
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ**
**DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA**


# GRAFICKÉ ROZHRANIE PRE TELEMETRIU DÁT "FORMULE STUDENT"

**GRAPHICS INTERFACE FOR DATA TELEMETRY OF "FORMULA STUDENT"**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **JURAJ MARTIČEK**
**AUTHOR**

**VEDOUCÍ PRÁCE** **prof. Dr. Ing. PAVEL ZEMČÍK**
**SUPERVISOR**



**BRNO 2022**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

# Zadání bakalářské práce


Student: **Martiček Juraj**
Program: Informační technologie
Název: **Grafické rozhraní pro telemetrii dat "Formule student"**
**Graphics Interface for Data Telemetry of "Formula Student"**
Kategorie: Vestavěné systémy

Zadání:

1. Prostudujte dostupné podklady ke sběrnici CAN a její aplikaci ve Formuli student. Prostudujte též možnosti ukládání a přenosu uložených dat.
2. Navrhněte způsob ukládání telemetrických dat Formule student a přenosu takových dat do grafického uživatelského rozhraní.
3. Navrhněte způsob implementace grafického rozhraní k telemetrickým datům s využitím technologií WWW.
4. Implementujte sběr telemetrických dat prostřednictvím CAN, jejich ukládání a uživatelské rozhraní.
5. Demonstrujte funkčnost rozhraní na vhodném příkladu přenesených dat a ověřte, že rozhraní má vlastnosti vhodné pro Formuli student.
6. Diskutujte dosažené vlastnosti a možnosti pokračování práce.

Literatura:

- Dle pokynů vedoucího práce

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3 zadání

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

Vedoucí práce: **Zemčík Pavel, prof. Dr. Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 3. listopadu 2021

# Abstrakt

Táto práca sa zaoberá vytvorením kompletného systému telemetrie pre účely zberu dát a ladenia pretekárskeho monopostu súťaže Formula Student. Tento systém pozostáva z používateľského rozhrania, ktoré je bezdrôtovo prepojené so zariadením nainštalovaným na pretekárskom vozidle s možnosťou či už živého prenosu dát, alebo aj zaznamenávania takýchto dát. Grafické rozhranie ponúka nástroje pre analýzu dát za účelom vývoja, testovania, ale aj závodenia formule vytvorenej tímom Formuly študent.



# Abstract

Thesis focuses on creation of complete telemetric system for data collection and tuning of a racing car in Formula Student competition. This system consists of graphical user interface, which is wirelessly connected to device installed on a racing car. Telemetry provides live data view coupled with recording features. Graphical user interface provides tools for data analysis to be used for development, testing, and even racing of a formula made by the Formula Student team.



# Kľúčové slová

analýza dát, telemetria, webové aplikácie, TU Brno Racing, Formula Student, grafické používateľské rozhranie, GUI, analýza zbernice CAN, WebGL grafy

# Keywords

data analysis, telemetry, web applications, TU Brno Racing, Formula Student, graphical user interface, GUI, CAN bus analysis, WebGL graphs

# Citácia

MARTIČEK, Juraj. *Grafické rozhranie pre telemetriu dát "Formule student"*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Dr. Ing. Pavel Zemčík

# Grafické rozhranie pre telemetriu dát "Formule student"

## Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením prof. Dr. Ing. Pavel Zemčík. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal

.....
Juraj Martiček
11. mája 2022

## Poďakovanie

Týmto by som chcel poďakovať vedúcemu práce za odbornú pomoc pri návrhu systému, a taktiež by som rád poďakoval celému tímu TU Brno Racing za spoluprácu pri špecifikovaní požiadaviek na systém.

# Obsah

# Kapitola 1

# Úvod

Grafické rozhranie pre telemetriu dát je dôležitou súčasťou pre vývoj formule v tíme TU Brno Racing, a taktiež nevyhnutnou pomôckou pre prácu s množstvom dát, ktoré sú súčasťou automobilového priemyslu, špeciálne pri ladení pretekárskych vozidiel. Táto práca sa zaoberá vývojom celého systému na zber a analýzu dát pre účely závodenia elektrickej formuly univerzitného tímu Formula Student TU Brno Racing. To ale znamená, že grafické rozhranie je len súčasťou celého systému, a podstatná časť pozostáva aj z implementácie spôsobu prenosu dát od mikrokontrolérov v aute až po koncového užívateľa.

Formula Student je študentská súťaž, kde je účelom postavenie jednomiestneho závodného auta (monopost), ktoré následne závodí s ostatnými tímami z univerzít z celého sveta. Formula SAE vznikla v roku 1981 v USA, a na základe nej v roku 1998 vznikla Formula Student (FS). V tejto súťaži sa nehodnotí len rýchlosť auta, ale aj jeho koncept, ciele, a celé finančné plánovanie. Tým pádom sa dá povedať, že tímy FS fungujú ako organizácia alebo reálny tím, ktorý ma svoj rozpočet, musí vybavovať zmluvy so sponzormi, a riadne zdokumentovať celý výrobný proces, a taktiež dbať na bezpečnosť a kvalitu svojho výrobku. V rámci VUT funguje náš tím TU Brno Racing od roku 2010 a je pod záštitou Fakulty strojného inžinierstva. Každý rok je úlohou tímu postaviť nový monopost. Po celý akademický rok sa pracuje na návrhu a výrobe auta, čo je ukončené pretekmi počas letných prázdnin. Tím patrí medzi svetové špičky, v sezóne 2020/2021 vyvíjal 2 formule, s tým, že jeden monopost bol na spaľovací pohon, a jeden na pohon elektrický. Pri spaľovacom motore sa v svetovom rebríčku umiestnil na 24. mieste a pri elektrike na 36. mieste[1]. Pre sezónu 2021/2022 vyvíja plne elektrický monopost s možnosťou plne autonómnej jazdy.

V následujúcej kapitole si vysvetlíme samotný tím TU Brno Racing, formulu, ktorá je týmto tímom vyvíjaná, a jej dôležité súčasti, ktoré sa budú používať v tejto práci. Následne si popíšeme technológie, ktoré boli použité alebo zohľadnené pri vývoji telemetrie. Kapitola Zhodnotenie súčasného stavu analyzuje aktuálne používané riešenia telemetrických systémov, a v druhej časti kapitoly sú zhrnuté požiadavky na výsledné riešenie spolu s návrhom tohto riešenia, ktorý je v nadchádzajúcej kapitole implementovaný postupne po častiach. Po popise implementácie riešenia a jej vyhodnotení nasleduje Zhrnutie a záver, kde je zhrnutá úspešnosť a použiteľnosť daného riešenia, vyhodnocujú sa požiadavky a splnenie zadania.

[1] https://fs-world.org/E/

# Kapitola 2

# Formula student a jej prvky

Kapitola popisuje fungovanie tímu Formuly student, vysvetľuje jeho fungovanie, a súčasti nezbytné pre účely tejto práce. Tieto súčasti a údaje udávané v nich majú priamu súvislosť s touto prácou, a nemajú plne výkladový charakter z dôvodu obmedzenia rozsahu práce.

## 2.1 Tím formule študent

Tímy zúčastnené na súťažiach formule študent majú za úlohu postaviť jednomiestne závodné auto[10]. To znamená, že každý rok prebieha proces plánovania, vývoja, výroby a testovania, ktoré sú ukončené závodmi počas prázdnin daného školského roka. Závody sú organizované v rôznych krajinách po svete, a sú od seba nezávislé. Výsledky z jednotlivých súťaží sú zrátané do výsledného svetového rebríčka, kde sa minulú sezónu s prvým elektrickým konceptným monopostom eD1 umiestil tím TU Brno Racing na 36. mieste.

Okrem závodenia sú hodnotené aj iné aspekty tímu. Na tím sa prihliada ako na organizáciu, ktorá sa musí držať jej finančného plánu, a celý zdroj príjmov pochádza len od sponzorov, ktorí obratom dostávajú reklamu vo forme polepov na aute, a rôznych výhod, ktoré mu tím vie poskytnúť, či sa jedná o propagáciu, ale aj o účasť na firemných akciách, a predovšetkým slúži ako zdroj kvalitných absolventov s praxou v danom priemysle.


Obr. 2.1: Tím TU Brno Racing na závodoch Formula Student Czech

### Súčasný návrh monopostu Dragon e2

Návrh monopostu je vytváraný elektronicky, kde sa počas celého zimného semestra vytvára model auta v CAD softvéri, so všetkými jeho komponentami vo vnútri, kabelážou, a mechanickými časťami. Tento model reprezentuje finálny dizajn auta, ktorý je po novom (kalendárnom) roku nemenný, a teda sa prechádza do ďalšej fázy - stavanie auta.

Dragon e2 vychádza z koncepcie eD1, pričom ešte počas prvotného plánovania sa rozhodlo, že sa nebude prechádzať na pohon všetkých kolies, ale bude implementovaný „driverless“ systém, ktorým vieme dosiahnuť plne autonómnu jazdu v rámci súťažnej trate, ktorá je vymedzená kužeľmi. Okrem úplne inej konštrukcie vozidla, e2 taktiež disponuje iným systémom batérie s LiPol akumulátormi.

Už po vyvinutí prvého elektrického pretekárskeho auta v tíme TU Brno Racing bolo zrejmé, že architektúra dátového toku v rámci takéhoto monopostu nie je vôbec jednoduchá, a pozostáva z mnoho signálov a informácií, ktoré sa prenášajú medzi riadiacimi jednotkami. Všetky takéto jednotky komunikujú medzi sebou po spoločných zberniciach CAN, a vďaka tomu vieme prevziať kontrolu nad tým, čo všetko sa počas jazdy deje, a poprípade vieme zistiť, aké chyby v sieti nastali. Túto sezónu je vyvíjaná formula s dvojnásobným počtom riadiacich jednotiek, a preto sa riziko vzniku problémov len zvyšuje. Taktiež bez-pilotný systém, ktorý sme sa rozhodli vyvíjať tento a následujúce roky takýto problém vôbec neuľahčuje, a len pridáva zložitosť danej implementácie z dôvodu použitia ROS (Robot Operating System) a CANopen siete. Toto riešenie si ale v práci nebudeme bližšie špecifikovať, keďže práca sa zameriava primárne na klasickú elektrickú formulu s pilotom a jej telemetriu.


Obr. 2.2: Monopost ročníka 2020/2021 s názvom eD1

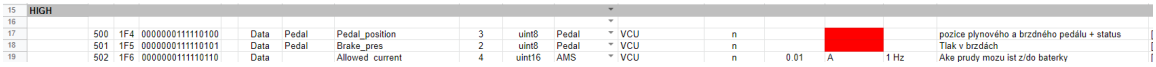
## 2.2 Dôležité systémy a zariadenia

Systémy v monoposte Dragon e2 a tíme TU Brno Racing, s ktorými sa pracuje pri riešení tejto práce, a sú relevantné pre účely tejto práce.

### CAN zbernica v rámci Dragon e2

Z architektonického hľadiska sa používajú 2 zbernice, z čoho je jedna určená na bežnú komunikáciu, nazývaná *GENERIC*, a druhá pre kritické správy nazývaná *CRITICAL*. Potom ešte existuje interná zbernica používaná v rámci akumulátora pre manažment jednotlivých článkov medzi *master* zariadením BMS (battery management system) a jeho *slaves* zariadeniami. Táto zbernica je nazývaná rovnomenne. Mimo účely telemetrie sa však používajú ešte 2 zbernice na ovládanie meniča vysokého napätia, a nabíjanie akumulátora.

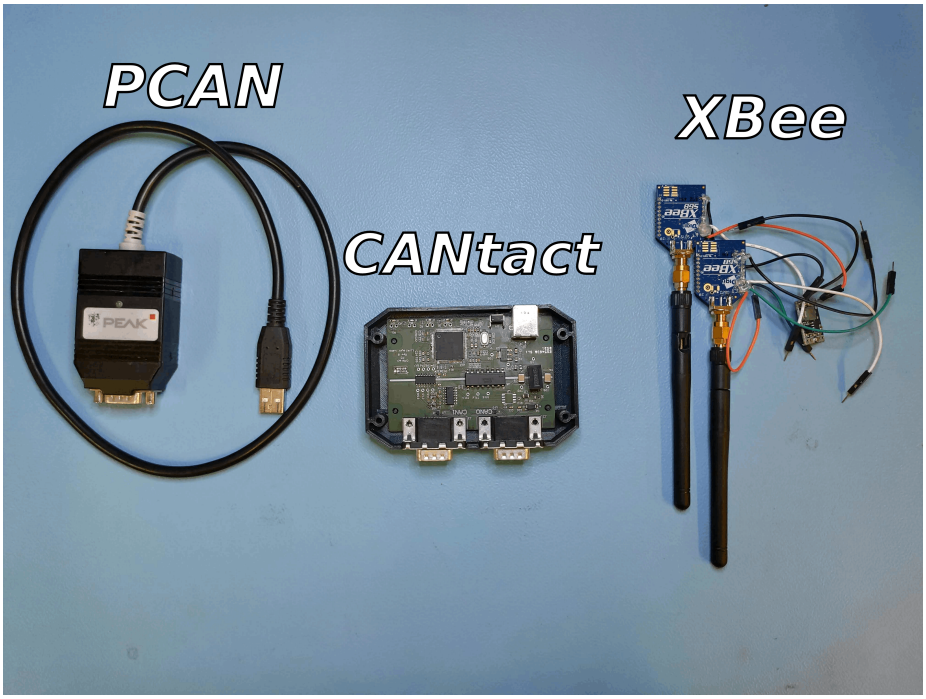
Každá takáto zbernica má zadefinovanú množinu ID správ ktoré sa na nej môžu vyskytnúť, a taktiež pre každú takúto správu je určená jej definícia, ktorá popisuje tvar správy-aký bajt/bajty zodpovedajú akej informácii a v akých jednotkách je daná informácia. Poprípade pri použití bitových polí alebo výčtového typu aj ich korešpondujúce hodnoty ktoré ich popisujú. Doteraz na tento účel bola používaná zdieľaná excelovská tabuľka (Obr. 2.3), kde tieto správy boli popísané, a „pravidelne“ aktualizované človekom ktorý takú správu implementoval, poprípade upravoval.




Obr. 2.3: Aktuálne používaná tabuľka s definíciami správ pre CAN

Zbernica CAN využíva na komunikáciu dva vodiče, pričom konektory a kabeláž si zostavujeme sami, a teda máme pripravený vývod týchto zberníc na rôznych miestach v monoposte, pričom sa vieme pripojiť ku takejto zbernici pomocou 9-pin D-Sub konektoru (podľa odporúčania CiA303-1 resp. CiA102 6.1)[3][4]. Momentálne sa z PC pripájame na zbernicu CAN pomocou:

- Proprietárneho zariadenia PCAN od firmy Peak systems.
- OpenSource riešenia CANtact.
- Vlastné preposielanie CAN správ na rozhranie UART, s ktorým vieme pracovať cez terminál. Napríklad spolu s XBee bezdrôtovými modulmi (vysvetlené v kapitole Technológie).




Obr. 2.4: Spôsoby pripojenia ku CAN zbernici

So zariadeniami PCAN a CANtact je možné komunikovať pomocou knižnice SocketCAN, ktorá je obsiahnutá v hlavičkových súboroch can-utils[1], čo je štandardnou utilitou pre Unix systémy. Posledný spôsob je univerzálne posielanie správ cez UART hardvérový protokol, kde sa posielajú CAN správy v tvare nami definovaného protokolu vyššej vrstvy, s tým že USB prevodníkom dostaneme tieto dáta sériovou linkou. Toto riešenie je najflexibilnejšie, a vieme ho použiť na rôznych zariadeniach - či už na mikrokontroléroch, alebo pri prenose pomocou bezdrôtových modulov.

## 2.3 Aktuálne používaný telemetrický systém

Aktuálny spôsob pre zaznamenávanie telemetrických dát v našom tíme je pomocou zariadenia Omega L2 vyrobené spoločnosťou Cosworth, ktorá sa zamierava výrobou športových automobilov, príslušenstva do F1 a pod[2]. Toto zariadenie je nainštalované fyzicky v monoposte, a pripojené ku zberniciam auta, s tým že zaznamenáva celú komunikáciu na CAN zbernici po konfigurácii v ich softvéri PI Toolset, kde vieme definíciu CAN správ nastaviť ručne, alebo importovať.

Následné nami zvolené správy vieme ukladať buď do internej pamäte, alebo na kompatibilnú telemetriu. Dáta sa ukladajú v proprietárnom binárnom formáte nazývanom dataset file. Tieto dáta vieme stiahnuť zo zariadenia Omega do počítača, kde ich vieme spracovávať v ďalšom softvéri od tejto spoločnosti s názvom PI Toolbox, ktorý nám ponúka jednoduchú prácu s grafmi, a rôzne spracovania dát, no export týchto dát pre ďalšie spracovanie nie je možný.

Fungovanie elektrického systému monopostu eD2 je priblížené v prílohe E.1.

