České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Roman Kubů**

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: **Integrace aplikací pro sledování automobilu**

Pokyny pro vypracování:

Nastudujte problematiku stávajících prototypů Androidních a serverových aplikací projektu Metrocar [3,4]. Odstraňte jejich zásadní nedostatky a integrujte je do jednoho funkčního celku, který bude umožňovat sledování provozních údajů automobilů. Hlavními výstupy práce budou:
- Funkční rozhraní pro přenos dat mezi palubní jednotkou a serverem.
- Uživatelské rozhraní pro správu nastavení palubní jednotky.
- Funkční palubní jednotka sledující pohyb a provozní údaje automobilu.

Seznam odborné literatury:

[1] SOMMERVILLE, Ian. Softwarové inženýrství. 1. vyd. Brno: Computer Press, 2013, 680 s. ISBN 978-80-251-3826-7.
[2] LARMAN, Craig. Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development. 3rd ed. Upper Saddle River, N.J.: Prentice Hall PTR, 2005, xxv, 702 p. ISBN 01-314-8906-2.
[3] JUNGMAN, Tomáš. Sber a zpracování dat o provozu automobilů [online]. 2015 [cit. 2015-09-30]. Dostupné z: https://dspace.cvut.cz/handle/10467/61504. Diplomová práce. ČVUT FEL.
[4] KUBU, Roman. Zpracování dat o provozu automobilů pro projekt Metrocar [online]. Praha, 2013 [cit. 2015-09-30]. Dostupné z: https://dspace.cvut.cz/handle/10467/18340. Bakalařská práce. ČVUT FEL.

Vedoucí: Ing. Martin Komarek

Platnost zadání: do konce zimního semestru 2017/2018

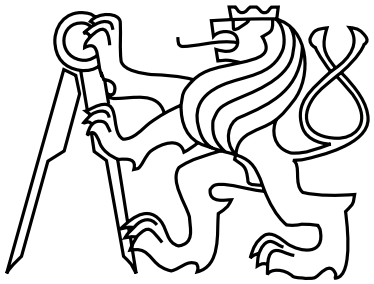
prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 19. 5. 2016

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Diplomová práce

Integrace aplikací pro sledování automobilu

Roman Kubů

Vedoucí práce: Ing. Martin Komárek

Studijní program: Otevřená informatika, Magisterský

Obor: Softwarové inženýrství

23. května 2016

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 20. 5. 2016 .....

Abstract

This work solves car unit integration with server part of project Metrocar. Car unit provides filtered data from OBD2 standard, GPS coordinates and acceleration of vehicle to server part of project. Server part solves user management, car unit configuration and presents obtained data.

Abstrakt

Práce řeší integraci palubní jednotky vozidla a serverové části projektu Metrocar. Palubní jednotka poskytuje filtrovaná data z OBD2 standardu, GPS souřadnic a z akcelerace vozidla serverové části projektu. Serverová část řeší správu uživatelů, konfiguraci palubní jednotky a zobrazení získaných dat.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

V dnešní době je řada malých a středně velkých společností, které se snaží na českém ale i světovém trhu uspět. Vždy však čelí řadě rizik a také zvýšenému množství výdajů oproti předchozím dobám. Tyto výdaje jsou také dávány na dopravu zaměstnanců. Řada společností z tohoto důvodu potřebuje mít vlastní vozový park. Výdaje na tento vozový park ovšem nekončí zakoupením automobilů, ba právě naopak. Musí mít zaměstnance, který se o tento vozový park stará jak po administrativní stránce tak i po technické. Další výraznou položkou při správě vozového parku jsou samozřejmě pohonné hmoty a amortizace vozidla, jak poukazuje tento článek [2]. Uvádí, že až 30% nákladů jde na pohonné hmoty a tato částka může být výrazně ovlivněna řidičským stylem. Následně také navrhuje monitoring vozidel, jelikož řidičský styl uživatele vozidla může být výrazně ovlivněn tím, když ví, že může být jeho počínání na silnici následně vyhodnoceno jeho nadřízeným.

V dnešní době již samozřejmě existuje řada systémů, které tento problém řeší, avšak jejich cena na pořízení a údržbu může být pro malé rozrůstající se firmy neúnosná. Na základně těchto myšlenek vznikla nová větev vývoje projektu Metrocar, který se hlavně zabývá vývojem řešením pro carsharingové společnosti[10]. Cílem tohoto projektu je tedy vytvořit ucelený softwarový a hardwarový celek pro monitoring a správu firemní flotily.

1.1 Historie projektu

Projekt Metrocar vznikl na ČVUT počátkem zimního semestru 2008/2009[16]. Již od počátku byl projekt rozdělen na dvě části a to na serverovou v PHP a palubní jednotku založenou na modulu Siemens XT75 osazeném na čipové sadě M2M Motherboard, pro kterou se programovalo v jazyce Java. V zimním semestru 2009/2010 byla změněna vývojová platforma pro serverovou část na Python s Django[18]. V letím semestru 2009/2010 a v zimním semestru 2010/2011 došlo k rozsáhlému refactoringu kódu palubní jednotky[17], který byl zaměřený na odstranění hardwarově specifických částí kódu.

V zimním semestru 2011/2012 byla změněna platforma vývoje palubní jednotky na platformu Android[19], vzhledem k dostupnosti těchto telefonů v široké veřejnosti. Dále také byla provedena analýza požadavků na palubní jednotku a z tohoto vyvstal požadavek na získávání informací od ECU (řídící jednotka vozidla), ovšem v té době nebylo nic v rámci palubní jednotky implementováno. Skupina v zimním semestru 2012/2013, které jsem se účastnil jako vedoucí projektu, úspěšně pokračovala v implementaci požadavků na serverovou část projektu, ale také zahájila práci na palubní jednotce a získávání údajů od ECU[20].

V rámci své bakalářské práce[7] v letním semestru 2012/2013 jsem úspěšně implementoval aplikaci na platformě Android umožňující získávání záznamů od ECU a jejich odesílání na server Metrocar. Na základě poznatků z této aplikace pokračoval Tomáš Jungman[5] v rámci své diplomové práce v letním semestru 2014/2015 v implementaci palubní jednotky. Jeho hlavním úkolem byla možnost přidání nových příkazů na ECU oproti původnímu fixnímu počtu. Bohužel se mu nepodařilo provést integraci se serverem Metrocar a z tohoto důvodu neobhájil svoji diplomovou práci v daném roce. V zimním semestru 2015/2016 bylo rozhodnuto o změněně platformy pro serverovou část na Play framework pro který je možno psát v jazycích Java a Scala. Tohoto projektu jsem se účastnil jako poradce.

Kapitola 2

Popis problému, specifikace cíle

Výsledné řešení se má skládat ze dvou základních částí a to palubní jednotky a serverového řešení. Účelem palubní jednotky je získávání dat o provozu automobilu a jejich následném odesílání na server. Serverová část má získané údaje ukládat a umožnit jejich zobrazení uživateli. Následující kapitoly popisují jednotlivé cíle práce.

2.1 Získávání dat od ECU

Palubní jednotka by měla umožňovat získávání údajů o jízdě automobilu od ECU pomocí OBD2 jednotky. Údaje které je možné takto získávat se mohou lišit automobil od automobilu. Je také třeba aby uživatel mohl zadat jaké údaje chce získávat.

2.2 Získávání GPS souřadnic vozidla

Palubní jednotka by měla umožňovat získávání informace o pozici vozidla.

2.3 Získávání akcelerace vozidla

Palubní jednotka by měla umožňovat získávání informace o akceleraci vozidla. Tyto informace by měly být následně zpracovány pro lepší čitelnost.

2.4 Filtrování získaných údajů

Palubní jednotka by měla umožňovat nastavení filtrování získávaných údajů od ECU, pomocí předem připravených algoritmů. Měla by také umožňovat změnit hodnotu nastavení filtru.

2.5 Odesílání získaných údajů

Získané údaje by měly být synchronizovány se serverovou částí projektu.

2.6 Synchronizace nastavení palubní jednotky

Palubní jednotka by měla se severem synchronizovat svoje nastavení. Mezi položky jejichž nastavení by se mělo synchronizovat se serverem patří nastavení jaké údaje chceme získávat od ECU, nastavení filtrů na získávané údaje a nastavení časových intervalů úkonů v palubní jednotce. Tedy například jak často má palubní jednotka odesílat získané údaje od ECU na server.

2.7 Získávání chybových stavů vozidla

Palubní jednotka by měla umožňovat získávání chybových stavů vozidla a jejich odeslání na server. Mezi tyto chybové stavy vozidla patří například informace o nefunkčním vstřikování do válce.

2.8 Offline funkčnost jednotky

Palubní jednotka by měla být schopná fungovat i bez stálého připojení na internet.

2.9 Mimo cíle řešení

Cílem tohoto projektu je vytvořit prototyp palubní jednotky. V rámci tohoto projektu tedy není řešeno zabezpečení dat, jak v rámci ukládání dat do databáze tak i jejich následné posílání na serverovou část a také UX návrh aplikací a jejich následný vzhled.

Kapitola 3

Analýza řešení

Vzhledem k novým požadavkům na funkčnost jak palubní jednotky tak i serverové části byla provedena analýza nového řešení pro správu firemní flotily a z té vznikly následující požadavky. Některé z nich již byly implementované v předchozích iteracích projektu a bylo třeba je zkontrolovat, popřípadě upravit, aby odpovídaly novému zadání.

3.1 Funkční požadavky

REQ1 Systém bude umožňovat připojení k ECU s protokolem OBD2.

REQ2 Systém bude umožňovat získávání libovolných údajů definovaných v protokolu OBD2.

REQ3 Systém bude umožňovat získávání pozice vozidla pomocí GPS.

REQ4 Systém bude umožňovat získávání akcelerace vozidla.

REQ5 Systém bude umožňovat autorizaci uživatele i bez připojení na internet.

REQ6 Systém bude odesílat získané informace na server Metrocar.

REQ7 Systém bude umožňovat filtrování získaných údajů.

REQ8 Systém bude umožňovat synchronizaci nastavení mezi palubní jednotkou a serverem.

REQ9 Systém bude umožňovat získávání chybových stavů vozidla od ECU.

REQ10 Systém bude odesílat získané chybové stavy vozidla na server.

3.2 Nefunkční požadavky

REQSYS1 Palubní jednotka bude implementován na platformě Android.

REQSYS1 Serverová část bude implementován na Play frameworku.

3.3 Analýza existujícího řešení

Jak již bylo zmiňováno v kapitole 1.1 aplikace jak na palubní jednotku tak i pro serverovou část již byla částečně implementována dříve. Proto byla provedena analýza těchto řešení s přihlédnutím k novým požadavkům, aby bylo určeno, jaké již existující části softwaru je možno použít.

3.3.1 Palubní jednotka

Tomáš Jungman v rámci své diplomové práce Tomáše Jungmana [5] implementoval palubní jednotku. Analýza tohoto řešení poukázala na následující nedostatky tohoto řešení pro budoucí použití.

Údaje o jízdě automobilu byly ukládány do textového souboru v rámci Android zařízení. Následně byla provedeno odeslání obsahu tohoto souboru na serverovou část. V rámci tohoto souboru, byly také uloženy informace o startu a konci dané jízdy, ujeté vzdálenosti a době trvání dané jízdy. Toto ovšem znemožňuje neustálé odesílání daných dat na serverovou část. Bude nutné předělat ukládání daných údajů na palubní jednotce pro umožnění kontinuální zasílá získaných údajů na serverovou část.

Akcelerace automobilu je získávána v neupravené formě bez odstranění gravitační složky akcelerace. Akcelerace automobilu je tedy následně ovlivněna tím jak je palubní jednotka uchycena v automobilu. Neboli následná akcelerace na některých osách akcelerometru může být až 9,8 m/s^2 aniž by se automobil jakkoliv pohyboval. K možnosti identifikace reálné akcelerace automobilu je tedy třeba odstranit gravitační složku, ale také na získané údaje použít filtr k odstranění rušení z daných dat.

Aplikace také využívá ve velké míře CPU zařízení. Pro první testovací jízdy byl využit tablet Nvidia Shield s procesorem Tegra K1 Quad-core 2.2 GHz a využití CPU nespadlo pod 90% výkonu. Pokud tablet nebyl napájen během jízdy došlo k jeho vybití do jedné hodiny, toto by nemělo tak velký vliv na výsledné řešení, jelikož je plánováno, že by palubní jednotka byla napájena. Avšak i při napájení tabletu z automobilu docházelo k poklesu nabití tabletu, což by v dlouhé době využívání vedlo k jeho úplnému vybití. Je tedy třeba najít důvod velkého využívání CPU a odstranit ho.

Přihlášení uživatele bez nutnosti připojení k internetu je pouze možné pro posledního přihlášeného uživatele s aktivním připojením k internetu. Toto by mělo být upraveno tak, aby uživatel který se již jednou přihlásil s aktivním připojením k internetu mohl přihlásit i bez něj.

Nedostatkem tohoto řešení je také neexistence jakýkoliv testů na funkčnost této aplikace, což znemožňuje provádění efektivního refaktorování dané aplikace.

3.3.2 Serverová část

Jak již bylo zmíněno dříve, serverová část byla vytvářena od nuly v zimním semestru 2015/2016 na platformě Playframework. Tohoto projektu jsem se účastnil jako poradce pro potřeby Palubní jednotky. V rámci tohoto projektu byly vytvořené základní rozhraní pro komunikaci s palubní jednotkou, neboli přihlášení uživatele a zasílání získaných údajů o provozu automobilu, ale také zobrazení získaných dat. Následně dále na tomto projektu v rámci své bakalářské práce pokračuje pan Polata. Proto se moje práce omezila na práci s rozhraními pro palubní jednotku. Kde je třeba upravit rozhraní pro přihlášení uživatele a zasílaní získaných údajů, ale také o rozšíření dalších rozhraních potřebných pro nastavení palubní jednotky.

3.4 Palubní jednotka

V rámci předělání serverové části z Pythonu do Play frameworku, byla v rámci úvah brána také v potaz možnost přesunutí tohoto projektu od Car-sharingové společnosti do systému na správu firemní flotily. Tato změna by umožnila dosáhnutí většího trhu, jelikož pro správu firemní flotily malého či středního rozměru není třeba mít povinnou homologaci a používat řadu procesů, které by nešlo obejít, jelikož v rámci firemní flotily by uživatelé byli povinni používat systém dle firemních nařízení a také by byli vázaní pracovní smlouvou. Toto by nám umožnilo zbavit se těžce realizovatelných částí systému jako je odemykání automobilu pomocí RFI karty či imobilizace vozidla pomocí telefonu.

Prvotní návrh procesů pro nový systém vycházel z původního systému pro Car-Sharingovou společnost, a proto jsme se rozhodli udělat revizi toho, jak by systém měl fungovat na straně automobilu. Z této revize vzešly tři hlavní možné způsoby realizace systému na straně automobilu. V následujícím odstavci rozeberu hlavní výhody či nevýhody jednotlivých realizací.

