Sem vložte zadání Vaší práce.

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

Bakalářská práce

Modul hlášení a správy závad pro projekt Metrocar

Jan Nováček

Vedoucí práce: Ing. Martin Komárek

7. května 2015

Poděkování

Mé díky patří vedoucímu práce, Ing. Martinu Komárkovi, za jeho cenné rady, ochotu a pomoc při tvorbě této 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 7. května 2015 .....

Odkaz na tuto práci

Nováček, Jan. *Modul hlášení a správy závad pro projekt Metrocar*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Cílem práce bylo vytvořit modul hlášení a správy závad pro existující informační systém pro sdílení vozidel. Existující systém běží jako webová aplikace, aplikace je postavena na platformě Python a využívá webový framework Django. Modul měl podporovat hlášení závad vozidel a správu nahlášených závad přes webové rozhraní, modul měl být postaven na platformě Python/Django. Práce obsahuje stručnou rešerši způsobů nahlašování závad používaných vybranými společnostmi v ČR a v zahraničí, které nabízejí služby sdílení vozidel. Do práce je zahrnuta rešerše vybraných existujících modulů pro hlášení a správu závad na platformě Python/Django. Dále obsahuje analýzu, návrh a implementaci modulu samotného s využitím poznatků z rešerší. Modul je otestován, zdokumentován a nasazen. Výsledkem práce je funkční část pro stávající systém.

Klíčová slova hlášení závad, správa závad, ticket systém, helpdesk, python, django, sdílení vozidel

Abstract

Purpose of this work was to create a module for defect reporting and management for existing information system for car sharing. The existing system runs as a web application, application is built on Python platform and uses Django web framework. The module supports car defect reporting and management of reported defects through a web interface, module is built on Python/Django platform. The work includes a brief research on methods used for reporting defects by selected Czech and foreign companies that offer car sharing services. Work also includes evaluation of selected existing modules for reporting and managing defects on Python/Django platform. It also includes analysis, design and implementation of the module itself, using knowledge acquired from research. Furthermore, the module is tested, documented and deployed. The result is a functional part of the current system.

Keywords defect reporting, defect management, ticket system, helpdesk, python, django, car sharing

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Informační systém, vyvíjený na Fakultě elektrotechnické ČVUT pro modelovou společnost nabízející službu sdílení vozidel, postrádá jakýkoliv způsob hlášení a správy závad vozidel. Cílem této práce je doplnit systém o modul hlášení a správy závad, který tento nedostatek napraví.

Hlavním důvodem výběru tohoto tématu je jeho zasazení do oboru Softwarové inženýrství. V této práci bych si chtěl zopakovat proces vývoje software, od analýzy, přes návrh, implementaci, testování, až po dokumentaci a případné nasazení.

Dalším důvodem výběru tématu je obsáhlost systému. Informační systém ve finální podobě musí nabízet velké množství služeb, jmenovitě například rezervační systém, komunikaci s palubními počítači vozidel, práce s geografickými daty, systém plateb a mnoho dalšího. Chtěl jsem získat nové znalosti a zároveň tento systém rozšířit o další část.

Dále jsem si chtěl vyzkoušet práci na rozsáhlejším informačním systému, na kterém někdo přede mnou dělal, v nějakém stavu jej zanechal a já měl na práci navázat.

V teoretické části práce je stručný úvod do problematiky služby sdílení vozidel a informačního systému. Dále je stručné zpracování způsobů, jakými ostatní společnosti s podobným zaměřením řeší hlášení závad. Implementační platforma je dána zadáním, proto se část rešerše věnuje i existujícím řešením hlášení a správy závad, která jsou volně dostupná pro danou platformu.

Praktická část práce se sestává z jednotlivých kroků procesu vývoje software. Začíná analýzou, kde se zjišťuje, co si zákazník (zadavatel) od výsledného modulu představuje. Pokračuje návrhem, kde je stanoven způsob, jakým se výsledný modul vytvoří. Další kapitola je věnována implementaci, ve které se realizují plány z návrhu. Kvalita implementace je ověřena testováním na několika úrovních. Poslední kapitola praktické části práce se věnuje nasazení výsledného modulu.

V závěru je shrnutí odvedené práce a sada doporučení pro případné pokračovatele.

Kapitola 1

Cíle práce

Cílem práce je vytvořit modul pro informační systém postavený na platformě Python [1] s využitím webového frameworku Django [2].

K dosažení tohoto cíle je zapotřebí seznámit se s informačním systémem, pro který je modul určen. Také je potřeba se seznámit s platformou Python/Django, na které je postaven.

Dílčí cíle v rámci řešerše jsou zorientování se ve způsobu nahlašování závad ostatními společnosti se stejným zaměřením a získání přehledu o existujících modulech postavených na platformě Python/Django.

Při samotné tvorbě modulu je cílem projít procesem tvorby softwarového produktu od začátku do konce. Dílčími cíli procesu tvorby modulu jsou vytvoření analýzy, vypracování návrhu a implementace modulu. K ověření implementace se vytvoří testy. Dále se modul patřičně zdokumentuje a nakonec se nasadí.

Výsledný modul má být součástí webové aplikace, má být přístupný přes webové rozhraní. Základní služby, které má modul poskytovat uživatelům, jsou hlášení a správa závad vozidel a sledování oprav vozidel.

Kapitola 2

Bližší popis domény

V této kapitole je blíže popsána služba sdílení vozidel a informační systém, pro který má být v rámci této práce vytvořen modul hlášení a správy závad.

2.1 Sdílení vozidel

Sdílení vozidel je služba, jejímž prostřednictvím může firma nabízet klientům svá vozidla k osobnímu užití. Služba je podobná půjčovně vozidel, rozdíl je v decentralizaci dostupnosti vozidel a bližším vztahu firmy a zákazníka. Účelem této služby je poskytnout klientům vozidla k osobnímu užití. Cílovou skupinu zákazníků tvoří občasní řidiči, lidé, kterým by se vlastnictví vozidla zejména po finanční stránce nevyplatilo.

Následuje příklad, jak tato služba může v principu fungovat. Společnost nakoupí vozidla pro sdílení a každému vozidlu specifikuje domovské místo. V domovském místě bude vozidlo dostupné k vyzvednutí zákazníkem a na toto místo zákazník vozidlo vrátí. Zákazník si zarezervuje vozidlo na určitý čas přes rezervační systém dostupný přes webové rozhraní, nebo domluvou s operátorem přes telefon. V době rezervace může zákazník vozidlo využívat, musí jej však vrátit do konce rezervace zpět na domovské místo. Při pozdním vrácení vozidla může být zákazník potrestán, například zaplacením pokuty ve výši dle délky zpoždění. Standardní poplatky jsou pouze za ujeté kilometry a dobu rezervace, částky jsou specifikovány jako cena za jeden ujetý kilometr a cena za jednu hodinu rezervace.

V České republice se sdílením vozidel zabývá několik firem, za zmínku stojí družstevní Autonapůl [3], působící v Brně, Praze, Liberci a Plzni. Další český reprezentant je CAR4WAY a.s. [4], působící v Praze. Další zmínka patří AJO.CZ [5], provozovatel je Klimek Motion s.r.o, působí v Brně. Tyto firmy se ve velké míře drží příkladu uvedeného v předchozím odstavci, rozdíly jsou zejména v ceně a dostupnosti služby.

V Praze také působí sharujeme.cz [6], to poskytuje lidem možnost nabídnout vlastní vozidlo ke sdílení ostatním. U vozidla je většinou stanovena cena

za kilometr, u některých ale i cena za nájem na den, za přistavení vozidla na určité místo a podobně. Aktuálně jsou v nabídce vozidla starší 7 let. Jedná se spíše o prostředníka mezi člověkem, co chce nabídnout své vozidlo, a člověkem, který chce vozidlo využít.

Služba sdílení vozidel je i u nás známá pod anglickým překladem car sharing, někdy také psáno jako car-sharing nebo carsharing.

2.2 Informační systém

Modul, který má být v rámci této práce vytvořen, je určen pro informační systém, který slouží modelové společnosti nabízející službu sdílení vozidel.

2.2.1 Historie

Informační systém je vyvíjen pod názvem Metrocar [7] přibližně šest let na Fakultě elektrotechnické ČVUT. Prvotní verze byla vytvořena v rámci bakalářské práce [8] Ondřejem Nebeským v PHP [9] za použití systému Drupal [10].

Po střízlivém hodnocení projektu v rámci magisterské práce [11] Filipem Vařechou se rozhodlo o přechodu na platformu Python a webový framework Django. V rámci té samé práce se projekt předělal na novou platformu a obohatil o další funkce.

Dalším přispěvatelem byl Jan Wagner, který měl ve své bakalářské práci [12] systém dodělat, opravit a připravit jej pro nasazení.

Další větší obohacení systému přišlo s bakalářskou prací [13] Jakuba Ječmínka, který se postaral o vytvoření účetního modulu.

Petr Pokorný ve své magisterské práci [14] zapracoval na kvalitě projektu a jeho dokumentace, vedle toho vytvořil systém pro komunikaci s palubními jednotkami vozidel a správu geografických dat.

Dalšími přispěvateli projektu byli studenti předmětu Softwarové inženýrství na Fakultě elektrotechnické ČVUT, ti měli za úkol posunout systém opět trochu blíže ostrému nasazení [15].

2.2.2 Technický stav

V kapitole je popis technického stavu systému ze zimy roku 2014, kdy byla tato práce v raném vývoji.

Systém je vyvíjen v Pythonu verze 2. Verze frameworku Django je 1.5.4, na tuto verzi se přešlo v zimě 2014 z verze 1.4. Převážná většina zdrojového kódu je v anglickém jazyce, stejně tak dokumentace systému. Jazyk internetových stránek je nyní kombinovaný, část je v českém jazyce, část v anglickém. Plán byl a stále je, že budou 2 oddělené jazykové verze stránek, tj. jedna v anglickém jazyce, jedna v českém jazyce, přičemž volba jazyka bude na uživateli.

Nynější stav systému není dobrý. Využívá mnoho zastaralých modulů/balíčků třetích stran, v mnoha případech neaktuální verze, jejich užití je mnohdy

nejasně definováno. Závislost je také na starší verzi databáze a jejího rozšíření. I přes snahu předchůdců je dokumentace systému poněkud stručnější a někdy tajemná. Z důvodu starších verzí a stručné dokumentace byla příprava vývojového prostředí velice nezáživná a časově náročná.

Na systému pracuje několik lidí, lze očekávat velké změny projektu. Na změny budu upozorňovat na vhodných místech v průběhu práce.

Kapitola 3

Průzkum

Účelem kapitoly je seznámit se s problematikou hlášení a správy závad. Tato problematika je známá hlavně při vývoji software, proto jsou prozkoumány i systémy z tohoto odvětví, nejen z odvětví sdílení vozidel. V první části je rešerše způsobů, jakými firmy nabízející službu sdílení vozidel řeší hlášení závad svých vozidel. Další rešerše se týká volně dostupných modulů pro Python/Django, které nějakým způsobem nabízejí možnost hlášení a správy závad.

3.1 Způsoby hlášení závad car-sharingovými společnostmi

V této části kapitoly je stručný přehled toho, jak společnosti nabízející sdílení vozidel řeší hlášení závad. Vychází se zejména z materiálů zveřejněných na webových stránkách jednotlivých firem. Cílem je najít způsoby, kterými společnosti řeší případy, kdy zákazník nalezne na vozidle závadu a chce závadu nahlásit. Z nalezených způsobů řešení tohoto problému se zaměřím na ty, ve kterých se hlášení provádí prostřednictvím webového formuláře, protože hlášení tímto způsobem je ve výsledném modulu vyžadováno.

Z České republiky se podíváme na již zmíněné společnosti, tedy Autonapůl, Car4Way a.s., AJO.cz s provozovatelem Klimek Motion s.r.o a sharujeme.cz.

Družstvo Autonapůl podle svých Všeobecných obchodních podmínek [16] definuje Patrona vozidla, který se stará o dokumentování veškerých závad jemu přiřazeného vozidla. Ačkoliv Všeobecné obchodní podmínky nedefinují způsob komunikace a ani se nezmiňují o nějakém webovém formuláři, domnívám se, že preferovaný způsob komunikace zákazníka s Patronem vozidla je přes telefon. Jeví se to jako nejpravděpodobnější řešení. Každý Patron se stará o své vozidlo a podle podmínek je zákazník povinen nahlásit Patronovi nalezenou závadu pokud možno neprodleně.

Společnost Car4Way a.s. na svých internetových stránkách neinformuje

o tom, jak hlásit nalezené závady. Nenalezl jsem ani právní dokumenty, které by o tom pojednávaly. Nicméně z komunikace [17] mezi společností a zákazníkem na sociální síti se lze domnívat, že řeší hlášení závad telefonicky na asistenční linku.

AJO.cz, dle [18] a [19], řeší hlášení závad při protokolárním předání vozidla, kdy zákazník telefonicky komunikuje s dispečerem a případné závady mu hlásí.

Český zástupce sharujeme.cz funguje spíš jako prostředník mezi člověkem, co chce nabídnout své auto ke sdílení, a člověkem, který chce vozidlo využít. Na jejich internetových stránkách [6] jsem nenašel nic o hlášení závad. Lze předpokládat, že v případě závady na vozidle se domluví uživatel s vlastníkem osobně, telefonicky, nebo e-mailem. Předpoklad vychází z toho, že sharujeme.cz je pouze prostředník, nabízející prostor pro nabídku vozidel ke sdílení, nic víc.

