Univerzita Pardubice
Dopravní fakulta Jana Pernera

Systém pro vzdálenou diagnostiku vozidla

Matouš Danielka

Diplomová práce
2018

Univerzita Pardubice
Dopravní fakulta Jana Pernera
Akademický rok: **2017/2018**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Matouš Danielka**
Osobní číslo:	**D16385**
Studijní program:	**N3708 Dopravní inženýrství a spoje**
Studijní obor:	**Elektrotechnické a elektronické systémy v dopravě**
Název tématu:	**Systém pro vzdálenou diagnostiku vozidla**
Zadávající katedra:	**Katedra elektrotechniky, elektroniky a zabezpečovací techniky v dopravě**

Zásady pro vypracování:

Cílem této diplomové práce je vytvoření prototypu systému pro diagnostiku vozidla na dálku. Navrhněte prototyp systému pro přenos diagnostických informací z řídícího systému pohonu kolejového vozidla do diagnostické aplikace na PC. Data z vozidla přenášejte bezdrátově. Navrhněte potřebný hardware a vytvořte potřebný software pro mobilní (vozidlovou) část a vytvořte jednoduchou diagnostickou aplikaci pro PC, pomocí které bude možné diagnostická data z vozidla prohlížet v reálném čase i zpětně ve formě historie událostí.
Doporučený postup:
1. Seznamte se s funkcemi již existujících komerčních systémů vzdálené diagnostiky.
2. Seznamte se formátem a způsobem získání diagnostických dat z konkrétního kolejového vozidla.
3. Navrhněte architekturu vlastního systému vzdálené diagnostiky.
4. Vytvořte HW a SW mobilní (vozidlové) části.
5. Vytvořte diagnostickou aplikaci pro zobrazování diagnostických dat.
6. Ověřte funkci celého systému.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

[1] HEROUT, Pavel. Učebnice jazyka C. 4., přeprac. vyd. České Budějovice: Kopp, 2004, 271, viii s. ISBN 8072322206.
[2] VOSS, Wilfried. A comprehensible guide to controller area network. Greenfield: Cooperhill Technologies Corporation, 2005, 150 s. ISBN 0976511606.
[3] VOSS, Wilfried. A comprehensible guide to J1939. Greenfield, Mass: Copperhill Technologies Corporation, 2008. ISBN 09-765-1163-0.
[4] WILLIAMS, Elliot. AVR programming: Learning to Write Software for Hardware.
[5] FMS standard [online]. [cit. 2017-10-21]. Dostupné z: http://www.fms-standard.com/
[6] Arduino [online]. [cit. 2017-10-21]. Dostupné z: https://www.arduino.cc/
[7] Podklady poskytnuté vedoucím práce.

Vedoucí diplomové práce: **Ing. Zdeněk Mašek, Ph.D.**

Katedra elektrotechniky, elektroniky a zabezpečovací techniky v dopravě

Datum zadání diplomové práce: **15. listopadu 2017**

Termín odevzdání diplomové práce: **18. května 2018**

doc. Ing. Libor Švadlenka, Ph.D.
děkan

L.S.

Ing. Dušan Čermák, Ph.D.
vedoucí katedry

V Pardubicích dne 12. března 2018

PROHLÁŠENÍ

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 15. 5. 2018 Matouš Danielka

PODĚKOVÁNÍ

Děkuji především vedoucímu mé práce, panu Ing. Zdeňku Maškovi, Ph.D. za odborné rady při návrhu obvodu, předání osobních zkušeností pro širší poznání dané problematiky a vstřícný přístup. V neposlední řadě také za zprostředkování možnosti zkušebního nasazení vytvořeného systému na reálné vozidlo.

ANOTACE

Diplomová práce se zabývá řešením přenosu diagnostických informací z vozidla na vzdálený server, zpracováním těchto dat a zobrazením v různých formátech z webového rozhraní pro potřeby diagnostikování stavu vozidel na dálku. Vzdálený přístup zároveň řeší dodatečné ovlivnění zasílaných dat na základě předem definovaných možností.

KLÍČOVÁ SLOVA

CAN, sběrnice, Arduino, GSM, GPS, GPRS, IoT, C, C++, PHP, Nette

TITLE

A system for remote vehicle diagnostics

ANNOTATION

Diploma thesis deals with solutions of transmission diagnostics information from vehicle to server, information processing and data displaying to web client for remote diagnostics of vehicle condition. The remote access also provides sending commands to vehicle for request sending extra data to the server.

KEYWORDS

CAN-bus, Arduino, GSM, GPS, GPRS, IoT, C, C++, PHP, Nette

OBSAH

1 ÚVOD

Tvorba webových stránek (aplikací) patří již několik let mezi oblast mých zájmů. Zároveň jsem již delší dobu měl nutkání sestavit technické řešení, které by využívalo mikrokontrolér, přičemž bych si ho sám k danému účelu naprogramoval. Ideální příležitostí k přetvoření touhy v realizaci se mi naskytla možná náplň diplomové práce. Pomocnou ruku ve výběrem tématu mi v tomto nabídl vedoucí mé práce Ing. Zdeněk Mašek, Ph.D. V rámci výrobního portfolia motorových univerzálních vozíků CZ LOKO a spoluprací s Dopravní fakultou Jana Pernera se podílel na vývoji řídícího systému vozidel MUV 75 a dále spolupracuje na navazujících projektech. Řídící systém vozidel MUV 75 využívá sběrnici CAN s protokolem J1939 zprostředkovávající diagnostické informace, které jsou k dispozici strojvedoucímu na diagnostickém displeji. Součástí přenášených informací jsou některé standardizované zprávy a zprávy uživatelsky definované, s diagnostickými informacemi hydrostatického pohonu vozidla přenášené protokolem J1939, který je nejrozšířenější v oblasti silničních nákladních vozidel s dieselovými spalovacími motory.

V souvislosti se současnými možnostmi a dnes již stále běžnějšími požadavky na dohled nad vozidly se jako téma práce nabízela realizace přenosu diagnostických dat z vozidla na server, jejich zpracování a zobrazení koncovému uživateli, který na jejich základě může přesněji předjímat plán údržby jednotlivých vozidel po výskytu závad. Téma mě velice zaujalo, a tak jsem na něm začal pracovat.

2 VLASTNOSTI SYSTÉMU VZDÁLENÉ DIAGNOSTIKY

2.1 Požadavky kladené na systém

Diagnostické informace by měly být dostupné z webových stránek po přihlášení k účtu zabezpečeným heslem. Stránky budou svým uživatelům poskytovat informace ze všech určených zpráv dostupných na CAN lince vozidel, od všech vozidel zapojených do vzdálené diagnostiky. Webová aplikace bude zobrazovat informace o aktuálním dění, ale zároveň také historické záznamy uchovávané po dobu jednoho měsíce.

Vzdálená diagnostika nebude sloužit pro sledování rychlých dějů spojených především s pohonem. Informace v rámci těchto dějů budou rovněž přenášeny na server pro potřeby vzdáleného dohledu, avšak si nekladou za cíl poskytovat informace v řádu milisekund, které jsou jinak potřebné pro naladění pohonu ve fázi vývoje vozidla. Data poslouží pro odhalení závad, které zle sledovat i s delší časovou periodou. Uvažováno je souhrnné zasílání dat na server jednotlivých zpráv linky CAN. Vždy poslední zaznamenané informace z daných zpráv na lince CAN budou hromadně odesílány na server se sekundovou periodou. Během této doby dojde vždy na CAN lince k aktualizaci všech zpráv, které jsou pravidelně zasílány. Zprávy jsou CAN lince zasílány s frekvencí 10 ms, 50 ms, 250 ms a 1 sekunda.

