VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

APLIKACE PRO EXTRAKCI A ANALÝZU JÍZDNÍCH DAT Z OBD-II NA IOS

APPLICATION FOR EXTRACTION AND ANALYSIS OF DATA FROM OBD-II ON IOS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. PATRIK PIHRT**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. PETR BOBÁK**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Pihrt Patrik, Bc.**
Program: Informační technologie
Obor: Inteligentní systémy
Název: **Aplikace pro extrakci a analýzu jízdních dat z OBD-II na iOS**
iOS Application for Driving Data Extraction and Analysis via OBD-II
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s protokolem OBD-II, mikrokontrolérem ELM327, architekturou iOS, jazykem Swift a frameworky SwiftUI a UIKit.
2. Prozkoumejte knihovny pro bezdrátovou komunikaci iOS zařízení s ELM327 a proveďte jejich analýzu.
3. Nastudujte designové principy návrhu uživatelského rozhraní pro platformu iOS, analyzujte existující aplikace a iterativním způsobem navrhněte uživatelské rozhraní aplikace.
4. Určete jízdní parametry vhodné k bližšímu zkoumání (např. spotřeba paliva - predikce dojezdu, míra akcelerace - odhad jízdního stylu) a vyberte vhodné techniky pro jejich analýzu.
5. Navrženou aplikaci implementujte.
6. Otestujte funkcionalitu a použitelnost výsledné aplikace.
7. Zhodnoťte dosažené výsledky, vytvořte krátké prezentační video, informační plakát a navrhněte možné pokračovaní.

Literatura:

- Human Interface Guidelines: https://developer.apple.com/design/human-interface-guidelines/
- SwiftUI: https://developer.apple.com/xcode/swiftui/
- dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 4 a částečně 5.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Bobák Petr, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 30. října 2020

Abstrakt

Cílem této práce je vytvoření mobilní aplikace pro iOS zařízení, která předává uživateli cenné informace o jeho jízdě automobilem a poskytuje hlavní diagnostické funkce. Pro získání informací je využito mikrokontroleru ELM327, který je obsažen v OBD-II skeneru, nabývající schopnosti komunikovat s automobilem prostřednictvím OBD-II standardu. Získaná data jsou analyzována za pomoci tabulky mezí. Výsledná aplikace informuje řidiče o jeho chybách při jízdě a dává mu rady jak se jim vyhnout. Jízdy je možné si ukládat a exportovat v podobě knihy jízd. Navíc poskytuje hlavní diagnostické funkce, které je možné najít v každé diagnostické aplikaci.

Abstract

This project aims to create a mobile application for iOS devices that provide valuable information about user's car rides. Needed data are obtained by ELM327 located in OBD-II scanner which can communicate with the car. Obtained data are analyzed using table of limits. The application informs riders about their driving mistakes and giving them tips how to fix them. Trips can be saved and exported in the form of logbook. In addition application includes diagnostic functions that can be found in every other diagnostic application.

Klíčová slova

Aplikace pro mobilní zařízení, Palubní diagnostika, OBD, OBD-II, ELM327, Bluetooth, iOS, Swift, SwiftUI, Analýza jízdy, Kniha jízd, Spotřeba paliva

Keywords

Application for mobile devices, On-Board Diagnostics, OBD, OBD-II, ELM327, Bluetooth, iOS, Swift, SwiftUI, Car trip analysis, Log book, Fuel consumption

Citace

PIHRT, Patrik. *Aplikace pro extrakci a analýzu jízdních dat z OBD-II na iOS*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Petr Bobák

Aplikace pro extrakci a analýzu jízdních dat z OBD-II na iOS

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana Ing. Petra Bobáka. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Patrik Pihrt
18. května 2021

Poděkování

Děkuji vedoucímu práce Ing. Petru Bobákovi za průběžné konzultace a ochotu, když se vyskytly nejasnosti při řešení práce. Dále bych rád poděkoval své rodině a hlavně své přítelkyni, že mě v práci podporovali.

Obsah

Kapitola 1

Úvod

Velká část dnešních domácností vlastní automobil. Jednou z povinností vlastníka automobilu je pravidelná technická prohlídka vozidla. Někteří řidiči také pravidelně navštěvují servisy, ve kterých se provádí například přezutí pneumatik či výměna oleje. Při obou službách mechanici využívají diagnostických zařízení, které jim zprostředkovávají podrobné informace o stavu vozidla. Mezi získatelné hodnoty patří například počet otáček motoru za minutu, počet najetých kilometrů od posledního mazání chybových hlášek nebo procentuální zatížení motoru. Díky pokroku mobilních telefonů jde ruku v ruce i vývoj aplikací, které umožňují přístup k těmto diagnostikám jak automechanikům, tak i běžným řidičům.

Tato diplomová práce se zabývá tvorbou mobilní aplikace pro systém iOS s využitím mikrokontroleru ELM327 a OBD-II. Pojednává o obecném úvodu do elektrických systémů v automobile, o historii a vzniku OBD-II. Dále se důkladněji věnuje OBD-II a jeho komunikačním protokolům, identifikátorům parametrů a chybovým kódům, které je možné prostřednictvím OBD-II získat. V další části práce jsou nastíněny možná vylepšení OBD-II a popis mikrokontroleru ELM327, který dokáže skrze např. Bluetooth komunikovat s mobilním zařízením. Následně se práce zmiňuje o způsobech programování aplikací pro zařízení od firmy Apple a knihovnách pro komunikaci s vozidlem. Rozebrány jsou i existující řešení, která jsou volně dostupná na trhu. Práce dále navazuje detailním rozebráním návrhu a implementace aplikace. Je zakončena testováním uživatelského rozhraní a využitých způsobů detekce chyb při jízdě.

Na začátku byla myšlenka vytvořit aplikaci, která bude sbírat data z vozidla a určovat, do jaké míry jel uživatel ekologicky správně. Po zjištění, že není úplně možné myšlenku zrealizovat (hlavně z důvodu neúplné definice toho, co je v daný okamžik správná jízda), byl návrh přepracován. Výsledná podoba aplikace si dává za úkol sbírat data z vozidla po dobu jedné jízdy. Sesbíraná data následně vyhodnotí a uživateli předá informace o úsecích, kde došlo k překročení limitů, které znamenají nesprávný způsob jízdy.

Motivací pro mě bylo získání nových znalostí ze světa motorových vozidel, o kterých jsem před začátkem tvorby práce mnoho nevěděl. Z nabytých vědomostí jsem si kladl za cíl vytvořit mobilní aplikaci pro iOS, která bude využitelná pro běžného uživatele, který by rád věděl více o jeho jízdě, a zároveň bude sloužit jako diagnostické zařízení v případě výskytu problému.

Hlavním rozdílem, kterým se aplikace liší od již existujících, je využití analýzy jízdy řidiče pomocí jízdních dat. Další odlišující funkcí bude poskytování rad řidiči po jízdě a upozornit na nesprávný způsob řízení na určitých úsecích trasy. Řidiči budou poskytnuty všechny podstatné funkce již existujících diagnostických aplikací. Cílem bylo vytvořit apli-

kaci, která bude přehledným způsobem předávat informace, a stane se tak jasnou volbou pro běžného uživatele.

Zadání mě oslovilo, protože jsem se již v bakalářské práci zabýval vývojem aplikace pro zařízení se systémem iOS. Lákalo mě reálné využití aplikace a možnost vyzkoušet si něco nového, s čím jsem do té doby neměl moc zkušeností, a to s daty automobilu.

Kapitola 2 přibližuje čtenáři senzory a elektronickou řídící jednotku. Kapitola 3 popisuje palubní diagnostiku od jejího počátku, až po možnou budoucnost. Důkladněji se zaobírá standardem OBD-II, a to komunikačními protokoly, identifikátory parametrů a chybovými kódy. Konec kapitoly se věnuje mikrokontroleru ELM327 a diagnostickým zařízením. Kapitola 4 seznamuje s historií vývoje aplikací na mobilní zařízení od společnosti Apple. Zaměřuje se na SwiftUI, popisuje smysl Human Interface Guidelines a informuje o knihovnách pro komunikaci s vozidly. Kapitola 5 pojednává o již existujících řešeních, jejich kladech a záporech. Kapitola 6 se zabývá návrhem aplikace. Je v ní popsán způsob získávání, zpracování a ukládání dat, detailní popis sbíraných dat, způsob detekce chyb, realizace diagnostiky, rozbor uživatelského rozhraní a vztahový diagram. Kapitola 7 informuje o naprogramovaném řešení a způsobech využití volně dostupných knihoven a modulů v různých částech aplikace. Kapitola 8 obsahuje zhodnocení práce v podobě testů uživatelského rozhraní a způsobů detekce chyb při jízdě.

Kapitola 2

Elektrický systém v automobilech

Motorová vozidla byla z počátku plně mechanická a nesledovaly se žádné hodnoty pomocí nichž by se vozidlo mohlo rozhodovat, jak se zachová. Největší změna v tomto odvětví nastala roku 1970, kdy byla představena elektronická řídící jednotka (ECU), která sleduje hodnoty senzorů v automobilech a na základě dat z nich ovlivňuje fungování určité části automobilu. ECU jsou v dnešních vozidlech samozřejmostí a to zejména z důvodu zpřísnění emisních podmínek, které auta bez mikrokontrolerů nejsou schopna splnit. Jedno vozidlo obsahuje až desítky řídících jednotek, které se starají o chod vozidla od motoru po ohřev sedadel [15].

2.1 Senzory

Senzory jsou nezbytnou součástí novodobých vozidel. Zprostředkovávají informace řídícím jednotkám, které následně ovlivňují aktuátory. Data ze senzorů jsou přístupná pomocí standardu OBD, popsaném v kapitole Palubní diagnostika. Tématem práce nejsou senzory jako takové, ale získávání jejich dat, a proto se nebude zacházet do hlubších podrobností, ale je potřeba se o nich zmínit. Pro přiblížení, jaké senzory můžeme v novějších automobilech najít a jak pracují, jsem vybral tři.

Senzor rychlosti motoru

Jedná se o nejdůležitější senzor v systému, který dohlíží na motor. Krom rychlosti motoru se dá využít na detekci pozice klikové, nebo váčkové hřídele. Informace o hřídeli se následně využívají při rozhodování, kdy se spustí doplňování paliva. Senzorů tohoto typu je více druhů, mezi ně patří Variable Reluctance (VR) a HALL effect. První zmíněný, který je možné vidět na obrázku 2.1, využívá magnetu, kolem kterého je omotána cívka. Při přiblížení/oddálení železného předmětu k/od senzoru se mění magnetické pole a cívka generuje napětí. Senzor je vhodný pro detekci pozice klikové hřídele, protože generované napětí cívkou tvoří sinusoidu podle níž lze určit, kde se daná hřídel nachází. HALL effect narozdíl od VR neprodukuje žádnou sinusoidu a jeho napětí nelze měřit. Pro zisk střídavého signálu je zapotřebí pull-up rezistoru. Senzor se skládá z magnetu a kousku elektroniky, která odpovídá na blízkost magnetu. V přítomnosti železa se senzor přepíná na zemniče. V opačném případě je signál přerušen. HALL senzor se spíše využívá ve spojení s váčkovou hřídelí. Ukázka, jak vypadá HALL senzor je na obrázku 2.2 [8].

Obrázek 2.1: Variable Reluctance senzor pro měření pozice klikové hřídele [8]

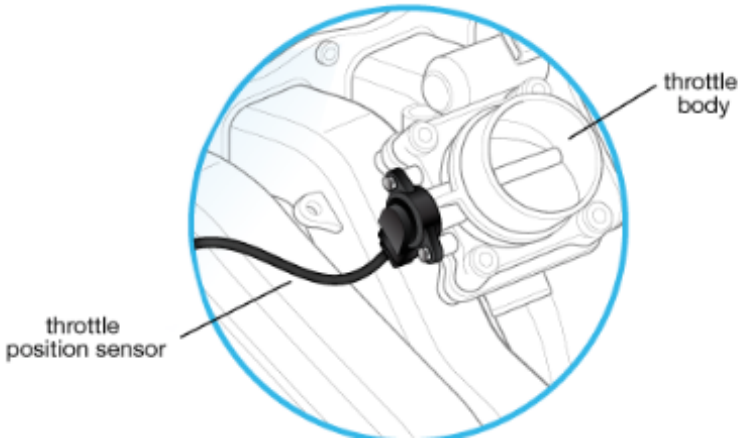
Obrázek 2.2: HALL effect senzor pro měření pozice váčkové hřídele [8]

Kyslíkový (O2) senzor

Také známý jako Lambda senzor. Má za účel monitorovat množství kyslíku ve výfukovém plynu. Pozná, zda motor používá směs bohatou, či chudou na palivo za pomocí oxidace části kyslíku procházejícího výfukovým systémem. Tuto informaci následně posílá řídící jednotce starající se o motor. U modernějších automobilů jsou senzory dva, a nebo čtyři podle toho, kolik má dané vozidlo výfukových trubek. První senzor se nachází za a druhý před katalyzátorem ve výfukové trubce. Senzor byl vytvořen za účelem snížení vytváření emisí a pro pomoc řídící jednotce starající se o motor, aby byla schopna vytvářet vyváženější kombinaci paliva a vzduchu [20].

Senzor pozice plynového pedálu

Senzor pozice plynového pedálu sleduje, jak moc je otevřený plynový ventil. Otevírá se na základě sešlápnutí plynového pedálu. Informace o plynovém ventilu postupuje do jednotky, která spravuje motor a hraje roli v tom, do jaké míry se ve výsledku vpustí vzduchu do sacího potrubí motoru. Senzor je možné najít připojený ke škrtící klapce (obrázek 2.3). V případě poruchy senzoru je buďto řidič informován pomocí varovného světla, nebo se auto postupně zastaví, protože řídící jednotka není schopna určit, kolik paliva je potřeba pro vytvoření správné směsi [4].



Obrázek 2.3: Plynový senzor a jeho umístění na škrtící klapce [4]

2.2 Elektronická řídící jednotka

Elektronická řídící jednotka (Electronic Control Unit – ECU) je zařízení odpovědné za čtení a zpracovávání signálů ze senzorů. Senzory je možné najít na různých místech automobilu. Získaná data jsou využita pro ovládání elektronicky řízených funkcí jako například elektronické vstřikování paliva. U vstřikování paliva je zapotřebí získání teploty, rychlost a pozice plynového pedálu. ECU následně porovná získaná data s daty v tabulce a zjistí, jak se má v dané situaci zachovat. Tabulka proměňuje vstupní senzorová data na výstupní. Jedním z typů tabulky je 3D tabulka, podle které se určuje poměr palivové směsi na základě otáček motoru za minutu a pozice plynového ventilu [16]. Vstřikovací palivová tryska se následně otevře podle pokynů řídící jednotky. Krom čtení z tabulky jsou novější ECU schopny provádět komplexnější výpočty pro ještě lepší kvalitu výsledné reakce na základě vstupů ze senzorů. Z důvodu velkého množství funkcí, které by ECU muselo naráz zvládat, je systém rozdělen na více řídících jednotek. Příklady ECU jsou sepsány níže [14].

Řídící modul motoru

Řídící modul motoru (Engine Control Modul – ECM) se zasluhuje za posuzování zatížení motoru a dodávky paliva do pístů, aby se dosáhlo optimální výkonosti, a co nejvíce se snížily emise. V minulosti nebyl kladen takový důraz na to, aby každé vozidlo obsahovalo ECM, protože se v minulosti tolik neřešily škodlivé látky, které vozidlo vylučuje, jako tomu je dnes. Po zavedení norem, které se zaměřují právě na problematiku emisí, je modul takřka nutností, a proto nechybí v žádném novějším vozidle [5].

Telematický řídící modul

Telematický řídící modul (Telematic Control Modul – TCM) poskytuje bezdrátovou komunikaci vozidlu a umožňuje připojení ke cloud službám. Modul sbírá telemetrická data jako pozici, rychlost a kvalitu připojení skrze komunikaci s podsystémy, které se zabývají daty a řízením vozidla. Může obsahovat i GPS přijímač pro navigační služby. Jednou z výhod je možnost sledovat automobil, když ho majiteli někdo odcizí. Modul napomáhá analyzovat nehody a chování řidiče při jízdě. Je vhodný například pro firmy, které disponují automobily, nad kterými chtějí mít dohled [18].

Kapitola 3

Palubní diagnostika

Palubní diagnostika, neboli On-Board Diagnostic (zkráceně OBD), je výraz využívaný v automobilovém průmyslu pro schopnost vozidla vlastní diagnostiky a nahlášení chyb. Palubní diagnostika umožňuje mechanikovi nebo vlastníkovi automobilu přístup k informacím o závadách a aktuálním stavu senzorů. Senzory mají za úkol sledovat komponenty, jejichž chybným fungováním by mohlo dojít k nadměrnému vylučování emisí z automobilu. Množství informací, které je uživatel schopen získat, razantně narostl od 80. let minulého století, kdy se objevily první implementace OBD. Nárůst byl způsoben stále se zvyšujícím počtem ECU ve vozidlech a snahou automatizovat různé funkce automobilu. OBD poskytuje mimo jiné i aktuální informace o stavu vozidla (stav palivové nádrže, stav světel...).

3.1 Historie

Roku 1970 vydal Kongres Spojených států amerických zákon o čistém ovzduší s cílem snížit znečištění vzdušného prostoru. Výrobci automobilů začali u svých nových modelů využívat senzorů a řídících jednotek, aby splnili kritéria, která jim byla stanovena novým zákonem. Už tehdy byly na světě konektory pro připojení diagnostických zařízení (DLC). Stejně tak byly v 70. letech vynalezen katalyzátor, elektronicky řízené vstřikování paliva a vznětový systém [21].

V roce 1980 firma General Motors přišla s Assembly Line Diagnostic Link (ALDL). Jedinou funkcí ALDL bylo vyblikávání pomocí diody. Dioda vyblikávala kód, který bylo poté možné najít v tabulce závad a podle toho určit, o jaký se jedná problém. Bylo potřeba měřit počet bliknutí a také délku k tomu, aby se dopátralo k správné chybové hlášce. Vyblikávání se provádělo při zapnutém zapalování a vypnutém motoru [21, 19].

California Air Resources Board (CARB) vydala roku 1983 předpisy o povinném zabudování palubní diagnostiky pro měření emisí do aut vyrobených po roce 1988. Tato povinnost začala platit pro auta po celých Spojených státech amerických o rok později. Časem se tomuto systému začalo říkat OBD-I. Jednalo se o jednoduchou řídící jednotku, která monitorovala řadu senzorů a upravovala mapu hodnot na základě stavu motoru. Důraz byl kladen na efektivní využití systému na celý život vozidla. S příchodem OBD-I přišla revoluce v servisním průmyslu. Detekce chyb byla od té doby o dost jednodušší, ale nebylo jasné, za jakých podmínek ke chybě došlo. Každý výrobce využíval svoje vlastní DLC, umístění DLC a kódy pro diagnostiku, a proto nebylo možné vytvořit univerzální zařízení, které by dokázalo pracovat s různými typy automobilů [21].

Ještě před tím, než přišel na scénu OBD-II systém, vytvořili General Motors OBD 1.5 využívaný na pár vozech z roku 1994 a 1995. Obsahoval podmnožinu chybových kódů svého nástupce, které jeho předchůdce neobsahoval. V dokumentaci k těmto vozidlům není ale o OBD 1.5 ani zmínka. Jsou zde pouze sekce OBD1 a OBD-II [19]. Roku 1994 vyšla nová generace palubní diagnostiky OBD-II, která se používá do dnes [21].

