektu,
3. aktivace virtuálního prostředí pro Python a sestavení serverové aplikace,
4. odinstalace aktuálně nasazené verze aplikace Metrocar,
5. instalace nové verze aplikace Metrocar, vytvoření souboru pro logování a nastavení práv,
6. migrace databáze,

7. sestavení klientské aplikace,
8. nasazení klientské aplikace.

Závěr

Cíl práce byl splněn na sto procent. Bylo vytvořeno jak REST API, tak i nová klientská aplikace. Návrh udělat klientskou aplikaci formou Single page aplikace se ukázalo jako více než vhodné řešení. Během práce bylo provedeno nad rámec zadání i mnoho dílčích oprav serverové aplikace. Dále byl také proveden výzkum ohledně možnosti automatického testování a nasazení projektu, neboť projekt se s další přidanou aplikací stal opět více komplexní a jeho nasazení tak již není triviální záležitostí.

Mezi hlavní přínosy této práce lze uvést skutečnost, že projekt má konečně plně funkční aplikaci pro zákazníky, a je tedy zase o kousek blíže reálnému nasazení. Vytvořením této aplikace se také doplnil poslední dílek pomyslné skládačky, a projekt už tak má aktuálně hotové všechny svoje části.

Možných dalších vylepšení existuje stále dost. Například by bylo jistě dobré dotáhnout proces automatizace testování a nasazení po vzoru continuous integration. S tím souvisí i důkladné otestování projektu jako celku s využitím technik quality assurance. V projektu též nejsou nijak komplexně řešená práva. Například zde není aktuálně možné nastavit, aby správce vozového parku měl práva například jen na určitou pobočku. Další možný návrh na zlepšení je tedy analýza a implementace komplexnější správy práv.

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

REST Representational state transfer

API Application Programming Interface

URL Uniform Resource Locator

WSGI Web Server Gateway Interface

HTTP Hypertext Transfer Protocol

CI Continuous Integration

Příloha **B**

Obsah přiloženého CD

```
readme.txt ..... stručný popis obsahu CD
src
    impl ..... zdrojové kódy implementace
    thesis ..... zdrojová forma práce ve formátu LaTeX
text ..... text práce
    thesis.pdf ..... text práce ve formátu PDF
    thesis.ps ..... text práce ve formátu PS
    zpp.pdf ..... zadání závěrečné práce
```

Literatura

[1] Wikipedia contributors. *Carsharing* [online]. 5. 3. 2015 [přístup 18. 4. 2015]. Dostupné z: http://cs.wikipedia.org/wiki/Sd%C3%ADlen%C3%AD_aut

[2] Nebeský, O.: *Rezervační systém carsharingové společnosti*. Diplomová práce, České vysoké učení technické, 2009.

[3] The PHP Group. *PHP* [software]. [přístup 18. 4. 2015]. Dostupné z: http://php.net

[4] Drupal Association. *Drupal* [software]. [přístup 18. 4. 2015]. Dostupné z: https://www.drupal.org

[5] Python Software Foundation. *Python* [software]. [přístup 18. 4. 2015]. Dostupné z: https://www.python.org

[6] The Django Software Foundation. *Django* [software]. [přístup 18. 4. 2015]. Dostupné z: https://www.djangoproject.com

[7] Vařecha, B. F.: *Transformace informačního systému pro carsharing do webové služby*. Diplomová práce, České vysoké učení technické, 2010.

[8] Wagner, J.: *Příprava systému autonapůl.cz na reálné nasazení*. Diplomová práce, České vysoké učení technické, 2012.

[9] Pokorný, P.: *Developing a geographic data management component for a car-sharing support information system*. Diplomová práce, České vysoké učení technické, 2013.

[10] Ječmínek, J.: *Developing an accounting component for a car-sharing support information system*. Diplomová práce, České vysoké učení technické, 2013.

