



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

Autó fedélzeti rendszer

OE-NIK
2021

Hallgató neve:
Hallgató törzskönyvi száma:

Korcsák Gergely
T/006263/FI12904/N

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Korcsák Gergely**
Törzskönyvi száma: T/006263/FI12904/N
Neptun kódja: 

A dolgozat címe:

Autó fedélzeti rendszer
Car on board computer

Intézményi konzulens: Lovas István
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Számítógép architektúrák
IoT, beágyazott rendszerek és robotika
specializáció

A feladat

Tervezzon és valósítson meg egy autó fedélzeti számítógépet, amely képes nyomon követni a gépjármű valós idejű helyzetét, teljesítményét, üzemanyagfogyasztását és egyéb számottevő adatait. A begyűjtött információkat tárolja és egy kardinális részét valós időben grafikus interfész segítségével jelenítse meg. Utazás közben továbbá vizuális formában jelenítse meg az autó vezetőjének aktuális vezetési stílusát és eközben nyújtson lehetőséget hang alapú médiatartalom lejátszására is. Az autó leállításkor a rendszer nyújtson visszajelzést az utazás információiról és egyéb gesztus hiányában kapcsoljon ki energiatakarékosság céljából.

A dolgozatnak tartalmaznia kell:

- a megoldandó feladat pontos leírását és részletes elemzését,
- a meglévő rendszerek vizsgálatának elemzését,
- az alkalmazni kívánt technológiák elemzését,
- a rendszertervet, döntések indoklásait,
- a megvalósítás leírását,
- a tesztelési tervet és a tesztek eredményeit.



Fleiner R

Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

[Signature]
.....
intézményi konzulens



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2021. 12. 13.

Korcsák Gergely

hallgató aláírása



KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:
Korcsák Gergely Nappali.....

Telefon Levelezési cím (pl: lakcím):

Szakdolgozat / Diplomamunka címe magyarul:

Autó fedélzeti rendszer.....

Szakdolgozat / Diplomamunka címe angolul:

Car on board computer

Intézményi konzulens: Külső konzulens:

Lovas István

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.09.08	Témaválasztási konzultáció	
2.	2021.09.14	Szakirodalom áttekintése	
3.	2021.09.28	Lehetséges megvalósítás áttekintése	
4.	2021.10.05	Tartalmi, formai ellenőrzés	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4 tantárgy követelményét teljesítette, beszámolóra / védésre bocsátható.

Budapest, 2021.12.06

.....
Intézményi konzulens



KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Korcsák Gergely Nappali

Telefon: Levelezési cím (pl: lakcím):

Szakdolgozat / Diplomamunka címe magyarul:

Autó fedélzeti rendszer

Szakdolgozat / Diplomamunka címe angolul:

Car on board computer

Intézményi konzulens: Külső konzulens:

Lovas István

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.10.20	Specifikáció áttekintése	
2.	2021.11.03	Rendszerterv áttekintése	
3.	2021.11.18	Fejlesztés, tesztelés ellenőrzés	
4.	2021.12.08	Tartalmi, formai ellenőrzés	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4 tantárgy követelményét teljesítette, beszámolóra / védésre bocsátható.

Budapest, 2021.12.09

.....
Intézményi konzulens

Tartalomjegyzék

1.	Bevezetés.....	4
2.	Az irodalom alapján a lehetséges megközelítési módok és megoldások áttekintése és elemzése	5
2.1.	Raspberry Pi okos autó.....	5
2.1.1.	Tolatókamera	5
2.1.2.	Összekapcsolás OBD diagnosztikával.....	6
2.1.3.	Grafikus interfész.....	7
2.1.4.	GPS hozzáadása	8
2.1.5.	Elemzés	8
2.2.	M3 Pi – Raspberry Pi OBD-II Érintőképernyős fedélzeti számítógép	9
2.2.1.	Dizájn.....	9
2.2.2.	Fejlesztés.....	10
2.2.3.	Elemzés	13
2.3.	OBD-Pi Feketedoboz [5].....	13
2.3.1.	OBD2	14
2.3.2.	PyOBD.....	14
2.3.3.	Autóba építés	14
2.4.	GPX Fájl Formátum.....	15
2.4.1.	Mi is az a GPX formátum	15
2.4.2.	Adattípusok	15
2.4.3.	Mértékegységek	16
2.4.4.	GeoJSON	16
2.5.	Angular GPS Tevékenység napló [8] [9]	16
2.5.1.	Térkép és leaflet.....	16
2.5.2.	Elemzés	17
3.	A megoldási módszer kiválasztása, a választás indoklása	18
4.	A részletes specifikáció leírása.....	19
4.1.	Hardveres	19
4.1.1.	Áramellátás és kikapcsolás	20
4.2.	Szoftveres Grafika.....	21
4.2.1.	Keret.....	21
4.2.2.	Főképernyő	22
4.2.3.	Zenelejátszó	22

4.2.4.	Vezetési összegzés	23
4.2.5.	Beállítások	23
4.3.	Szoftveres Logika.....	24
4.3.1.	OBD2-es adatfeldolgozás	24
4.3.2.	Vezetés értékelés meghatározása.....	24
4.3.3.	Sebességváltó fokozat meghatározása	25
4.3.4.	Térkép	26
4.3.5.	Zenelejátszó	26
5.	A tervezés során végzett munkafázisok és tapasztalataik leírása	27
5.1.	Hardverelemek kiválasztásával kapcsolatos tapasztalatok	27
5.1.1.	Vezérlő egység.....	27
5.1.2.	Kijelző.....	27
5.1.3.	GPS modul.....	27
5.1.4.	OBD modul.....	27
5.1.5.	File elérés	28
5.1.6.	Energia ellátás.....	28
5.2.	Az Operációs rendszer kiválasztása	28
5.3.	Nyelv és a Fejlesztőkörnyezet kiválasztása	29
5.3.1.	Python	29
5.3.2.	Qt	29
5.3.3.	Windows 10 IoT	29
5.3.4.	Angular - Electron	30
5.4.	Fejlesztési folyamat.....	31
5.4.1.	Felépítés	31
5.4.2.	Beállítások	31
5.4.3.	Szolgáltatás elérés	32
5.4.4.	OBD2 Lekérdezés és kiíratás.....	32
5.4.5.	GPS helyadat elérés, mentés és kiértékelés	33
6.	A megvalósítás leírása	34
6.1.	Főképernyő.....	34
6.1.1.	OBD2-es adatok.....	34
6.1.2.	Térkép	35
6.2.	Keret.....	35
6.2.1.	Fejléc.....	35
6.2.2.	Lábléc.....	35

6.3.	Zenelejátszó.....	36
6.3.1.	Kezelőfelület.....	36
6.3.2.	Zeneszámok	37
6.3.3.	Lejátszási lista.....	37
6.4.	Beállítások.....	38
6.5.	Összegzés	38
6.6.	Áramellátás, kikapcsolás.....	39
6.7.	Fizikai beépítés.....	39
7.	Tesztelés	41
7.1.	GUI tesztelés	41
7.2.	Logikai tesztelés.....	42
7.3.	Fizikai tesztelés	42
7.3.1.	Működés igazolása.....	42
7.3.2.	Áramellátás tesztelése.....	42
7.3.3.	Végső teszt.....	43
8.	Az eredmények bemutatása, értékelése, hasonló rendszerek eredményeivel összevetése.....	46
9.	A megvalósítás elemzése, alkalmazásának és továbbfejlesztési lehetőségeinek számbavétele.....	47
10.	Tartalmi összefoglaló.....	48
11.	Content Summary	49
12.	Szójegyzék	50
13.	Irodalomjegyzék	52
14.	Ábrajegyzék.....	54
15.	Táblázatjegyzék	55

1. BEVEZETÉS

Napjainkban, mint ahogy a mobiltelefonjaink is egyre több mindenhez értenek, a személyautóinkból sem maradhatnak ki az okos funkciók. Egyre nagyobb a kereslet az autókba gyártáskor beépített extrának számító és utólag beszerelhető fedélzeti számítógépekre. Habár utángyártott formában ezt régebbi autókba szokás beépíteni a rádió helyére, viszont nagy kényelmet adhat amennyiben képesek vagyunk az árát megfizetni.

Legtöbb esetben egy mostani fedélzeti számítógépnek a funkciói közé kell, hogy tartozzon, a rádió és média tartalom lejátszása is, illetve valamilyen beépített navigációs rendszer megvalósítása. Ezen felül, mára szinte elengedhetetlen valamilyen elemzési vezérlő beépítése az eszközbe. Tehát vezetőként egyre jobban érdekelhet minket, hogy hogyan is vezetünk, milyennek számítunk és mennyi az energia- és/vagy üzemanyag-fogyasztásunk. Ezen szolgáltatásokra, nincsen feltétlenül szükségünk, és ha nem lennének, nem is foglalkoznának a számolgatásaikkal, de megjelenésükkor a társadalom mégis igényt mutatott a használatukra és sikerült megszeretnie az e féle prémium eszközöket.

Gondoljunk csak bele mennyi az átlagfogyasztásunk és mennyit költünk benzinre. Ennek a számításával sokan nem foglalkoznak, és nem is érdekel mindenkit, hogy mennyit fogyaszt az autója, hiszen így is úgyis tankolni kell és csak a kiadásokat nézzük, hogy finanszírozzuk-e, de ugyanez nagyvonalakban megfigyelhető pénzköltési szokásainkra is. Természetesen vannak, akik feljegyzik és figyelemmel kísérik a számláikat, kiadásait, de újabban a kártyával történő kiadásainkat, amire kapjuk a fizetésünket is, mind vezeti a bank helyettünk és ezeket kényelmesen egy alkalmazásból megtekinthetjük. Az, hogy ezeket a számításokat nem nekünk kell elvégezni, igényt is nyújthat az eredményük megismerésére. Tehát igényt tarthatunk olyan szolgáltatásokra, amiket akár csak lustaságból, vagy figyelmetlenségből nem akartunk mi magunk elvégezni, de manapság szinte már a világ is egy az életünket megkönnyítő irány felé halad.

Ezért ebben a dolgozatban egy fedélzeti számítógép elkészítése a cél, amely lekérdezi az autónk valós idejű vezetési adatait, nyomon követi annak a helyzetét, és az út végén ezeket az adatokat összesítve, elemezve kijelzi a számunkra. Ezen felül, mivel ezt a rendszert a rádió helyére szeretném beépíteni, ezért fontosnak tartom, hogy legyen legalább egy féle zene lejátszási lehetőség is. Az elkészült rendszer az 1996-tól széleskörűen használt OBD2-es szabványos csatlakozón keresztül fog kommunikálni az autóval és áramellátása, egyéb bekötései is a további szabványos csatlakozók felhasználásával valósul meg, a könnyebb beszerelés érdekében. [1] A rendszer elkészítésénél figyelmet kap az is, hogy egy rádió helyére egy 2 DIN-es foglalatba is beférjen, és használata vezetés közben is könnyű legyen, hiszen az ilyen eszközöket vezetés közben használjuk.

2. AZ IRODALOM ALAPJÁN A LEHETSÉGES MEGKÖZELÍTÉSI MÓDOK ÉS MEGOLDÁSOK ÁTTEKINTÉSE ÉS ELEMZÉSE

Nem meglepő módon, nem én vagyok az egyetlen, akinek ilyen, vagy hasonló dolog már az eszébe jutott. Sokan szeretnék feldobni, modernebbé tenni régebbi gyártású gépjárműveiket és erre elektronikai tudásukat is felhasználják. Ezért az alábbiakban pillantást veszek egy pár hasonló megvalósítási esetre, hogy ezek majd kiindulási alapot adhassanak számomra a megvalósításban.

2.1. Raspberry Pi okos autó

Ebben a megközelítésben Tinkernut projektjét találtam meg, amiben egy fedélzeti számítógépet Raspberry Pi segítségével valósított meg. [1] Projektjével azt szeretne volna elérni, hogy autója „okosabb” legyen, és újabb funkciókkal gazdagodjon egy új autóra való beruházás nélkül.

Értelmezése szerint a mai világban az „Okos autó” alatt mindenki más-más fogalmat érthet. Ezért megközelítésében annak a következőszempontok számítanak:

- Érintő kijelző
- Tolató kamera, amely figyelmeztet, ha túl közel érünk valamihez
- Általános vezetési információk, mint üzemanyaggazdaságosság
- Esetleges Bluetooth médiakapcsolat

2.1.1. Tolatókamera

A legelső gondolat lehet, amit az autónkba szerelnénk, az egy tolatókamera. Ugyan rengeteg féle megvalósítás és kamera létezik, de ebben részben csak a megvalósíthatóság lett tesztelve, hogy megfelelően működik-e, és működhet e. Ezt egy kijelző és egy olcsó kamera segítségével lehet a legkönnyebben elérni.



2.1 Ábra: Kamera és kijelző

A rendes tolatókamerák áramellátása a rükkerc vonalára vannak bekötve, hogy csak akkor kapcsoljanak be, amikor arra szükség van. Ebben a megvalósításban az áramot a szivargyújtó szolgáltatta. RCA kábeles bekötés után a kamera megfelelően működött, és látni lehetett a gépkocsi háta mögötti teret. Természetesen egy ilyen megvalósítás eléggé kezdetleges és senki sem használná szívesen az utcán, de elsőnek teljesen megfelel. Amennyiben viszont egy okosabb megoldást szeretnénk, használhatjuk a Raspberry Pi kameráját, amelyhez nem kell külön áramforrást bekötni, de ilyenkor is figyelniünk kell azt, hogy mikor kapcsoljuk azt be. További előnye a Raspberry Pi-nek, hogy használhatunk USB-s kamerát is.



2.2 Ábra: Raspberry Pi, kijelző és kamera

A Pi kamerának az egyetlen nagy hátránya, hogy meg kell várni a felfutási idejét, ami lényegesen több idő, mint egy sima kamera bekapcsolása.

A projektben továbbá részletesebb leírás kapcsolódik a tárgy, vagy akadály detektálásához is. Ennek a legkönnyebb megoldása, hogy a kamera képére teszünk egy közelségjelző képet, amely segítségével be tudjuk löni, hogy milyen messze van egy tárgy az autóunktól. A második megoldás, hogy visszajelezzük a tárgy távolságát is az autótól. Ez megoldható egy külön szenzorral, vagy esetünkben Computer Vision-nel is. Ez annyit tesz, hogy a kamerán keresztül egy AI kitalálja milyen messze lehet egy tárgy a kamerától.

2.1.2. Összekapcsolás OBD diagnosztikával

Az OBD2 (On Board Diagnostics) diagnosztikai port elsőként 1996-ban jelent meg az Egyesült Államokban, majd szabvánnyá nőtte ki magát. Az EU-s forgalomba gyártott autókban ezt a szabványt 2001-ben tették kötelezővé, tehát egy ideje már velünk van, de nem szerepel minden régi autóban, amelyből elég sok járja még utcáinkat.

Az OBD2 lehetővé teszi az információk valós idejű kinyerését vezetés közben is. Ilyenek lehetnek a hibák, a sebesség, fordulatszám, fogyasztás és még a motor meghibásodást jelző hibalámpát is ki lehet ezzel kapcsolni.

Elsősorban viszont kapcsolatot kell teremtenünk az OBD2-es port és a Raspberry Pi között. Az OBD2-es csatlakozó általában a kormány alatt az ajtó felőli oldalon található, néha fedetlenül, holott a biztosítéktáblán, amely lefedhető. OBD2 adapterből továbbá létezik Bluetooth-os, USB-s és COM csatlakozós is. Míg a kábeles megoldások egyszerűek, és nem is kell sokat érteni egy USB bedugásához, azonban ezek az adapterek lényegesen drágábban akár 4-6x-os áron kaphatók, mint egy Bluetooth-os társuk. Azonban meg kell jegyezni, hogy a Bluetooth adapter jelentős veszélyforrást is jelenthet. Mivel az OBD2 porton közvetlen az akkumulátorról érkező 12V-is szerepel ezért a Bluetooth mindig aktív maradhat, ha leállítjuk az autót is, és ez így egy könnyű támadási lehetőséget biztosíthat. Azonban ennek tudatában ezt okosan is kezelhetjük és az eszköz megfelelően lehet használható.

Mivel a Raspberry Pi modulok rendelkeznek Bluetooth kapcsolattal ezért a OBD2 olvasót könnyen kezelhetjük az eszköz segítségével. Miután csatlakoztattuk, egy „screen” nevű alkalmazás segítségével kommunikálhatunk is az olvasóval, beállíthatjuk azt és nyerhetünk is ki belőle adatokat. Ebben csak arra kell figyelni, hogy visszakapott értékek melyik részét kell felhasználnunk, és hogy azt hexadecimális értékekben kapjuk meg és a parancsokat is így kell kommunikáljuk az olvasó felé.