Pro zobrazení uživateli by měla být k dispozici data v podobě, která nebude pouze surová, tak jak data přichází z CAN linky. Jednotlivým informací bude nadefinován popis a hodnotám převod na fyzikální jednotku, aby byla data jednoduše vizuálně čitelná. V úvahu připadá určitá podoba exportu dat do souboru, pokud by bylo žádoucí s data dále pracovat například v prostředí programu Excel, Matlab apod. Data bude možné vykreslovat do grafů.

Spolu s diagnostickými daty by mělo vozidlo rovněž oznamovat svou geografickou polohu, která bude přenášena spolu s dalšími informacemi a aktuální poloha vozidel bude vykreslována do mapy.

Systémové požadavky

1. Měření polohy a rychlosti vozidla pomocí GPS.
2. Jednorázové vyčtení měřených hodnot a závad z řídícího systému trakčního pohonu vozidla.
3. Periodické vyčítání měřených hodnot a závad z řídícího systému trakčního pohonu vozidla. Perioda čtení v řádu jednotek až desítek sekund.

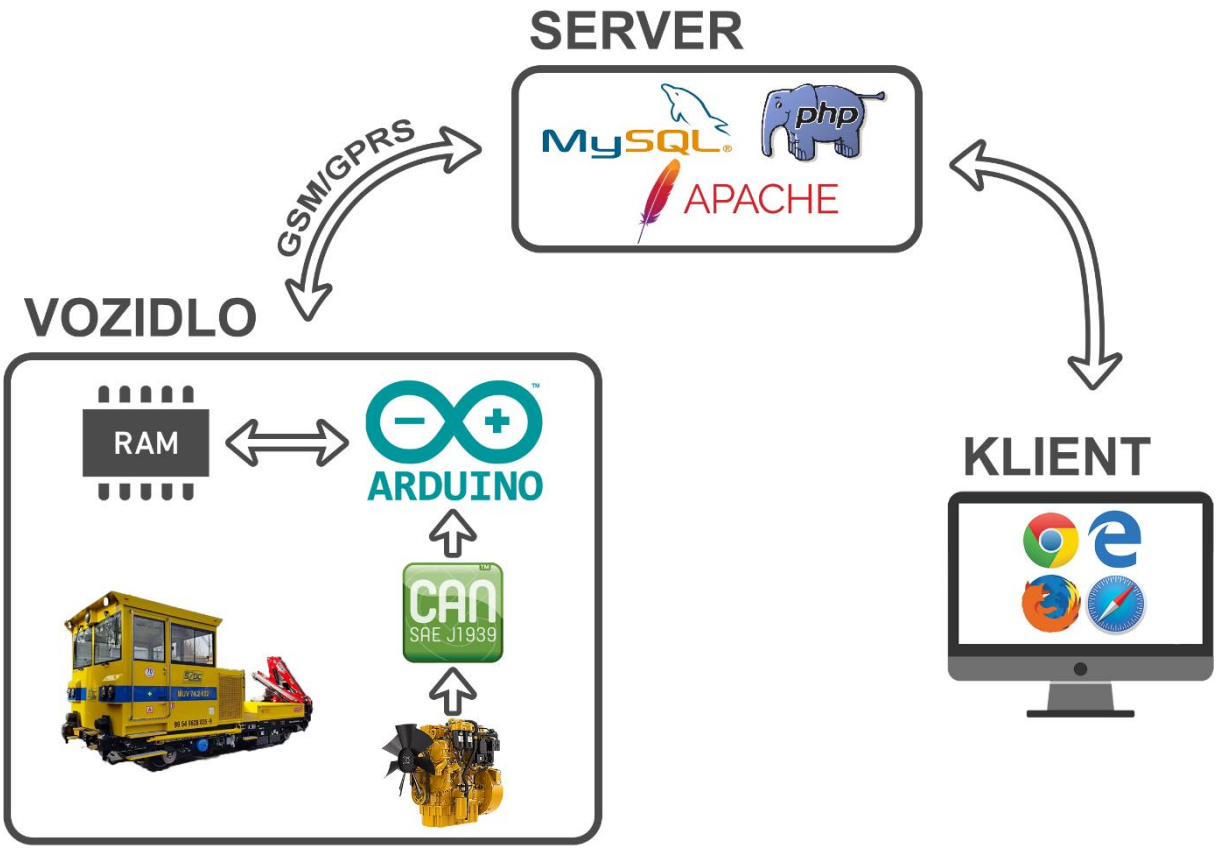
4. Záznam měřených hodnot spouštěný výskytem závady v řídícím systému trakčního pohonu. Požadovaná délka záznamu před a po výskytu chyby desítky sekund. Perioda ukládání dat max. 1 s.

Aplikace

1. Zobrazování měřených hodnot a závad z vybraného vozidla v tabulkách a v grafu.
2. Zobrazování polohy a rychlosti vybraného vozidla v mapě.
3. Správa vozidel (přidání, odebrání vozidla do aplikace, přehled stavu vozidel).

2.2 Funkce vytvořeného systému

Architektura navrženého systému by se dala pro shrnutí popsat následujícím přehledovým schématem. Architekturu reprezentují tři základní bloky, které spojuje internetová síť. Data putují obousměrně mezi jednotlivými bloky, ale komunikaci zahajuje pouze vozidlo, nebo klient a probíhá způsobem: dotaz na server – odpověď serveru.



Obrázek 1 Architektura systému

Systém umožňuje

1. Sledování dat odkudkoliv, pouze s připojením na internet. Třeba pomocí „chytrého“ telefonu.
2. Přístup pouze skrz zabezpečené přihlášení k uživatelskému účtu.
3. Zapojit do systému libovolné množství vozidel.
4. Sledování polohy vozidel a rychlosti jízdy z GPS on-line.
5. Odesílání dat ze sběrnice CAN na server.
6. Reagovat na výskyt závady v definovaných zprávách CAN s příznaky závad odesíláním zaznamenaných dat na server v časovém intervalu kolem výskytu závady.
7. Vzdáleně, ze strany klienta (webový prohlížeč), vyžádat odeslání dat na server nad rámec běžného objemu při provozu bez detekovaných závad s možností zapnutí trvalého odesílání většího množství dat se zprávami CAN.
8. Kontrolu timeoutu CAN zpráv. Nedojde-li další zpráva s daným identifikátorem do stanoveného času, je toto ve zprávě přenášené na server indikováno.
9. Zpracování dat včetně převodu na fyzikální veličiny.
10. Zobrazení veličin v tabulce.
11. Exportování dat do formátu CSV.
12. Zobrazení časového vývoje veličin v grafu.
13. Porovnávání různých veličin v jednom grafu.
14. Vedení záznamů o výskytu závad a provedených opravách, případně o dalších informacích ke každému z vozidel.
15. Automatické mazaní starých dat na serveru po uplynutí stanoveného času od jejich přijetí.

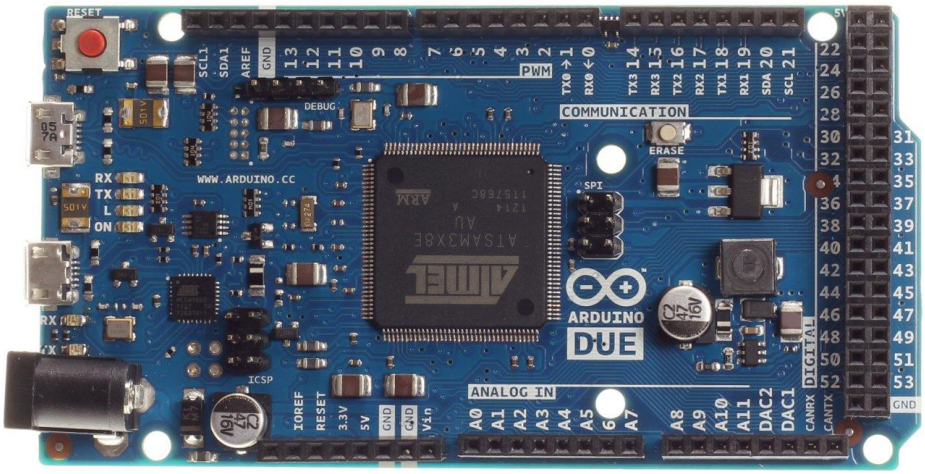
Při zasílání zpráv z vozidla na server se vozidlo identifikuje svým ICCID (Integrated Circuit Card Identifier). ICCID je unikátní identifikátor SIM karty, který karta získá při svém vyrobení. Nové vozidlo tak lze do systému jednoduše přidat. Do mobilní části se vloží SIM karta bez kódu PIN a do webové aplikace stačí z obalu k SIM kartě opsat kód ICCID a zapsat název vozidla.