[11] ABRA. *Flexibee* [software]. [přístup 18. 4. 2015]. Dostupné z: https://www.flexibee.eu

[12] Jesper Nøhr. *Django Piston* [software]. [přístup 18. 4. 2015]. Dostupné z: https://bitbucket.org/jespern/django-piston/wiki/Home

[13] Tom Christie. *Django REST Framework* [software]. [přístup 18. 4. 2015]. Dostupné z: http://www.django-rest-framework.org

[14] *Maven: the definitive guide*. Sebastopol, Calif: Oreilly, 2008, ISBN 978-0-596-51733-5.

[15] Moore, D.: *Professional Python frameworks Web 2.0 programming with Django and Turbogears*. Indianapolis, Ind: Wiley Pub, 2007, ISBN 9780470245347.

[16] Edgewall Software. *South* [software]. [přístup 18. 4. 2015]. Dostupné z: http://south.aeracode.org

[17] Mikowski, M.; Powell, J.: *Single page web applications: JavaScript end-to-end*. Shelter Island, NY: Manning, 2013, ISBN 9781617290756.

[18] Adobe. *Adobe Flash* [software]. [přístup 18. 4. 2015]. Dostupné z: http://www.adobe.com/cz/products/flashplayer.html

[19] Oracle. *Java Applet* [software]. [přístup 18. 4. 2015]. Dostupné z: https://java.com/

[20] Google. *AngularJS* [software]. [přístup 18. 4. 2015]. Dostupné z: https://angularjs.org

[21] Facebook Inc. *React* [software]. [přístup 18. 4. 2015]. Dostupné z: https://facebook.github.io/react/

[22] TILDE INC. *Ember* [software]. [přístup 18. 4. 2015]. Dostupné z: http://emberjs.com

[23] Jeremy Ashkenas. *Backbone.js* [software]. [přístup 18. 4. 2015]. Dostupné z: http://backbonejs.org

[24] The World Wide Web Consortium. *IndexedDB* [software]. [přístup 18. 4. 2015]. Dostupné z: http://www.w3.org/TR/IndexedDB/

[25] Google. *Two-way Data Binding* [online]. 22. 12. 20154 [přístup 18. 4. 2015]. Dostupné z: https://docs.angularjs.org/tutorial/step_04

[26] Stefan Penner, Robert Jackson. *Ember CLI* [software]. [přístup 18. 4. 2015]. Dostupné z: http://www.ember-cli.com

[27] Yehuda Katz. *Handlebars.js* [software]. [přístup 18. 4. 2015]. Dostupné z: http://handlebarsjs.com

[28] Jeremy Ashkenas. *CoffeeScript* [software]. [přístup 18. 4. 2015]. Dostupné z: http://coffeescript.org

[29] Dan Wheeler. *Dropbox dives into CoffeeScript* [online]. 13. 9. 2012 [přístup 18. 4. 2015]. Dostupné z: https://blogs.dropbox.com/tech/2012/09/dropbox-dives-into-coffeescript/

[30] Alex Matchneer. *Emblem.js* [software]. [přístup 18. 4. 2015]. Dostupné z: http://emblemjs.com

[31] The PostgreSQL Global Development Group. *PostgreSQL* [software]. [přístup 18. 4. 2015]. Dostupné z: http://www.postgresql.org

[32] Fowler, M.: *Patterns of enterprise application architecture*. Boston: Addison-Wesley, 2003, ISBN 0321127420.

[33] simplabs. *Ember Simple Auth* [software]. [přístup 18. 4. 2015]. Dostupné z: http://ember-simple-auth.com

[34] DockYard, Inc. *Ember Validations* [software]. [přístup 18. 4. 2015]. Dostupné z: https://github.com/dockyard/ember-cli-i18n

[35] DockYard, Inc. *Ember CLI i18n* [software]. [přístup 18. 4. 2015]. Dostupné z: https://github.com/dockyard/ember-cli-i18n

[36] Hudson Labs. *Jenkins CI* [software]. [přístup 18. 4. 2015]. Dostupné z: https://jenkins-ci.org