Services / Modes [edit]

There are 10 diagnostic services described in the latest OBD-II standard SAE J1979. Before 2002, J1979 referred to these services as "modes". They are as follows:

Service / Mode (hex)	Description	0A	10	1	Fuel pressure (gauge pressure)	0	765	kPa	3A
01	Show current data	0B	11	1	Intake manifold absolute pressure	0	255	kPa	A
02	Show freeze frame data	0C	12	2	Engine speed	0	16,383.75	rpm	$\frac{256A + B}{4}$
03	Show stored Diagnostic Trouble Codes	0D	13	1	Vehicle speed	0	255	km/h	A
04	Clear Diagnostic Trouble Codes and stored values	0E	14	1	Timing advance	-64	63.5	° before TDC	$\frac{A}{2} - 64$
05	Test results, oxygen sensor monitoring (non CAN only)	0F	15	1	Intake air temperature	-40	215	°C	A - 40
06	Test results, other components/system monitoring (Test results, oxygen sensor monitoring for CAN only)								
07	Show pending Diagnostic Trouble Codes (detected during current or last driving cycle)								
08	Control operation of on-board component/system								
09	Request vehicle information								
0A	Permanent Diagnostic Trouble Codes (DTCs) (Cleared DTCs)								

2.3 Ábra: OBD2 parancsok táblázat [2]

A fenti kép alapján a jármű aktuális sebességét a „010D” parancs megadásával kérhetjük le. Ebből egy példa visszakapható érték „41 0D 32 11”, melyek jelentései a következők:

- 41: Válaszérték a kiválasztott módunkra (01)
- 0D: Válaszérték a PID parancsunkra (0D)
- 32: A jármű sebessége hexadecimális értékben (32h km/h = 50 km/h)
- 11: Kihasználatlan adatbitek

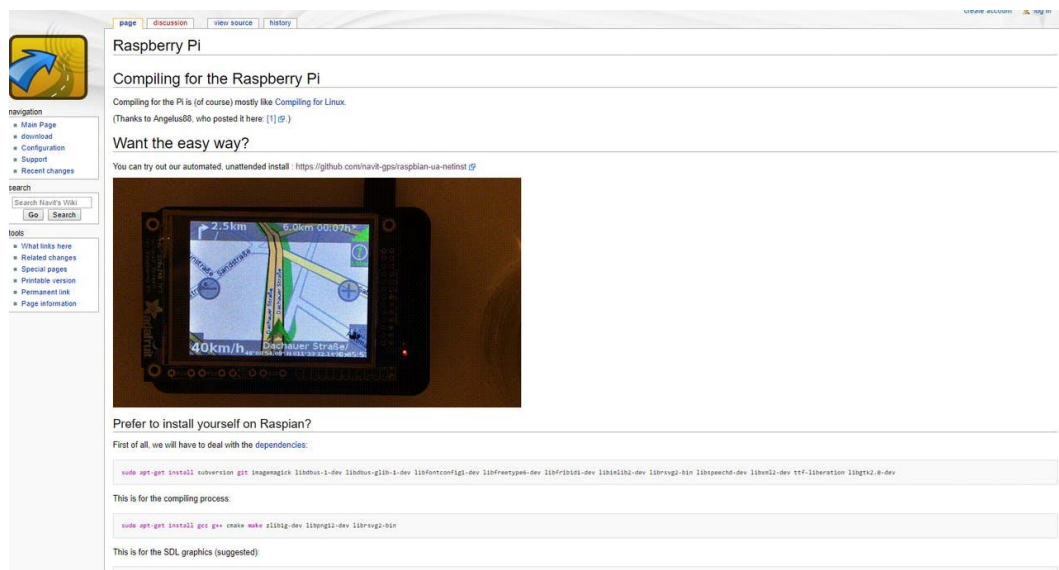
2.1.3. Grafikus interfész

Grafikus interfész egyik legkézenfekvőbb megvalósítása Pythonnal lehetséges, mivel alaptól létezik hozzá OBD és soros könyvtár implementáció is. Ezt általában egy script formájában valósíthatjuk meg ami konzolos formában fut alapértelmezetten, ezért a grafikus megjelenítéshez egy bővítményt is használnunk kell. Ilyen bővítmény lehet a PyQT, Tkinter és a Pygame. Mivel a Pygame viszonylag egyszerű és alapértelmezetten telepítve van a Raspberryre ezért ebben a projektben az került használatra.

2.1.4. GPS hozzáadása

Mivel a modern okos eszközökben már alapértelmezett egység a GPS navigáció ezért ebből a rendszerből sem maradhat ki. Szinte az is mindegy, hogy milyen egységet választunk, csak legyen kompatibilis a Raspberry Pi-vel. Legtöbb esetben amennyiben USB-s megoldást választunk telepítés nélkül használható, azonban előfordulhat, hogy használatához szükséges egy illesztőprogram beszerzése.

A navigációra rátérve a legtöbb mobil eszközös navigáció a térképek letöltésére mobilhálózatot használ, azonban ez egy havi elfizetéssel is járhat, tehát a letöltött térképeknél maradhatunk. Ezek közül szinte az egyetlen navigációs rendszer, ami offline is működik a Pi-n az a Navit.



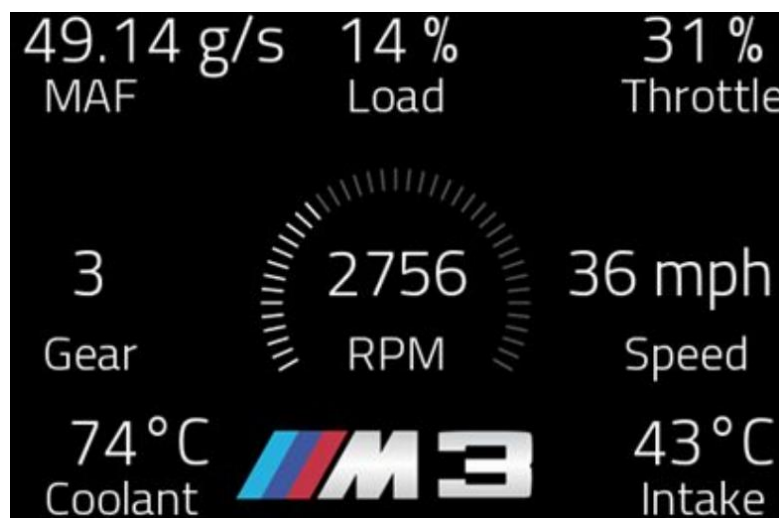
2.4 Ábra: Navit GPS navigáció oldala

2.1.5. Elemzés

A leírt projekt kisebb nagyobb mértékben hasonlít az általam megvalósítani kívánt fejegységre, tehát egyes részei alapul is tudnak szolgálni az én megvalósításomban. Elsősorban az OBD kommunikáció és a grafikus interfész megvalósítása tűnt érdekesnek, mivel ezeket én is szeretném implementálni. Hasznosnak találok továbbá a GPS bekötési részt, mivel menteni is szeretném az aktuális helyzetet és egy navigáció sem lenne túl rossz megoldás. Az OBD2-es kapcsolatot én nem Bluetoothal szeretném megvalósítani, de a kommunikációs és a lekérdezési rész hasznosnak bizonyul. Grafikus interfész kapcsán elég kevés leírást találtam, illetve egy példaprogramot, amely lehetséges, hogy felhasználható lesz számomra, de ez még egy hosszabb elbírálásra kerül.

2.2. M3 Pi – Raspberry Pi OBD-II Érintőképernyős fedélzeti számítógép

Ebben a részben Geoffrey Wacker által készített érintőképernyős állapot visszajelzős fedélzeti számítógépe kerül bemutatásra, amit az 1997-es BMW M3-as autójába készített. [3] A munkában bemutatásra kerül az OBD2 interfész, a Raspberry Pi rendszere és annak programozása. A projekt fő célja, hogy segítséget és információt nyújtson az autó állapotáról.



2.5 Ábra: M3 Pi főképernyő

2.2.1. Dizájn

Elsőnek tehát az esztétikai megvalósítással kezdek az OBD-2 protokoll részletezése nélkül.

A modul beépítésének a tervezett helye a műszerfal közepén elhelyezkedő napszemüveg-/hamu-tartó helyettesítése. Itt elsőnek megfogant a gondolat egy komplett analóg műszerfalrész megvásárlására és átalakítására, de az drágának bizonyult, mivel a projektet minél olcsóbban szeretnék volna megvalósítani, és ez a keretükbe sem fért volna bele. Ebből kifolyólag az egyénileg tervezett fedőlapnál maradtak a projekt elkészítésekor. Ez a fedőlap elsősorban fából majd 3D nyomtatással kerül elkészítésre.

Ezek után a grafikus interfész (GUI) megvalósítására esett a hangsúly. Elgondolás szerint a kijelző kiesik a vezető közvetlen látásteréből ezért egy könnyen értelmezhető felület létrehozására kell törekedni. Mivel a cél a jól olvashatóság ezért minél nagyobb betűket kellett használni miközben minden fontos információ megjelenik a képernyőn. Ebből kifolyólag a GUI a lehető legkevesebb felhasználói beavatkozással rendelkezik.

Szoftveres oldalon a feladat több fő részre lett felosztva. Szempont volt a könnyű bővíthetőség lehetősége is, hogy a későbbiekben lehessen bővítményeket is hozzáadni az elkészült projekthez. Ilyen lehet egy GPS helyadatok mentésére alkalmas kiegészítés is. Továbbiakban a szoftver Pythonban kerül megírásra. Ezeket meghatározva a szoftver 4 fő Pythonscriptből épül fel. A „config.py”, „m3_pi.py”, „ecu.py” és „log.py”. A

„config.py” felelős a beállítások megtartásáért és betöltéséért, hogy a kezelés személyre szabható legyen, és tárolná az összes globális változót. Az „m3_pi.py” a fő vezérlő, amely meghívja a részeket, felel értük és futtatja a GUI-t. Az „ecu.py” engedélyezné a különféle moduloknak, hogy elérjék a konfigurációt és a „log.py” pedig, a logolásért felelős egy .CSV fájlba. Ezen felül megvalósításra kerül még egy „shutdown.py” amely a Raspberry Pi leállításáért felelős, amennyiben jelet kap rá, vagyis ez a script szeparáltan fut az előbbi négytől.

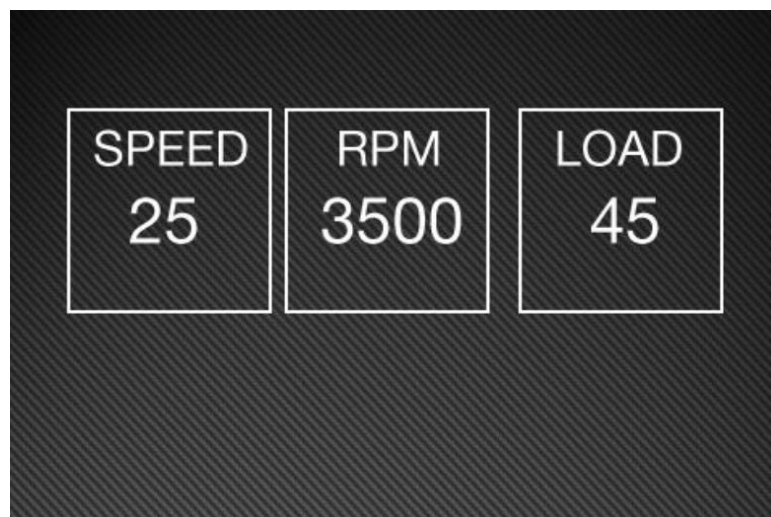
Mégiscsak a legnagyobb kihívást jelentő feladatnak a hardver bizonyult, mivel az autóból két féle módon kaphatunk feszültséget, amelyek mindegyike 12V-os. Első az akkumulátor konstans 12V-os vonala, míg a második az ACC vonal, ami csak akkor van áram alatt, hogyha elindítottuk az autót, vagy gyújtásra kapcsoltuk azt. Ebből kifolyólag érezhető, hogy valamilyen egységet be kell építenünk a Raspberry Pi és az akkumulátor közé, hogy jelezzük amikor annak le kellene állnia. Amennyiben az akkumulátorra kötnénk akkor az lemerítené azt, míg, ha az ACC vonalra, akkor az pedig károsíthatja a Pi fájlrendszerét lehetetlenné téve annak elindulását, de legalábbis megnehezítve azt amennyiben hirtelen megvonjuk tőle az áramot. Ennek orvoslására egy egyszerű áramkör megépítése lett a cél, ami az autó leállítása után jelez egy mikrokontrollernek, ami 10 másodpercen belül leáll, majd 45 másodperc múlva megvonjuk tőle az áramot a következő elindításig.

Mindent egybevéve szükség lesz még egy érintő kijelzőre, ami megfelelő beviteli és megjelenítési eszköznek bizonyul. Ennek a választása az „Adafruit Plus 3.5” rezisztív kijelzőre esett. Ennek előnye, hogy a kijelző nem igényel külön áramforrást, hanem közvetlenül a Pi áramellátása elegendő neki.

2.2.2. Fejlesztés

Miután átgondolásra került, hogy hogyan is szeretnénk felépíteni projektet, bele is foghattak a fejlesztésbe. A Raspberry Pi 3-ra egy Linux operációs rendszert telepítve kezdődhet is a szoftver megvalósítása. Az operációs rendszernek a „Raspbian Light” mellett döntöttek, mivel azt alapból a Raspberry Pi-ra írtak. Bekonfigurálása után, azt SSH kliensről is el lehet érni hálózati csatlakozással, ami egyszerűsíti a hozzáférést és ezáltal a fejlesztést.

Ezután a GUI fejlesztése következett. Az alkalmazás Pythonban lesz megvalósítva, viszont abban alapértelmezetten nem szerepel grafikus interfész. Ezért ezt egy külső könyvtárral kellett megvalósítani, amelyből számos lehetőség található, viszont ehhez a projekthez a Pygame bizonyult a legkönnyebben kezelhetőnek. Ez alapértelmezetten videójátékok készítésére szolgál, viszont kellően egyszerű a megvalósítása így számottevő projekthez alkalmazható.



2.6 Ábra: Pygame GUI "Proof of Concept"

Ezek után nehézségnek az élő adatok lekérése bizonyult. A fő probléma az volt, hogy adat lekérésekor az ODB2 könyvtár szinkron módon működik, mellyel várakoztatja a rendszert, és ezzel várakoztatva a GUI-t is. Azonban ez egy aszinkron metódushívással orvosolható volt. Teszteléskor felmerülő problémák közé tartozott még, hogy csak az autóban szerepel OBD2es használható interfész így-is nehezkesebbé téve az OBD2 tesztelését, az adatok feldolgozásának helyességét, és a program futását. Ennek megoldása egy OBD2 emulátor, amely beállítása után emulálja egy autó OBD2es portját, és az abból kinyerhető adatokat és válaszcímeket.

Miután mind az aszinkron OBD2 kommunikációt és az emulátort is beállítottuk az olvasóból a „0100” paranccsal kinyerhető annak támogatott PID listája. (pl. sebesség, fordulatszám, vízhőfok stb.) Mindezek után, még egyetlen adat hiányzott, amely pedig a sebességváltó aktuális fokozata. Mivel ezt alapértelmezetten nem lehetett kinyerni az OBD2es protokollon keresztül ezért ez egy saját keresőtáblával lett megvalósítva, mely értékeinek kiszámítása az aktuális sebességből és a fordulatszámából adódott.

$$tRPM = \frac{\text{áttételi arány} * \text{Jármű sebessége} * \text{átviteli arány} * 336.13}{\text{kerék átmérője}} \quad (1)$$

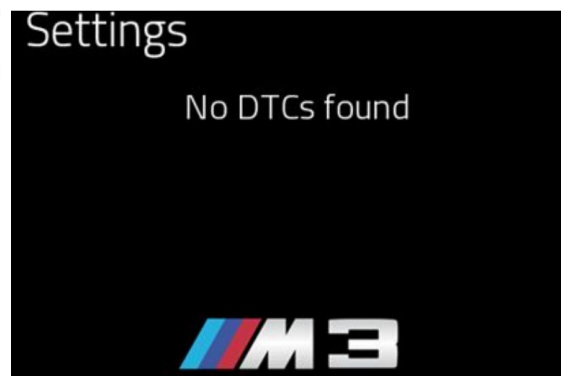
Az M3as esetében az áttételi arány az a fokozati fogaskerék aránya, az átviteli arány pedig az aktuális fogaskerék átmérője amelyikben a sebességváltó áll. Az M3as sebességváltója 5 fokozatos és az utolsó fogaskerék aránya 1:1 méretű. Végezetül a 336.13 konvertálja át az értéket fordulatszámra, és a kerekek átmérője pedig 17”.

Ez a megoldás meglepően jónak bizonyult, főként egy kis logikával, hogy tudjuk mikor van a sebességváltó üres állapotban. Feltételezhetjük, hogy 0 sebességnél és < 800 fordulatszámnál a sebességváltó üres (N) fokozatban van.

Ezek után megtervezésre kerülhetett a GUI, amelyen megjelenik minden lényeges információ, és megérintésével a program beállításaiba lehet eljutni.



2.7 Ábra: Fő képernyő

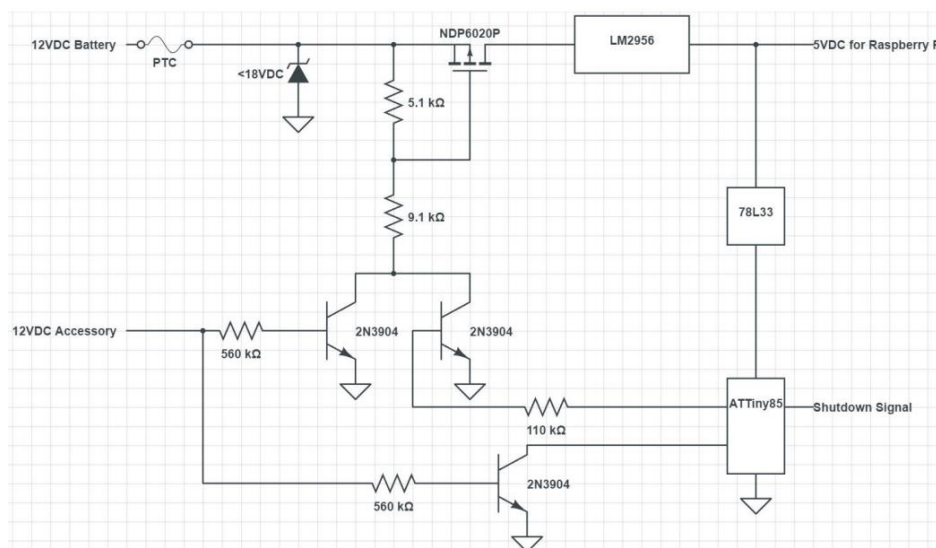


2.8 Ábra: Beállítások képernyő

A fordulatszám-mérő összesen 42 képből áll, amelyek 200-as fordulatszám-onként váltakoznak. Ezek a képek Adobe Illustrator-ban kerültek megalkotásra és miután elérik a 7000-es fordulatszámot pirosra váltanak. A készítő elmondása szerint, ezen jól látható a motor fordulatszáma az aktuális értékre nézés nélkül is.