Ze zahraničních zástupců jsem vybral společnost GoCar [20] z Irska. Při nalezení závady je zákazník povinen informovat telefonicky společnost s použitím konzole ve vozidle. Pokud se nebude moci dovolat, má poslat zprávu. Opět není zmínka o webovém formuláři.

Z přehledu vyplývá, že společnosti řeší hlášení závad většinou telefonicky. Tento způsob je celkem pochopitelný, telefon má v dnešní době téměř každý a vytočit jedno telefonní číslo je relativně jednoduchý úkon. Nepodařilo se najít společnost, která by hlášení závad vozidel řešila skrze webový formulář. Telefon je nejspíš preferován ještě z dalšího důvodu, při hovoru se může operátor společnosti zeptat zákazníka na všechny důležité informace, zatímco při vyplňování formuláře může zákazník důležitou informaci opomenout. Telefon lze stále považovat za dostupnější než připojení k internetu a pro zákazníka je hovor jistě pohodlnější než vyplňování formuláře.

3.2 Veřejně dostupné balíčky pro Django

Framework Django má početnou komunitu vývojářů a část z nich nabízí své produkty zdarma. Účelem této části kapitoly je výběr vhodného produktu, který by se dal použít k implementaci modulu do systému Metrocar. Vedle toho je také účelem dozvědět se více o možných řešení problematiky hlášení a správy závad při použití frameworku Django.

3.2.1 Publikované balíčky

Na [21] lze nalézt balíčky s moduly, aplikacemi nebo celými systémy ve frameworku Django. V uvedené databázi se nachází seznam balíčků se zaměřením na hlášení a správu závad [22].

Seznam jednotlivých balíčků a jejich zaměření:

- Django-Knowledge [23] — Jednoduchý modul. Forma otázek a odpovědí, neobsahuje složitější hlášení a jejich správu.

- Django-TODO [24] — Jednoduchý modul hlášení a správy závad.
- Django-Helpdesk [25] — Obecně zaměřený modul hlášení a správy závad. Lze použít jako modul i jako samostatnou aplikaci.
- DONER [26] — Aplikace pro hlášení a správu závad.
- YATS [27] — Komplexní systém hlášení a správy závad, zaměřený na vývoj softwaru.
- Mr. Wolfe [28] — Aplikace pro hlášení a správu závad. Zaměřuje se na dodržování SLA [29].
- ITSY [30] — Komplexní systém správy vývoje softwaru, zahrnuje hlášení a správu závad.

Z uvedeného seznamu se vyberou vhodní kandidáti na základě následujících kritérií:

- Správa hlášení — Manipulace s hlášeními, přístup k nim, dělení.
  - Vyhledávání — Vyhledávání hlášení dle parametrů. Vyžaduje se alespoň zobrazení všech hlášení.
  - Přidělování — Uživatel s určitými privilegii může přidělit hlášení jinému uživateli, nebo si může uživatel sám sobě přidělit hlášení.
  - Kategorizace — Základní skupiny hlášení. Není nutné, ale vítané.
- Dokumentace — Kvalita a množství dokumentace. Ve výběru má největší váhu.
  - Množství — Množství dostupné dokumentace.
  - Věcnost — Použitelnost dokumentace.
- Nároky na systém — Vyžaduje se podpora Pythonu 2 a Djanga 1.5. Speciální nároky nejsou příliš vítány, ale záleží na jejich složitosti a problémech kolem případného použití.
  - Podpora Django 1.5 — Vyžadováno.
  - Podpora Python 2 — Vyžadováno.
  - Speciální nároky na databázi — Speciální databáze, speciální operace. Není vítáno.
  - Speciální nároky na systém — Dodatečná systémová konfigurace. Není příliš vítané.
- Ostatní

  - Datum poslední aktualizace — Kdy naposledy byl balíček aktualizován.
  - Licence — Pod jakou licencí je modul vydán. Vyžaduje se licence umožňující úpravu a volné užití kódu.
  - Django admin — Podpora Django admin. Nevyžaduje se, ale je velice vítána.
  - Lokalizace — Podpora lokalizace, nebo alespoň připravená kostra.
  - Podpora e-mailu — Modul pracuje s e-mailem, umožňuje alespoň odesílání e-mailů.
  - Testy — Prokázání správné funkčnosti modulu. Nevyžadováno, vítáno.

Na základě Tab. 3.1 lze vyřadit následující moduly:

- DONER - Nepodporuje Django verze 1.5, vyžaduje verzi 1.6.5.
- YATS - Nepodporuje Django verze 1.5, jedny z posledních úprav byly pro Django verze 1.6.6.
- ITSY - Nepodporuje Django verze 1.5, vyžaduje verzi 1.6.2.
- Knowledge - Více než rok a půl od poslední aktualizace. Nepodporuje lokalizaci. Pouze systém otázka-odpověď.

Zbývají moduly Helpdesk, TODO a Mr. Wolfe. Z porovnání těchto tří vychází nejlépe Helpdesk, důvody jsou následující:

- Helpdesk lze použít přímo jako modul, je tak nastaven a vytvořen. Helpdesk dokáže pracovat s upravenými třídami, které reprezentují uživatele. Mr. Wolfe je v základu samostatná aplikace a vytváří vlastní třídu pro uživatele.
- Helpdesk je oproti TODO obsáhlejší a obecnější. Helpdesk počítá s tím, že hlášení mohou posílat zákazníci. TODO míří na přidělování úkolů (hlášení) v rámci týmu.
- Helpdesk podporuje přílohy ve formě souborů. Přílohy aktuálně nejsou v požadavcích, ale v budoucnu by se mohlo zavést, že se závada na vozidle vyfotí a přiloží k hlášení.
- Helpdesk podporuje lokalizaci, TODO ani Mr. Wolfe nikoliv.

Jediné, co mluví proti Helpdesku, je počet testů. Obsahuje jich jen několik, ale tento nedostatek lze odstranit.

Požadavek	*Knowledge*	*TODO*	*Helpdesk*	*DONER*	*YATS*	*Mr. Wolfe*	*ITSY*
Správa hlášení							
Vyhledávání	Ano	Ano	Ano	Ne	Ano	Ano	Ano
Přidělování	Ne	Ano	Ano	Ano	Ano	Ano	Ano
Kategorizace	Ano	Ano	Ano	Ano	Ano	Ano	Ano
Dokumentace							
Množství	Velké	Malé	Velké	Malé	Malé	Velké	Velké
Věcnost	Dobrá	Dobrá	Dobrá	Špatná	Špatná	Dobrá	Dobrá
Nároky na systém							
Podpora Django 1.5	Ano	Ano	Ano	Ne	Ne	Ano	Ne
Podpora Python 2	Ano	Ano	Ano	Ano	Ano	Ano	Ano
Speciální nároky na databázi	Ne	Ne	Ne	Ne	Ne	Ne	Ne
Speciální nároky na systém	Ne	Ne	Ne	Ne	Ano	Ano	Ano
Ostatní							
Poslední aktualizace	20.9.2013	11.12.2014	22.2.2015	28.10.2014	8.12.2014	17.2.2015	9.11.2014
Licence	BSD	BSD	BSD	MIT	MIT	Beer-ware	GPL
Django admin	Ano	Ano	Ano	Ano	Ano	Ano	Ano
Lokalizace	Ne	Ne	Ano	Ano	Ano	Ne	Ne
Podpora e-mailu	Ano	Ano	Ano	Ano	Ano	Ano	Ano
Testy	Ano	Ne	Ano	Ne	Ne	Ano	Ne

Tabulka 3.1: Vlastnosti modulů

3.3 Bugzilla, JIRA a podobné systémy

Výsledný modul má být co se účelu týče podobný softwarovým produktům jako Bugzilla nebo JIRA, respektive jejich částem. Tyto produkty jsou na poli hlášení a sledování řešení chyb zejména při vývoji a údržbě softwaru známé a využívané, proto považuji za vhodné se o nich zmínit.

Jistě by se jeden z těchto systémů dal nasadit vedle aplikace Metrocar, ovšem samotný požadavek na vytvoření modulu do aplikace Metrocar toto řešení znemožňuje.

Nicméně i kdyby požadavek na vytvoření modulu neexistoval, dalším z důvodů, proč toto řešení nepřipadá v úvahu, je nynější rozsáhlost aplikace Metrocar. Přestože Petr Pokorný ve své diplomové práci zredukoval počet závislostí a balíčků, instalace je stále, zejména kvůli neaktuálním verzím balíčků, nesnadná. Cílem práce není rozšířit seznam závislostí o celý další externí systém. Další problémy mohou nastat s provázáním hlášení, uživatelů a vozidel a s komunikací mezi systémy.

Kapitola 4

Analýza modulu pro hlášení a správu závad

Kapitola obsahuje analýzu modulu pro hlášení a správu závad pro informační systém společnosti, nabízející službu sdílení vozidel. Analýza je zpracována formou textu a UML [31] diagramů, diagramy jsou vytvořeny v aplikaci Enterprise Architect [32]. V elektronické příloze je přiložen soubor projektu pro Enterprise Architect, který obsahuje vedle diagramů i přesnější popis entit. Tento text a zmíněný projekt se vzájemně doplňují.

Požadavky na modul a obecně zadání pochází od zadavatele práce.

4.1 Business doménový model

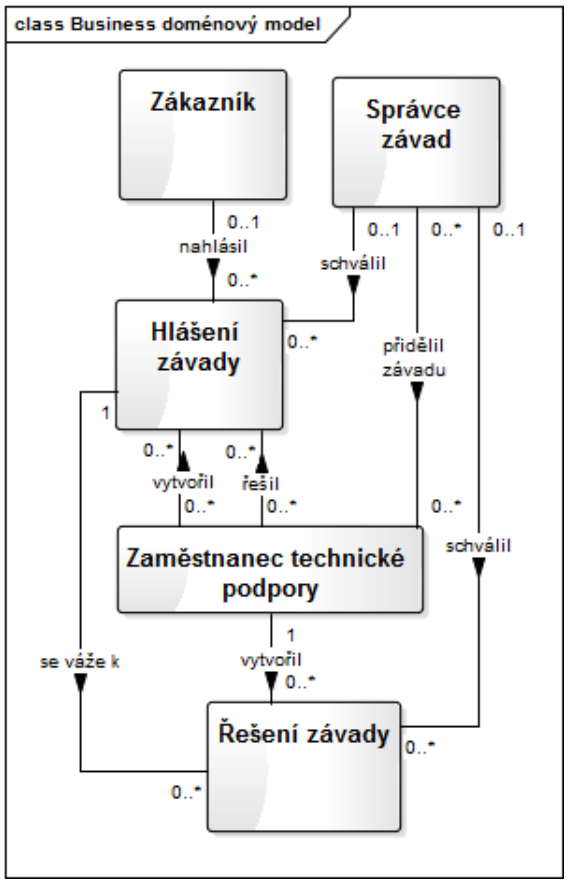
V této sekci je popsán systém z obecného pohledu. Závada je zatím brána obecně s nespecifikovaným předmětem závady.

V systému bude entita, která bude reprezentovat nahlášenou závadu, tedy hlášení závady. Hlášení závady může vytvořit zákazník, nebo to za něj může udělat zaměstnanec firmy, například technik nebo operátor.

Zejména v případě nahlášení závady zákazníkem může dojít k chybnému hlášení, ať už se jedná o nedostatek informací, nebo o nahlášení závady, která vlastně závadou není. Proto by měl někdo vyhodnocovat validitu hlášení, například správce závad.

Validní závada by se měla vyřešit, o řešení se postarají technici. O tom, který technik vyřeší kterou závadu, může rozhodnout správce závad, který určitou závadu přidělí určitému technikovi s požadovanými schopnostmi. Pokud ovšem technik nedokáže vyřešit přidělenou závadu, může se řešení vzdát. Vzhledem k možnému růstu společnosti a rozšíření nabídky vozidel si sám technik bude moci vzít závadu k řešení.

Řešení závady technikem nemusí být nutně správné, o kontrolu řešení se opět postará správce závad.



Obrázek 4.1: Business doménový model

Vztahy mezi entitami jsou znázorněny na Obr. 4.1.