3.4.1 Skrytá palubní jednotka s klientem

Tento způsob by se skládal ze dvou mobilních telefonů, kde jeden by byl nepřístupný obyčejnému uživateli (palubní jednotka) a druhého mobilního (klient). Palubní jednotka by sbírala informace o pohybu automobilu, synchronizovala svoje nastavení se serverovou částí. Druhý mobilní telefon by sloužil jako klient, který by se k druhému telefonu připojoval například přes bluetooth a sloužil by jako klientská aplikace. Neboli přes něj by se uživatel přihlašoval a mohl sledovat hodnoty získané z automobilu.

Tento přístup má výhodu v tom, že by obyčejný uživatel neměl přístup k palubní jednotce a nemohl by ji tam například vypnout, pokud by nechtěl, aby jeho cesta byla sledována. Avšak největší nevýhodou tohoto systému je potřeba dvou mobilních telefonů, kde by oba musely být napájené a uložení palubní jednotky, tak aby s ní nešlo jednoduše interagovat, což by vedlo ke zvýšení ceny nákladu na instalaci tohoto systému do automobilu. Druhou neméně malou nevýhodou by byla nutnost vyvinutí dvou aplikací, jedné pro klientský telefon a druhé pro palubní jednotku, což by vedlo ke zvýšení nákladu na vývoj palubní jednotky.

3.4.2 Fixní palubní jednotka

V tomto způsobu by byl jeden telefon fixně instalován v automobilu a veškerá interakce s palubní jednotkou by byla realizovaná přes tento telefon.

Nevýhodou je nutnost fixního umístění telefonu, což jsou náklady pro společnost, která by tento systém chtěla využívat. Toto ovšem skrývá výhodu z pohledu, že instalace zařízení včetně prvotního spárování s OBD2 jednotkou by prováděl vyškolený uživatel, který by následně musel provést i prvotní nastavení systému. Takže obyčejný uživatel by následně nemusel provádět už žádné kroky k zprovoznění palubní jednotky, pouze se přihlásit a spustit jízdu.

Výhodou je také to, že by palubní jednotka tím pádem byla spojená s automobilem a tím pádem by musela aktualizovat pouze data spojená s tímto automobilem, jako nastavení dotazů na OBD2 jednotku a další.

3.4.3 Mobilní palubní jednotka

V tomto způsobu by nebylo třeba instalování telefonu do automobilu a uživatel by používal svůj vlastní telefon jako palubní jednotku, jelikož v řadě firem je přidělen zaměstnanci služební telefon. Výhodou tohoto systému jsou tedy minimální pořizovací náklady.

Tento systém má ale také bohužel řadu nevýhod, kde největší je nutnost na podrobnější proškolení uživatele na zacházení s palubní jednotkou, aby byl schopný spárovat mobil s OBD2 jednotkou. Také, jelikož v automobilu nebude trvale umístěna palubní jednotka, nebude možné realizovat sledování automobilu, pokud se uživatel sám nerozhodne data posílat. Další nevýhodou je také nutnost umožnit aplikaci aktualizovat údaje a nastavení pro všechny automobily ve skupině, jelikož předem není jisté s jakým automobilem se rozhodne uživatel cestovat.

3.4.4 Výsledné rozhodnutí

Po konzultaci s vedoucím práce bylo rozhodnuto, že pro další vývoj bude použita varianta fixní palubní jednotky, jelikož ta nejvíce splňovala požadavky na budoucí používání systému.

Tabulka 3.1: Rychlost dotazů

	Dotaz na Audi v ms	Dotazu na Octavia v ms
1 bytový	40	250
2 bytový	60	340

3.5 Množství dat produkovaných systémem

Smyslem palubní jednotky je sbírat informace o pohybu a funkčnosti vozidla. Množství těchto záznamů ovšem může způsobit problém s jejich přenosem a ukládáním.

Pro pochopení o jaké množství záznamů se může jednat je nejdříve nutné pochopit o jaký druh záznamů se jedná a rychlost, kterou tyto záznamy můžeme získávat. Tyto záznamy můžeme rozdělit na data od ECU a záznamy od mobilního telefonu.

3.5.1 Data od ECU

Tyto záznamy jsou získávány od ECU pomocí OBD2 rozhraní přes Bluetooth. Z těchto záznamů je možné zjistit rychlost vozidla, otáčky motoru, teplotu motoru a další podobné informace.

3.5.2 Data od mobilního telefonu

Od mobilního telefonu jsme schopni získat informace o pozici vozidla a hodnotách od senzorů mobilního telefonu. V nynější chvíli získáváme údaje o pozici vozidla neboli GPS souřadnice a záznamy od akcelerometru.

3.6 Rychlost sběru dat

Na počtu záznamů, které palubní jednotka produkuje, má vliv řada faktorů, které se opět dají rozdělit podle toho, o jaký druh záznamu se jedná.

3.6.1 Data od ECU

Tyto záznamy mají omezení kolik jich můžeme získat vzhledem k rychlosti odpovědi od ECU. Tato rychlost je silně rozdílná vzhledem k typu ECU, ale také OBD2 zařízení použitého pro sběr záznamů. Pro demonstraci rychlost sběru záznamů z Octavie V2 2.0 litru benzín je průměrně okolo 290 ms, zatímco u Audi 1.9 litru nafta je průměrná rychlost sběru záznamů okolo 50 ms. Tento rozdíl je téměř 6x, což znamená, že máme mnohem podrobnější přehled o pohybu a funkčnosti vozidla, ale také posíláme a ukládáme 6x více záznamů než u druhého vozidla.

Vliv na rychlost má také, jaké záznamy získáváme, jelikož je zde rozdíl v čase, ve kterém získáme 1 bytovou odpověď oproti 2 bytové, jak je vidět v tabulce 3.1. Tento rozdíl u rychlé jednotky není tak markantní, avšak u pomalejší je znatelný.

3.6.2 Data od mobilního telefonu

U sbírání informací o pozici automobilu můžeme nastavit v jakém minimálním intervalu mají tyto údaje být získávány, ale také po jaké ujeté minimální vzdálenosti. Rozhodli jsme se, že minimální vzdálenost mezi dvěma GPS souřadnicemi by měla být asi 2 metry. Což by hrubě odpovídalo 150 milisekundám pro auto pohybující se maximální rychlostí ve městě neboli 50 km/h. Při maximální povolené rychlosti v České Republice, což je 120 km/h, by získávání údajů každých 150 milisekund představovalo získávání GPS souřadnic každých 5 metrů. Z těchto důvodů jsme se rozhodli nastavit minimální dobu pro získávání GPS souřadnic na 150 milisekund a minimální vzdálenost 2 metrů.

Rychlost sbírání informací o akceleraci vozidla můžeme nastavit na jednu z přednastavených možností nebo nastavit vlastní interval. Avšak tento nastavený interval je pouze doporučení pro Android v jakém nejvyšším časovém intervalu tyto hodnoty chcete získávat. Obvykle bude tento interval menší, například při nastavení intervalu na Nvidia Shield na 200 milisekund byl tento interval okolo 20 milisekund. Pro potřeby našeho projektu jsem tento interval nastavil na SENSOR_DELAY_UI neboli na interval 60 milisekund.

3.7 Sběr informací

V rámci diplomové práce Tomáše Jungmana [5] došlo také ke změně v jakém formátu jsou získávány záznamy a jak jsou ukládány, což má vliv na to, jak můžeme se záznamy pracovat, ale také na objem dat, které záznamy představují.

3.7.1 Původní systém

Původní systém měl předem definováno, jaké záznamy má získávat od ECU, což byly záznamy o rychlosti vozidla, teplotě motoru, otáčky motoru, pozice pedálu, ke kterým po jejich získání byly navíc přidány informace o GPS souřadnicích, jejich přesnosti a čas. Tyto záznamy byly následně spojeny do jednoho záznamu.

Tento systém měl nevýhodu v tom, že byly předem definované záznamy, které se mají získávat. Pokud bychom se tedy rozhodli získávat navíc například průtok vzduchu, znamenalo by to provést úpravu jak na serverové části, kde bychom museli upravit rozhraní, tak i datový model, ale také na Androidí části, kde by opět muselo dojít ke změně rozhraní tak i datového modelu.

3.7.2 Nový systém

V novém systému se informace pro Android o tom, jaké záznamy chceme získávat stahují ze serveru, což nám umožňuje dynamicky měnit záznamy, které chceme získávat. Každý záznam nyní obsahuje samotná data, čas, kdy byly získané a jaký příkaz byl použit pro jejich získání. Tato změna umožňuje přidání nového příkazu bez nutnosti jakékoliv změny již hotového řešení.

Tabulka 3.2: Objem dat za minutu

	Doba trvání jízdy v ms	velikost v kB	velikost kB za minuty
1	349 596	969	167,745
2	326 020	917	168,773
3	584 361	1 640	168,389

3.8 Objem dat

Tento nový systém ovšem znamená navýšení dat, se kterými pracujeme. V původní verzi měla všechna data pouze jeden čas, zatímco v nové verzi má každý údaj svůj čas, což vede ke zpřesnění výpočtů, ale navýšení datového přenosu.

Z tohoto důvodu jsme použili 3 testovací (viz table 3.2) jízdy vytvořené Tomášem Jungmanem na vozidle Audi a vypočetli, kolik jedna minuta jízdy vyprodukuje průměrně dat.

Výsledkem je, že auto vyprodukuje průměrně 168 kB za minutu funkčnosti. Toto se zdá jako velmi malé číslo, ale pokud bychom předpokládali, že auto bude využíváno 8 hodin denně, 20 dní za měsíc, auto vyprodukuje okolo 1600 MB dat za měsíc. Toto by pro menší společnost s 10 auty nepředstavovalo velký problém. Ovšem velké Car-Sharingové společnosti mají znatelně více automobilů například ZipCar využívá přes 11 000 automobilů a Car2Go 12 000, což by znamenalo okolo 19 TB dat každý měsíc, což by už bylo znatelně problematické.

3.9 Snížení objemu dat

Objem dat v systému lze rozdělit na 3 části a to na část mobilní jednotky, data posílaná z mobilní jednotky na server a na data uložená na serveru.

3.9.1 Mobilní jednotka

Mobilní jednotka produkuje data získaná z ECU tak i od mobilního telefonu, jak již bylo dříve vysvětleno. K tomu, aby bylo možné redukovat počet těchto dat, je nejdříve nutno upravit systém, jakým jsou vyprodukovaná data odesílaná.

Dříve jakmile systém získal jeden celý záznam dat, tak ho systém uložil do databáze mobilní jednotky a pokusil se jej následně odeslat a pokud bylo odeslání úspěšné, tak ho smazal z databáze. Záznamy se do databáze ukládaly, jelikož mohlo dojít k selhání při odesílání a při dalším pokusu o odeslání záznamů, již byla odeslána část skupiny záznamů, kterou se nepodařilo dříve odeslat. Jelikož každý záznam měl být odeslán okamžitě, jak bylo možné, tak tento systém neumožňoval provádět úpravu dat.

Nyní při startu jednotky spustíme dvě různá vlákna. Jedno vlákno bude získávat záznamy od ECU, případně od mobilního telefonu a ukládat je do databáze mobilní jednotky. Druhé vlákno v nastaveném časovém intervalu přečte tyto záznamy, pro každý druh záznamu (rychlost, teplota motoru atd.) provede jejich promazání, pokud má záznam nastavený algoritmus na promazávání a uloží je do databáze na odeslání. Poté se druhé vlákno pokusí již promazaná data z databáze odeslat na server. Tento systém by měl u více jádrových telefonů

zajistit, že úprava dat a jejich odesílání nezpomalí interval, ve kterém získáváme záznamy od ECU.

3.10 Algoritmy pro úpravu záznamů

Každý druh záznamu (CarRequest) může mít nastavený právě jeden algoritmus pro zmenšení počtu získaných údajů. Nastavení algoritmu může být opět provedeno přes server bez nutnosti interakce s mobilní jednotkou, ale také je možnost měnit toto nastavení přímo na mobilní jednotce. Pro začátek budou implementovány 2 algoritmy pro zmenšení počtu záznamů před odesláním na server.

3.10.1 Procentuální

Tento algoritmus by získaná data porovnával s předchozím záznamem a pokud se hodnota nezměnila o určitý počet procent, tak je záznam smazán. Účinnost tohoto algoritmu je silně ovlivněna druhem dat a také stylem jízdy. Pokud například auto jede na dálnici se zapnutým tempomatem, většina záznamů bude obsahovat stejnou hodnotu, tím pádem počet smazaných záznamů větší než když automobil popojíždí v zácpě, kde se záznamy neustále mění.

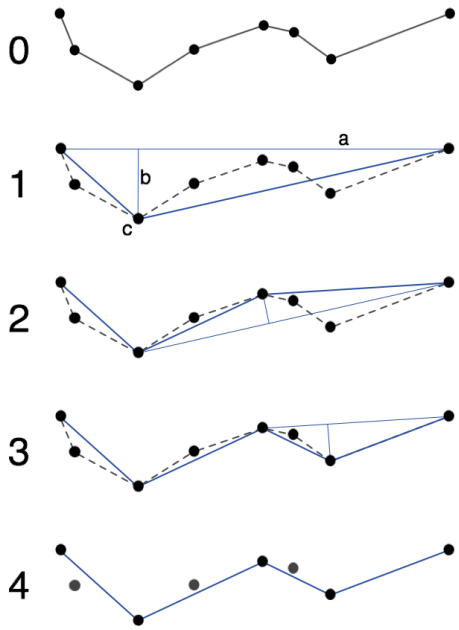
Tento algoritmus byl vyzkoušen na testovací jízdě trvající okolo 10 minut, která obsahovala 2800 záznamů o rychlosti vozidla, kde se rychlost vozidla používala pro výpočet ujeté vzdálenosti. Algoritmus byl nastaven na změnu o 0.1%, neboli stačilo aby se hodnota jakkoliv změnila od předchozího záznamu, aby byl záznam zachován. Již toto jednoduché omezení způsobilo snížení počtu záznamů z původních 2800 na 800 aniž byl ovlivněn výsledek výpočtu ujeté vzdálenosti.

3.10.2 Ramer–Douglas–Peucker algoritmus

Mezi nejznámější algoritmy na zmenšení počtu bodů na křivce je Ramar-Douglas-Peucker algoritmus. Tento algoritmus nezávisle na sobě navrhli Urs Ramer v roce 1972[25] a pánové David Douglas, Thomas Peucker v roce 1973[12].

Tento algoritmus se většinou vyskytuje ve dvou variantách a to iterativní a rekurzivní. V rámci naší aplikace jsme použili iterativní verzi, ale jelikož je rekurzivní verze jednoduší na vysvětlení funkčnosti tohoto algoritmu, budu zde dále uvádět rekurzivní verzi.