Legutolsó szoftverkomponensnek még a logolás maradt, amelyet kellő egyszerűséggel a Pi SD kártyájára lehetett menteni egy .CSV szöveges állományba. Ennek a mentése másodpercenként történik meg, egy dátum és idő elnevezésű fájlba.

A szoftver elkészülte után az áramellátásra fordult a hangsúly. Az elképzelt dizájn alapján, a Raspberry Pi-nek egy közvetett áramkörön keresztül történik az áramellátás, amelyet egy ATtiny85 mikrokontroller vezérel. Ennek a feladata, hogy gyújtáskapcsolásnál szolgáltatssa az akkumulátor áramát a Pi-nek és az autó leállítása után jelezzon, majd 45 másodperc múlva vonja meg az akkumulátor áramát a Raspberry Pi-től. A jelzést megkapva a Pi körülbelül 10 másodpercen belül leáll és áramellátása biztonságosan eltávolítható az SD kártya fájlrendszerének sérülése nélkül.



2.9 Ábra: Áramellátásért felelős áramkör

Ezen felül az áramkörben, a 12V átalakításához egy LM2596 DC-DC konvertert használtak, aminek a segítségével a bementi 12V-ot a Raspberry Pi-nek is megfelelő 5V-

ra alakít át. Az áramkör 3 bemenettel rendelkezik, amelyek: a 12V-os közvetlen akkumulátorra kapcsolt feszültség, az indítást jelző 12V-os ACC feszültség és a föld. Amikor a Raspberry Pi megkapja a kikapcsolási jelet, akkor az a „`sudo shutdown -h now`” paranccsal leáll. Ezek után már csak mindent megfelelően rögzíteni kell az autóban.

A legutolsó feladat volt, hogy a GUI induljon is el a rendszer felfutásánál. Ezt egyszerűen egy a rendszer indulásánál meghívott „`/etc/rc.local`” fájlba kell beírunk, hogy a programunk a felfutás után elinduljon. Ez, a parancsokat új sorokba írva, a következőképpen néznek ki:

- „`sudo python home/pi/shutdown.py &`”
- „`sudo python home/pi/m3_pi.py &`” [4]

Fontos, hogy a sorok végén szerepeljen az `'&'` karakter, mivel a rendszer ekkor tudja, hogy a scripteket különálló új szálokon kell indítania, és így azok a háttérben futtassanak.

2.2.3. Elemzés

Ebben a projektben részletesebben kitér a szerző a fizikai megvalósításokra és a megvalósítási módszerekre. Nekem felettébb tetszett a megvalósítási ötlet és sok mindent tudok belőle alapul venni, hiszen én is egy hasonló eszközt szeretnék megvalósítani. A sebességváltó fokozatának számítása is felkeltette a fantáziámat, amelyet még a későbbiekben meglátom, hogy megvalósítom-e, esetleg milyen formában lehet a saját elképzeléseimmel megvalósítani. Összességében alapként is vehetném a projektet, és megvalósíthatnám belőle az én elképzeléseimet, viszont ezen felül a saját kutatásomat hozzátéve, egy összességében funkciókban gazdagabb megvalósítást szeretnék elérni. Igazán hasznosnak találom a logfájlok készítését és az OBD2-es protokoll késleltetési idejének figyelembevételét is. Grafikus interfésznek, mivel én egy nagyobb kijelzőt szeretnék használni, egy részletekben gazdagabb megvalósításért, ezért még megnézek többféle grafikus megvalósítást is. Kezdetnek viszont ebben a munkában szerepel egy részletes Pigame GUI megvalósítás is. Az egyetlen egység, amit azonban szinte egy az egyben átmenelnék a projektembe esetleges kis változtatásokkal, az az áramellátásért és a leállítási figyelmeztetésért felelő tápegység modul lenne, hiszen ez a sajátkezüleg megépíthető ATtiny85-re épülő modul, amely meglehetően hasznos áramkörnek bizonyul az áramellátásra.

2.3. OBD-Pi Feketedoboz [5]

Ebben a projektben, Raspberry Pi felhasználásával készül egy OBD2-es adatmentő, amit a későbbiekben táblázat segítségével is vissza lehet nézni, azonban nem definiálják az adapter fajtáját, csak, hogy lehet az Bluetooth-os, vagy USB kábeles is, viszont a Bluetooth-os megvalósításhoz egy USB-s Bluetooth adaptert ajánlanak a Raspberry Pi oldaláról.



2.10 Ábra: OBD-Pi Állapotjelző kijelző

A projektben, a továbbiakban felhasznált eszközök:

- Raspberry Pi Model B, vagy B+
- Bluetooth USB Adapter
- 2A-es autós töltő
- ELM327 Bluetooth vagy USB Adapter

2.3.1. OBD2

Az OBD2 (On-Board Diagnostics), másnéven fedélzeti diagnosztika, amit elsőnek az USA-ban vezettek be 1996-ban. Ma ez már minden autóban egy kötelező feltétel, hogy monitorozni lehessen az autó motorját, váltóját, és vezérlőelemeit. Az OBD2-es csatlakozót mindig a kormánykerék alatt találjuk az egyik oldalon.

2.3.2. PyOBD

A PyOBD egy nyílt forrású Pythonban írt szoftver. A szoftver arra szolgál, hogy az olcsóbb ELM327 OBD2-es Adapterekkel kommunikáljon és megjelenítse az azok által közvetített adatokat. Ez ki tud olvasni hibakódokat és mért értékeket is.

Ez megtalálható giten és Linux-on 'apt-get'-el is telepíthető. [6] Ehhez csak az internet elérés és egy SSH kapcsolat, vagy egy monitor csatlakozása szükséges.

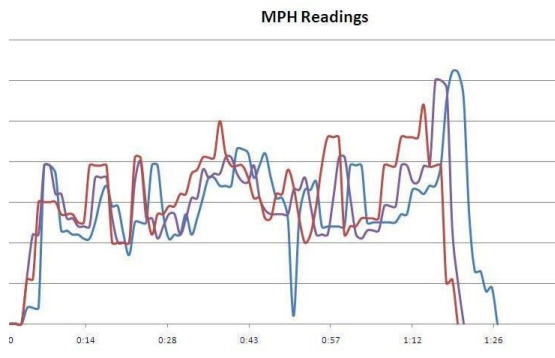
2.3.3. Autóba építés

A projektben az elmondás szerint a rendszert az autóba beépíteni viszonylag egyszerű, azonban erre az általam látottak alapján az autó beépített kijelzőjét használták:

1. Helyezzük az USB-a Bluetooth adaptert a Pi-be az SD kártyával együtt,
2. Illesszük be az OBD Adaptert az autó megfelelő portjába,

3. Dugjuk be az autós töltőnket,
4. És végezetül indítsuk el az autót.

Lépünk be, indítsuk el a programot és már láthatjuk is a kijelzőn megjelenő adatokat, és utólag olvashatjuk is a mentett logokat.



2.11 Ábra: OBD-Pi Grafikus ábra

2.4. GPX Fájl Formátum

Ebben a részfejezetben, nem egy kifejezett projektet mutatok be, hanem egy fájlformátumot, amelyet GPS helyadatok és útinformációk mentésére használnak. Ezt a standardot azért találták ki, hogy egyéb okos eszközeinken (vagy csak régebbi GPS-ünkön) lemerített GPS koordinátáink listáját egy egységes formátumban mentjük le, hogy azt utána egy másik, például asztali szoftver segítségével is szépen meg tudjuk tekinteni.

2.4.1. Mi is az a GPX formátum

A GPX formátumot 2002-ben adták ki elsőnek. [7] A formátum egy XML sémára alapuló dizájn, szoftveres applikációk számára. A formátum útvonalpontok, nyomvonalak és utak mentésére szolgál, továbbá nyílt forrású, tehát bárki használhatja és rengeteg applikációban támogatott.

2.4.2. Adattípusok

A legújabb szabványban a névjegyadatoktól kezdve rengeteg mindent el lehet tárolni, azonban számunkra a fontosabbak azok a: waypoint (wpt), route (rte) és a track (trk) típusok. Az útvonalpont-al (waypoint) olyan pontokat definiálunk, amelyek egymással nincsenek összekötve, csak emlékeztetőnek felvettük őket. Az út-al (route) iránymódisító pontokat vehetünk fel, például, ha egyenesen haladtunk és jobbra kanyarodtunk akkor csak a kiinduló pontot és a kanyart jegyezzük fel. A pályával (track) pedig folyamatos haladási utat rögzítünk. Ez egy sorbarendeztet lista, ami az út kezdetétől a végéig, vagy a gps jel szakadásáig tart.

Minden típusnak két kötelező adata van, ez pedig a szélességi és a hosszúsági fok. Ezen felül kiterjesztésekkel ezt még bővíthetjük páratartalommal, utcanevekkel és személyes adatokkal is.

2.4.3. Mértékegységek

A szélesség és hosszúság elvárása, hogy decimális érték legyen, az emelkedést méterben, az időt pedig központi (UTC) időszámítás szerint kell megadni, nem lokális érték szerint. [7]

2.4.4. GeoJSON

Ezt a formátumot azért említem meg, mert ez is egy egyre elterjedtebb JSON alapú formátum. Megvalósítása hasonló a .GPX-hez, viszont kevesebb utófeldolgozást igényel, és rengeteg mai fejlesztési eszközben is használatos. Tehát néhány esetben felesleges .GPX formátumba menteni, amennyiben azt minden esetben át fogjuk alakítani GeoJSON formátumba. Megvalósítása egyszerűbb és JavaScript által könnyebben is olvasható és menthető.

```
{
  type: 'LineString',
  coordinates: [
    [18.882160833333334, 47.780973],
    [18.882160833333334, 47.780973],
    [18.88216, 47.780973833333334],
    [18.88216, 47.780973833333334],
    [18.88216, 47.780973833333334],
    [18.882157833333334, 47.780976166666667],
    ...
  ]
}
```

2.12 Ábra: GeoJSON minta

2.5. Angular GPS Tevékenység napló [8] [9]

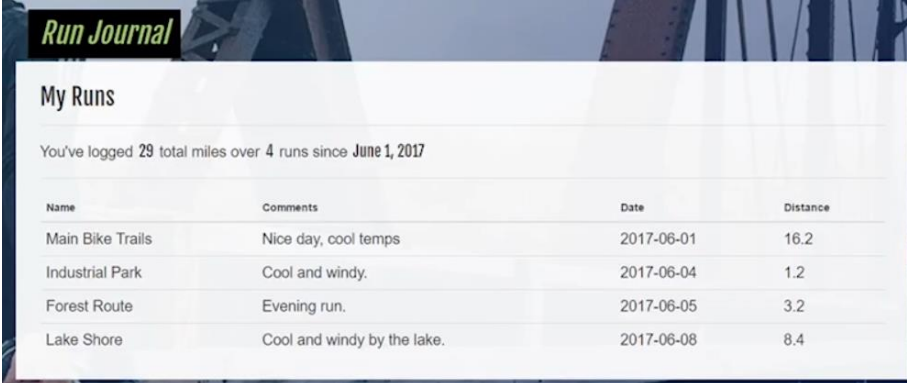
A következőkben Wes Doyle tevékenységnaplóját mutatom be, amit Angular 4-ben írt és segítségével feltöltheti és megtekintheti a futás közben telefonról lementett .gpx fájljai tartalmát bárholnan térképen ábrázolva. A bemutatott videóban Doyle két fő komponensre bontja az oldalát. Ezekből az első a listázásért, a második pedig a térképre rajzolt útvonal megjelenítéséért felelős.

2.5.1. Térkép és leaflet

Térképként a leaflet JavaScript alapú könyvtárat használja és ehhez megfelelően az adatokat GeoJSON formátumba alakítja, amit a leaflet könyvtár támogat. Erre szolgál a leaflet-omnivore könyvtár, amivel a GPX kiterjesztést át lehet alakítani. [10] Ezt a könyvtárat az idők alatt beépítették a '@mapbox/leaflet-omnivore' könyvtár alá. Továbbá megvalósításához a mapbox által biztosított térképeket használja.

A megoldásban a keresett fájlok elérését hardkódolja az alkalmazásban, tehát nem használ backendet, ugyan ez lekérésekkel a weben megvalósítható lenne. A megtett útvonalait egyénileg dolgozza fel és a kész térképen csak megjelenítendő adatok szerepelnek. Ezért az első oldalon egy listából betölti, hogy milyen útvonalakat tett eddig

meg, és kiírja ezek adatait. A második oldala egy térképet valósít meg amire a leaflet könyvtár segítségével rávetíti a GeoJSON-né alakított útvonalát és szépen megjeleníti azt.



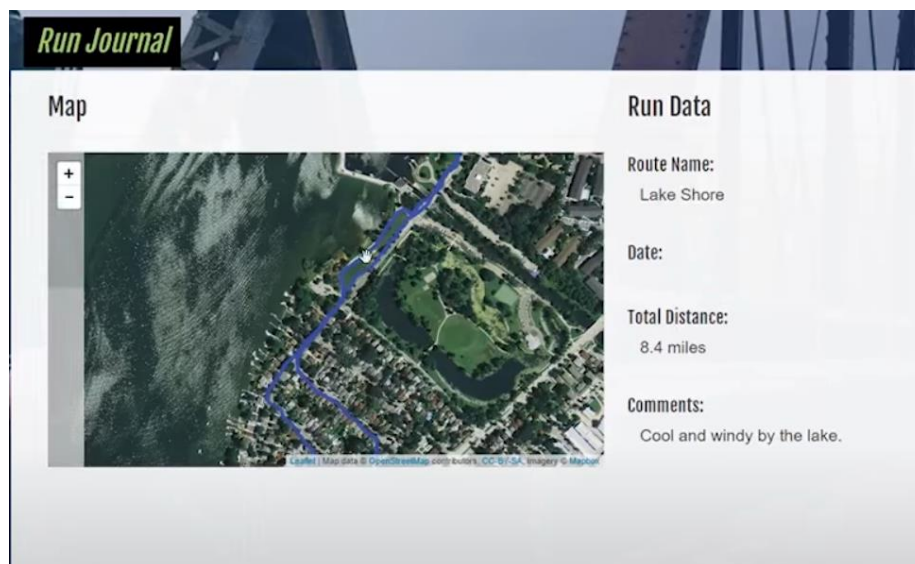
Run Journal

My Runs

You've logged 29 total miles over 4 runs since June 1, 2017

Name	Comments	Date	Distance
Main Bike Trails	Nice day, cool temps	2017-06-01	16.2
Industrial Park	Cool and windy.	2017-06-04	1.2
Forest Route	Evening run.	2017-06-05	3.2
Lake Shore	Cool and windy by the lake.	2017-06-08	8.4

2.13 Ábra: Kocogási táblázat



2.14 Ábra: Térképes útvonal

2.5.2. Elemzés

A videóban bemutatottak alapján hasznosnak találtam a megvalósított térkép funkciót, mivel hasonlóan ki tudnám mutatni az általam megtett útvonalat, és ezáltal egy szép visszajelzést adni, a felhasználónak az útvonal értékelése mellett. Természetesen ez csak egy online megvalósítást taglal, tehát előfordulhat, hogy a felhasznált térkép internet elérést fog igényelni, amihez legalább egy mobilnet megosztás szükséges. Azonban ez egy egész szépnek mondható megvalósítást eredményez.

3. A MEGOLDÁSI MÓDSZER KIVÁLASZTÁSA, A VÁLASZTÁS INDOKLÁSA

Hardveres szempontból a rendszert egy Raspberry Pi fogja vezérelni, amely felelős lesz a szoftver futtatásáért, a megjelenítésért és a teljes rendszer működéséért. Az autóval való kapcsolatot egy ELM327 OBD-II USB-s diagnosztikai kábel fogja biztosítani és emellett a helymeghatározáshoz egy USB GPS modul kerül beépítésre. A Raspberry Pi képes hang lejátszására is, tehát nem lenne feltétlenül felszükség egy külön erősítő modulra, viszont ez nem biztosítana elegendő feszültséget az autó hangszóróinak, tehát elengedhetetlen egy külső erősítő felhasználása is. A médiatartalom szolgáltatásáért egy kivezetett USB port lesz a felelős, amelybe egy pendrive-ot helyezve el is kezdhethetjük a zenelejátszást. Az áramellátásért és az automatikus kikapcsolásért egy házilag szerkesztett áramkört fogok elkészíteni, amely az ATtiny85 mikrokontrollerre épült volna, viszont megoldásomban ennek a szerepét is a Raspberry Pi fogja ellátni. [3] Végül megjelenítőként pedig a Raspberry Pi 7" kijelzőt fogom használni.

Fizikailag a rendszer egy 2 DIN-es rádiókeretben elhelyezhetőre készül, ezen felül az autó aktuális vezetési adatainak és helyzetének mentése a Raspberry Pi rendszerének SD kártyájára történhet.

A szoftver megvalósítása AngularJS-ben készül, amelyet Electron segítségével fordítok Armv7l Linux platformra. Ennek hátránya, hogy lényegesen sok erőforrást igényel, mivel az elkészített programunkat egy beépített Chromium böngészőn keresztül fogja a program futtatni. Előnye viszont, hogy ezáltal HTML, CSS, JavaScript nyelveken lehet egy programot megírni, amellyel egy reszponzív és érintő kijelzőn könnyen kezelhető applikációt lehet készíteni, amelyhez nem feltétlenül kell egy külön GUI tervezőt használni. Vetélytársai voltak a PyQt, és a c++ alapú Qt, amelyek az Angularral ellentétben sokkal kevesebb erőforrási igényelnek, viszont a PyQt nehezebb dizájn megvalósítást biztosít, a Qt pedig fizetős és használata körülményes, habár létezik hallgatói ingyenes licence. Egy alternatívának találtam még a Windows10 IoT operációs rendszert, amelyre szintén C++, akár C# segítségével lehet programozni .NET Core-ban és UWP-ben, viszont ez eszközeimet jelenleg nem támogatja.