[1] https://github.com/linux-can/can-utils
[2] https://web.archive.org/web/20160401090702/http://www.cosworth.com/default.aspx?id=1089555

# Kapitola 3

# Technológie

Popis technológií použitých a zohľadnených pri riešení práce. V prvej časti sú vysvetlené hardvérové technológie, a postupne sa dostaneme vyššie až ku softvérovým knižniciam v grafickom používateľskom rozhraní. Táto kapitola obsahuje údaje priamo súvisiace s bakalárskou prácou, a nemá encyklopedický charakter kvôli obmedzeniu rozsahu práce.

## 3.1 Zbernica CAN

Controller Area Network - sériová zbernica pôvodne vytvorená pre automobilový priemysel. Používa sa na komunikáciu riadiacich jednotiek a senzorov v rámci auta. Z hľadiska adresovania sa na zbernici CAN využíva spôsob označovania správ pomocou ich ID, čo je 11 bitové číslo na začiatku správy, a teda nám CAN umožňuje rozlišovať až $2^{11} = 2048$ rôznych správ. Táto zbernica pozostáva z 2 vodičov, je vysoko-rýchlostná, a duplexné spojenie je polovičné (half-duplex). A teda naraz môže vysielať na zbernici len 1 zariadenie[8]. Z hardvérového hľadiska sa na kábloch zbernice rozpoznávajú signály log. 0 a log.1 s tým, že vodiče sú diferenčnými pármi (jeden signál je Y a druhý neg. Y). Pri CAN je 0 dominantná, čo znamená, že ak chce uzol zaslať 0 a iný 1, tak výsledok na zbernici bude 0. Zariadenie ktoré vysiela vždy počúva na zbernici počas vysielania, a teda ak pošle log. 1 na zbernicu, ale na zbernici sa objavila log. 0, zariadenie vie, že iné zariadenie s vyššou prioritou posiela na zbernici, a teda prestane vysielať, a počká kým druhý uzol prestane vysielať. Toto nám zaručuje arbitráčné pole, v ktorom sa nachádza ID správy. Vďaka tomuto systému vieme prioritizovať správy podľa ich ID, a teda najprioritnejšia správa má ID 0, a najmenej prioritná má ID 2031. Tento spôsob komunikácie sa nazýva CSMA/CD+AMP (Carrier Sense Multiple Access/Collision Detection with Arbitration on Message Priority)[9]. Vďaka CAN sú zaručené:

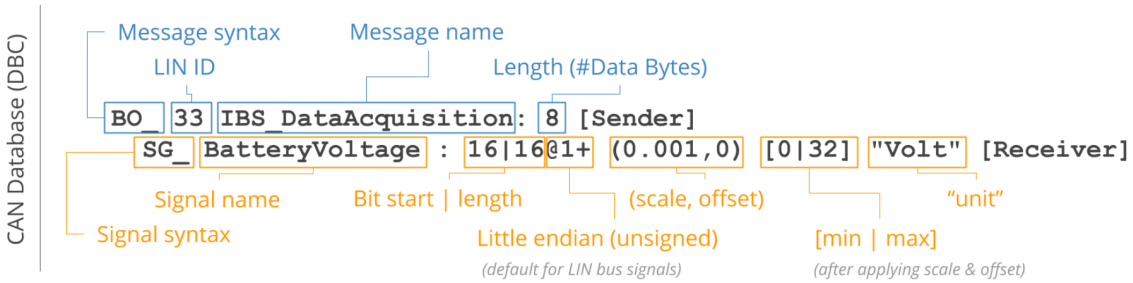
- Viacnásobný prístup
- Detekcia kolízií
- Arbitrácia (rozhodovanie o pridelení zbernice)
- Prioritizácia správ

**Definícia CAN správ** Dáta prenášané po zbernici sa skladajú z 11 bitového ID, a 1-8 bajtov obsahu správy. Aby sme danú informáciu vedeli správne predať prijímateľovi, musíme tieto neznáme dáta dekódovať na základe nejakých pravidiel. Potrebné tvary správ

prenášaných po sieti sme doposiaľ definovali v excelovskej tabuľke, s tým, že najpotrebnejšie informácie pre dekódovanie správy sú: dátový typ správy, ak sa prenáša v rámci 8 bajtov tela správy viacero informácii, tak dátové typy jednotlivých informácii v správe, ich modifier (akým násobkom je potrebné hodnotu prenásobiť), a unit - jednotka, v akej daná správa momentálne je (napr. volty). Momentálne sa používa niekoľko unifikovaných spôsobov na popis takýchto správ, no žiaden nie je štandardom, len niektoré sa používajú na základe konvencií. Aktuálne používaný spôsob v TU Brno Racing sme si popísali vyššie v časti CAN zbernica v rámci Dragon e2. Najčastejšie používaným formátom správ je formát DBC.

DBC formát popisuje CAN správu ako správu s jej ID a dĺžkou dátovej časti (DLC). Taktiež pri správe jej vieme pridať popis, a zariadenie (uzol) od akého sa odosiela (Obr. 3.1). Dôležité je si uvedomiť, že tzv. message v DBC formáte nám nepopisuje to, čo správa obsahuje. Na to nám slúžia signály v rámci danej správy[5]. Signál je definovaný svojou pozíciou v rámci dátovej časti správy - tzn. bitová pozícia začiatku, a bitová dĺžka signálu. Pomocou tohto signálu vieme z dát dostať informáciu, a to tak, že danú hodnotu vymedzenú jej pozíciou v rámci správy, prevedieme na dátový typ podľa daného signálu, a následne z definície signálu vieme získať *factor* a *offset*, kde sa výsledná hodnota vypočíta ako $value = (raw\_value * factor) + offset$. Okrem spomínaných parametrov vieme v definícii signálu nájsť aj jeho minimálnu a maximálnu hodnotu, jednotku, v akej je informácia, a zoznam prijímateľov signálu. Práve vďaka prijímateľom signálu, a odosielateľom správy vieme zostaviť jednoduchú mapu pohybu dát medzi jednotlivými zariadeniami v rámci systému.

Formát DBC bol prvý krát použitý spoločnosťou Vector Informatik GmbH[1]. Tento formát je textový, a neexistuje pre neho špecifikácia. Výrobcovia programov pracujúcich s takýmto formátom využívajú práve voľne dostupné špecifikácie takýchto súborov, ktoré si často sami prispôsobujú/rozširujú podľa potreby a účelu programu. My budeme používať obecnú špecifikáciu, ktorá vychádza z špecifikácie Vector používanú aj knižnicou CANpy[2].

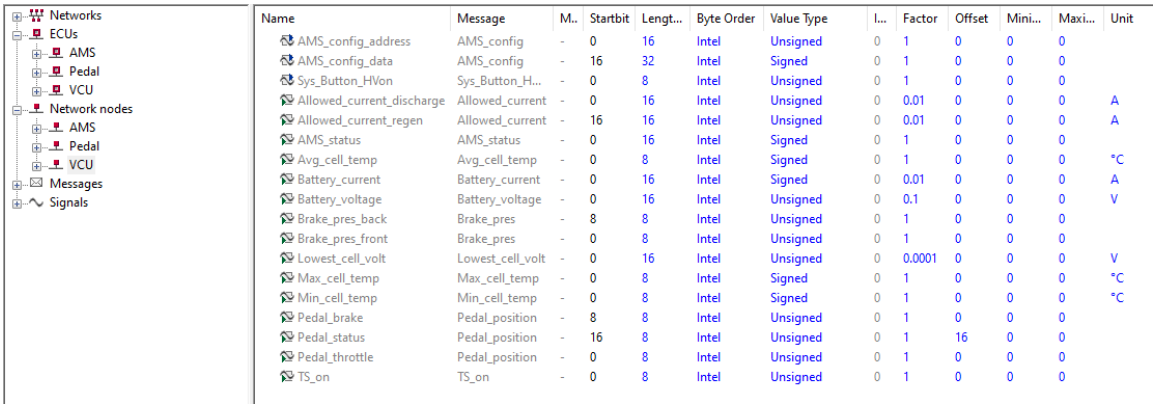



Obr. 3.1: Správa vo formáte DBC (prebraté z[5] viz. príloha B)

Jednou z možností pre vytváranie DBC definícií je použitie aplikácie od firmy Vector s názvom CANdb++ (Obr. 3.2), kde si vieme jednoducho nakonfigurovať prepojenie a správy v rámci jednej siete. Táto aplikácia podporuje aj iné implementácie CAN zbernice na vyššej vrstve. To znamená, že napríklad vieme použiť sieť J1939, CANfd, Ethernet, a iné. Pre naše účely je použitý jednoduchý CAN len na fyzickej a linkovej vrstve, a to znamená, že v rámci DBC vieme mať len jednu sieť, v nej riadiace jednotky, a v rámci nich správy a v správach signály. Okrem toho je často používaná aplikácia od firmy Kvaser s názvom

[1] https://www.vector.com/
[2] https://github.com/stefanhoelzl/CANpy/blob/master/docs/DBC_Specification.md

*Database Editor*. Na zvolenej aplikácii nezáleží, keďže pracujú na rovnakom princípe, a používateľské rozhranie sa líši minimálne.




| Name | Message | M.. | Startbit | Lengt... | Byte Order | Value Type | I... | Factor | Offset | Mini... | Maxi... | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AMS_config_address | AMS_config | - | 0 | 16 | Intel | Unsigned | 0 | 1 | 0 | 0 | 0 | |
| AMS_config_data | AMS_config | - | 16 | 32 | Intel | Signed | 0 | 1 | 0 | 0 | 0 | |
| Sys_Button_HVon | Sys_Button_H... | - | 0 | 8 | Intel | Unsigned | 0 | 1 | 0 | 0 | 0 | |
| Allowed_current_discharge | Allowed_current | - | 0 | 16 | Intel | Unsigned | 0 | 0.01 | 0 | 0 | 0 | A |
| Allowed_current_regen | Allowed_current | - | 16 | 16 | Intel | Unsigned | 0 | 0.01 | 0 | 0 | 0 | A |
| AMS_status | AMS_status | - | 0 | 16 | Intel | Signed | 0 | 1 | 0 | 0 | 0 | |
| Avg_cell_temp | Avg_cell_temp | - | 0 | 8 | Intel | Signed | 0 | 1 | 0 | 0 | 0 | °C |
| Battery_current | Battery_current | - | 0 | 16 | Intel | Signed | 0 | 0.01 | 0 | 0 | 0 | A |
| Battery_voltage | Battery_voltage | - | 0 | 16 | Intel | Unsigned | 0 | 0.1 | 0 | 0 | 0 | V |
| Brake_pres_back | Brake_pres | - | 8 | 8 | Intel | Unsigned | 0 | 1 | 0 | 0 | 0 | |
| Brake_pres_front | Brake_pres | - | 0 | 8 | Intel | Unsigned | 0 | 1 | 0 | 0 | 0 | |
| Lowest_cell_volt | Lowest_cell_volt | - | 0 | 16 | Intel | Unsigned | 0 | 0.0001 | 0 | 0 | 0 | V |
| Max_cell_temp | Max_cell_temp | - | 0 | 8 | Intel | Signed | 0 | 1 | 0 | 0 | 0 | °C |
| Min_cell_temp | Min_cell_temp | - | 0 | 8 | Intel | Signed | 0 | 1 | 0 | 0 | 0 | °C |
| Pedal_brake | Pedal_position | - | 8 | 8 | Intel | Unsigned | 0 | 1 | 0 | 0 | 0 | |
| Pedal_status | Pedal_position | - | 16 | 8 | Intel | Unsigned | 0 | 1 | 16 | 0 | 0 | |
| Pedal_throttle | Pedal_position | - | 0 | 8 | Intel | Unsigned | 0 | 1 | 0 | 0 | 0 | |
| TS_on | TS_on | - | 0 | 8 | Intel | Unsigned | 0 | 1 | 0 | 0 | 0 | |

Obr. 3.2: CANdb++ signály definované v rámci jednotky VCU

## 3.2 Mikrokontroléry v automobilovom priemysle

Vstavané (anglicky embedded) zariadenia sú základným stavebným prvkom predovšetkým elektrických automobilov. Problém s takýmito zariadeniami je to, že nám neposkytujú, resp. je nevyhnutné, aby fungovali v reálnom čase bez spomalenia operačných systémov, a preto nám neposkytujú jednoduché I/O rozhranie s pamäťovými úložiskami. Vo všeobecnosti platí, že jednoduchosť implementácie pri takýchto zariadeniach je veľmi dôležitá z dôvodu vysokej zložitosti samotného kódu. Ako už vieme, poznáme mnoho možností perzistentného úložiska, no treba uvažovať vonkajšie vplyvy ktoré pôsobia na mechanickú stránku daného pamäťového úložiska. To znamená že sily a vibrácie pôsobiace na pamäť sú pomerne vysoké, a preto prichádza do úvahy len pamäť, ktorá nie je založená na mechanickom princípe.

V automobilovom priemysle sa používajú rôzne mikrokontroléry na báze ARM mikroprocesorov, najznámejšie z nich sú od firiem STMicroelectronics a NXP s inštrukčnou sadou Cortex-M, alebo procesory na báze AVR inštrukcií (podobné Arduino). Dôležité ale je zohľadniť použitie takýchto procesorov práve na pretekárskom vozidle, kde kvôli vysokým vibráciám je veľmi zložité vyvinúť dosku, ktorá takúto mechanickú námahu zvládne. Klasické procesory, alebo zariadenia bežne používané (napr. RaspberryPI) nie sú stavané na takúto námahu a je potrebné si navrhovať vlastné riešenia založené na vyššie spomínaných procesoroch a vlastných doskách plošných spojov.

Pre bezdrôtový prenos sa aktuálne v automobilovom priemysle používajú IOT riešenia (v2v komunikácia), alebo taktiež sa využívajú iné point-to-point riešenia, poprípade jednoduché pripojenie na existujúcu mobilnú sieť[1]. Jednou z možností pre bezdrôtový prenos dát je použiť pár XBee modulov, ktoré nám zaručujú možnosť komunikácie rôznymi protokolmi, ako napríklad ZigBee, ale podporuje aj WiFi, a iné možnosti protokolov najnižšej fyzickej vrstvy. Najväčšou výhodou týchto modulov, je to, že vedia pracovať v bez-konfiguračnom móde, kde len zrkadlia všetky dáta medzi sebou, a tým mnohonásobne zjednodušujú prácu s nimi práve ak vyžadujeme len jednoduchú point-to-point komunikáciu.

## 3.3 Webové technológie a komunikácia s ich využitím

V tejto časti je vysvetlené použitie webových technológií pri tvorbe aplikácií, sú znázornené knižnice, ktoré sa považujú za trendy v oblasti súčasného vývoja webových aplikácií, a taktiež je popísané využitie takýchto technológií pri komunikácií po sieti v reálnom čase.

### Node.js

Technológia Node.js[3] nám ponúka behové prostredie webového prehliadača použiť aj priamo na serveri. Node.js je asynchrónne, udalosťami poháňané prostredie pre beh JavaScriptu, navrhnuté pre stavbu škálovateľných sieťových aplikácií. Node.js vzniklo úpravou open-source prostredia pre vykonávanie JavaScriptu používaného v prehliadačoch Google Chrome, V8, pre beh na server s obohatením o štandardnú knižnicu umožňujúcu efektívnu tvorbu výkonných sieťových aplikácií. O túto funkcionalitu sa budeme opierať v tejto práci.

Ako bolo spomenuté vyššie, v jazyku JavaScript sa používa udalosťami poháňané prostredie (v anglickej terminológii *events*). Udalosti (napríklad klikanie myšou, stláčanie kláves, a.i.) vyvolané asynchrónne sú postupné obsluhované jadrom behového interpretu, a to tak, že sa zavolajú všetky obslužné funkcie (callbacky), ktoré sú naviazané na daný typ eventu. Tieto callback funkcie si vieme vytvárať a mazať sami. Node.js nám namiesto obsluhy používateľských vstupov ponúka rôznu komunikáciu so súčasťami operačného systému práve cez rozhranie týchto eventov. Táto funkcia je interne implementovaná pomocou slučky, pri ktorej jadro behového programu periodicky a postupne obsluhuje kód, užívateľské vstupy a eventy[4].

Tradičné multi-page aplikácie fungujú na princípe požiadavkov od webového prehliadača, ktoré sú odosielané webovému serveru, ktorý na nich odpovie HTML obsahom. Na základe odpovede prehliadač vykreslí webovú stránku. Preklikmi na hypertextové odkazy sa používateľ vie navigovať medzi stránkami. Pri každej navigačnej akcii prebehne plné načítavanie, ktoré trvá aj niekoľko sekúnd. Toto riešenie sa však nevyrovnáva natívnym aplikáciam. Alternatívu ponúkal vznik nových technológií známy ako Web 2.0, a spolu s ním aj vznik tzv. **single-page aplikácií (SPA)**. V tomto prípade prehliadač urobí požiadavku len pri prvom načítaní, a následne aplikácia komunikuje so serverom prostredníctvom technológie AJAX[5], kde sa na pozadí odosielajú len dáta[2].