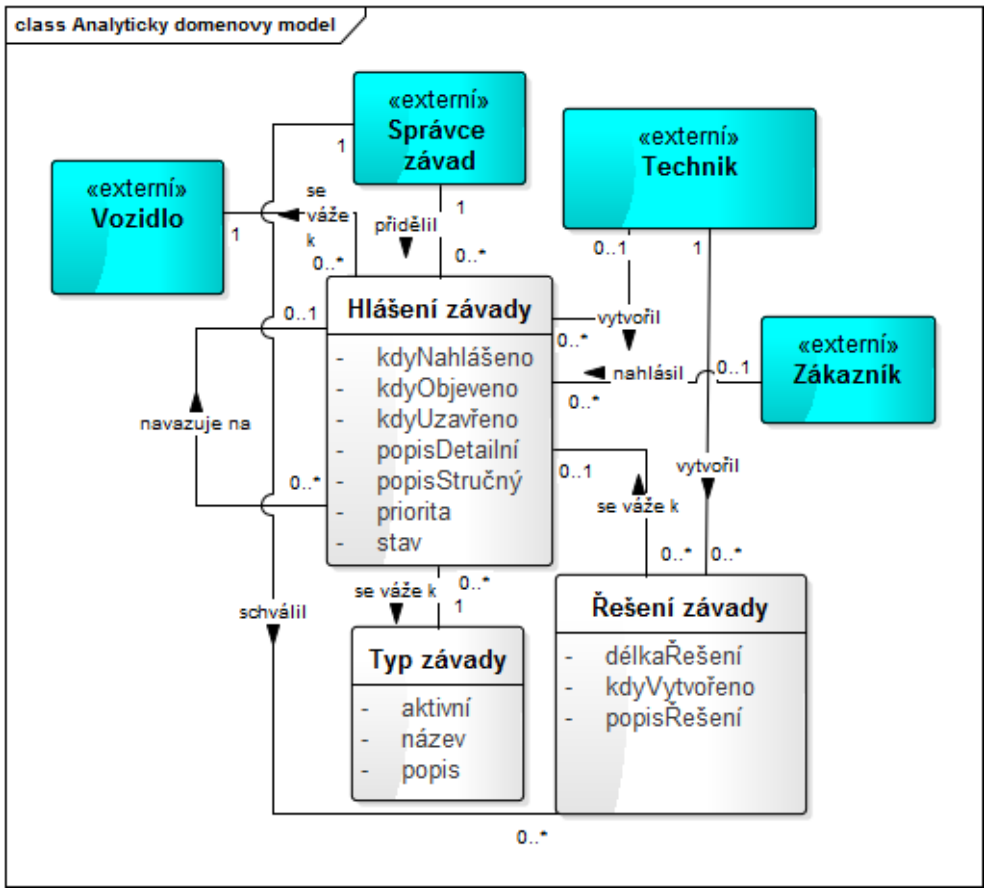
4.2 Analytický doménový model

V přechozí kapitole byly vyjmenovány stěžejní entity systému, v této kapitole je jejich bližší popis.

Informační systém již obsahuje entity pro uživatele a vozidla, v diagramu jsou označeny stereotypem externí. Entita pro uživatele bude reprezentovat všechny osoby v systému, tj. zákazník, technik a správce závad. Entita vozidla se bude vázat k hlášení závady daného vozidla.

Následuje bližší popis entit hlášení závady, řešení závady a typ závady:

- Hlášení závady — Reprezentuje ohlášenou závadu. Obsahuje její stručný a detailní popis, kdo závadu nahlásil a kdo vložil hlášení do systému, typ



Obrázek 4.2: Analytický doménový model

(kategorii) závady. Evidovat se také bude čas objevení závady, čas nahlášení závady a uzavření (vyřešení) hlášení.

- Řešení závady — Reprezentuje řešení jedné závady. Řešení obsahuje popis, jak se závada odstranila. Řešení může být rozděleno do více kroků, proto může mít jedno hlášení závady vícero řešení. Technik při vytvoření řešení také zaznamená, kolik hodin strávil řešením závady.

- Typ závady — Reprezentuje typ závady. Obsahuje unikátní název typu, jeho popis a příznak aktivity. Příznak aktivity určuje, zda je možné vytvářet nové hlášení závady s tímto typem.

Vztahy mezi entitami včetně jejich atributů jsou znázorněny na Obr. 4.2.

4.2.1 Stavový diagram hlášení

Pro lepší pochopení celého procesu opravy závady, tj. od nahlášení závady až po schválení řešení a uzavření závady, byl vytvořen stavový diagram hlášení a řešení závady. Ten popisuje stavy, ve kterých se může závada nacházet, a přechody, včetně podmínek a akcí, které přechod do jiného stavu podmiňují a doprovázejí.

V této části sekce je velká část věnována studiu stavových diagramů závad, používaných v jiných systémech. Cílem bylo lépe pochopit problematiku životního cyklu závady a vyvarovat se případných chyb v návrhu stavového automatu hlášení a řešení závady.

Mezi prvními analyzovanými stavovými automaty hlášení a řešení závady jsou stavy používané v již zmíněných systémech Bugzilla a JIRA.

Hlášení, resp. řešení závady, se v systému Bugzilla řídí stavy popsanými na [33]. V Tab. 4.1 jsou detailněji rozebrány stavy.

Stav hlášení	*Přechody do:*	*Popis*
Nepotvrzeno	Nové, Vyřešeno	Nové hlášení závady, čeká na validaci.
Nové	Přiřazeno	Závada potvrzena, čeká se na řešitele, nebo může být závada přímo označena za vyřešenou.
Přiřazeno	Nové, Vyřešeno	Závada v řešení. Řešitel se může vzdát a označit závadu za novou, nebo závadu vyřešit.
Znovuotevřeno	Přiřazeno, Vyřešeno	Již jednou vyřešený problém se znovu objevil, nebo samotné řešení bylo špatné. Problém může být znovu přiřazen řešiteli, nebo rovnou označen za vyřešený.
Vyřešeno	Zkontrolováno, Znovuotevřeno	Problém vyřešen a čeká na kontrolu. Řešení může být přijato a problém se tím uzavře (stav Zkontrolováno), nebo je řešení nepřijatelné a problém je znovu otevřen.
Zkontrolováno	Znovuotevřeno	Řešení problému bylo přijato. Pokud se ovšem stejný problém objeví znovu, může být znovu otevřen.

Tabulka 4.1: Stavy závady v systému Bugzilla

Hlášení se nejprve vyhodnotí, relevantní lze přiřadit řešiteli, irelevantní

se označí za vyřešené. Relevantní hlášení si tedy vybere řešitel, nahlášený problém vyřeší a označí jej za vyřešený. Takto označené řešení problému je zkontrolováno a buď je přijato a označeno za zkontrolované, nebo odmítnuto a problém se znovu otevře. Vyřešený a zkontrolovaný problém se může znovu otevřít v případě, že se problém objevil znovu.

Systém JIRA se standardně řídí stavy popsanými v oficiální dokumentaci [34]. Stavy jsou detailněji popsány v Tab. 4.2. Hlášení problému je otevřeno do doby, než jej začne někdo řešit, poté se jeví jako v řešení. Po vyřešení problému se čeká na kontrolu, přijatelné řešení problém uzavře, nepřijatelné řešení problém znovu otevře. Pokud se problém v budoucnu objeví, hlášení problému se znovu otevře.

Stav hlášení	*Přechody do:*	*Popis*
Otevřeno	V řešení, Vyřešeno	Nový problém, zatím se validuje.
V řešení	Otevřeno, Vyřešeno, Uzavřeno	Závada je v řešení.
Vyřešeno	Znovuotevřeno, Uzavřeno	Problém vyřešen, čeká se na kontrolu. Přijatelné řešení problém uzavře, nepřijatelné znovu otevře.
Znovuotevřeno	V řešení, Vyřešeno, Uzavřeno	Problém se objevil znovu, opět se musí zvalidovat.
Uzavřeno	Znovuotevřeno	Řešení problému bylo přijato. Pokud se ovšem stejný problém objeví znovu, může přejít do Znovuotevřeno.

Tabulka 4.2: Stavy závady v systému JIRA

Další poměrně známý systém pro sledování chyb je Trac [35], přesněji jeho rozšíření Trac Tickets [36]. Diagram [37], rozebraný v Tab. 4.3, je hodně stručně pojatý. Nové hlášení se ohodnotí, relevantní jsou přiřazeny a řešeny, po vyřešení se uzavřou. Uzavřené se mohou znovu otevřít.

Systém Trac má ještě rozšířenou verzi [38], ta je rozebrána v Tab. 4.4, začíná hodnocením hlášení závady, relevantní se přiřadí řešiteli, irelevantní je považováno za vyřešené. Vyřešené se zkontroluje a v případě správného řešení se uzavře, jinak se znovu řeší. Uzavřené hlášení se v případě objevení stejné závady znovu otevře.

Další stavový diagram [39] je od PrimaryGoals [40], orientuje se na vývoj software. Rozebrán po stavech v Tab. 4.5 (část validace hlášení a tvorby řešení) a Tab. 4.6 (část testovací). Soustředí se zejména na kontrolní a testovací

Stav hlášení	*Přechody do:*	*Popis*
Nový	Přiřazeno (Znovuotevřeno), Uzavřeno	Nové hlášení závady.
Přiřazeno (Znovuotevřeno)	Nový, Uzavřeno	Hlášení závady je přiřazeno řešiteli, resp. hlášení závady popisuje již řešenou závadu a musí se vyřešit znovu.
Uzavřeno	Přiřazeno (Znovuotevřeno)	Závada vyřešena.

Tabulka 4.3: Stavy závady v systému Trac

Stav hlášení	*Přechody do:*	*Popis*
Nový	Přiřazeno, Vyřešeno	Nové hlášení závady.
Přiřazeno	Nový, Vyřešeno	Hlášení závady je přiřazeno řešiteli.
Vyřešeno	Ověřeno, Znovuotevřeno, Uzavřeno	Závada vyřešena, postupuje ke kontrole.
Ověřeno	Vyřešeno, Znovuotevřeno, Uzavřeno	Řešení závady bude buď přijato, nebo odmítnuto.
Znovuotevřeno	Přiřazeno, Vyřešeno	Nevyhovující oprava závady, resp. závada se vyskytla znovu.
Uzavřeno	Vyřešeno, Znovuotevřeno	Závada vyřešena.

Tabulka 4.4: Stavy závady v systému Trac, rozšířená verze

fáze, ty jsou zastoupeny celkem třemi stavy. Hlášení je nejprve ve stavu, kdy se buď podá žádost na další informace, nebo se vyhodnotí relevance nahlášené závady. Pokud se jedná o závadu, přiřadí se řešiteli. Dále se řešení předá testerům, řešení se otestuje, případně si tester získá další informace k testům. Po úspěšném průchodu testy se čeká na finální hodnocení řešení.

Stav hlášení	*Přechody do:*	*Popis*
Nový	Otevřeno, Nejasný popis, Uzavřeno	Nové hlášení závady, probíhá validace.
Otevřeno	Nejasný popis, Opravená verze, Kontrola řešení	Nahlášena je skutečně závada. Mohou být potřeba další informace (přechod do Nejasný popis), může dojít k nápravě (Opravená verze, čeká na testera) nebo může dojít k testování (Kontrola řešení).
Nejasný popis	Otevřeno, Uzavřeno	Požaduje se doplnění hlášení. Po doplnění dochází opět k validaci.
Opravená verze	Kontrola řešení	Verze systému s opravou chyby, čeká na testera.
Uzavřeno	Otevřeno	Závada vyřešena. Pokud se objeví znovu, může dojít k otevření (stav Otevřeno).

Tabulka 4.5: Stavy závady podle PrimaryGoals, část validace hlášení a tvorby řešení

Diagram [41] dle BEA Systems, Inc. [42] se orientuje na chybu v softwarových produktech. Rozebrán po stavech je v Tab. 4.7. Oproti jiným má poněkud nezvyklou posloupnost stavů. Hlášení se nejprve přiřadí řešiteli, vyhodnotí se relevance hlášení a v případě, že je nahlášená závada relevantní, se řeší a teprve poté se případně získávají další informace od toho, kdo hlášení podal. Na diagramu je také zajímavé, že z téměř každého stavu může dojít k uzavření řešení problému zákazníkem.

Diagram [43] podle MSDN [44], stavy rozebrány v Tab. 4.8, je podobný prvnímu analyzovanému diagramu Trac Tickets (popsán v Tab. 4.3), také se orientuje na vývoj software. Nové hlášení se označí za aktivní, vyhodnotí se, relevatní se řeší, irelevantní uzavřou. Hlášení se stává vyřešeným, pokud jej řešitel označí za vyřešené a pokud projdou testy. Pokud projde řešení testováním, uzavře se, jinak přechází zpět do aktivního stavu. Uzavřená hlášení se mohou znovuotevřít.

Z předchozího popisu vybraných stavových diagramů hlášení závad lze vy-

Stav hlášení	*Přechody do:*	*Popis*
Kontrola řešení	Otevřeno, Částečně otestováno, Nejasný test, Uzavřeno	Řešení závady čeká na kontrolu. Špatným řešením se problém dostane zpět do otevřeného stavu. Testy mohou být v pokročilém stádiu, ještě však nedokončeny (Částečně otestováno). Testy mohou vyžadovat více informací (Nejasný test). Řešení může být přijato a problém se tím uzavře.
Částečně otestováno	Otevřeno, Uzavřeno	Řešení závady bylo z větší části otestováno, tento stav slouží pro složité problémy. Pokud projdou i zbylé testy, pak se problém uzavře, jinak se vrací zpět do otevřeného stavu.
Nejasný test	Kontrola řešení	Řešení závady nelze otestovat, čeká se na doplnění informací nutných pro vznik testů. Po doplnění se může zkontrolovat řešení.

Tabulka 4.6: Stavy závady podle PrimaryGoals, část testovací

číst tři důležité milníky v životnosti hlášení. Prvním je přijetí hlášení, druhým řešení a třetím kontrola. Na základě těchto poznatků vznikl životní cyklus hlášení, který je uzpůsoben cílové doméně.

Životní cyklus hlášení začíná hodnocením, kdy je potřeba rozhodnout o relevanci nahlášené závady a případně požádat o další informace o závadě toho, kdo závadu nahlásil. Druhý krok se zabývá řešením nahlášené závady řešitelem. Třetím krokem je kontrola řešení správcem a uzavření hlášení. Pokud se již popsaná závada objeví znovu, může se hlášení dostat zpět na začátek do bodu validace hlášení. Na základě tohoto postupu vznikl výsledný diagram Obr. 4.3 se stavy popsanými v Tab. 4.9. V Tab. 4.10 je zdůvodněno, proč se stavy, které byly ve zmíněných stavových diagramech, nevyskytují ve finálním diagramu. Stavy stejného nebo velmi podobného významu jsou sloučeny.