A megvalósításhoz azért ezt a módszert választottam, mert vételárban egy hasonló 2 DIN-es rádió sem sokkal gazdaságosabb, és ezzel egy később programozható, könnyen továbbfejleszthető és hozzáférhető rendszer megvalósítása lehetséges. A Raspberry Pi elegendő egy grafikus interfész megjelenítéséhez és az adatok egyidejű feldolgozásához. Továbbá támogat szálkezelést is, amely nagyban megkönnyítheti az elvégezni kívánt feladatot, nem is beszélve a rengeteg alapértelmezett felszereltségéről, amelyek alpból rendelkezésünkre állnak.

4. A RÉSZLETES SPECIFIKÁCIÓ LEÍRÁSA

A rendszerem specifikálását két fő részre osztom és sorolom fel. Egyik a hardveres, a másik pedig a szoftveres. Hardveres tekintetben nézem meg a fizikai megvalósítást, szoftveresben pedig a program működését, és a kezelőfelületeket.

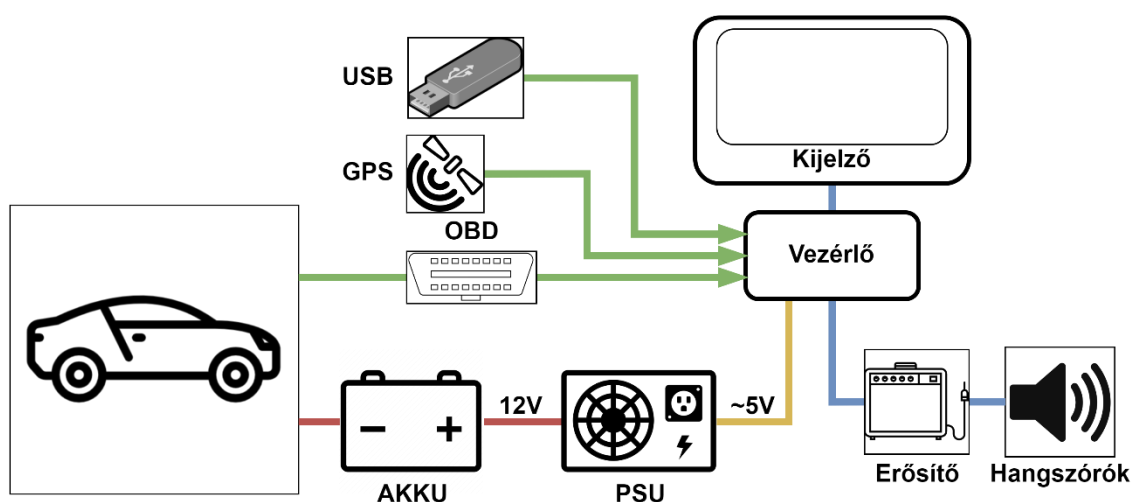
4.1. Hardveres

Mivel a hardveren futtatja a szoftvert és arra megfelelően kell tervezni azt, ezért ennek a tervezése élvez számomra elsőbbséget. Mivel a Raspberry Pi tekinthető egy számítógépnek ezért az széles körben felhasználható, amikor a szoftveres utófeldolgozás a fő cél. Az fejlesztői eszközt kiválasztva a hardveres felépítésbe a következő modulok kerültek bele:

Megnevezés	Szerepkör	Ár (2021)
• Raspberry Pi 4	Vezérlőegység	20 280 Ft
• RPi 7" kijelző 800x480p	Megjelenítés	27 990 Ft
• GPS modul [3]	Helymeghatározás	5 962 Ft
• TPA3116D2 Erősítő	Hang erősítése	5 500 Ft
• OBDII USB Kábel	OBDII-es diagnosztika	4 830 Ft
• Power modul	Áramellátás és annak megvonása	~ 3 000 Ft
• Kábelek és beépítés	Autóba beszerelés, USB csatlakozás	~ 2 000 Ft
Összesen:		~ 69 000 Ft

4.1. Táblázat: BILL OF MATERIAL Táblázat

A felsorolt elemek kapcsolata az alábbi diagrammon látható:

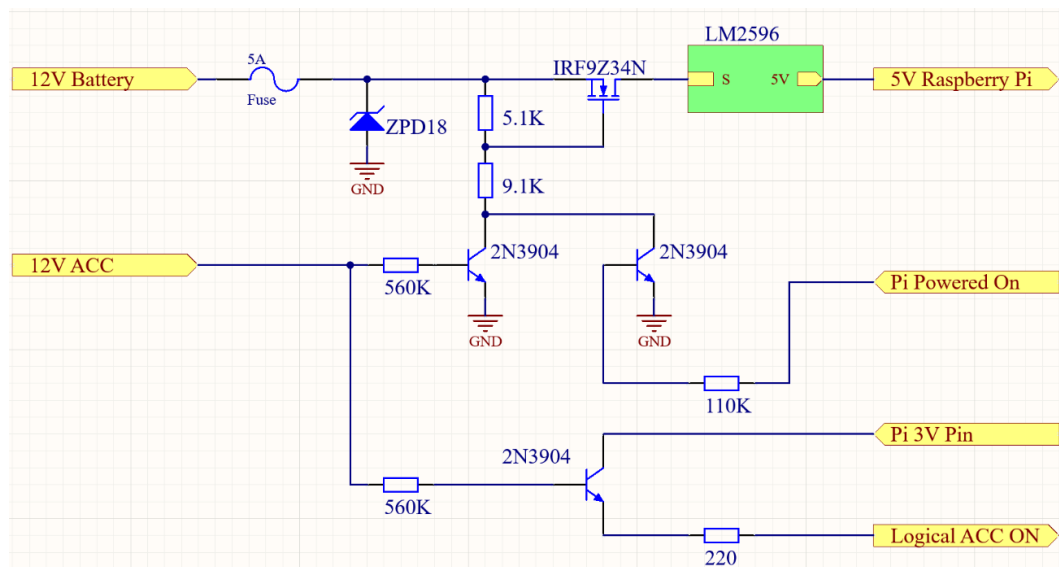


4.1 Ábra: Fizikai összeköttetések

A megvalósítás során minden modul a Raspberry Pi-hoz csatlakozik, melyeknek bekötése a fenti ábrán látható. Mivel a csatlakoztatott eszközök nagyrésze USB és szabványos kapcsolatokon keresztül kommunikál ezért csatlakoztatásuk egyszerű.

4.1.1. Áramellátás és kikapcsolás

Azonban az energiaellátására szolgáló modult azt nekem kell megterveznem, melyre egy AtTiny85-öt terveztem felhasználni, azonban végül közvetlen a Raspberry Pi-n keresztül fogok vezérelni. Ez a modul kis energiafogyasztású, és önmagában buta, ezért vezérlése csak egy kimeneti és bemeneti logikai jelből áll. Ennek a megvalósításához az M3-Pi munkában bemutatott megoldást fogom felhasználni, annyi változtatással, hogy az nem 45ms múlva vonja meg az áramot a jelzés után, hanem azután, hogy a Raspberry Pi-től már nem kap engedélyező jelet. [3] Ennek oka, hogy lehetőséget tart fent, későbbi interakció általi, vagy hiányi befejezésre is. Ez azért lehetséges mert a Raspberry Pi-n beállított GPIO kimeneti jelek megszűnnek, amennyiben az megfelelően leállt, ezáltal ezt lehet aktiválási jelnek érzékelni és a kikapcsoláshoz használni.



4.2 Ábra: Áramellátó modul 2 lehetséges tervezett megvalósítása

Mindezek után a rendszert egy beépítőkeretbe helyezem, hogy az ne mozogjon az autóban. Elsősorban a keret fából készül és hátulja nem lesz teljesen zárt, de a későbbiekben ez egy rendes hátlappal bővíthető, illetve a szebb megjelenés érdekében 3D nyomtatással esztétikusabbá tehető.

4.2. Szoftveres Grafika

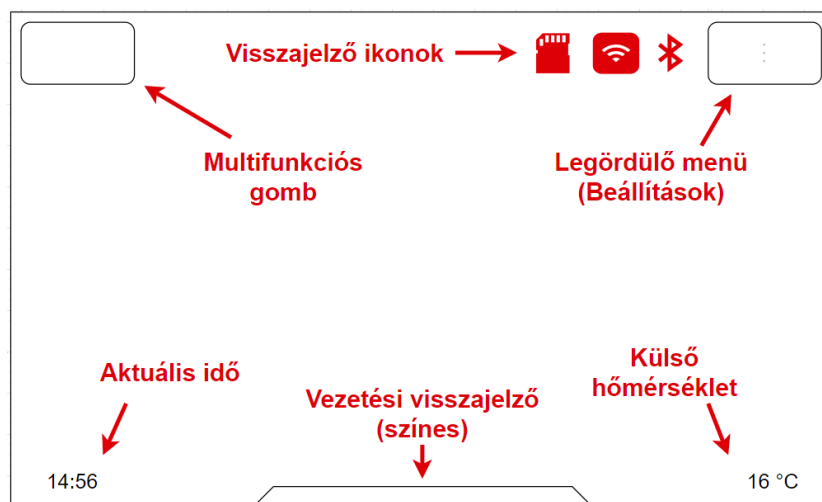
Mivel a rendszer működőképességét a szoftver adja ezért ez is nagy fontosságot élvez, és elsősorban a megjelenő oldalakat és felületeket határoztam meg, hogy mit is szeretnék elkészíteni. Ezek alapján a szoftver két különböző elemre fog bomlani, melyek a(z):

1. adatgyűjtő scriptek,
2. és a **GUI**.

Ugyan a felsorolt szoftverrészek különállóak, mindet ugyanaz a gazdaszoftver fogja futtatni külön szálakon. Ugyan csak két rész van itt kifejtve, de ezek sok kis részből fognak felépülni. Az adatgyűjtő scriptek külön-külön lesznek felelősek az OBD, a helyadat, és az USB csatlakozás begyűjtéséért, mentéséért, a GUI pedig ezeknek a begyűjtött adatoknak a megjelenítéséért. Ezáltal az adatok vándorlása esemény vezérelt (event driven) lesz.

4.2.1. Keret

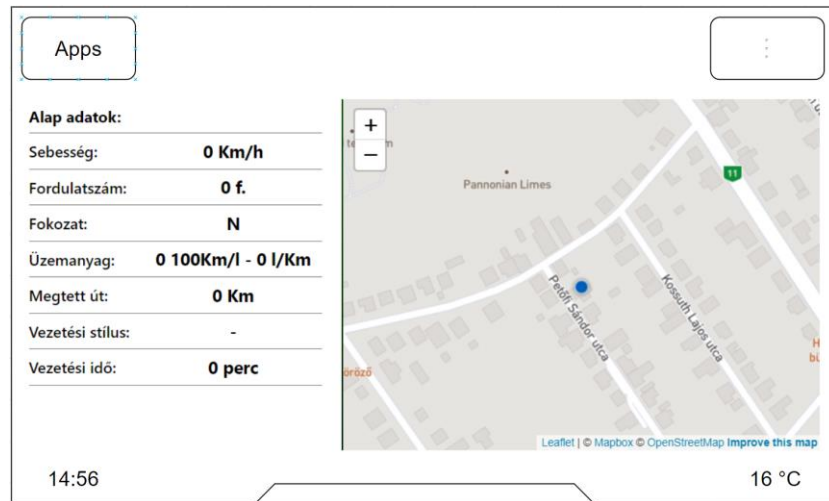
Mielőtt megnéznénk, hogy milyen képernyők lesznek és azok, hogyan épülnek fel, nézzük meg, hogy a rajtuk többségben azonos keret milyen funkciókat lát el. Ez a szoftverben két részből épül fel és mindegyik elemhez külön kell hozzáadni, viszont saját vezérlővel bír, így néhol el is lehet hagyni. A felső részen egy multifunkciós gomb található, amelyen a megjelenő szöveget és funkcióját minden oldalon külön állíthatjuk, és egy menü gomb, amivel elérhetjük a beállításokat is. Az alsó részen az idő, a külső hőmérséklet és a színes vezetési visszajelző kap helyet.



4.3 Ábra: Szoftver keret magyarázat

4.2.2. Főképernyő

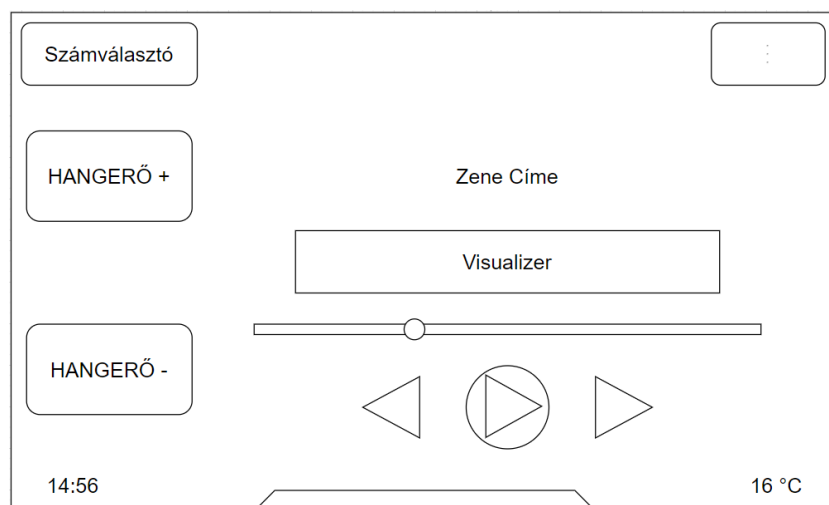
A főképernyő jelenik meg alaphoz az autó elindításakor, és ezen keresztül is tudjuk elérni a többi szolgáltatást is. Megjelennek rajta az OBD által begyűjtött adatok, a megtett út és az elindulás óta eltelt idő is. Ez egy kisebb aktuális összegzést nyújt a kirándulásunkról és az aktuális vezetési stílusunkról.



4.4 Ábra: Főképernyő ábra

4.2.3. Zenelejátszó

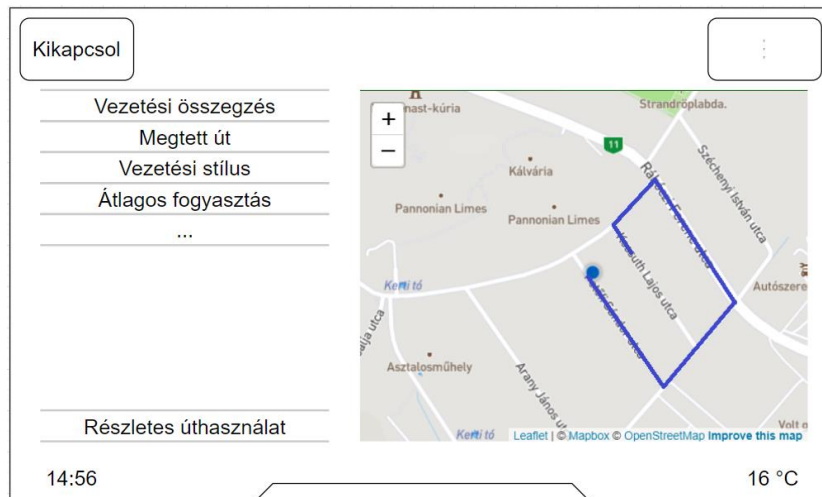
A zenelejátszó elérése az a **Fő képernyő**ről lehetséges és feladata a zenés médiatartalmak lejátszása. Alapértelmezetten a tervezésbe a zene lejátszás pendrive-ról történhet. Elsősorban a hanglejátszást csak erről a forrásról tervezem, de a későbbiekben bővíthető rádió és Bluetooth hanglejátszására is. A médialejátszó megvalósításában nehézséget nyújthat, hogy a hanglejátszásnak nem kellene megakadnia amint ellépünk a képernyőről, de ez a szoftver más részében orvosolható.



4.5 Ábra: Zenelejátszó ábra

4.2.4. Vezetési összegzés

Legtöbb esetben ezt a képernyőt csak akkor fogjuk látni, ha már megálltunk az autóval és leállítottuk a motort. Ennek a képernyőnek a szerepe összegezni a megtett utunkat, és részletezni, hogy milyen a vezetési stílusunk. Itt kikapcsolhatjuk az eszközt manuálisan, vagy az érintés nélkül 20mp múlva leáll. Továbbá a térképre kattintva kinagyíthatjuk azt, egy másik térképes oldalra ugorva, ahol nagyobbban láthatjuk a megtett utunkat.



4.6 Ábra: Vezetési összegzés ábra

4.2.5. Beállítások

Ez a menüpont szinte önmagáért beszél, és alapértelmezetten mindenhol elérhető. Itt tudjuk megváltoztatni a program beállításait és személyre szabni, hogy milyen funkciókat, hogyan szeretnénk használni, például milyen legyen a zenelejátszó fő színe, milyen hőmérsékletet használjunk, és hogy milyen mértékegységeket.



4.7 Ábra: Beállítások ábra

4.3. Szoftveres Logika

Logikai folyamatok alatt értem azokat a megvalósítási módszereket, hogy miszerint számolom az általam kijelzett értékeket és hogyan is működik a program háttére.

4.3.1. OBD2-es adatfeldolgozás

Mivel az adatokat OBD2-es porton keresztül kapom meg ezért szükségem van egy interfészre, ami képes azt beolvasni és lefordítani számomra érthető és felhasználható formátumba. Természetesen ezt én is megvalósíthatnám, de mivel létezik rá számomra használható megoldás, így amellet döntöttem. Ezt tehát egy könyvtárral valósítom meg, kiszervezett programrészként. Az így megkapott adatokat JSON formátumból változókká alakítom és közvetítem a megjelenítő és az adatgyűjtő felé. Ezáltal az adatokat mentem, és az összegzésre is gyűjtöm.