**REST API** je v súčasnosti najviac používanou architektúrou pre komunikáciu medzi klientom a serverom pri webových aplikáciach, ale je možné ho použiť aj pri aplikáciach natívnych. RESTful architektúra a API navrhnuté podľa jej odporúčaní voláme REST API[2]. REST je skratka pre REpresentational State Transfer, a táto architektúra pre svoje fungovanie ťaží z dostupných možností HTTP protokolu. Nad jednotlivými metódami HTTP umožňuje REST vykonávať rôzne operácie.

### React

React[6] je knižnica vyvinutá spoločnosťou Meta (skôr Facebook) s určením na tvorbu používateľských rozhraní s širokou škálou možností pre asynchrónne správanie, a pre prácu so stavmi, ktoré nám umožňujú tieto stavy meniť, poprípade ukladať v jednoduchom formáte.

[3] www.nodejs.org
[4] https://nodejs.dev/learn/the-nodejs-event-emitter
[5] https://developer.mozilla.org/docs/Web/Guide/AJAX
[6] www.reactjs.org

Webový prehliadač vnútorne reprezentuje štruktúru stránky pomocou objektového modelu dokumentu (DOM). Manipuláciu s týmto modelom nám ponúka webové API prístupné v JavaScripte. Bohužiaľ práca s týmto modelom je výpočetne náročná, a preto React.js tento problém rieši zavedením technológie Virtual DOM, ktorá nám bude nápomocná pri našej aplikácii[2]. Taktiež si React interne udržiava stavu aplikácie (stav elementu DOM, a jeho atribútov[7]). Tento stav aplikácie má svoj životný cyklus, a celá aktualizácia elementov je zostavená na základe stromu aktualizácií, kde sa vždy v rámci stránky obnovia len tie komponenty, ktoré sa zmenili, a to čo sa nezmenilo sa neaktualizuje. Toto chovanie sa opiera o hierarchiu elementov, a ich props, ktoré ak sa zmenia  zmení sa aj korešpondujúci element. Takýto stav  tzv. state sa dá použiť nielen na atribúty DOM prvkov, ale aj na ukladanie a prácu s dátami.

React hooky nám poskytujú prístup ku takémuto stavu aplikácie, čo je základnou štruktúrou pre prácu dát, a taktiež vďaka nim môžeme pristupovať ku jeho iným funkciám bez nutnosti použitia triedy. Pred uvedeným tejto funkcionality bol pri funkcionálnych komponentoch[8] len jediný spôsob. Rodičovský komponent vždy musel predať dáta ďalej svojmu synovskému pomocou props, čo je implementované predávaním cez parametre funkcie. Hooky nám ale poskytujú omnoho väčšiu flexibilitu pri práci s dátami. Nielen nám poskytujú prístup k React state, ale môžeme s nimi využívať napríklad React context, vďaka ktorému predávanie dát nemusí byť len hierarchické, ale vieme dáta získať aj z iných miest ako od rodiča.

**WebGL**[9] je medziplatformový otvorený webový štandard pre grafické API ku nízkolevelovým rozhraniam založeným na OpenGL ES. Je to spôsob akým mať OpenGL aplikácie v prehliadači. Poskytuje nám aplikačné rozhranie pre 3D grafiku, programuje sa v prehliadači a je prepojené s OpenGL ES[12]. Toto prepojenie je vystavené jazyku JavaScript cez jeho canvas okno. Vďaka dostupnej akcelerácii na grafickej karte vieme v pomalom DOM, poprípade *virtual DOM* vykresľovať vysoké toky dát bez dopadu na výkonnosť aplikácie. .

**Electron**[10] je knižnica pre vytváranie desktopových a prenositeľných aplikácii napísaných v jazyku JavaScript. Vďaka tomuto vieme zaručiť jednoduchú prenositeľnosť medzi zariadeniami, a taktiež vieme vytvoriť GUI, ktoré bude jednoducho fungovať ako aj vo webovom prehliadači, ale aj na akomkoľvek zariadení. Funguje len ako „obal“ okolo webovej aplikácie.

### SerialPort

Keďže dáta nám môžu prichádzať sériovo (protokol UART), tak potrebujeme mať spôsob, akým tieto dáta budeme spracovávať. Na tento účel sme použili knižnicu SerialPort[11], ktorá ako natívny modul ponúka možnosť komunikácie cez sériové rozhrania počítača. Architektúra knižnice serialport spočíva z tzv. bindings, interface a parserov.

Keďže Node.js nevie pracovať priamo s periférіami počítača, je potrebné napojiť knižnicu pomocou "bindings"na binárne rozhranie, ktoré už má priamo prístup k hardvéru. Serialport využíva cpp-bindings ktoré poskytujú napojenie ku C++ rozhraniu, ktoré s týmito hardvérovými periférіami pracovať vie. Tento modul nám poskytuje prepojenia pre Linux, Mac a Windows operačné systémy.

[7] atribúty pre komponenty = react props
[8] https://djoech.medium.com/functional-vs-class-components-in-react-231e3fbd7108
[9] https://www.khronos.org/webgl/
[10] www.electronjs.org
[11] www.npmjs.com/package/serialport

Interface rozhranie poskytuje API postavené nad bindings, ktoré si vieme naprogramovať sami, alebo poprípade použiť stream interface, ktorý nám poskytuje knižnica. Toto rozhranie využíva štandardné implementáciu abstraktného rozhrania stream z Node.js, a teda práca s ním je pomerne jednoduchá, a hlavne natívne funguje so spomínaným Node.js.

Parsery nám poskytujú samotné spracovanie prichádzajúcich dát zo sériovej linky. V podstate implementujú jednoduchý protokol na zasielanie správ. Napríklad ak máme 13 bajtovú CAN správu, tak vieme použiť "parser-byte-length"ktorý nám vždy po 13-tich bajtoch správu predá našej callback funkcii. Ale taktiež vieme použiť aj iné spôsoby parsovania správy, poprípade si navrhnúť vlastný protokol, ktorým budeme spracovávať dáta prichádzajúce z protokolu UART[11].

**SocketIO**[12] je knižnica, ktorá nám zaručuje to, že dáta vieme rozosielať všetkým klientom naraz, a taktiež vieme klientovi pri novom pripojení zaslať všetky dáta, ktoré boli doposiaľ zaznamenané, a tým žiaden klient nepríde o dôležité dáta. Táto knižnica pracuje na princípe WebSocketov, kde sa udržiava tzv. „keep-alive“ spojenie a aj pri vypadnutí komunikácie je kladená snaha o znovupripojenie

**Knižnica SocketCAN**[13] je dostupná po nainštalovaní can-utils unixových hlavičkových súborov. Vieme ju použiť pre prijímanie dát z PCan, CanTact a iných zdrojov. Bohužiaľ jednoduchá alternatíva tejto knižnice pre Windows neexistuje.

**Návrhový vzor „adaptér“** slúži na adaptovanie rôznych zdrojov dát bez nutnosti úpravy programu, ktoré tieto dáta prijíma. Ako sme si popísali vyššie, dáta z CAN zbernice vieme získať až troma (možno aj viac) spôsobmi. Práve takýto problém rieši návrhový vzor adaptér vďaka ktorému je jednoduché napojiť rôzne knižnice, a spôsoby získavania tých istých dát s tým, že aplikácia stále pracuje s jedným rozhraním.

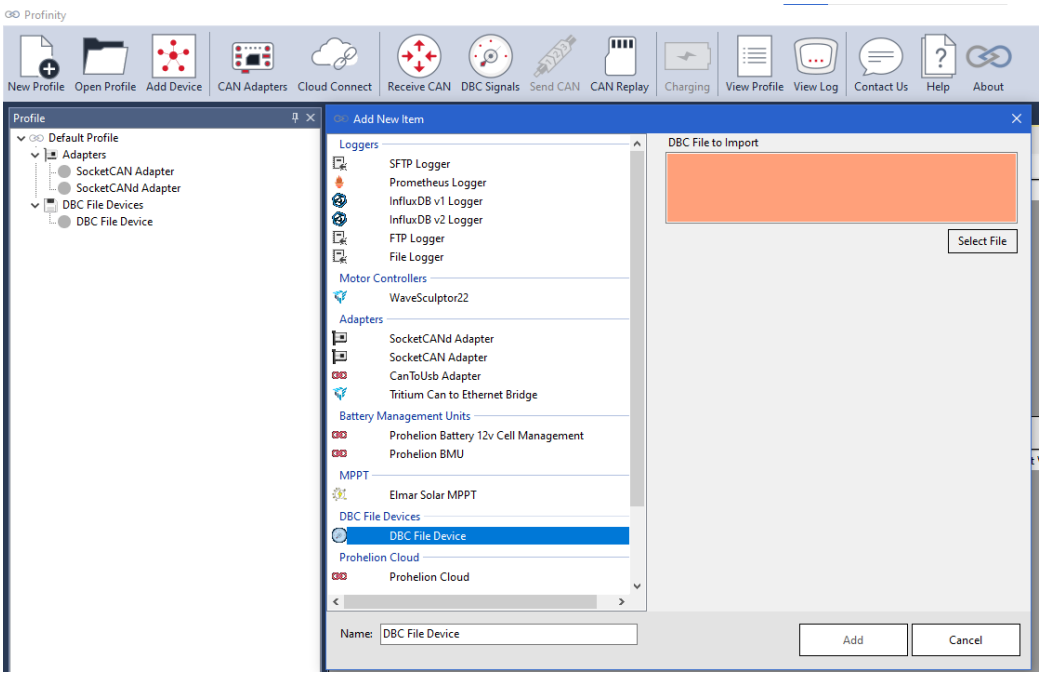
## 3.4 Telemetrické systémy

V tejto časti si popíšeme rôzne existujúce riešenia telemetrie, ktoré sa v praxi používajú. Ide hlavne o softvér, a z časti aj hardvér, ktorý je používaný nielen tímami Formula student, ale aj v oblasti pretekárskych vozidiel vo svete.

### Prohelion Profinity

Softvér Profinity od firmy Prohelion (Obr. 3.3) je voľne stiahnuteľný a veľmi jednoduchý na používanie. Základný stavebný prvok aplikácie spočíva v pridávaní tzv. devices, ktoré môžu mať rôznu funkcionalitu. Pre zaznamenávanie dát ponúka tzv. SocketCANd Adapter, čo znamená linuxový "server", na ktorom beží utilita socketcand (súčasťou can-utils). Na toto zariadenie sa následne pripájame s tým, že zariadenie je fyzicky pripojené ku zbernici CAN. Po pripojení vidíme v spodnom stavovom riadku (aplikačný log) stav pripojenia. Momentálne vidíme každú správu, ktorá príde na zbernicu CAN. Ďalší „device“, ktorý vieme pridať je DBC File Device, ktorý po zadaní DBC súboru (ako môžete vidieť na obr. 4.1) pripraví možnosť dekódovania správ pomocou tejto databázy. Po otvorení záložky *DBC Signals* môžeme tieto správy vidieť v štandardnom zobrazení, a po rozkliku aj ich signály.

[12] https://socket.io
[13] viz. Príloha B (SocketCAN - The official CAN API of the Linux kernel)

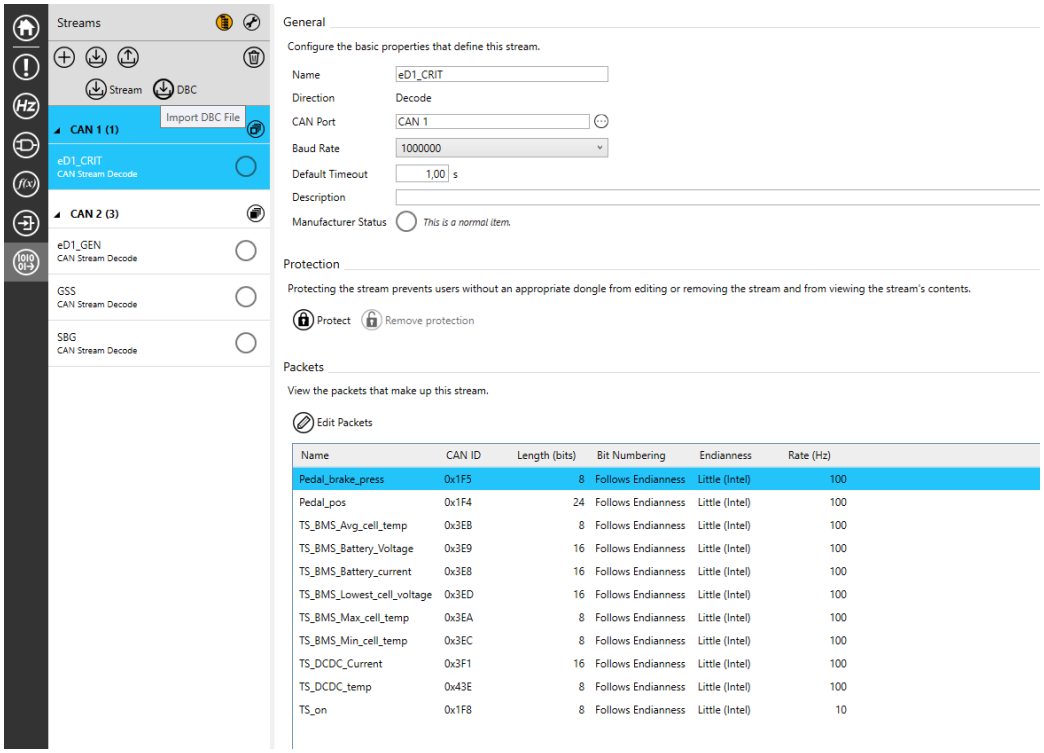



Obr. 3.3: Prohelion používateľské rozhranie

## Cosworth

Ako sme si popisovali vyššie, spoločnosť Cosworth nám ponúka celý systém telemetrie, a to znamená, že z krabičky Omega L2, kde sú zaznamenávané dáta si vieme tieto dáta zobraziť v softvéri PI Toolbox.

Omega L2 sa konfiguruje cez softvér PI Toolset, kde si vieme nastaviť ktoré správy (v ich slovníku kanály) vieme zaznamenávať do pamäte telemetrie. Dekódovanie (prijímanie a spracovávanie) CAN správ prebieha ešte v jednotke, ktoré sa konfiguruje v ich rozhraní (Obr. 3.4), s tým, že rozhranie ponúka jednoduchý prevod súboru DBC do ich formátu. Správy treba ešte ale upraviť, a používať v ich formáte Toolset Library File. PI Toolset v kombinácii s Omega L2 zariadením prináša ale aj mnoho užitočných funkcií, z ktorých je napríklad veľmi užitočná možnosť vytvárania tzv. matematických kanálov, ktoré nám vedia spracovávať už nakonfigurované kanály v aute, a vytvárať nové pomocou jednoduchej syntaxe v jazyku PI Math. Ďalšou z nami využívaných funkcionalít je možnosť pripojenia senzorov rýchlosti kolies, kde jednotka spracováva jednotlivé senzory otáčok, a vytvára z nich presnú rýchlosť auta, ktorú vieme použiť na ďalšie funkcionality vďaka možnosti dekódovania takýchto kanálov späť na zbernicu CAN.




Obr. 3.4: Konfigurácia dekódovania CAN správ v PI Toolset s možnosťou DBC importu

# Kapitola 4

# Zhodnotenie súčasného stavu a návrh riešenia

Prvý krok pri vývoji novej aplikácie je ujasnenie požiadavkov, aké by malo spĺňať dané riešenie. Ak sú požiadavky jasne definované, prechádza sa na návrh riešenia, na základe ktorého sa riešenie aplikuje. My sme tento postup zvolili tiež. Najskôr je zhodnotený aktuálny stav telemetrických systémov, a v ďalších sekciách sa nachádza podrobný opis takéhoto návrhu riešenia.

## 4.1 Zhodnotenie stavu

Pri použití softvéru Prohelion Profinity sa prijímané správy zobrazujú, ale neagregujú podľa ID, a len vypisujú pod seba, čo pri väčšom zaťažení zbernice je veľmi neprehľadné. Pri zobrazení DBC dekódovania nevidíme kedy správa prichádza, a či sa jej hodnota vôbec aktualizuje. Jedným s veľkých plusov je možnosť nahrávania týchto správ, a následnej možnosti „CAN Replay“, za pomoci ktorej si vieme prehrať priebeh správ počas merania alebo jazdenia, a na základe toho ladiť potrebné parametre vozidla. Tento softvér nepodporuje žiadnu ďalšiu analýzu dát ani posielanie na zbernicu CAN, a jedinou z výhod je možnosť pretáčania príchodzej komunikácie v čase zo zaznamenaných dát.