Stav hlášení	*Přechody do:*	*Popis*
Otevřeno	Přiřazeno, Uzavřeno	Nové hlášení závady, k přidělení řešiteli. Problém může uzavřít sám zákazník.
Uzavřeno	-	Závada vyřešena.
Přiřazeno	Přečteno, Uzavřeno	Hlášení závady je přiřazeno řešiteli. Problém může uzavřít sám zákazník.
Přečteno	Přijato, Odmítnuto, Uzavřeno	Přiřazenou závadu si řešitel přečetl a přijal, nebo zamítl. Problém může uzavřít sám zákazník.
Odmítnuto	-	Řešitel označil závadu jako irelevantní.
Přijato	Přijato, Zpětná vazba, Uzavřeno	Řešitel závadu přijal k řešení. Přechodem do stavu Přijato může dojít k záznamu dílčího kroku řešení. Může se od zákazníka požadovat zpětná vazba. Problém může uzavřít sám zákazník.
Zpětná vazba	Přijato, Uzavřeno	Je požadována zpětná vazba od zákazníka, po doplnění jde zpět řešiteli (Přijato). Problém může uzavřít sám zákazník.

Tabulka 4.7: Stavy závady podle BEA Systems, Inc.

Stav hlášení	*Přechody do:*	*Popis*
Aktivní	Vyřešeno, Uzavřeno	Nové hlášení závady se řeší. Buď se problém vyřeší a projdou základní testy (Vyřešeno), nebo problém není třeba řešit (Uzavřeno).
Vyřešeno	Aktivní, Uzavřeno	Závada vyřešena, řešení se kontroluje. Pokud projde, problém se uzavře, jinak se stane opět aktivním.
Uzavřeno	Aktivní	Závada vyřešena. Pokud se objeví, znovu se otevře.

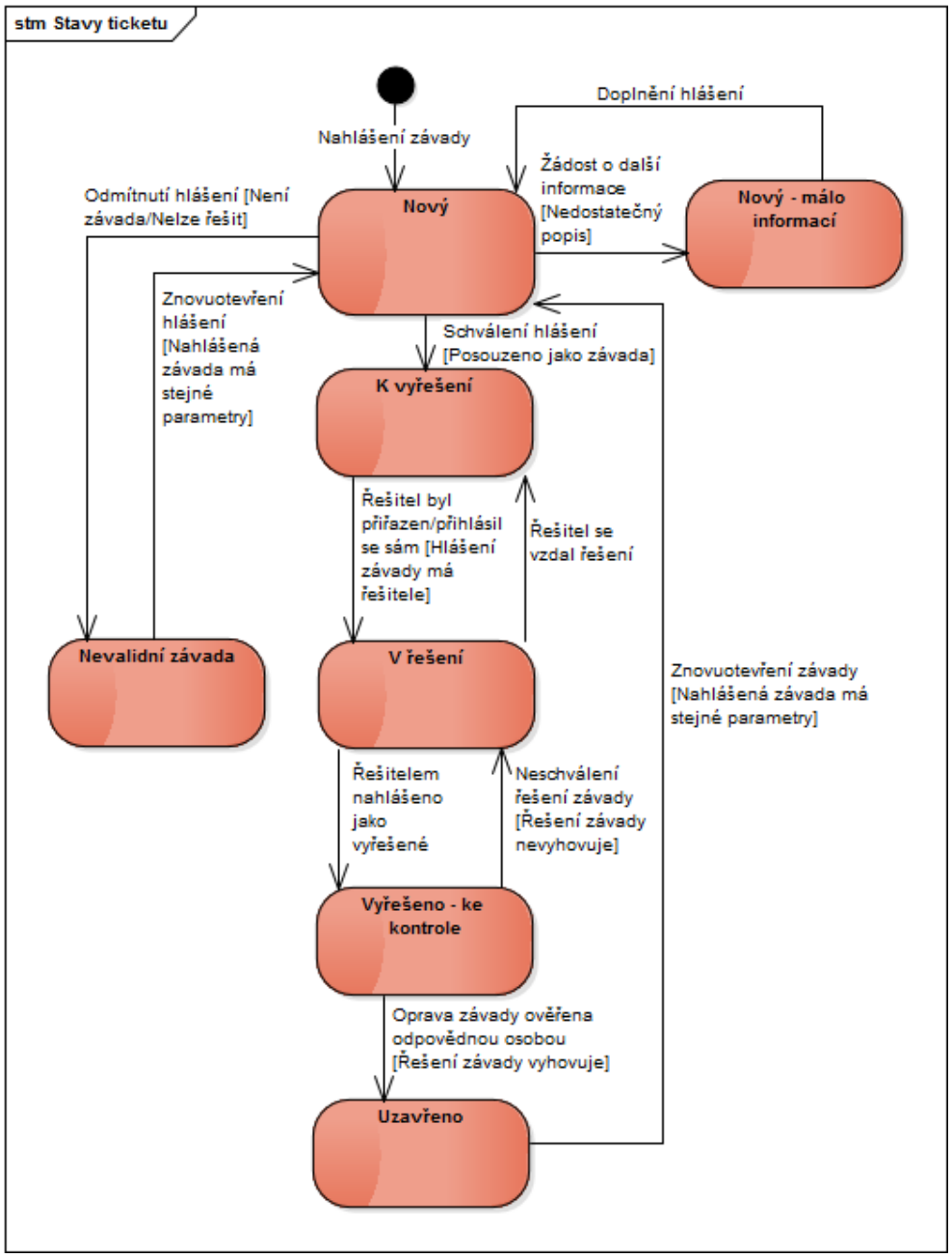
Tabulka 4.8: Stavy závady dle MSDN

Stav hlášení	*Popis stavu*
Nové	Nové hlášení závady.
Nové - málo informací	Popis u nového hlášení je nedostačující, je potřeba doplnit další informace.
K vyřešení	Nové hlášení obsahuje popis relevantní závady, zatím není řešeno, není nikomu přiděleno.
V řešení	Hlášení přiděleno, závada je v řešení.
Vyřešeno - ke kontrole	Závada je pravděpodobně vyřešena, čeká se na kontrolu, jestli tomu opravdu tak je.
Uzavřeno	Řešení přijato. Nahlášená závada je aktuálně opravena.
Nevalidní závada	Nahlášená závada není závadou.

Tabulka 4.9: Stavy použité v systému

Stav hlášení	*Zdroj*	*Důvod odmítnutí*
Částečně zkontrolováno	[39]	Granularita systému nevyžaduje více stavů na kontrolu.
Otevřené	[39]	Po dlouhém zvažování bylo rozhodnuto o dostatečnosti jediného stavu, který reprezentuje Otevřený a zároveň Nový stav.
Nejasný test	[39]	Kontrola řešení závady buď projde, nebo ne, nehledě na testy. Zváženo jako zbytečný stav pro analyzovaný systém.
Opravená verze	[39]	Ekvivalent ke stavu Vyřešené - ke kontrole.
Přiřazeno/Znovuotevřeno	[37]	Sdružuje v sobě moc příznaků.
Znovuotevřeno	[38]	Ve výsledném stavovém diagramu se bude tvářit jako nové hlášení.
Aktivní	[43]	Ekvivalent stavu V řešení.

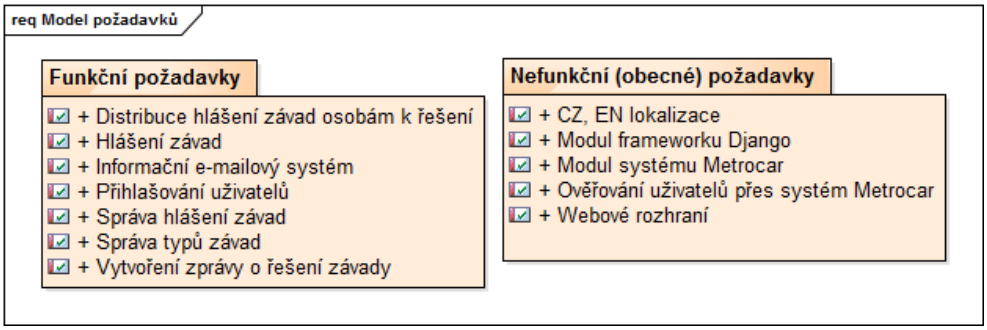
Tabulka 4.10: Stavy nepoužité



Obrázek 4.3: Stavový diagram hlášení závad

4.3 Analýza požadavků

V kapitole jsou specifikovány požadavky kladené na výsledný modul. Požadavky jsou rozdělené na funkční (co má systém umět) a nefunkční (co musí splňovat, technické parametry). Požadavky jsou shrnuty na Obr. 4.4. Bližší popis požadavků je dále v této sekci.



Obrázek 4.4: Model požadavků

4.3.1 Funkční požadavky

V této sekci jsou vyjmenovány funkční požadavky na modul.

- Distribuce hlášení závad osobám k řešení — Modul bude umožňovat přiřazování hlášení závad technikům k řešení. Technik si zároveň bude moci sám vzít hlášení závady k řešení.
- Hlášení závad — Modul umožní vytvořit hlášení závady a jejich úpravu skrze formulář na webových stránkách. Nebude umožňovat smazání hlášení.
- Informační e-mailový systém — Modul bude umožňovat informování uživatelů systému prostřednictvím e-mailu. Informovat bude o změnách stavu hlášeních souvisejících s daným uživatelem. Uživatel si bude moci zvolit, zda chce být informován.
- Přihlašování uživatelů — Modul bude umožňovat přihlášování uživatelů přes uživatelské jméno a heslo.
- Správa hlášení závad — Modul bude poskytovat základní funkce správy závad (vytvoření, úprava) a manipulaci s nimi (změna stavu).
- Správa typů závad — V modulu bude možno specifikovat nové typy závad. Typy závad nelze smazat. Typům závad lze nastavit viditelnost ve formuláři pro vytvoření nového hlášení.

- Vytvoření zprávy o řešení závady — Modul bude umožňovat vytvoření zprávy o řešení závady.

O registraci nových uživatelů se stará samotná aplikace Metrocar.

4.3.2 Nefunkční požadavky

Zde jsou uvedeny nefunkční požadavky na systém, technické parametry, atd.

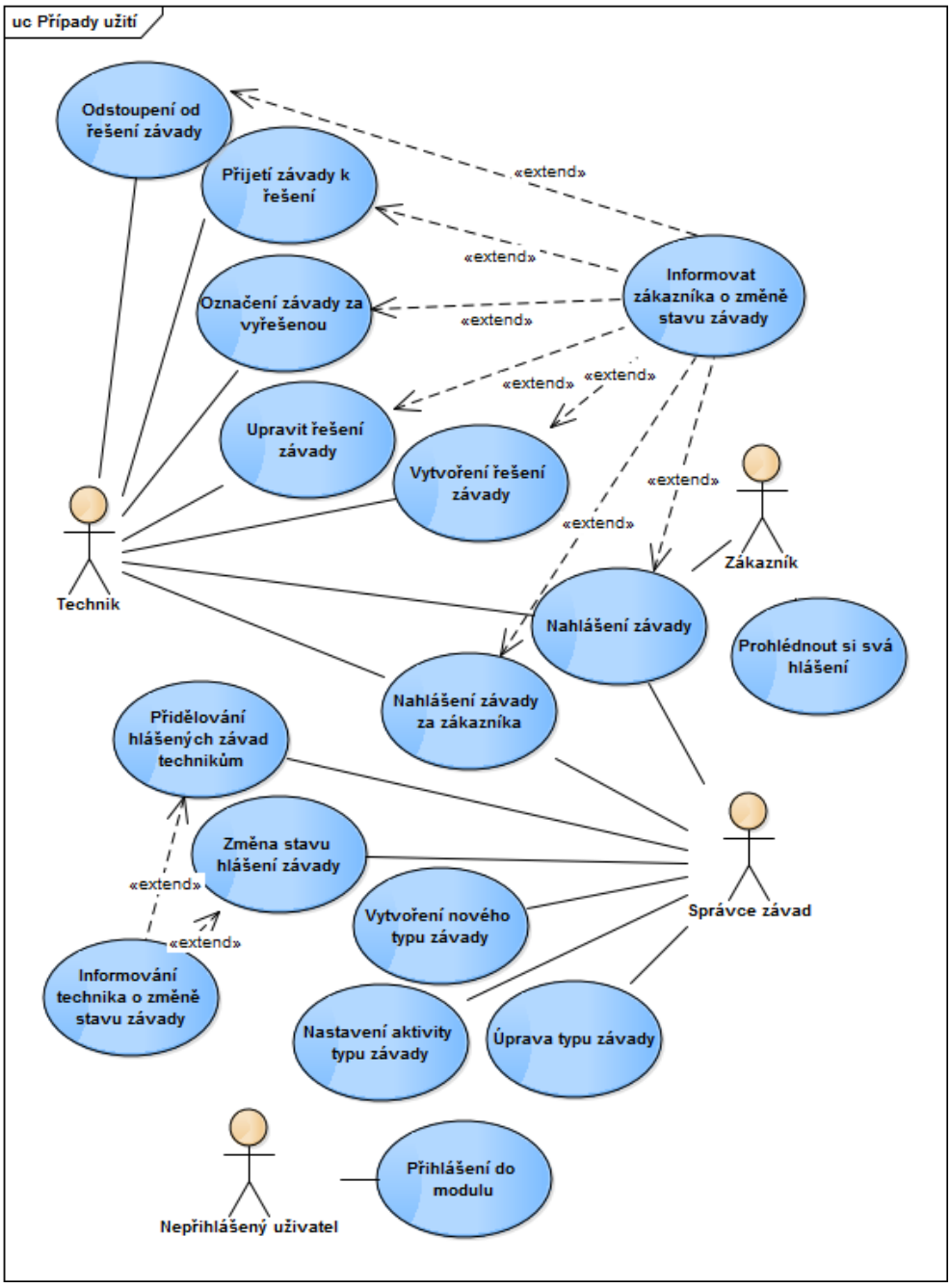
- CZ, EN lokalizace — Webové rozhraní modulu má být dostupné v českém a anglickém jazyce.
- Modul frameworku Django — Výsledný produkt bude vytvořen jako modul pro webový framework Django.
- Modul systému Metrocar — Výsledný produkt bude vytvořen jako modul, který lze integrovat do systému Metrocar.
- Ověřování uživatelů přes systém Metrocar — Modul bude k ověřování uživatelů (např. při přihlašování) využívat systém Metrocar.
- Webové rozhraní — Do modulu se bude přistupovat přes síť prostřednictvím webového prohlížeče.