3 HARDWAROVÉ ŘEŠENÍ VOZIDLOVÉ ČÁSTI

Rozsah požadavků na funkcionalitu z časových důvodů neumožňuje návrh hardwaru od základu z diskrétních součástek. Podobná situace panuje také v oblasti softwarového řešení. Časovou úsporu přináší použití hotových obvodů zpravidla s jednou funkcí v kombinaci s využitím souvisejících softwarových knihoven. Dílčí části pak vytváří výsledný celek a jejich propojení, hardwarové i softwarové, je hlavní oblastí realizace. Z tohoto důvodu byla pro mobilní část zvolena platforma Arduino se spoustou rozšiřujících modulů a hotových softwarových knihoven.

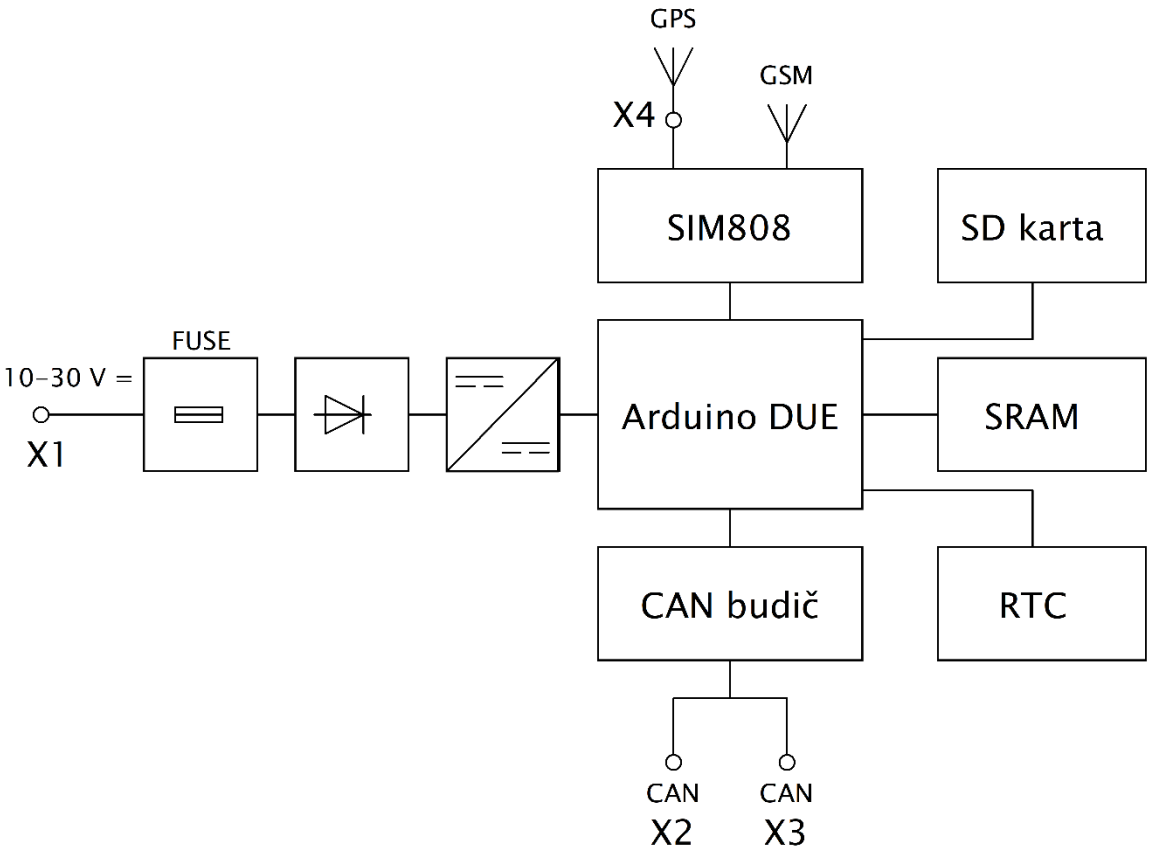
3.1 Použité obvody

Nosným prvkem vozidlového řešení byla zvolena vývojová deska Arduino DUE s 32 bitovým mikrokontrolerem AT91SAM3X8E architektury ARM. Starší desky Arduino jsou pouze 8 bitové, architektury AVR. DUE v porovnání s nimi využívá nižší pracovní napětí 3,3/5 V, vyšší taktovací frekvenci 84/16 MHz, nabízí více paměti. Deska je nejvíce podobná s dříve vydaným typem Arduino MEGA 2560. V porovnání s touto verzí však nově nabízí například dvojici sběrnic CAN (Arduino DUE má integrovaný CAN řadič). Jejich piny jsou však jen propojeny s mikrokontrolerem a nejsou tak vybaveny budiči nezbytnými pro správnou funkci.



Obrázek 2 Arduino DUE, zdroj: www.cnx-software.com

Hardwarovému řešení celé mobilní části odpovídá následující blokové schéma.



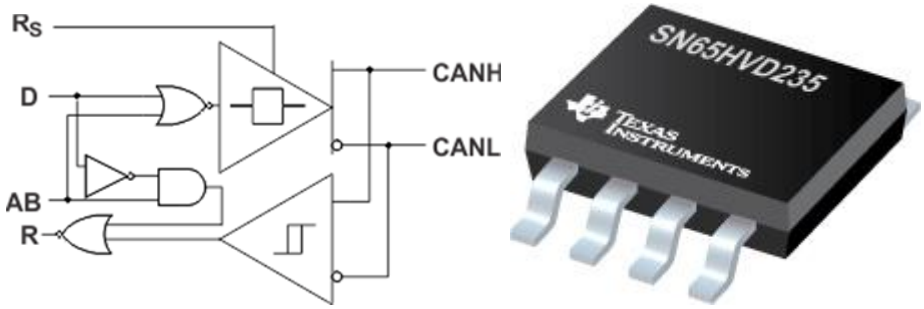
Obrázek 3 Blokové schéma vozidlové části

Koncepce dvou konektorů jedné sběrnice CAN umožňuje jednodušší připojení zařízení do stávající sítě CAN na vozidle (dovoluje prosmyčkování). Sběrnice CAN se rozpojí a do série se vloží tato část bez vlivu na ostatní zařízení po zapojení.

3.1.1 Budič CAN

Pro řešení na vozidle plně postačuje jedna sběrnice CAN doplněná o budič CAN SN65HVD235 od společnosti Texas Instruments s napájecím napětím 3,3 V a parametry vhodnými pro použití na 24 V palubní síti vozidla. Touto kombinací bude tedy po hardwarové stránce vyřešena komunikace se sběrnicí CAN. Budič je připojen k sběrnici CAN piny CANH, CANL. Dále do Arduina pinem R na pin CANRX a pinem D na pin CANTX Arduina. Pin AB je připojen na jeden z digitálních vstupně/výstupních portů Arduina. Ten může posloužit pro detekci přenosové rychlosti na sběrnici CAN. Pin R_S budiče slouží pro hardwarovou volbu komunikace, kde je na výběr mezi vysokorychlostním přenosem, nízko příkonovou komunikací a přenosem bez vlivu budiče na rychlost komunikace. Byla zvolena poslední možnost

s připojením pinu R_S přes pulldown rezistor 10 kΩ na GND. Poslední dvojicí pinů jsou piny napájecí, které jsou připojeny na 3,3V a GND.