4.3.2. Vezetés értékelés meghatározása

A vezetési stílusunk részletes meghatározását számos adatból számolhatjuk. Elsősorban figyelembe vehetjük a sebességet, az ahhoz tartozó fordulatszámot, az aktuális fogyasztást és ezen felül a minél pontosabb és részletesebb elemzés érdekében akár a fékezési erőt, az autóra ható centrifugális erőket (kanyarodásnál), és végsősorban a motor terhelését is. Mivel ezen adatok számottevő részét le tudjuk kérdezni az autótól OBD kapcsolaton keresztül ezért nem feltétlenül szükséges egyéb mérőeszközöket használni, de centrifugális és az autóra ható fékezési erőt már nehezebb így meghatározni.

Munkámban a vezetői stílus meghatározására egy mondhatni egyszerű képletet használok, ami jelen állapotában pillanatnyi visszajelzést ad a vezető aktuális stílusáról. Ezt vizuálisan ábrázolom egy általam látott megoldás képére, egy színes csíkkal, amellyel főként azt szeretném demonstrálni, hogy mennyire energiatakarékosan kezeljük az autónkat. Például kigyorsításnál és nem megfelelő fokozatnál, magasabb fogyasztásnál jelezzem egy rosszabb értéket. Erre a következő képletet alkottam meg:

$$DS = \frac{rev}{vss} * fc = \frac{fordulatszám (rpm)}{sebesség (km/h)} * \frac{aktuális fogyasztás (l/100km)}{100} \quad (2)$$

A képletben a DS (Driving Style) jelenti a meghatározott értéket, aminek a meghatározott értéktartományai a 4.2. Táblázat: Lehetséges vezetési stílusok táblázatban láthatók. A fordulatszámot és a sebességet nyers adatként kérem le az autó vezérlőjétől, a fogyasztást pedig a következőképpen számolom:

$$fogyasztás (l/100km) = \frac{maf}{sebesség} * 0.3969 = \frac{3600 * maf}{9069.90 * vss} \quad (3)$$

Amiben a maf (Massive Airflow) a motorba befolyó levegő értéke.

Vezetési stílus	Értéktartomány	Megjegyzés	Szín(kód)
Lehetetlen	$DS \leq 0$	Álló motor, vagy 0 sebesség.	#05cdf5
Nagyon jó	$0 < DS \leq 3$	Díjazott jobbnál jobb érték.	#00b500
Jó	$3 < DS \leq 5$		#02f72b
Átlagos	$5 < DS \leq 8$		#ffd500
Rossz	$DS > 8$		#f70212

4.2. Táblázat: Lehetséges vezetési stílusok

Ezen értékek meghatározásának egy részeként a következő számításokat végeztem, hogy meghatározzam az egyes stílusok közötti határok értékét:

$$vss = 20; rev = 3000; fc = 0.14 \text{ (14 l/100km)}$$

$$DS = \frac{3000}{20} * 0.14 = 21 \Rightarrow \text{Rossz} \quad (4)$$

$$vss = 40; rev = 1700; fc = 0.06 \text{ (6 l/100km)}$$

$$DS = \frac{1700}{40} * 0.06 = 2.66 \Rightarrow \text{Nagyon jó} \quad (5)$$

$$vss = 80; rev = 4000; fc = 0.2 \text{ (20 l/100km)}$$

$$DS = \frac{4000}{80} * 0.2 = 40 \Rightarrow \text{Rossz} \quad (6)$$

4.3.3. Sebességváltó fokozat meghatározása

A sebességváltó fokozatszámításának meghatározása egy általam megtekintett irodalom alapján fogant meg a fejemben. [3] A munkában erre egy keresőtáblát alkottak meg, használnak, és abból keresték ki az illeszkedő fokozatot, azonban én ennél egy egyszerűbb megvalósítást szerettem volna találni. Megfogant a fejemben, hogy mely ismert adatok függenek a sebességváltó állapotához, és arra jutottam, hogy elegendő csak a fordulatszám és a sebesség hányadosát tekinteni, hiszen minél nagyobb fokozatban vagyunk, a sebességünk annál nagyobb lesz és ehhez képest ideális környezetben (városban, főúton) a fordulatszám az szinte ugyanazon értékek között stagnál.

$$GV = \frac{rev}{vss} = \frac{\text{fordulatszám (rpm)}}{\text{sebesség (km/h)}} \quad (7)$$

A képletben a GV (Gear Value) normál esetben egy nem negatív, valós szám: fokozati ráta. Természetesen a képlet autópályán magasabb sebességnél is működőképes lehet, hiszen annyival nem változik a képzett hányados értéke, jól meghatározott határértékek után. Az egyes fokozatokhoz következő értékekhez rendeltem: (4.3. Táblázat)

Fokozat	Értéktartomány
N - Üres	$vss = 0, vagy$ $vss > 10 \text{ és } rev < 1000$
1	$GV \geq 100$
2	$70 \leq GV < 100$
3	$50 \leq GV < 70$
4	$35 \leq GV < 50$
5	$GV < 35$

4.3. Táblázat: Sebességváltó fokozatszámítás értéktartományai

4.3.4. Térkép

A térkép megvalósításában két feladatra figyelek, legyen képes mutatni az aktuális helyzetemet és legyen képes kijelezni egy megtett útvonalat. Erre a leaflet bővítményt találtam a legkönnyebben kezelhetőnek, tehát erre esett a választás.

4.3.5. Zenelejátszó

Mivel a programom webfejlesztési nyelvben írom, egy kliensre, ezért számos lehetőség közül tudok hanglejátszót választani, azonban a könnyű kezelés érdekében szinte az első kézenfekvő megoldásnál maradtam. Tehát a zenelejátszást egy HTMLAudioElement típusú osztály segítségével valósítom meg, aminek a kezelése viszonylag könnyű és ehhez is létezik rengeteg példaprogram. Magát a lejátszási listát és az USB felismerését, természetesen én írom meg, viszont a lejátszásra segítségül veszek egy kódrészletet. [11]

A zenelejátszóban szerepelni fognak hangerő állító gombok, egy lista, amin szerepel az összes zenénk, és egy lejátszási lista, amin szerepelnek a lejátszásra váró számok.

5. A TERVEZÉS SORÁN VÉGZETT MUNKAFÁZISOK ÉS TAPASZTALATAIK LEÍRÁSA

5.1. Hardverelemek kiválasztásával kapcsolatos tapasztalatok

5.1.1. Vezérlő egység

A tervezés során elsősorban a megfelelő fejlesztői környezetet és a fejlesztőeszközt kellett kiválasztanom. Ezen belül elsősorban a Raspberry Pi 4B lett kiválasztva fejlesztőeszközként, mivel a munkámhoz elengedhetetlen egy grafikus kijelzőt megléte. Ugyan, ezt meg lehetett volna oldani leegyszerűsített GUI elemekkel, egy kisebb vagy ugyanekkora Arduino támogatott kijelzőn, bár ez a megoldás egy sokkal rezponzívabb visszajelzést ad és jobb érzés használni, nem is beszélve a megvalósítható kijelző elemekről.

5.1.2. Kijelző

A számítógép kiválasztásának szempontjait ismertetve, egy ehhez megfelelő méretű kijelző kiválasztására esett a hangsúly. Esetemben ez a választás, a számítógép kiválasztása után az alapértelmezett Raspberry Pi 7"-os LCD kijelzőre esett. Szóba került egy más rögzítési módokat támogató érintőképernyő is, amely nem rendelkezik ekkora kerettel, és készíthető hozzá akár nyomtatott, vagy fa keret, és ellátható akár mellékes gombokkal is, azonban emellől a rögzítőpontjai elhelyezkedése miatt elpártoltam.

5.1.3. GPS modul

A Raspberry Pi 4 megvétele után lehetőségem volt letesztelni a kiválasztott GPS modult, amely szinte egyből megfelelően működött egyszer. A tesztelt modul egy Arduinoval is kompatibilis Adafruit GPS modul, amely pontos helyadatokkal szolgált, így valós időben autóban is megfelelő lehetett volna, azonban ennek a működéséhez mindig szabad égre van szükség, tehát sajnos szélvédő mögül sem működik. Ezért esett a választás egy USB-s változatra, amit akár a műszerfal alá beragasztva is lehet használni a szélvédő alatt.

5.1.4. OBD modul

Az autóval folytatott OBD2 kommunikáció érdekében először egy Bluetooth-os ELM327 OBD olvasóval próbálkoztam. A terv az volt, hogy vagy sikerül felcsatlakozni a Pi-ről Bluetooth segítségével kipróbálás végett, és utána soros kommunikációra fizikailag a Bluetoothot eltávolítva átalakítom. Az eszköz elsősorban telefonnal megfelelően működött, viszont a Raspberry Pi-ről a Bluetooth csatlakozás után nem ismerte fel az eszközt. Ezt megpróbáltam orvosolni, hogy fizikailag bekötöm, azonban a modul nem olyan megoldással volt Bluetooth-hoz csatlakoztatva, amit könnyen vissza tudtam fejteni, tehát idő hiányában ezt a megoldást hanyagoltam, és egy kicsivel drágább, de egyből működő USB-s változat mellett döntöttem.

5.1.5. File elérés

Az autóba beépítve továbbá szeretnék hordozható eszköz támogatást, hogy azt behelyezve médiatartalmakat lehessen lejátszani. Ez alatt értve, egy egyszerű USB hosszabbítót kivezetek a Pi egyik USB portjából. Ennek hátránya és veszélye lehet, hogy mivel a Pi ismeri fel az USB eszközt ezért az lehet beviteli és kommunikációs eszköz is. Akár csak egy billentyűzetet rákötve a rendszerből ki lehet lépni és más felhasználási módot találni, de ezekre a veszélyekre és kezelésükre munkámban nem térek ki, viszont így az említettek alapján tesztelésképpen akár egy billentyűzet és később igény szerint akár USB elosztó is csatlakoztatható.

Lehetett volna továbbá SD kártyás és Bluetooth-os megoldást is használni azonban ehhez egy SD kártya olvasóra is szükség lenne, a Bluetooth-hoz pedig jelen pillanatban nem tudtam megfelelő könyvtárat, illetve megoldást találni, hogy a pivel működjön. Ennek ellenére használhattam volna egy külön csak Bluetooth audio modult, azonban erre nem szerettem volna költeni, és olcsóbb megoldások pedig érezhető késleltetéssel üzemelnek.

5.1.6. Energia ellátás

Ebben a részben elsősorban a megvalósítást egy AtTyni85 mikrokontroller segítségével terveztem, amit igény hiányában végül kihagytam a tervből. Magam által tesztelve rájöttem, hogy a Raspberry Pi megszünteti a GPIO jeleket kikapcsolása után, így ezek megfelelően használhatóak kikapcsolás jelzésére. Ezek után áramellátásként egy kisebb STDN-3A24 step-down modult szerettem volna használni, ami az autóban bekötve feladta a szolgálatot, melynek oka lehetett egy nemkívánatos vezeték, vagy szabadon hagyott elemek által okozott rövidzár. Ugyanakkor, a modul előnye lett volna, hogy a lapkáról vezérelhető az áram áteresztése és így az nem igényel egy külső MOSFET-et, de végül egy, az eredetihez nagyon hasonló megoldás mellett kellett döntenem.

5.2. Az Operációs rendszer kiválasztása

A megfelelő szoftver és fejlesztőkörnyezet kiválasztása egy fontos feladat, hiszen mind a szoftvernek és a kiválasztott nyelvnek és fejlesztőkörnyezetnek támogatnia kell az általunk kívánt felhasználási lehetőségeket, és elvárásainkat.

Legelsőként, mivel a Raspberry Pi egy viszonylag erős mikroprocesszorral van ellátva, így képes Linux disztribúciók futtatására. Lehet szó médiaszerverről, karakteres felületről, vagy Retró Játék felületről, ezeknek mind létezik támogatása. Esetemben a kiválasztott disztribúció, a kifejezetten Raspberry Pi-ra készült Raspberry Pi OS lett, ami egy 32 bites grafikus felületet támogató operációs rendszer.

5.3. Nyelv és a Fejlesztőkörnyezet kiválasztása

5.3.1. Python

Mivel ez egy Linux disztribúció, ezért szinte minden Linuxra és kereszt platformra írt szoftver megtalálható. Elsőre egy megfelelő fejlesztési környezet után kutatva a Python és azon belül is a PyQt bővítményre találtam, amivel egy Cross Platform ablakos applikációt lehet készíteni. Persze vannak olyan platformfüggő modulok, amik csak a Pi-n működnek GPIO vezérlésként, de nem is lenne másra szükség, és egyszerűen lehetne Windowsról is fejleszteni. Ezzel csak az a problémám adódott, hogy nem találtam eléggé reszponzívnak a felületét egy kisebb tesztelés után a Pi-re leportolva, és a fejlesztése is nehézkesen ment, mivel ezeket a környezeteket, elég ritkán használtam, közelítőleg soha tanulmányaim során. Pythonon belül még megemlíteném, hogy vetettem egy pillantást a PyGames könyvtárra is, de ez főként játékok egyszerű fejlesztésére van kitalálva és kevésbé támogatja az alap ablakelemeket, amelyeket én szerettem volna használni.

5.3.2. Qt

Ezután mivel a Python miatt megismerkedtem a Qt-val, ezért erre vettem egy pillantást, és meg is próbáltam, milyen alkalmazásokat lehet vele a Pi-re fejleszteni. Qt-n belül eléggé ígéretes beágyazott eszköz rendszereket lehet fejleszteni, amelyek rendelkeznek valamilyen grafikus felülettel, és igen széleskörben alkalmazható az iparban, akár okosfogkefén is. Viszont nehézkesnek bizonyult a mostani fejlesztőeszközök legális beszerzése, és internetről mókolt megoldással a Raspberry-ről futtatva lehetett volna csak Windowsról fejleszteni. Ezt azért vettem el, mert a Pi-nek azért egy asztali géphez képest viszonylag gyenge processzora van, és elég sok időbe telhet egyes esetekben a fordítás. Ezen felül próbálkoztam a Qt operációs rendszerrel is, ami sajnos az én eszközeimen egyáltalán nem indult el, amivel lehetett volna távolról is fejleszteni interneten keresztül. Később rájöttem, hogy mind ez, mind az ezután említett Windows 10 IoT is csak a Raspberry Pi 2, 3 és 3B eszközökre elérhető.

5.3.3. Windows 10 IoT

Vettem egy pillantást a Windows 10 IoT világra, ahol a leírás szerint támogatják a Raspberry Pi-t, legalábbis a 3B verziót, viszont számottevő próbálkozás után nekem nem sikerült ezt sem elindítanom az én 4B eszközömmön. Hozzáteszem létezik még Microsoft Cross Platform megoldás a .NET Core UWP applikációként, azonban ehhez nem találtam elegendő segédanyagot, az armv7l-re készült Windows verziókkal pedig nem próbálkoztam, viszont később megtudtam, hogy létezik Windows 10 Enterprise a Raspberry Pi-ra és lehet, hogy ez fel is fogja váltani az imént említett Windows 10 IoT verziót. [12] [13]

5.3.4. Angular - Electron

Végezetül mivel van tapasztalatom a JavaScript és TypeScript nyelvekkel, ezért egy korábbi beadandóból eszembe jutott, hogy elhangzott a megfogalmazás, miszerint az Angular (vagyis Type- és JavaScript) kódot is lehet közvetlenül Cross Platform „fordítani”, amit számos alkalmazás használ is. Vegyük példának a Visual Studio Code, Discord, vagy MsTeams alkalmazásokat. Tehát, mivel ebben is és a HTML, CSS-ben is jártasabb vagyok ezért ez elég kecsegtetőnek bizonyult. Futtató környezetnek végül az ElectronJS-t választottam, ami támogatja a Raspberry Pi architektúráját is.

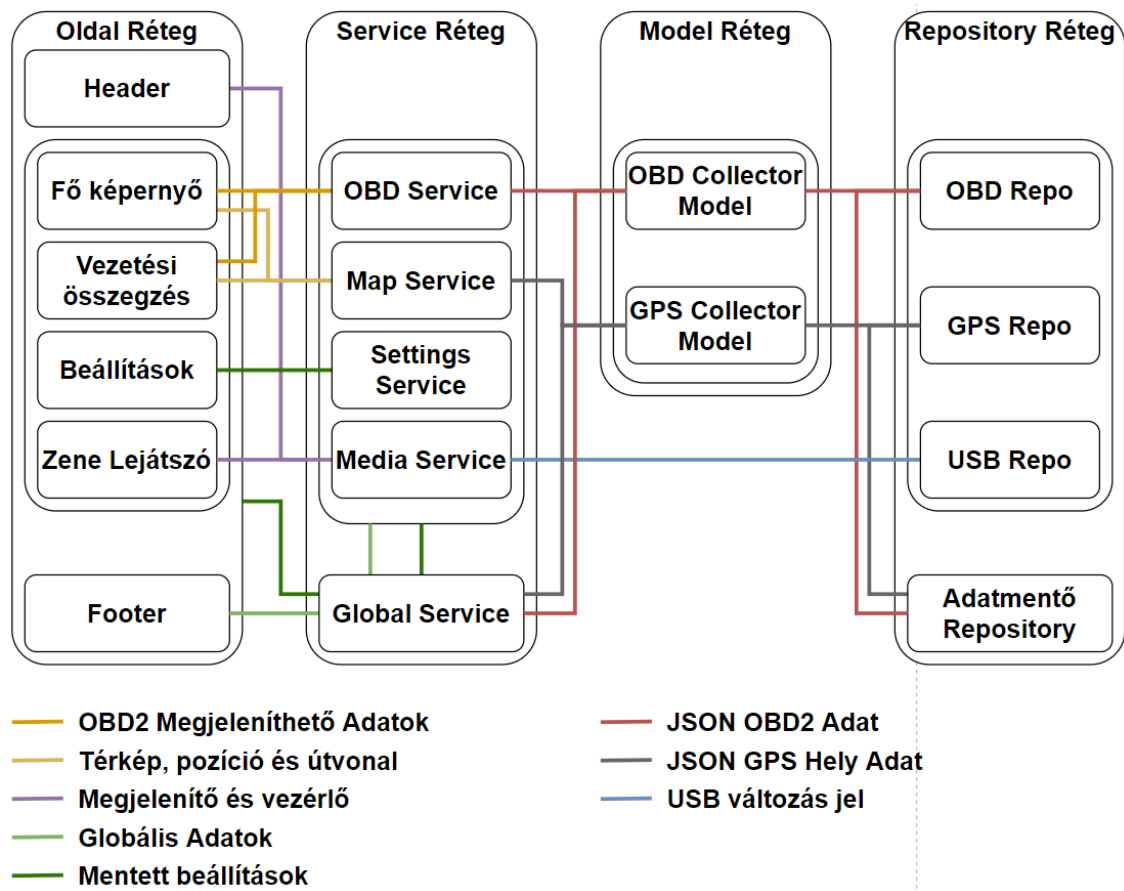
Végősorban meg szeretném jegyezni az Electron és a NodeJS-el kapcsolatban, hogy ezek a rendszerek sokkal erőforrás és memória igényesebbek, mint például egy rendesen platformra lefordított programkód, hiszen itt elsősorban az Electron fut a Node-on belül, és magát a végleges applikációt/webappot az Electron futtatja egy Chromium webböngészőn belül, ami végül is megjelenik nekünk interakcióra. [14] Ezért belátható, hogy lényegesen lassabb lehet, mint egy C++, vagy C#-ban megírt program, viszont egy rezponzívabb és könnyebben szerkeszthető felületet ad, mint például a Python és a PyQt.