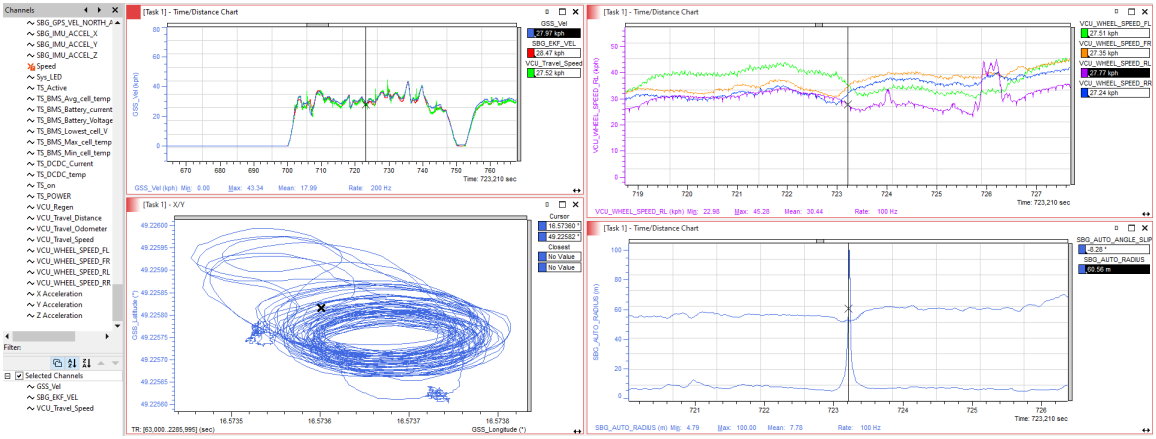
Pre zmenu telemetrické riešenie od firmy Cosworth je omnoho viac prepracované, no softvér nie je dokumentovaný, a jediný spôsob ako zistiť jeho funkcionalitu a správne fungovanie bolo metódou pokus-omyl. Namerané dáta, ktoré sú uložené na zariadení Omega L2 v proprietárnom formáte PI Dataset si vieme stiahnuť do počítača po sieti Ethernet. Na analýzu týchto dát je potrebné použiť pomerne neintuitívne rozhranie PI Toolbox (Obr. 4.1), kde na prácu s ním bolo potrebné zaškolenie od starších kolegov z tímu. Po zoznámení sa z rozhraním je práca s touto aplikáciou pomerne jednoduchá, a má veľmi rozsiahle možnosti analýzy dát merania. Problém nastáva pri potrebe tieto dáta dostať do iných aplikácií ako tabuľkový editor alebo MatLab. A teda týmto je rozsiahlejšia práca s dátami obmedzená. Na druhú stranu ponúka táto utilita množstvo nástrojov, z ktorých je mojim najobľúbenejším možnosť aplikovať na signál matematické funkcie, poprípade tieto signály spájať, a vytvárať z toho nové signály. Medzi hlavné výhody patrí:

- Možnosť vytvárania matematických kanálov či už na zariadení počas jazdy auta, alebo pri analýze dát

- Funkcionalita kanálov, vďaka čomu vieme jednotku na aute používať ako hlavnú riadiacu jednotku, ktorá má aj logiku, a nie len naslúcha dátam
- Podpora GPS s trasovaním trate, ktorú si vieme nahodiť pred jazdou s tým, že nám meria časy kôl, a následne tieto kolá vieme cez seba analyzovať v Toolboxe.
- Natívna podpora enkóderov rýchlosti kolies z ktorých automaticky vypočítava rýchlosť auta, a podpora odometra[1]

Nevýhody:

- Neintuitívne rozhranie, hlavne to pre analýzu dát
- Ekosystém nekompatibilný s iným softvérom, a proprietárne riešenia
- Cena
- Vysoká hmotnosť na vozidle
- Bez akejkoľvek dokumentácie, alebo podpory na internete



Obr. 4.1: PI Toolbox analýza nameraných dát

Takéto riešenia sú zdĺhavé na konfiguráciu. Omega jednotka taktiež zaberá drahocennú hmotnosť v rámci formule, ktorú sa snažíme minimalizovať do posledného gramu, keďže to je jedna z vecí, ktorá nás vie deliť od pódiového miesta na závodoch. Najväčší problém pri používaní takýchto riešení je jeho finančná náročnosť. Keďže tímy vychádzajú len z nízkeho rozpočtu, ktorý je tvorený sponzormi, ktorí vo svojej podstate podporujú len študentov, a vyvíjané autá sú drahé na materiály, tak sa snažíme minimalizovať náklady na veci, ktoré nie sú vitálne pre chod formule.

## 4.2 Požiadavky na funkčnosť aplikácie

Pred vytvorením návrhu aplikácie bolo potrebné si ujasniť požiadavky na funkcionalitu a parametre aplikácie. Keďže táto práca ma za úlohu zjednodušiť mnohé aspekty sezóny tímu TU Brno Racing, požiadavky na aplikáciu vychádzajú od žiadostí jeho členov. Preto

[1] palubný počítač - meranie prejdenej vzdialenosti auta

| Dotazovaná funkcionalita | Miera žiadanosti |
|---|---|
| Grafy | 95,38% |
| Export a import merania | 92,31% |
| Enum (stavy jednotiek) | 92,31% |
| Grafy s viacej čiarami (viac správ naraz) | 86,15% |
| Odosielanie dát na CAN | 83,08% |
| Offline logovanie na aute (bez pripojenia PC) | 81,54% |
| Tabuľka všetkých prijatých správ | 80,00% |
| Export do CSV (matlab, excel, ...) | 78,46% |
| Progress bar | 78,46% |
| Bezdrôtová telemetria | 76,92% |
| Dark mode | 75,38% |
| Live video z formule a intercom | 72,31% |
| Vytváranie vlastných ukazateľov/komponentov (grafy, progress bary, ...) | 72,31% |
| Vlastné usporiadanie rozloženia okien (presúvanie, zmena veľkosti) | 70,77% |
| Live video z formule a intercom | 63,08% |
| Možnosť aplikácie bežať vo webovom prehliadači | 58,46% |
| Math channels | 58,46% |
| Možnosť viacero používateľov | 56,92% |

Obr. 4.2: Dotazník na požadovanú funkcionalitu telemetrie

sme vytvorili dotazník (Obr. 4.2), ktorého účelom bolo spresniť žiadanosť jednotlivých parametrov aplikácie. Dotazník bol koncipovaný formou otázky, ktorá spočívala v nejakej funkcionalite, a výberu od 1 po 5, ktorý značil potrebu danej funkcionality (1 = Musím to mať, 5 = Nebudem to používať). Kompletné výsledky si môžete prezrieť v Prílohe C.

Z požiadavkov „zákazníkov“ si vieme zhodnotiť, ktoré funkcie by mala naša aplikácia spĺňovať. No samozrejme sa nevyhneme niektorým implementačným detailom, bez ktorých by bola aplikácia nedostačujúca pre každodennú prácu. Jednou z takých vecí je Definícia CAN správ. Mnohé automobilky, a zariadenia využívajúce CAN zbernicu pracujú s dátovým formátom DBC. My samozrejme chceme držať kompatibilitu s ostatnými zariadeniami, a taktiež nám to ponúka uľahčenie práce ohľadom definície CAN správ vďaka možnosti používať už existujúce riešenia pre vytváranie DBC. Preto jednou z dôležitých súčastí systému je možnosť importovať DBC súbor, ktorý bude dekódovať prichádzajúce správy zo zbernice CAN.

Najžiadanejšou funkcionalitou je možnosť **grafov pre zobrazovanie prichádzajúcich dát**. Či už ide o zobrazovanie jedného zdroju dát, alebo viacerých v jednom grafe, stále ide o nejednoduchú úlohu, na ktorú treba dbať zvýšený ohľad pri implementácii. Ide o to, že za použitia spomínaných webových technológií je zložitá implementácia takto hardvérovo náročných funkcií. Implementáciu danej funkcionality si bližšie popíšeme v časti Implementácia požadovanej funkcionality.

**Ukladanie a načítavanie merania** (tzv. import a export) je samozrejme nevyhnutnou funkcionalitou, ktorou musí takáto telemetria disponovať. Bez týchto funkcií by telemetria bola len obyčajným zobrazovačom dát, kde by sme dáta analyzovať nevedeli, a bola by aplikácia nepoužiteľná.

Jednou z požiadaviek je možnosť vytvárania vlastných komponentov. Je to z dôvodu zjednodušenia vývoja, keďže každý, kto vytvára nový zdroj dát do formule, si môže otestovať

funkcionalitu dopredu, ale taktiež máme možnosť prispôsobiteľnosti samotného rozhrania, čo nám ponúka určitú flexibilitu napríklad pri efektívnom využití viacerých klientov, pričom každý sleduje inú časť vozidla.

Ďalšou z požiadaviek, ktorá úzko súvisí s vlastnými komponentami je možnosť **upravovania rozloženia** v rámci aplikácie, a taktiež možnosť zmeny veľkosti okien so zobrazovačmi dát pre lepšiu prehľadnosť a konfigurovateľnosť. Toto riešenie je potreba implementovať intuitívne, a hlavne zamedziť náhodnému upraveniu rozloženia. To znamená, že je za potreby pridať funkciu, ktorá zamkne rozloženie a veľkosť okien práve pre zamedzenie opísaného problému.

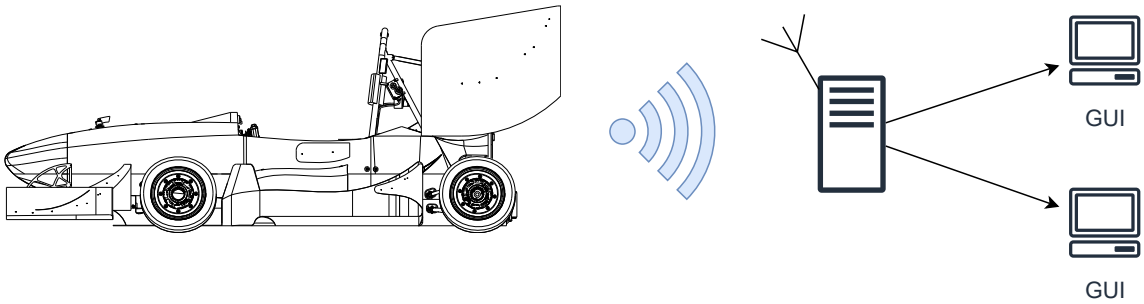
Použitie modulov Xbee pre bezdrôtovú komunikáciu je nevyhnutné, pretože toto zariadenie bolo zakúpené, a vybraté členmi tímu TU Brno Racing ešte pred vývojom tejto aplikácie.

## 4.3 Návrh riešenia

V tejto kapitole je popísaný postup vytvárania návrhu, a výsledný návrh riešenia celého telemetrického systému od monopostu až po obrazovku používateľa.

### Požadovaná architektúra telemetrie

Pri architektúre aplikácie je dôležité myslieť na to, aký spôsob použitia bude uplatňovaný, a aké sú požiadavky na funkcionalitu. Keďže chceme aby sa mohlo pripojiť viacero zariadení, ale k autu sa vieme pripojiť len jedným, tak sme zvolili kombináciu klient-server architektúry, ktorá je implementovaná mimo auta, až po point-to-point pripojenie priamo k autu bezdrôtovo (Obr. 4.3).

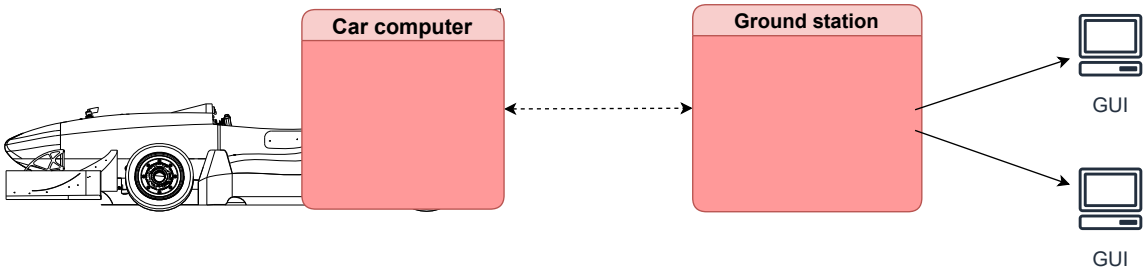



Obr. 4.3: Zjednodušená architektúra telemetrie

Naším cieľom je zachytávať všetky správy, ktoré sú zasielané na viacerých zberniciach CAN v rámci auta po celú dobu zapnutia napájania. Tieto dáta sú zachytávané na vozidle a zasielané do používateľského rozhrania s možnosťou zobrazenia týchto dát či už v reálnom čase, alebo aj retrospektívne od danej časovej značky. Zariadenie zbierajúce dáta (*Car computer*) je nainštalované fyzicky vo vozidle, a bezdrôtovo zasiela tieto správy klientom.

Pre podporu viacerých klientov je potrebné mať klientskú stanicu, ktorá môže byť osobným počítačom, alebo aj špecializovaným zariadením. Táto klientská stanica, nazývaná *Ground station* prijíma dáta prichádzajúce z *Car computer*, a preposiela ich všetkým pripojeným klientom a teda slúži ako prostredník medzi používateľmi a vozidlom. Je to server pre klientov s grafickým rozhraním, ktorí si chcú v danom čase zobrazovať dáta, poprípade

si pretáčať priebeh správ v čase. Toto prepojenie medzi klientmi a serverom (Obr. 4.4) je realizované po počítačovej sieti, a nezávisí od daného prenosového média.




Obr. 4.4: Architektúra fyzických zariadení

Zariadenie nainštalované na vozidle musí byť taktiež vybavené možnosťou ukladania dát. Ak by sa nikto k vozidlu nepripojil, dáta by boli navždy stratené, a to nemôžeme dovoliť. Práve preto sa všetky dáta zo zberníc ukladajú na SD kartu, ktorou je vybavený *Car computer*. Následne toto zariadenie preposiela správy do bezdrôtového modulu XBee. Xbee moduly iba jednoducho preposielajú celú komunikáciu prichádzajúcu po protokole UART[2] na jeho protichodný kus, kde zas tieto prichádzajúce dáta z UARTu spracujeme na *Ground station*, a následne prepošleme klientom.

### 4.3.1 Predpokladané parametre aplikácie

Táto časť slúži ako sumár všetkých parametrov a požiadaviek na aplikáciu. Telemetria obsahuje zariadenie na vozidle, so zariadením prijímajúce dáta, na ktoré sa napájajú používateľské rozhrania. Aplikácia musí podporovať zaznamenávanie dát (bez ohľadu na pripojených klientov). Z požiadavkov „zákazníkov“ vychádza, že musí byť dostupná možnosť zobrazovania grafov, exportu a importu meraní, mať možnosť zobrazovať stavy jednotiek vo vozidle. Okrem iného aplikácia musí byť schopná odosielať dáta aj späť na zbernicu CAN a musíme mať možnosť zobrazenia sumárnej tabuľky všetkých prijatých hodnôt.

Okrem konkrétnych funkčných požiadavkov samozrejme musíme dbať na to, aby bolo výsledné riešenie rýchle a jednoduché na inštaláciu. Veľmi dôležitý ohľad treba dbať na to, aby bolo jednoduché na používanie, a hlavne použiteľné pre prácu v teréne pri testovaní alebo závodoch. Práve posledný bod je zložité kvantitatívne zhodnotiť, pretože len čas ukáže, či je takéto používateľské rozhranie užitočné, a taktiež praktické pre požadované použitie. Taktiež treba dbať aj na dostatočnú optickú kontrastnosť obsahu, a taktiež nízku spotrebu energie zariadenia. Robustnosť a perzistentnou je taktiež veľmi dôležitá, pretože strata cenných dát nás môže stáť aj niekoľko umiestnení vo výslednom rebríčku.

[2] https://www.nxp.com/docs/en/data-sheet/SCC2691.pdf

# Kapitola 5

# Postup riešenia

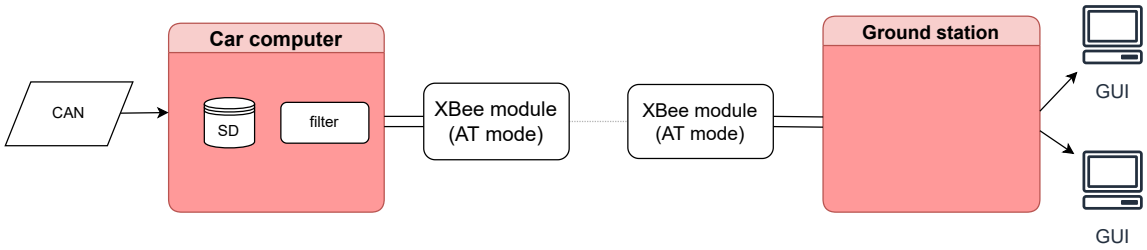
Táto kapitola popisuje detailný postup riešenia od návrhu architektúry grafického používateľského rozhrania, cez implementáciu jednotlivých funkcionalít až po funkčný popis práce s aplikáciou.