V prvním kroku algoritmu vybereme první a poslední bod, které spojíme úsečkou, tyto body jsou základem nové zjednodušené křivky. Kolem této spojnice vytvoříme koridor o šířce naší definované tolerance. Následně hledáme bod nejvzdálenější od této spojnice. Pokud námi nalezený bod je v rámci koridoru, algoritmus tím končí. Pokud je v ně tohoto koridoru přidáme ho jako nový bod na křivce. Pomocí tohoto bodu rozdělíme dříve existující spojnici na dvě nové spojnice a algoritmus opakujeme pro obě jeho části. Ilustrace této funkčnosti je vidět na obrázku 3.1 a také na pseudo kódu xx.



Obrázek 3.1: Postup Ramer-Deuglas-Paucker algoritmu

Algorithm 1 My algorithm

```
1:  function DouglasPeucker(PointList[], epsilon)
2:      dmax = 0
3:      index = 0
4:      end = length(PointList)
5:      for i = 2 to (end - 1) do
6:          d = perpendicularDistance(PointList[i], Line(PointList[1], PointList[end]))
7:          if d > max then
8:              dmax = d
9:              index = i
10:     if dmax > epsilon then
11:         recResults1[] = DouglasPeucker(PointList[1...index], epsilon)
12:         recResults2[] = DouglasPeucker(PointList[index...end], epsilon)
13:
14:         ResultList[] = {recResults1[1...length(recResults1) − 1],
15:         recResults2[1...length(recResults2)]}
16:     else
17:         ResultList[] = PointList[1], PointList[end]
        return ResultList[]
```

3.10.3 Přidání nového algoritmu

Přidání nového algoritmu na zmenšení počtu záznamů zde znamená naprogramování nového modulu pro mobilní jednotku. Na serverové části je již třeba pouze přidat informaci o novém možném algoritmu na ořezávání dat.

3.10.4 Posílání dat

Posílání dat z mobilní jednotky na serverovou část je provedeno pomocí REST rozhraní s formátem dat JSON.

Zde je možné udělat zmenšení velikosti dat pro přenos tak, že získané záznamy převedeme na JSON a ten komprimujeme. Následně převedeme takto komprimovaný soubor na binární řetězec, který následně ve formátu JSON pošleme na server. Vzhledem k redundanci dat v JSON formátu tak i opakujících se příkazů na ECU by komprimace mohla výrazně snížit velikost dat.

Zřejmou nevýhodou tohoto řešení je, že jak mobilní jednotka tak i serverová část bude potřebovat čas navíc na dekomprimaci těchto záznamů, což by při větší zátěži mohlo způsobit problémy hlavně na serverové části. Jasnou výhodou je zde zmenšení velikosti posílaných dat, což přímo ovlivňuje využití datového limitu na mobilní jednotce.

Po následné konzultaci s vedoucím práce bylo rozhodnuto, že tato funkčnost nebude součástí nynější implementace.

3.11 Vyhodnocení řidičského stylu

Dříve, než můžeme začít pracovat na snížení počtu dat odeslaných z mobilní jednotky na server, je třeba definovat k čemu budou data použita, aby bylo možné porovnat výsledek před snížením objemu dat a po jeho snížení.

Naše společnost se mění velkou rychlostí a veškeré lidské aktivity jsou tím silně ovlivněny. Mezi ně také patří zvětšující se počet automobilů, jezdících po našich silnicích a také nutnost dostat se na místo určení větší rychlostí. Z tohoto důvodu se také zvětšuje agresivita některých řidičů na silnicích, což často vede k nehodám a v nemalé míře také smrtelným nehodám. Dalším problémem s větším počtem automobilů na silnicích je také dopad silniční dopravy na životní prostředí, speciálně na CO_2 produkované automobily.

3.11.1 Podobné práce

Problematikou vyhodnocení řidičského stylu se zabývá řada prací, ať již z pohledu možnosti detekování agresivního řidičského stylu, tak i dopad jeho jízdy na životní prostředí. Ovšem není žádný ověřený ucelený systém na ohodnocení řidiče na základě jeho jízdních dat. Většina těchto systémů funguje na sběru uživatelských dat z jízdy, jako je například rychlost, otáčky motoru atd. a jejich následné subjektivní vyhodnocení expertem na danou problematiku. Níže je výčet několika prací, zabývajících touto problematikou, ze které jsme vycházeli při určení našich pravidel na ohodnocení řidiče.

- Driving Style Analysis Using Data Mining Techniques[3] - Se zabývá analýzou řidičských parametrů získaných systémem Gipix, který získává data o rychlosti, akceleraci, nadmořské výšce, GPS souřadnicích a chybě GPS. Pro potřeby analýzy si definují několik statických parametrů získaných z dat. Dobu, kterou se vozidlo pohybuje přes 60 km/h, což berou jako překročení povolené rychlosti, zrychlení a brzdění vozidla a také celkovou nutnou mechanickou práci potřebnou ke zrychlení vozidla. Z této statické analýzy vyvozují, že agresivita řidiče má silný vztah s akcelerací a brzdění vozidla. A tendence k rychlé jízdě má silný vztah s údaji o překročené rychlosti a okamžité rychlosti vozidla.

- Design of Algorithms for Payment Telematics Systems Evaluating Driver's Driving[6] - Se zabývá návrhem algoritmu pro: "Pay How You Drive" neboli plať tak jak jezdíš. Tento systém využívá řada pojišťovacích společností pro výši pojistného na automobil. Avšak i v dnešní technologicky vyspělé společnosti řada těchto společností bere v potaz pouze počet ujetých kilometrů. Přitom by měli více vyhodnocovat chování řidiče, které vede k nebezpečným situacím, které mohou vést k újmě na majetku. Z toho důvodu navrhují algoritmus, který se skládá z šesti kroků.

  1 Získání provozních dat vozidla.
    - Informace o vozidle jako výkon vozidla, kolika stupňová je převodovka a váha vozidla.
    - GPS pozice vozidla a akcelerace na ose x a y.
    - Data o provozu vozidla jako rychlost, zařazený převodový stupeň, otáčky motoru, používání blinkrů, venkovní teplota, použití stěračů a zda jsou zapnuta světla do mlhy.
  2 Vyhodnocení venkovního počasí
    - Toto vyhodnocení je děláno na základě venkovní teploty, frekvence stěračů a mlhových světel.
  3 Vypočítání maximálního možného výkonu v daný okamžik.
  4 Vyhodnocení manévrů řidiče.
    - Na základě dříve získaných informací vyhodnocují několik možný typů předjíždění, zatáčení vozidla a agresivní brzdění.
  5 Vyhodnocení řidičských manévrů.
  6 Přidělení trestných bodů na základě provedených akcí.

  Přidělení trestných bodů řidiči se provádí na základě počtu ujetých kilometrů.

- Modelling safety-related driving behaviour—impact of parameter values[1] - Se zabývá nalezením a vyhodnocením parametrů spojených s bezpečným chováním řidičů v silničním provozu.

  - Rychlost ve volně plynoucím provozu
  - Odstup vozidel
  - Akcelerace a brzdění vozidla
  - Chování k prioritním vozidlům (Autobus, Sanitka, ...)

- Předjíždění a změna pruhu
- Plnění silničních pravidel

Pro náš projekt je zajímavá definice ohledně akcelerace a brzdění vozidla, které definují následovně.

- Normální akcelerace 0.9 - 1.5 m/s^2
- Maximální akcelerace 1.5 - 3.6 m/s^2
- Normální brzdění 0.9 - 1.5 m/s^2
- Nouzové 1.5 - 2.4 m/s^2

- Driver Classification and Driving Style Recognition using Inertial Sensors[8] - Se zabývá vytvořením profilu řidiče a jeho následné použití k identifikaci určitého řidiče vozidla. Rozpoznáváni řidiče provádí na základě tří základních akcí a těmi jsou akcelerace vozidla, brzdění a zatáčení.

  - Brzdění identifikují na základě rozsvícení brzdových světel, neboli pokud jsou brzdová světla rozsvícená po dobu delší než 1.
  - Akceleraci vozidla definují na základě pozice plynového pedálu kde počátek zrychlení definují jako AccPedal ¿ předchozí pozice a konec jako AccPedal ¡= předchozí pozice kde opět změny menší jak 1 sekunda neberou v potaz.
  - Zatáčení vozidla definují na Základě natočení volantu.

  Bohužel OBD2 neposkytuje informace o rozsvícení brzdových světel ani pozici volantu, které by mohly být využity k identifikaci těchto akcí. OBD2 poskytuje informaci o pozici plynového pedálu, která by mohla být v našem projektu využita k identifikaci zrychlení vozidla.

- Driving Style Recognition Using a Smartphone as a Sensor Platform[4] - Upozorňuje na několik zajímavých faktů a to že 56% smrtelných nehod je způsobeno agresivními řidiči. Řidiči řídí bezpečněji, když vědí, že jejich aktivita na silnici je monitorována a také že 76% pracovníků jezdí každý den do práce samo. Proto navrhují systém MIROAD, který je schopný detekovat agresivní chování řidiče na základě informací získaných pomocí telefonu. Systém ukládá data z gyroskopu, akcelerometru a úhel otočení vozidla. Na základě těchto dat následně vyhodnocuje několik typů akcí.

  - Zatočení doleva, doprava a otočení o 180%.
  - Agresivní zatočení doleva, doprava a otočení o 180%.
  - Agresivní akceleraci a brzdění vozidla.
  - Překročení povolené rychlosti.
  - Agresivní změna jízdních pruhů.

  Avšak tato práce se zabývá pouze rozeznáním těchto akcí a už ne jejich ohodnocením vzhledem k celkové jízdě řidiče. Neboli na základě získané akcelerace by jsme měli být schopni určit agresivní zatočení.

- Driving style and traffic measures—influence on vehicle emissions and fuel consumption[9] - Studuje vliv řidičského stylu na spotřebu vozidla a emise. V rámci této práce zkoumají program Nizozemské vlády na ekonomické jezdění zvaný "New Driving Style". Z tohoto programu vzešlo několik typů na ekonomickou jízdu. Nejdříve byly emise a spotřeba sledovány bez těchto typů a následně po jejich zavedení.

  1 Řadit co nejdříve je to možné na co největší možný převodový stupeň, kde jako možné maximum otáček motoru je doporučeno 2500 otáček za minutu pro benzínová vozidla a 2000 otáček za minutu pro naftové automobily.

  2 Stlačení plynového pedálu co nejrychleji a nejdůrazněji jak je to možné, aby řidiči vyhověli situaci v silničním provozu.

  3 Nepodřazovat na nižší převodový stupeň příliš brzo a nechat auto jet co nejdéle bez využití spojky na nejvyšším možném převodovém stupni.

  Bohužel se jim nepodařilo prokázat spojení mezi typem 2 a šetřením paliva, nejspíše protože tento typ byl buď špatně pochopen nebo vůbec využit. Avšak při dodržování tipů 1 a 3 dochází ke snížení spotřeby paliva mezi 5-25%. Výsledkem jejich zkoumání je ale také obecná rada jezdit co nejvíce plynule jak je to možné v rámci dopravy.

3.11.2 Algoritmus na ohodnocení uživatelské jízdy

Bohužel, nemáme k dispozici řadu informací používaných v předchozích pracích jelikož OBD2 jednotka k těmto informacím nemá přístup. Data, která jsme schopni o provozu vozidla získat, jsou následující.

- Rychlost vozidla
- Otáčky motoru
- Pozice plynového pedálu
- MAF(Mass Air Flow) neboli množství nasávaného vzduchu
- Teplota motoru
- GPS souřadnice
- Údaje z akcelerometru

S využitím výše zmíněných dat jsme navrhli několik parametrů, podle kterých může být řidičský výkon objektivně hodnocen. Každý hodnocený parametr je ohodnocen výsledným skórem od 0 (nejhorší) po 100 (nejlepší) možného dosaženého výsledku.

1 Spotřeba vozidla - Řidiči je vypočítána spotřeba paliva vozidla na kilometr, které při své jízdě dosáhl. Následně je porovnán s nejnižší doposud dosaženou spotřebou pro dané vozidlo.

   - řidičova > nejnižší => skóre = (nejnižší / řidičova) * 100
   - řidičova <= nejnižší => skóre = 100 a nejnižší = řidičova

   Jelikož nejnižší možná spotřeba se u každého automobilu liší ať už od výroby nebo na základě jeho technického stavu, je nutné dodržet počítání skóre dle nejnižší spotřeby pouze u automobilu, ze kterého byly údaje získané.

2 Otáčky motoru - Skóre pro otáčky motoru je vypočítáno jako poměr doby, po kterou jsou otáčky přes maximální povolenou hranici a doby jízdy vozidla. Maximální povolená hranice je nastavena na 2500 pro benzínové a 2000 pro naftové automobily.

3 Akcelerace, brzdění a zatáčení - Data získaná z akcelerometru v m/s^2 využíváme k výpočtu skóre za jízdu, kde hodnoty překračující 1,5 m/s^2 bereme jako nejhorší možné skóre neboli 0 až po nejlepší možnou hodnotu 0 m/s^2 což je nejlepší možné skóre neboli 100. K vyhodnocení tohoto pravidla využíváme společný vektor akcelerace.

4 Překročení maximální povolené rychlosti - Skóre je přiděleno na základě doby, kterou řidič jel rychlostí vyšší než povolená. V případě, že informace o maximální rychlosti pro danou oblast není k dispozici, je defaultní maximální rychlost nastavena na 50 km/h.

3.12 Vyhodnocení algoritmu

Díky výše definovaným parametrům můžeme začít s vyhodnocením algoritmu na snížení objemu dat zasílaných na server, jelikož můžeme vyhodnotit na kolik snížení objemu dat změní výslednou hodnotu.

3.12.1 Vyhodnocení percentuálního algoritmu

Percentuální algoritmus byl využit pouze v počátečních stádiích projektu, jelikož byl navržen pro práci s daty od OBD jednotky. Nelze ho využít na zmenšení počtu dat u GSP souřadnic a také jeho účinnost při práci s daty ohledně akcelerace vozidla byly nedostačující. V systému byl ponechán jako ukázka možného rozšíření pomocí dalšího algoritmu na snížení objemu dat. Z těchto důvodu dále již pouze uvedu doporučené nastavení 3.3 pro tento algoritmus.

3.12.2 Vyhodnocení Ramer-Daugler-Paucker algoritmu

Pro vyhodnocení účinnosti Ramer-Daugler-Paucker algoritmu jsme využili 5 testovacích jízd. Data 1 až 3 byla nasbírána pomocí automobilu Audi 1.9 litru nafta a Data 4 až 5 automobilem Octavia V2 2.0 litru benzín. Automobil Octavia je navíc upraven na LPG, což mělo vliv na výpočet spotřeby automobilu. Pro stanovení účinnosti algoritmu po ořezání bylo nutné definovat podle čeho budeme jednotlivé údaje porovnávat.