4.4 Účastníci, uživatelé

- Zákazník — Využívá služby společnosti, v informačním systému je úspěšně zaregistrován.
- Technik — Zodpovědný za řešení nahlášených závad, které byly zároveň schváleny Správcem závad jako validní.
- Správce závad — Vrchní technik, odpovědný za kontrolu řešení závad. Nová hlášení závad validuje, kontroluje práci Techniků.
- Nepřihlášený uživatel — Návštěvník webových stránek, který nemá účet, resp. není přihlášen.

4.5 Případy užití

Případy užití jsou rozděleny dle čtyř základních rolí uživatelů, znázorněny jsou na Obr. 4.5. Popis jednotlivých případů užití je dostupný v elektronické příloze, je součástí projektu pro Enterprise Archtitect.



Obrázek 4.5: Model případů užití

4.6 Pokrytí případů užití funkčními požadavky

Pokrytí je znázorněno na Obr. 4.6.

Source \ Target	Informování technika o změně stavu závady	Informovat zákazníka o změně stavu závady	Nahlášení závady	Nahlášení závady za zákazníka	Nastavení aktivity typu závady	Odstoupení od řešení závady	Označení závady za vyřešenou	Prohlédnout si svá hlášení	Přidělování hlášených závad technikům	Přihlášení do modulu	Přijetí závady k řešení	Úprava typu závady	Upravit řešení závady	Vytvoření nového typu závady	Vytvoření řešení závady	Změna stavu hlášení závady
Distribuce hlášení závad osobám k řešení						←			←		←					
Hlášení závad			←	←												
Informační e-mailový systém	←	←														
Přihlašování uživatelů										←						
Správa hlášení závad							←	←								←
Správa typů závad					←							←		←		
Vytvoření zprávy o řešení závady													←		←	

Obrázek 4.6: Pokrytí případů užití funkčními požadavky

Kapitola 5

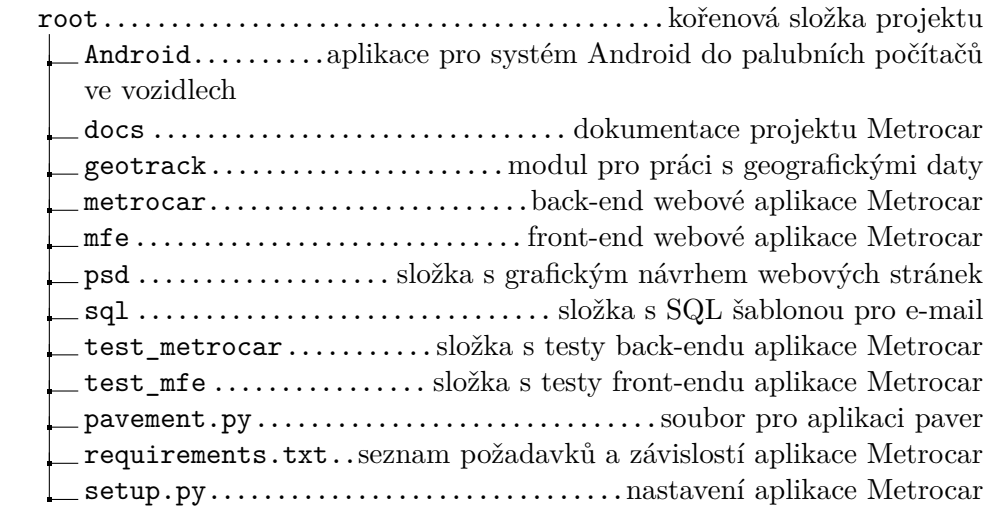
Návrh, architektura, design

V kapitole je popsána struktura aplikace Metrocar, dále struktura použitého balíčku a nakonec návrh výsledného modulu.

5.1 Struktura aplikace Metrocar

Když tato práce začínala, struktura celého projektu Metrocar byla dle Obr. 5.1.

Souběžně s touto prací na projektu Metrocar pracoval Vojtěch Knaisl, výsledek jeho práce je popsán v jeho bakalářské práci [45], kde hlavním cílem bylo vytvořit nové uživatelské rozhraní. V březnu 2015 se došlo do bodu, kdy se projekt pročistil a předělal na strukturu na Obr. 5.2, která ještě není finální a bude se měnit. Dosavadní front-end, postavený na HTML [46], CSS [47] a



Obrázek 5.1: Původní struktura projektu Metrocar

- `root` ..... kořenová složka projektu
  - `Android` ..... aplikace pro systém Android do palubních počítačů ve vozidlech
  - `docs` ..... dokumentace projektu Metrocar
  - `geotrack` ..... modul pro práci s geografickými daty
  - `server` ..... složka s aplikací Metrocar
    - `metrocar` ..... back-end webové aplikace Metrocar
    - `test_metrocar` ..... složka s testy back-endu aplikace Metrocar
    - `MANIFEST.in` ..... informace k distribuci aplikace Metrocar
    - `pavement.py` ..... soubor pro aplikaci paver
    - `README.rst` ..... informace k aplikaci Metrocar
    - `requirements.txt` ..... seznam požadavků a závislostí aplikace Metrocar
    - `setup.cfg` ... nastavení pro jeden balíček použitý v rámci aplikace Metrocar
    - `setup.py` ..... nastavení aplikace Metrocar
  - `sql` ..... složka s SQL šablonou pro e-mail

Obrázek 5.2: Nová struktura projektu Metrocar

JavaScriptu [48], byl smazán. Začalo se pracovat na novém front-endu, který by měl fungovat jako jednostránková webová aplikace [49]. Nový front-end by měl být tedy z velké části napsaný v JavaScriptu a server (aktuální back-end) používat jen jako zdroj dat, komunikaci by mělo zařizovat REST API [50].

Tyto změny se do výsledného modulu promítnou jen minimálně. Modul bude navržen s minimální závislostí na aplikaci Metrocar, z aplikace Metrocar bude využívat pouze část datové vrstvy a logiky.

5.2 Architektura modulu

Modul bude používat architektonický vzor MVC [51], který framework Django přímo podporuje a využívá [52].

Modul pro hlášení a správu závad bude postaven na volně dostupném balíčku Helpdesk (licence BSD [53]), který vyšel ze srovnání v kapitole 3.2.1 nejlépe.

5.2.1 Struktura balíčku Helpdesk

Helpdesk je šířen jako balíček, lze jej použít jako modul, nebo jako samostatnou aplikaci. Strukturou se drží zvyklostí aplikací v Django frameworku, struktura je znázorněna na Obr. 5.3.

Heldesk dodržuje architekturu MVC. Popis datové vrstvy je v souboru `models.py`, logickou vrstvu tvoří soubory ve složce `views` a prezentační vrstvu

```
helpdesk ..... kořenová složka balíčku
├── fixtures ..... složka s e-mailovými šablonami ve formátu JSON
├── locale ..... složka s lokalizací
├── migrations ..... databázové migrace
├── south_migrations ..... složka s dalšími migracemi, využívá South
├── static ..... složka pro statické části webu - CSS, JavaScript, obrázky
├── templates ..... složka s šablonami pro webové stránky
├── templatetags ..... složka s rozšířeními pro šablony
├── tests ..... složka s testy
├── views ..... složka s logickou vrstvou
├── admin.py ..... specifikace administrátorských stránek
├── akismet.py ..... rozšíření proti nevyžádané poště
├── apps.py ..... soubor s popisem aplikace
├── forms.py ..... soubor se specifikací webových formulářů
├── lib.py ..... soubor s knihovními funkcemi, co se jinam nevešlo
├── models.py ..... soubor s popisem datové vrstvy
├── poll_helpdesk_email_queues.sh .. soubor s nastavením periodických
│   kontrol e-mailové schránky
├── settings.py ... soubor s nastavením specifickým pro modul Helpdesk
└── urls.py ..... směrovač webových požadavků
```

Obrázek 5.3: Struktura balíčku Helpdesk

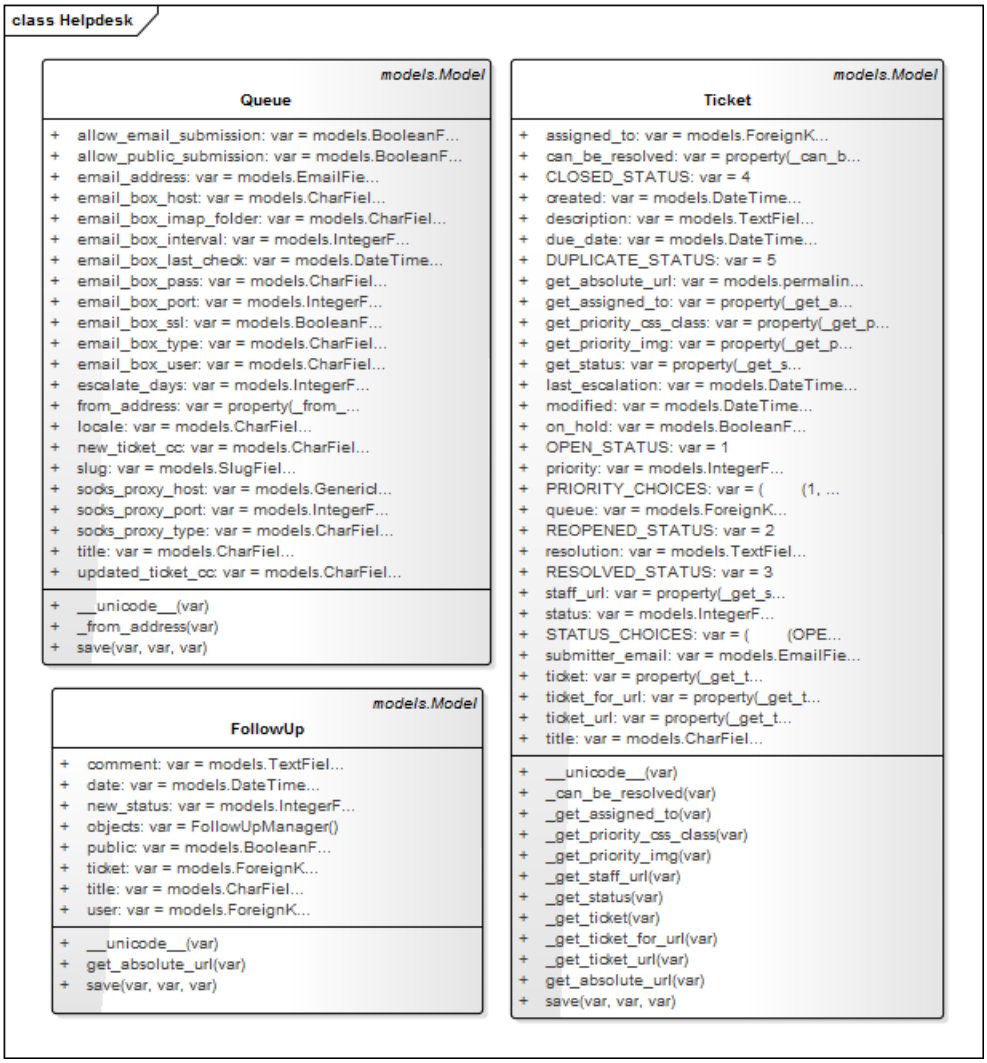
šablony ve složce `templates`. Zejména tyto soubory a složky bude potřeba upravit, aby výsledný modul splňoval požadavky.

5.2.2 Datová vrstva balíčku Helpdesk

Celý přehled tříd balíčku Helpdesk je kvůli svému rozhsahu dostupný pouze v elektronické příloze v projektu pro Enterprise Architect. V této části sekce jsou stručně popsány třídy, které budou reprezentovat hlášení závady, řešení hlášení a typ závady. Seznam popsaných tříd, co reprezentují a které informace vyžadované v analýze již obsahují:

- `Ticket` - Reprezentuje hlášení. Obsahuje stručný a detailní popis závady, typ závady, popis stavů, datum a čas vytvoření.
- `FollowUp` - Reprezentuje řešení závady. Obsahuje popis řešení a přechod do jiného stavu. Dále datum vytvoření řešení.
- `Queue` - Reprezentuje typ závady. Obsahuje název typu.

Co všechno třídy `Ticket`, `FollowUp` a `Queue` obsahují je znázorněno na Obr. 5.4.