5.4. Fejlesztési folyamat

A fejlesztési eszközök, környezet és nyelv kiválasztása után ebben a részben foglalom össze a fejlesztés során felmerült észrevételeimet és tapasztalataimat.

5.4.1. Felépítés

Elsősorban meghatároztam, hogy milyen szerkezetre szeretném felépíteni az alkalmazásomat, mik voltak az elvárásaim és hogy milyen funkcióim lesznek. Ezekhez a funkciókhoz később először oldalakat és megjelenítéseket, majd Vezérlőket, Kiszolgálókat, Modelleket és Osztályokat, típusokat kötöttem.



5.1 Ábra: Szoftver felépítési séma és a rétegek közötti adatvándorlás

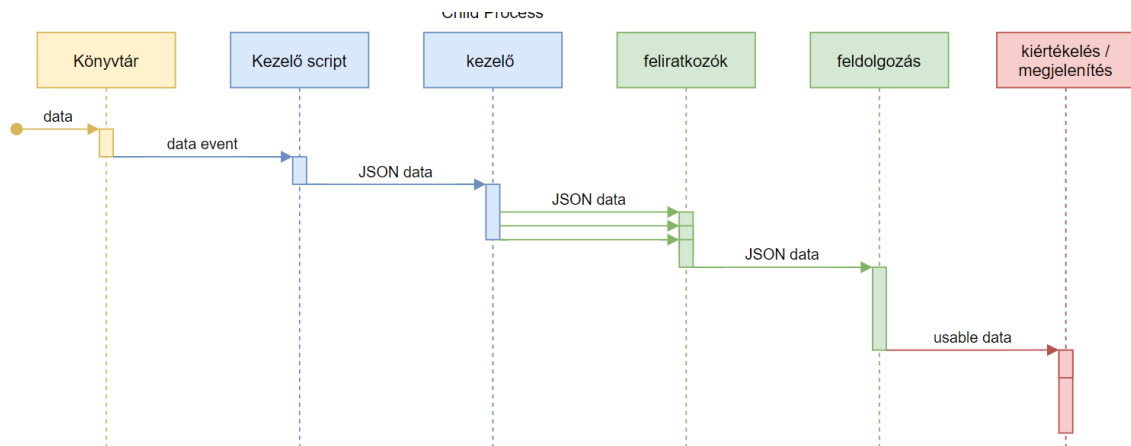
A felépítésem alapvetően a Részletes Specifikációk fejezetben megadottak szerint alakítottam, helyenként kisebb eltérésekkel.

5.4.2. Beállítások

Igyekeztem a lehető legtöbb szoftverelemet személyre szabhatóvá tenni, legalább egy kis, de jelentős mértékben, hogy az a felhasználónak igénye szerint, személyre szabottan nézhessen ki. Ebben főként a színvilág és a mértékegységek lettek állíthatóak.

5.4.3. Szolgáltatás elérés

A következő (5.2 Ábra: Külső szolgáltatások folyamatára. ábra) képen azt ábrázolom, hogy milyen fázisokon mennek keresztül az egyes külső scriptekkel megvalósított szolgáltatásaim. Mivel a kezelő az tulajdonképpen nem változik, ezért szinte ugyanaz a módszer alkalmazható mindegyik script meghívására. Ilyen formában kerülnek lekérésre az OBD-s adatok, a GPS pozíciónk és az USB csatlakoztatás érzékelése.



5.2 Ábra: Külső szolgáltatások folyamatára

5.4.4. OBD2 Lekérdezés és kiíratás

Az OBD2-es adatok lekérdezéséhez az előzőekben említett szolgáltatáselérési sémát használok, amiben az 'obd2-over-serial' npm könyvtárat használok. [15] Erre a megvalósításra, vagyis külsőszervezésre azért volt szükség, mert a könyvtár által és még pár helyen használt 'serialport' könyvtár és az Electron által használt NodeJS modul verziószáma nem egyezik meg. Ez azért fordul elő, mert az Electron nem szokványos NodeJS modulverziókat használ, és ezeknek a beszerzése nehézkes. Ezen belül létezik az Electronnak saját fordító bővítménye (electron-build), amellyel ezt képesek vagyunk a megfelelő verziószámra fordítani sajátkezűleg, azonban mivel a 'serialport' egy tulajdonságából adódóan erősen hardverközelítő bővítmény, így nem fut el más platformokra építve, és az Electron-build nem támogatja az Armv7l architektúrát, tehát kénytelen voltam ezt és a hozzá hasonló folyamatokat egy gyermekfolyamatba szervezni. Ez lekéri nekünk a konfigurációs fájlban beállított szükséges PID-eket („Parameter IDs”) és JSON formátumban továbbítja azokat az általános kimenetére.

A beolvasott adatok után már csak fel kell dolgoznunk őket és eltárolni a megfelelő formátumban, hogy ellenőrzésre megmaradjanak. Erre elsősorban sorokra darabolom a beérkező szöveget, annak sorain végig haladva azokat változóba fordítom, letárolom és megjelenítem a főoldalon. Ezek után hajtódnak végre a különböző kiértékelések, amik az elemzésekhez és a vezetésértékeléshez vezetnek.

5.4.5. GPS helyadat elérés, mentés és kiértékelés

GPS helymeghatározásra egy 'gps' elnevezésű npm könyvtárat használok, amely lefordítja számomra a GPS modul által kapott adatokat értelmezhető formátumba, majd ezeket a megfelelő helyekre továbbjuttatva megváltoztatja pozíciónkat a térképen, tárolja azt, majd időközönként GeoJSON formátumban is lementi. [16] Ezt a megvalósítás GPX fájlformátumúnak indult, azonban, mivel a térképen való megjelenítéshez azt át kellene alakítani GeoJSON formátumban, ezért végül kizárólag a GeoJSON mellett döntöttem. Ennek előnye, hogy kevesebb műveletbe kerül a beérkezett adatot átalakítani és tárolni is, továbbá egyre elterjedtebb útvonalkövető formátum, és rengeteg rajzoló formát támogat.

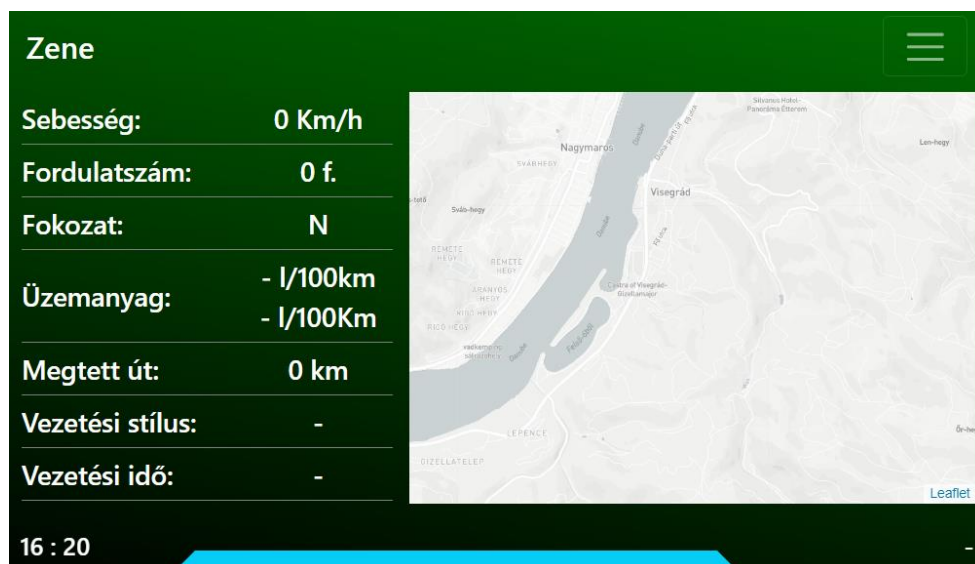
Továbbá ide tartozik az is, hogy az aktuális rendszeridőt a Raspberry Pi elmenti, viszont áramforrás hiányában nem számolja tovább a kikapcsolása után. Ezért a rendszer indításakor, az elsők között megkapott GPS UTC időt rendszeridőként állítom be az óra pontos működéséért. Ez okozott gondot a vezetési idő számításában, mert annak az indítása előbb kezdődik, mint ahogyan beállítanánk az időt, viszont ez egy szimpla nullázással, avagy frissítési metódushívással az óra állításakor orvosolható.

6. A MEGVALÓSÍTÁS LEÍRÁSA

A megvalósítás korai szakaszában a fejlesztést, a Főképernyő és az Applikáció képernyő vázlatos megvalósításával kezdtem. A továbbiakban a képernyőkre oldalakként is referálok, mivel az applikáció elsősorban webfejlesztési technikákkal készül el.

6.1. Főképernyő

Ez az oldal jelenik meg elsőre az autó elindítása után, tehát ezen az oldalon látszódnia kell és elérhetőnek kell lennie minden lényeges információnak, amely éppen zajlik, hogy vezetés közben ne kelljen váltogatni az oldalak között, hiszen ez elvonhatja a vezető figyelmét, és balesetveszélyes lehet. Ezen vizsgálatokba nem mentem bele, vagyis nem figyeltem, hogy mennyire befolyásolhatja az alkalmazás a vezető vezetési képességét, de igyekeztem minél kényelmesebbé és könnyebbé tenni a használatát és az adatok elérhetőségét.



6.1 Ábra: Kezdőképernyő megvalósítása

6.1.1. OBD2-es adatok

Ez a képernyő felel a begyűjtött OBD2-es adatok megjelenítéséért a látható térképért, és az esetleg futó médiatartalom részleges megjelenítéséért.

Az OBD-s adatokat A Raspberry Pi /dev/ttyUSB0 soros portjáról kérem le, amihez az 'obd2-over-serial' bővítményt használom. [15] Ezt ki kellett szerveznem egy Node Child processbe, mert az egyik használt bővítménye a 'serialport' nem támogat cross platformot és fordítási nehézségeiről a 5.4.4 fejezetben részletesebben beszámolok, tehát lényeges ráfordított idő után kiszerveztem, és összekötött IO-val megfelelően működött.

6.1.2. Térkép

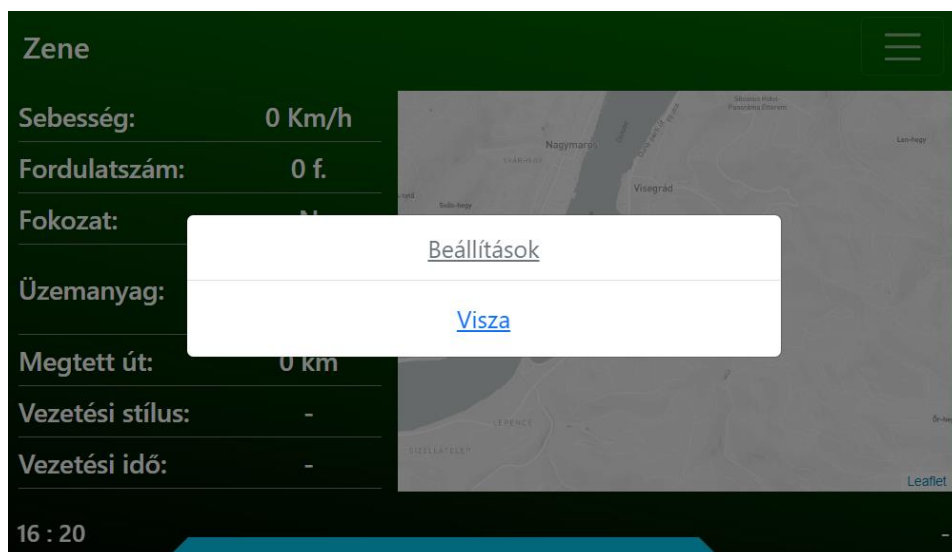
A térképen mindig az aktuális pozíciónk látszódik, amelyre rányomva ez nagyban jelenik meg. Ez adatait közvetlenül a GPS modultól kapja és mindig a legújabb beolvasott pozícióra ugrik. A térképhez továbbá internetelérés is szükséges, mivel a textúrákat szerverről tölti le, de ez a későbbiekben szeretném offline térképpel, vagy textúrákkal helyettesíteni.

6.2. Keret

Keret alatt értem az alul és felül elhelyezkedő képernyőelemeket, ami akár mindegyik oldalon megjeleníthető. Mivel ez a projekt jelenleg főként 4 oldalnak készült ezért testreszabhatósága egyelőre a bal felső gomb funkciójára, és nevére szűkölt. Azonban a későbbiekben a lenyíló menüt akár állítható és megadható részelemekkel is lehetne bővíteni.

6.2.1. Fejléc

A fejléchez tartozik egy Multifunkciós gomb a bal felső sarokban, illetve jobb oldalon a lenyíló menü gombja. Ebben a lenyíló menüben találjuk a beállításokat és bekapcsolt „Fejlesztői” mód mellett az alkalmazásból való kilépési lehetőséget.



6.2 Ábra: Lenyíló menü megvalósítása

6.2.2. Lábléc

Ide kapcsolódik a megjelenő idő, amit az alkalmazás a Raspberry Pi rendszer idejéről kérdezi le másodpercenként a pontosság tartása érdekében. Illetve az aktuális hőmérséklet, amit az autótól kapunk meg az OBD adatokat lekérve, a motorba beáramló levegő hőmérsékletét. A hőmérsékletnél lehetőségünk van a beállításokban meghatározni, hogy milyen mértékegységeket szeretnénk látni. Legutolsó sorban pedig középen láthatjuk az aktuális vezetési statisztikánkat színes formában.

6.3. Zenelejátszó

A zenelejátszó nevéhez hűen a zenetartalmak lejátszására szolgál. Egyelőre csak USB-s zenelejátszást támogat, amit a későbbiekben bővíteni szeretnék. Ránézésre a GUI leegyszerűsített és minden fontos dolog kézközelben megtalálható.

6.3.1. Kezelőfelület

A felületen balról jobbra haladva elsőként a hangerő szabályzó gombokat látjuk. Ezekkel a hangerőt állítva a hangerő a képernyő tetején jelenik meg a színsémához illően. Továbbá alattuk található a Keverés és az Újrakezdés gomb is.

Felül visszaléphetünk a főoldalra, megnézhetjük az eszköz által látott zeneszámainkat, és megnézhetjük az aktuális lejátszási listánkat. Ez a sáv nem használja a keret által nyújtott fejléct, hanem azt egyedileg valósítja meg az imént felsorolt gombok miatt.

A felület közepére tekintve legfelül az aktuálisan lejátszásra kerülő szám címe látható, amely alatt a kezelőfelületek találhatók. Elsőként balra a lejátszás ideje aztán egy állítható csúszka és végül a zeneszám időtartama látható. Ezalatt a gombok balról jobbra haladva:

- Előző szám (10mp felett előről kezdés)
- Vissza tekerés
- Lejátszás/Szünet
- Előre tekerés
- Következő szám



6.3 Ábra: Zenelejátszó megvalósítása

6.3.2. Zeneszámok

A következő egy legördülő lista, amin szerepelnek az eszközünkön szereplő zeneszámok, illetve két gomb azoknak sorban, illetve keverten történő lejátszására. Ezen felül minden egyes zeneszámot külön-külön a lejátszási listára helyezhetünk akár több alkalommal is.



6.4 Ábra: Zeneszámok kiválasztása

6.3.3. Lejátszási lista

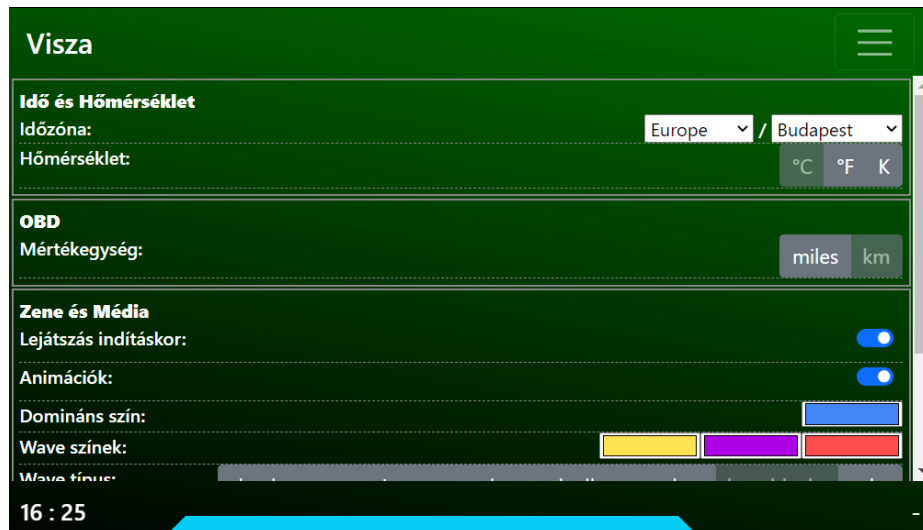
Ezen a listán jelennek meg az aktuális lejátszási sorban szereplő zeneszámok olyan sorrendben, amelyben le fognak játszódni. A listáról tudunk számokat törölni, illetve közöttük ugrálni és váltani. A számok keverésére csak a főoldalon és a Zeneszámok között van lehetőség. Ennek a lehetősége a véletlen félrenyomások elkerülése érdekében ebben a menüben nem található meg.