Obrázek 4 Budič SN65HVD235, zdroj: Texas Instruments

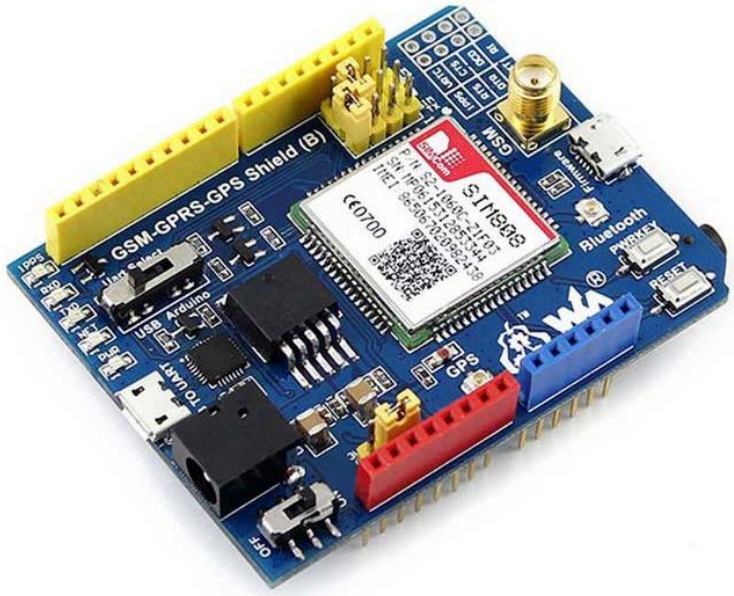
3.1.2 Modul SIM808

Dalším nezbytným prvkem mobilního zařízení je připojení k mobilní síti, která zajistí nezbytný přenos dat na server. V rámci dostatečného pokrytí území České republiky mobilními sítěmi čtvrté generace, které je zároveň územím s akčním rádiem nových motorových vozíků z produkce CZ LOKO, by se nabízelo využít již této rychlé mobilní sítě. Bohužel moduly s touto technologií v porovnání s GSM/GPRS moduly nejsou příliš dostupné a zároveň neúměrně nákladné. Pro potřeby přenosu diagnostických dat však není rychlé připojení nezbytné.

Na trhu je dostupný shield od společnosti Waveshare, svými rozměry a rozložením pinů navržený přímo pro Arduino. Využívá integrovaný čip SIM808, který kromě GSM/GPRS přináší výhodu v podobě připojení k GPS. GPS tak není nutné řešit zvlášť, pouze využívá další anténu, která je součástí balení od Waveshare. Obvod SIM808 navíc disponuje technologií Bluetooth, která ale pro nasazení na vozidle nemá uplatnění. Modul SIM808 bude zároveň v mobilním zařízení dílčím obvodem s největším příkonem. V rámci GSM komunikace má zařízení v krátkých špičkách příkon až 2 A při napájecím napětí kolem 9 V, které je nutné zohlednit při výběru napájecího zdroje.

Shield SIM808 od Waveshare je kompatibilní s mikrokontrolery ARM i AVR. Pro změnu napěťové úrovně 3,3/5 V slouží propojka na horní straně modulu. Se zvolenou napěťovou úrovní se pak shield propojuje s napájecími piny Arduina a samozřejmě se pak také komunikace odehrává ve zvolené napěťové úrovni. Shield má možnost vlastního napájení se stabilizátorem napětí. Lze tak připojit stejnosměrné napájení 9 V stejně úspěšně do Arduina, jako do shieldu SIM808, aby zařízení fungovalo správně jako celek. Shield komunikuje po jedné sběrnici UART pomocí AT příkazů. K Arduinu je sběrnice připojená na piny RX a TX, které lze také

dohledat pod názvem HardwareSerial jako opaku k SoftwareSerialu na vstupně/výstupních portech. Posledním pinem nutným pro komunikaci je v minimalistické dokumentaci k shieldu trochu utajený pin s názvem PWRKEY, který zajistí oživnutí obvodu SIM808 se shodnou funkcí, jakou disponuje stejnojmenné hardwarové tlačítko umístěné na vrchní straně modulu. Pin PWRKEY je připojen k Arduinu na vstupně/výstupní pin, který je po zapnutí nastaven na jednu sekundu nastaven do úrovně HIGH a poté se opět vrací do úrovně LOW, čímž se modul zapne. Nakonec je také nutné k shieldu připojit GSM anténu a již zmíněnou anténu GPS. Obě jsou součástí balení.



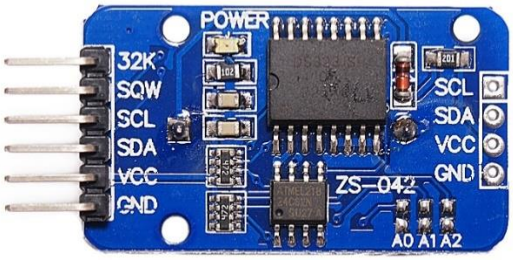
Obrázek 5 Shield SIM808 – GSM, GPS, zdroj: www.dx.com

3.1.3 Obvod reálného času

Aktuální čas je možné získat z GSM i GPS dat. Nezajistí však inkrementaci času a neustálé dotazování na aktuální čas by výrazně zpomalovalo chod programu a v krátkodobé nedostupnosti sítě bychom aktuální čas neměli k dispozici. Odvození aktuálního času podle taktovací frekvence procesoru Arduina by také nebylo příliš šťastným řešením. Proto bude aktuální čas z modulu SIM808 vyžádán pouze při spouštění zařízení (aktivaci vozidla) a odeslán do modulu reálného času DS3231.

Modul DS3231 zajistí dostatečně přesný čas během celé doby provozu vozidla do jeho odstavení. Obvod je vybaven teplotně kompenzovaným krystalovým oscilátorem. Zároveň je možné, díky vlastnímu napájení z monočlánku CR2032, využívat uložený čas i po opětovném spuštění po odpojení napájení. V aktivním stavu je obvod napájen z Arduina. Komunikace s Arduinem probíhá za pomoci I2C sběrnice s piny SCL a SDA, kterou Arduino také disponuje.

Obvod DS3231 zároveň obsahuje další dva piny, které zůstanou nevyužity. Jde konkrétně o pin 32K, jako výstup oscilátoru 32 KHz a pin SQW jako výstup pro volitelné přerušení na straně Arduina, případně jakékoliv jiné využití s volitelným periodickým spínáním.



Obrázek 6 Obvod RTC, zdroj: www.continentalee.com.sg

3.1.4 Čtečka SD karet

Mobilní zařízení je vybaveno také čtečkou SD karet, která však není v aktuální verzi použita. V budoucnu by mohla sloužit jako prozatímní úložiště při aktualizaci FW v Arduinu na dálku (příjem firmwaru ze serveru, jeho uložení na SD kartu a následné přeprogramování paměti FLASH). Vybrán byl tedy modul se čtečkou SD karet, jehož funkce byla následně prakticky otestována ve spolupráci s Arduinem. Modul obsahuje 8 dvojic pinů. Každé dva piny jsou mezi sebou vodivě propojeny. Důvodem je zřejmě požadavek na lepší mechanické upevnění vzhledem k působení síly na čtečku při zasouvání a vyjímání karet.