Obrázek 5.4: Helpdesk - přehled vybraných tříd

5.2.3 Logická vrstva balíčku Helpdesk

Logická vrstva balíčku Helpdesk je ve složce `views`, vrstva je rozdělena následovně:

- `api.py` - Logika pro REST API. Nebude se používat.
- `feeds.py` - Logika pro RSS [54]. Nebude se používat.
- `kb.py` - Logika pro bázi znalostí. Báze znalostí představuje často kladené dotazy, systém otázka-odpověď, uživatel může přínos odpovědi ohodnotit. Použije se. Nejsou nutné velké úpravy.
- `staff.py` - Logika pro zaměstnance. Bude se upravovat, opravovat, doplňovat.
- `public.py` - Logika pro veřejnost. Část této logiky se odstraní, část upraví.

5.2.4 Prezentační vrstva balíčku Helpdesk

Prezentační vrstva je vytvořena za použití technologií HTML, CSS, JavaScript a jazyka šablon frameworku Django [55]. Prezentační vrstva je uložena ve formě `.html` souborů ve složce `templates`, soubory `.html` obsahují popis webových stránek.

5.2.5 Úpravy balíčku Helpdesk

Požadavky výsledného modulu práce vyžadují změny některých částí balíčku Helpdesk, v této kapitole je jejich přehled. Protože je změn mnoho, následující seznamy obsahují výčet toho důležitého. Změny v datové vrstvě:

- Do třídy reprezentující hlášení se musí přidat záznamy o tom, kdo závadu nahlásil, kdo hlášení zadal do systému, které vozidlo má závadu a kdy byla závada objevena.
- Předělání stavů hlášení tak, aby korespondovaly s výslednými stavy v diagramu v analýze (kapitola 4.2.1).
- Do třídy reprezentující řešení závady přidat záznam o času stráveném řešením problému.
- Do třídy reprezentující typ závady přidat popis typu závady a příznak aktivity.

Změny v logické vrstvě:

- Předělání logiky určené veřejnosti.

- Předělání logiky pro zaměstnance (Techniky).
- Vytvořit část logiky pro zákazníka.
- Předělání přístupových práv do webových sekcí.
- Znemožnění smazání hlášení.

Změny v prezentační vrstvě:

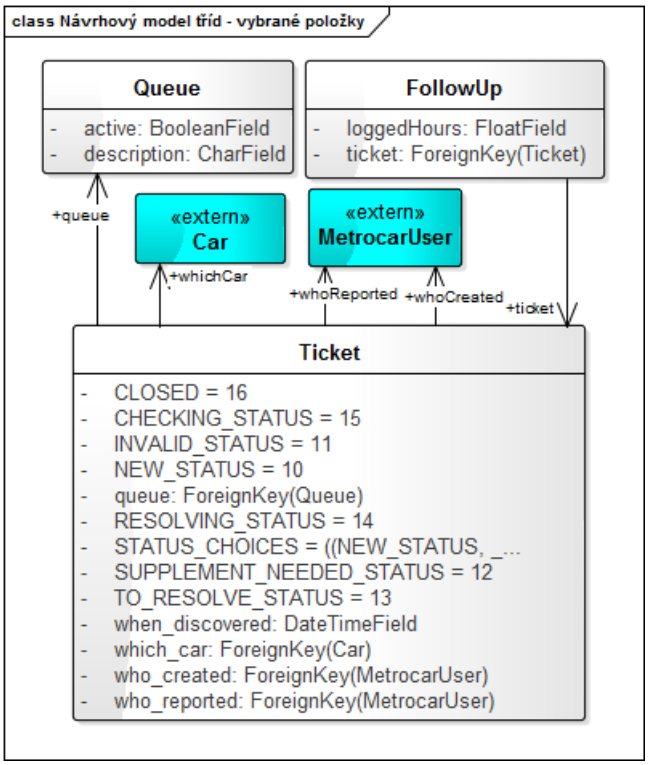
- Úprava zobrazovaných informací ve stávajících šablonách.
- Vytvoření nových šablon a doplnění stávajících.
- Předělání přístupových práv pro zobrazování obsahu v místech, kde to není možné v logické vrstvě.

5.3 Způsob využití balíčku Helpdesk

Výsledný modul bude postaven na balíčku Helpdesk. Z repozitáře [56] balíčku Helpdesk bude stažena nejaktuálnější verze a úpravy budou prováděny přímo do jeho kódu. Zvolený způsob sice není nejvhodnější pro případnou aktualizaci balíčku, ale má své opodstatnění:

- Moje znalost samotného frameworku Django není příliš hluboká.
- Společně s tím nepovažuji za dostatečné své znalosti balíčku Helpdesk. Přímou úpravou kódu Helpdesku lépe pochopím, jak funguje.
- Používaná verze frameworku Django je již nyní označena za nepodporovanou a nedoporučuje se její použití. Při rozhodování jsem zvolil konzervativnější způsob, kdy jsem chtěl raději upravit balíček tak, aby fungoval a fungoval s použitou verzí frameworku.
- Logická vrstva se musí uzpůsobit požadavkům. Z Helpdesku se využije menší část původní logiky, drtivá většina musí být přepracována, včetně například přístupových práv.

Lepší řešení z hlediska nezávislosti výsledného modulu a případných aktualizací balíčku Helpdesk je použití Helpdesku jen jako externí knihovny. Helpdesk by se pouze přidal do závislostí v nastavení projektu Metrocar, výsledný modul by se založil jako čistě nová Django aplikace (resp. modul) a třídy a funkce Helpdesku by se využívaly skrze importy. Toto řešení ovšem nese riziko toho, že by po aktualizaci balíčku Helpdesk mohl modul přestat fungovat. Společně s tím, že tento modul neznám tak dobře a i framework je pro mě, co se využití knihoven takovým způsobem týče, velkou neznámou, jsem se rozhodl touto cestou nevydat.



Obrázek 5.5: Návrhový model tříd - vybrané položky

5.4 Návrhový model tříd

V sekci 5.2.5 jsou vyjmenovány změny datové vrstvy, ty se promítnou do návrhového modelu tříd. Existující třídy, které jsou zobrazeny na Obr. 5.4, jsou doplněny o nové atributy. Na Obr. 5.5 je zjednodušený návrhový model tříd, který obsahuje přehled tříd a vazeb mezi nimi, třídy jsou navíc doplněny pouze o nové doplněné atributy. Názvy tříd a atributů jsou v anglickém jazyce, aby korespondovaly s implementací.

Stavy hlášení v balíčku Helpdesk jsou reprezentovány číslem, o propojení číselné reprezentace stavu a textové reprezentace se stará proměnná třídy `Ticket` s názvem `STATUS_CHOICES`. Tato proměnná je typu matice, ve které jsou matice dvojic (číselná reprezentace stavu, textová reprezentace stavu). Na tento způsob reprezentace stavů se naváže, existující stavy se smažou a doplní se nové stavy podle analýzy (kapitola 4.2.1). Mapování čísel na stavy je vidět na Obr. 5.5.

5.5 Role a přístupová práva

Přístupová práva jsou v balíčku Helpdesk řešena přes role. Role jsou určeny přes proměnné třídy reprezentující uživatele, tj. `is_active`, `is_staff` a `is_superuser`. Toto použití bude ve výsledném modulu zachováno, přičemž role se rozdělí podle Tab. 5.1. Jeden uživatel může mít více rolí.

Role (z analýzy)	*Příznak (hodnota proměnné)*
Zákazník	`is_active=True`
Technik	`is_staff=True`
Správce závad	`is_superuser=True`

Tabulka 5.1: Mapování rolí na proměnné třídy reprezentující uživatele

5.6 Lokalizace

Lokalizace je částečně podporována samotným balíčkem Helpdesk, ten využívá technologii [57] poskytovanou přímo frameworkem Django. V použití této technologie se bude pokračovat, framework v tomto ohledu nabízí dobré možnosti.

Systém Metrocar pro tuto technologii není nastaven, i když pro to má připravenou kostru. Do nastavení aplikace Metrocar se musí doplnit seznam podporovaných jazyků (CZ, EN). Pro nastavení podpory lokalizace poskytované přímo frameworkem Django se musí doplnit do seznamu používaných middleware tříd [58] třída LocaleMiddleware [58].

Do směrovače výsledného modulu bude přidán odkaz, na který se bude přesměrovávat z formuláře při změně jazyka. Tím dojde k propojení logiky a prezentační vrstvy.

Text ve zdrojových kódech, určený pro zobrazení uživatelům např. na webových stránkách, bude psán v anglickém jazyce. Ze souborů se zdrojovými kódy se vygeneruje soubor s kostrou pro překlad do českého jazyka, překlad textů se provede ručně v tomto souboru a následně se soubor zkompiluje tak, aby jej byla aplikace schopná použít.

5.7 Databáze

Pro práci s databází poskytuje framework Django bohaté rozhraní a podporuje nejrozšířenější databázové stroje. Co se týče výběru konkrétního databázového stroje, aplikace Metrocar je závislá na databázi PostgreSQL [59] verze 9.2. Je to z důvodu využití PostGIS [60] verze 1.5. PostGIS se stará o práci s geografickými daty, přesněji s polohou vozidel. Pro změnu databáze není důvod, způsob ukládání dat je tedy daný. Aktuální databáze se rozšíří o několik ta-

bulek vyžadovaných modulem, modul bude komunikovat s databází pomocí rozhraní nabízeného samotným frameworkem. Nemělo by být potřeba psát přímo SQL [61] dotazy.

Kapitola 6

Implementace

Implementace byla provedena z větší části dle návrhu.

6.1 Závilosti výsledného modulu na aplikaci Metrocar

V implementaci se podařilo zobecnit použití třídy reprezentující uživatele takovým způsobem, že výsledný modul bere pro práci s uživateli tu třídu, která je specifikována v nastavení aplikace, ve které je modul použit. V tomto případě to znamená, že třída `MetrocarUser`, zmíněná v návrhu, není vyloženě nutná. Modul si vystačí se standardní třídou `User`, kterou poskytuje framework Django (a jejímž potomkem je právě třída `MetrocarUser`). Modul by si měl umět poradit s každým potomkem třídy `User`, pokud nebude potomek nějak silně narušovat zděděnou strukturu.

Výsledný modul tedy potřebuje pouze přístup k třídám `Car` a `Reservation`. Data o vozidlech (třída `Car`) potřebuje kvůli specifikaci vozidla při hlášení závady. Data o rezervacích (třída `Reservation`) vyžaduje ze stejného důvodu, do zákaznického formuláře pro hlášení závady se vyplňují pouze ta vozidla, která měl daný zákazník rezervovaná.

Výsledný modul je tedy poměrně nezávislý na aplikaci Metrocar, vyžaduje přístup ke dvěma třídám z aplikace Metrocar a dokáže si poradit se třídou `User` a jejími potomky.

6.2 Opravy chyb z balíčku Helpdesk

Během implementace se objevilo několik chyb v balíčku Helpdesk, které musely být vyřešeny:

- Chyba v databázové migraci — Balíček dle dokumentace podporuje Django verze 1.4 a vyšší, už v něm jsou aktualizace pro vyšší verze

a jedna z těchto aktualizací zapříčinila jednu nefunkční databázovou migraci. Chyba spočívala v přístupu k neexistující proměnné třídy datové vrstvy (modelu). Problém byl navíc s tím, že se k proměnné přistupovalo přes volání funkce samotného frameworku a oprava se nedala vyřešit pouhým explicitním přejmenováním v kódu. K vyřešení chyby jsem použil existující opravu na oficiálních stránkách frameworku [62].

- Nefunkční změna jazyka (lokalizace) stránek — Formulář pro změnu jazyka sice existoval, ale byl nepřístupný a nefunkční. Ve formuláři bylo nutné upravit odkaz, na který se odesílá požadavek pro změnu jazyka po potvrzení formuláře. Odkaz musel korespondovat se záznamem v souboru směrovače (`urls.py`), který odkazoval na logiku starající se o změnu jazyka. Poslední potřebnou úpravou bylo doplnění odkazu na formulář do navigace (menu).

- Nefunkční generování uživatelských nastavení balíčku Helpdesk — Toto nastavení, uložené jako tabulka v databázi, by měl mít každý uživatel. Obsahuje například nastavení posílání e-mailů a maximální počet zobrazených hlášení na stránce. Implementace z nějakého důvodu nefungovala, nastavení se mělo generovat automaticky, domnívám se, že v tom hrála roli verze frameworku. Oprava spočívá v explicitní kontrole existence uživatelského nastavení při přístupu k němu. V případě, že nastavení neexistuje, se vygeneruje nové se základním nastavením.

Při implementaci jsem dále doplnil zdrojový kód balíčku Helpdesk o dokumentaci a komentáře, v některých částech byl popis kódu až příliš stručný.

Kapitola 7

Testování

K ověření správnosti implementace bylo provedeno testování. Testování bylo rozděleno do vrstev podle V-modelu [63]. Testy byly vytvořeny s ohledem na možnou automatizaci.

7.1 Jednotkové testování

Jednotkové testy se zaměřují na testování jednotlivých tříd výsledného modulu bez napojení na aplikaci Metrocar. Vazby mezi aplikací Metrocar a modulem byly v případě potřeby nahrazeny jednoúčelovými testovacími třídami, vzniklými pouze pro uspokojení povinných vazeb. Jednotkové testy využívají testovací rozhraní [64] frameworku Django, zejména třídu `unittest.TestCase`.

7.2 Integrační testování

Integrační testy se zaměřují na komunikaci mezi aplikací Metrocar a výsledným modulem, zejména pak použití tříd aplikace Metrocar `Car` a `Reservation`. K tomu bylo potřeba vytvořit generátory pomocných tříd pro uspokojení povinných vazeb. Integrační testy využívají testovací rozhraní frameworku Django.