6.5 Ábra: Lejátszási lista

6.4. Beállítások

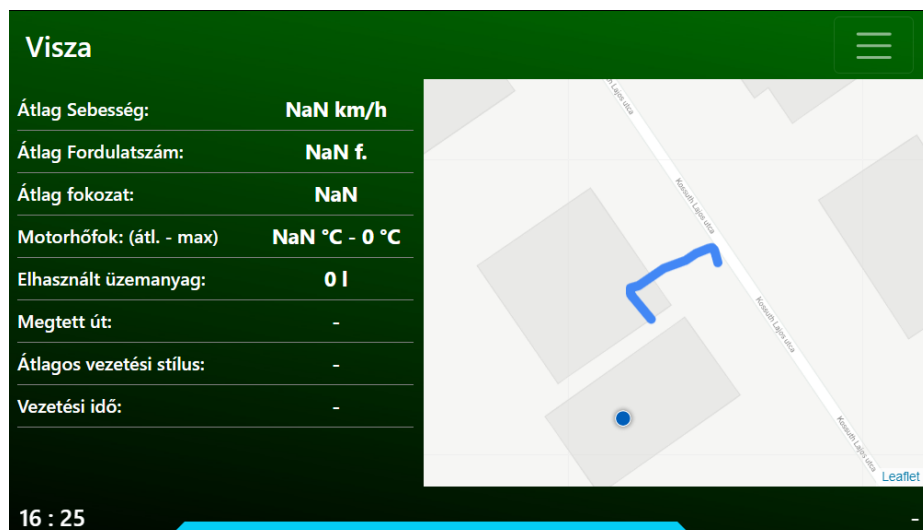
A program beállíthatósága és személyre szabhatósága majdnem legutolsónak maradt, de ez sem maradhat ki a teljes képből. A felületen elsősorban az alap programnak a beállításai elérhetőek, köztük a mértékegységek, az időzóna és a színek kompozíciók. A színek állításánál lehetőségünk van minden beállításnál néhány alapértelmezett szín közül választani, de választhatunk akár színkód, vagy skála alapján is.



6.6 Ábra: Beállítások megvalósítása

6.5. Összegzés

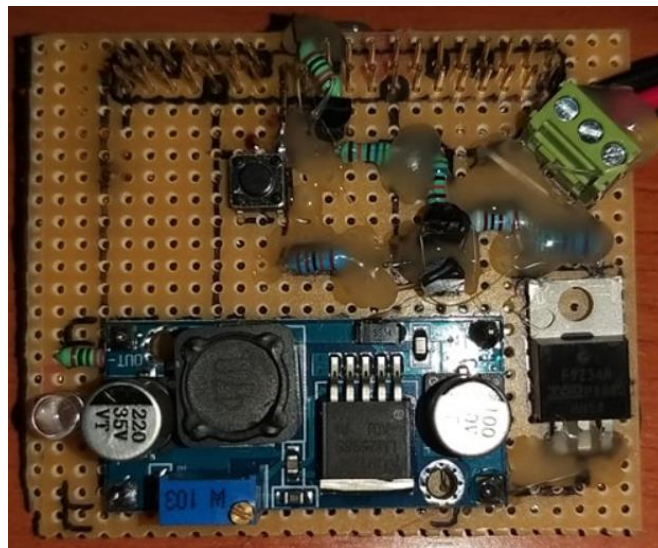
Ez a képernyő a jármű leállítása után jelenik meg. Bal oldalon találhatóak a begyűjtött adatok végleges átlagai, a megtett út és az eltelt utazással töltött idő. A kép jobb oldalán a megtett út látható térképre rajzolva, amire rányomva az teljes képen jelenik meg.



6.7 Ábra: Összegzés megvalósítása

6.6. Áramellátás, kikapcsolás

Az áramellátást végül a tervezettek alapján egy sajátkezűleg megépített modul szolgáltatja, melynek a megépítése nyáklapon valósult meg. A végleges áramkör egy I/O shield lett, ami a Raspberry Pi-re helyezve, annak áramot szolgáltat. Az eredeti terve szerint ez kábellel csatlakozott volna a Pi-hez, azonban ezt átgondolva és egy ezt megelőző szerencsétlen esetek következményeként, ezt úgy alkottam meg, hogy minél kevesebb esélye lehessen mozogni, és rövidzárlattal érintkezni. Ebből kifolyólag a lapka hátsó oldalát házilag szigeteltem is. A képen (6.8. ábra) továbbá látható, hogy az áramkört tartalmazó lapkának a 4 sarka üresen lett hagyva. Ennek az az oka, hogy a lap távtartókkal ellátható legyen, amik megtartják azt a Pi felett, hogy azt ne az I/O tűskéknek kelljen. Az áramkör üzemben tartására a 16-os(GPIO23), az ACC jel érzékelésére pedig a 18-as(GPIO24) tűskéket használtam. Az áramot a 2-es, a földelést pedig a 39-es tűskén kapja.



6.8 Ábra: Áramellátás fizikai megvalósítása

6.7. Fizikai beépítés

Fizikailag a rendszernek az autóba való beszerelésre kezdetben egy fadobozban valósult meg. Az OBD olvasót a helyére bedugva, annak kábelét a műszerfal mögött vezettem fel a rádió mögé, míg a GPS modult a műszerfalban ragasztottam közvetlen a szélvédő alatti részre fejjel lefelé és végül kivezetésül hagytam egy USB kábelt, hogy egy pendrive-ot tesztelés jellegével be lehessen dugni a zenelejátszás miatt. A fadobozt pont egy 2 DIN-es rádió méretére készítettem, és beillesztve az autóba pontosan passzolt a helyére. A dobozt a helyén csavarral nem rögzítettem, mert ez még csak egy kezdetleges verzió és szeretném egy gyári alumínium kerettel és 3D nyomtatott elemekkel helyettesíteni. Az eszközök ebben beragasztva kapnak helyet, és ebben is valósul meg az autó áramára és hangfalaira való csatlakoztatás egy szabványos csatlakoztató segítségével.



6.9 Ábra: Kijelző dobozzal

A kijelzőt és a fedelet egy lábakkal ellátott falpra csavarokkal rögzítettem és ezt helyére tolvá látja el funkcióját. A rendszer az utóba behelyezve a következő képeken látható:



6.10 Ábra: Csatlakozó kábelek



6.11 Ábra: Keret az autóban



6.12 Ábra: Kijelző rögzítési keret

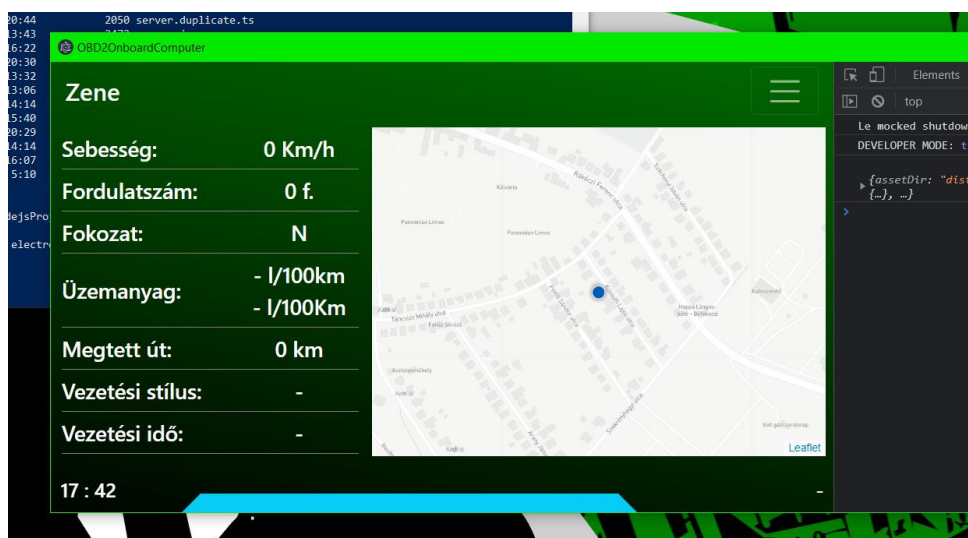


6.13 Ábra: Kijelzővel az autóba beszerelés után

7. TESZTELÉS

A tesztelés első fázisában, mivel Angular-ban írtam a munkám, weben kezdtem, hogy láthassam élőben milyen változásokat érint a kód, amit írok. Ez nagy segítség volt a GUI megformálására, hiszen nem kell folyton újra fordítanunk a teljes kódot, hanem az automatikusan újra fordítja a legújabb mentett kódrészletet. Ezt a fájlműveletek bevezetéséig tudtam folytatni, mivel az Angular egy elsősorban webfejlesztésre szánt nyelv, és a kliens számítógépen fut, böngészőben, ezért nem lehet belőle file inputon kívül a számítógép tartalmát olvasni. Ezért a fileműveletek bevezetésénél nem szerveren futtattam tovább, hanem electronban nyitottam meg, aminek a változott kódot a háttérben továbbra is folyamatosan újra tudtam fordítani a megfelelő beállítások után. A tesztelést a továbbiakban az is könnyítette, hogy az egyszer legenerált kódot azt könnyen át tudtam emelni egy az egyben a Raspberry Pi-ra is, mivel a generált JavaScript állományokat az Angular egyetlen dist/ mappában tárolja, így azok szabadon hordozhatók, és elegendő egyszer a Pi-ra generálni a teljes futtatási könyvtárat és keretrendszert.

7.1. GUI tesztelés



7.1 Ábra: GUI tesztelése a fejlesztőgépen

Első körben az oldalakat készítettem el, hogy milyen kinézete és milyen funkciói is lesznek az alkalmazásnak. Ezen belül elsőre a főképernyő készült el, amin helyet foglalnak az OBD-s adatok és a térkép. Ekkor még nem volt mögötte logika, tehát csak grafikailag győződtem meg róla, hogy minden megfelelő és látható. Ezután készült zenelejátszó, amit a beállítások, a vezetés utáni áttekintés és egy nagy-térképes oldal követett.

Ezen részek tesztelésnél azt a szempontot vettem alapul, hogy minden GUI elem neki rendeltetésszerűen funkcionáljon, könnyen kezelhető legyen, és ne legyen rajtuk a felbontás miatti összenyomott és képből kieső elemek. Ezt asztaligépen az electronban lévő böngésző fejlesztőpaneljének megnyitása után, a megfelelő felbontást beállítva teszteltem. (Lásd: 7.1. ábra)

7.2. Logikai tesztelés

A kinézet elkészülte után a logika következett. Ennek tesztelése sem maradhatott el, melyet elsősorban az Angularon belül Karmában szerettem volna megvalósítani, majd végezetül mockolt adatgyűjtőkkel tettem. Ennek az volt az oka, hogy mivel nem az Angular alapértelmezett szerverével fejlesztettem, futtattam a programot, és mivel használok benne fájlműveleteket, ezért az alapértelmezett unitteszteket nem tudtam lefuttatni. Itt próbáltam átállítani a tesztkörnyezet böngészőjét, hogy az az Electronon belül induljon el, de nem jártam sikerrel az én általam használt verziókkal.

A mockolt tesztek úgy végeztem el, hogy a már elkészített adatgyűjtő scripteket számítógépen behelyettesítettem, hogy azok valósnak tűnő adatokon biztosítsanak a megfelelő hardverrel való kommunikáció nélkül. Így sikeresen tudtam tesztelni az OBD és a GPS adatgyűjtését is.

Sajnálatos módon ez a megoldás nem volt elegendő az összes hiba biztos kiszűrésére, tehát ezek fizikailag az autóban történő tesztelés után kerültek elő, de nagy segítséget nyújtott, és valódira váltás után a legtöbb elem egyből megfelelően működött.

7.3. Fizikai tesztelés

7.3.1. Működés igazolása

Elsőre az eszközt fizikailag összeszerelve az autóban teszteltem, amiben elsősorban rákötöttem az OBD olvasót, egy USB hosszabbítót, egy Jack-to-Jack kábelt dugtam az autóban elhelyezkedő rádióba, majd az áramot a tesztelés idejére hordozható teleppel biztosítottam.



7.2 Ábra: "Proof of Concept" a rendszer első autóba kerülése

7.3.2. Áramellátás tesztelése

A végső teszt előtt még beiktatásra került egy egyszerű autóra csatlakoztatása az áramellátó modulnak. Ugyan ez a modul elkészülése előtt, közben és után is tesztelésre került egy 12V-os zselés akkumulátorral, viszont az első próbálkozásra ezen lépést

kihagyva a teljes rendszer feladta a szolgálatot. Tehát hibáimból tanulva az áramkört mind próbapanelen, összeforrasztás után, és az autóba csatlakoztatás után egy multiméterrel teszteltem. Az elkészült áramkört továbbá elsőnek egy olcsóbb mikrokontrollerrel is teszteltem, hogy az működése közben az elvártaknak megfelelően üzemel. A tesztek után az elkészült végleges modult megfelelőnek találtam és ezáltal elfoglalhatta végső helyét a Raspberry Pi-on.

7.3.3. Végső teszt

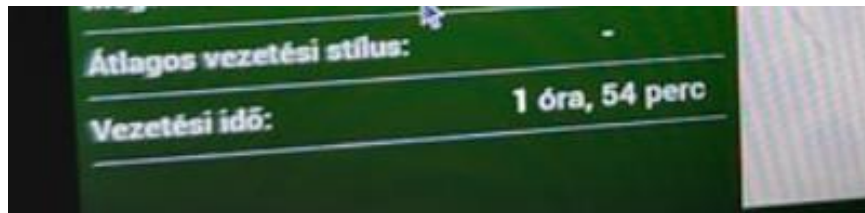
Legutolsó sorban az elkészült rendszert a helyére illesztve tesztelte, ami szinte egyből megfelelően szuperált. Kezelésileg nem találtam rajta különösebb gondot, mind az OBD-s és a GPS hely-adatokat menetközben gyűjtötte, és rendeltetésszerűen felismerte a bedugott pendrive-ot is. Hibái ugyan adódta, elsőre, halk volt a zene hangereje, amin SSH-n keresztül az 'alsamixer' használatával lehetett állítani.



7.3 Ábra: A rendszer működésének bemutatása

Hasonlóan a térképnek a nagyítása sem volt egyből megfelelő, amit ugyanígy SSH-n keresztül az alkalmazás konfigurációs fájljában lehetett állítani. Ez előfordult mind a vezetés közben látott panelen, ahol túl kicsinek látszódtak az utcák, és mind az összesítés képernyőjén, ahol a kevés nagyítási határok miatt, a teljes útvonal nem fért bele a megjelenített térképrészletbe.

Ezekén felül előfordult, hogy az általam GPS által beállított idő miatt, a feljegyzett vezetési idő valótlan értékeket mutatott, amely azért volt lehetséges, mert a vezetési időt az indulás óta eltelt idővel számolom. Ugyan alkalmanként ez segített tesztelni azt, hogy az időformátumokkal jól számolja az értéket, viszont ezt a későbbiekben csak a rendszer frissítésével tudtam javítani.



7.4 Ábra: Hibás vezetési idővel ábrázolt, huzamosabb üzemidők kijelzése

Észrevételre került még, hogy az általam írt sebességváltó kisebb sebességeknél előszeretetten becsüli magát félre, illetve ezen felül a begyűjtött adatok hiányosságából adódóan a vezetési stílus képes menet közben is lehetetlen értékek felvételére. Azonban mivel az utóbbiakban felsoroltak előfordulása kellően ritka és mérsékelt, ezért a dolgozatban nem foglalkoztam a javításukkal.



7.5 Ábra: Az összegzés kijelzésének bemutatása

Mindezek után, az autó leállítását követően megjelenik az összegzés, ami 15-20 másodpercig látható, majd ezután a Raspberry Pi megfelelően leáll. Továbbá az áramellátás lekapcsolása hallható az autó hangszóróin is, mivel kis idő elteltével a rajtuk hallható statikus zaj megszűnik. Végző sorban a rendszer az autó akkumulátorát sem merítette le, tehát áramának lekapcsolása sikeresnek mondható és a következő elindulásnál is rendeltetésszerűen működik.



7.6 Ábra: Főképernyő nappali működés közben



7.7 Ábra: Zenelejátszó nappali működés közben

8. AZ EREDMÉNYEK BEMUTATÁSA, ÉRTÉKELÉSE, HASONLÓ RENDSZEREK EREDMÉNYEIVEL ÖSSZEVETÉSE

Az elkészült eszköz az autóba beépítve, követi a helyemet és kijelzi az autónak a valós idejű adatait is. Az elkészült rendszer minden oldalán vizuális visszajelzést ad az aktuális vezetési stílusunkról és a kinti hőmérsékletről, továbbá lehetőséget biztosít zene lejátszásra is. Az út végén összesítést az átlagos sebességünkről, fordulatszámról, fogyasztásról és megtett útról és utazási időről is. Ezen felül szövegesen kijelzi, hogy súlyozottan melyik vezetési stílusunk fordult elő a legtöbbször. A kijelzett adatok mellett megtekinthetjük a megtett útvonalunkat, amelyre rányomva az betölti a teljes képernyőt.