Čtečka má zvlášť piny pro napájení 3,3 V a 5 V a vlastní napěťový stabilizátor. Čtečka využívá sběrnici SPI, kterou je rovněž Arduino vybaveno. Jedna sběrnice SPI umožnuje komunikaci s více zařízeními v rozdílných časových okamžicích. Sběrnice má společné piny MISO (Master In, Slave Out), MOSI (Master Out, Slave In) a SCK (hodinový signál od mastera) a samozřejmě napájení. Každé z připojených zařízení na sběrnici SPI má pak unikátní pin CS, který definuje, se kterým zařízením v daný čas master (Arduino) komunikuje.

Čtečka karet spolu s kartou SD nemůže zajišťovat funkci periodického ukládání dat ze sběrnice CAN, protože životnost karet by byla v takovémto provozním režimu velice krátká. Podle dohledatelných zkušeností by se v závislosti na četnosti provozu mohla životnost přiblížit bezproblémovému ročnímu provozu.

Obrázek 7 Čtečka SD karet, zdroj: www.lumisense.in

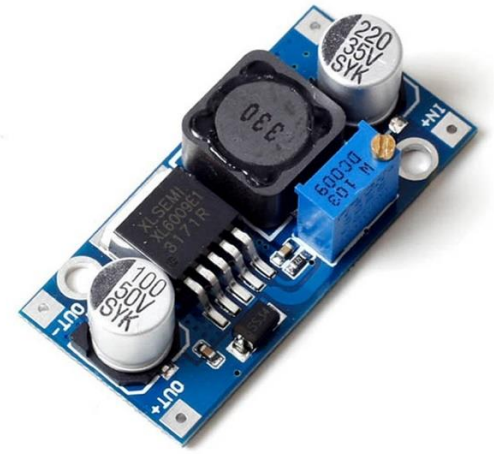
3.1.5 Externí paměť SRAM

Funkci kruhového bufferu nejlépe zajistí paměť RAM. Konkrétně byla vybrána paměť SRAM 23LC1024 společnosti Microchip s velikostí paměti 1024 kb (128 KB), která může vystačit až na deset minut záznamu při periodě ukládání všech požadovaných zpráv ze sběrnice CAN jednou za sekundu. V mezidobí budou data aktualizována ze sběrnice CAN. Spolu se zprávami ze sběrnice CAN bude v rámci jednoho vzorku rovněž ukládána časová značka, informace o poloze a rychlosti jízdy. Paměť SRAM rovněž komunikuje po sběrnici SPI.

Obrázek 8 Paměť SRAM 23LC1024, zdroj: www.microchip.com

3.1.6 Obvod zdroje

Na vozidle je k dispozici stejnosměrná napájecí síť 24 V. Pro Arduino a další obvodové součásti je zapotřebí stejnosměrné napájení 9 V s proudem až 2 A. Pro tento účel byl vybrán step-down DC-DC měnič založený na obvodu LM2956 od Texas Instruments. Jde o spínaný zdroj s frekvencí 150 kHz. Obvod je schopen dodávat výkon až 10 W. Maximální proud 3 A postačuje pro potřeby mobilní části. Součástí modulu je více otáčkový trimr, který umožňuje nastavit výstupní napětí v rozsahu 1,3 V až 30 V.



Obrázek 9 Step down DC-DC měnič s obvodem LM2956, zdroj: hackstore.co.il

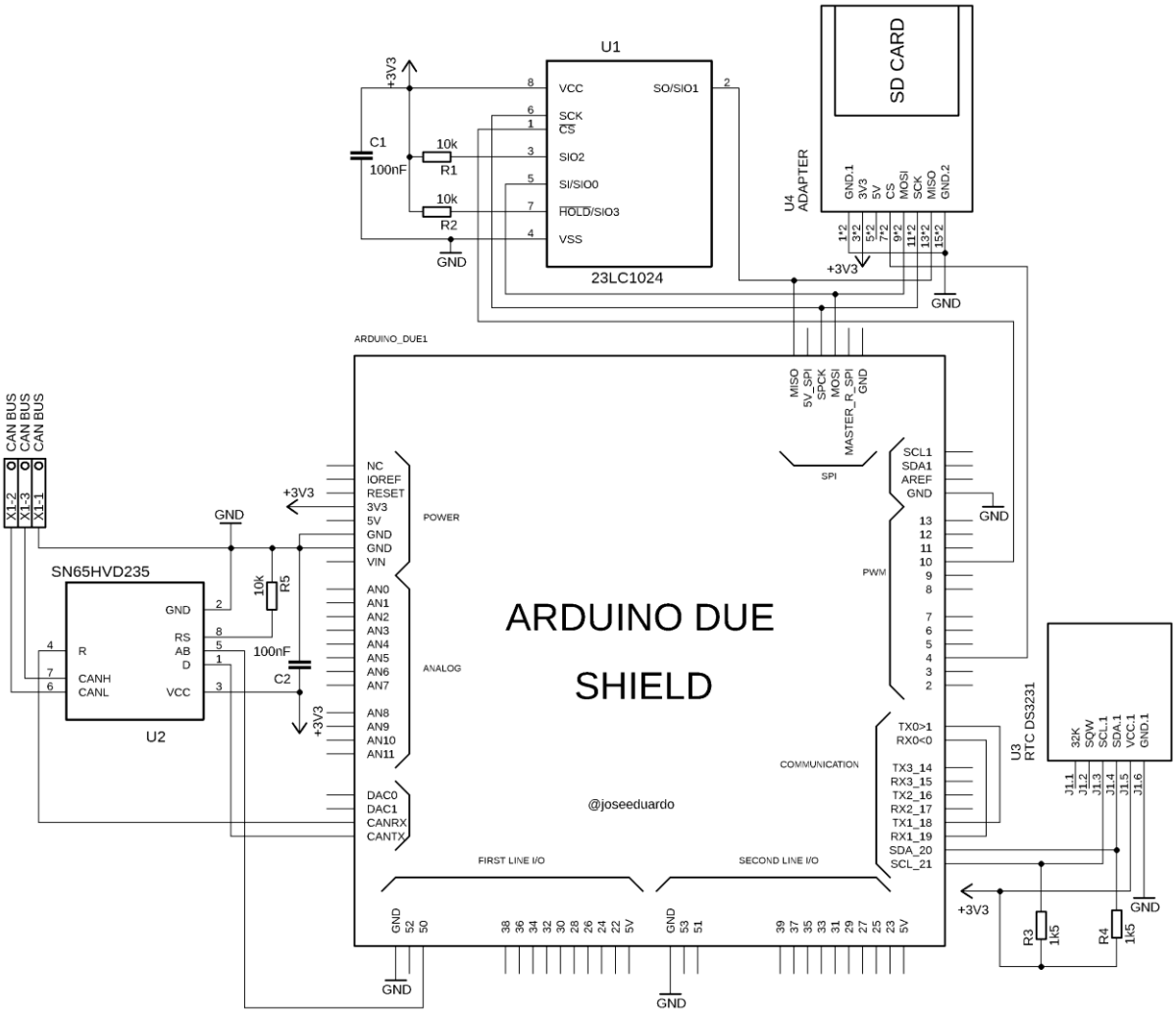
Zařízení jako celek pak umožní trvalý provoz v rozsahu napětí 10–30 V. Lze tak napájet i ze stejnosměrné sítě 12 V. V motorovém vozíku je zařízení napájeno po sepnutí hlavního stykače z vozidlové baterie se jmenovitým napětím 24 V.