7.3 Systémové testování

Systémové testy se zaměřují na testování systému jako celku z pohledu zákazníka. Výsledný modul se testuje skrz webové rozhraní, testují se jednotlivé kroky uživatele - při přihlášení do modulu, při nahlášení závady, atd. Systémové testy jsou vytvořeny s pomocí přídavku do webového prohlížeče Mozilla Firefox [65] jménem Selenium IDE [66].

7.4 Akceptační testování

Akceptační testy se zaměřují na celkovou funkčnost systému z pohledu zákazníka, testy se řídí analýzou. Jsou otestovány vybrané případy užití a složitější procesy, například celý životní cyklus hlášení, od vzniku hlášení závady po úspěšné vyřešení a uzavření. Akceptační testy jsou vytvořeny s pomocí přídavku do webového prohlížeče Mozilla Firefox jménem Selenium IDE.

Kapitola 8

Nasazení

Metrocar je webová aplikace, beží na webovém serveru s podporou Pythonu verze 2.7, databázi používá PostgreSQL verze 9.2 s rozšířením PostGIS. Vytvořený modul je integrován do aplikace Metrocar. Fyzické rozmístění komponent je zobrazeno na diagramu nasazení Obr. 8.1. V diagramu se klientem rozumí zařízení s webovým prohlížečem, tedy osobní počítač, mobilní telefon, atd. Server může být postaven na libovolném operačním systému, stačí, aby podporoval výše zmíněné technologie. Databáze nemusí nutně běžet na stejném stroji.

Server s běžící aplikací Metrocar má pod správou kolega Knaisl, adresa webových stránek je `server.metrocar.knaisl.cz`, resp. `metrocar.knaisl.cz`. Stránky modulu by měly být dostupné přes `server.metrocar.knaisl.cz/bugrep`, resp. `metrocar.knaisl.cz/bugrep`.

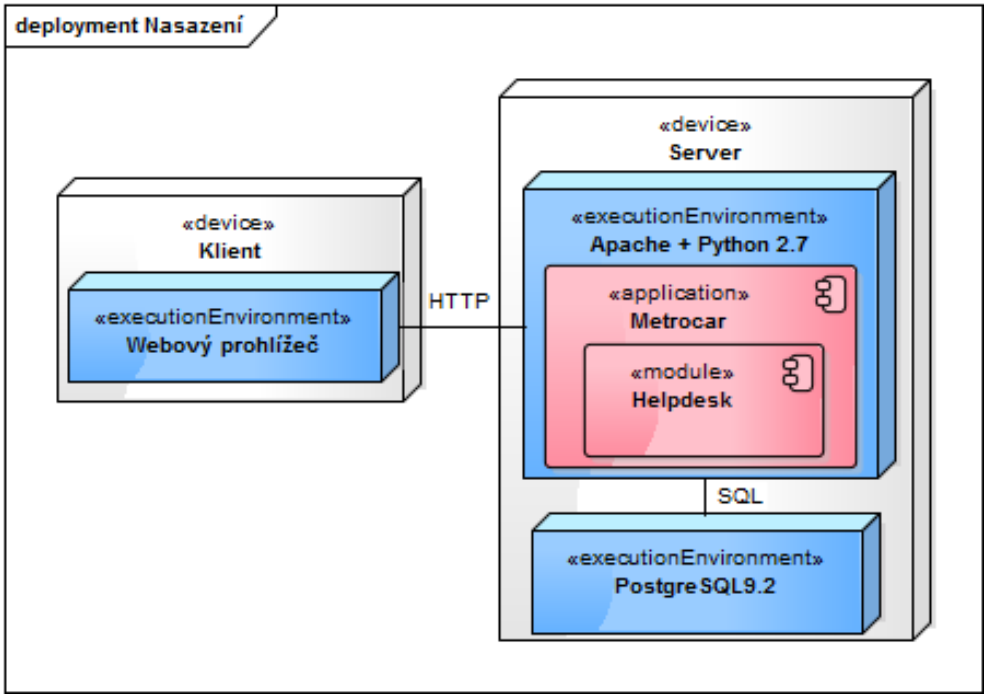
Popis nasazení modulu do běžící aplikace Metrocar je v příloze C.

8.1 Ukázková aplikace

Pro předvedení funkčnosti byla vytvořena jednoduchá webová aplikace, která obsahuje výsledný modul a pouze nezbytné části aplikace Metrocar. Nezbytnými částmi jsou datové vrstvy a upravené administrátorské rozhraní těch modulů, které se starají o vozidla a rezervace.

Pro nasazení ukázkové webové aplikace se použila jedna ze zdarma dostupných platforem, která podporuje Python verze 2.7 a databázi alespoň SQLite [67] (pro ukázkovou aplikaci stačí SQLite, PostgreSQL není nutné). Takové služby nabízí například OpenShift [68], Heroku [69] nebo Stackato [70]. Z dostupných možností byl vybrán OpenShift, důvodem je dostatečné množství služeb zdarma, minimální omezení použití a předešlé zkušenosti s touto platformou.

Na OpenShift byl založen účet a vytvořena doména a aplikace. Nasazení samotné aplikace se neobešlo bez problémů. Na OpenShift lze založit aplikaci přímo pro framework Django, ovšem kvůli starší verzi frameworku je mno-



Obrázek 8.1: Diagram nasazení

hem schůdnější cestou založení aplikace postavené na Pythonu verze 2.7 a doinstalování potřebných balíčků.

Také bylo potřeba upravit strukturu aplikace pro OpenShift, k tomu se použila struktura volně dostupného projektu DjangoShift [71].

Ukázková aplikace na OpenShift používá databázi SQLite, ta pro ukázkové účely stačí.

Výsledný modul je nasazen na `bugrep-novacorp.rhcloud.com`.

Záložní nasazení bylo provedeno na Heroku. Opět byl založen účet a profil aplikace, struktura webové aplikace se upravila pro potřeby této platformy, vycházelo se přitom z ukázkového Django projektu od Heroku [72]. Zároveň se pro běh aplikace nastavila platforma i aplikace samotná. Změna oproti nasazení na OpenShift je v použité databázi, aplikace na Heroku používá PostgreSQL. Webová aplikace je dostupná na `bugrep-novacorp.herokuapp.com`.

Další informace k nasazeným ukázkovým aplikacím jsou v elektronické příloze, obsahují například údaje k vytvořeným uživatelským účtům, které lze použít k testování.

Závěr

Cílem práce bylo navrhnout a implementovat modul hlášení a správy závad pro informační systém Metrocar. Zadání práce lze považovat za splněné, postupoval jsem dle pokynů, jednotlivé kroky vypracoval a výsledkem práce je funkční modul hlášení a správy závad včetně požadovaných částí. Co se týče práce nad rámec zadání, respektive vypracovaných kroků, které nebyly explicitně uvedeny v zadání:

- Věnoval jsem se řešení hlášení závad u jiných společností, které nabízejí službu sdílení vozidel.
- Vypracoval jsem přehled a srovnání volně dostupných balíčků pro platformu Python/Django, které se zabývají hlášením a správou závad.
- Implementace je rozšířena o možnosti použitého balíčku Helpdesk, například pokročilé vyhledávání hlášení s několika možnými filtry, statistiky, báze znalostí, apod. K dalším funkcím balíčku Helpdesk (RSS, API) byl znemožněn přístup, ovšem nemělo by být obtížné je zprovoznit a upravit pro danou doménu.
- Byly vytvořeny a nasazeny celkem 2 ukázkové aplikace.
- V elektronické příloze je dostupná uživatelská příručka.

Návrhy a doporučení

Další postup při vývoji modulu je zejména v doplnění funkcí.

K hlášením závad a k jejich řešením by se mohly přikládat soubory, například fotodokumentaci závady, účty, apod. Použitý balíček Helpdesk přílohy k hlášení z části podporuje, ovšem mezi požadavky na výsledný modul přílohy nebyly, proto je tato funkcionalita ve výsledku zablokována.

Další možnost je v rozšíření lokalizace, aktuálně je podporován český jazyk a anglický jazyk, kostra je připravena pro více jazyků.

Webové stránky by se mohly doplnit o dynamiku, například ve formuláři při nahlašování závady za zákazníka by se výběr vozidel mohl omezit podle vybraného zákazníka, který závadu nahlašoval.

Co se týče užší spolupráce modulu s aplikací Metrocar a jejím novým front-endem, za úvahu stojí vložení zákaznického formuláře pro hlášení závad do nového front-endu aplikace Metrocar, vytvoření nového REST API (nebo úprava stávající API) pro tento modul a propojení formuláře v aplikaci s REST API modulu. Zákazník by tak měl možnost hlásit závadu z nového front-endu a bylo by to pro něj jistě pohodlnější.

Literatura

[1] Python Software Foundation: *Python*. [online]. [cit. 2014-11-15]. Dostupné z: `https://www.python.org/`

[2] Django Software Foundation: *Django*. [online]. [cit. 2014-11-15]. Dostupné z: `https://www.djangoproject.com/`

[3] Autonapůl, družstvo: *Autonapůl - první český carsharing*. [online]. [cit. 2014-11-15]. Dostupné z: `http://www.autonapul.org`

[4] Car4Way a.s.: *Car4Way*. [online]. [cit. 2014-11-15]. Dostupné z: `http://www.car4way.cz`

[5] Klimek Motion s.r.o.: *AJO.CZ - český carsharing*. [online]. [cit. 2014-11-15]. Dostupné z: `http://www.ajo.cz`

[6] Jansa, P.: *SHARUJEME.cz*. [online]. [cit. 2014-1-30]. Dostupné z: `http://www.sharujeme.cz`

[7] Wagner, J.: *Metrocar*. [online]. [cit. 2014-11-15]. Dostupné z: `https://www.assembla.com/spaces/wagnejan_metrocar/wiki`

[8] Nebeský, O.: *Rezervační systém carsharingové společnosti*. ČVUT FEL, 2009.

[9] The PHP Group: *PHP*. [online]. [cit. 2014-11-15]. Dostupné z: `http://php.net/`

[10] Buytaert, D.: *Drupal*. [online]. [cit. 2014-11-15]. Dostupné z: `https://www.drupal.org/`

[11] Vařecha, F.: *Transformace informačního systému pro carsharing do webové služby*. ČVUT FEL, 2010.

[12] Wagner, J.: *Příprava systému autonapul.cz na reálné nasazení*. ČVUT FEL, 2012.

[13] Ječmínek, J.: *Vývoj účetní komponenty do systému pro podporu sdílení aut*. ČVUT FEL, 2013.

[14] Pokorný, P.: *Vývoj geografického modulu pro systém carsharingové společnosti*. ČVUT FIT, 2013.

[15] Ječmínek, J., et al.: *Metrocar - práce studentů SI2*. [online]. [cit. 2014-11-15]. Dostupné z: `https://www.assembla.com/spaces/wagnejan_metrocar/wiki/SI2_-_Home`

[16] Autonapůl, družstvo: *Všeobecné obchodní podmínky Autonapůl, družstva pro fyzické a právnické osoby*. [online]. 31.1.2014 [cit. 2014-11-16]. Dostupné z: `http://www.autonapul.org/sites/default/files/pdfs/2014_01_31_VOP.pdf/`

[17] Car4Way a.s.: *Car4Way - hlášení závady*. In: Facebook [online]. 27.1.2014 [cit. 2014-11-16]. Dostupné z: `https://www.facebook.com/CAR4WAY/posts/246586508852342`

[18] Klimek Motion s.r.o.: *AJO.CZ - hlášení závad*. [online]. [cit. 2014-11-16]. Dostupné z: `www.ajo.cz/autopujcovna-popis`

[19] Klimek Motion s.r.o.: *AJO.CZ - přílohy k obchodním podmínkám*. [online]. 1.7.2014 [cit. 2014-11-16]. Dostupné z: `http://carsharing.ajo.cz/wp-content/uploads/2014/07/op-prilohy-2-az-7-v1_6.pdf`

[20] GoCar CarSharing Ltd.: *GoCar - drive different*. [online]. [cit. 2014-11-16]. Dostupné z: `https://www.gocar.ie/`

[21] Daniel Greenfeld, Audrey Roy and Two Scoops Press: *Django Packages*. [online]. [cit. 2014-11-17]. Dostupné z: `https://www.djangopackages.com/`

[22] Daniel Greenfeld, Audrey Roy and Two Scoops Press: *Django Packages - Ticketing*. [online]. [cit. 2014-11-17]. Dostupné z: `https://www.djangopackages.com/grids/g/ticketing/`

[23] Helmig, B., et al.: *Django-Knowledge*. [online]. [cit. 2015-4-5]. Dostupné z: `https://www.djangopackages.com/packages/p/django-knowledge/`

[24] Hacker, S., et al.: *Django-TODO*. [online]. [cit. 2015-4-5]. Dostupné z: `https://www.djangopackages.com/packages/p/django-todo/`

[25] Poulton, R., et al.: *Django-Helpdesk*. [online]. [cit. 2015-4-5]. Dostupné z: `https://www.djangopackages.com/packages/p/django-helpdesk/`

[26] Karbownicki, T.: *DONER - Ticket Management System for Getting Things Done^{TM} (TMS4GTD)*. [online]. [cit. 2015-4-5]. Dostupné z: https://www.djangopackages.com/packages/p/doner/