V roce 2001 se ke Spojeným státům přidala i Evropa se svou podobou předpisu OBD-II s názvem EOBD, která platila pro benzinové automobily. O dva roky později začal předpis platit o pro automobily dieselové [19].

3.2 OBD-II

Příchodem nové generace OBD systému vyvstávají otázky, čím se liší od předchozí verze. OBD-II je oproti svému předchůdci zabudován přímo ve vozidle, čímž je schopen udržovat stav vozidla, kterého automobil nabýval při výskytu poruchy. Přichází se standardizací diagnostického konektoru a přesně daným popisem pinů. Jedním z nich je pin poskytující energii pro diagnostická zařízení, čímž se eliminuje potřeba připojovat diagnostiku zvlášť do elektřiny. OBD-II určuje, které parametry automobilu je možné sledovat, a jak pro ně kódovat data. Obsahuje také rozšířený seznam chybových kódů s pevně daným formátem zpráv. Díky této standardizaci je možné pomocí jednoho diagnostického zařízení diagnostikovat jakýkoliv automobil, který obsahuje OBD-II. I když prostřednictvím standardizace mohou být přenášeny pouze kódy a data související s emisemi, stal se OBD-II konektor jediným diagnostickým konektorem u mnoha výrobců automobilů [21, 19].

3.3 EOBD

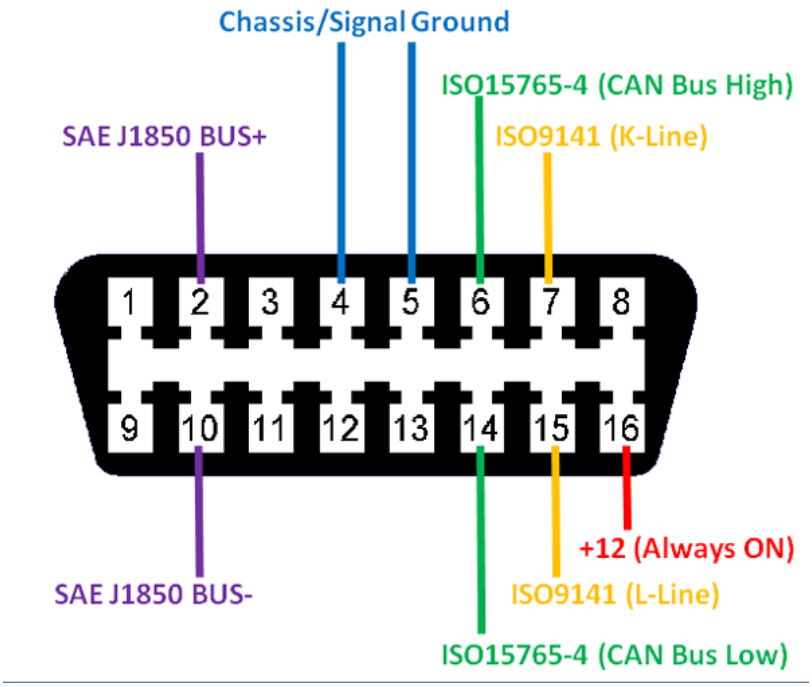
EOBD (European On Board Diagnostics) je ekvivalent k OBD-II. Jedná se o evropský standard, který byl poprve obsažen v benzínových automobilech roku 2001 a u dieselových automobilů roku 2004 (u nových modelů tomu bylo již o rok dříve). Využívá stejného diagnostického konektoru a komunikačních protokolů [19].

3.4 OBD-II konektor

Konektor je umístěn maximálně 60 centimetrů od volantu. Jedná se o 16ti pinový konektor typu samice, jehož vizuální podobu je možné vidět na obrázku 3.1. Řídí se standardem SAE J1962, který jasně definuje výstup každého z pinů. Jejich popis je v tabulce 3.1 [21].

3.5 Komunikační protokoly

OBD-II umožňuje výběr z pěti komunikačních protokolů. Pro zjištění, který z komunikačních protokolů je využíván v daném automobilu, je možné se podívat na to, který pin je v konektoru přítomen. Piny 4 a 16 jsou přítomny vždy, protože se jedná o uzemnění a pozitivní napětí baterie. Ve většině automobilů se využívá pouze jednoho protokolu, ale není to podmínkou.



Obrázek 3.1: OBD-II konektor podle normy SAE J1962 s popisem pinů [1]

3.5.1 SAE J1850 PWM

PWM je zkratkou pro Pulzně šířkovou modulaci, která slouží pro přenos analogového signálu pomocí dvouhodnotového signálu. Dvouhodnotovou veličinou může být například napětí a proud. Přenos informací je zprostředkován pomocí hodnot vypnuto/zapnuto, nebo také logická 1/logická 0. Logická 1 nabývá hodnoty +5V. Informace jako taková je pak zjištěna z poměru mezi těmito dvěma hodnotami. Přenosová rychlost signálu je 41.6 kB/sec. Protokol je standardem pro automobilního výrobce Ford Motor. Aktivní piny za přítomnosti SAE J1850 PWM v automobile jsou následující:

- Pin 2 – SEA J1850 PWM Bus+
- Pin 10 – SAE J1850 PWM Bus-

Délka zprávy je omezena na 12bitů a obsahuje i CRC (cyklickou kontrolu redundance) [21].

3.5.2 SAE J1850 VPW

VPW je zkratkou pro proměnnou pulzní šířku. Přenosová rychlost signálu je 10.4/41.6 kB/sec. Logická 1 nabývá hodnoty +7V a rozhodovací bod je nastaven na +3.5V. Protokol je standardem pro automobilního výrobce General Motors. Aktivní piny za přítomnosti SAE J1850 VPW v automobile jsou následující:

- Pin 2 – SEA J1850 VPW Bus+

Délka zprávy je stejně jako u předchozího protokolu 12bitů i s kontrolou redundance (CRC) [21].

Pin	Popis	Pin	Popis
1	Určeno pro potřeby výrobce	9	Určeno pro potřeby výrobce
2	SEA J1850 PWM+VPW Bus+	10	SAE J1850 PWM Bus- (bez VPW)
3	Určeno pro potřeby výrobce	11	Určeno pro potřeby výrobce
4	Uzemění podvozku	12	Určeno pro potřeby výrobce
5	Uzemění signálu	13	Určeno pro potřeby výrobce
6	CAN high (ISO 15765-4 a SAE J2284)	14	CAN low (ISO 15765-4 a SAE J2284)
7	ISO 9141-2 a ISO 14230-4 K-line	15	ISO 9141-2 a ISO 14230-4 L-line
8	Určeno pro potřeby výrobce	16	Napájení z baterie

Tabulka 3.1: Popis pinů podle normy SAE J1962 [1]

3.5.3 ISO 9141-2

Disponuje asynchronní přenosovou rychlostí 10.4 kb/s sériových dat. Protokol je podobný RS-232, až na úrovně signálu a obousměrnou linku bez ustálení spojení. ISO 9141-2 je využíván ve vozidlech pocházejících z Evropy a Asie. Další firmou, která využívá protokolu, je Chrysler. Aktivní piny za přítomnosti ISO 9141-2 v automobile jsou následující:

- Pin 7 – ISO 9141-2 K-line
- Pin 15 – SO 9141-2 L-line (volitelné)

Využívá UART signalizaci pro přenos dat. Délka zprávy je maximálně 260 bitů a největší délka pole pro data je 255 bitů [21].

3.5.4 ISO 14230 KWP2000

KWP2000 je zkrácená verze Keyword Protocol 2000. Stejně jako u předchozího protokolu je využito jedné oboustranné linky pro sériovou komunikaci nazývanou K-line. Přenosová rychlost se pohybuje mezi 1.2 až 10.4 kb/s. Stejně jako u ISO 9141-2 je maximální velikost datového pole 255 bitů. Standard ISO 14230 je rozdělen na části [21, 19]:

- ISO 14230-1 fyzická vrstva
- ISO 14230-2 datová vrstva
- ISO 14230-3 aplikační vrstva
- ISO 14230-4 požadavky pro systémy spjaté s emisemi

Aktivní piny za přítomnosti ISO 14230 KWP2000 v automobile jsou následující:

- Pin 7 – ISO 14230-4 K-line
- Pin 15 – ISO 14230-4 L-line (volitelné)

3.5.5 ISO 15765 CAN

CAN (Controller Area Network) je protokol vytvořený firmou Bosch pro automobilové a průmyslové řízení. Zakládá se na posílání zpráv pomocí sběrnice. Hlavním cílem bylo ušetřit měď využívající se pro výrobu kabelů. Přenos dat je realizován pomocí priorit. V případě,

že se více jak jedno zařízení snaží vysílat informace, je upřednostněno to, které má nejvyšší prioritu a ostatní jsou pozastaveny. Vysílaná data přijímají všechny zařízení, i to, které data vysílá. Každé zařízení si samo určí, zda data přijmou, či ignorovat. CAN sběrnice přichází se čtyřmi benefity, díky kterým se využívá skoro ve všech automobilech a ve velkém množství strojů.

- Jednoduchý a nízkonákladový – komunikace přes jeden CAN systém
- Centralizovaný – možnost komunikace se všemi ostatními jednotkami prostřednictvím jednoho vstupu (one point-of-entry). Umožňuje centralizované logování, diagnostiku a konfiguraci.
- Odolnost – odolný vůči elektrickému a elektromagnetickému rušení.
- Efektivní – Předávání informací je spravováno pomocí priorit, čímž se všechny kritické informace vždy dostanou na řadu jako první.

Komunikace na CAN sběrnici je řešena pomocí rámců (angl. frames). Na obrázku 3.2 je možné vidět strukturu rámce CAN 2.0 verze A. Ta se liší od B tím, že má 11 bitů oproti 29 bitům [21]. Popis zkratek je následující:

Obrázek 3.2: Rámec u CAN2.0A [21]

- SOF – State of frame – dominantní 0 v případě, že chce uzel komunikovat
- ID – Standard identifier – nižší hodnoty identifikátoru mají vyšší prioritu
- RTR – Remote transmission request – určuje, zda se jedná o odesílání, či přijímání dat
- Control – obsahuje dominantní 0, v případě CAN2.0 typu A. Navíc 4 bity, které informují o délce dat v bajtech (0-8 bajtů).
- Data – extrahovatelné a dekódovatelné CAN signály
- CRC – Cyclic redundancy check – kontrola datové integrity
- ACK – potvrzení doručení dat
- EOF – značky určující konec rámce

Přenosová rychlost CAN protokolu je 250kbit/s, nebo 500kbit/s. V případě, že se CAN nachází ve vozidle, jsou přístupné tyto piny:

- Pin 6 – CAN High
- Pin 14 – CAN Low

3.6 Identifikátory parametrů

Identifikátor parametru (PID) je kód, pomocí kterého se dotazujeme vozidla na data, která se využívají k diagnostice. Výrobci mají povinnost poskytovat určitou podmnožinu předdefinovaných PIDů. Standard SAE J1979 popisuje deset služeb, které byly v minulosti nazývány režimy. Jejich hexadecimální zápis a popis je možné vidět v tabulce 3.2. Vozidla nejsou povinna podporovat všechny služby. Výrobci mají možnost definovat svoje vlastní. Například Toyota využívá ve svých vozidlech službu s hexadecimálním zápisem 21. Každá služba obsahuje určitý počet PIDů, na které je možné se dotazovat [13].

Služba (Hexa)	Popis
01	Aktuální data
02	Zmražený rámec dat
03	Výpis uložených diagnostických chybových hlášek
04	Vyčištění diagnostických chybových hlášek a uložených hodnot
05	Výsledky testů O2 senzorů (mimo CAN)
06	Výsledky testů O2 senzorů (pouze CAN)
07	Výpis nevyřízených chybových hlášek
08	Řídící operace palubního systému
09	Vyžádání informací o vozidle
0A	Permanentní seznam chybových hlášek (i smazaných)

Tabulka 3.2: Služby OBD-II standardu SAE J1979 [13]

Služba 01 a 02

Služba 01 je takřka identická s 02 co se obsahu PIDů týče. Jedním z rozdílů je, že služba 01 poskytuje aktuální realtime data ze senzorů, kdežto služba 02 poskytuje zmražené informace ze senzorů v okamžiku výskytu chyby na vozidle. Další dva rozdíly jsou v PIDech, a to konkrétně PID 01, který je přítomen jen u služby 01 a PID 02, který je zase pro změnu přítomen jen u služby 02. PID 01 monitoruje stav od doby, kdy byly promazány všechny záznamy o chybách ve vozidle. PID 02 nám říká, o jakou konkrétní diagnostickou chybu se jedná, kvůli které se vytvořil rámec zmražených dat. Služby poskytují příliš mnoho PIDů na to, abych je zde mohl vyzobrazit všechny, a proto jsem se rozhodl vypsat jen pár, které jsou důležité pro mou práci [13]. Některé z PIDů nelze popsat pomocí výrazu, ale čtou se bit po

PID (Hexa)	Popis	Min	Max	Veličina	Vzorec
06	Krátkodobá úprava paliva	-100	99.2	%	$\frac{100}{128}A - 100$
07	Dlouhodobá úprava paliva	-100	99.2	%	$\frac{100}{128}A - 100$
0C	Rychlost motoru	0	16,383.75	rpm	$\frac{256A+B}{4}$
0D	Rychlost vozidla	0	255	km/h	A
11	Pozice plynového pedálu	0	100	%	$\frac{100}{255}A$
5E	Spotřeba paliva	0	3212.75	l/h	$\frac{256A+B}{20}$

Tabulka 3.3: Výběr z PIDů služeb 01 a 02 [13]

bitu, díky čemuž dosáhneme plného pochopení informace. Jedním z příkladů je vůbec první

PID služby 01 (00), pomocí něž se získá informace o přítomnosti dalších 32 PIDů. Každý jeden PID reprezentuje jeden bit v řetězci a odpověď vozidla je v podobě hexadecimálního výrazu (celkově 4 bajty). Příkladem takového výrazu může být řetězec BE1FA813, který říká, že je ve vozidle aktivní seznam PIDů: 01, 03, 04, 05, 06, 07, 0C, 0D, 0E, 0F, 10, 11, 13, 15, 1C, 1F a 20.

Služba 03

Není vyžadován žádný PID. Při dotazu na službu vrátí seznam diagnostických chybových kódů (DTC). Konkrétní popis kódů je popsán v kapitole Chybové kódy.

3.7 Chybové kódy

Chybové kódy, nebo-li Diagnostic trouble codes (DTCs), popisují poruchy systému v určité oblasti vozidla. Pro určení, ve kterém místě nastal problém, jsou kódy řešené v podobě řetězce s jedním písmenem na začátku řetězce a pěti čísly na konci. Seznam písmen, které může kód obsahovat je následující [2]:

- Bxxxx – Tělo (Body Systems) – světla, airbagy, klimatizaci, atd.
- Cxxxx – Podvozek (Chassis Systems) – ABS, systém řízení, atd.
- Pxxxx – Pohonné jednotky (Powertrain Systems) – Motor, převodovka, emisní systém, atd.
- Uxxxx – Síťová komunikace (Network Communication) – síťové kabely

Prvním číslem chybového kódu je číslo určující, zda se jedná o standartizovaný kód, či kód výrobce vozu. Výčet možných čísel je následující:

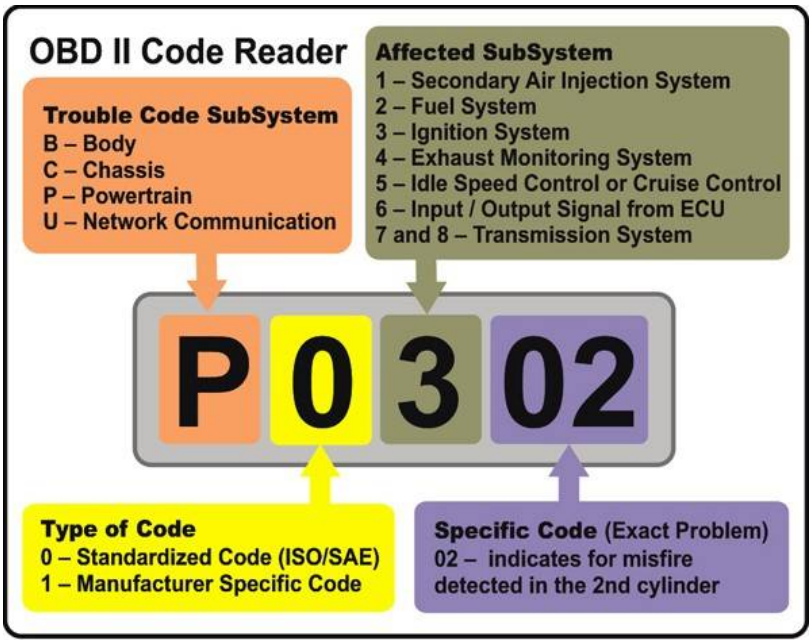
- x0xxx – Standardizovaný ISO/SAE kód
- x1xxx – Kód vytvořený výrobcem automobilu
- x2xxx – Může se jednat o standardizovaný, i o výrobcem vytvořený kód
- x3xxx – Může se jednat o standardizovaný, i o výrobcem vytvořený kód

Druhé číslo reprezentuje systém, ve kterém se daný problém vyskytl. Seznam čísel, kterých může druhé číslo nabývat, je následující:

- xx0xx – Celkový systém
- xx1xx – Systém vstřikování sekundárního vzduchu
- xx2xx – Palivový systém
- xx3xx – Vznětový systém
- xx4xx – Systém monitorující výfukové plyny
- xx5xx – Ovládání rychlosti vozidla
- xx6xx – Výstupní a vstupní signály z řídících jednotek

- xx7xx-xx9xx – Převodovka

Poslední dvoučíslí ukazuje přesně na danou komponentu, která vykazuje známky poruchy. Názornou ukázku, i s popisem, jak takový kód může vypadat, je možné vidět na obrázku 3.3.



Obrázek 3.3: Ukázka chybového kódu pro selhání druhého válce [2]

3.8 Budoucnost OBD (OBD-III)

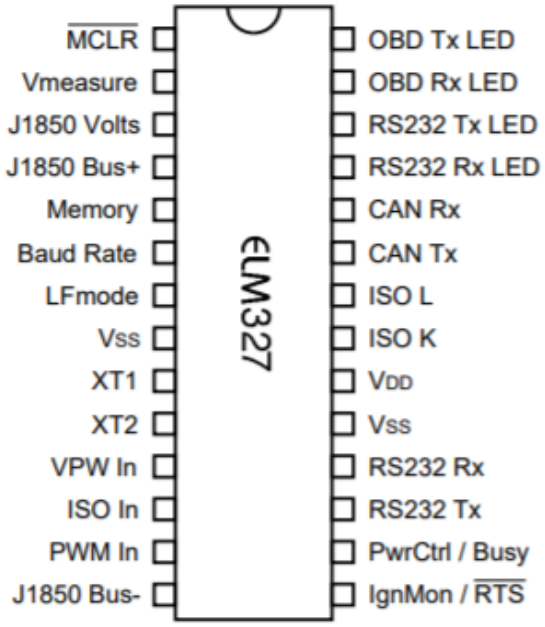
O zlepšení OBD-II se začalo mluvit již v roce 2001, kdy Charlie Gorman z institutu ETI (Equipment and Tool Institute) přišel s předpovědí, že by nové OBD mohlo využívat bezdrátové komunikace. Jednalo se o reakci na vytrácející se množství emisních testů prostřednictvím výfuku, které bylo více a více nahrazováno testováním pomocí OBD-II. V roce 2004 Agentura na ochranu životního prostředí podpořila Oregonský plán na implementaci bezdrátové kontroly emisí.