## 5.1 Architektúra telemetrie

V tejto časti si vysvetlíme jednotlivé body architektúry spolu s riešeniami, ktoré sme sa rozhodli vybrať, aby čo najviac vyhovovali definovanej architektúre

**Car computer** zbiera všetky dáta z CAN zberníc a ukladá ich do lokálnej databázy (Local DB). Keďže dáta, ktoré chceme preposielať ďalej majú vyššiu šírku pásma ako dokáže zasielať bezdrôtový modul, je potreba implementácie CAN filtra, cez ktorý sú filtrované správy zasielané rovno na XBee modul, ktorý slúži na výber zdroju dát (Live dáta z CAN filtra určené na telemetriu počas závodu, alebo dáta z lokálnej databázy, ktorých množstvo prevažuje rýchlosť komunikácie Zigbee, a teda tento spôsob získavania dát je určený na príjem dát mimo jazdy). *Data interface* nám taktiež ponúka možnosť preposielania správ späť na CAN zbernice vo vozidle.

**XBee modul** podporuje viacero protokolov, pre jeho jednoduchosť sme si vybrali Zigbee, a to aj pre to že najviac sedí nášmu určeniu[1], keďže chceme point-to-point spojene s minimálnou réžiou a konfiguráciou. Tento modul je nakonfigurovaný v móde AT - Application transparent mode, ktorý len preposiela celú komunikáciu na jeho sériovú linku pomocou protokolu UART, ktorou je pripojený či už k počítaču na aute, alebo serveru.




Obr. 5.1: Architektúra - Prenos dát

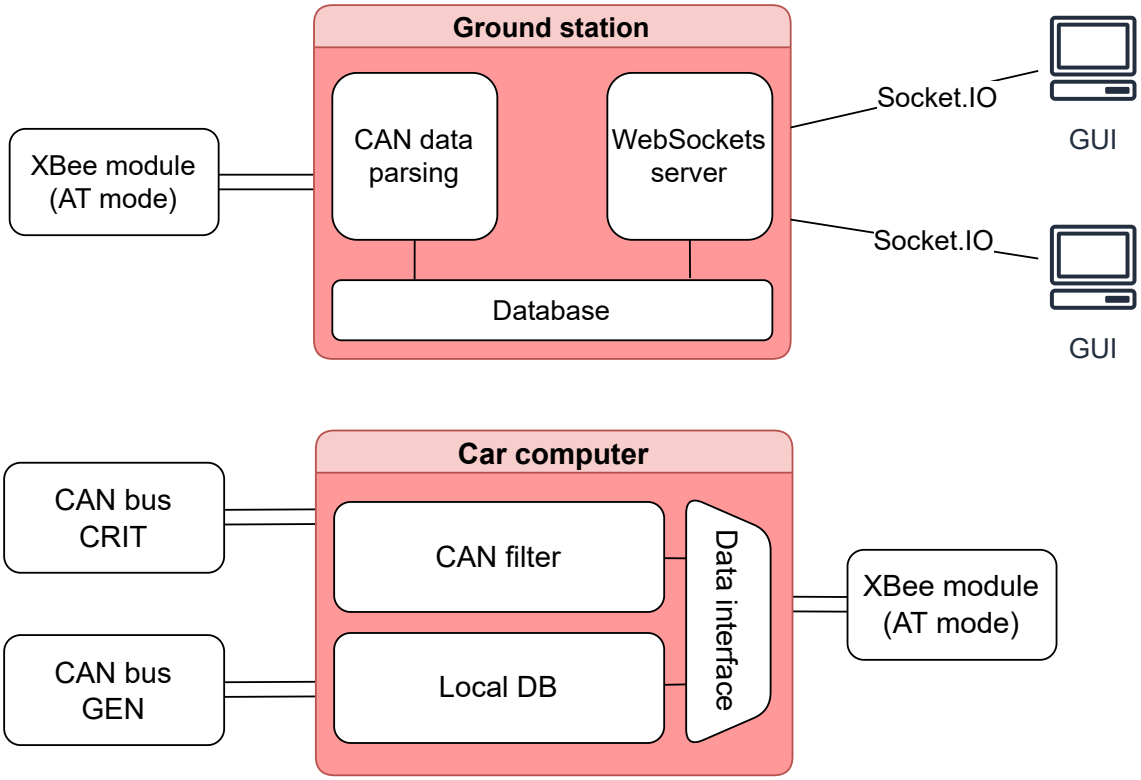
**Ground station (server)** zbiera celú komunikáciu, ktorá nám prichádza z XBee modulu, následne spracováva dáta z CAN zbernice, a ukladá ich do svojej databázy. Na tomto

[1] https://xbplib.readthedocs.io/en/latest/getting_started_with_xbee_python_library.html

zariadení taktiež beží komunikačný protokol WebSockets, ktorým ako server informuje všesmerovou[2] správou svojich klientov o novej správe, ktorá dorazila od auta, a taktiež podporuje pripojenie nového klienta, ktorému po pripojení zašle obsah aktuálnej databázy, aby všetci používatelia videli rovnaké dáta.

**Klient (grafické používateľské rozhranie)** je koncová aplikácia, ktorá podporuje prechádzanie v čase v rámci telemetrie. Preto klient si musí držať aktuálny stav, a taktiež predchádzajúce obsahy jednotlivých správ. A teda pri novom pripojení dostane od servera obsah databázy, a následne pri prichádzajúcich správach si rozširuje obsah svojej databázy o dané správy. Používateľ si vie zobrazovať grafy, tabuľky, a má k dispozície nástroje na prácu s dátami, a taktiež možnosť exportu dát do formátu CSV pre ďalšiu analýzu.

Týmto sme si navrhli celú architektúru aplikácie od zariadenia na vozidle až po koncového užívateľa (Obr. 5.2). Vďaka tomuto si vieme rozdeliť implementáciu na jednotlivé prvky riešenia, a prácu na jednotlivých častiach si vieme rozdeliť.



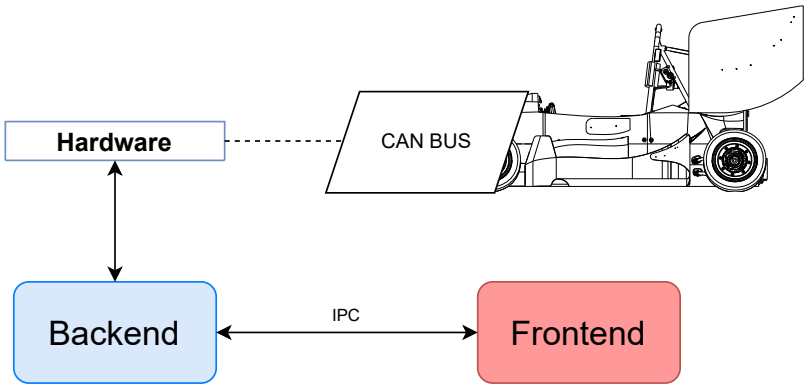

Obr. 5.2: Návrh systému a jeho výsledná architektúra

## 5.2 Architektúra grafického používateľského rozhrania

Narozdiel od celej telemetrie, kde sa pracuje s *GUI* len ako atomickým prvkom, pri implementácii bolo potrebné riadne zadefinovať architektúru aj z pohľadu používateľského rozhrania. Používateľská aplikácia je postavená na frameworku Electron, ktorý nám slúži ako obal pre Reactovú aplikáciu. Tieto dve časti sa dajú predstaviť ako backend a frontend aplikácie. No ak chceme aby spolu komunikovali, medzi takýmito dvoma modulmi je po-

[2] broadcast

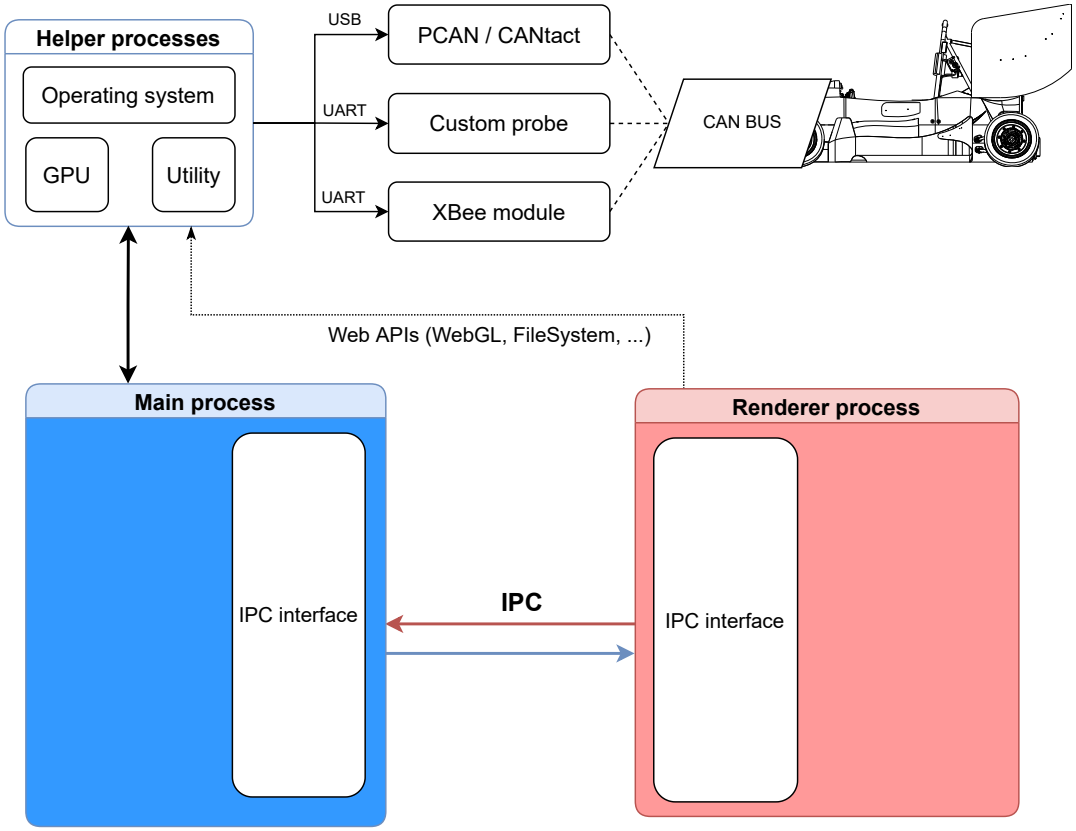
trebné mať aplikačné rozhranie. Electron nám ale poskytuje medziprocesovú komunikáciu (IPC - inter-process communication), ktorá je odlišná oproti protokolu HTTP, tým, že sa správy medzi jednotlivými komunikantmi zasielajú na základe názvu (al. kľúča) správy, a obsah je telom danej správy. Tento spôsob komunikácie sa používa práve pre posielanie dát medzi dvoma hlavnými prvkami Electron aplikácie. A to main process, čo reprezentuje samotné okno spúšťané v operačnom systéme, ktoré má prístup ku hardvéru PC, a renderer procesy, ktoré slúžia na zobrazovanie obsahu v danom okne. Takýchto renderer procesov môže byť viac, no my používame jeden.




Obr. 5.3: Prepojenie grafického rozhrania s hardvérovým rozhraním

*Main process* beží v prostredí Node.js, s tým, že má prístupné aplikačné rozhrania na komunikáciu s operačným systémom, a perifériami zariadenia na ktorom je spúšťaný. Toto nám prináša možnosť priamej komunikácie s modulmi CAN rozhrania na fyzickej vrstve, a odľahčuje nás od potreby použitia iných programovacích jazykov na nižšej úrovni. Main process komunikuje pomocou IPC s jeho renderer procesmi za pomoci serializácie zasielaných objektov (v skutočnosti ide o marshalling[3] pomocou structured clone algoritmu [4]. Následne má renderer process (v našom prípade Reactová aplikácia) dostupné funkcie na prijímanie takýchto správ na princípe Node eventov, ktoré už dobre poznáme z popisu technológií[7]. Dôležitý problém pri návrhu tejto architektúry bolo umožnenie použitia Node eventov práve v prostredí React.js. React svoje komponenty neustále obnovuje, aby mali obsah najaktuálnejší. To prináša mnoho problémov práve ak vytvárame event callback v tele komponentu. Ak by sme tento problém neriešili, do niekoľko minút máme plnú pamäť, a teda týmto vytvárame tzv. memory leak. Jedným zo spôsobov ako sa vyhnúť tomuto problému je pridanie automatického zmazania callback funkcie pri znovunačítaní komponentu, s tým že po načítaní sa callback znova vytvorí. Týmto sme zamedzili možným pamäťovým problémom, čo si vieme overiť aj pomocou profilovacích nástrojov poskytovaných webovým prehliadačom.

[3] https://en.wikipedia.org/wiki/Marshalling_(computer_science)
[4] https://developer.mozilla.org/docs/Web/API/Web_Workers_API/Structured_clone_algorithm




Obr. 5.4: Umiestnenie medziprocesovej komunikácie medzi main a renderer procesmi

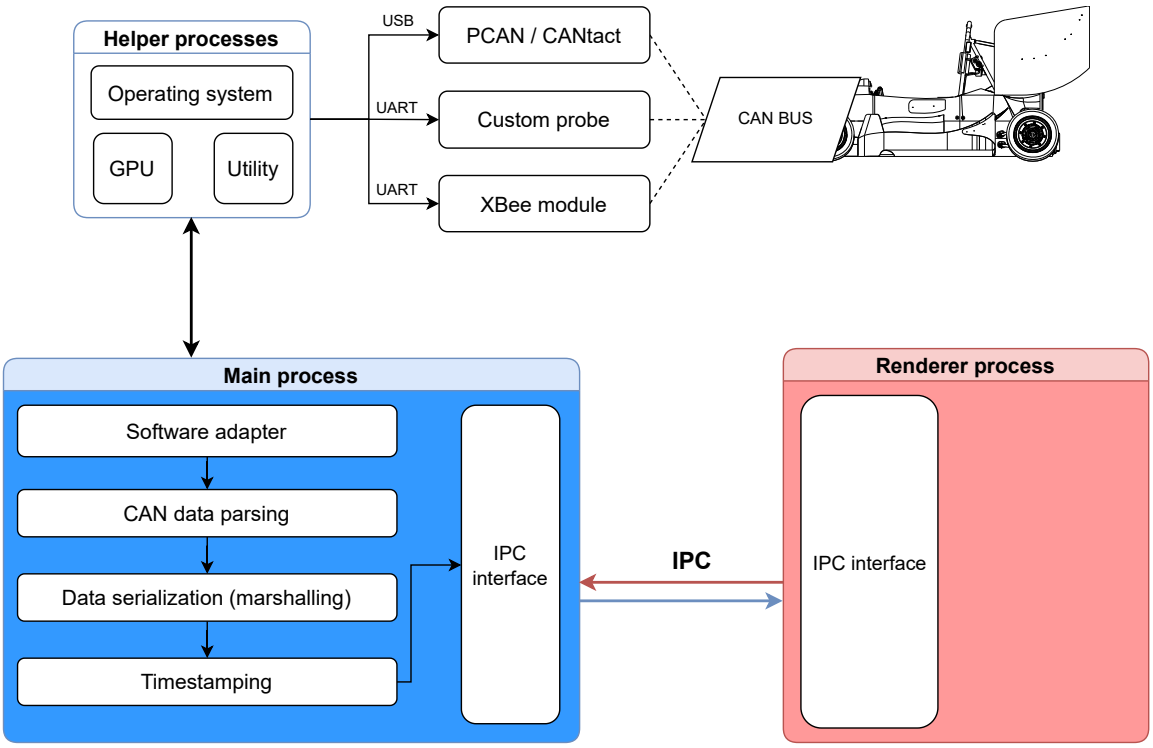
### Main process

*Main process* nám slúži na komunikáciu s hardvérovými perifériami, otvára nám okno aplikácie, a zabezpečuje nám celý beh aplikácie v operačnom systéme nezávisle od platformy (multiplatform).

Pri spúšťaní aplikácie sa inicializuje knižnica *Electron*, ktorá okrem riadenia chýb definuje samotné okno aplikácie s jeho aplikačným menu. Po inicializácii sa spúšťa inštancia *BrowserWindow*, čo reprezentuje náš *Renderer process* a funguje ako obal pre prehliadačové okno, pričom samotný frontend s ním pracuje natívne ako v prehliadači. Ďalším z dôležitých prvkov main processu je funkcionalita IPC komunikácie. Je v nej implementované samotné IPC rozhranie, ktoré sme si definovali vyššie. Najdôležitejšou funkciou tohto súboru je využívanie rozhrania DataSource pre získavanie jednotlivých CAN správ, ktoré zasiela *renderer procesu* pomocou spomínaného IPC rozhrania. Poslednou z častí tohto procesu je rozhranie návrhového vzoru adaptér, ktoré je implementované spôsobom rodičovskej triedy BaseDataSource, ktorej metódy implementujú adaptéry pre rôzne fyzické pripojenia ku CAN.

- *WindowsCantactDataSource* pre použitie CANtact na systémoch Windows
- *LinuxDataSource* pre použitie PCAN a CANtact na Unix podsystémoch
- *SerialDataSource* pre použitie vlastného protokolu na sériovej linke nezávisle od OS.

- *SocketIODataSource* pri používaní Ground station sa pripájajú klienti cez toto rozhranie.