[27] Henner, et al.: *YATS - yet another ticketing system*. [online]. [cit. 2015-4-5]. Dostupné z: https://www.djangopackages.com/packages/p/yats/

[28] Helmantel, W.: *Mr. Wolfe*. [online]. [cit. 2015-4-5]. Dostupné z: https://www.djangopackages.com/packages/p/mrwolfe/

[29] ManagementMania.com: *SLA - Service Level Agreement*. [online]. [cit. 2015-4-5]. Dostupné z: https://managementmania.com/cs/service-level-agreement

[30] Ozturk, Y., et al.: *ITSY - Issue Tracking System*. [online]. [cit. 2015-4-5]. Dostupné z: https://www.djangopackages.com/packages/p/itsy/

[31] Object Management Group, Inc.: *UML - Unified Modeling Language*. [online]. [cit. 2014-11-17]. Dostupné z: http://www.uml.org/

[32] Sparx Systems Pty Ltd.: *Enterprise Architect v.11.1.1110*. [přístup 17. listopadu 2014]. Dostupné z: http://www.sparxsystems.com/products/ea/

[33] Mozilla Foundation: *Bugzilla Bug Status*. [online]. [cit. 2015-3-4]. Dostupné z: https://bugzilla.mozilla.org/page.cgi?id=fields.html

[34] Atlassian: *What is an Issue*. [online]. [cit. 2015-3-4]. Dostupné z: https://confluence.atlassian.com/display/JIRA/What+is+an+Issue

[35] Edgewall Software: *Trac*. [online]. [cit. 2015-3-2]. Dostupné z: http://trac.edgewall.org/

[36] Edgewall Software: *The Trac Ticket System*. [online]. [cit. 2015-3-2]. Dostupné z: http://trac.edgewall.org/wiki/TracTickets

[37] Edgewall Software: *Trac Ticket State Chart*. [online]. [cit. 2014-11-17]. Dostupné z: http://trac.edgewall.org/attachment/wiki/TracTickets/TracTicketStateChart20060603DF.png

[38] Edgewall Software: *Trac Enhanced Workflow*. [online]. [cit. 2014-11-17]. Dostupné z: http://trac.edgewall.org/attachment/wiki/NewWorkflow/trac-enh-workflow.gif

[39] Guberman, A.: *Primary Goals - Defect State Transition Map*. [online]. [cit. 2014-11-17]. Dostupné z: http://www.primarygoals.net/config/StateTransitions.htm

[40] Guberman, A.: *Primary Goals*. [online]. [cit. 2015-4-5]. Dostupné z: http://www.primarygoals.com

[41] BEA Systems, Inc.: *BEA Systems, Inc. - Trouble Ticket*. [online]. [cit. 2014-11-17]. Dostupné z: https://docs.oracle.com/cd/E13206_01/wlac/components/docs/javadoc/theory/smart/ebusiness/troubleticket/package-summary.html

[42] BEA Systems, Inc.: *BEA Systems, Inc.* [online]. [cit. 2015-4-5]. Dostupné z: http://www.oracle.com/us/corporate/Acquisitions/bea/index.html

[43] Microsoft: *MSDN User Story State Diagram*. [online]. [cit. 2014-11-17]. Dostupné z: http://i.msdn.microsoft.com/dynimg/IC309449.png

[44] Microsoft: *MSDN*. [online]. [cit. 2014-4-5]. Dostupné z: http://i.msdn.microsoft.com/

[45] Knaisl, V.: *Moderní uživatelské rozhraní projektu Metrocar*. ČVUT FIT, 2015.

[46] W3Schools: *HTML - HyperText Markup Language*. [online]. [cit. 2015-4-5]. Dostupné z: http://www.w3schools.com/html/

[47] W3Schools: *CSS - Cascading Style Sheets*. [online]. [cit. 2015-4-5]. Dostupné z: http://www.w3schools.com/css/

[48] W3Schools: *JavaScript*. [online]. [cit. 2015-4-5]. Dostupné z: http://www.w3schools.com/js/

[49] Malý, M.: *Single Page Apps a řešení problémů s historií*. [online]. 1.6.2011 [cit. 2015-4-5]. Dostupné z: http://www.zdrojak.cz/clanky/single-page-apps-a-reseni-problemu-s-historii/

[50] Malý, M.: *REST: architektura pro webové API*. [online]. 3.8.2009 [cit. 2015-4-5]. Dostupné z: http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[51] Bernard, R.: *Úvod do architektury MVC*. [online]. 7.5.2009 [cit. 2015-3-8]. Dostupné z: http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[52] Django Software Foundation: *FAQ: General*. [online]. [cit. 2015-3-8]. Dostupné z: https://docs.djangoproject.com/en/1.5/faq/general

[53] University of California, Berkeley: *Berkeley Software Distribution*. [online]. [cit. 2015-5-4]. Dostupné z: http://opensource.org/licenses/BSD-3-Clause

[54] Pilgrim, M.: *What Is RSS*. [online]. [cit. 2015-4-5]. Dostupné z: http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html

[55] Django Software Foundation: *The Django template language*. [online]. [cit. 2015-4-5]. Dostupné z: https://docs.djangoproject.com/en/1.5/topics/templates/

[56] Poulton, R., et al.: *GitHub: rossp/django-helpdesk*. [online]. [cit. 2015-4-5]. Dostupné z: https://github.com/rossp/django-helpdesk

[57] Django Software Foundation: *Translation*. [online]. [cit. 2015-4-23]. Dostupné z: https://docs.djangoproject.com/en/1.5/topics/i18n/translation/

[58] Django Software Foundation: *Middleware*. [online]. [cit. 2015-4-23]. Dostupné z: https://docs.djangoproject.com/en/1.5/ref/middleware/

[59] The PostgreSQL Global Development Group: *PostgreSQL*. [online]. [cit. 2015-4-5]. Dostupné z: http://www.postgresql.org/

[60] PostGIS Project Steering Committee: *PostGIS*. [online]. [cit. 2015-4-5]. Dostupné z: http://postgis.net/

[61] W3Schools: *SQL*. [online]. [cit. 2015-4-5]. Dostupné z: http://www.w3schools.com/sql/

[62] Django Software Foundation: *Django's bug tracker and wiki: bug n.20853*. [online]. [cit. 2015-4-5]. Dostupné z: https://code.djangoproject.com/ticket/20853

[63] Štrobl, R.: *Handout č. 1 - Úvod do testování software*. [online]. [cit. 2015-5-7]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-ATS/_media/lectures/bi-ats-h01.pdf

[64] Django Software Foundation: *Testing Django applications*. [online]. [cit. 2015-5-5]. Dostupné z: https://docs.djangoproject.com/en/1.5/topics/testing/overview/

[65] Mozilla Foundation: *Firefox v.37.0.2*. [přístup 1. května 2015]. Dostupné z: https://www.mozilla.org/cs/firefox/desktop/

[66] SeleniumHQ: *Selenium IDE v.2.9.0*. [přístup 1. května 2015]. Dostupné z: http://www.seleniumhq.org/projects/ide/

[67] Hipp, Wyrick and Company, Inc.: *SQLite*. [online]. [cit. 2015-5-5]. Dostupné z: https://www.sqlite.org/

[68] Red Hat, Inc.: *OpenShift*. [online]. [cit. 2015-5-5]. Dostupné z: https://www.openshift.com/

[69] Heroku Inc.: *heroku*. [online]. [cit. 2015-5-5]. Dostupné z: `https://www.heroku.com/`

[70] ActiveState Software Inc.: *Stackato*. [online]. [cit. 2015-5-5]. Dostupné z: `http://www.activestate.com/stackato`

[71] Ranganathan, R., et al.: *DjangoShift*. [online]. [cit. 2015-5-5]. Dostupné z: `https://github.com/ramr/DjangoShift`

[72] Heroku Inc.: *GitHub: Heroku/python-getting-started*. [online]. [cit. 2015-5-5]. Dostupné z: `https://github.com/heroku/python-getting-started`

Příloha A

Seznam použitých zkratek

a.s. Akciová společnost

aj. a jiné

apod. a podobně

atd. a tak dále

API Application Programming Interface

BSD Berkeley Software Distribution

CSS Cascading Style Sheets

ČVUT České vysoké učení technické

GIS Geographic Information System

GPL General Public License

HTML HyperText Markup Language

IDE Integrated Development Environment

ITSY Issue Tracking System

MIT Massachusetts Institute of Technology

MSDN Microsoft Developer Network

MVC Model-View-Controller

např. například

PHP Personal Home Page

resp. respektive

REST Representational State Transfer

RSS Rich Site Summary

s.r.o. Společnost s ručením omezeným

SLA Service Level Agreement

SQL Structured Query Language

UML Unified Modeling Language

URL Uniform Resource Locator

YATS Yet Another Ticketing System

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src` ..... složka se zdrojovými kódy
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Nováček_Jan_2015.pdf` ..... text práce ve formátu PDF
- `attachments` ..... složka s přílohami
  - `TicketSystem.eap` ..... soubor projektu pro Enterprise Architect
  - `UserManual.pdf` ..... uživatelská příručka

Příloha C

Instalační příručka

Příručka instalace vytvořeného modulu je pro aktuální verzi aplikace Metrocar se strukturou na Obr. C.1. Příručka vychází z instalačního postupu balíčku Helpdesk[1]. Příručka je pro linuxový operační systém, testována byla na Lubuntu 14.04. Předpokládá se, že používaná verze frameworku Django je 1.5.4, výsledný modul byl vyvíjen pro tuto verzi.

Zdrojové kódy z vývojové verze aplikace Metrocar jsou dostupné v elektronické příloze, přičemž změny aplikace Metrocar, které byly potřebné pro nasazení výsledného modulu, jsou v kódech zvýrazněny a okomentovány. V této příručce jsou potřebné změny vyjmenovány a v případě potřeby zdůvodněny, ovšem pro přesnou podobu změn je nutné nahlédnout do odkazovaných souborů.

```
server ..... složka s aplikací Metrocar
├── metrocar ..... back-end webové aplikace Metrocar
│   ├── settings ..... složka s nastaveními
│   │   └── base.py ..... hlavní nastavení aplikace Metrocar
│   ├── helpdesk ..... výsledný modul
│   └── urls.py ..... směrovač URL
├── test_metrocar ..... složka s testy back-endu aplikace Metrocar
├── MANIFEST.in ..... informace k distribuci aplikace Metrocar
├── pavement.py ..... soubor pro aplikaci paver
├── README.rst ..... informace k aplikaci Metrocar
├── requirements.txt .. seznam požadavků a závislostí aplikace Metrocar
├── setup.cfg ..... nastavení pro jeden balíček použitý v rámci aplikace
│   Metrocar
└── setup.py ..... nastavení aplikace Metrocar
```

Obrázek C.1: Struktura aplikace Metrocar

[1] https://django-helpdesk.readthedocs.org/en/latest/install.html

V souboru `setup.py` je potřeba přidat do proměnné `install_requires` položky dle Obr. C.2.

```
install_requires=['django-markdown-deux==1.0.5',
'django-bootstrap-form==3.1',
'simplejson==2.3.3',
'email-reply-parser==0.3.0']
```

Obrázek C.2: Doplnění souboru `setup.py`

Dále do souboru `requirements.py` doplnit položky dle Obr. C.3.

```
# Helpdesk
django-markdown-deux==1.0.5
django-bootstrap-form==3.1
simplejson==2.3.3
email-reply-parser==0.3.0
```

Obrázek C.3: Doplnění souboru `requirements.py`

Jedná se o balíčky, které je potřeba doinstalovat. Pro instalaci stačí ve složce `server` přes příkazovou řádku zadat příkazy `sudo python setup.py develop` a `sudo paver install_dependencies`.

V souboru `base.py`, který je ve složce `metrocar/settings`, je potřeba donastavit aplikaci Metrocar. Soubor `base.py`, použitý na vývojovém prostředí, je dostupný v elektronické příloze, včetně okomentovaného a zvýrazněného kódu, který je potřeba doplnit do čisté instalace aplikace Metrocar. Zde je vyjmenovaný seznam kroků, které je potřeba udělat:

- Doplnění nainstalovaných aplikací (proměnná `INSTALLED_APPS`).
- Doplnění proměnné `LANGUAGES`, definující seznam jazyků, které lze zvolit v formuláři pro změnu jazyka na webových stránkách.
- Doplnění proměnné `MIDDLEWARE_CLASSES`.
- Nastavení `LOGIN_URL` na hodnotu `'/bugrep/login/'`. Toto nastavuje přihlašovací stránku pro celou aplikaci Metrocar na tu, která se používá ve výsledném modulu. Implicitní hodnota proměnné `LOGIN_URL` je `/accounts/login/`, tuto hodnotu žádná část aplikace nepoužívá a ani není dostupná přes `urls.py`, lze ji tedy přepsat.

Dále je potřeba upravit soubor `urls.py`. Do proměnné `urlpatterns` je potřeba přidat záznam, který bude odkazovat na stránky výsledného modulu. Záznam vypadá takto: `(r'^bugrep/', include('helpdesk.urls')),`.

Nakonec se složka `django-helpdesk` nakopíruje do složky `metrocar`, tak, jak je to zobrazeno na Obr. C.1.

Nyní jsou soubory připraveny a stačí ve složce `server/metrocar/` zadat příkazy (pro UNIX):

- `sudo python manage.py syncdb`
- `sudo python manage.py migrate helpdesk`
- `sudo python manage.py collectstatic`
- `sudo python manage.py loaddata emailtemplate.json`

Nyní by měl být modul připraven k použití.