Tabulka 3.3: Nastavení pro Procentuální algoritmus

Údaj	Nastavení
Otáčky motoru	8
Rychlost	6
Akcelerace	50
Teplota motoru	0,1
Průtok vzduchu	30

Pro GPS souřadnice a rychlost jsme spočítali ujetou vzdálenost v metrech. Pro RPM jsme počítali dobu, po kterou automobil udržoval doporučené RPM. Neboli pro data 1 až 3 byla hranice 1500 rpm, jelikož se jednalo o naftový automobil a pro data 4 až 5 2000, jelikož se jednalo o automobil na benzín potažmo LPG. Pro průtok vzduchu jsme vypočetli spotřebované palivo v litrech za kilometr. Tento výpočet nemusí být úplně přesný, jelikož musíme vycházet z vypočítané vzdálenosti pomocí rychlosti vozidla a tyto údaje mohou být sami nepřesné. U teploty motoru jsme spočítali průměrnou teplotu motoru za jízdu. Data ohledně akcelerace vozidla v tomto případě byla speciální, jelikož údaje z Dat 1 až 3 nebylo možné použít, protože nebyly nijak upravené a obsahovali gravitační složku. Proto následné údaje o akceleraci vozidla jsou vyvozeny pouze z Dat 4 až 5, které byly zachyceny jedním typem vozidla a jedním řidičem což mohlo způsobit negativní dopad na výsledky měření

V tabulce 3.4 je vidět takovéto porovnání pro rychlost vozidla. Epsilon je nastavení naší tolerance. Nastavení na -1 bylo přidáno aby bylo vidět výsledky z neupraveného záznamu, jelikož již při nastavení algoritmu na epsilon 0 již dojde ke zmenšení počtu údajů.

Tabulka 3.4: Nastavení pro RDP algoritmus

Epsilon		Data 1	Data 2	Data 3	Data 4	Data 5
Původní	Metry	13142.23	4103.58	1745.38	16685.78	18781.41
	Počet	2814	1654	1566	1244	1287
0	Metry	0,00	0,00	0,00	0,00	0,00
	Počet	53,45	47,76	56,64	21,06	21,68
0.1	Metry	-0,02	-0,17	-0,12	0,00	0,04
	Počet	60,16	55,80	64,18	27,97	28,90
0.2	Metry	-0,03	-0,17	-0,07	0,02	0,06
	Počet	61,09	57,01	64,69	29,10	30,15
0.3	Metry	-0,04	-0,15	-0,09	0,03	0,08
	Počet	61,27	57,32	64,81	31,03	31,39
0.4	Metry	-0,05	-0,08	-0,22	0,06	0,10
	Počet	66,45	63,24	69,73	33,68	35,74
0.5	Metry	-0,06	-0,01	-0,09	0,11	0,16
	Počet	71,71	69,65	75,10	40,03	40,64
0.6	Metry	-0,06	-0,33	0,43	0,08	0,19
	Počet	78,25	77,03	80,59	45,66	46,08
0.7	Metry	-0,07	-0,15	0,57	0,14	0,11
	Počet	82,55	83,68	84,10	49,36	50,12
0.8	Metry	0,29	-0,11	0,91	0,17	0,29
	Počet	87,95	87,18	87,87	52,73	52,68
0.9	Metry	0,24	-0,54	0,57	0,33	0,39
	Počet	91,40	89,96	90,87	54,34	54,55
1.0	Metry	0,28	-0,18	1,49	0,66	0,43
	Počet	93,03	91,35	92,85	56,67	57,58
1.1	Metry	0,25	0,10	1,67	0,70	0,64
	Počet	93,85	92,14	93,68	61,33	61,23
1.2	Metry	0,22	0,11	2,21	0,51	0,70
	Počet	94,24	92,44	94,32	62,62	62,39
1.3	Metry	0,46	-0,23	2,25	0,35	0,57
	Počet	94,71	93,11	94,38	64,07	63,56
1.4	Metry	0,59	-0,20	2,20	0,29	0,21
	Počet	95,02	93,35	94,44	66,16	66,05
1.5	Metry	0,89	0,04	2,52	0,44	0,10
	Počet	95,38	94,26	94,70	68,01	67,68

Hodnoty v rámci tabulky byly zaokrouhlené na dvě desetinná místa.

Metry představují rozdíl v % oproti původní hodnotě.

Počet představuje o kolik % se zmenšil počet údajů oproti původní hodnotě.

Podobné vyhodnocení bylo provedeno i u ostatních údajů a z nich bylo odvozeno doporučené nastavení algoritmu, které je vidět v tabulce 3.5. Tyto hodnoty byly vybrány tak, aby ztráta přesnosti nebyla vyšší než 5%. Toto pravidlo neplatí pro akceleraci vozidla vzhledem k enormnímu počtu údajů. Tabulky pro zbylé typy údajů je možné vidět v elektronické příloze v souboru RDP.

Tabulka 3.5: Doporučené nastavení pro RDP

	Nastavení
RPM	9.8
Rychlost	1.2
GPS	6.7
Akcelerace	4.3
Teplota	0
MAF	1.0

V tabulce 3.6 je uvedena maximální ztráta přesnosti v procentech a také o kolik se zmenšil počet údajů oproti nefiltrovaným hodnotám v procentech, při nastavení algoritmu na doporučené hodnoty filtru.

Tabulka 3.6: Hodnoty pro doporučené nastavení RDP

		Data 1	Data 2	Data 3	Data 4	Data 5
RPM	Přesnost *	4.70	4.42	0.14	0.08	0.12
	Počet **	83.19	71.22	66.09	30.47	27.97
Rychlost	Přesnost	0.22	0.11	2.21	0.51	0.70
	Počet	94.24	92.44	94.37	62.62	62.39
GPS	Přesnost	-0.04	-0.08	-0.41	-0.01	-0.01
	Počet	46.58	38.33	47.24	77.94	80.75
Akcelerace	Přesnost	NA	NA	NA	5.44	11.09
	Počet	NA	NA	NA	74.07	77.41
Teplota	Přesnost	0	0	0	0	0
	Počet	96.48	95.83	90.29	81.19	88.81
MAF	Přesnost	0.11	-0.38	0.68	-4.85	-4.17
	Počet	69.08	68.22	77.91	45.90	42.97

* Přesnost určuje o kolik % se hodnota po ořezání dat změnila oproti původní.

** Počet představuje o kolik % se zmenšil počet údajů oproti původní hodnotě.

Z tabulky 3.6 je také patrné, že algoritmus má větší účinnost na zmenšení počtu údajů

3.12.3 Porovnání ujeté vzdálenosti

Z těchto údajů je také možné provést porovnání výpočtu vzdálenosti pomocí GPS a rychlosti vozidla. Toto porovnání bylo provedeno pro Data 4 a 5, jelikož se jednalo o delší jízdy trvající průměrně 30 min a také jejich interval sběru údajů o rychlosti vozidla byl větší. Reálná vzdálenost byla určena pomocí měření vzdálenosti na stránkách mapy.cz, google.maps.com.

V tabulce 3.7 je vidět porovnání ujeté vzdálenosti. Ze získaných dat je možné usoudit. že výpočet ujeté vzdálenosti pomocí rychlosti vozidla je přesnější než z GPS souřadnic vozidla. Tento výsledek ale nemusí být prokazatelný, jelikož se jedná pouze o dvě jízdy se stejným typem vozidla a stejným Androidím zařízením. Pro lepší prokazatelnost tohoto výsledku by bylo třeba tento test provést s větším počtem automobilů a různými Androidími zařízeními. V tabulce 3.8 jsou vidět výsledky porovnání ujeté vzdálenosti před použitím filtru a po jeho použití.

Tabulka 3.7: Porovnání ujeté vzdálenosti

	Data 4	Data 5
Vzdálenost skutečná *	17200.00	19000.00
Vzdálenost dle GPS	21679.60	23007.11
Vzdálenost dle rychlosti **	16770.60	18913.41

* Změřená dle google.maps.com a mapy.cz

** Rychlost získaná od ECU.

Tabulka 3.8: Porovnání ujeté vzdálenosti vzhledem k filtrování

		Data 4	Data 5
Před filtrování	Vzdálenost skutečná *	17200.00	19000.00
	Chyba dle GPS v %	26.60	21.11
	Chyba dle rychlosti v % **	-2.99	-0.46
Po filtrování	Chyba dle GPS v %	26.04	21.09
	Chyba dle rychlosti v % **	-2.50	-0.46

* Změřená dle google.maps.com a mapy.cz

** Rychlost získaná od ECU.

Tabulka 3.9: Přesnost GPS záznamů

	Přesnost GPS záznamů v metrech					
	0 - 3	4 - 5	6-15	16 - 20	21 - 39	40 +
Jízda 1	0	22	557	4	1	1
Jízda 2	0	7	269	63	8	0
Jízda 3	0	0	241	60	25	0
Jízda 4	1685	52	0	3	0	1
Jízda 5	1781	42	3	3	0	0

3.12.4 Úprava GPS dat

Získané GPS souřadnice slouží k zobrazení cesty vozidla, ale také k výpočtu ujeté vzdálenosti. Výpočet ujeté vzdálenosti je silně ovlivněn přesností GPS souřadnic. Pokud například automobil vjede do tunelu, může i nadále poskytovat GPS souřadnice avšak jejich přesnost může být až ve stovkách metrů, což silně ovlivní výpočet ujeté vzdálenosti. K minimalizaci této chyby je proto dobré odstraňovat GPS souřadnice, které překročily limit námi zadané přesnosti.

Na základě dříve získaných testovacích jízd, jsme se rozhodli nastavit maximální přesnost GPS záznamů na 20 metrů. V tabulce 3.9 jsou vidět jednotlivé testovací jízdy a přesnost GPS záznamů. Z tabulky je také zřejmé, že Jízda 1 až 3 byly zaznamenané jiným Androidím zařízením než Jízdy 4 až 5. Pro Jízdy 1 až 3 to byl telefon HTC One S s androidem 4.1.1, kde se přesnost většiny záznamů pohybovala mezi 6 až 20 metry. Pro jízdy 4 až 5 by využit tablet Nvidia Shield s androidem 6.0 kde se přesnost většiny záznamů pohybuje mezi 3 až 5 metry.

V úvahu byla také brána možnost odstranění GPS záznamů, jejichž přesnost by přesáhla 15 metrů. Toto by ovšem mělo velký vliv na jízdy 2 a 3, kde by bylo až 35% záznamů odstraněno. Zároveň by to také znamenalo, že by až 22 po sobě jdoucích záznamů bylo odstraněno. Vzhledem k tomu že v době získávání těchto záznamů automobil cestoval městskou zástavbou a nikoli například tunelem kde očekáváme zhoršení přesnosti GPS není v našem zájmu tyto záznamy odstranit.

Z tohoto důvodu by měl být proveden test přesnosti GPS u Androidích zařízení, které by byly uvažovány jako palubní jednotka pro firemní flotilu.

Tabulka 3.10: Poměr vzduchu a paliva

	Poměr vzduchu a paliva
Benzín	14.7:1
Diesel	14.6:1
Methanol	6.4:1
Ethanol	9:1
LPG	15.5:1

Tabulka 3.11: Spotřeba vozidla na 100 km po ořezání dat

	Jízda 1	Jízda 2	Jízda 3	Jízda 4	Jízda 5
Spotřebované palivo v litrech	1.151274	0.513757	0.641295	0.93941	0.936443
Ujetá vzdálenost v metrech	13170.52	4107.93	1783.99	16770.6	18913.41
Spotřeba na 100 km	8.74	12.50647	35.947049	5.601528	4.951211

3.12.5 Spotřebované palivo

Jednou z důležitých informací pro každého správce firemní flotily je samozřejmě spotřeba vozidla. Spotřebu vozidla je možné vypočítat z průtoku vzduchu neboli MAF (Mass Air Flow), což představuje váhu nasátého vzduchu. Tento údaj vrací OBD2 zařízení v gramech za sekundu. Získané hodnoty vynásobíme časem a sečteme, abychom získali celkový průtok vzduchu za jízdu v gramech. Následně převedeme na litry pro lepší představu a vydělíme konstantou pro poměr vzduchu a paliva z tabulky 3.10. Tímto způsobem vypočítáme spotřebované palivo v litrech na jízdu.

Aby hodnota spotřebovaného paliva mohla být účinně interpretována a vyhodnocena, je třeba spotřebované palivo převést na spotřebu na 100 km, která se běžně uvádí u většiny vozidel. Pro výpočet spotřeby na 100 km potřebujeme ale také ujetou vzdálenost, pro naši potřebu využijeme vzdálenost vypočítanou pomocí rychlosti vozidla, jelikož je to nejlepší údaj, který máme jak je ukázáno v kapitole 3.12.3. To také ale znamená, že výpočet spotřeby bude ovlivněn nejen chybou při ořezávání MAF, ale také chybou při výpočtu ujeté vzdálenosti. V tabulce 3.12 je vidět tento výpočet před ořezáním dat a v tabulce 3.11 je vidět tento výsledek po ořezání dat. Nakonec v tabulce 3.13 je vidět procentuální rozdíl mezi těmito výpočty.

Jízdy 1 až 3 byly zaznamenané na automobilu Audi 1.9 diesel, kde pro jízdu 1 spotřeba odpovídá reálné spotřebě, avšak pro jízdu 3 je vypočítaná spotřeba 35 litrů, což je mnohem výše než průměrná spotřeba. Toto je způsobeno tím, že se jednalo o velice krátkou agresivní jízdu a také že auto stálo nějakou chvíli před vyjetím.

Jízdy 4 až 5 byly zaznamenány na automobilu Škoda Octavia V2 2.0 benzín, které je upraveno na LPG. Výpočet spotřeby u tohoto auta je velice nepřesný, jelikož při nastartování

Tabulka 3.12: Spotřeba vozidla na 100 km před ořezání dat

	Jízda 1	Jízda 2	Jízda 3	Jízda 4	Jízda 5
Spotřebované palivo v litrech	1.149961	0.515761	0.636935	0.987298	0.9772
Ujetá vzdálenost v metrech	13142.23	4103.583	1745.387	16685.78	18781.41
Spotřeba na 100 km	8.750120	12.56855	36.492479	5.917002	5.203017

Tabulka 3.13: Rozdíl ve spotřebě paliva na 100 km před a po ořezání

	Jízda 1	Jízda 2	Jízda 3	Jízda 4	Jízda 5
Před ořezáním	8,750120	12,568552	36,492479	5,917002	5,203017
Po ořezání	8,741294	12,506472	35,947049	5,601528	4,951211
Rorzdíl v %	-0,10	-0,49	-1,49	-5,33	-4,84

automobilu s nezahřátým motorem je použit benzín a teprve po nějaké době kdy se motor dostatečně zahřeje se začne používat LPG. Spotřeba zde neodpovídá průměrné spotřebě, která je u tohoto automobilu okolo 8 litrů na 100 km. V rámci obou těchto jízd byly ale také zaznamenány chybové stavy P0171 [23] (Příliš chudá směs) a P0172 [24] (Příliš bohatá směs), pro obě tyto závady je uvedená možná náprava opravením či očištěním senzoru MAF. Dle článku [15] může právě zašpiněný či poškozený senzor poskytovat špatné většinou nižší hodnoty, což vedlo ke špatnému výpočtu spotřeby paliva.