3.2 Návrh DPS

Použití Arduina se shieldem SIM808 by samo o sobě nevyžadovalo výrobu další desky, avšak pro všechny ostatní dílčí obvody je to nutnost. Nakonec však ani jednoduché zasunutí pinů do desky Arduina v rámci dodržení koncepce Arduina není úplně ideální. Arduino DUE má totiž další specifikum týkající se sběrnic UART a komunikace s PC. Pro připojení do počítače jsou k dispozici dva microUSB konektory. Jeden s označením Programming portu, druhý Native port. Arduina s procesory AVR mají k dispozici pouze jednu sběrnici pro přístup k USB a všechny sběrnice oddělené. Arduino DUE má však Programming port na společné sběrnici se sběrnicí Serial0, což je první sběrnice UART Arduina, která má jednotnou pozici na desce na všech rozměrově obdobných deskách počínaje Arduinem UNO. Výrobce shieldu SIM808 se samozřejmě držel standardu a využil sběrnice Serial0. Ostatně neměl ani jinou možnost, protože rozměrově menší desky (UNO) ani jinou sběrnicí nedisponují, pokud opomeneme možnost softwarové realizace sběrnice na ostatních vstupně/výstupních portech, která má řadu omezení, a tak jde o faktickou „z nouze ctnost“. Arduino DUE má však 4 sběrnice UART, pokud započítáme i společnou pro Programming port. Pro potřeby mobilní část stačí právě jedna komunikující s modulem SIM808. Je tedy více než žádoucí převést komunikaci s modulem SIM808 na jinou sběrnici UART Arduina.

Uvažovány byly dvě koncepce řešení DPS. Jednou z možností bylo vytvoření jedné velké desky, na kterou by byly umístěny všechny dosud uvažované obvody včetně samotného Arduina. Druhou nabízející se volbou bylo vyjít z filozofie Arduina a její metody shieldů. To v tomto případě představuje vytvoření prostřední desky v této sendvičové koncepci za předpokladu, že se na ní podaří vměstnat všechny dílčí části, mimo napájecí desku. Ve spodní vrstvě by tak bylo Arduino, uprostřed uvažovaná deska a horní část by obsadil shield SIM808. Zároveň by i tato koncepce umožňovala převedení komunikace SIM808 na jinou sběrnici UART Arduina. Rozhodl jsem se pro druhou, rozměrově minimalističtější variantu.

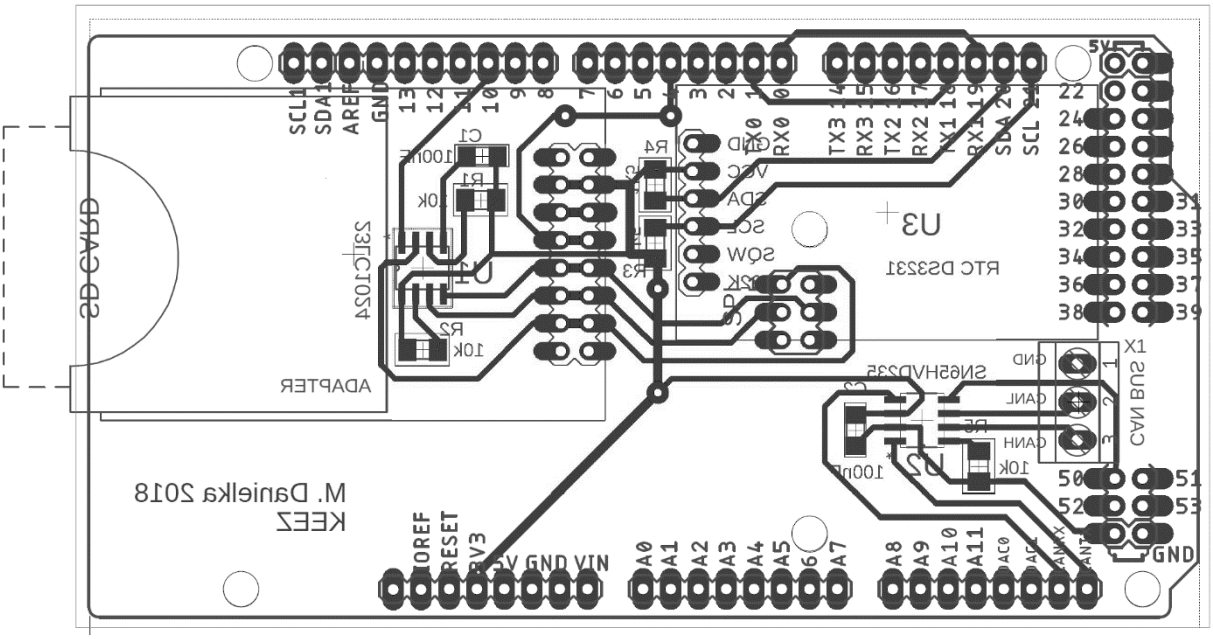
Navrhovaná deska byla z důvodu možnosti vlastnoruční výroby v prostorách katedry uvažována jako jednovrstvá. Obvod paměti SRAM 23LC1024, stejně jako obvod budiče CAN SN65HVD235 jsou v provedení SMD. Zapojení bylo dále nutné doplnit o několik pasivních součástek, které jsou jednotně voleny také v provedení SMD. Veškeré moduly s vlastními vývody a konektory k propojení shieldů s Arduinem, tak připadnou na druhou stranu desky. Návrh obvodu byl realizován v programu Eagle. Vzhledem k rozsáhlé komunitě kolem Arduina a programu Eagle se podařilo nákresy všech modulů včetně Arduina samotného dohledat on-line, a tak nebylo nutné znovu „vynalézat kolo“, avšak s ohledem na nevyužité piny, které navíc bránili v umístění některých prvků na vhodná místa, jsem přistoupil k některým úpravám využitých součástí.



Obrázek 10 Obvodové zapojení vytvářeného shieldu

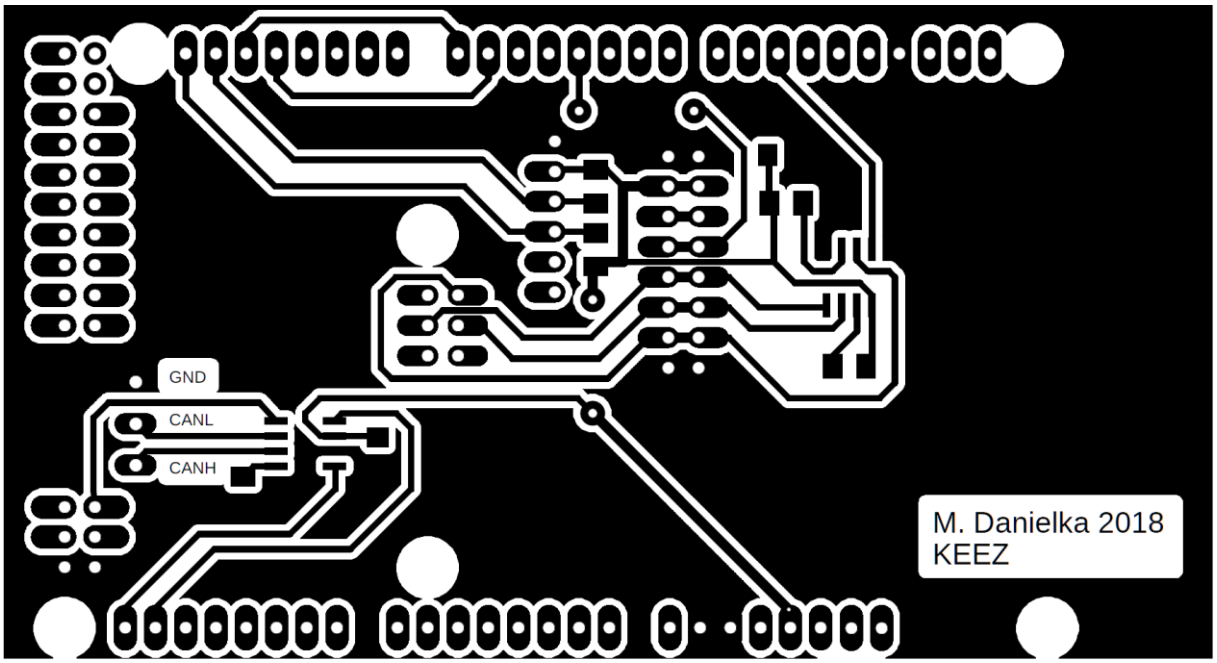
Tabulka 1 Tabulka součástek vytvořené desky vyjma konektorů