Obr. 5.5: Detailný popis Main process častí

### Renderer process

Tento proces je single-page app (SPA viz. Technológie) vytvorený vo frameworku[5] React, ktorý vie bežať či už v prostredí Electron ako renderer process, alebo aj v prostredí webového prehliadača bez značnej zmeny fungovania aplikácie.

*Dataloader* je vstupný modul slúžiaci na prenos a spracovávanie dát z/do main procesu. Podporuje načítavanie a ukladanie merania. Používa *JSON* súbor, a pri jeho načítaní sa aplikácia dostane do stavu „pozastavená“ s otvoreným časovým bežcom[6] pre posúvanie (prehrávanie) dát v čase.

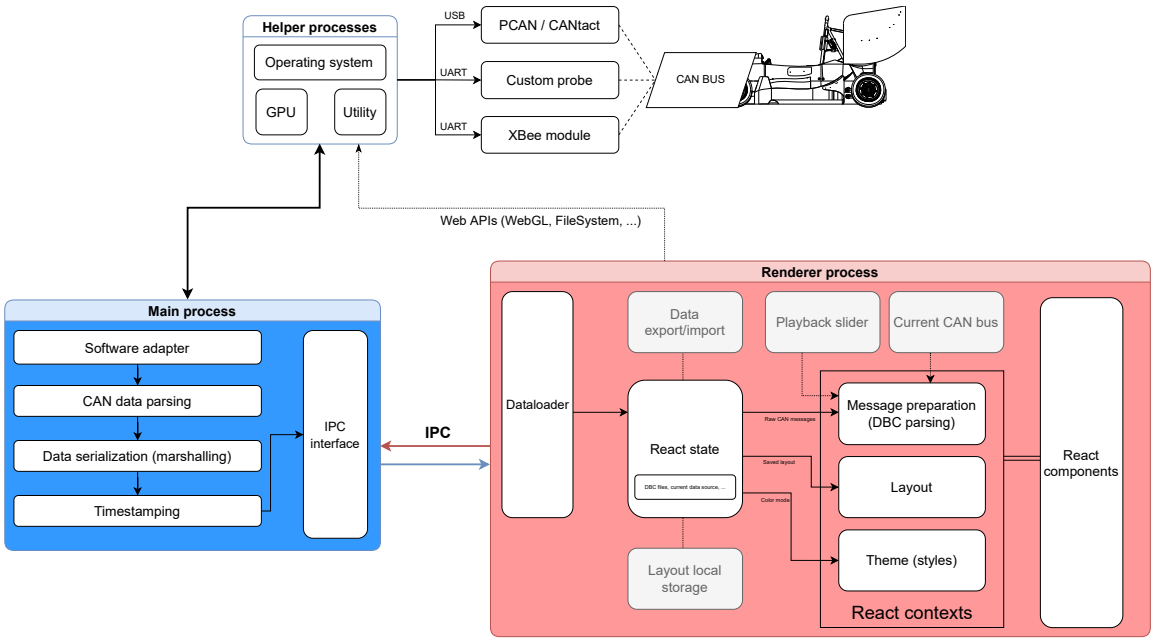
Ďalšia veľmi dôležitá funkcionalita je samotné spracovávanie dát, a ich ukladanie do stavu aplikácie (React state). Tento stav aplikácie musí mať jasne definovanú štruktúru, a musíme vedieť jednoducho (s čo najnižšou časovou náročnosťou) pridávať dáta do takejto štruktúry. Používame na to JavaScriptovský objekt s kľúčom ID správy. Prichádzajúca správa sa pridá ku jej existujúcim záznamom v stave aplikácie[7]. Takýto objekt obsahuje jednotlivé správy uložené zvlášť podľa ich ID v poli, a posledný prvok objektu s kľúčom `t` uchováva časové značky týchto správ. Vďaka týmto značkám je možné jednoducho orezávať správy podľa vybratého času, a tým pádom si vieme pretáčať komunikáciu po zbernici.

[5] knižnici
[6] Playback slider
[7] React state

Okrem iného sa tu implementujú React hooky na získavanie samotných hodnôt nachádzajúcich vo vyššie spomínanom state. Vieme získavať pole všetkých hodnôt danej správy (pre grafy), ale aj len poslednú hodnotu (pre indikátory). Aplikácia komponentom poskytuje jednoduchý spôsob získania všetkých alebo poslednej hodnoty pre dané ID správy. Táto funkcionalita implementovaná hookmi[8] je využívaná všetkými komponentami, ktoré použitím jedinej funkcie majú vždy správnu aktuálnu hodnotu. Dôležité je spomenúť, že tu taktiež je implementovaný systém zálohovania meraní do cache súborov pravidelne pri každom meraní.

Spracovávanie signálov postupuje podľa načítaného DBC súboru, kde sa nájde záznam správy, a správa sa rozdelí na signály. Toto spracovávanie je implementované až pri komponentoch zobrazovaných dáta práve kvôli možnosti flexibility viacerých DBC súborov.

Aktuálne rozloženie okien je synchronizované s localstorage[9] pomocou hooku, ktorý nám zabezpečuje synchronizáciu medzi prehliadačovým *LocalStorage*, a našou aplikáciou. Používateľ teda má stále rovnaké prispôsobené rozloženie medzi spusteniami aplikácie.




Obr. 5.6: Diagram s detailným popisom renderer procesu

## Generované obrazovky

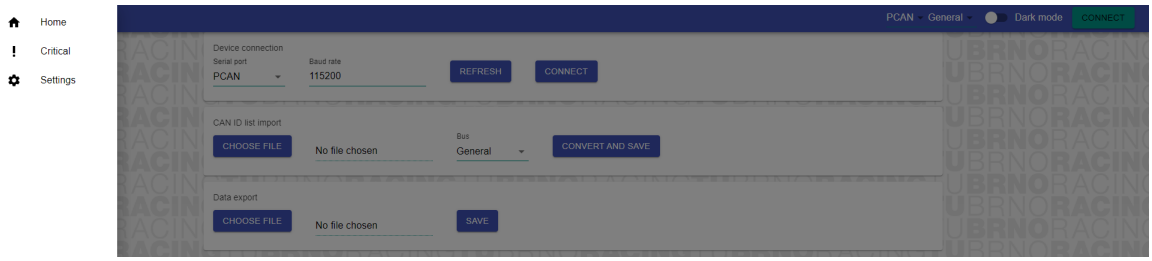
Obsah obrazoviek, a teda komponenty, ktoré sa na jednotlivých obrazovkách zobrazujú, sú dynamicky načítané z JSON súboru a generované pre zvolenú obrazovku. Tieto užívateľsky vytvorené komponenty obsahujú svoj globálny kľúč, pomocou ktorého sa identifikuje korešpondujúci záznam z layout, pomocou ktorého sa určí ich pozícia a veľkosť. Taktiež v tomto súbore sú obsiahnuté hodnoty atribútov týchto komponentov, ako napríklad odkaz na akú obrazovku sa komponent odkazuje, alebo aj label signálu, poprípade ID správy, s ktorou pracuje.

[8] viz. React stav a kontexty (Webové technológie)

[9] https://developer.mozilla.org/docs/Web/API/Window/localStorage

## 5.3 UI a UX

Dizajn aplikácie je založený na UI kite Material UI, z ktorého sú používané základné komponenty používateľského rozhrania ako *AppShell*, *Menu*, tlačidlá, slidery, ... Dlaždicové komponenty taktiež využívajú Material UI na svoje obaľovacie okno v ktorom je uložený samotný ukazovateľ dát. Ako jediný komponent nevyužívajúci túto knižnicu pre vzhľad je LinearGraph, ktorý je naše riešenie[10] za použitia `webgl-plot` knižnice. V menu sa vieme prepínať medzi obrazovkami, a taktiež máme dostupné samotné aplikačné menu z ktorého vieme otvárať a ukladať súbory, obnoviť a zavrieť aplikáciu, a rôzne iné tradičné prvky aplikačného menu. Obrazovky si vie užívateľ sám pridávať alebo odoberať, nastavovať im ikonu, a názov. V rámci obrazovky po odomknutí možnosti úpravy rozloženia si vie presúvať a meniť veľkosť zvoleným komponentom v okne, ktoré si vie vytvárať a upravovať v nastaveniach. Po zamknutí úpravy rozloženia sa používateľ v časti AppShell vie pripojiť na jemu zvolené zariadenie a zbernicu.




Obr. 5.7: Výber obrazovky z menu (vľavo), a obrazovka s nastaveniami

## 5.4 Implementácia požadovanej funkcionality

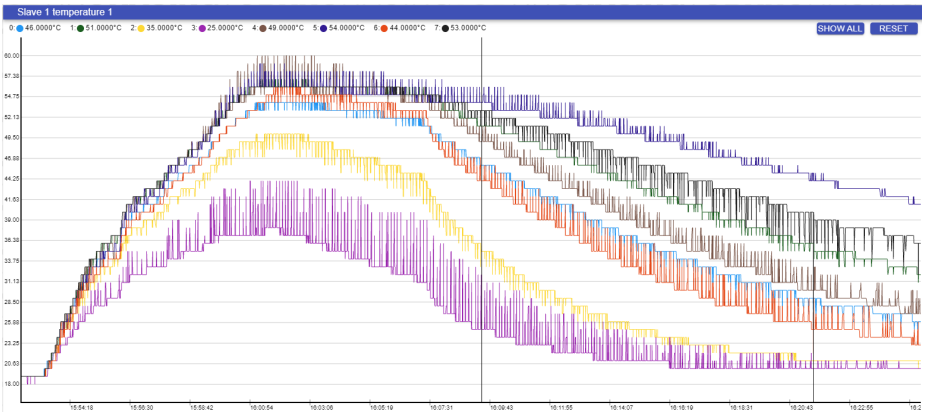
### Grafy

Prostredie webového prehliadača, v ktorom aplikácia beží nie je optimálne na vykresľovanie veľkého množstva dát napríklad vo forme grafov. Dáta môžu chodiť s frekvenciou až 1000Hz, a klasické DOM vykresľovanie (poprípade canvas) na to výkonovo nestačili. Tento problém sme vyriešili použitím vykresľovania grafov akcelerované grafickou kartou pomocou webového API *WebGL*[11]. Vďaka ktorému vykresľovanie grafov nemá takmer žiaden vplyv na celkovú výkonnosť aplikácie.

### Návrhový vzor adaptér

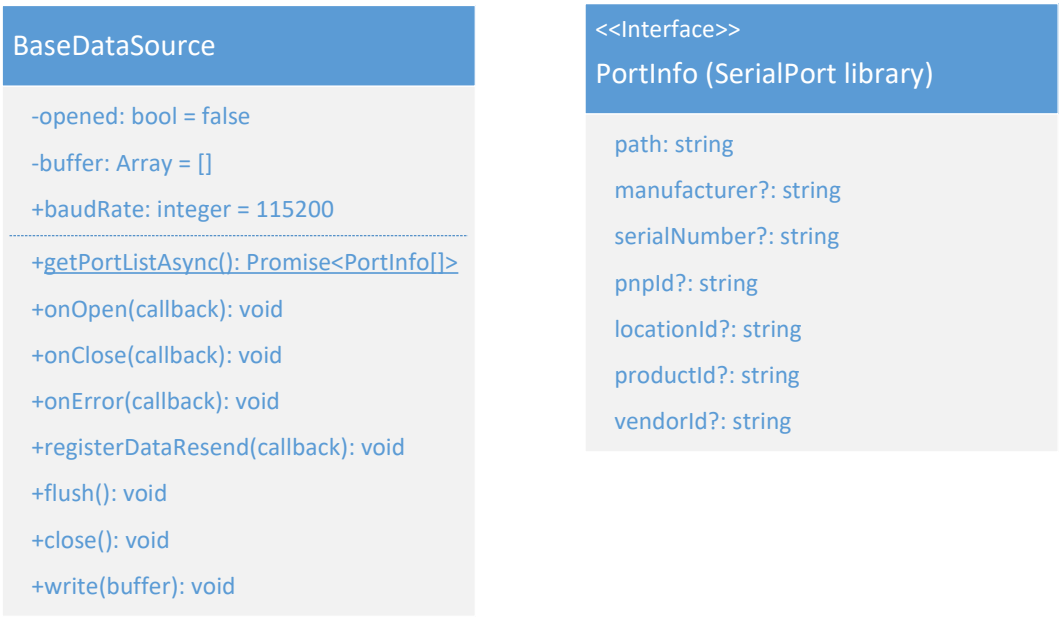
Po navrhnutí vzoru typu adaptér ho bolo potrebné implementovať. Problém s danou úlohou je to, že JavaScript natívne nepodporuje triedy, a preto by tento návrhový vzor nebol realizovateľný. To nám však jednoducho vyriešilo použitie novších ECMAscript[6] štandardov s transpilérom vďaka ktorému vieme písať kód v najnovšom štandarde ES2020 bez potreby riešenia spätnej kompatibility, ktorú tento transpiler (Babel) rieši za nás. Zadefinovali sme si otcovskú triedu *BaseDataSource*, z ktorej sme následne dedili jednotlivé adaptéry pre rôzne zdroje dát. PCAN a CANtact majú rovnaký driver pre Unix, a preto majú spoločný adaptér. Pre náš vlastný protokol sme si vytvorili adaptér, ktorý využíva knižnicu SerialPort,

[10] V tomto prípade vyvinuté tímom TU Brno Racing
[11] https://developer.mozilla.org/docs/Web/API/WebGL_API




Obr. 5.8: Grafy v aplikácii

ktorú sme si popisovali vyššie na prácu so sériovou linkou. *IPC interface*, ktorý celú komunikáciu zastrešuje nám z pohľadu GUI poskytuje len jedno rozhranie s ktorým pohodlne vieme pracovať, a taktiež vieme implementovaný adaptér veľmi jednoducho rozširovať.




Obr. 5.9: BaseDataSource diagram triedy

Implementácia smerovania (routing) a navigácie v rámci aplikácie je popísaná v Prílohe D.

### Prispôsobiteľné rozloženie okien

Funkcionalitu presúvania, zmeny veľkosti a ukladania komponentov nám zaručuje knižnica `react-grid-layout`[12], ktorá nám poskytuje jednoduché rozhranie pre vytváranie takýchto rozložení, a taktiež má jednoduchý formát ukladania vytvoreného rozloženia v súborovom formáte JSON, s tým, že každý komponent má svoj kľúč, pozíciu, a rozmery. Bohužiaľ toto riešenie sa vzťahuje len na jednu obrazovku, a medzi obrazovkami sa komponenty miešajú. Preto sme komponenty za pomoci automatického generovania komponentov okľúčovali globálnym ID, ktoré je unikátne v rámci aplikácie. Toto riešenie nám zároveň odľahčuje problémy s odstraňovaním, a opätovným pridávaním prvkov „custom layoutu“.

### Ukladanie a formát dát

Ako sme si vysvetľovali pri popise knižnice React, táto knižnica si interne ukladá štruktúru elementov HTML formátom VirtualDOM. Táto technika pracovania s elementami nám nielen zrýchľuje prácu s nimi, no taktiež prináša aj iné výhody ako napríklad Reactový stav. Vytvorili sme si vlastný hook (Webové techonológie - hooky)), ktorý nám sprístupní naše dáta na akomkoľvek mieste chceme. My sme si teda navrhli štruktúru dát (obr. 6.8) s ktorou pracujeme všade v aplikácii, a na prácu s týmito dátami nám slúžia obslužné funkcie ktoré už využívajú jednotlivé komponenty ako hlavný zdroj dát pre zobrazovanie.


Obr. 5.10: Štruktúra ukladaných dát

Tento formát dát je následne serializovaný, a ukladaný priamo do JSON formátu. Keďže React podporuje priamy import JSON formátu, výber bol jasný. Taktiež práca s týmto formátom je v JavaScripte veľmi rýchla, a jednoduchá, čo len uľahčilo toto rozhodovanie. Súbor v tomto formáte je automaticky ukladaný do priečinku inštalácie aplikácie (resp. servera) s názvom cache, a slúži na zálohu merania pri prípadnom zlyhaní zaznamenávania dat. Taktiež je dostupná možnosť uloženia merania, ktoré funguje takmer rovnako ako cache, no používateľ si zadá cestu a názov súboru sám.

### 5.4.1 DBC spracovávanie

Správa, zložená len z ID a 64-bitového čísla, ktorá nám prišla z „backendu“ musí byť správne spracovaná, a rozdelená aby priniesla danú informáciu. Pri spracovávaní správy je nájdená existujúcu definícia JSON definícii správ podľa jej ID. Táto definícia vznikla prevedením DBC súboru pomocou knižnice `dbc-to-json` do formátu JSON pre jednotlivé zbernice, ktoré sa nachádzajú vo vozidle. Ak nenájdeme korešpondujúcu definíciu ku správe, vytvorí

[12] https://github.com/react-grid-layout/react-grid-layout

sa obecná definícia, a pracuje sa s ňou tak isto ako s ostatnými. Ak sme našli danú správu, dáta spracujeme a rozdelíme na rôzne signály podľa definície DBC. To znamená, že napríklad z 64-bitovej správy vytiahneme 4 po sebe idúce 16-bitové čísla, a týmto zostavíme dané pole hodnôt signálov. Ak definícia správy obsahuje tabuľku korešpondujúcich hodnôt (Value table), čo je funkcionalita podporovaná DBC[5] pre výčtové typy a bitové polia, tak si dané hodnoty preložíme podľa tejto tabuľky na reťazce popisujúce hodnotu, a následne tieto skonvertované, a rozšírené hodnoty s ich definíciou vrátime funkcii, ktorá konverziu dát volala.