3.12.6 Poruchy získané od ECU

Opravy automobilu mohou být velice nákladné, pokud nejsou poruchy zjištěny včas a mohou stát i lidské životy či v případě dlouhodobého provozu s danou chybou mohou způsobit dlouhodobé poškození součástek automobilu. Pomocí protokolu OBD2 můžeme získat informace o poruchách vozidla, které jsou od poruchy hnací jednotky až po komunikační rozhraní. Kód poruchy je normován ISO 15031-6 popřípadě SAE J20125, je definován 5 znaky. Vysvětlení, co tyto znaky znamenají, je možné vidět v tabulce 3.14.

Tyto informace OBD2 jednotka vrací v takzvaném Single Frame formátu, neboli je vrací v rámcích, kde se každý rámec skládá ze 7 bajtů. První bajt obsahuje informace o tom, o jaký mód se jedná a stav odpovědi. Zbylé bajty jsou rozděleny po dvou, kde každá tato dvojice určuje právě jeden DTC kód. První dva bity popisují systém, druhé dva bity typ kódu, následující čtveřice bitů popisuje, kde porucha vznikla a poslední osmice bitů popisuje specifikaci chyby.

Speciálním DTC kódem je P0000, který říká, že je vše v pořádku nebo také že není chyba k nahlášení. Tento kód se může objevit v případě užívání špatného ELM327 zařízení. Pokud jednotka nemá žádné chyby uložené, měla by vrátit řetězec "NO DATA".

Tabulka 3.14: Vysvětlení DTC znaků

	Znak	Popis
Systém	P	Pohonná jednotka
	B	Výbava
	C	Šasi
	U	Komunikační rozhraní
Typ kódu	0	Kódy SEA J2012
	1	Kódy výrobce
	2 - 3	Nespecifikované
Kde porucha vznikla	0	Volné
	1	Řízení a kontrola emisí
	2	Vstřikování
	3	Systém zapalování
	4	Agregáty emisí
	5	Kontrola rychlosti
	6	Palubní řídící systém a vstupní obvody
	7 - 9	Převodovka
	A - C	Pro hybridní pohony
	D - F	Volné
Specifikace chyby	00 - FF	Označení chyby

Kapitola 4

Serverová část návrh a implementace

V této kapitole bude čtenář seznámen s návrhem a implementací serverové části projektu.

Na projektu ve stejné době pracuje Michal Polata v rámci své bakalářské práce. Proto jsem svoji činnost v rámci serverové části projektu omezil na práci s rozhraním potřebným pro palubní jednotku a přidání obrazovek pro úpravu nově přidaných entit.

4.1 Výběr vývojových prostředí

Pro implementaci palubní jednotky, která je psaná v jazyce Java na platformě Android, jsem vybral vývojové prostředí Android Studio, jelikož se jedná o oficiální vývojové prostředí pro Android. Toto prostředí také obsahuje emulátor zařízení Android, kde by bylo možné otestovat aplikaci na více zařízeních. Emulátor ovšem nepodporuje emulaci Bluetooth a je také mnohem pomalejší než reálné zařízení, z těchto důvodů nebyl emulátor využit a většina testování byla provedena na reálných zařízeních. Android Studio 2.0 má také přímou integraci na Cloud Test Lab, takže je možné z vývojového prostředí spouštět instrumentální testy oproti Google Test Lab, více v kapitole 6. V neposlední řadě je také Android studio založeno na základě vývojového prostředí IntelliJ IDEA, které jsem vybral pro implementaci serverové části.

Pro implementaci serverové části projektu bylo vybráno vývojové prostředí Intellij IDEA Ultimate, tato verze vývojového prostředí je placená, ale pro studenty ČVUT je možné jej využívat zdarma pro školní účely. Při využití pluginu SBT[27] také plně podporuje vývoj Play frameworku, kde je možné aplikaci spouštět přímo z vývojového prostředí. Také obsahuje podporu pro testování restového rozhraní. Neposlední výhodou je také to, že jak vývojové prostředí pro serverovou část tak i pro palubní jednotku je založeno na stejném základě, takže má společné zkratky a další funkce.

4.2 RESTové rozhraní

Pro komunikaci mezi palubní jednotkou a serverem je třeba vytvořit RESTové rozhraní, které používá formátu JSON. O jaký automobil se jedná je odvozeno z "name"v URL a "Secret-key"v hlavičce HTTP, podle kterého je vybrána správná palubní jednotka (boardUnit), pokud není nalezena palubní jednotka na serveru se stejným jménem a tajným klíčem je vrácena chybová odpověď. Pokud je palubní jednotka nalezena, je provedena kontrola, zda má přiřazený automobil, pokud nemá, je vrácená chybová odpověď. Pro parametr "name"je použito jméno palubní jednotky místo id, aby nebylo id záznamu šířeno mimo serverovou část.

4.2.1 Přihlášení uživatele

GET /boardUnit/:name/obdPids ? lastUpdateTime: Long

Pro přihlášení uživatele je na server zasláno jméno a heslo uživatele. Podle těchto údajů je vyhledán uživatel a následně je zkontrolováno, zda uživatel má právo využívat palubní jednotku, pod kterou byl požadavek na přihlášení zaslán. Pokud je uživatel nalezen a má právo využívat danou palubní jednotku, je zaslána odpověď s informací, zda se jedná o obyčejného uživatele nebo administrátora, jelikož administrátor má větší práva v rámci palubní jednotky. Pokud uživatel není nalezen je vrácena odpověď se statusem 403 a chybou která obsahuje informaci co se stalo.

4.2.2 Přijetí dat o jízdě automobilu

POST /boardUnit/:name/trip

Záznamy o jízdě automobilu jsou zasílány na server ve skupině, která obsahuje maximálně 100 záznamů. Každá skupina obsahuje uživatelské jméno (user) a identifikaci, o kterou jízdu (trip) se jedná. Identifikátor jízdy je složen z uživatelského jména a doby, kdy byla jízda zahájena v milisekundách.

Jednotlivé záznamy o jízdě obsahují čas získání tohoto záznamu, kód, pomocí kterého je tento záznam identifikován a objekt s názvem json. Obsah atributu json záleží na kódu, pro kódy "gps"neboli GPS souřadnice vozidla a "acc"neboli akcelerace vozidla, je použitá jiná struktura než pro záznamy od OBD.

Ukázka vzorového postu palubní jednotkou na server:


```
{
   "trip":"pepa1462557338026",
   "user":"pepa",
   "records":[
      {
         "time":"1451047180526",
         "code":"obd_speed",
         "json":{
            "value":"125"
         }
      },
      {
         "time":"1451047180528",
         "code":"gps",
         "json":{
            "lat":"14.59887445",
            "lng":"50.485789788",
            "alt":"450",
            "acc":"16",
            "speed":"100"
         }
      },
      {
         "time":"1451047180536",
         "code":"acc",
         "json":{
            "x":"0.18784413",
            "y":"0.55343541",
            "z":"0.05684684"
         }
      }
   ]
}
```

4.2.3 Synchronizace nastavení OBD

GET /boardUnit/:name/obdPids ? lastUpdateTime: Long

POST /boardUnit/:name/obdPids

Palubní jednotka umožňuje získávání různých dat od OBD jednotky, ale aby věděla jaké data má sbírat je nutné ji předat seznam nastavení těchto OBD pidů. U každého OBD pidu uvádíme "name"tohoto typu, což je pouze informace pro uživatele a systém ji nijak nevyužívá. Dále "tag", podle kterého identifikujeme o jaký typ OBD příkazu se jedná, tento údaj musí být pro automobil unikátní. Dále "pidCode", který je použit pro samotný příkaz na OBD jednotku tedy například 010D1. Hodnota od OBD jednotky je vrácena v binární formě a je třeba jej přepočítat na reálnou hodnotu k čemu slouží položka "formule". Pro vrácenou hodnotu také uvádíme "max"a "min", neboli maximum a minimum dané hodnoty. Aby bylo možné zrušit příkaz bez nutnosti jej smazat, je možné je deaktivovat pomocí položky "active". Poslední položkou je "updateTime", který představuje datum a čas v milisekundách, kdy byl záznam upraven.

Při GET operaci je také předáván URL parametr "lastUpdateTime", který určuje poslední čas změny záznamů. Pokud má serverová strana uloženy záznamy s novějším časem změny, jsou vráceny všechny záznamy, pokud je čas změny na serverové straně starší, je vrácená chybová odpověď s informací, že by měly být záznamy aktualizovány.

Při operaci POST je provedena kontrola, zda serverová část nemá uloženy záznamy s novějším časem úpravy, v případě že tomu tak je vrácená chybová odpověď která informuje, že serverová část má novější záznamy. Pokud má serverová část starší záznamy, jsou aktuální záznamy smazány a jsou nahrazeny nově přijatými.

Ukázka vzorového postu palubní jednotkou na server:

```
{
  "records":[
     {
        "name": "Rychlost",
        "tag": "obd_speed",
        "pidCode": "010D1",
        "formula": "A",
        "min": "-40",
        "max": "255",
        "active": true,
        "updateTime": "1255520000"
     }
  ]
}
```

4.2.4 Synchronizace nastavení procesů automobilů

GET /boardUnit/:name/carSettings ? lastUpdateTime: Long

POST /boardUnit/:name/carSettings

Palubní jednotka má několik procesů, které jsou v určitém časovém intervalu spouštěny, například odesílání záznamů o jízdě na server. Toto rozhraní umožňuje provést nastavení tohoto časového intervalu na palubní jednotce.

Každý záznam se skládá z "code", což je označení nastavení, jehož hodnota musí pocházet z výčtu typů kódů (CarSettingEnum). Následně atribut "value"představuje samotné nastavení časového intervalu v sekundách. Zbylé dva atributy včetně chování těchto rozhraní je stejné jako u Synchronizace nastavení OBD 4.2.3.

Ukázka vzorového postu palubní jednotkou na server:

```
{
  "records":[
     {
        "code": "FILTER_SETTING_SYNC_PERIOD",
        "value": "60",
        "active": true,
        "updateTime": "1255520000"
     }
  ]
}
```

4.2.5 Synchronizace nastavení filtrů

GET /boardUnit/:name/filterSettings ? lastUpdateTime: Long

POST /boardUnit/:name/filterSettings

Tato rozhraní slouží k nastavení filtrování záznamů od OBD jednotky. O jaký algoritmus se jedná je možné vybrat v atributu "algorithm", jehož hodnota musí být z výčtového typu (FilterEnum). Následuje informace o jaký typ záznamů se jedná v atributu "tag"jehož hodnota musí odpovídat některému ze záznamů z OBD Pid, nebo také "gps", "acc". Atribut "value"obsahuje nastavení pro algoritmus, který provádí samotné filtrování. Zbylé dva atributy včetně chování těchto rozhraní je stejné jako u Synchronizace nastavení OBD 4.2.3.

Ukázka vzorového postu palubní jednotkou na server:

```
{
  "records":[
     {
        "algorithm": "FILTER_SETTING_SYNC_PERIOD",
        "tag": "obd_speed",
        "value": "2.2",
        "active": true,
        "updateTime": "1255520000"
     }
  ]
}
```

4.2.6 Přijetí chybových údajů automobilu

GET /boardUnit/:name/dtcs

Toto rozhraní slouží k uložení informace o chybovém stavu automobilu. Obsahuje informaci, při které jízdě došlo "trip", v jakém čase "time"a samotný kód obsahující chybu "code".

Ukázka vzorového postu palubní jednotkou na server:

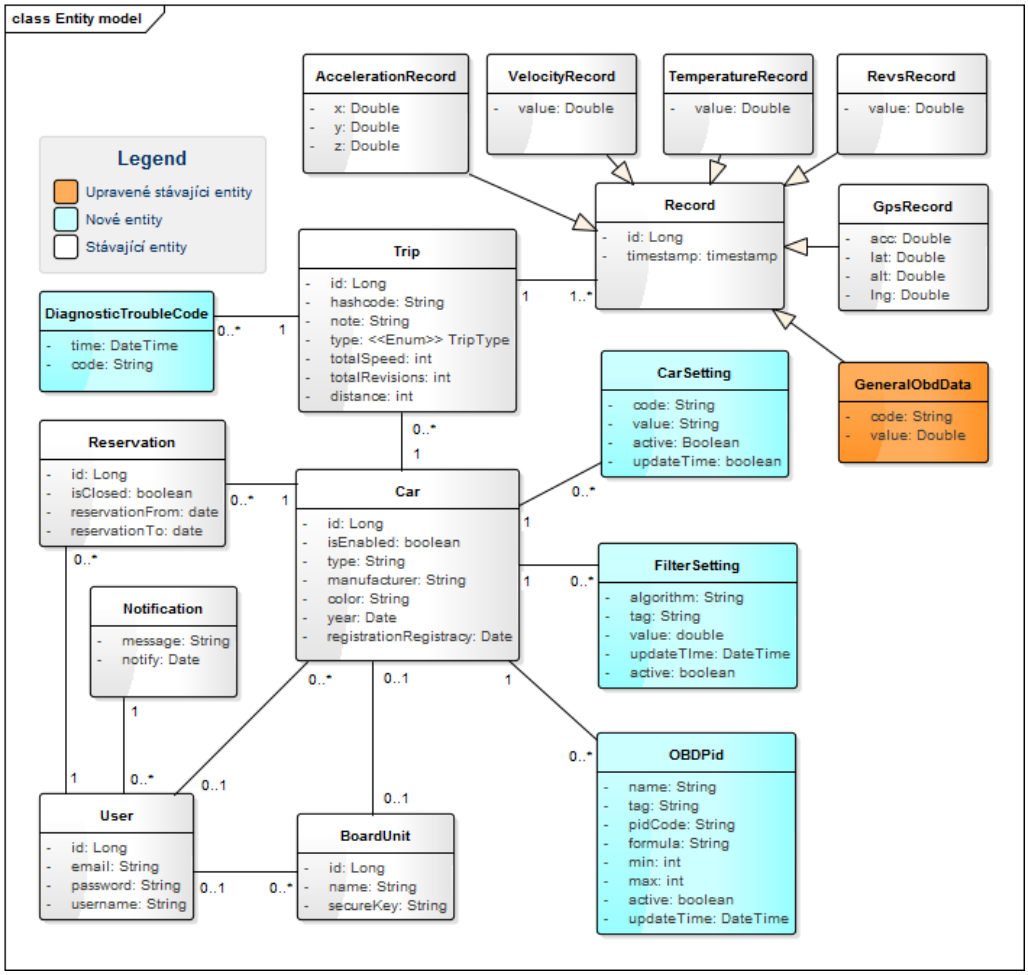
```
{
  "records":[
     {
        "trip": "pepa1462557338026",
        "time": "1462557338026",
        "code": "P0171"
     }
  ]
}
```

4.3 Úprava databáze

Vzhledem k novým požadavkům na funkčnost serverové části bylo nutné upravit či vytvořit nové entity, které slouží k ukládání dat do databáze.