Další snahou o vylepšení je spojení všech komunikačních protokolů, pomocí nichž je aktuálně možné komunikovat prostřednictvím OBD s automobilem. Jedná se o Modulární komunikační protokol vozidla (MVGI), který se chová jako překladač pro všechny typy protokolů. Dává si za úkol, aby bylo možné komunikovat s jakýmkoliv z vozidel pomocí kteréhokoliv diagnostického zařízení [11].

3.9 ELM 327

V dnešní době je skoro každý automobil povinen poskytovat rozhraní pro připojení diagnostických zařízení. Přenos dat je realizován pomocí devíti standardizovaných protokolů, ale žádný z nich není na přímo využíván jak počítači, tak chytrými mobilními telefony. Tento problém se snaží řešit ELM327, který byl vytvořen, aby zastával roli prostředníka mezi

palubními diagnostickými porty (OBD ports) a rozhraním RS232. Výstupy pinů kontroleru jsou vyobrazeny na obrázku 3.4 [9].



Obrázek 3.4: Výpis pinů mikrokontroleru ELM327

ELM je schopen automaticky rozpoznat protokol, kterým automobil komunikuje. Navíc disponuje podporou pro vysokorychlostní komunikaci, spánkový režim o nízké spotřebě a protokolem pro komunikaci s nákladními vozidly a autobusy. Je plně přizpůsobivý potřebám uživatele, aby bylo možné splnit vše, co se od něj žádá. Pro účely konfigurace kontroleru je možné využít AT příkazů, které jsou dále popisovány v podkapitole AT příkazy [9].

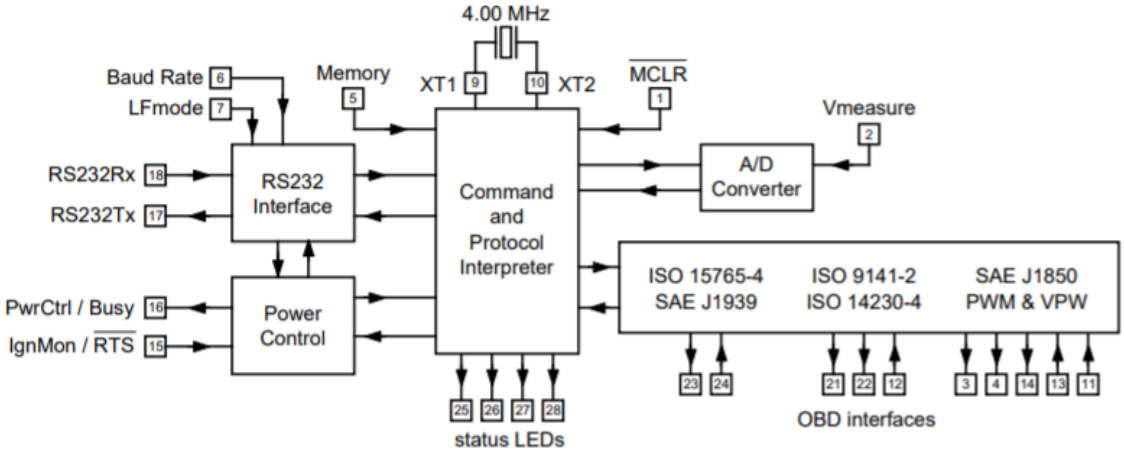
Mikrokontroler se využívá zejména pro komunikaci s aplikacemi pro čtení chybových kódů, skenovací nástroje pro kontrolu vozidla a materiály pomáhající řidiči zlepšit jeho jízdní schopnosti [9].

3.9.1 Architektura

Jak je již z obrázku 3.4 zřejmé, ELM327 nabývá 28 pinů. Obvod na obrázku 3.5 je rozdělen na čtyři části (moduly). Modul, na který jsou napojeny všechny ostatní moduly, se stará o interpretaci příkazů a protokolů. K modulu jsou připojeny čtyři led diody (pin 25-28), které informují o tom, co ELM327 právě dělá [9].

- Pin 25 – RS232 Rx LED – Přijímá data od RS232 protokolu
- Pin 26 – RS232 Tx LED – Posílá data skrze RS232 protokol
- Pin 27 – OBD Rx LED – Přijímá data od OBD
- Pin 28 – OBD Tx LED – Posílá data přes OBD

Napájen je prostřednictvím pinu 2, který vstupuje do A/D převodníku pro zisk diskrétního signálu. Na pinech 3, 4, 11-14 a 21-24 se nacházejí komunikační protokoly palubní diagnostiky (OBD). Další moduly, které jsou k hlavnímu řídicímu modulu připojeny, jsou modul řízení spotřeby a rozhraní RS232 [9].



Obrázek 3.5: Blokový diagram ELM327 mikrokontroleru

3.9.2 Komunikace

ELM327 počítá primárně s komunikací s počítačem prostřednictvím sériového RS232 spojení. Modernější počítače ale tuto možnost neposkytují, a proto existují i jiné, alternativní metody. Mezi jedny z nejběžnějších způsobů komunikace patří USB, WiFi a Bluetooth, které jsou připojeny k RS232 rozhraní [9].

Nejjednodušší způsob, jak posílat a přijímat data, je prostřednictvím programů na způsob terminálu. Aby bylo možné správně propojit terminální program s ELMem, je zapotřebí upravit nastavení programu. ELM podporuje dva typy příkazů, kterými jsou OBD příkazy, o kterých je možné se dočíst v podkapitole Identifikátory parametrů, a AT příkazy, o kterých je zmínka v podkapitole AT příkazy. Díky omezenému množství znaků u OBD příkazů (jen hexadecimální číslice) není možné, aby ELM nerozpoznal, o jaký typ příkazu se jedná, protože AT příkazy musí začínat řetězcem znaků „AT“ a „T“ nepatří mezi hexadecimální. V případě chyby vrátí ELM otazník („?“), ale žádné informace o jejím původu. ELM327 není citlivý na velikost písmen a na vstupu ignoruje bílé znaky [9].

3.9.3 AT příkazy

AT příkazy slouží pro změnu parametrů a chování mikrokontroleru ELM327. Pro komunikaci s vozidlem není nutné těchto příkazu využít, ale může se naskytnout situace, kdy menší modifikace může být užitečná. Stejný způsob interní konfigurace je využíván u počítačových modemů.

3.10 Diagnostická zařízení

Diagnostická zařízení se dělí na dva typy. Jedny fungují na principu čtení kódů a mají možnost kódy mazat. Z důvodu omezení funkcionality jsou tato zařízení levnější. Druhou z variant jsou Skenery, které jsou sice dražší, ale poskytují více informací. Poskytují rozsáhlý přístup k aktuálním datům, usnadňují pokročilejší řešení problémů a jsou schopny číst výrobcem specifikované chybové kódy.

Každý skener poskytuje základní funkcionalitu, kterou poskytují i čtečky kódů, a liší se zejména v tom, kolik informací je možné díky nim získat a zda mají přístup k chybovým kódům čekajících na vyřízení.

Velké procento vlastníků diagnostických zařízení ani neví jak s nimi správně zacházet. Pořizují si je proto, protože si myslí, že se dokáží o jakýkoliv problém postarat bez nutnosti zásahu vlastníka, či automechaniků. To ovšem není pravda. Zařízení slouží k odhalení nikoliv k vyřešení problémů [10].

3.10.1 Viecar

Viecar[1] je zařízení, které mi bylo poskytnuto pro moji práci. Jedná se o vysoce kvalitní skener poskytující Bluetooth připojení, které podporuje jak Android, tak iOS. Jedná se o univerzální zařízení vhodné pro kterékoliv vozidlo obsahující OBD-II. Cena výrobku se pohybuje kolem 900 až 1000 korun českých. Zařízení podporuje širokou škálu platforem, mezi které patří iOS, Android, Symbian a Windows. Poskytuje možnost komunikovat prostřednictvím kteréhokoliv z OBD-II protokolů, které se v novějších automobilech nacházejí. Na obrázku 3.6 je možné vidět design skeneru. Ke správné funkcionalitě je potřeba udržovat hodnoty v daném rozmezí:

- Napětí v rozmezí – 9V až 16V
- Proud – 45mA
- Teplota v rozmezí – -40°C až 85°C
- Vlhkost – nižší jak 60%

Obrázek 3.6: Viecar skener

[1] "https://www.czc.cz/viecar-obd-ii-autodiagnostika-bluetooth-4-0-cz-sw-zdarma/257631/produkt"

Kapitola 4

Platforma iOS

Platforma iOS, dříve známa pod jménem Phone OS, je operační systém vyvíjený společností Apple Inc. Jedná se o druhý nejpopulárnější operační systém na světě. První zařízení, které obsahovalo systém, bylo mobilní zařízení iPhone, který byl unikátní tím, že neměl žádnou hardwarovou klávesnici. Téhož roku byl se stejným operačním systémem uveden na trh iPod Touch a o tři roky později iPad. Hlavním rozdílem iOS od ostatních operačních systémů Applu, kterými jsou macOS a tvOS, je možnost práce se zařízením pomocí dotyků s obrazovkou. K dnešnímu dni je aktuální verze iOS 14.5. Pro vývoj aplikací na iOS je zapotřebí zařízení s operačním systémem macOS a na něm vývojářské prostředí s názvem Xcode.

Prvním programovacím jazykem pro programování aplikací pro iOS byl jazyk Objective-C. Jedná se o objektově orientovaný jazyk vycházející ze Smalltalku a jazyka C. Do roku 2014 se jednalo o jediný programovací jazyk pro programování iOS aplikací. Toho roku se na vývojářské konferenci WWDC představil úplně nový programovací jazyk, a to konkrétně jazyk Swift. Pro vývoj aplikací je možné využít obou jazyků, ale Swift je díky svému modernějšímu a jednoduššímu zápisu stále více oblíbenější.

4.1 Programovací jazyk Swift

Swift je objektově orientovaný jazyk. Lidé mu dali přezdívku „Objective-C bez C“, protože zakazuje přístup k ukazatelům a různým dalším nebezpečným prvkům jazyka C. Swift oproti Objective-C nevyžaduje hlavičkové soubory. Vše je psáno v jednom .swift souboru, který obsahuje typické informace vyskytující se v hlavičkovém souboru, vlastnosti a definice třídy. Další z výhod jazyka je možnost otestovat kód před vložením do programu za pomoci Read-Eval-Print cyklu (zkráceně REPL). Jedná se o možnost okamžitého vyhodnocení kódu a vytisknutí výsledku například v příkazovém řádku [3].

Swift byl navržen, aby byl výkonnější než jeho předchůdce. Je více jak dva a půlkrát rychlejší než Objective-C a více jak osm a půlkrát rychlejší než Python [3].

4.2 Vývoj uživatelského rozhraní – SwiftUI

SwiftUI je nový způsob vývoje uživatelského rozhraní pro Apple zařízení. Nahrazuje Storyboard, který slouží pro stavbu rozhraní s využitím knihovny UIKit. Byl představen na konferenci WWDC v červnu roku 2019. SwiftUI umožňuje vývojáři vytvářet uživatelské rozhraní za pomoci deklarativní Swift syntaxe. Programátor definuje, co by mělo rozhraní

dělat a iOS se postará o správnou vizualizaci uživateli. SwiftUI přichází s možností nahlížení na změny v reálném čase, bez nutnosti spouštět simulátor. SwiftUI je možné použít na vývoj aplikací pro všechny operační systémy od společnosti Apple Inc. Díky tomu je možné aplikaci vytvořenou pro iOS lehce přenést na jakýkoli jiný operační systém od Applu. V nastavení projektu stačí jen přenastavit operační systém, na který se má software nasadit[3].

Apple pasivně nutí vývojáře, aby přešli na SwiftUI tím, že umožnili tvorbu widgetů pouze prostřednictvím právě SwiftUI. Z toho se dá předpokládat, že Apple bude chtít kompletně přejít na SwiftUI a opustit Storyboard společně s UIKitem[3].

SwiftUI spoléhá silně na modifikátory proměnných[1], které dělají kód čitelnější, lépe udržovatelný a ulehčují psaní. Jedná se o způsob obalení proměnné, která poté získá více schopností. Obalení je realizováno přidáním modifikátoru před proměnnou. Modifikátory proměnných byly představeny společně se Swiftem 5.1 a jsou běžnou součástí každé SwiftUI aplikace. SwiftUI nabízí 17 modifikátorů a každý z nich se řadí do skupin podle toho, jestli:

- uchovávají dočasná/dlouhodobá data,
- čtou data z prostředí uživatele (například pro získání barevného schématu),
- odkazují se na části rozhraní,
- zacházejí s aplikací,
- (ne)vlastní informace.

Jedny z nejčastěji používaných modifikátorů jsou:

@State nám umožňuje měnit proměnné s malým objemem dat, které není normálně možné modifikovat, protože prvky uživatelského rozhraní ve SwiftUI jsou typu struktury a ta nedovoluje přímou modifikaci hodnot proměnných. Přidáním @State před proměnnou se hodnota uložená v proměnné přesune do sdíleného úložiště, které spravuje SwiftUI. Díky tomu je možné při každé změně hodnoty dané proměnné strukturu sestavit znovu bez ztráty informace.

@Binding říká, že se jedná o hodnotu, která je převzatá a je tím pádem sdílená. V případě změny proměnné, ke které je přiřazen @Binding, se změní hodnota i proměnné, ze které vychází.

@StateObject se využívá pro ukládání nových instancí objektů, které vycházejí z třídy splňující ObservableObject protokol. Pro zjištění, že se změnila hodnota objektu, je zapotřebí, aby proměnné v třídě měly před sebou @Published. Stejný výsledek lze získat s pomocí modifikátoru @ObservedObject, ale v tomto případě modifikátor nevlastní data objektu a může se stát, že v průběhu běhu aplikace se objekt uvolní z paměti.

@EnvironmentObject funguje na podobném principu jako @ObservedObject, ale s tím rozdílem, že se stačí objekt jednou vytvořit (s použitím @EnvironmentObject) a jakákoli jiná obrazovka ho pak může využívat. Není tím pádem potřeba přenášet odkaz na objekt přes všechny obrazovky, které jsou mezi obrazovkou, co objekt vytváří a tou, která ho dále využívá.

[1] https://www.swiftbysundell.com/articles/property-wrappers-in-swift/

4.3 Vývojové prostředí – Xcode

Xcode je vývojové prostředí dostupné jako aplikace na zařízeních s operačním systémem OS X. Obsahuje sadu nástrojů pro vytváření programů pro všechny operační systémy od Apple Inc., kterými jsou OS X, iOS, watchOS a tvOS. Podporuje jazyky C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, ResEdit (Rez), a Swift. Hlavní výhodou Xcode je, že obsahuje vše potřebné pro napsání, přeložení a otestování naprogramovaného kódu. Nabízí dvě možnosti vývojového cyklu aplikace a to UIKit, nebo SwiftUI. UIKit využívá Storyboard, který funguje na bázi „co vidíš, to dostaneš“ (zkráceně anglicky WYSIWYG). Jedná se o skupinu obrazovek, do kterých jsou vkládány objekty (UIViews) z knihovny UIKit. Hierarchie obrazovek ve Storyboardu tvoří výsledný vzhled aplikace.

SwiftUI, jak už bylo zmíněno výše, je novým způsobem návrhu uživatelského rozhraní, který se kompletně zbavuje Storyboardu. Navíc neobsahuje AppDelegate soubor, který u UIKitu nese přezdívku „srdce aplikace“. Nově vytvořený SwiftUI projekt z počátku obsahuje minimálně dva soubory, což je dvakrát méně než u UIKit cyklu. Prvním je soubor se strukturou odpovídající aplikačnímu protokolu a stejně jako AppDelegate soubor se jedná o vstupní místo do aplikace. Druhý obsahuje strukturu odpovídající View protokolu, která zastupuje první obrazovku programu. Xcode nabízí možnost testování SwiftUI programu pomocí živého náhledu, simulátoru, či na mobilním zařízení.

4.4 Human Interface Guidelines

Human Interface Guidelines, zkráceně HIG, je model dokumentace s doporučeními o tom, jak správně vyvíjet software na Apple zařízení. Dokumentace popisuje pravidla pro vizuální podobu aplikace společně se stylem a designem pro ikony. Určují, jakým způsobem uživatel zadává vstupy a jak interaguje. HIG míří na zlepšení uživatelské přívětivosti tím, že dělá uživatelské rozhraní vysoce efektivní. Všechno je dáno limitacemi, které dokumentace poskytuje, čímž tlačí vývojáře k tvorbě prostředí, na které jsou uživatelé zvyklí z jiných aplikací.

Některé ze zásad jsou na základě interakcí člověka s počítačem. Většina je však založena na rozhodnutí těch, kteří vyvíjí platformu a na designérech zaměřujících se na uživatelskou zkušenost. Cílem dokumentace je sjednotit vzhled pro různá zařízení jako například počítač, či mobilní zařízení. Uživatel je tak v novém prostředí schopen pracovat s aplikací bez jakéhokoliv problému, protože všechny komponenty zná z jiných zařízení a ví, jak s nimi pracovat.

HIG je pouze sada doporučení pro vznik kvalitní aplikace. Stává se, že vývojáři si zvolí pravidla úmyslně porušit a implementují danou komponentu jinak. Jedná se často o situace, kdy to považují za lepší variantu. Následně je otázkou, jak se k tomu postaví Apple, který má možnost takovou aplikaci neschválit.

4.5 Knihovny pro komunikaci s vozidlem

Knihovny pro komunikaci s vozidlem prostřednictvím OBD-II se rozdělují do dvou hlavních podskupin. První podskupinou jsou knihovny podporující připojení se zařízením skrze WiFi připojení. Druhou skupinou jsou ty, které umožňují připojení skrze Bluetooth. Při výběru správné knihovny proběhla analýza existujících možností, jejímž výsledkem bylo, že jediná použitelná knihovna pro komunikaci skrze Bluetooth na iOS je LTSupportAutomotive.

Přesto jsem se rozhodl zde zmínit i dvě další, které sice komunikují se zařízením pouze skrze WiFi, ale i tak stojí za zmínku.

4.5.1 OBD2Kit

OBD2Kit[2] je ze zde zmíněných knihoven nejstarší. Poslední úpravy byly provedeny před více jak deseti lety, což vede k tomu, že je naprogramována v jazyce Objective-C. Jedná se o knihovnu umožňující spojení s vozidlem prostřednictvím Wifi připojení. Napsal ji Michael Gile jako odnož aplikací FuzzyCar a goLINK. Jednou z nevýhod knihovny je absence stručného popisu práce s knihovnou. I když je u knihovny napsáno, že dokumentace přijde v blízké době, tak se za deset let stále nic nezměnilo. Nejlepší způsob, jak přijít na to, jak pracovat s danou knihovnou, je prohlédnout si demo aplikaci, která je společně s knihovnou na stejném odkaze. Osobně jsem neměl tu příležitost knihovnu otestovat, ale existuje aplikace OBD2Analyzer[3], která využívá právě této knihovny, z čehož usuzuji, že knihovna je plně funkční a použitelná.