## 5.5 Výsledná funkcionalita telemetrie a výsledky riešenia

Systém pozostáva z niekoľko častí, z ktorých najdôležitejšie prvky sú grafické používateľské rozhranie, prepojenie s vozidlom, a zariadenie na zber dát nainštalované fyzicky na formuly. Telemetria vie pracovať v dvoch módoch. Jedným z nich je priame zaznamenávanie dát na počítači. To prebieha výberom zariadenia, na ktoré sa chceme pripojiť, a výberom zbernice, na akú sa pripájame. Po stlačení tlačidla „Connect“ sa pripojíme, a zaznamenávanie dát začína. Druhý mód je pripojenie na vyššie spomínaný *Ground station*. Toto zariadenie


Obr. 5.11: Výber zariadenia a zbernice s tlačidlom na pripojenie

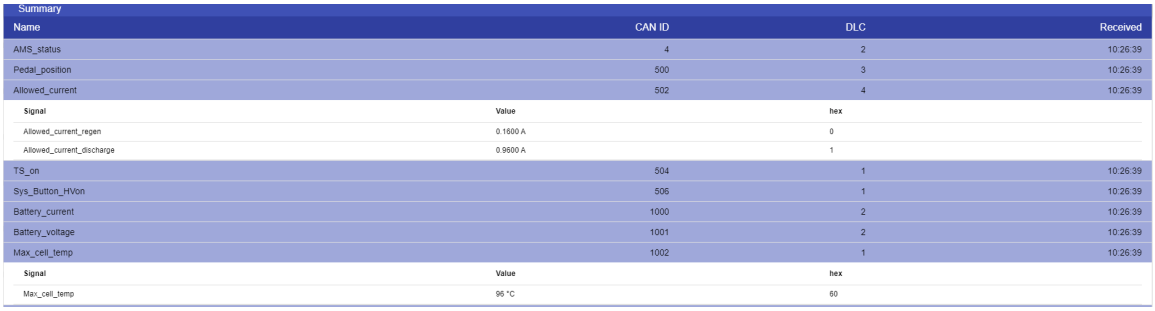
nám poskytuje vzdialenú telemetriu, na ktorú sa vieme napojiť rovnako ako pri priamom zaznamenávaní dát, ale pri výbere iného hardvérového rozhrania. Z pohľadu používateľa je rozhranie identické, no na pozadí grafické rozhranie komunikuje so vzdialenou telemetriou, ktorá podporuje viacero pripojení súčasne, a zároveň zaznamenáva všetku príchodziu komunikáciu zo zbernice CAN.

Aplikácia nemá problém s plnou „produkčnou“ komunikáciou na zbernici. Toto sme si overili testovaním telemetrie so zapnutými všetkými riadiacimi jednotkami v aute a počas jazdy. Grafické rozhranie výkonovo zvládalo spracovávať dané množstvo dát bez zvýšenia odozvy aplikácie. Príloha A nám poskytuje obrázky z používania aplikácie v praxi s reálnymi dátami.

Dôležité je overiť splnenie požiadaviek, ktoré sme si definovali v predchádzajúcich kapitolách, menovite v sekcii Predpokladané parametre aplikácie kapitoly *Požiadavky a návrh riešenia*.

Zobrazovanie grafov je implementované za použitia API WebGL, a s výkonom problém nemajú. Import a export merania prebieha pomocou uloženia vnútorného React stavu do formátu JSON na disk. Znovu-načítanie funguje analogicky. Stavy jednotiek sú mapované z DBC súboru, kde sú definované textové vysvetlivky daných číselných hodnôt (stavu jednotky). Zaznamenávanie dát bez pripojenia grafického rozhrania prebieha za prítomnosti *Car computer* na vozidle automaticky pri zapnutí napájania. Export do CSV je zaručený nami implementovaným modulom pre prevod JSON na CSV. Bezdrôtová telemetria je zaručená pomocou XBee páru bezdrôtových modulov.

Sumárna tabuľka zobrazuje jednotlivé prichádzajúce správy, s možnosťou rozkliku danej správy, po ktorom sa nám zobrazia jednotlivé signály tejto správy. Tabuľka zobrazuje spracované prichádzajúce hodnoty spolu s časovou značkou, ktorá reprezentuje čas príchodu správy.




Obr. 5.12: Tabuľka všetkých hodnôt zaznamenaných v používateľskom rozhraní

Inštalácia prebieha použitím inštalátora jedným kliknutím. Po inštalácii sa nám aplikácia hneď zapne, a ďalšia konfigurácia nie je potrebná. Rýchlosť aplikácie sme odmerali profilovaním vo vývojárskych možnostiach aplikácie. Výsledok profilovania môžete vidieť v Prílohe F. Pri testovaní členmi TU Brno Racing tímu sme neprišli k väčšiemu problému návrhu aplikácie. Po ukážke práce s takýmto rozhraním nemal žiaden používateľ problém pochopiť rozhranie, a taktiež si vedel jednoducho prispôsobiť rozloženie a okná aplikácie. Zároveň sme si ujasnili ďalšie požiadavky od členov.

# Kapitola 6

# Zhrnutie a záver

Cieľom tejto práce bolo vytvoriť systém pre monitorovanie dát prichádzajúcich zo zbernice CAN monopostu Dragon e2 Formuly študent, na ich analýzu, ale taktiež aj na konfiguráciu tohto vozidla. Tento cieľ bol splnený plne bez menších výhrad, a po testovaní kolegami TU Brno Racing ako cieľovými používateľmi aplikácia vyhodnotená ako intuitívna na používanie, rýchla na prácu a robustná s jednoduchou rozšíriteľnosťou.

Preštudoval som podklady ku zbernici CAN a jej aplikáciu v tíme TU Brno Racing spolu s možnosťami ukladania a prenosu uložených dát. Spôsob ukladania dát bol navrhnutý a popísaný v časti návrhu riešenia. Grafické rozhranie k telemetrickým dátam s využitím webových technológií bolo úspešne navrhnuté, implementované a otestované kolegami tímu Formuly Student. Zber a ukladanie telemetrických dát zo zbernice CAN je implementovaný pomocou fyzického zariadenia nainštalovaného na vozidle.

Výsledkom je telemetrický systém, ktorý podporuje pripojenie neobmedzeného počtu zberníc CAN, pričom prenosová rýchlosť je limitovaná bezdrôtovou technológie. Systém zvláda vizualizovať dáta prichádzajúce až 100 000 krát za sekundu. Zaznamenávanie takýchto dát má prakticky neobmedzenú rýchlosť. Na vozidlo sa dokáže pripojiť až 254 grafických rozhraní.

Pomerne zložitá úloha nás čakala pri práci s binárnymi dátami v JavaScripte (problém vysvetlený v prílohe E.2). Spracovávanie sekvenčných správ pri vysokej rýchlosti dá zabrať aj jazykom ako C++, a práve preto nás prekvapila výkonnosť jazyka JavaScript, ktorý zvláda prichádzajúce správy spracovávať v reálnom čase. V neposlednom rade by som rád spomenul formát DBC, ktorý sa ukázal ako veľmi jednoduchý a dobre vymyslený textový formát so širokou škálou využitia.

Do budúcna by som ako prvé chcel pridať funkcionalitu intercomu[1] pomocou protokolu WebRTC, a následne podporu pre fotobunku a GPS trackovanie pozície a časov kôl auta, ako je popísané pri Cosworth telemetrii. Taktiež by som rád pridal podporu pre CANOpen špecifikáciu s podporou kombinácie DBC a EDS súborov[2], ktorá sa bude používať pri *driverless vozidle*. Taktiež by som rád rozšíril funkcionalitu GUI pre autonómny systém s podporou ROS operačného systému. V neposlednom rade je dôležité rozsiahle používateľské testovanie na zistenie ďalších požiadaviek používateľov, a následná implementácia klasického vývojového cyklu aplikácie s plne funkčným CI/CD prostredím s automatickými kompiláciami inštalačných súborov.

[1] komunikácia s pilotom na diaľku
[2] Slúžia na definíciu CANOpen uzlov

# Literatúra


[1] Arce Plevnik, L., Cohen, A. a Shor, T. *IoT IN AUTOMOTIVE INDUSTRY: CONNECTING CARS*. Apríl 2016. 8 s. DOI 10.13140/RG.2.1.2116.6489. Dostupné z: https://www.researchgate.net/publication/301494385_IoT_IN_AUTOMOTIVE_INDUSTRY_CONNECTING_CARS.

[2] Benyak, P. *Systém na evidenciu rezervácií a výpožičiek materiálov na katedre*. Ružomberok, SK, 2019. Bakalárska práca. Katolícka univerzita v Ružomberku PF KU KIN. Dostupné z: https://opac.crzp.sk/?fn=detailBiblioForm&sid=D39197AF4F77EC855AA82BACEBCE.

[3] CiA 102. *CAN physical layer for industrial applications*. Standard, 3. vyd. Nuremberg, DE: CAN in Automation e. V., február 2010.

[4] CiA 303 1. *CiA Recommendation 303-1*. Standard, 1. vyd. Nuremberg, DE: CAN in Automation e. V., december 2001.

[5] CSS Electronics. *CAN Bus, the ultimate guide* [online, offline]. 2022 [cit. 2022-08-05]. Dostupné aj v offline verzii, viz. príloha B. Dostupné z: https://www.csselectronics.com/pages/can-bus-intros-tutorials.

[6] ECMA 262. *ECMAScript ISO/IEC 16262:2011*. Standard, 12. vyd. Geneva, CH: Ecma International, október 2020. 880 s.

[7] *WebView2 and Electron* [online]. Júl 2021 [cit. 2022-05-09]. Dostupné z: https://www.electronjs.org/blog/webview2.

[8] ISO 11898 1:2015(E). *Controller area network (CAN) (Data link layer and physical signalling)*. Standard 1, 2. vyd. Geneva, CH: International Organization for Standardization, december 2015. 85 s.

[9] ISO 11898 2:2016(E). *Controller area network (CAN) (High-speed medium access unit)*. Standard 2, 2. vyd. Geneva, CH: International Organization for Standardization, december 2016. 30 s.

[10] Macko, M. *Bateriový monitorovací systém pro Formuli Student*. Brno, CZ, 2021. Bakalárska práca. Vysoké učení technické v Brně. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/134677.

[11] *SerialPort docs* [online]. 2022 [cit. 2022-05-09]. Dostupné z: https://serialport.io/docs/.

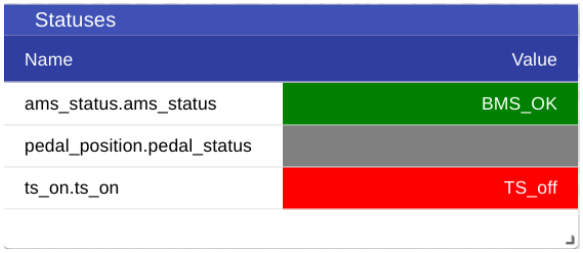
[12] Milet, T. *Úvod to OpenGL* [online]. Apríl 2016 [cit. 2022-05-09]. Dostupné z: https://www.fit.vutbr.cz/study/courses/IZG/private/lecture/izg_slide_opengl.pdf.

# Príloha A

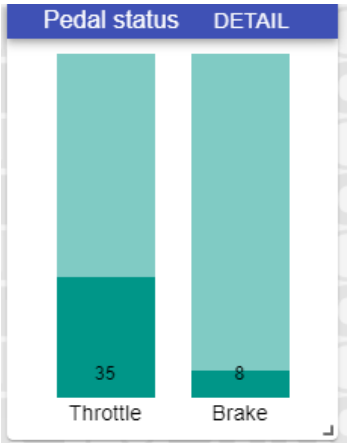
# Ukážky grafického používateľského rozhrania


Obr. A.1: Aplikácia s dátami (svetlý režim)

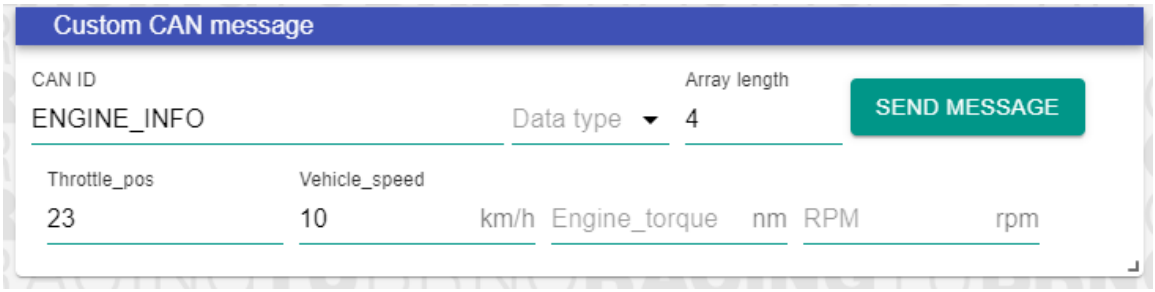

Obr. A.2: Aplikácia s dátami (tmavý režim)




Obr. A.3: Zobrazovanie stavu jednotiek



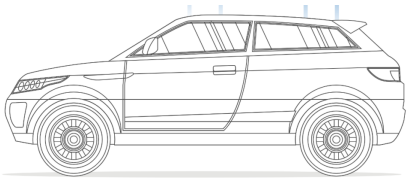

Obr. A.4: Zobrazovanie pozície pedálov




Obr. A.5: Možnosť zasielania vlastných správ

# Príloha B

# Prílohy uvedené na SDHC karte



CAN bus

the ultimate guide

CAN | J1939 | OBD2 | UDS | CANopen | CAN FD | LIN | DBC

CSS Electronics | www.csselectronics.com | contact@csselectronics.com
Last modified: April 20, 2022

Príloha *CAN Bus - The Ultimate Guide* sa nachádza v koreňovom adresári s názvom `CAN-Bus-The-Ultimate-Guide.pdf`



**SocketCAN - The official CAN API of the Linux kernel**

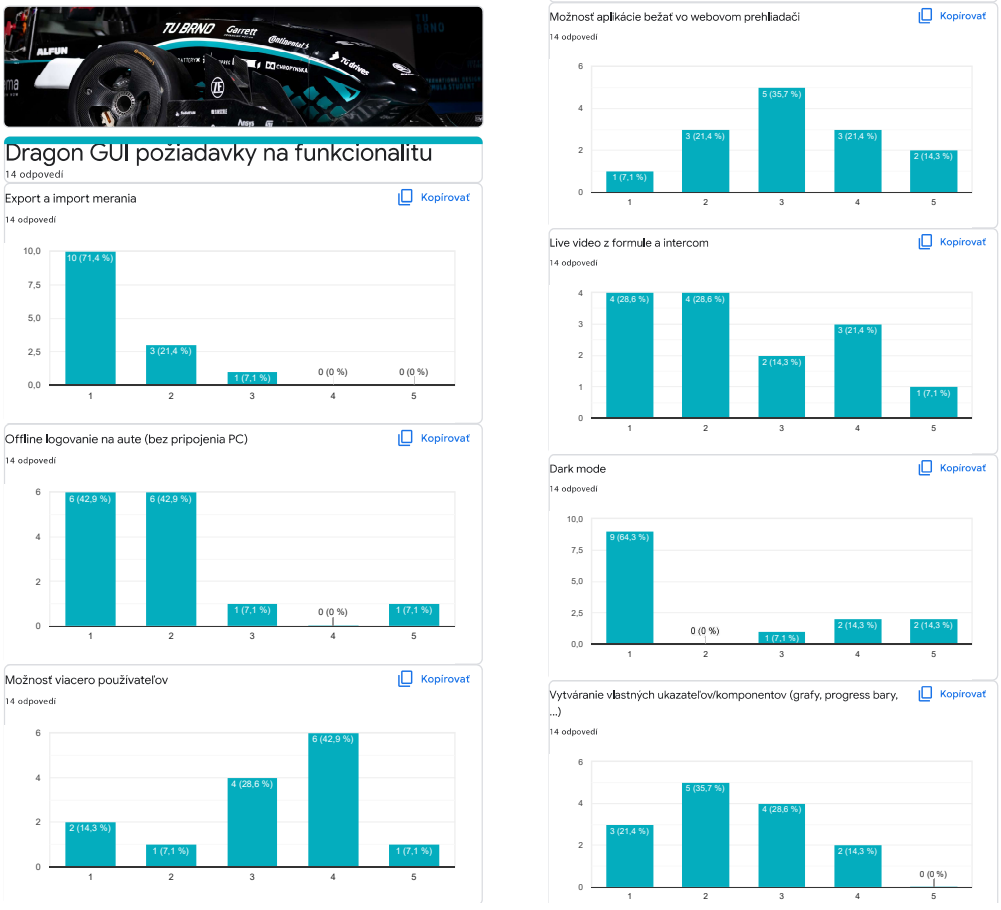
Marc Kleine-Budde, Pengutronix

**SocketCAN, the official CAN API of the Linux kernel, has been included in the kernel more than 3 years ago. Meanwhile, the official Linux repository has device drivers for all major**