A rendszert használhatónak tartom, mivel az általam elképzelt kritériumokat teljesíti, tehát mutatja a valós idejű vezetési adatunkat és a stílusunkat is. A zenelejátszó használható és személyre szabható, illetve a térkép is tetszetős. Ugyanakkor a rendszer hiányosságai is szembe ötlőek lehetnek, hiszen a térkép megjelenítéséhez internetelés szükséges, a zenelejátszó pedig csak USB-a adathordozón keresztül képes zeneszámokat lejátszani.

Hasonló rendszerekhez képest, amelyeket munkám során találtam, melyeknek egy része elő is fordul az irodalomkutatásban, viszont egy számomra sokkal megnyerőbb grafikus interfésszel rendelkezik, ugyanakkor fizikai kinézete jelenleg nem a leg megnyerőbb és téteznék ennél szebb megvalósítások is. Erre példaként megemlítem az M3-Pi munkát, amelyben ugyan a kijelző kevés adat megjelenítésére képes, viszont az külsőre, mivel külön egyetlen autóba készült, lényegesen megnyerőbb. [3] Ugyanakkor munkámban számottevően több funkció található, például a térkép, vagy a zenelejátszó formájában.

Ezzel ellentétben a gyártók által készített fedélzeti számítógépek komplexitásával, munkám fel sem veszi a versenyt, hiszen azokban rengeteg egyéb funkció is található, mint például a navigáció és a Bluetooth-os zenelejátszás, viszont ez ennek a munkának nem is volt a célja.

9. A MEGVALÓSÍTÁS ELEMZÉSE, ALKALMAZÁSÁNAK ÉS TOVÁBBFEJLESZTÉSI LEHETŐSÉGEINEK SZÁMBAVÉTELE

Megvalósításomban a kezdeti fejlesztésekben a tapasztalatom hiányában rengeteg hibát vétettem. Rengeteg időt fordítottam egyes másodlagos komponensek elkészítésére, amelyeket utána vagy másféleképpen valósítottam meg, vagy fáradtságból kifolyólag figyelmetlenségből nem vettem észre szinte egyértelmű hibáikat.

Mivel autóban tesztelve számomra a GUI áttekinthető és könnyen kezelhető volt, ezért megvalósításomat jónak tartom. A megjelenített adatok a valóságnak megfelelnek, a térkép követi a helyzetemet, és pendrive-ot bedugva majdnem azonnal zenét is tudunk hallgatni. A megjelenített utazási idő és km rendeltetésszerűen számolja annak mért értékét, a vezetési összegzésben a térképre kirajzolásra kerül a megtett útvonal és a beállítható lehetőségek is személyre szabhatóan működőképesek. Tehát a rendszer akár napi használatban is alkalmazható.

Mivel viszont, egy elkészült rendszer szinte sosem tekinthető teljesen késznek és tökéletesnek, ezért mindig felmerülhetnek hiányzó és továbbfejlesztést igénylő komponensek és rendszerelemek. Tehát továbbfejlesztési lehetőség lehet elsősorban a Bluetooth-os, illetve rádiós zenelejátszás is. Ezen felül a lehetőségek közé tudom még sorolni az offline térkép és akár egy applikációs képernyő megvalósítását is. Továbbá a néhány kijelzővel ellátott autórádióban és egyéb projektekben előforduló Android Auto féle telefonhoz való csatlakozás lehetőségét is, de legalábbis egy online médialejátszási platform támogatását, mint például a YouTube Music, a Spotify, vagy a Netflix, illetve egy tekerőgomb megvalósítását a hangerő kényelmes állítása végett.

Ugyanakkor elképzelhetőnek tartom a rendszer szinte teljes újrafejlesztését C++, vagy C# nyelven, a jobb teljesítmény és átláthatóság érdekében.

10. TARTALMI ÖSSZEFOGLALÓ

Ebben a dolgozatban egy autó fedélzeti számítógép megvalósítását tűztem ki célnak, és ennek a megvalósítási folyamatait vezettem végig. Az általam megvalósított fedélzeti eszköz kezdetben egy OBD feketedoboznak indult azonban használati céljai egyre csak bővültek a dolgozat érdekében, és a végeredmény egy majdnem teljesen másik projekt lett. Ebben munkában összesen 5 adatot kérek le a jármű számítógépétől („sebesség”, „fordulatszám”, „befogadott levegő hőmérséklet”, „befogadott légmennyiség” és „motorhőfok”), megjelenítem azokat és a megtett utat, illetve a program képes zenelejátszásra is.

Megvalósításomban kevés használatban lévő mintát vettem alapul, és inkább ötleteket vettem át, minthogy megvalósításokat. A programot TypeScript nyelvben írtam, mivel a választható nyelvek közül ezt ismertem a legjobban és ez tűnt a leg szebben megvalósíthatónak. Ebben az esetben viszont a kész program sokkal több erőforrást használ, mint egy natív nyelven írt társa, hiszen ezt megvalósítva én a programot Angularban írtam meg, Electron platformra, amit utána lefordítva használhatok egy Armv7l alapú Raspberry Pi-on. Ennek a lehetőségére nagy szerepet játszik, hogy a Raspberry Pi egy meglehetően erős számítógép és még a saját böngészőjénél is jobban futtatja az általam írt applikációt.

Sajnálatos módon az általam választott nyelven elég kevesen írnak a Raspberry Pi-ra, vagy akármilyen szimpla mikrokontrollerre igény hiányában és erőforráshatékonysági okokból applikációt, illetve annak egyes rendszerelemeit még a fejlesztőkörnyezet sem teljeskörűen támogatja. Például a serialport npm könyvtárat, ami a soros interfészekkel való kommunikációért felelős, ezért azt külön mellékfolyamatként kellett futtatnom.

A felhasználói felület alkalmas az autóban történő használatra, kellően könnyen kezelhető és jól átlátható. A megvalósításban a bekapcsolás után a főképernyőre jutunk, ahol láthatjuk az autó valós idejű adatait és a helyzetünket egy térkép segítségével. Innen tudunk továbblépni egy zenelejátszóra és a beállításokba. A beállításokban személyre szabhatjuk az applikációt, különféle színeket választhatunk és fejlesztői módot kapcsolhatunk be. A zenelejátszó pedig egy USB-s adathordozó segítségével képes a zenetartalmak megszólaltatására.

Az utazás végével a rendszer egy összesítő kijelzőre dob minket, amin láthatjuk az átlagos sebességünket, fordulatszámunkat, a megtett utat és a vezetési időt is. Ezen felül látható a megtett út vizuális formában térképen is kirajzolva és arra rányomva az nagyobb alakban is megtekinthető. Itt egy rövid idő elteltével a rendszer leáll, és megvonja az áramellátását a következő indításig.

11.CONTENT SUMMARY

The main goal of this paper was to create a car on-board computer for aftermarket usage. My project started as an OBD black box that can save and monitor the cars real time data, but its functions kept growing and, in the end, it ended up as nearly a different project. So, in my work I request altogether 5 data's ("vss", "rev", "iat", "maf" and "tmp") from the cars computer and pleasingly present it to the user, with the calculated travelled road on a map, and as a number. Furthermore, the program has a built-in music player that can play songs form an insertable medium through an open USB port.

In my implementation I used little existing patterns, and instead I just collected some good ideas. I written the program in Angular based on TypeScript and I run it on electron trough the Armv7l platform. My choice was made because in my opinion that was the language, I was most familiar with, which can be written to the platform and the one that is looks the best that I can create. I used the Raspberry Pi 4B microcontroller to house my program, and it was thanks to the Pi that I could create it with such a nice GUI.

Sadly, not much people program in Angular or in Electron for the Raspberry Pi, that is completely understandable due to the usage of a lot of space, memory, and processing power. The best part is that the there are some components that not even the environment fully supports. It could be my loss of knowledge on the depths of the NodeJS runtime environment, but I found it quicker to implement some parts externally in child processes, than to rebuild to a yet unsupported platform to build packages from the Electron environment.

The user interface is usable in the car and can be appropriately easy to use and easy to look at. In the execution after powering up, it takes the user to the main screen, where it shows the current real time speed, rev, gear, travelled distance and travelled time. On the other side it shows a map with the current location on it. From this screen you can navigate to the music player and the settings. On the settings you can customize the color scheme, measurement units, and even switch do a developer mode. On the music player you can play songs from a USB insertable medium.

On the end of the trip the system shows you an overview page of your average speed, rev, gear, total fuel consumption, total travelled distance, and travelled time. On the right side the program shows you a map with the route you took and on pushing it, it will show it on a bigger map screen. After waiting a short time, the system will shut off and will deprive its power until the next ignition.

12.SZÓJEGYZÉK

OBD	On Board Diagnostics: Magyarul Fedélzeti Diagnosztika . Használata mostanra az egész világon kötelező. Szerepe az autók állapotának számítógéppel támogatott nyomonkövetése, a hibák kijelzése, és az autóval kapcsolatos adatok nyújtása. Aktuálisan az OBD-II-es szabványt kell használni.
PID	Parameter ID: Magyarul Paraméter Azonosító . Az OBD, illetve OBD-II-es szabványokban az adatok lekérdezésére használjuk. Az egyes parancsoknak (paramétereknek) a számszerűsített értéke.
DIN	Deutsches Institut für Normung: Német méretezési szabvány. A dolgozatban az autórádió méretére referálok vele. Egy átlagos kis autórádióra szimpla DIN-es ként, a nagyobb, vagy 2x akkora társaira 2/két/dupla DIN-es ként tekinthetünk. [17]
ACC	Accessory: Autóinkban a kulcs elfordítása utáni első lehetőség. Ilyenkor kapnak áramot az elektromos kiegészítők, mint például az elektromos ablak és a klíma.
Pi–RPI	Raspberry Pi: A Raspberry Pi egy kisméretű számítógép, amelyet előszeretettel használnak képfeldolgozásra és több számítást igénylő IoT projektekhez. Esetemben ez legtöbb esetben a Raspberry Pi 4B 2GB modult jelenti.
GPIO	General Purpose Input/Output: Magyarul Általános célú bemenet/kimenet .
GUI	Graphical User Interface: Magyarul Grafikus felhasználói felület . Az a felület, amit a felhasználó lát és amivel kapcsolatba kerül. [18]
SSH	Secure Shell: Magyarul Biztonságos Csatorna . Linux és egyéb karakteres interfészek interneten keresztüli biztonságos elérése.
GV	Gear Value: Magyarul sebesség érték . A dolgozatban a sebességváltó állásának kiszámítása érdekében számított értékének, amelyből kiválasztom a megfelelő sebességfokozatot.
DS	Driving Style: Magyarul Vezetési Stílus . A dolgozatban a pillanatnyi vezetési stílus számszerűsített értékének használok.
vss	Sebesség
rev	Fordulatszám
iat	Intake air temperature: Magyarul Beáramló levegő hőmérséklet . Az OBD PID listájában használok, a levegő hőmérsékletének értékét kérem le vele.
temp	Motor hőmérséklete
maf	Air flow rate: A motorba beáramló levegő mennyisége. Értékéből az üzemanyagfogyasztás számolható.

fc **Fuel Consumption:** Magyarul **Üzemanyagfogyasztás.**

log Log fájloknak nevezzük az egy adott program, vagy rendszer által létrehozott automatikus állapotot, és hibákat jellemző fájlokat vagy egyes folyamatosan mentett értékeket.

13. IRODALOMJEGYZÉK

- [1] Tinkernut, „hackster.io,” 2018 júni. 23.. [Online]. Elérhető: <https://www.hackster.io/tinkernut/raspberry-pi-smart-car-8641ca>. [Hozzáférés dátuma: 2021 szeptember 21.].
- [2] „WikipediA,” [Online]. Elérhető: https://en.wikipedia.org/wiki/OBD-II_PIDs. [Hozzáférés dátuma: 2021 december 4.].
- [3] G. Wacker. [Online]. Elérhető: <https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1232&context=cpsp>. [Hozzáférés dátuma: 2021 szeptember 21.].
- [4] philmerrel, „github,” [Online]. Elérhető: <https://gist.github.com/philmerrell/d65655ef73a5b3be863491b19b3902ba>. [Hozzáférés dátuma: 2021 október 11.].
- [5] SURYATEJA, „PROJECT HUB,” [Online]. Elérhető: <https://create.arduino.cc/projecthub/SURYATEJA/black-box-obd-pi-using-raspberry-pi-c363aa>. [Hozzáférés dátuma: 2021 október 18.].
- [6] JoakimSoderberg, „github,” [Online]. Elérhető: <https://github.com/peterh/pyobd>. [Hozzáférés dátuma: 2021 szeptember 16.].
- [7] „TopoGrafix,” [Online]. Elérhető: <https://www.topografix.com/gpx.asp>. [Hozzáférés dátuma: 2021 november 17.].
- [8] W. Doyle, „YouTube,” 2017 jún. 6.. [Online]. Elérhető: <https://www.youtube.com/watch?v=EHdSb279Lzg>. [Hozzáférés dátuma: 2021 szeptember 1.].
- [9] j. wesdoyle, „github,” [Online]. Elérhető: <https://github.com/wesdoyle/ng-run-journal>. [Hozzáférés dátuma: 2021 november 7.].
- [10] tmcw, „npm,” [Online]. Elérhető: <https://www.npmjs.com/package/leaflet-omnivore>. [Hozzáférés dátuma: 2021 november 18.].
- [11] phillmerrel, „github,” [Online]. Elérhető: <https://gist.github.com/philmerrell/d65655ef73a5b3be863491b19b3902ba>. [Hozzáférés dátuma: 2021 október 10.].
- [12] rsameser, „Microsoft,” [Online]. Elérhető: https://docs.microsoft.com/en-us/windows/iot/iot-enterprise/getting_started#documentation-overview. [Hozzáférés dátuma: 2021 december 4.].

- [13] M. Parmar, „Windows Latest,” [Online]. Elérhető:
<https://www.windowslatest.com/2020/02/09/heres-how-windows-10-runs-on-raspberry-pi-4-and-3/>. [Hozzáférés dátuma: 2021 december 4.].
- [14] Fireship, „Build a Desktop App with Electron... But Should You?,” 2020 március 4.. [Online]. Elérhető: <https://www.youtube.com/watch?v=3yqDxhR2XxE>.
[Hozzáférés dátuma: 2021 szeptember 5.].
- [15] E. Smekens, „npmjs, github,” [Online]. Elérhető:
<https://www.npmjs.com/package/obd2-over-serial>,
<https://github.com/tuxbox/node-serial-obd>. [Hozzáférés dátuma: 2021 október 19.].
- [16] „npm,” [Online]. Elérhető: <https://www.npmjs.com/package/gps>. [Hozzáférés dátuma: 2021 december 5.].
- [17] J. Laukkonen, „Lifewire,” [Online]. Elérhető: <https://www.lifewire.com/what-is-a-1-din-car-stereo-534607>. [Hozzáférés dátuma: 2021 december 5.].
- [18] „AbbreviationFinder,” [Online]. Elérhető:
<https://www.abbreviationfinder.org/hu/acronyms/gpio.html>. [Hozzáférés dátuma: 2021 december 5.].

14.ÁBRAJEGYZÉK

2.1 Ábra: Kamera és kijelző	5
2.2 Ábra: Raspberry Pi, kijelző és kamera	6
2.3 Ábra: OBD2 parancsok táblázat [2]	7
2.4 Ábra: Navit GPS navigáció oldala	8
2.5 Ábra: M3 Pi főképernyő	9
2.6 Ábra: Pygame GUI "Proof of Concept"	11
2.7 Ábra: Fő képernyő	12
2.8 Ábra: Beállítások képernyő	12
2.9 Ábra: Áramellátásért felelős áramkör	12
2.10 Ábra: OBD-Pi Állapotjelző kijelző	14
2.11 Ábra: OBD-PI Grafikus ábra	15
2.12 Ábra: GeoJSON minta	16
2.13 Ábra: Kocogási táblázat	17
2.14 Ábra: Térképes útvonal.....	17
4.1 Ábra: Fizikai összeköttetések.....	19
4.2 Ábra: Áramellátó modul 2 lehetséges tervezett megvalósítása	20
4.3 Ábra: Szoftver keret magyarázat	21
4.4 Ábra: Főképernyő ábra	22
4.5 Ábra: Zenelejátszó ábra.....	22
4.6 Ábra: Vezetési összegzés ábra	23
4.7 Ábra: Beállítások ábra.....	23
5.1 Ábra: Szoftver felépítési séma és a rétegek közötti adatvándorlás.....	31
5.2 Ábra: Külső szolgáltatások folyamatára	32
6.1 Ábra: Kezdőképernyő megvalósítása.....	34
6.2 Ábra: Lenyíló menü megvalósítása.....	35
6.3 Ábra: Zenelejátszó megvalósítása	36
6.4 Ábra: Zeneszámok kiválasztása	37
6.5 Ábra: Lejátszási lista	37
6.6 Ábra: Beállítások megvalósítása	38
6.7 Ábra: Összegzés megvalósítása	38
6.8 Ábra: Áramellátás fizikai megvalósítása	39
6.9 Ábra: Kijelző dobozzal.....	40
6.10 Ábra: Csatlakozó kábelek	40
6.11 Ábra: Keret az autóban.....	40
6.12 Ábra: Kijelző rögzítési keret	40
6.13 Ábra: Kijelzővel az autóba beszerelés után	40
7.1 Ábra: GUI tesztelése a fejlesztőgépen	41
7.2 Ábra: "Proof of Concept" a rendszer első autóba kerülése.....	42
7.3 Ábra: A rendszer működésének bemutatása	43
7.4 Ábra: Hibás vezetési idővel ábrázolt, huzamosabb üzemidők kijelzése	44
7.5 Ábra: Az összegzés kijelzésének bemutatása	44
7.6 Ábra: Főképernyő nappali működés közben.....	45
7.7 Ábra: Zenelejátszó nappali működés közben.....	45

15.TÁBLÁZATJEGYZÉK

4.1. Táblázat: <i>BILL OF MATERIAL</i> Táblázat.....	19
4.2. Táblázat: <i>Lehetséges vezetési stílusok</i>	25
4.3. Táblázat: <i>Sebességváltó fokozatszámítás értéktartományai</i>	26