K ukládání dat o jízdě k automobilu slouží několik entit, jejichž název končí na Record. Například VelocityRecord slouží k uložení informace o rychlosti automobilu pro které platí tag "obd_speed". Pokud přijde tag, jehož záznam není rozpoznán, je tento záznam uložen do entity GeneralObdData. Zde musela být provedena úprava, jelikož zde nebyl ukládán tag, o který záznam se jedná, takže následně nebylo možné s těmito údaji pracovat.

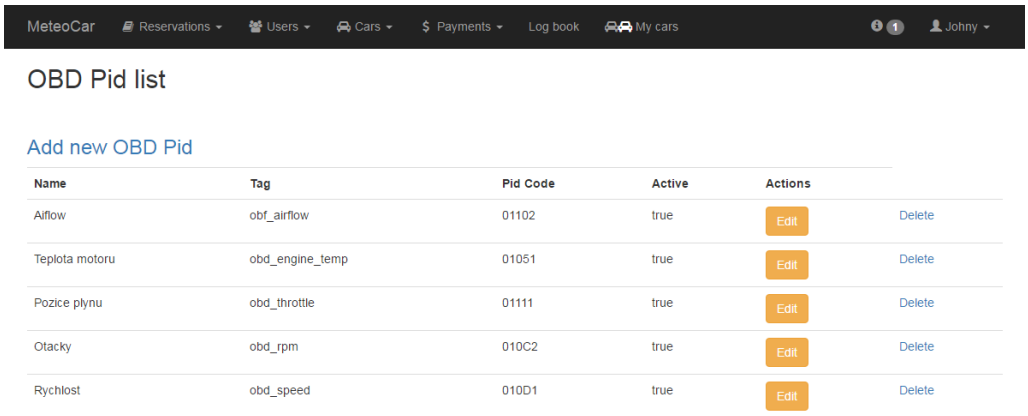
Bylo přidáno také několik nových entit. DiagnosticTroubleCode slouží k uchování informace o chybě získané od ECU automobilu. CarSetting slouží k uložení nastavení pro procesy palubní jednotky. FilterSettings slouží k uložení nastavení filtru použitého pro filtrování získaných dat na palubní jednotce. OBDPid uchovává nastavení jaké data chceme od palubní jednotky získávat.



Obrázek 4.1: Model entit

4.4 Zobrazení nastavení

Aby bylo možné spravovat nastavení palubní jednotky, bylo vytvořeno několik obrazovek pro úpravu nastavení. Na obrázku 4.2 je vidět toto nastavenií pro OBD Pid.



Obrázek 4.2: Zobrazení nastavení OBD Pidů

Kapitola 5

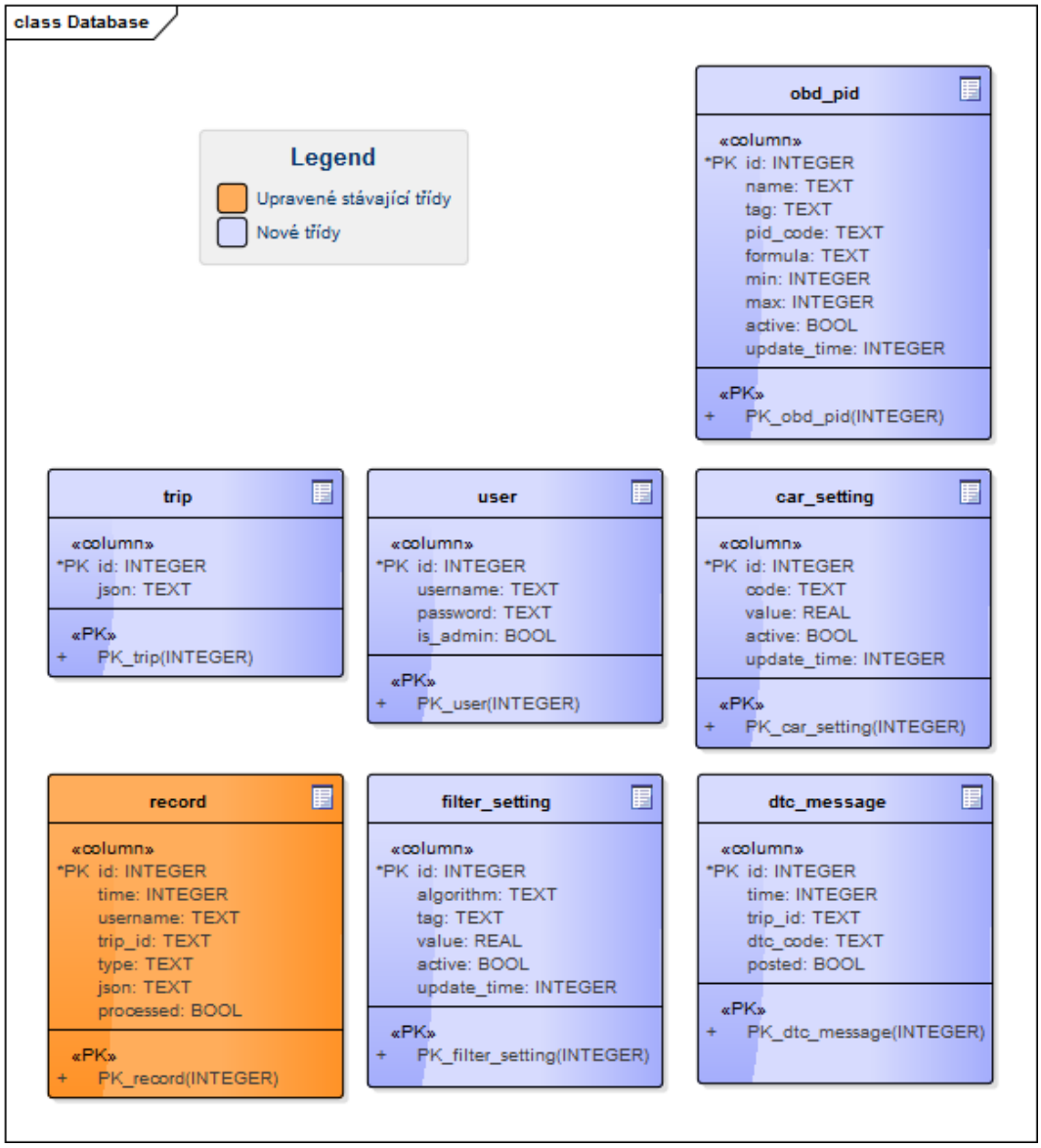
Palubní jednotka - návrh a implementace

V této kapitole bude čtenář seznámen s návrhem a implementací serverové části projektu.

5.1 Databázoví model

Vzhledem k novým požadavkům bylo třeba rozšířit stávající databázi o nové tabulky viz 5.1. Databáze je implementována nad databází Android SQLite.

Tabulka "record"slouží k ukládání nově přijatých záznamů, tato tabulka platí pro záznamy s různými obsahy, například pro záznam o GPS souřadnicích je hodnota uložena do JSON formátu, který je následně uložen do parametru json jako řetězec. Tabulka "trip"slouží k uložení už zpracovaných částí jízd, které jsou připraveny k odeslání na server. Tabulky "obd_pid", "filter_setting", "car_setting"slouží k uložení jednotlivých nastavení palubní jednotky. Tabulka "user"ukládá jméno a heslo uživatele, který se přihlásil do aplikace, aby ho bylo možné následně v offline módu přihlásit. Poslední tabulkou je "dtc_message", která uchovává záznamy o chybových stavech vozidla.



Obrázek 5.1: Databázový model palubní jednotky

5.2 Přihlášení do aplikace

Jelikož se jedná o aplikaci na správu firemní flotily, je potřeba, aby se uživatel do aplikace přihlásil, aby bylo možné dané jízdy přiřadit k jednotlivým uživatelům. Jelikož je možné, že automobil může stát i v podzemních garážích, je tedy třeba, aby se uživatel mohl do aplikace přihlásit i bez aktivního připojení na internet.

Pokud má palubní jednotka aktivní připojení na internet, je uživatelské jméno a heslo ověřováno proti serveru. Odpověď obsahuje také informaci, zda má uživatel administrátorská práva. Při úspěšném ověření uživatele je uživatelské jméno a heslo uloženo do databáze pro případné offline přihlášení uživatele. Jelikož dotaz na server může trvat delší dobu, je třeba tento dotaz dělat z jiného než hlavního vlákna aplikace, jinak by mohlo dojít k zaseknutí aplikace.

Tento problém byl v aplikaci řešen pomocí speciálního vlákna, kterému se nové dotazy ukládaly do fronty. Toto vlákno zpracovalo všechny dotazy připravené ve frontě a poté se na určitou dobu uspalo. Avšak toto řešení mělo negativní dopad na výkon aplikace, proto jsem vlákno nahradil třídou AsyncTask, která běží ve vlastním vlákně a je právě určena na déle trvající úkony a nemá tak velký negativní dopad na výkon aplikace.

Uživatel s administrátorskými právy má navíc možnost měnit nastavení palubní jednotky. Do nastavení palubní jednotky je také možné se přihlásit aktuálním jménem a tajným klíčem palubní jednotky. Tato možnost byla přidána, aby bylo možné provést prvotní nastavení palubní jednotky a pro technické pracovníky, kteří by mohli chtít přenastavit palubní jednotku.

5.3 Zpracování dat o provozu automobilu

Data o provozu automobilu jsou ukládána do databázové tabulky "record", kde je k ukládaným záznamům nastavena hodnota, že záznamy nebyly zatím zpracovány. V předem nastavené periodě je spouštěno vlákno starající se o přípravu dat k odeslání na server.

Nejdříve zjistí, jestli jsou v databázi nějaké záznamy o provozu automobilu, které nebyly ještě zpracovány. Pokud žádné takové záznamy nejsou uloženy, tak vlákno končí. Pokud takové záznamy existují, tak vybere jízdy, pro které existují nezpracované záznamy. Následně pro každou nalezenou jízdu hledá množinu nezpracovaných typů záznamů. Pro daný typ záznamů vybereme nastavení filtru, pokud žádné nastavení filtrů pro daný typ není definováno, tak dále pracujeme se všemi záznamy. Pokud takové nastavení nalezneme, je množina záznamů zpracována filtrem, který nám vrátí množinu záznamů ořezanou o nepotřebné záznamy. Nepotřebné záznamy následně smažeme z databáze Androidu. Zbylým záznamům nastavíme, že již byly zpracovány a necháváme je v databázi, jelikož by jsme s nimi mohli chtít v budoucnosti pracovat. Například ukázat uživateli po skončení jízdy celkovou ujetou vzdálenost a spotřebované palivo.

Vyfiltrované záznamy převedeme do JSON formátu pro odeslání na server a uložíme je do tabulky "trip". Druhé vlákno spouštěné opět v předem nastavené periodické době se snaží tyto záznamy odeslat na server. Po úspěšném odeslání záznamu na server je tento záznam z tabulky smazán.

5.4 Získávání chybových stavů automobilu

Chybové stavy automobilu jsou získávány od OBD jednotky. Pro získávání OBD údajů slouží vlákno, které má frontu připravených dotazů na OBD jednotku. V každé iteraci se provedou všechny dotazy na OBD jednotku a iterace je opět opakovaná.

Do této fronty je třeba přidat příkaz na zjištění chybových stavů automobilu. Z tohoto důvodu je spuštěno další vlákno, které v periodické době přidá tento dotaz do fronty před začátkem další iterace dotazů. Po zavolání dotazu na chybové stavy automobilu je tento dotaz odstraněn z fronty dotazů. Pro ostatní dotazy na OBD jednotku platí, že po obdržení odpovědi je vypočítaná reálná hodnota dle vzorce, který je uveden pro tento typ dotazu. Pro chybové stavy ovšem toto neplatí, proto odesíláme dále řetězec, který jsme přijali od OBD jednotky. Před uložením je tento řetězec převeden na kód chyby a je provedena kontrola, zda pro jízdu, kde jsme tuto chybu zachytili, již není uložená stejná chyba. Jelikož chyba zůstává uložená v ECU do doby, kdy je provedeno mazání chyb, nechceme ukládat jednu chybu vícekrát. Chybu následně uložíme do tabulky "dtc_message"s příznakem jestli již je odeslaná na server. Následně další vlákno vybírá ještě neodeslané zprávy na server.

5.5 Nastavení procesů palubní jednotky

Palubní jednotka má několik procesů, které v pravidelných intervalech provádějí nějakou práci. Mezi tyto procesy patří synchronizace nastavení pro filtry, OBD dotazy a nastavení automobilu. Dále proces pro přípravu záznamů na odeslání na server, proces, který připravené záznamy odesílá na server, proces co přidává do fronty dotaz na chybové stavy a proces, který chybové stavy odesílá na server.

Pro tyto procesy chceme mít možnost nastavit, jak často se mají provádět. Například pro běžné použití můžeme nastavit interval odesílání záznamů na server na 10 min, neboli každých 10 minut uvidíme na serveru data z jízdy. Pokud bychom například ale automobil testovali a chtěli bychom vidět všechny data co nejdříve, můžeme tento interval nastavit na 10 sekund.

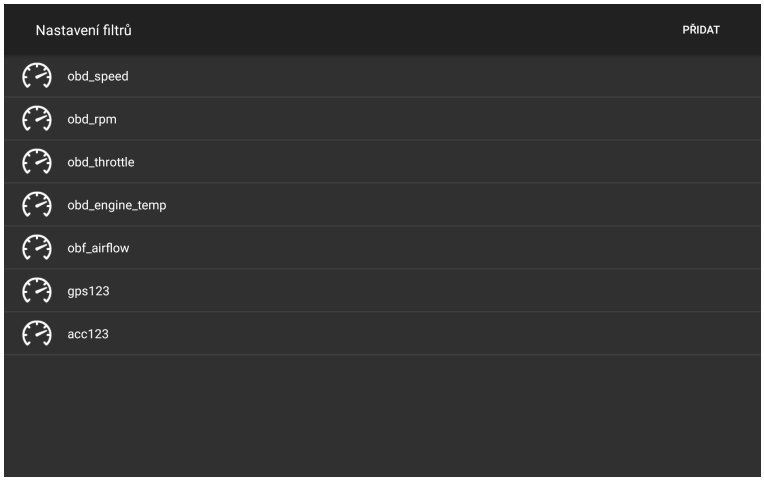
Pro tyto potřeby má Android API třídu TimerTask a servisu ScheduleExecutorService. Pomocí servisy tedy následně můžeme provést spuštění procesu s určitým nastaveným intervalem, kde funkčnost samotného procesu je obsažena v rámci třídy TimerTask. Můžeme také proces ukončit a znova ho spustit v případě přenastavení intervalu, v jakém se má proces provádět.