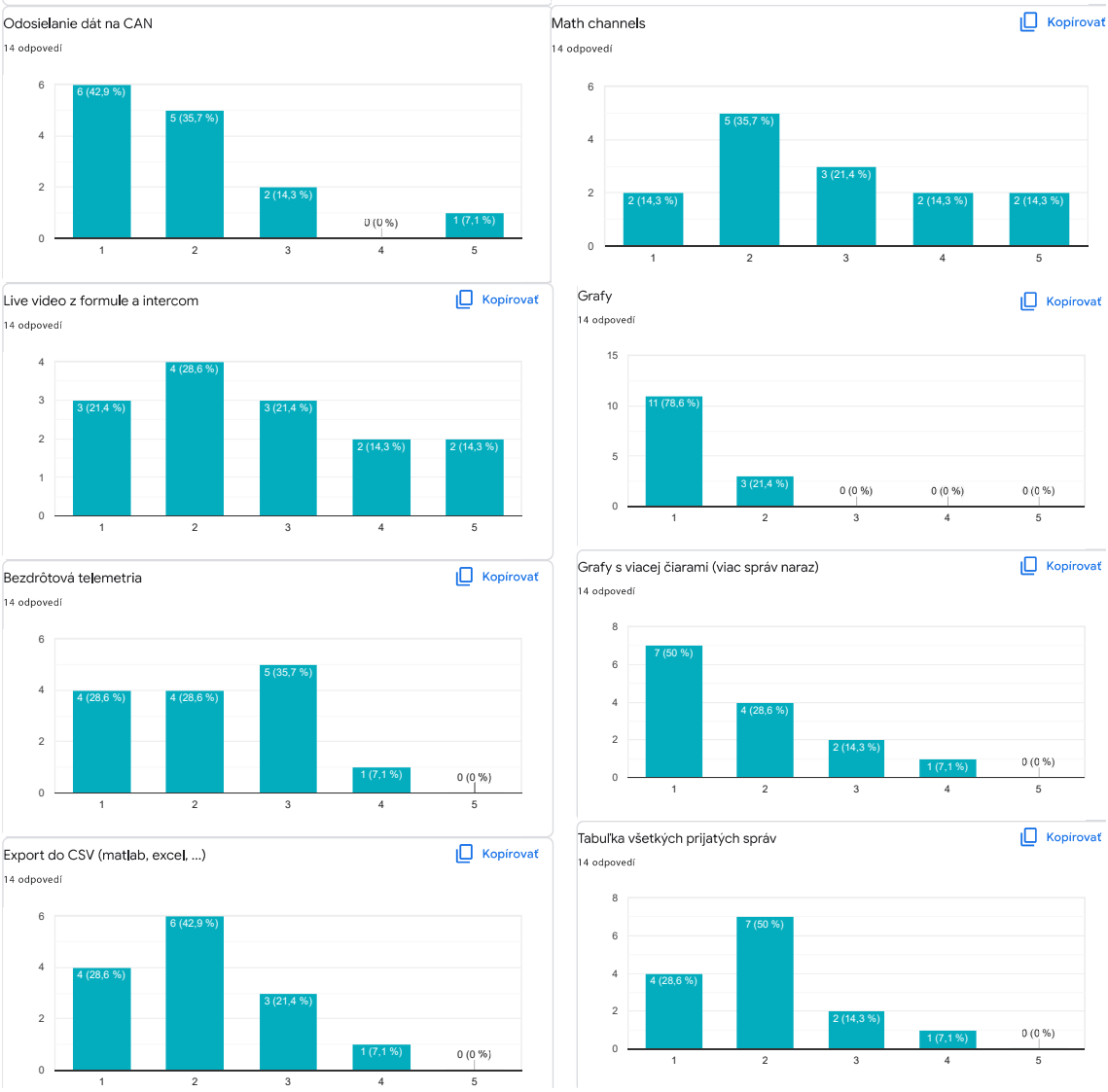
Príloha *SocketCAN - The official CAN API of the Linux kernel* sa nachádza v koreňovom adresári s názvom `SocketCAN-Official-CAN-API.pdf`

# Príloha C

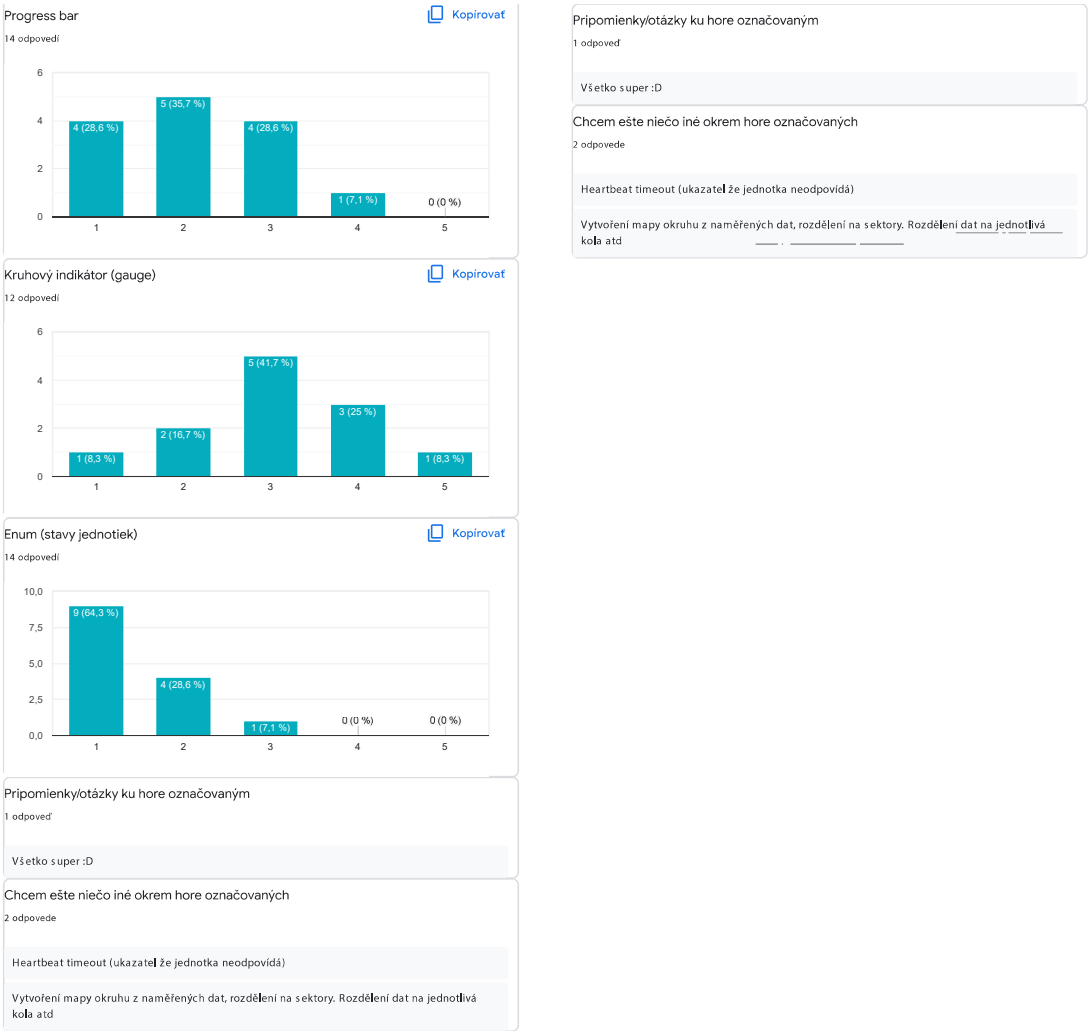
# Dotazník požiadavok na funkcionalitu telemetrie




Obr. C.1: Strana 1 dotazníku




Obr. C.2: Strana 2 dotazníku




Obr. C.3: Strana 3 dotazníku

# Príloha D

# Smerovanie a navigácia v rámci grafického používateľského rozhrania

Architektúra samotnej aplikácie spočíva v hierarchickom usporiadaní ciest (routes) a ich rozdelení do jednotlivých okien a navigácie. Túto funkcionalitu nám poskytuje knižnica react-router-dom[1]. Vďaka nej vieme vytvoriť odkaz, ktorý nás po kliknutí alebo presmerovaní dostane na požadovanú obrazovku  na oko jednoduchá úloha, no pri našom použití electron frameworku sa router nemôže správať ako jednoduchý smerovač v prehliadači ktorý mení cesty, ale je potrebné rozpoznávať prostredie a podľa toho smerovať buď podľa ciest URL, alebo podľa ciest v súborovom systéme OS. Každá cesta reprezentuje jednu obrazovku, ktorá obsahuje svoje komponenty. Tieto komponenty sú zaobalené v nami vytvorenom komponente GridWrapper, ktorý bol automaticky aplikovaný na všetky routes, a zaručuje nám kontext a funkcionalitu pre vytváranie vlastného rozloženia komponentov. Jednotlivé komponenty sú zaobalené v okne, ktoré sa dá presúvať a meniť veľkosť, a tým si používateľ vie vytvoriť vlastné rozloženie komponentov na svojej obrazovke, ktoré za pomoci localstorage, ktoré nám poskytuje prostredie prehliadača a nami vytvorenému hook rozhraniu vie uložiť do pamäte, a mať toto rozloženie perzistentné na zariadení.

[1] https://reactrouter.com

# Príloha E

# Bližší popis technologických častí

## E.1 Elektrický systém

Formula disponuje dvoma okruhmi napätia, ktoré sú galvanicky oddelené, a je neustále kontrolovaná ich izolovanosť, aby bolo zamedzené riziko úrazu zásahom elektrického prúdu.

Vysoké napätie, čo je napätie trakčného systému, ktoré poháňa motory, a teda celé vozidlo. Tento obvod má napätie medzi 300 a 400V jednosmerného napätia a je životu nebezpečné pristupovať k autu počas nabitia trakčného systému (jazdy). To že je auto nabité vysokým napätím je znázornené červeným svetlom nad hlavou vodiča. Prácu na takomto vozidle je podľa pravidiel povolené len certifikovanou osobou s certifikátom DGUV 209-093 stage 2E alebo vyšší zaškolené externou osobou (FSG pravidlá A4.3.8). My sme si takýmto školením prešli, a to znamená, že vieme vozidlo uviesť do certifikovaného beznapäťového stavu, a tým vieme umožniť prácu na vozidle aj ostatným necertifikovaným kolegom.

Druhý okruh pozostáva z nízkeho napätia na úrovni 24V, ktorým sú napájané mikrokontroléry a všetky ostatné prvky elektroniky. Jedným z takých je tzv. Shutdown Circut (SC) ktorý zabezpečuje bezpečnosť fungovania vozidla, a je to jeden obvod ktorý prechádza celým vozidlom, či už bezpečnostnými prvkami ako zotrvačný prepínač (ak zrýchlenie presiahne určitú hodnotu, rozpojí daný obvod), tak aj jednotlivými doskami, pri ktorých ak sa dostanú do chybového stavu, SC sa rozpojí, a vozidlo sa kompletne vypne. SC teda slúži ako hlavný elektrický bezpečnostný prvok vozidla. Tento spôsob implementácie je analogický ku aktuálne používaným bezpečnostným prvkom v sériových elektrický vozidlách s názvom pilotná línia (pilot line / interlock).

## E.2 Serializácia a práca s binárnymi dátami naprieč node a javascript

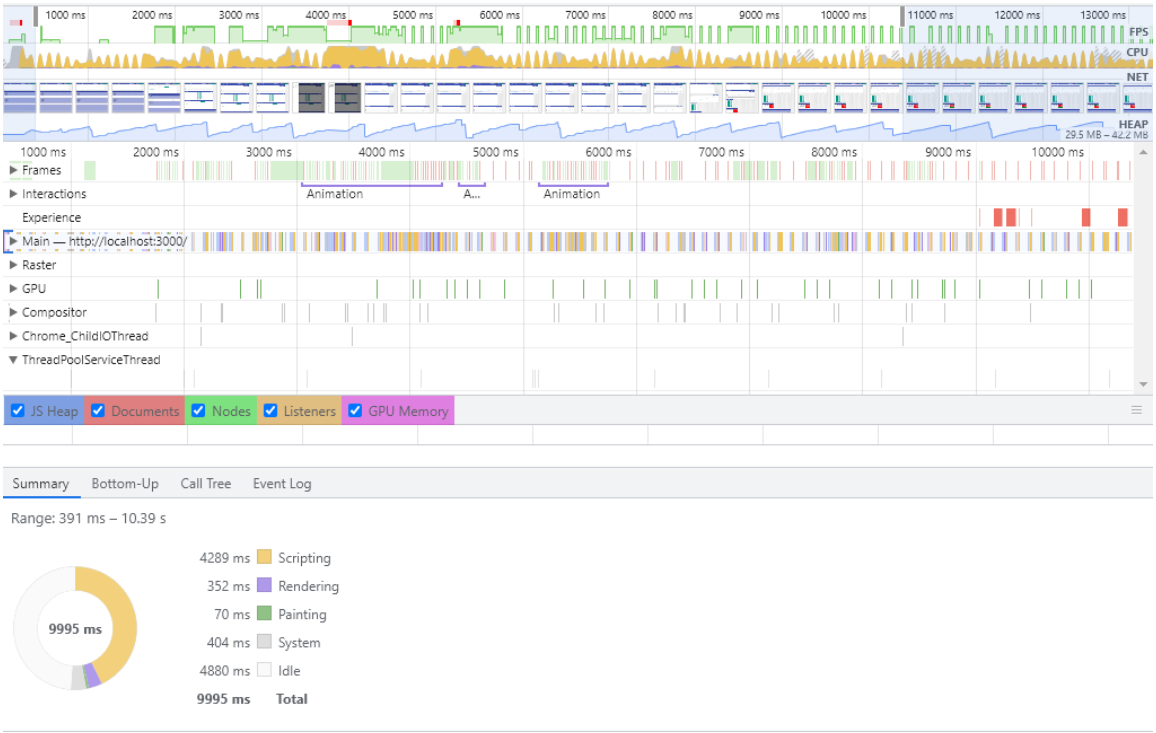
Dáta zo zbernice CAN majú veľkosť najviac 8 bajtov, a prichádzajú v Node bufferi. To problémy samé o sebe nerobí, no problém nastane, ak chceme posielať tieto dáta do renderer procesu. Ten zas s Node bufferom pracovať nevie, a je potrebné použiť ArrayBuffer [1]. Ten sa ale cez IPC serializovať nedá, a teda celý problém sme vyriešili použitím novšieho dátového typu v ES2020 s menom *BigInt*. Tento dátový typ má veľkosť 64 bitov, a teda nám presne vyhovuje. Ak by sme ale potrebovali viac dát v jednej správe, riešenie by muselo byť iné.

[1] https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer

# Príloha F

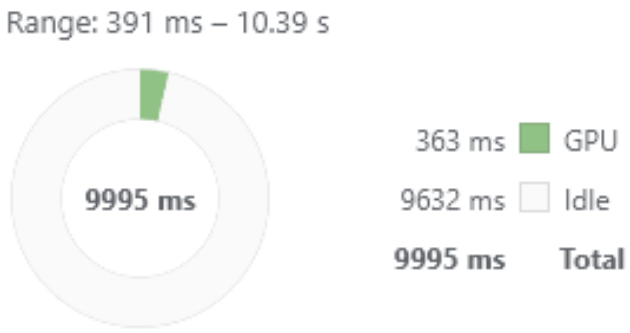
# Profilovanie aplikácie (výkonnostné testy)

Profilovanie prebiehalo pri pripojení telemetrie ku zbernici CRITIAL uloženej vo vozidle eD1 so všetkými zapnutými systémami.




Obr. F.1: Profilovanie pripojenia na zbernicu CRITICAL

Ako môžeme vidieť na obrázku F.1, čas profilovania je 10 sekúnd, z čoho okolo 4,5 sekundy grafické rozhranie vykonáva záťažovú aktivitu. Taktiež je patrné, že 5 sekúnd je v stave idle, z čoho vyplýva, že aplikácia je vyťažená na menej ako 50%. Na obrázku F.2 môžete vidieť analýzu vyťaženia GPU, pričom grafická karta je 96% času v stave idle, a máme ešte možnosť pridať omnoho viac grafov.




Obr. F.2: Výsledy profilovania grafickej karty