4.5.2 OBD2-Swift

OBD2-Swift[4] knihovna byla naposledy upravena před čtyřmi lety. Jak už jméno napovídá, je naprogramována převážně v jazyce Swift. Stejně jako předchozí knihovna, umožňuje způsob připojení pouze skrze WiFi. Dokumentace ke knihovně je srozumitelná a nastiňuje, jak s danou knihovnou začít pracovat. Po ustálení spojení nabízí možnost využití Command struktury pro vybrání informace, kterou chceme získat. Nabízí i možnost Command.Custom, kde je možné zadat číslo módu a číslo PIDu ručně. Další z vlastností knihovny je možnost vytvořit opakující se volání zvoleného kódu. Poté jen stačí přihlásit observera, který sleduje daný PID konkrétního módu a změny ukládá.

4.5.3 LTSupportAutomotive

LTSupportAutomotive[5] je knihovna, která je k dnešnímu dni stále upravována. Aktuálně se pracuje na zlepšení podpory jazyka Swift. Knihovna je napsána v Objective-C a sám autor píše, že zatím neplánuje přepisovat kód do Swiftu. Na rozdíl od předchozích dvou knihoven, poskytuje možnost připojení zařízení prostřednictvím Bluetooth. Skládá se ze čtyř důležitých tříd:

- **LTOBD2Adapter** – abstraktní třída reprezentující OBD-II adaptér posílající požadavky a přijímající odpovědi dle stanového protokolu,
- **LTOBD2Command** – abstraktní třída, ze které dědí třídy pro všechny známé OBD-II PIDy,
- **LTOBD2Protocol** – abstraktní třída reprezentující protokol vozidla. Knihovna podporuje protokoly ISO14230-4 (CAN), ISO15765-4, ISO9141-2, a SAEJ1850,
- **LTBTLESerialTransporter** – třída tvořící most mezi LTBTLE charakteristikami (čtení a zápis) a podtřídami NSStream (NSInputStream a NSOutputStream).

[2] Dostupná na: https://github.com/FuzzyLuke/OBD2Kit
[3] Dostupná na: https://github.com/amaechler/OBD2Analyzer
[4] Dostupná na: https://github.com/lemberg/obd2-swift-lib
[5] https://github.com/mickeyl/LTSupportAutomotive/tree/SPM

LTSupportAutomotive nepřichází s návodem, jak knihovnu používat. Poskytuje pouze demonstrační příklad, který je přehledný a využívají se zde všechny komponenty knihovny.

Doporučil bych ji těm, kteří plánují v budoucnu pracovat na iOS aplikaci, která má za úkol komunikovat s vozidlem. Hlavním důvodem je stálé zlepšování a kvalita aktuálního stavu knihovny. Mezi vylepšení, které by mohly v budoucnu přijít je například rozšíření lokalizace (aktuálně je přeložena knihovna do tří jazyků, a to angličtiny, němčiny a francouzštiny), přidání diagnostických chyb a PIDů specifikovaných prodejcem, či přímou komunikaci s ECU.

Kapitola 5

Existující řešení

Na trhu existuje spousta aplikací, které uživateli poskytují informace pomocí nástrojů s ELM327 a Bluetooth modulem. Aplikace byly vybrány na základě hodnocení na stránce Magneto[1]. Článek byl zveřejněn 29.12.2018 a vypisuje 17 nejlepších aplikací podle stránkou určených kritérií. Vybral jsem tři nejlépe hodnocené aplikace. Všechny aplikace mají společné, že komunikují z valné většiny přes Bluetooth a snaží se různými způsoby poskytnout uživateli cenné informace ze senzorů. Způsoby, kterými se informace zobrazují, jsou hlavním rozdílem mezi aplikacemi, a proto budou detailněji popsány v kapitolách níže.

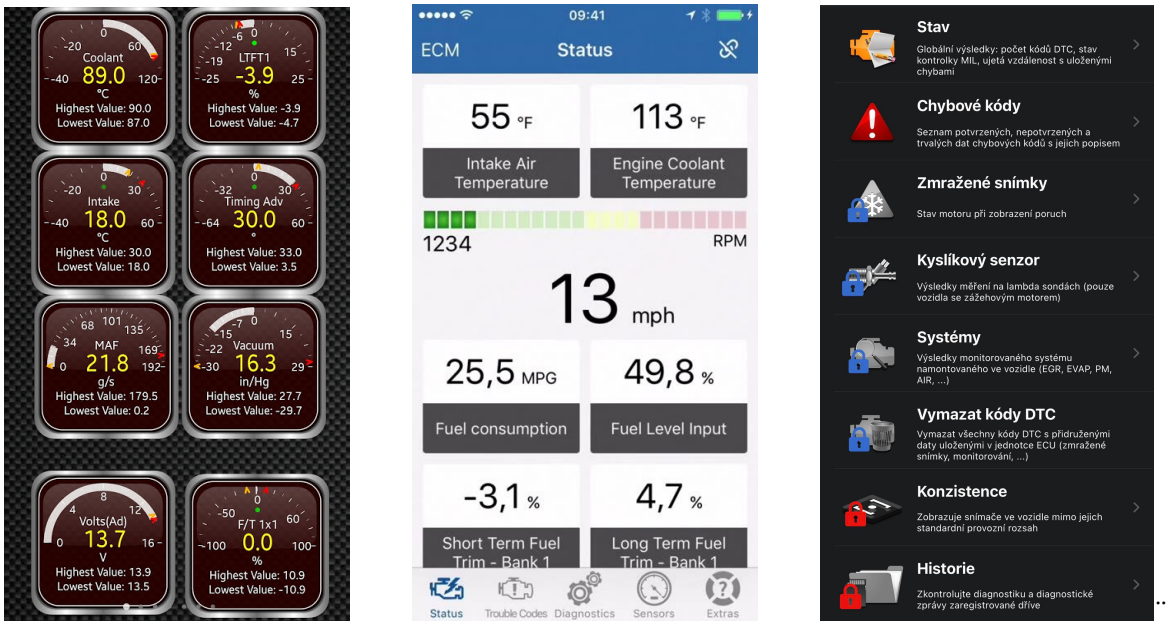
5.1 Torque Pro

Torque Pro je jedna z nejlepších aplikací, kterou si může vlastník mobilního telefonu s operačním systémem Android pořídit. Poskytuje uživateli možnost pořídit si aplikaci zdarma s omezenou funkcionalitou (Torque light) a umožnit mu tím náhled do aplikace, aby se mohl rozhodnout, zda má o placenou funkcionalitu zájem. Velice příjemná je možnost vytvoření vlastní obrazovky složené z různých ukazatelů aktuálních hodnot automobilu, jejíž ukázku je možné vidět na obrázku 5.1a. V případě, že ve vozidle přesáhne některá z hodnot svou kritickou hranici, je na událost uživatel upozorněn. Aplikace jako taková dokáže přehledně poskytnout všechny zobrazitelné informace, které by uživatel mohl chtít, ale s tím přichází i hůře pochopitelné prostředí pro nové uživatele. Používání bere též dost výkonu telefonu, čímž vyvíjí tlak na spotřebu baterie, která může být vyšší.

5.2 OBD Auto Doctor

OBD Auto Doctor je další aplikací, která stojí za zmínku. Oproti Torque nenabízí uživateli sestavit svoji vlastní obrazovku plnou dat ze senzorů, ale má přednastavenou sadu informací, kterou ocení jakýkoliv typ uživatele. Díky přednastavené sadě se stává aplikace jednodušší pro uživatele s méně zkušenostmi s podobnými aplikacemi. Dále nabízí grafy udávající změnu vybraného senzoru za posledních 30 sekund. Pomocí grafů je možné výrazně lépe odhalit, zda se sledovaná část automobilu chová správně. Aplikaci je možné si bezplatně stáhnout a vyzkoušet různé funkce. Plná verze aplikace se řeší pomocí předplatného, které činí 550kč/rok. Cena za předplatné se pohybuje na úrovni aplikací, které poskytují plnou funkcionalitu na neomezenou dobu. Čas potřebný pro navázání spojení je delší, než u jiných

[1] https://magnetoitsolutions.com/blog/best-obd2-android-ios-apps-for-cars-review



(a) Torque Pro [12] (b) OBD Auto Doctor [7] (c) EOBD Facile

Obrázek 5.1: Ukázka obrazovek aplikací

aplikací. Pro účely předvedení, jak aplikace vypadá, jsem si vybral úvodní obrazovku, která obsahuje předdefinované informace (obrázek 5.1b).

5.3 EOBD Facile

Poslední zmíněnou aplikací je EOBD Facile. Přichází s celou řadou zajímavých funkcí, kterými jsou terminální příkazy, diagnostika jednotlivých řídicích jednotek, získání informací o vozidle na základě identifikačního čísla automobilu (VIN), atd. Oproti výše zmíněným nenabízí možnost sledování hodnot na uživatelsky přívětivé obrazovce. Zaměřuje se spíše na důkladné zpracování diagnostických chyb a výpis aktuálních dat poskytuje jen prostřednictvím tabulky a grafů. Hlavní nevýhodou aplikace je její cena za přístup ke všem funkcím, které nabízí. Nabízeny jsou dvě varianty. Základní edice stojí kolem 450kč a Plus edice 930kč. Funkce základní edice jsou na obrázku 5.1c označeny modrým zámkem a Plus funkce červeným. Různé jiné aplikace podobného typu nabízejí též některé funkce za příplatek, ale ceny za ně se pohybují kolem 100 až 150kč a výsledky jsou srovnatelné. Za zmínku stojí omezený počet viditelných grafů, které sledují hodnoty senzorů. Je možné sledovat jen čtyři senzory v jeden okamžik, což může být v určitých situacích nedostatečný počet.

5.4 Názor na existující aplikace

Mezi hojně používané diagnostické aplikace, jak už bylo zmíněno v kapitole 5, patří Torque Pro, OBD Auto Doctor a EOBD Facile. Většina existujících aplikací komunikujících s vozidly za pomocí OBD-II jsou aplikace čistě na diagnostiku vozidla. Po vyzkoušení tohoto druhu aplikací jsem došel k závěru, že běžný vlastník vozidla neporozumí takřka polovině informací, které po něm aplikace vyžaduje. Dalším negativním aspektem u většiny aplikací

je způsob předávání informací. Často je uživatel zahlcen informacemi, které se aplikace snaží vměstnat na co nejmenší prostor a mohou působit nepřehledným dojmem.

Přesně na tyto problémy jsem se snažil zaměřit při implementaci mnou vytvořené aplikace. Uživatel dostává jen to podstatné a není zahlcen otázkami.

Kapitola 6

Návrh aplikace

V následující kapitole se věnuji návrhu aplikace. Konkrétně se budu zabývat návrhem řešení, poté rozeberu jednotlivé části aplikace a vysvětlím jejich smysl. Popíši vývoj uživatelského rozhraní a nakonec rozeberu objektový graf CoreData, který vznikl pro uchování dat jednotlivých jízd.

6.1 Získání a zpracování dat

Získání a zpracování dat je hlavní funkcí aplikace. Data jsou sbírána v době, kdy uživatel provádí jízdu svým vozidlem. Po ukončení jízdy jsou data zpracována a následně uložena. V průběhu snímání se sbíraná data ukládají do polí, ve kterých přetrvávají do doby, než je uživatel bude chtít vyhodnotit. Po zadání příkazu, aby se data vyhodnotily, se provedou operace potřebné pro získání dat, která nejsou přímo získatelná z vozidla. Výsledky jízdy jsou předány uživateli a uloženy do databáze s možností zpětného náhledu na provedenou jízdu.

6.2 Sbíraná data

Mezi sbíraná data byla vybrána jen ta, která se dají získat z jakéhokoliv vozidla podporující OBD-II. I když vozidlo podporuje OBD-II, není povinností, aby všechny PIDy poskytovaly hodnoty. Například teplota oleje je sice jedním ze standardních PIDů, ale neposkytují ho všechna vozidla. Finální seznam sbíraných dat z vozidla je následující:

- carSpeed – rychlost vozidla,
- engineLoad – zatížení motoru,
- coolantTemp – teplota chladící kapaliny,
- engineRPM – otáčky motoru za minutu,
- longTermFuel – dlouhodobá úprava paliva,
- shortTermFuel – krátkodobá úprava paliva,
- relativeThrottlePos – jak moc je otevřená škrticí klapka,
- intakeMAP – tlak vzduchu vstupující do vozidla,

- relativeAcceleratorPedalPos – pozice plynového pedálu,
- tankLevel – procentuální zaplnění palivové nádrže,
- fuelRailGaugePressure – tlak v rozvaděči paliva.

Další ukládaná informace je spotřeba paliva podobu jízdy. OBD-II neposkytuje informace o spotřebě paliva, lze ji ale přibližně spočítat vícero způsoby. V každém případě je potřeba znát masové proudění vzduchu, zkráceně MAF. OBD-II nenabízí tuto hodnotu vždy, a proto není možné se na přítomnost informace o MAF spoléhat. Obrázek 6.1, který pochází z článku [17] ukazuje, co je nutností, aby jsme MAF mohli spočítat i bez jeho znalosti. Přichází s dvěma rovnicemi:

$$MAF[g/s] = 1.184[g/l] \cdot EngDisp[l/intakestroke] \cdot \frac{loadabs}{100} \cdot \frac{\frac{enginespeed[rpm]}{2[rpm/intakestroke]}}{60[sec/min]} \tag{6.1}$$

Kde $EngDisp$ je objem motoru, $loadabs$ absolutní zatížení motoru, $enginespeed$ otáčky motoru a 2 počet otáčet za jedno nasání paliva a vzduchu. V rovnici v článku je možné vidět chybu a to za číslem dva, kde se nachází násobení, které je zde omylem.

$$MAF[g/s] = \frac{RPM \cdot MAP}{IAT \cdot 2 \cdot 60[sec/min]} \cdot \frac{VolEff}{100} \cdot EngineDisp \cdot \frac{MMAir}{R} \tag{6.2}$$

Kde RPM jsou otáčky za minutu, MAP tlak vzduchu, IAT teplota vstupního vzduchu ve stupních Kelvina, $VolEff$ objemová účinnost, $EngDisp$ objem motoru, $MMAir$ molární hmotnost vzduchu a R konstanta $8.314 J/°K/mole$.

Po výpočtu MAF je možné spočítat vzorec 6.3 za pomoci tabulky 6.1, kde AFR je ideální poměr vzduchu a paliva a FD hustota paliva.

$$FuelFlow[l/h] = \frac{MAF \cdot 3600}{AFR \cdot FD} \tag{6.3}$$

Typ paliva	Poměr vzduchu a paliva	Hustota
Benzín	14.7:1	820
Nafta	14.5:1	750

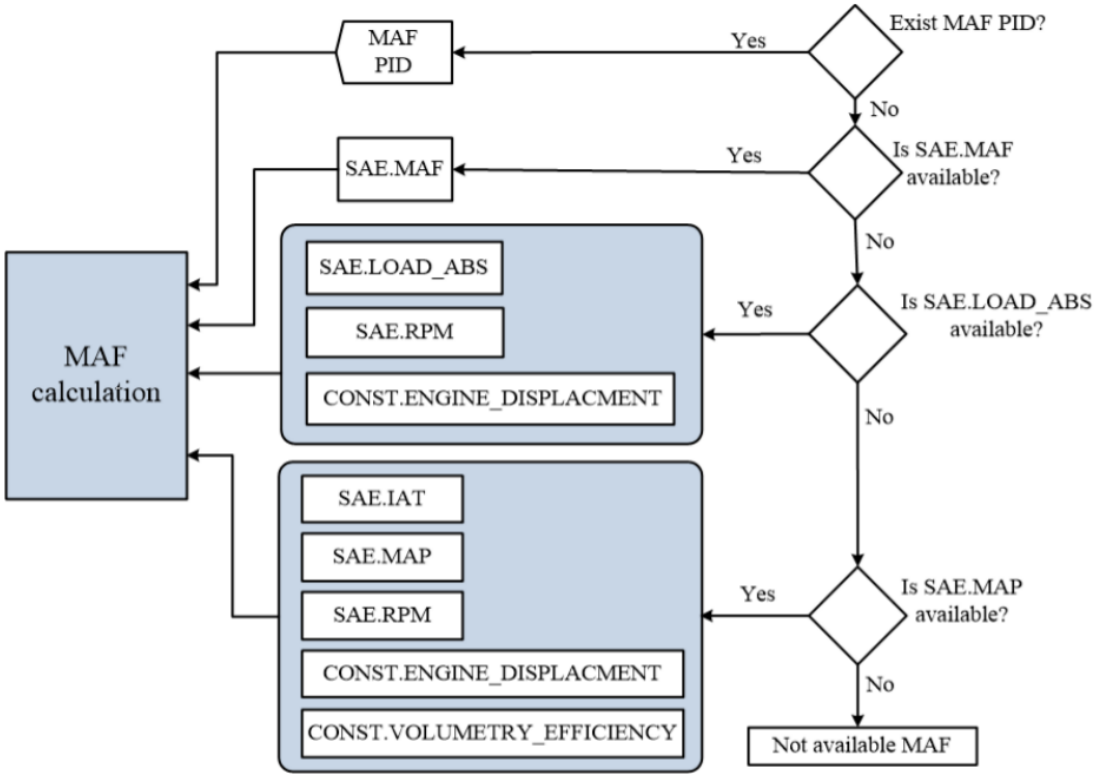
Tabulka 6.1: Ideální poměr vzduchu na gram paliva a hustota paliva

Výsledkem je výpočet spotřeby litrů za hodinu. Pro vytvoření spotřeby litrů za 100 kilometrů bylo využito vzorce:

$$Consumption[l/100km] = \frac{FuelFlow[l/h]}{vehicleSpeed[km/h]} \cdot 100 \tag{6.4}$$

Výpočet spotřeby tímto způsobem je pouze přibližný, protože přesně neznáme objemovou účinnost ($VolEff$), jenž je možné přesně spočítat pouze v případě, že známe MAF. V mém případě počítám s průměrnou hodnotou $VolEff$, a to 80%, čímž do výpočtu vnáším nepřesnost.

Kombinací všech výše zmíněných vzorců dostávám přibližnou spotřebu vozidla, která je taktéž sbírána a ukládána.



Obrázek 6.1: Způsoby výpočtu MAF

6.3 Detekce chyb při jízdě

Chyby při jízdě jsou jednou z nejčastějších příčin dopravních nehod na silnici. Je dost pravděpodobné, že si některé chyby řidič po několika letech za volantem ani neuvědomuje. Aplikace, které sledují řidičovu jízdu a dávají mu rady, jak se v určitých částech jízdy zlepšit, napomáhají menšímu počtu nehod, a také úsporám za nové náhradní díly. Za dobu vývoje aplikace byly vymyšleny dva způsoby, kterými by se jízda analyzovala.

Prvním z návrhů bylo využití neuronové sítě (konkrétně LSTM), která se naučí na sadě dat rozlišovat správně jezdící a agresivně jezdící jezdce a bude jízdy zařazovat do spektra těchto dvou extrémů. Bohužel jsem nenašel volně dostupnou datovou sadu dostatečně rozsáhlou a popsanou na to, abych mohl neuronovou síť naučit validním výsledkům.