Položka	Hodnota / typ	Pouzdro
C1	100 nF	C1206
C2	100 nF	C1206
R1	10 kΩ	M1206
R2	10 kΩ	M1206
R3	1,5 kΩ	M1206
R4	1,5 kΩ	M1206
R5	10 kΩ	M1206
U1	23LC1024	SOIC127P600X175-8N
U2	SN65HVD235	SOIC127P600X175-8N
U3	TC DS3231	
U4	CARD SD ADAPTER	



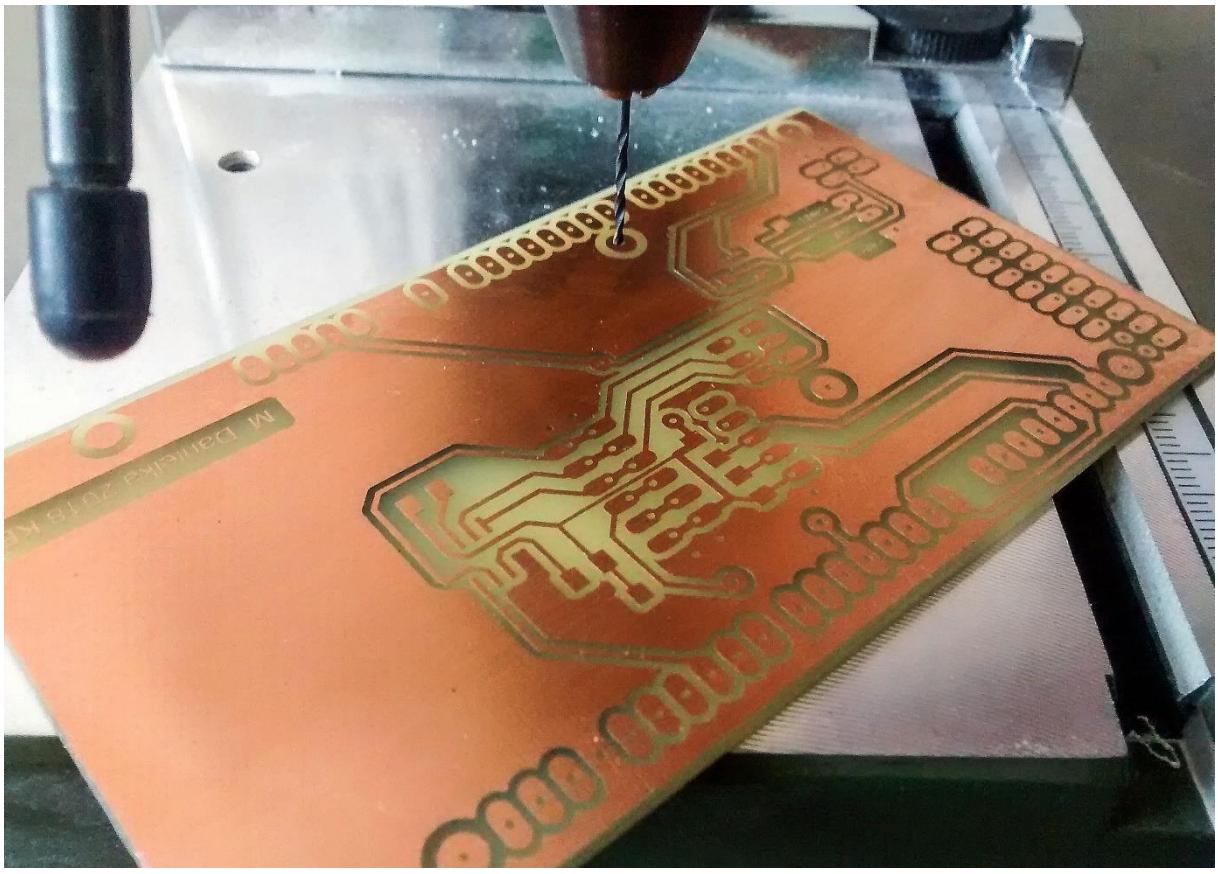
Obrázek 11 Schéma navržené DPS (pohled ze strany TOP)

Zem (GND) byla v Eagle vytvořena jako polygon vyplňující jinak nevyužitou část desky, aby nebylo nutné odleptat tolik mědi.

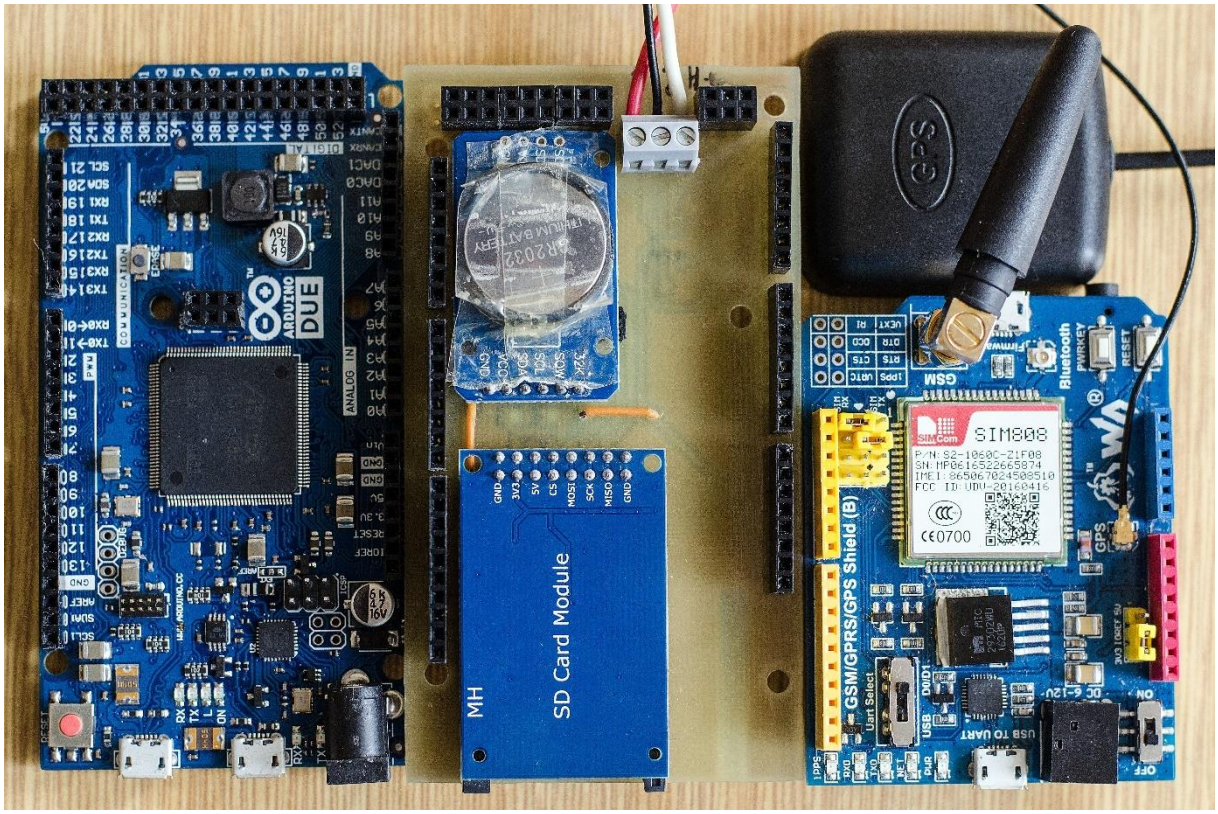


Obrázek 12 Vodivé cesty navržené DPS (pohled ze strany BOTTOM)

Výroba následně proběhla tzv. fotocestou. Osvitem fotocitlivé vrstvy UV světlem skrz masku, vývojkou v roztoku NaOH a samotné vyleptání proběhlo v roztoku kyseliny chlorovodíkové, peroxidu vodíku a vody.