V rámci vývoje také přibyl požadavek, aby bylo možné nastavit, jaké procesy mají běžet. Jelikož tato palubní jednotka bude používaná i v rámci výuky, kde studenti budou analyzovat a implementovat serverovou část od nuly. Hlavním cílem bude vytvořit funkčnost, která umožní přihlášení uživatele a získávání jízd uživatele. Procesy na synchronizaci nastavení palubní jednotky a zasílání informací o chybových stavech vozidla je tedy možné vypnout. Ke splnění tohoto požadavku byl tedy přidán přepínač tohoto nastavení do menu.

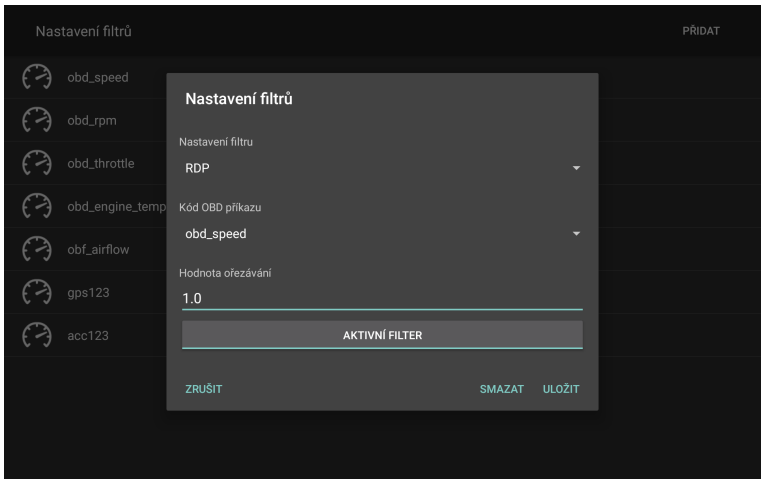
5.6 Nastavení aplikace

Pro funkčnost palubní jednotky je třeba primárně nastavit jméno, tajný klíč jednotky a adresu serveru. Při změně jména a tajného klíče palubní jednotky je třeba provést promazání databáze palubní jednotky, jelikož dříve uložené hodnoty již pro ní neplatí. Například údaje uživatelů pro offline přihlášení už nemusejí platit. Nastavení filtrů, procesů jednotky a OBD dotazů je obnoveno do základního nastavení.

Pro nastavení filtrů, procesů jednotky a OBD dotazů jsme vytvořili zobrazení, na kterých je možné tyto záznamy upravit. Na obrázku 5.2 je vidět obrazovka se seznamem nastavení filtrů a na obrázku 5.3 je vidět obrazovka pro úpravu či vytvoření nového filtru.



Obrázek 5.2: Seznam filtrů



Obrázek 5.3: Nastavení filtru

5.7 Práce s akcelerací vozidla

Akcelerace vozidla je získávána od mobilního zařízení pomocí SenzorManager, kde nastavíme, že chceme získávat data od senzoru typu "TYPE_ACCELEROMETER". Data získaná od tohoto senzoru ovšem obsahují také gravitační složku a mohou být ovlivněna také tím, jak je palubní jednotka nainstalovaná ve vozidle. Z tohoto důvodu musíme použít low-pass filter, kterým odstraníme tyto vedlejší složky. Tyto údaje je třeba dále následně upravit, aby byl odstraněn šum z dat. K tomu použijeme Mean smoothing filter, který na základě posledních 20 údajů vrátí střední hodnotu záznamu a tu dále použijeme.

Jelikož záznam o akceleraci vozidla obsahuje celkem 4 údaje a to je akcelerace na osách x, y, z a čas, kdy byl tento záznam získán, nelze tento typ záznamu filtrovat. Pro potřeby filtrace vypočítáme celkové zrychlení vozidla pomocí vzorečku $a_{total} = \sqrt{a_x^2 + a_y^2 + a_z^2}$. Pro filtraci údajů o akceleraci vozidla tedy následně využijeme celkovou akceleraci a čas, kdy byl tento záznam získán.

Kapitola 6

Testování

Testování aplikace probíhalo v několika úrovních a to pomocí automatických testů, testováním v simulovaném prostředí a testováním v reálném prostředí.

6.1 Zařízení použitá při testování

Automobil

Škoda Octavia

Rok výroby: 2000
Motor: Benzín/LPG 2.0 litru

Škoda Octavia

Rok výroby: 1999
Motor: Diesel 1896 ccm

Audi Q3

Rok výroby: 2013
Motor: Diesel 2.0 litru

OBD jednotka

Bluetooth ELM 327 v1.5

Mobilní zařízení na platformě android

Nvidia Shield tablet LTE 32GB, Android 6.0

Google Nexus 7, Android 5.1

Sony Xperia E1 Dual, Android 4.4.2

LGE LG G3, Android 5.0

Lenovo TAB 2 A8-50F,Android 5.0

6.2 Automatické testování

Automatické testování umožňuje opakovaně spustit testy nad aplikací. Automatické testy jsou složeny z jednotkových testů, instrumentálních testů a testování uživatelského rozhraní.

6.2.1 Jednotkové testy

Jednotkové testy mají v Androidím prostředí výhodu, že nepotřebují simulátor ani reálné zařízení pro jejich spuštění. Z tohoto důvodu je jejich provedení velmi rychlé. Jejich velkou nevýhodou je však to, že pro tyto testy je vytvořená speciální verze android.jar, která implementuje veškeré metody API, avšak volání na tyto metody buď vrátí předdefinovanou odpověď nebo skončí chybou. Proto je třeba veškerá volání na toto API mockovat, což následně vede k tomu, že testy jsou pouze obrazem testovaného kódu. Z tohoto důvodu se jednotkové testování používá pouze na třídy s minimálním využití Android API.

Pro mockování Android API jsem využil framework Mockito[21] jelikož umožňuje jednoduché mockování tříd, neboli simulaci chování třídy.

6.2.2 Instrumentální testy

Instrumentální testy umožňují otestování tříd, které využívají Android API bez nutnosti jejich mockování. Tyto testy ke svému provedení potřebují simulátor či reálné zařízení, což znamená, že provedení testů je delší než u jednotkových testů. Tento typ testů byl v rámci aplikace hlavně využit pro otestování přístupu k databázi.

Bohužel, instrumentální testy neumožňují jednoduché mockování, připojení k internetu ani na mockování Bluetooth. Na tyto části bylo nutné využít jednotkových testů.

Aby testy neměnily data v produkční databázi, je před každým testem upraven název databáze, která se v rámci testu bude využívat a také je tato databáze smazána a znovu vytvořena, aby se testy nemohly navzájem ovlivňovat.

6.2.3 Testy uživatelského rozhraní

Testy uživatelského rozhraní jsou vytvořeny na základě Instrumentálních testů s použitím frameworku Robotium[26], který umožňuje jednoduché testování uživatelského rozhraní. V rámci naší aplikace byl tento typ testů hlavně využit pro otestování různých nastavení a přechodů z jedné obrazovky na druhou.

6.3 Testování v simulovaném prostředí

Hlavním cílem palubní jednotky je sbírat údaje o provozu automobilu od OBD jednotky, k otestování správné funkčnosti této části je tedy třeba mít vozidlo s OBD jednotkou. Časté testování aplikace s automobilem je tedy časově i finančně náročné, bylo tedy třeba vytvořit simulované prostředí pro otestování aplikace.

Pro simulaci OBD jednotky jsem využil program OBDSim[22], který umožňuje simulaci základních údajů o provozu automobilu, ale také informace o chybových stavech řídící jednotky. K jeho fungování je třeba pouze počítač s Bluetooth.

Testování v simulovaném prostředí bylo využito pro otestování funkčnosti jako celku včetně získávání údajů o provozu automobilu tak i jeho následná integrace na serverovou část projektu. Také byl využit pro otestování využití výkonu aplikace, jelikož OBDSim vrací údaje v minimálním časovém intervalu.

6.3.1 Testování využitého výkonu

Pro zjištění důvodu vysokého využití CPU na zařízení byl použit Android Device Monitor, který je součástí Android Studia. V rámci tohoto software byl použit Method Profiling, který sleduje dobu, po kterou byl kód v rámci dané metody vykonáván.

Testování výkonu bylo prováděno na Nvidia Shield, kde aplikace před úpravami využívala okolo 95% možného výkonu, což vzhledem k tomu, že tablet má procesor Tegra K1, bylo velmi nežádoucí. Důvodem tohoto vysokého vytížení CPU byla metoda "wait"ve vlákně, které se staralo o posílání dotazů na serverovou část projektu. Do fronty tohoto vlákna byly ukládány dotazy na server a po jejich vyřízení bylo vlákno opět na určitou dobu uspáno. Tento problém byl vyřešen odstraněním tohoto vlákna a jeho nahrazením třídou TimerTask, která je spouštěna v periodické době.

Následně výkon na tabletu klesl na asi 50%. Důvodem tohoto stále vysokého využívání CPU jsou budíky, které se překreslují při změně rychlosti či otáček motoru automobilu. Pokud dojde ke změně této hodnoty, je volán ValueAnimator, který animuje změnu z původní hodnoty na novou a v určitém časovém intervalu překresluje obrazovku. Při samotném překreslování obrazovky je volána vždy metoda drawBitmap, která překreslí daný budík a ta způsobuje velké zatížení CPU. Proto byl do aplikace přidán přepínač, který umožňuje vypnutí překreslování budíků aplikace a tím i snížit využití CPU. Po těchto úpravách se využití CPU na zařízení pohybovalo maximálně do 10%. Tyto problémy pocházely již z práce Tomáše Jungmana.

6.4 Testování v reálném prostředí

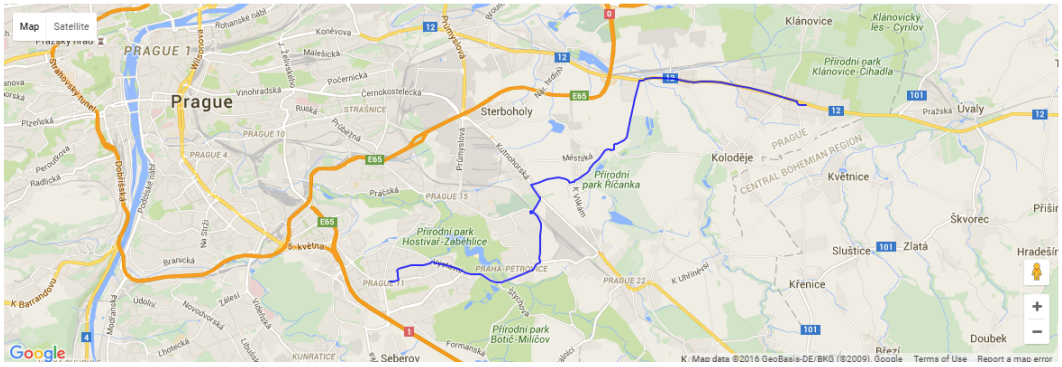
Testování v reálném prostředí probíhalo za využití reálných vozidel a OBD jednotky. Tato testování poukázala na chybu v OBD jednotce. Pokud je palubní jednotka připojena k OBD jednotce v době, kdy dojde ke zhasnutí a následnému nastartování motoru, může se stát, že OBD jednotka zamrzne a nelze se k ní již připojit. V takovém případě je nutné OBD jednotku vyjmout a opět vložit zpět, aby bylo možné se opět připojit. Jednotky použité při testování jsou levné kopie koupené z Číny, které je možné koupit za cenu okolo 7 dolarů. A toto je jedna z jejich známých chyb. K odstranění tohoto problému by bylo tedy třeba koupit dražší OBD jednotky.

Aplikace palubní jednotky byla pro lepší distribuci aplikace mezi účastníky testovacích jízd nasazena na Google Play ve fázi betaverze. K dostupnosti této aplikace v rámci Google Play stačí potvrdit odkaz, že se chcete účastnit beta testování aplikace. Následně si již můžeme aplikaci stáhnout. Velkou výhodou je také to, že při vydání nové verze je uživatel upozorněn, že je možné provést aktualizaci aplikaci.

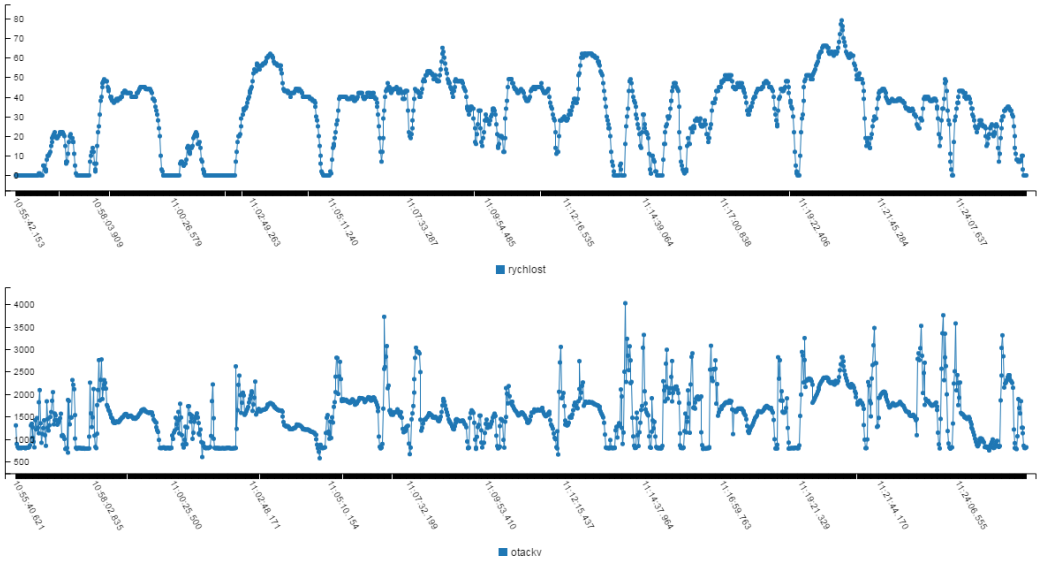
Serverová část byla nasazena na Heroku[14]. Server je sice možné využívat zdarma, ale má také řadu omezení. Velkým omezením pro projekt je to, že je do jedné tabulky možné uložit

maximálně sto tisíc záznamů, což komplikovalo testování aplikace. Jelikož po překročení tohoto limitu se databáze začala chovat nepředvídatelně a vracela špatná data. Z tohoto důvodu Michal Polata nasadil aplikaci na svůj lokální server.

Na obrázku 6.1 a 6.2 je možné vidět údaje zaznamenané během jedné testovací jízdy.