Druhý návrh, který byl i nakonec zrealizován, je založen na hlídání určitého počtu hodnot a zda nevykračují mimo hranice znamenající přechod mezi normální a špatnou jízdou. Hranice z tabulky 6.2 byly definovány v článku [6], který se zabývá analýzou řidičova chování a algoritmem AdaBoost. Pro výpočet relativního poměru rychlosti vozidla a motoru v čase t je definována rovnice:

$$R_{cz}(t) = \frac{\frac{cs(t)}{220}}{\frac{zs(t)}{8000}} \tag{6.5}$$

Kde $cs(t)$ je rychlost v čase t a $zs(t)$ jsou otáčky v čase t. Relativní poměr rychlosti motoru a pozice škrticí klapky je počítán následovně:

$$R_{jz}(t) = \frac{\frac{jq'(t)}{max(jq')}}{\frac{zs'(t)}{max(zs')}} \tag{6.6}$$

Kde $jq'(t)$ značí změnu pozice škrticí klapky v čase t od času $t-1$, $max(jq')$ maximální změnu pozice klapky za celou dobu jízdy, $zs'(t)$ změnu v otáčkách za minutu a $max(zs')$ maximální změnu v otáčkách.

Rovnici 6.5 lze počítat v průběhu jízdy, zatím co rovnici 6.6 je nutné propočítávat až po ukončení jízdy.

Proměnná	Normální jízda	Špatná jízda
Relativní poměr rychlosti vozidla a motoru	0.9 – 1.3	> 1.3 nebo < 0.9
Relativní poměr rychlosti motoru a škrticí klapky	0.9 – 1.3	> 1.3 nebo < 0.9
Zatížení motoru	20%–50%	$< 20\%$ nebo $> 50\%$

Tabulka 6.2: Tabulka určující rozmezí hodnot proměnných pro normální a špatnou jízdu

V článku [6] na určování správnosti jízdy, se využívá algoritmu AdaBoost. Učí se za pomoci datové sady jaké z pravidel má jakou vypovídající hodnotu a podle toho se řídí při rozhodování. V mém případě jsem se rozhodl využít pouze definované tabulky mezí a předávat uživateli informace o tom, v jakých úsecích jízdy vykročil z normy a jak se této chybě příště vyhnout.

6.4 Ukládání jízd

Sbíraná data mohou být uživateli užitečná. Po ukončení jízdy nejsou informace předány pouze jednorázově, ale jsou zároveň uloženy. Uživatel se na ně může zpětně podívat a porovnat několik jízd na stejné trase. Další výhodou ukládání je možnost přeposlat si jízdy ve formátu PDF na e-mail. Výsledné PDF má vzhled knihy jízd. Kniha se může hodit např. těm uživatelům, kteří si své jízdy musí zaznamenávat.

6.5 Diagnostika

Diagnostika je hlavní funkcí valné většiny aplikací, které využívají pro svou funkčnost OBD-II. Mnou navržená aplikace bere diagnostickou část jako funkci vedlejší a zároveň uživatelsky užitečnou. Určitá část uživatelů diagnostických aplikací si aplikaci nainstalují až po tom, co se vyskytne na jejich vozidle porucha. Z tohoto důvodu byly do aplikace přidány hlavní funkce diagnostických zařízení jako sledování hodnot měnících se v čase, monitorování senzorů a diagnostické chybové hlášky.

6.6 Uživatelské rozhraní

Uživatelské rozhraní je jeden ze stavebních kamenů aplikace. Pro jeho správnou implementaci je na místě znalost Human Interface Guidelines. Mnou navržená aplikace je rozdělena do čtyř částí, do kterých se lze dostat prostřednictvím rozcestníku.

Rozcestník je úvodní obrazovkou aplikace. Skládá se ze čtyř částí. První z nich jsou rafičky imitující informace na palubní desce. Slouží zejména pro ujištění, že informace z vozidla jsou sbírána správně. V druhé části obrazovky je poskytována rychlost společně s aktuálním omezením na daném úseku. Třetí část je ta, díky které obrazovka získává svůj název rozcestník. Obsahuje sadu čtyř tlačítek, které uživateli umožňují přejít například na snímání jízdy. Poslední část je zaměřena na způsob spojení s OBD-II zařízením, jenž je realizováno prostřednictvím tlačítka na připojení, nebo odpojení. Dále obsahuje informaci o aktuálním stavu připojení, která hraje hlavní roli informátora při připojování k zařízení.

Snímání jízdy je obrazovkou, kterou uživatel vidí při snímání jeho jízdy. Obsahuje celou řadu informací od vývoje otáček za minutu za posledních 20 sekund, přes spotřebu a rychlost vozidla, až po méně důležité informace, jako například nadmořskou výšku. Obrazovka se zaměřuje výhradně jen na předávání zajímavých informací pro řidiče a neumožňuje jiné akce než dokončení snímání a zrušení snímání. V případě dokončení snímání se přejde na obrazovku **Vyhodnocení snímání**. Poskytuje základní informace o jízdě – odkud kam vedla cesta, procentuální ukazatel správně zajeté trasy a mapu trasy společně se seznamem chyb.

Kniha jízd slouží jako seznam jízd, které daný uživatel absolvoval. Každý záznam je popsán názvem jízdy, start a cíl trasy a kdy byla provedena. Jedná se o klíčové informace pro vyhledávání jízd v seznamu. Z knihy jízd je možné přejít na specifický záznam jízdy, či na PDF podobu knihy jízd. V případě volby přejít na specifickou jízdu, se přejde na obrazovku **Detail jízdy**, která poskytuje detailnější informace o jízdě. Je možné si zde prohlédnout chyby, či graf vývoje hodnot všech sbíraných proměnných popsaných v podkapitole 6.2.

Diagnostika je jednou ze dvou obrazovek, které nabízí hlavní funkce diagnostických aplikací. Konkrétně se zde jedná o monitorování senzorů a výpis diagnostických chybových kódy, ať už se jedná o kódy aktuální, čekající, nebo trvalé. Spuštění monitorování a získání výpisu chyb je možné za pomoci tlačítka v pravém horním rohu. V případě potřeby je možné aktuálně aktivní chybové kódy smazat pomocí tlačítka vedle nápisu „Chybové kódy“.

Hodnoty senzorů zastupují druhou z funkcí diagnostických aplikací, a to seznam hodnot všech základních PIDů módu 1. Jedná se o seznam, který je rozdělen do tříd podle toho, jakou z věcí daný senzor popisuje (například motor). Třídy je možné zabalit v případě, že o ně uživatel nemá zájem, čímž sníží počet informací, které vidí a nechá si jen na ty preferované. Pro získání nových dat je připraveno tlačítko v pravém horním rohu, které způsobí vyžádání nových hodnot z vozidla pro aktuálně viditelné senzory.

Aplikace dále v případě, že se nacházíme na obrazovce s rozcestníkem, obsahuje postranní menu. Přistupovat se k němu dá prostřednictvím ikony v levém horním rohu, nebo pomocí gesta táhnutí obrazovky směrem doprava. Menu obsahuje volbu aktuálně aktivního uživatele, jeho přidání a smazání. Stejně tak tomu je s vozidly, kde uživatel zadává název pro své vozidlo a jeho objem motoru pro výpočet průměrné spotřeby. Posledním prvkem je nastavení zobrazení sekcí u hodnot senzorů. Uživatel má možnost rozhodnout se mezi tím, jestli budou všechny sekce otevřeny, nebo zavřeny.

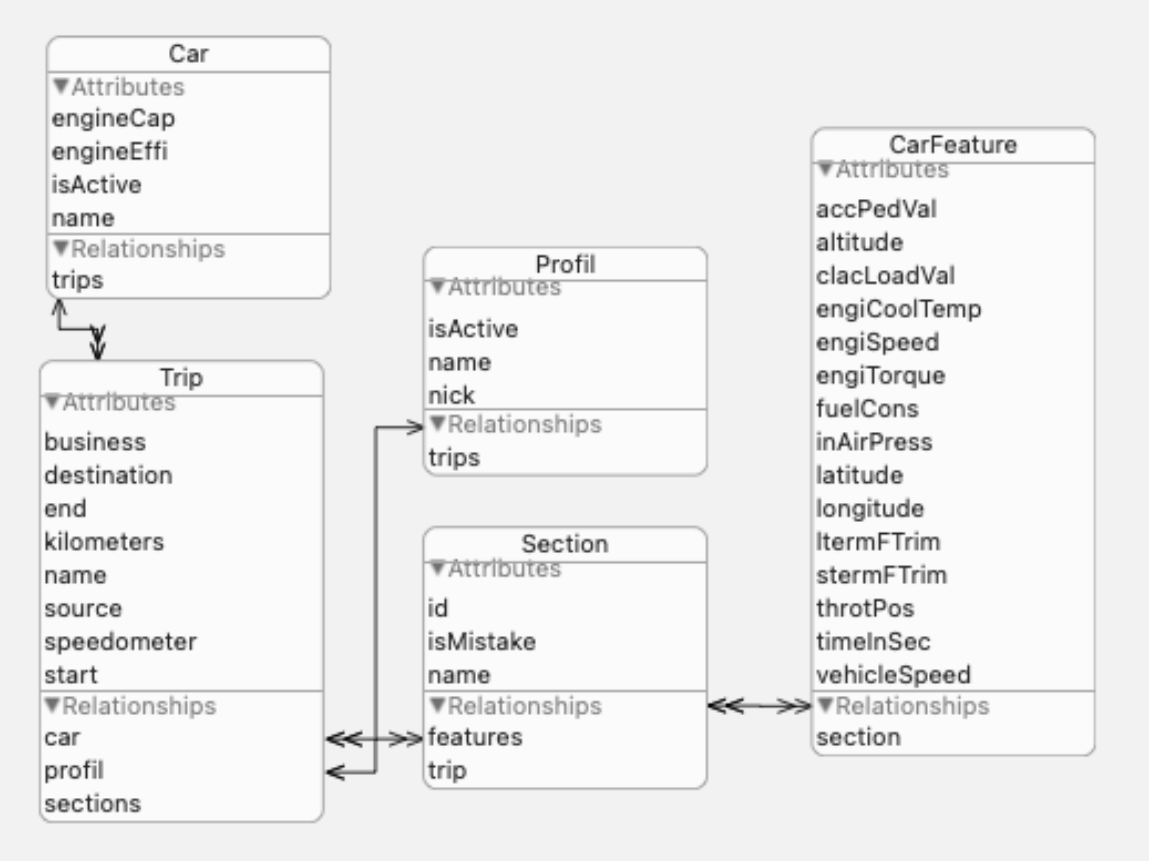
6.7 Vztahový diagram

Vztahový diagram je způsob vizualizace ukládaných dat v podobě entit a vztahů mezi entitami. Aplikace ukládá pět různých entit:

- Car – nese informace o vozidle,
- Profil – obsahuje informace o uživateli,
- Trip – poskytuje informace o jízdě (například kde začala a skončila),
- Section – je sekce jízdy, která nemá předem definovanou délku,
- CarFeature – je seznam hodnot proměnných v dané fázi jízdy.

Obsah entit je znázorněn v diagramu 6.2, který poskytuje celkový pohled na to co se ukládá. K jednomu profilu uživatele i vozidla je možné přiřadit až nekonečně mnoho jízd. Stejně tak tomu je s jízdou a sekcemi, kde jízda může mít až nekonečně mnoho sekcí. Sekce díky tomu, že není časově omezená, může mít až nekonečně mnoho seznamů hodnot vozidla. Naopak daný seznam hodnot vozidla musí patřit právě jedné sekci, daná sekce právě jedné jízdě a jízda pouze jednomu uživateli a vozidlu. Entita Car disponuje atributy nesoucí následující význam:

- engineCap – kapacita motoru vozidla,
- engineEffi – účinnost motoru,
- isActive – zda má být vozidlo aktuálně využito při snímání,
- name – identifikátor vozidla.



Obrázek 6.2: Objektový graf CoreData

EngineCap je důležitá informace pro výpočet průměrné spotřeby, o které jsou zmínky v podkapitole 6.2. IsActive atribut určuje, které vozidlo je momentálně využíváno (aktivní může být v jeden čas právě jedno vozidlo). Name složí pro rozeznání vozidel při výběru toho, které má být zrovna aktivní.

Další entitou je Profil jehož atributy jsou:

- isActive – zda je profil aktivní,
- name – jméno řidiče,
- nick – přezdívka řidiče.

IsActive má stejný účel jako u entity Car a to určovat, zda se jedná o právě používaný profil. Name a nick mají podobnou vlastnost, a to rozlišování uživatelů s rozdílem, že nick může obsahovat i číslice, zatímco name nikoliv.

Ke každému profilu se přiřazují jízdy, které absolvoval. Každá jízda je popsána atributy:

- business – zda se jedná o služební jízdu, či nikoli,
- destination – adresa místa, kde skončila jízda,
- end – čas dokončení jízdy,
- kolometers – počet ujetých kilometrů,

- name – jméno jízdy ,
- source – adresa místa, kde začala jízda,
- speedometer – stav tachometru před jízdou,
- start – čas započetí jízdy.

Jako u každé správné jízdy je důležité vědět, kdy a kde začala a skončila. Dále byly vytvořeny dva atributy konkrétně business a speedometer, které jsou určené pro možné exportování jízd v podobě knihy jízd. Jméno jízdy je určeno hlavně k identifikaci při zpětném dohledávání v seznamu jízd.

Všechny jízdy se sestávají ze sekcí. Každá sekce je tvořena skupinou atributů:

- id – identifikátor sekce,
- isMistake – zda se jedná o chybovou sekci,
- name – název sekce.

Id je zde z důvodu správného seřazení sekcí jedné jízdy za sebou. IsMistake informace je využita při vykreslování trasy jízdy, pro rozlišení správně zajetých a špatně zajetých úseků. Name poskytuje seznamu chyb informaci o tom, jaká chyba se v daném úseku stala.

Každá sekce je složena z bodů jízdy, ve kterých se snímá hodnota určitého počtu proměnných. Jeden bod jízdy je reprezentován jako objekt s názvem CarFeature, který obsahuje atributy:

- accPedVal – hodnota akceleračního pedálu,
- altitude – nadmořská výška,
- clacLoadVal – zatížení motoru,
- engiCoolTemp – teplota chladicí kapaliny,
- endiSpeed – otáčky za minutu,
- engiTorque – točivý moment motoru,
- fuelCons – spotřeba paliva,
- inAirPress – tlak vstupního vzduchu,
- latitude – zeměpisná šířka,
- longitude – zeměpisná délka,
- ltermFTrim – dlouhodobá úprava paliva,
- stermFTrim – krátkodobá úprava paliva,
- throtPos – pozice škrticího ventilu,
- timeInSec – čas od začátku jízdy,
- vehicleSpeed – rychlost vozidla.

Většina atributů je známá již z podkapitoly 6.2, ale jsou zde i atributy, které jsou získávány ze senzorů, které jsou obsažené v mobilním zařízení. Navíc je zde informace o čase, kdy byla sada hodnot získána.

Kapitola 7

Implementace

Aplikace byla implementována v programovacím jazyce Swift verze 5. Vývoj aplikace probíhal ve vývojovém prostředí Xcode verze 12.4 na MacBooku od firmy Apple. V kapitole se nejdříve zabývám uživatelským rozhraním, které bylo vytvořeno s pomocí SwiftUI a dále dopodrobna popisuji způsoby implementace komunikace, předávání informací, ukládání dat, diagnostiky a knihy jízd. V podkapitolách jsou poté popsány opensource knihovny jako například SwiftUICharts[1] a jejich modifikace pro potřeby aplikace.

7.1 Uživatelské rozhraní

Stejně jak aplikace, které jsou vyvíjené za pomoci UIKitu a Storyboardu, tak i SwiftUI poskytuje předem definované prvky rozhraní. Mezi ty patří například:

Text je jedním z nejzákladnějších prvků. Jeho funkcí je předávat na obrazovku text, který není možné uživatelem jakkoliv upravovat. Text je možné měnit za běhu aplikace pouze uvnitř kódu prostřednictvím proměnných, které mají před sebou některý z modifikátorů proměnných, třeba @State. K textu je dost často potřeba připojit takové modifikátory, které budou měnit velikost textu podle toho, jak je velká obrazovka, na které se aplikace právě zobrazuje. Při nastavení pevné velikosti textu může docházet k problémům malé velikosti textu u velkých obrazovek a naopak příliš velkého textu u menších obrazovek.

Button se stará o interakci s uživatelem pomocí gest. Po stisknutí tlačítka se provede akce, která je předem definována.

List je SwiftUI verze `UITableView`, která se zbavuje potřeb definovat řadu funkcí, která udává počet řádků, sekcí a vzhled řádku. Díky způsobu, kterým se ve SwiftUI vytváří uživatelské rozhraní, je možné vše nadefinovat v Listu. List podporuje For cykly, ve kterých se iterativně vytvoří řádky podle toho, jak si je uživatel sám vytvořil pomocí jiné struktury. List sebou přináší řadu nevýhod jako například oddělovací čáru mezi každým řádkem, což je limitující při snaze vytvořit vlastní vzhled. Možnosti, jak se zbavit limitací na tvorbu vlastního seznamu, je využití `ScrollView`, jenž pochází z `UIScrollView`. Jedná se o prvek rozhraní umožňující vertikální, či horizontální pohyb po obrazovce. S využitím vertikálního zásobníku VStack je možné replikovat funkcionalitu Listu.

[1] Dostupné na: https://github.com/AppPear/ChartView

Pro tvorbu vizuální stránky aplikace se v případě SwiftUI nevyužívá Storyboardu, jak je tomu u vývoje aplikací s využitím frameworku UIKit. SwiftUI využívá struktur, které odpovídají View protokolu. View protokol je typ, který představuje část uživatelského rozhraní a poskytuje modifikátory, které se využívají pro konfiguraci této části. Každá struktura musí obsahovat tělo, které se skládá ze základních prvků frameworku SwiftUI, nebo vámi již definovaných struktur. Pro ukázku takové struktury jsem si připravil krátký výpis z aplikace7.1 znázorňující styl řádku diagnostických chybových kódů. Obsahuje dva prvky typu Text, které se starají o vizualizaci textu na obrazovku aplikace, a HStack, který řadí prvky na obrazovce horizontálně vedle sebe. Ke každému z prvků je přiřazena sada modifikátorů, které upřesňují chování každého z nich. Například *.background(Color.white)* říká, že má být pozadí horizontálního zásobníku bílé barvy.

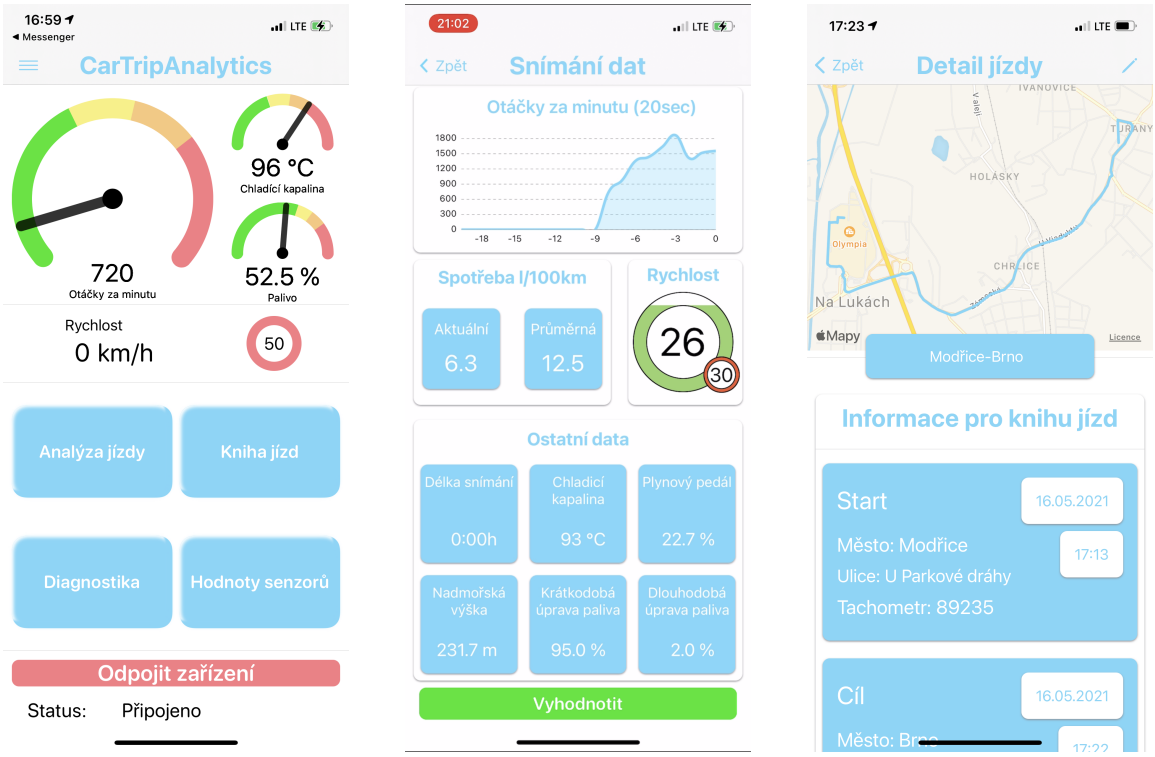
```
1   import SwiftUI
2
3   struct dtcRow: View {
4       var text: String = "Loading..."
5       var detail: String = "--"
6       var body: some View {
7           HStack {
8               Text(text)
9                   .accentColor(.primary)
10                  .padding()
11              Spacer()
12              Text(detail)
13                  .accentColor(.primary)
14                  .padding()
15          }
16          .background(Color.white)
17          .cornerRadius(8)
18          .shadow(color: .gray, radius: 1, y: 1)
19          .padding(.horizontal)
20      }
21  }
22
```

Výpis 7.1: Struktura odpovídající View protokolu

Pro navigaci mezi obrazovkami bylo využito `NavigationView` a `NavigationLink`. Prvek `NavigationView` obaluje prostor, který se bude při přechodu na jinou obrazovku měnit. V mém případě `NavigationView` obaluje obsah celé první obrazovky, aby byla při přechodu kompletně vyměněna. Pro přechod z jedné obrazovky na druhou se při použití `NavigationView` využívá `NavigationLink`, který nahradí obsah `NavigationView` za jiný. Vizuální reprezentace `NavigationLink` může být jakákoli jako například text, tlačítko nebo obrázek. Výsledné uživatelské rozhraní je možné vidět na obrázku 7.1 a v příloze C. Všechny využité knihovny na obrázcích budou popsány v podkapitolách níže.

7.2 Komunikace s vozidlem

Pro komunikaci s vozidlem jsem si vybral knihovnu LTSupportAutomotive, která je jedinou knihovnou podporující připojení k zařízení za pomoci Bluetooth. Uživatel má možnost se k zařízení připojit prostřednictvím tlačítka ve spodní části obrazovky 7.1a. O kompletní komunikaci se stará třída OBD2Controls, která se pomocí funkcí `prepareConnection()` a `connection()` postará o propojení aplikace a OBD-II zařízení. Spojení může nabývat tří stavů a to „Připojen“, „Odpojen“ a „Protokol není podporován“.



(a) Rozcestník (b) Snímání dat (c) Uložená jízda

Obrázek 7.1: Ukázka vybraných obrazovek aplikace

V případě bezproblémového spojení začne aplikace automaticky každou sekundu sbírat data z vozidla a ukládat je do předem určených proměnných třídy. Uživatel je o průběhu připojování informován pomocí textu stavu pod tlačítkem pro připojení na obrázku 7.1a. Jakmile je spojení dokončeno, je stav připojení nastaven na „Připojeno“, rafičky v horní částku obrazovky rozcestníku se nastaví na hodnoty získané od vozidla a tlačítko se změní na možnost odpojení se od zařízení.

Jakmile je spojení ustáleno, je aktivován časovač, který každou sekundu zavolá funkci `getValuesFromCar(pids: [LTOBD2Command])` objektu OBD2Controls. Funkce je volána, aby upravovala hodnoty otáček motoru, rychlosti vozidla, teploty chladicí kapaliny a procentuální naplnění palivové nádrže. Získané hodnoty nastavují pozici rafiček u tachometrů ve vrchní části rozcestníku 7.1a.

7.3 Způsoby předávání informací uživateli

Aplikace předává celkem tři typy informací. Ty, které získá z vozidla, informace o maximální povolené rychlosti v dané lokaci a polohu. Pro každý z typů informací byl vytvořen jiný způsob prezentace dat.

7.3.1 OBD-II hodnoty

OBD-II hodnoty jsou hodnoty, které se sbírají v čase a mohou být užitečné i zpětně. Při akci sbírání dat jsem zvolil variantu předávání aktuálních dat v textové podobě společně s využitím grafové reprezentace otáček za minutu, u kterých je dobré vidět změny za poslední

dobu místo jednoho čísla. Na vizualizaci grafu otáček jsem využil knihovnu Charts[2], která jako jedna z mála grafových knihoven použitelných ve SwiftUI poskytuje grafy s hodnotami na obou osách. Obrazovka snímání 7.1b znázorňuje přístup, který jsem zvolil. Uchovává se pouze posledních 20 sekund záznamu o otáčkách, díky kterým není graf přeplněný hodnotami a zároveň jsme schopni detekovat neobvyklé chování vozidla.

7.3.2 Rychlostní limit

Dodržování rychlostního limitu je jednou ze zásad bezpečné jízdy. Pro získání rychlostního limitu v místě, kde se uživatel zrovna nachází, jsem využil projektu OpenStreetMap[3]. V mém případě jsem OpenStreetMapy použil jako zdroj informací o pozemních komunikacích, na kterých se uživatel právě pohybuje. Komunikace probíhá prostřednictvím zasílání žádostí každých pět sekund, které v sobě obsahují lokaci pro kterou bych rád znal rychlostní limit. Lokace se zadává v podobě obdélníku, který v aplikaci vzniká modifikací aktuální polohy uživatele. Odpovědí je XML[4], který obsahuje různé informace o silnicích, které se v daném obdélníku nacházejí. V získané odpovědi nalezne element s názvem *tag*, který obsahuje atribut *k* s hodnotou *maxspeed* a uloží si hodnotu, která je u atributu *v* stejného elementu. Výpis 7.2 představuje způsob realizace komunikace a nalezení hledané hodnoty.

```
1         //Zaslani pozadavku a vlozeni odpovedi do XMLParseru
2     @Published var limit: String = "50"
3     private var offset: Double = 0.00005
4     guard let url = URL(string:
      "https://www.overpass-api.de/api/xapi?*[maxspeed=*]
5             [bbox=\(longitude-offset),\(latitude-offset),
6             \(longitude+offset),\(latitude+offset)]")
7     else {
8         print("Invalid URL")
9         return
10    }
11    let request = URLRequest(url: url)
12    URLSession.shared.dataTask(with: request) { (data, response, error) in
13        if let error = error {return}
14        guard let data = data else {return}
15        let parser = XMLParser(data: data)
16        parser.delegate = self
17        parser.parse()
18    }
19    .resume()
20    //Nalezeni hodnoty v odpovedi (XMLParser)
21    DispatchQueue.main.async {
22        if elementName == "tag" && !attributeDict.isEmpty{
23            if Array(attributeDict.values).contains("maxspeed") {
24                for (key, value) in attributeDict {
25                    if key == "v" { self.limit = value }
26                }
27            }
28        }
29    }
30
```

Výpis 7.2: Komunikace s OpenStreetMap

[2]Dostupná na: https://github.com/danielgindi/Charts
[3]Dostupné na: https://wiki.openstreetmap.org/wiki/Main_Page
[4]https://en.wikipedia.org/wiki/XML

Limit rychlosti je přítomen na dvou obrazovkách. Jednou z nich je rozcestník 7.1a, který vizualizuje limit v podobě dopravní značky. V případě změny limitu se limit na značce změní. Zde ale nijak nezasahuje do vizuální podoby čísla rychlosti, protože nepředpokládá, že se bude provádět jízda bez zapnutého snímání. Druhé místo, kde se rychlostní limit vyskytuje, je obrazovka, při které se snímá jízda. Zde je limit také vizualizován v podobě dopravní značky, ale na rozdíl od rozcestníku se zde mění i podoba aktuální rychlosti vzhledem k limitu. Na obrázku 7.1b jsou vidět různé stavy ukazatele aktuální rychlosti, kdy se postupně naplňuje podle toho, jak moc se rychlost blíží k limitu. Za předpokladu, že je obrazovka snímání jízdy aktivní právě při jízdě, je zapotřebí uzpůsobit prvky tak, aby byly lehce čitelné a neodváděly pozornost od řízení. Realizací rychlostního limitu se takového efektu snažím dosáhnout.

7.3.3 Poloha

Posledním ze sbíraných dat je poloha. Sbírání polohy je u aplikací, které sbírají v průběhu času data o vozidle, velice důležité. Díky sbírání polohy je následně možné zjistit, v jakých místech se řidič dopustil svých chyb. Aby měl uživatel možnost podívat se na svou zajetou jízdu, je trať po ukončení vykreslována na mapu. Ukázku je možné vidět na obrázku C.1a.

7.4 Chyby řidiče

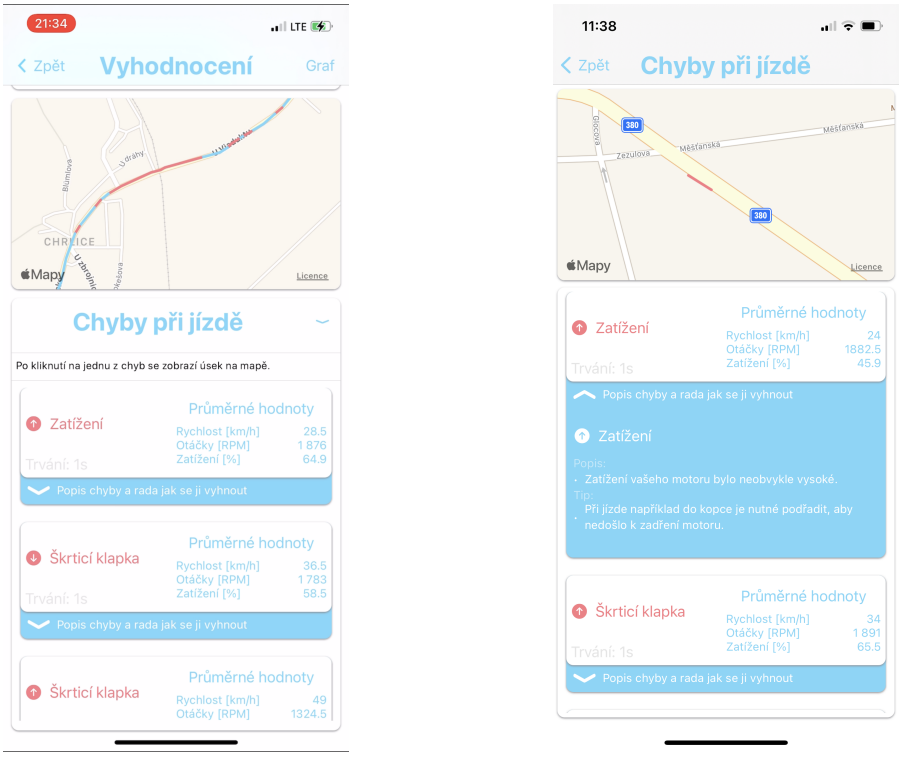
Poté, co uživatel ukončí snímání, se spustí vyhodnocování každé vteřiny jízdy. Každá vteřina je následně ohodnocena podle toho, zda překročila meze některé z proměnných zmíněných v podkapitole 6.3. V případě nepřekročení žádné hranice je vteřina považována za bezchybnou. V případě překročení alespoň jedné hranice je vteřina považována za chybnou. Pro rozlišení, o jakou proměnnou se jedná a zda je výsledek pod, či nad hranicí, se informace ukládají do pole. Každá proměnná má svůj index, kde:

- 0 – značí hodnotu v mezích,
- 1 – říká, že hodnota je nad mezí,
- 2 – říká, že hodnota je pod mezí.

V případě bezchybné vteřiny jízdy je tedy pole plné nul, zatímco při chybě je alespoň jeden index nastaven na jinou hodnotu.

Jakmile je celá jízda ohodnocena, je rozdělena do sekcí. Každá sekce je tvořena neomezeně dlouhou částí jízdy, ve které mají všechny vteřiny stejné pole (jedná se buď o správně zajetou část, či o část, ve které se vyskytovala stejná chyba). Sekce jsou pojmenovány podle chyb, které obsahují. Pro kratší zápis, jenž je možné vidět na obrázku 7.2a, jsem využil ikon, které znázorňují šipku nahoru (vykročení nad mezi) a šipku dolů (vykročení pod mezi).

Všechny chyby jsou vizualizovány jako seznam, kde jedna buňka představuje jednu sekci s chybou. Na obrázku 7.2a je vidět, že se skládá ze čtyř prvků. Prvním z nich je název, který může obsahovat až tři chyby najednou. Další informací je čas, jak dlouho se kombinace chyb vyskytovala. Ta není pro uživatele tolik důležitá, a proto není tolik výrazná, aby na sebe nestrhávala pozornost. Buňka navíc předává informaci o hodnotách proměnných stojících za rozhodnutím, že se jedná o chybu. V případě potřeby uživatele vědět více o významu a možnému řešení chyby, nabízí buňka ve své spodní části možnost rozbalit více informací. Názorná ukázka je na obrázku 7.2b.



(a) Seznam chyb

(b) Popis chyby a rada

Obrázek 7.2: Chyby při jízdě

Může se stát, že jízda bude dlouhá a chyb se při jízdě naskytne více, ale i přesto by uživatel moc rád viděl, kde přesně k dané chybě došlo. Pro tento případ aplikace nabízí možnost zaměření se pouze na úsek související s danou chybou, jak je lze vidět na obrázku 7.2b. Po stisknutí na uživatelem zvolenou buňku se mapa zaměří pouze na vybraný úsek. Po opakovaném stisknutí na buňku se na mapě opět zobrazí celá trasa.

Pro práci s mapou se v iOS využívá frameworku MapKit. Jedná se o jednoduché řešení, jak použít mapu světa ve své aplikaci. Na mapě je možné připínat body zájmu, vykreslovat trasu mezi dvěma body, nebo vkládat přes mapu námi zvolené tvary tzv. overlay. V případě potřeby je možné zobrazovat polohu uživatele například při snaze zobrazit nejbližší body zájmu v okolí.

V mém případě jsem se rozhodl použít MapKit z toho důvodu, aby uživatel dostal vizuální představu o to, kde se dané chyby dějí. S informací o úseku je pak lehčí vzpomenout si na to, jak se uživatel v daný okamžik choval. SwiftUI zatím neposkytuje MapKit, ale je možné využít protokolu `UIViewRepresentable`, který obalí prvek z UIKit a přidá mu vlastnosti, díky nímž se může ve SwiftUI použít. Funkce, které struktura splňující protokol musí obsahovat jsou `makeUIView(context: Context) -> MKMapView` vytvářející prvek rozhraní a `updateUIView(_uiView: MKMapView, context: UIViewRepresentableContext<MapView>)` upravující vzhled, když dostane prvek od SwiftUI informaci o tom, že by se měl překreslit.

Pro vykreslování trasy jsem využil `MKPolyline`, z níž jsem vytvořil `ColorPolyline`, která nese oproti `MKPolyline` informaci o barvě, kterou se má vykreslit. `ColorPolyline` jsem předal funkci `mapView(_mapView: MKMapView, rendererFor overlay: MKOverlay) -> MKOverlayRenderer`, která se postarala o vykreslení.

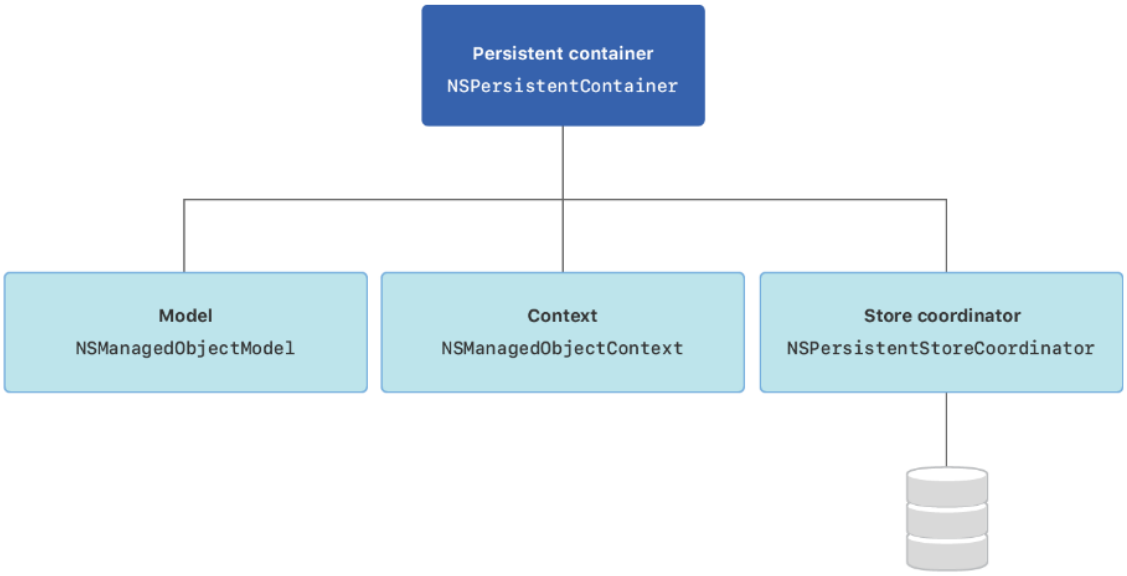
7.5 Ukládání dat

Způsoby, kterými lze uložit data v iOS, je velké množství. Jednou z kategorií je ukládání dat na zařízení. V úvahu přichází dvě varianty, a to CoreData a UserDefaults. Právě ukládání dat přímo v zařízení bylo vybráno při vytváření aplikace a bude dále popsáno níže. Dalšími způsoby jsou například CloudKit nebo SQLite, kteří ukládají data mimo zařízení.

7.5.1 CoreData

CoreData je jeden z nejpopulárnějších frameworků, který se stará o modelovou vrstvu aplikace. Umožňuje iOS aplikacím ukládat data na zařízení, sledovat je a načítat. Framework jako takový není databází. Pro trvalé uložení dat využívá SQLite databázi. Smyslem CoreData je správa objektového grafu, kde se stará o životní cyklus objektů (jejich vytváření, ukládání a načítání). Každý objekt má své atributy a vztahy k ostatním objektům v grafu. Pro jednodušší použití poskytuje Xcode možnost vytvoření souboru .xcdatamodeld, který ulehčuje práci s vytvářením grafu objektů.