Obrázek 6.1: Mapa cesty testovací jízdy



Obrázek 6.2: Data testovací jízdy

Tabulka 6.1: Zařízení v Cloud Test Lab

Zařízení	API Level
HTC One (M8)	19
LG G3	19
Moto G (1st Gen)	19
Moto G (2nd Gen)	19
Moto X	19
Moto E	19
Nexus 4	19/22
Nexus 5	19/21/22
Nexus 6	21/22
Nexus 7 (2013)	19/21
Nexus 9	21
Galaxy S4 (3G)	19
Galaxy S4 mini	19
Galaxy S6	22
Galaxy Note 2	19
Galaxy Note 3 Duos	19

6.5 Cloud Test Lab

Cloud Test Lab[11] umožňuje otestování aplikace na více reálných zařízeních. Tato služba je poskytována společností Google, nyní je ve fázi beta a není třeba za tuto službu platit. Je třeba se zaregistrovat na Google Cloud Platform[13], kde je nutné vložit číslo kreditní karty.

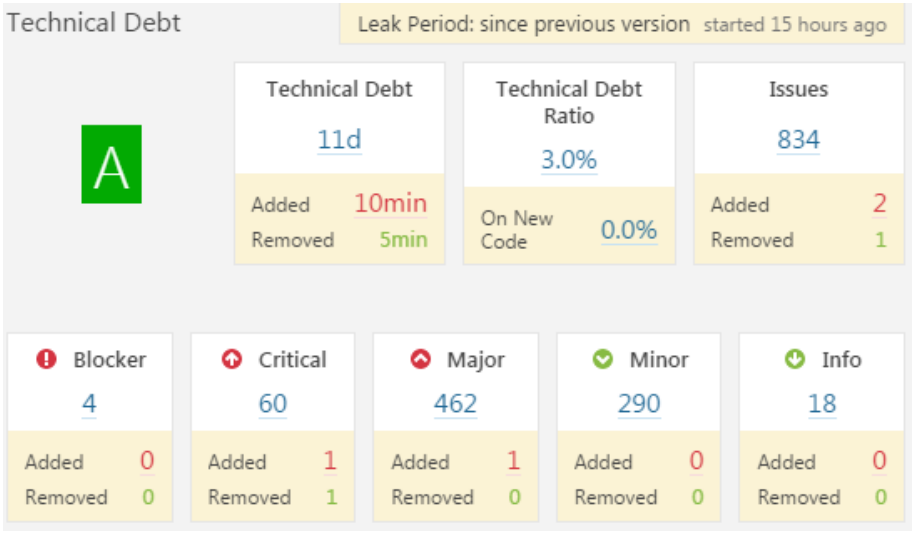
Cloud Test Lab umožňuje spuštění instrumentálních testů, ke kterým patří také naše testy uživatelského rozhraní. Během průběhu testování je pořizován záznam obrazovky zařízení, takže v případě chyby testu na uživatelského rozhraní, je možné vidět co se děje i na obrazovce. Tato služba nyní obsahuje omezený počet zařízení, který je uveden v tabulce 6.1.

Pomocí tohoto testování byla objevena chyba, kdy aplikace nahrávala Bitmapy, ale už je nerecyklovala, což po několikanásobném spuštění testů vedlo k nedostatku paměti na zařízení. Tento problém byl vyřešen recyklací daných bitmap při ukončení aktivity.

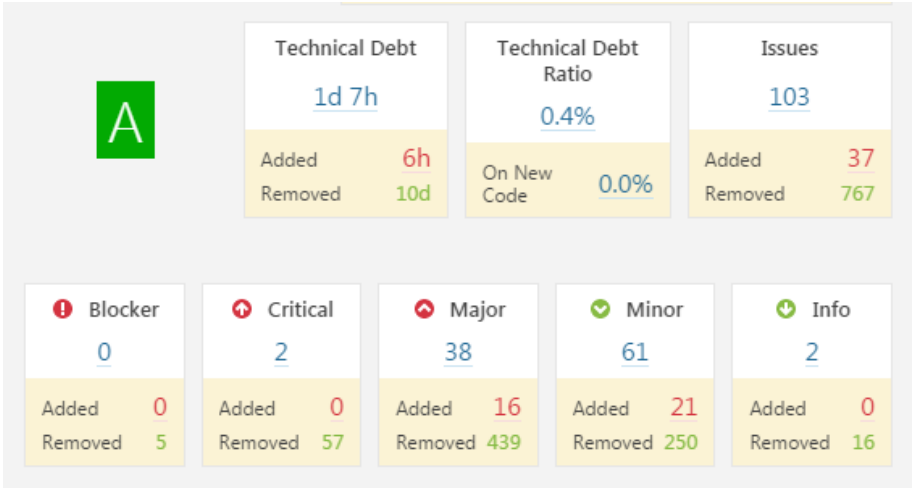
6.6 Sonar

Pro kontrolu kvality kódu a pokrytí aplikace testy v rámci palubní jednotky jsem použil SonarQube[28].

Na obrázku 6.3 je vidět kvalita kódu na začátku projektu po převzetí projektu. Celkový počet problému v aplikaci, je na projekt tohoto rozměru velký. Hlavním problémem je ale počet blokujících a kritických chyb v aplikaci. Obrázek 6.4 zobrazuje počet problému v rámci na konci vývoje. Byla odstraněna velká část část chyb v rámci aplikace.



Obrázek 6.3: Kvalita kódu na začátku projektu

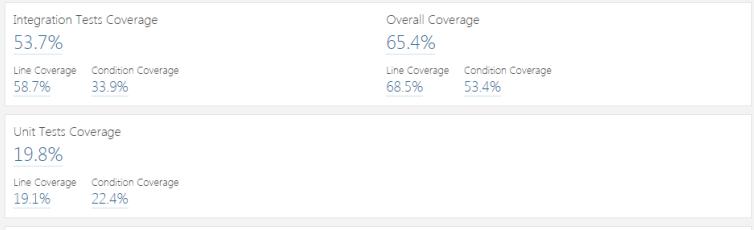


Obrázek 6.4: Kvalita kódu na konci projektu

Po převzetí aplikace neobsahoval projekt žádné testy což komplikovalo rozšiřování této aplikace. SonarQube obsahuje podporu pro pokrytí instrumentálními testy, ale tuto podporu již nemá pro jednotkové testy. Po spuštění jednotkových testů je nutné převést výsledek testů tak, aby bylo možné tyto výsledky poslat na SonarQube. Pro spuštění všech testů a převedení jednotkových testů do formy použitelné Sonarem je třeba do konzole zadat následující příkaz.

```
gradlew clean createDevelDebugCoverageReport jacocoTestReport
```

Na obrázku 6.5 je vidět výsledné pokrytí aplikace testy.



Obrázek 6.5: Pokrytí aplikace testy

Kapitola 7

Závěr

Cíle práce byly splněny. Byl vytvořen prototyp palubní jednotky, který umožňuje získávání údajů o provozu automobilu jak od OBD jednotky, tak také od Androidího zařízení. Umožňuje filtrování dat dle nastavitelných parametrů před jejich odesláním na serverovou část projektu. Na serverové části byla vytvořena potřebná rozhraní pro integraci s palubní jednotkou. Funkčnost prototypu palubní jednotky a následná integrace se serverovou částí byla otestovaná v reálném provozu na více typech automobilů a mobilních zařízeních. Aplikace palubní jednotky byla zveřejněna v beta verzi na Google Play. Serverová část byla nasazena na server Heroku.

Přínosem této práce pro projekt Metrocar je vytvoření prototypu palubní jednotky. Integrace palubní jednotky a serverové části projektu, která umožňuje získávání reálných dat o provozu vozidel. Dále filtrování získaných dat, což snižuje požadavky na velikost datového úložiště potřebného na serveru a také snižuje počet dat přenesených z palubní jednotky na server. Posledním přínosem je návrh na hodnocení jízdního stylu uživatelů na serverové části dle dat získaných od palubní jednotky.

Dalším pokračováním této práce by měla zaprvé být realizace funkčnosti, která umožňuje vyhodnocení jízdního stylu uživatele na serverové části projektu a prezentace tohoto výsledku zainteresovaným osobám. Zadruhé vytvoření uživatelsky přívětivého rozhraní palubní jednotky s nižšími nároky na výkonnost operačního systému Androidu.

Literatura

[1] BONSALL, P. – LIU, R. – YOUNG, W. Modelling safety-related driving behaviour—impact of parameter values, 2004.

[2] *Cena provozu vozidla* [online]. [cit. 10. 4. 2016]. Dostupné z: http://byznys.ihned.cz/podnikani/provoz-firmy-autopark/c1-60421690-chcete-usetrit-ve-firemni-vozove-flotile-zacnete-lepe-pocitat.

[3] CONSTANTINESCU, Z. – MARINOIU, C. – VLADOIU, C. Driving Style Analysis Using Data Mining Techniques, 2010.

[4] JOHNSON, D. A. – TRIVEDI, M. M. Driving Style Recognition Using a Smartphone as a Sensor Platform, 2011.

[5] JUNGMAN, T. *Diplomová práce* [online].

[6] KANTOR, S. – STáREK, T. Design of Algorithms for Payment Telematics Systems Evaluating Drivers Driving Style, 2014.

[7] KUBů, R. *bakalářská práce* [online].

[8] VAN LYY, M. – MARTINY, S. – TRIVEDI, M. M. Driver Classification and Driving Style Recognition using Inertial Sensors, 2013.

[9] VAN MIERLO, J. et al. Driving style and traffic measures—influence on vehicle emissions and fuel consumption, 2004.

[10] *Carsharing* [online]. [cit. 18. 5. 2016]. Dostupné z: http://ceskycarsharing.cz/.

[11] *Cloud Test Lab* [online]. [cit. 7. 5. 2016]. Dostupné z: https://developers.google.com/cloud-test-lab/.

[12] *Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature* [online]. [cit. 23. 4. 2016]. Dostupné z: https://www.researchgate.net/publication/242506399_Algorithms_for_the_Reduction_of_the_Number_of_Points_Required_to_Represent_a_Digitized_Line_or_Its_Caricature.

[13] *Google Cloud Platform* [online]. [cit. 7. 5. 2016]. Dostupné z: https://cloud.google.com/.

[14] *Heroku* [online]. [cit. 18. 5. 2016]. Dostupné z: https://www.heroku.com/.

[15] *Vysvětlení MAF.* [online]. [cit. 27. 4. 2016]. Dostupné z: https://audiklub.cz/techwiki/maf.

[16] *Projekt Metrocar zimní semestr 2008/2009* [online]. [cit. 14. 4. 2013]. Dostupné z: https://www.assembla.com/wiki/show/metrocar.

[17] *Závěrečná zpráva z projekt Metrocar letní semestr 2009/2010 až zimní semestr 2010/2011* [online]. [cit. 14. 4. 2013]. Dostupné z: https://www.assembla.com/spaces/metrocar/documents/cEdGsqzq4r4kcFeJe5cbLr/download/cEdGsqzq4r4kcFeJe5cbLr.

[18] *Závěrečná zpráva z projekt Metrocar zimní semestr 2009/2010* [online]. [cit. 14. 4. 2013]. Dostupné z: https://www.assembla.com/spaces/metrocar/documents/aT1N08g_ar35nSeJe5avMc/download/aT1N08g_ar35nSeJe5avMc.

[19] *Projekt Metrocar zimní semestr 2011/2012* [online]. [cit. 14. 4. 2013]. Dostupné z: https://www.assembla.com/spaces/wagnejan_metrocar/wiki/SI2_-_Home.

[20] *Projekt Metrocar zimní semestr 2012/2013* [online]. [cit. 14. 4. 2013]. Dostupné z: https://www.assembla.com/spaces/wagnejan_metrocar/wiki/SI2_-_2012_-_Home.

[21] *Mockito* [online]. [cit. 7. 5. 2016]. Dostupné z: http://mockito.org/.

[22] *OBDSim* [online]. [cit. 7. 5. 2016]. Dostupné z: http://icculus.org/obdgpslogger/obdsim.html.

[23] *DTC P0171* [online]. [cit. 27. 4. 2016]. Dostupné z: http://www.obd-codes.com/p0171.

[24] *DTC P0172* [online]. [cit. 27. 4. 2016]. Dostupné z: http://www.obd-codes.com/p0172.

[25] *An iterative procedure for the polygonal approximation of plane curves* [online]. [cit. 23. 4. 2016]. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0146664X72800170.

[26] *Robotium* [online]. [cit. 7. 5. 2016]. Dostupné z: http://www.robotium.org.

[27] *Intellij IDEA SBT* [online]. [cit. 27. 4. 2016]. Dostupné z: https://www.jetbrains.com/help/idea/2016.1/getting-started-with-sbt.html.

[28] *SonarQube* [online]. [cit. 14. 5. 2016]. Dostupné z: http://www.sonarqube.org/.

Kapitola 8

Seznam použitých zkratek

GPS Global Positioning System

ECU Engine Control Unit

EOBD European On Board Diagnostic

RDP Ramer–Douglas–Peucker algorithm

API Application Programming Interface

CPU Central Processing Unit

RPM Revolutions Per Minute

MAF Mass Air Flow

UX User experience

Kapitola 9

Instalační příručka

9.1 Hardwarové požadavky

Aplikace vyžaduje minimálně zařízení se systémem Android 4.1 (JELLY BEAN) či vyšším, což odpovídá API levelu 16 a vyššímu. Zařízení také musí obsahovat Bluetooth.

9.2 Instalační příručka

K instalaci aplikace stačí do zařízení na platformě Android stáhnout soubor Meteocar.apk, uložený na přiloženém CD. Následně spustit instalaci tohoto souboru. Při instalaci aplikace bude uživatel upozorněn, že aplikace získává přístup k několika oprávněním, například přesná poloha, přibližná poloha, úplný přístup k síti atd.

9.3 Úvodní nastavení aplikace

Pro úvodní nastavení aplikace je třeba se nejdříve přihlásit na stránky "http://meteocar.herokuapp.com/"pod uživatelským jménem "Johny"a heslem "root". Kde si následně vytvořte uživatele a pro něho automobil s nastavenou vaší palubní jednotkou. Následně se do palubní jednotky přihlaste se jménem a heslem "root", se zatrženou možností že chcete přejít do nastavení. V nastavení jděte do "Nastavení palubní jednotky"a opět to samé. Zde můžete zadat jméno a heslo vaší vytvořené jednotky na serveru Meteocar. Poté zkontrolujte že v nastavení sítě máte adresu serveru "http://meteocar.herokuapp.com/". Po odchodu z nastavení aplikace se již můžete přihlásit pod jménem a heslem vytvořeného uživatele.

Kapitola 10

Obsah přiloženého CD

- `readme.txt` ..... Stručný popis obsahu CD
- `src`
  - `impl` ..... Zdrojové kódy implementace
    - `Metrocar` ..... Intellij IDEA projekt serverové části
    - `MetrocarUnit` ..... Android Studio project palubní jednotky
  - `thesis` ..... Zdrojová forma práce ve formátu LaTeX
- `exe` ..... Adresář se spustitelnou formou implementace
  - `MeteocarUnit.apk` ..... Aplikace palubní jednotky
- `text` ..... Text práce
  - `thesis.pdf` ..... Text práce ve formátu PDF
- `ObdSim` ..... ObdSim simulátor
- `javadoc` ..... Javadoc MetrocarUnit
- `RDP.xlsx` ..... Tabulky pro RDP