Aby bylo možné objekty ukládat, je zapotřebí `CoreData stack` 7.3, který se skládá z komponent:



Obrázek 7.3: CoreData stack

- `NSManagedObjectModel` – reprezentuje strukturu instancí aplikace a vztahy mezi nimi,
- `NSManagedObjectContext` – sleduje změny instancí aplikace,
- `NSPersistentStoreCoordinator` – načítá a ukládá instance.

Jejich kombinací vzniká `NSPersistentContainer`.

Životní cyklus SwiftUI aplikace zatím neposkytuje automatické vytvoření objektů třídy `NSPersistentContaineru`. Proto je zapotřebí si `NSPersistentContainer` vytvořit sám. Kód pro implementaci je vidět ve výpise 7.3 a v aplikaci se zpřístupní všem

prvkům rozhraní následovně 7.4. Způsob, kterým se objekty do CoreData ukládají je ukázán ve výpise 7.5, kde `moc` značí `managedObjectContext`. Aplikace využívá CoreData na ukládání objektů, které jsou popsány v podkapitole 6.7.

```
import CoreData
struct PersistenceController {
    static let shared = PersistenceController()
    let container: NSPersistentContainer
    init() {
        container = NSPersistentContainer(name: "CarTripAnalytics")
        container.loadPersistentStores(completionHandler: {
    (storeDescription, error) in
            if let error = error as NSError? {
                fatalError("Unresolved error \(error),
                           \(error.userInfo)")
            }
        })
    }
}

```

Výpis 7.3: Implementace `NSPersistentContaineru`

```
@main
struct CarTripAnalyticsApp: App {
    let persistenceController = PersistenceController.shared

    var body: some Scene {
        WindowGroup {
            homeView()
                .environment(\.managedObjectContext,
                persistenceController.container.viewContext)
        }
    }
}

```

Výpis 7.4: Využití `NSPersistentContaineru` v aplikaci

```
let car = Car(context: self.moc)
car.name = self.carName
car.engineCap = Int16(self.carEngineCappacity) ?? 0
car.isActive = true

do {
    try self.moc.save()
}catch let nserror as NSError{
    print("ERROR: Coredata error \(nserror)")
}

```

Výpis 7.5: Uklání do CoreData

7.5.2 UserDefaults

UserDefaults využívá způsobu ukládání dat do souboru. Při načítání dat je načten celý soubor, čímž se tato varianta stává neoptimální pro větší objemy dat. Jedná se o ideální

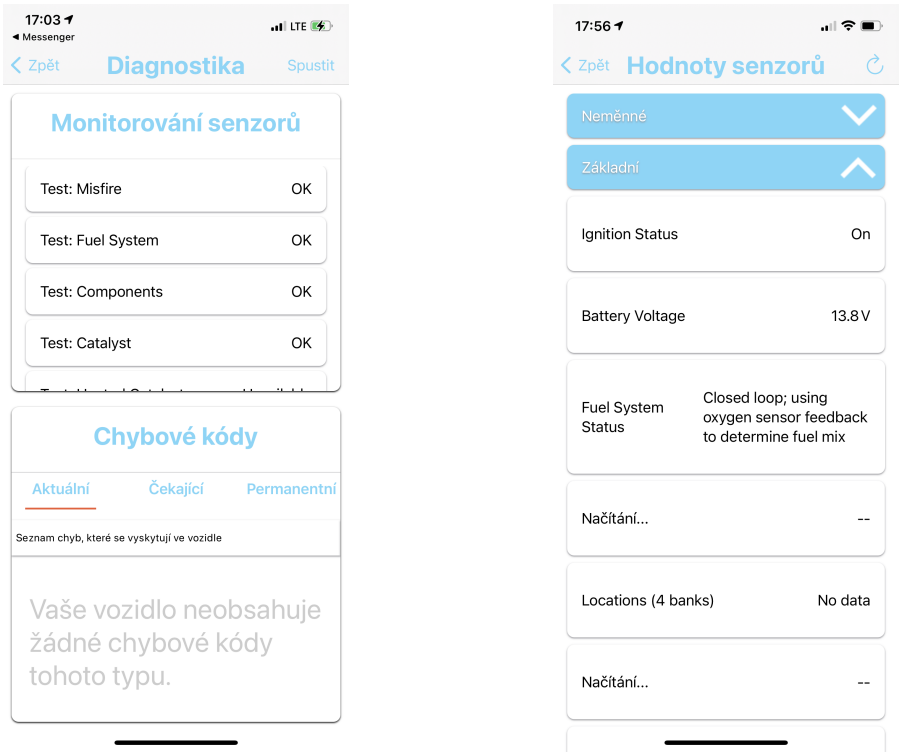
řešení, jak ukládat data jako například uživatelské nastavení aplikace. UserDefaults je založen na principu klíč-hodnota, kde hodnoty mohou nabývat všech základních datových typů. V aplikaci se do UserDefaults ukládá nastavení, zda mají být všechny sekce na obrazovce **Hodnoty senzorů** otevřeny, či naopak zavřeny.

7.6 Diagnostika

Diagnostikou se v aplikaci zabývají dvě obrazovky, které je možné vidět na obrázku 7.4. Na obrazovku s názvem „Diagnostika“ jsem umístil monitorování a diagnostické chybové kódy. Jsou realizovány pomocí `ScrollView`, kde chybové kódy nabízí `ScrollView` jak ve vertikálním, tak horizontálním směru. Human Interface Guidelines říká, že by se na jedné obrazovce měl objevovat jen jeden `ScrollView` v každém směru, což je v tomto případě porušeno. K porušení jsem se rozhodl z toho důvodu, protože počet výsledků monitorování a počet chybových kódů nikdy nedosáhne vysokých čísel, a proto je zbytečné vytvářet oddělené obrazovky. Získávání odpovědí ohledně diagnostických chyb fungovalo do poslední aktualizace iOS bez problému. Po aktualizaci se při zaslání požadavku na diagnostické chyby knihovna pro komunikaci zastaví a nelze získat další hodnoty z vozidla. Aktualizace, u které se projevil tento druh problému, byla nainstalována pár dní před odevzdáním práce a za takový krátký čas se nepovedlo chybu odstranit. Obrazovka „Hodnoty senzorů“ je vytvořena z líného vertikálního zásobníku (LazyVStack), který vykresluje pouze ty objekty, které jsou zrovna na obrazovce. Na základě toho, že se vykreslují jen viditelné prvky, jsem vytvořil způsob komunikace s vozidlem 7.6, který se ptá pouze na viditelná data. Dotaz na data se spustí ve chvíli, kdy je prvek vykreslen, nebo když už je vykreslen a uživatel zmáčkne na tlačítko v pravém horním rohu.

```
1     func updateRow() {
2         if let myPid = self.obd2.pids[name] {
3             self.obd2._obd2Adapter?.transmitCommand(myPid, responseHandler:
4             {_ in
5                 DispatchQueue.main.async {
6                     self.text = myPid.purpose
7                     self.detail = myPid.formattedResponse
8                 }
9             })
10        }
11    }
12    var body: some View {
13        HStack {
14            //Content
15        }
16        .onAppear {
17            self.updateRow()
18        }
19        .onChange(of: self.reload) {_ in
20            self.updateRow()
21        }
22    }
23
```

Výpis 7.6: Získání hodnot pro buňku seznamu



(a) Monitorování a DTC (b) Hodnoty senzorů

Obrázek 7.4: Diagnostická část aplikace

7.7 Kniha jízd

Kniha jízd je dokladem, který slouží pro zaznamenávání jízd prováděných vozidlem určenému k podnikání. Jedná se o průkazný materiál o tom, že se vozidlo opravdu používá k dosažení a udržení příjmů. V minulosti se využívala výhradně v papírové formě, ale dnes už existují i v softwarové podobě. Softwarová podoba nabízí výhody jako například přepočet kilometrů a různých dalších matematických výpočtů, díky nimž je práce s knihou jízd příjemnější.

7.7.1 Tvorba

Kniha jízd je nejčastěji znázorňována v podobě tabulky a měla by obsahovat:

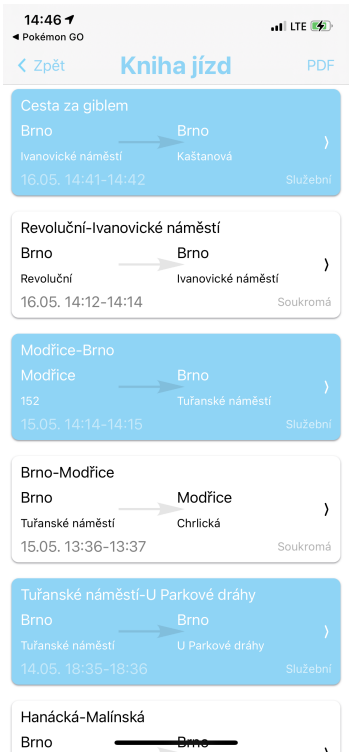
- datum jízdy,
- čas zahájení/ukončení jízdy,
- stav tachometru před a po jízdě,
- místo, kde jízda začala/skončila,
- počet ujetých kilometrů.

Pro tyto účely jsem se rozhodl seznam jízd převádět do tabulky ve formátu PDF. Na obrázku 7.5a je vidět příklad finálního vzhledu knihy jízd, která obsahuje všechny výše zmíněné informace. Pro vytvoření PDF souboru jsem využil knihovny PDFKit, pro kterou

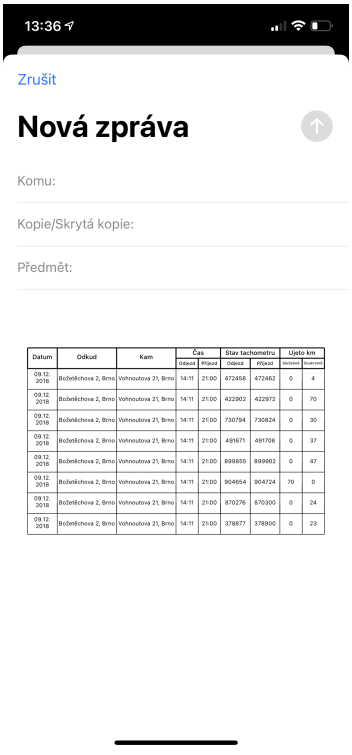
bylo stejně jako u MapKitu potřeba využít `UIViewRepresentable`. Pro vytvoření stylu tabulky a textu v ní jsem vytvořil třídu `PDFCreator`, konkrétně funkci `create()`, která má za úkol podle počtu dat vytvořit potřebný počet tabulek, kde každá tabulka má maximální velikost jedné stránky. Po vytvoření tabulky do ní vloží text a výsledek vrátí v datovém typu Data. Výstup z funkce `create()` se využívá pro zobrazení PDF dokumentu pomocí `PDFDocument(data: data)`.

7.7.2 Exportování

V případě zájmu nabízí aplikace možnost zaslání knihy jízd na e-mail. Jediné, co je potřeba, je zmáčknout „Exportovat“ v pravém horním rohu na obrazovce 7.5a obsahující ukázku knihy jízd, která bude uživateli zaslána. Po zmáčknutí tlačítka se uživateli zobrazí obrazovka 7.5b s možností vyplnění informací pro zaslání e-mailu s přílohou obsahující automaticky vytvořenou knihu jízd. Mail je po vyplnění odeslán z e-mailové adresy, kterou má uživatel připojenou ke svému Apple ID.



(a) Ukázka knihy jízd v aplikaci



(b) Obrazovka pro exportování knihy jízd na e-mail

Obrázek 7.5: Obrazovky zabývající se knihou jízd

7.8 Sledování dat

Po ukončení jízdy jsou všechna data sbíraná podobu jízdy uložena a nastává otázka způsobu vizualizace. Způsobů, jak uživateli předat tento typ informací, je mnoho. Aplikace konkrétně poskytuje dva způsoby.

Prvním z nich je grafová reprezentace, kterou je možné vidět na obrázku 7.6a. Jedná se o volně dostupný modul, který se nazývá ChartsView-SwiftUI[5]. Modul bylo nutné mírně upravit, protože se při testování ukázaly chyby, které vedly k pádu aplikace. Okno určující délku trasy je možné libovolně zmenšovat, či zvětšovat. Flexibilita okna umožňuje náhled na vývoj hodnot v průběhu celé jízdy a zároveň i na specifické úseky. Do grafu je možné zakreslovat i více hodnot proměnných ve stejný čas, což je možné vidět na obrázku 7.6b. Vše, co je potřeba, je zaškrtnutí proměnné ve spodní části obrazovky, jenž obsahuje `ScrollView`.

Druhou variantou je seznam. Seznam se v aplikaci využívá v části s detailním pohledem na jízdu, do které je možné se dostat přes seznam uložených jízd. Názornou ukázku takového seznamu je možné vidět na obrázku 7.6c. Seznam obsahuje tolik řádků, kolik se z vozidla sbírá hodnot v průběhu snímání. Každý řádek obsahuje název sbírané proměnné a její maximální, minimální a průměrnou hodnotu. Jedná se o způsob, který dá uživateli méně informací, ale na druhou stranu se jedná o informace, které by složitě získával z grafu (například průměrnou hodnotu).

(a) Graf s jednou proměnnou (b) Graf s více proměnnými (c) Data v podobě seznamu

Obrázek 7.6: Způsob předávání informací v aplikaci

[5]Dostupný na: https://github.com/BestKora/ChartsView-SwiftUI

Kapitola 8

Testování

V následující kapitole se píše o testování aplikace. Mobilním zařízením, na kterém byla aplikace testována, byl iPhone 12. Vozidlo, které sloužilo pro testování, byla Škoda Fabie s automatickou převodovkou. V kapitole se nejdříve zaměřuji na testování v průběhu vývoje aplikace. Následně se soustředím na testování prvků uživatelského rozhraní a v poslední řadě na testování uživateli.

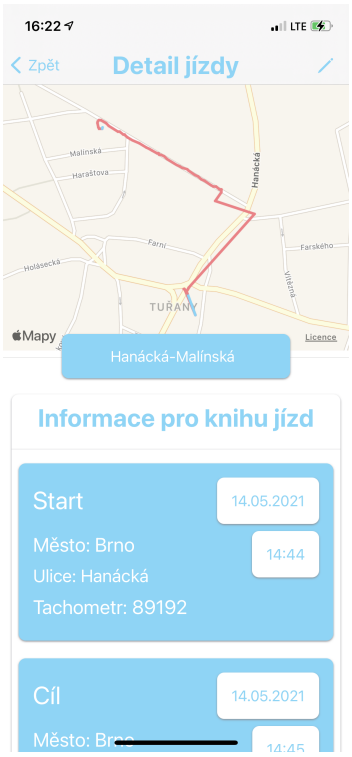
8.1 Průběžné testování

Průběžné testování aplikace je jednou z nejdůležitějších praktik vývoje softwaru. Z toho důvodu byla testována každá menší změna, která mohla ovlivnit chod aplikace. Co se týče prvků uživatelského rozhraní, poskytuje Xcode ve spojení s SwiftUI náhled na výsledný vzhled aplikace. Problém nastával u komplikovanějších obrazovek obsahujících například mapu a při snaze otestovat fungování získávání dat z vozidla. Pro některé složité případy bylo nutné využít mobilního zařízení , někdy přímo ve vozidle se zapnutým zapalováním.

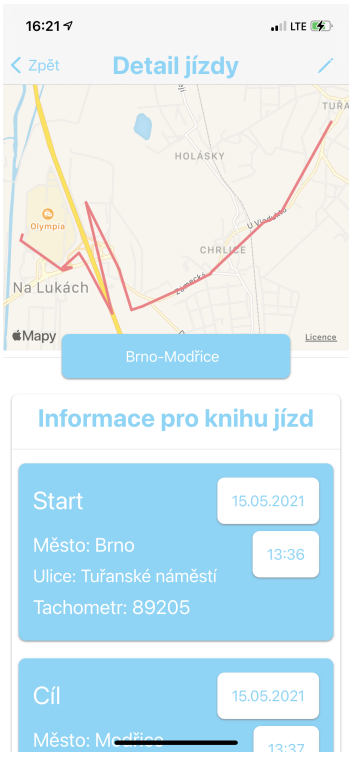
Po vytvoření základního vzhledu aplikace, který je možné vidět v příloze A, proběhlo první větší testování ve vozidle. Testovalo se získávání dat a zjišťoval jsem, jaká data je reálné z vozidla získat. Informace o získatelných hodnotách jsem si poznamenal pro následnou analýzu. Výsledný způsob analýzy, který byl popsán v podkapitole 7.4, byl testován a došlo se k závěru, že vozidla s automatickou převodovkou využívají jiného způsobu otevírání škrticí klapky, než vozidla s převodovkou manuální, a proto není možné chyby spojené s klapkou brát vážně.

Dalším z testovaných byl vzorec na výpočet přibližné průměrné spotřeby. Výstupy byly vždy porovnávány s informací na palubní desce vozidla. Při využití vzorce 6.1 jsem bohužel nedošel k hodnotám, které by byly podobné hodnotám na palubní desce. Za to u vzorce 6.2 jsem se dočkal překvapivě dobrých výsledků, které byly přesnější, čím delší byla snímaná jízda.

Co se týče snímání lokace uživatele při jízdě, bylo využito nejdříve nastavení `kCLLocationAccuracyBest`, které má poskytovat nejlepší přesnost. Přesnost nastavení při jízdě pod 50km/h je možné vidět na obrázku 8.1a a při jízdě nad 50km/h na obrázku 8.1b. Správná lokalizace uživatele je jednou z nejdůležitějších prvků aplikace, a proto bylo nakonec vyžito nastavení `kCLLocationAccuracyBestForNavigation`, které využívá dalších senzorů (jako například akcelerometr) k lepší přesnosti i při vyšších rychlostech. Výslednou podobu trasy s využitím `kCLLocationAccuracyBestForNavigation` je možné vidět na obrázku 7.1c.



(a) Výsledek trasy celé pod 50km/h



(b) Výsledek trasy s úseky přes 50km/h

Obrázek 8.1: Výsledky s využitím `kCLLocationAccuracyBest`

8.2 Vzhled uživatelského rozhraní

Uživatelské rozhraní je to hlavním, co uživatel zprvu vidí, a je tím pádem nezbytné, aby se věnovala velká část času pro jeho vývoj. Testování zahrnovalo rozmístění prvků na obrazovce a výběr barevného spektra, které z velké část rozhodují o tom, zda uživatel bude aplikaci používat. Volil jsem pastelové barvy, které jsou obecně velice příjemné na pohled. V průběhu vývoje jsem se rozhodl využít zaoblených obdélníků pro oddělení částí obrazovky pro lepší přehled o tom, kde mají prvky své hranice. Všechny úpravy rozhraní probíhaly v Xcode za pomoci náhledu, který zprostředkovává změny v reálném čase, a tím urychluje vývoj rozhraní.

8.3 Testování uživateli

Jakmile byla práce hotova, přišlo na řadu testování uživateli. Z důvodu absence mikrokontroleru u všech dotázaných na testování jsem se rozhodl pro způsob testování prostřednictvím snímků z aplikace a videí. Uživatelům byly předány snímky všech obrazovek aplikace a video, které sloužilo pro hodnocení funkčnosti. Aplikace se hodnotila ve čtyřech kategoriích: design, intuitivnost, srozumitelnost a využití. Testování se účastnilo šest osob, které byly zařazeny do tří skupin a to začátečník, běžný řidič, zkušený řidič. V tabulce 8.1 je možné vidět výsledky testování jednotlivých respondentů. Nejmladší z dotázaných měl 21 let a nejstarší 51. Čtyři lidé měli dlouholeté zkušenosti s používáním Apple zařízení a zbylí dva neměli s používáním iOS zařízení skoro žádné zkušenosti.

Prvky testování	Sabina	Monika	Marek	Adam	Zdeněk	František
Design	6/10	7/10	7/10	9/10	10/10	7/10
Uspořádání prvků	9/10	10/10	10/10	9/10	10/10	9/10
Intuitivnost	10/10	10/10	10/10	7/10	10/10	7/10
Srozumitelnost	10/10	9/10	9/10	8/10	10/10	8/10
Využití	5/10	9/10	10/10	6/10	6/10	7/10

Tabulka 8.1: Výsledky testování uživateli

První z dotázaných byla řidička Sabina, která řidičský průkaz nevlastní ještě ani rok, a proto jsem ji zařadil mezi řidiče začátečníky. Její hodnocení designu aplikace bylo z velké části ovlivněno názorem, že vybraná modrá barva není dostatečně tmavá na to, aby na ní šel dostatečně vidět bílý text. K ostatním designovým prvkům aplikace neměla výhrady. K uspořádání prvků v aplikaci neměla žádné připomínky a stejně tak tomu bylo u intuitivnosti aplikace. Všechny prvky aplikace byly předvídatelné. Díky nadpisům obrazovek bylo prý srozumitelné, co obrazovka nabízí. Co se týče otázky, zda by aplikaci při svých jízdách používala odpověděla, že spíše ne, protože ji informace z vozidla tolik nezajímají.

Druhou řidičkou byla Monika, která vlastní řidičský průkaz již více jak 30 let. Ačkoli má spoustu zkušeností s řízením vozidla, tak se v nich tolik nevyzná, a proto jsem ji zařadil do kategorie běžný řidič. Stejně jako u Sabiny byla, co se týče designu, vytknuta horší čitelnost textu na modrém pozadí. K uspořádání a intuitivnosti neměla výhrady a velice ji zaujala možnost exportování knihy jízd přímo z aplikace. Co se týče využití, tak by aplikaci moc ráda vyzkoušela v běžném provozu a poté se rozhodla, zda by aplikaci dále využívala.

Prvním z dotázaných řidičů byl Marek, který má stejně jako Monika plno zkušeností za volantem. Oproti Monice se navíc vyzná skvěle v tom, co má vozidlo pod kapotou. Opět byla zmíněna příliš světlá modrá barva, ale jinak se design aplikace velice líbil. Uspořádání prvků, intuitivnost a srozumitelnost byly takřka bez výhrad. Jedná se o jediného řidiče, který by aplikaci moc rád používal na denní bázi.

Druhým z řidičů je Adam, který sice řidičský průkaz má, ale tak často nejezdí. Adam je jedním ze dvou dotázaných, kteří nemají moc zkušeností s iOS zařízením. K designu jako jeden z mála neměl skoro žádné připomínky. Jediné, na co se Adam zeptal, byl význam kombinování více hodnot v grafu vývoje hodnot, který mu byl vzápětí vysvětlen. Z důvodu, že se jedná o řidiče, který tak často nejezdí, bylo zhodnoceno využití aplikace nižší hodnotou, přesně šest.

Dalším dotázaným byl Zdeněk, který se zařazuje do kategorie běžných řidičů. Zkušeností s Apple zařízeními má dost a zřejmě i z toho důvodu neměl jediný problém s ovládáním a srozumitelností aplikace. K designu neměl jedinou připomínku a jako jediný ohodnotil design aplikace nejvyšší možnou hodnotou. Stejně jako Adam nejezdí tak často, aby měl potřebu využívat aplikaci na denní bázi, a proto ohodnotil využitelnost nižší hodnotou.

Jako poslední, kdo se účastnil testování, byl František. František je řazen mezi běžné řidiče a stejně jako Adam nemá zkušenosti s Apple zařízením. Z toho důvodů byla srozumitelnost a intuitivnost aplikace ohodnocena hodnotou osm a sedm. I František byl názoru, že by se měla použít tmavší modrá. Aplikaci zhodnotil jako velice zajímavou, ale nevyužil by všechno, co aplikace nabízí.

Kapitola 9

Závěr

Cílem diplomové práce bylo vytvořit aplikaci, která bude schopna komunikovat s vozidlem a předávat uživateli zajímavé informace, které budou vycházet z analýzy dat vozidla. Na začátku bylo v plánu implementovat klasifikátor pomocí neuronové sítě, jenž bude klasifikovat jízdu podle toho, do jaké míry byla ekologická. Návrh byl nakonec přepracován hlavně z důvodu nedostatečného popisu trénovacích dat a jejich celkového nedostatku.

Výsledná aplikace se zaměřuje na detekci úseků, ve kterých uživatelova jízda vystupuje z mezí definující normální jízdu. Může sloužit také jako černá skřínka v případě nehody, kdy podstatně usnadní vyšetřování podobně jako to umí Tesla[1]. Po ukončení snímání jízdy je jízda uložena a uživatel dostane informace o vývoji hodnot vozidla a seznamu chyb, které se podobu jízdy vyskytly. Poskytuje hlavní diagnostické funkce, které je možné najít ve všech diagnostických aplikacích, aby měl uživatel možnost si vozidlo po jízdě vždy zkontrolovat. Všechny uložené jízdy je možné exportovat v podobě knihy jízd, která je takřka nutností například pro firmy využívající vozidla pro vykonávání práce.

Cílovou skupinou aplikace by měli být běžní uživatelé, kteří mají zájem o informace z jejich automobilu a informace o tom, kde dochází při jejich jízdě k chybám. Aplikace by měla vynikat přehledností a srozumitelností, aby ji mohlo používat širší spektrum uživatelů. Využití vidím například ve firmách, které pro svůj chod využívají vozidla a je tedy potřeba vést knihu jízd. Aplikace řidičům ušetří čas při vyplňování a zaměstnavatel bude mít jistotu, že zaměstnanci vyplňují pole správně. Další uplatnění vidím v autoškole, kde si instruktor s žákem mohou rozebrat absolvovanou jízdu zpětně.

Aplikace nabízí spoustu možností pro rozšíření. Například identifikaci uživatele, který absolvoval právě snímanou jízdu. Jedná se o funkci využívající neuronové sítě, konkrétně rekurentní neuronové sítě LSTM. Výsledná architektura obsahovala dvě vrstvy, kde v první vrstvě bylo 128 a v druhé 256 obousměrných LSTM. Síť jsem učil za pomoci Tripletloss funkce. Z důvodu nedostatku volně dostupných dat a času, tato funkce nebyla v aplikaci využita, ale pracovalo se na ní. Na obrázku v příloze D.1 je možné vidět graf znázorňující tři jezdce a jejich úseky jízdy. Střed třídy jezdce je vždy počítán z předešlých jízd a v grafu je znázorněn růžovou barvou pro všechny tři třídy. Jak je možné vidět, součet vzdáleností bodů od jednotlivých středů je vždy nejbližší pro střed jezdce, kterému patří daná jízda. Díky tomu aplikace ze znalosti středů jezdců dokáže spočítat, ke kterému daná jízda nejvíce sedí. V případě neshody s právě aktivním uživatelem se může zeptat, zda uživatel nezapomněl přepnout na jiného. Pomocí rozšíření bude docházet při větším množství uživatelů na jednom zařízení k méně problémům se špatným zařazením jízdy.

[1] Nehoda vozidla Tesla: https://www.bbc.com/news/technology-57072778

Dalším rozšířením může být možnost porovnat dvě jízdy, které vedly stejnou trasou a ukázat na nich rozdíly v hodnotách. Díky porovnání jízd může uživatel sledovat své pokroky v jízdě. Po drobných úpravách aplikace by bylo možné sdílet data s pojišťovnou, která by následně mohla určovat výši pojistného. Řidiče by byl veden k bezpečnější jízdě díky níž by platil nižší pojistné.

Literatura

[1] 101, C. *ODB-II Connector*. červenec 2018. [Online; navštíveno 11.1.2021]. Dostupné z: https://components101.com/connectors/obd2.

[2] B.A.D.S. *OBD2 Codes and Meanings*. Květen 2020. [Online; navštíveno 9.1.2021]. Dostupné z: https://bads.lt/en/obd2-codes-and-meanings-2/.

[3] Bohon, C. *Apple's Swift programming language: Cheat sheet*. Září 2020. [Online; navštíveno 1.5.2021]. Dostupné z: https://www.techrepublic.com/article/apples-swift-programming-language-the-smart-persons-guide/.

[4] cars.com. *Throttle Position Sensor*. Září 2015. [Online; navštíveno 10.1.2021]. Dostupné z: https://www.cars.com/auto-repair/glossary/throttle-position-sensor/.

[5] Center, C. K. *What is an Engine Control Module (ECM)?* Listopad 2018. [Online; navštíveno 10.1.2021]. Dostupné z: https://www.fixdapp.com/blog/engine-control-module.

[6] Chen, S.-H., Pan, J.-S. a Lu, K. Driving Behavior Analysis Based on Vehicle OBD Information and AdaBoost Algorithms. *Lecture Notes in Engineering and Computer Science*. Březen 2015, sv. 1, s. 102–106.

[7] Doctor, O. A. *Dark mode now available for iOS and Android*. Březen 2020. [Online; navštíveno 12.1.2021]. Dostupné z: https://www.obdautodoctor.com/scantool-garage/mobile.

[8] EFIgnition. *Engine speed sensor*. říjen 2020. [Online; navštíveno 9.1.2021]. Dostupné z: https://www.efignition.com/c-3632494/engine-speed-sensor/.

[9] ElmElectronics. *The ELM327 data sheet*. Leden 2017. [Online; navštíveno 7.1.2021]. Dostupné z: https://www.elmelectronics.com/wp-content/uploads/2017/01/ELM327DS.pdf.

[10] GOFAR. *WHAT IS AN OBD SCANNER?* [Online; navštíveno 13.1.2021]. Dostupné z: https://www.gofar.co/blog/what-is-an-obd-scanner/.

[11] Gordon, J. Look who's talking: OBD III is on the way and offers many new ways to communicate with and diagnose vehicles. *Motor Age*. Advanstar Communications, Inc. 2006, sv. 125, č. 9, s. 30. ISSN 1520-9385.

[12] Hawkins, I. *Torque Pro (OBD 2 & Car)*. Září 2020. [Online; navštíveno 12.1.2021]. Dostupné z: https://play.google.com/store/apps/details?id=org.prowl.torque.

[13] *Road vehicles — Communication between vehicle and external equipment for emmissions-related diagnostics*. Standard. International Organization for Standardization, leden 2006.

[14] KINGSTON, L. *What is an Electronic Control Unit? PH Explains*. Březen 2018. [Online; navštíveno 10.1.2021]. Dostupné z: https://www.pistonheads.com/news/ph-features/what-is-an-electronic-control-unit-ph-explains/37771.

[15] LEEN, G. a HEFFERNAN, D. Expanding automotive electronic systems. *Computer*. IEEE. 2002, sv. 35, č. 1, s. 88–93. ISSN 0018-9162.

[16] MARTIN, J. *ECU Diagnostics – part 7 : ECU Maps and Mapping*. Září 2019. [Online; navštíveno 13.1.2021]. Dostupné z: https://www.purplemeanie.co.uk/index.php/2019/09/12/ecu-diagnostics-part-7-ecu-maps-and-mapping/.

[17] MESEGUER, J., CALAFATE, C., CANO, J.-C. a MANZONI, P. Assessing the impact of driving behavior on instantaneous fuel consumption. Červenec 2015, s. 443–448. DOI: 10.1109/CCNC.2015.7158016.

[18] NOVINY, D. *Telematika v autech vede k vyšší bezpečnosti řidičů i úsporám*. červenec 2015. [Online; navštíveno 13.1.2021]. Dostupné z: https://www.dnoviny.cz/silnicni-doprava/pruzkum-telematika-v-autech-vede-k-vyssi-bezpecnosti-ridicu-i-usporam.

[19] OBDINNOVATIONS. *On-Board Diagnostics (OBD) Background History*. říjen 2018. [Online; navštíveno 9.1.2021]. Dostupné z: https://obdinnovations.freshdesk.com/support/solutions/articles/36000019429-on-board-diagnostics-obd-background-history.

[20] RENSHAW, J. *What Is an O2 Sensor?* červen 2020. [Online; navštíveno 10.1.2021]. Dostupné z: https://shop.advanceautoparts.com/r/advice/cars-101/what-is-an-o2-sensor.

[21] V, H. S. Understanding OBD II. In: *Computerized Engine Controls*. 9th Edition. Cengage Learning, Inc, 2012, s. 1–1. ISBN 9781111134907. Dostupné z: https://app.knovel.com/hotlink/pdf/rcid:kpCECE0001/id:kt00B8Y8E9/computerized-engine-controls/understanding-obd-ii?kpromoter=Summon.

Příloha A

Prvotní návrh aplikace

Obrázek A.1: Aplikace, která umí komunikovat s automobilem a reprezentovat jeho data

Příloha B

Vizuální návrh dalších částí aplikace

(a) Obrazovka při zapnutém snímání data za jízdy

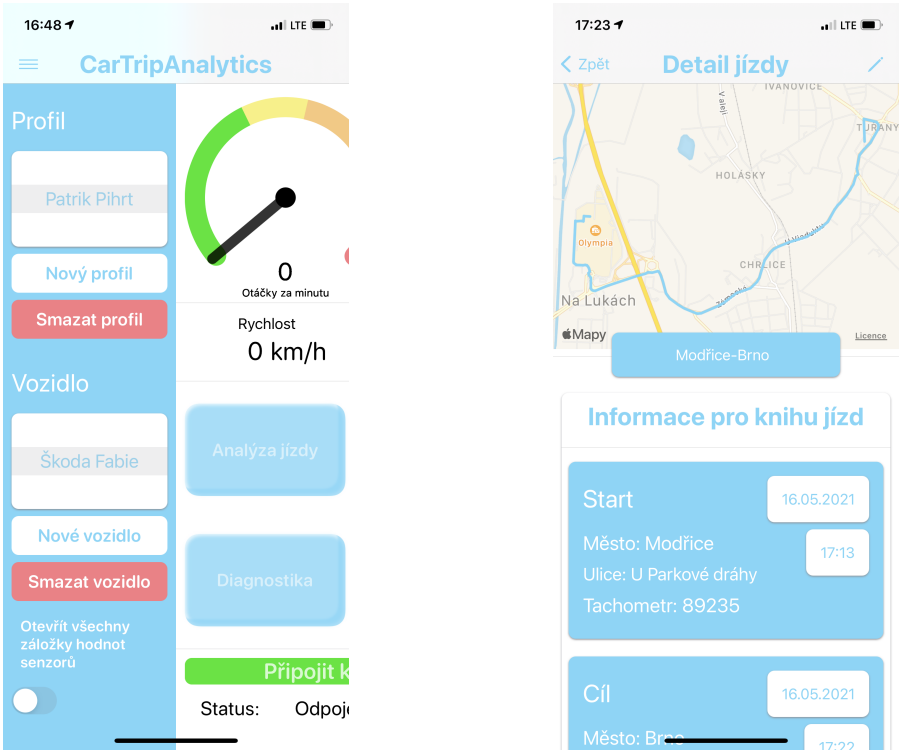
(b) Obrazovka vyhodnocení jízdy

(c) Způsob reprezentace dat po jízdě

Obrázek B.1: Návrh obrazovek

Příloha C

Finální vzhled aplikace

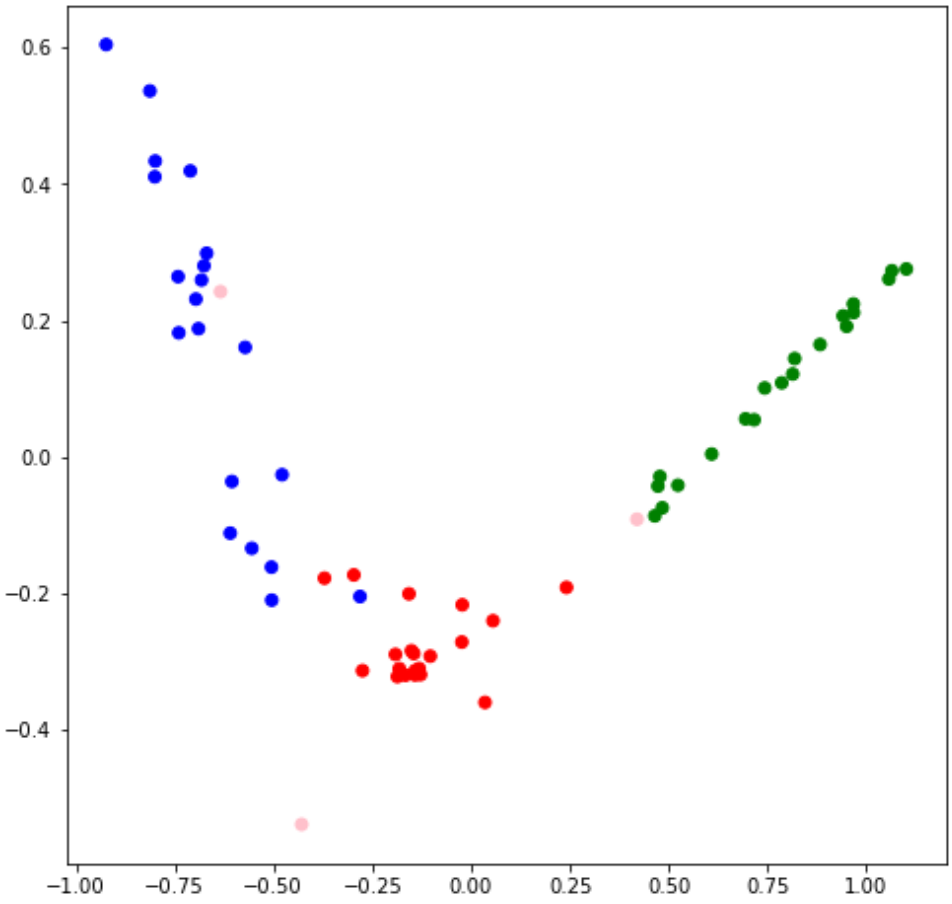


(a) Boční menu (b) Uložená jízda

Obrázek C.1: Další obrazovky aplikace

Příloha D

Možná rozšíření



Obrázek D.1: Převedení vektoru o velikosti 256 do 2D prostoru prostřednictvím analýzy hlavních komponent (PCA)

Příloha E

Plakát

Aplikace pro extrakci a analýzu jízdních dat z OBD-II na iOS

- Analýza dat jízdy
  (hodnocení jízdy, chybové úseky)
- Kniha jízd
  (historie jízd, export do PDF)
- Diagnostika vozidla
  (monitorování, chyby a hodnoty senzorů)
- Reprezentace dat
  (Charts, ChartsView-SwiftUI)

- Jazyk Swift
- Vývojové prostředí Xcode
- SwiftUI, CoreData
- LTSupportAutomotive

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DIPLOMOVÁ PRÁCE

AUTOR: Bc. Patrik Pihrt

VEDOUCÍ: Ing. Petr Bobák

Obrázek E.1: Plakát aplikace