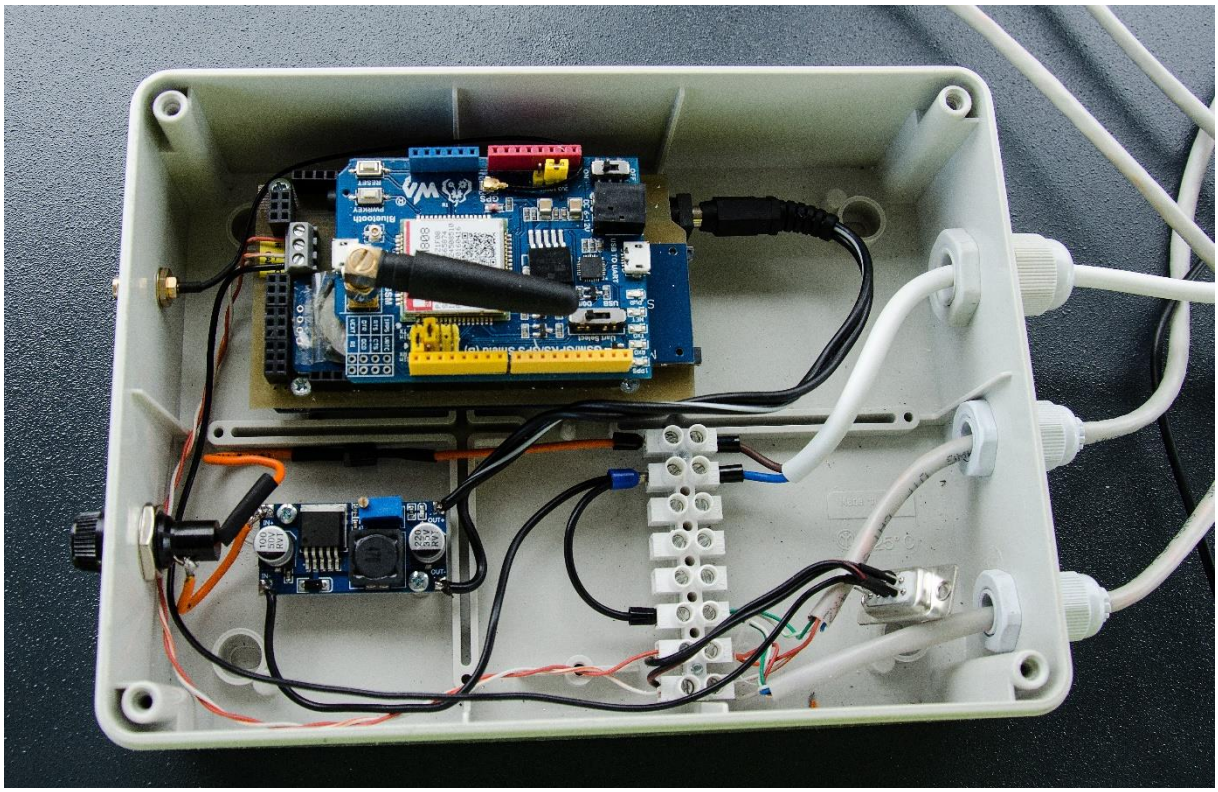
Obrázek 13 Výroba DPS



Obrázek 14 Mobilní část v rozloženém stavu

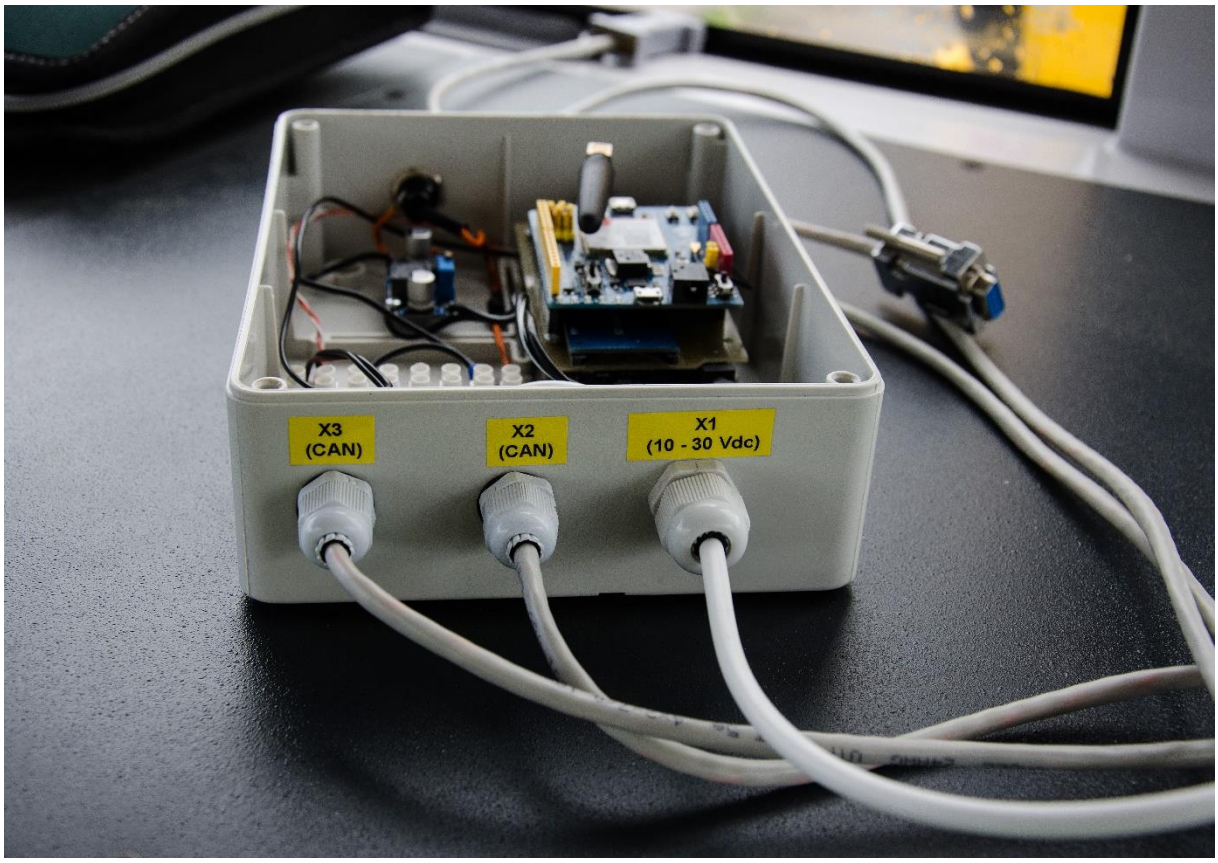
3.3 Montáž do elektroinstalační krabice

Další fází je montáž do elektroinstalační krabice. Byla vybrána elektroinstalační krabice s vnitřními rozměry 190x140x70 mm. Pro zvýšení mechanické odolnosti byly všechny vrstvy sendvičové konstrukce kromě konektorů navíc mechanicky propojeny distančními sloupky o výšce 10 mm. Ze spodu Arduina bylo zařízení skrz další řadu distančních sloupků upevněno do elektroinstalační krabice, jemuž předcházelo vyvrtání otvorů. Dále byla do krabice osazena lámací svorkovnice a odděleně také DC-DC měnič, který má na svém vstupu ochranu proti přepólovaní pomocí usměrňovací diody umístěné v + přívodu napájení a rychlou pojistku 1 A proti zkratu.

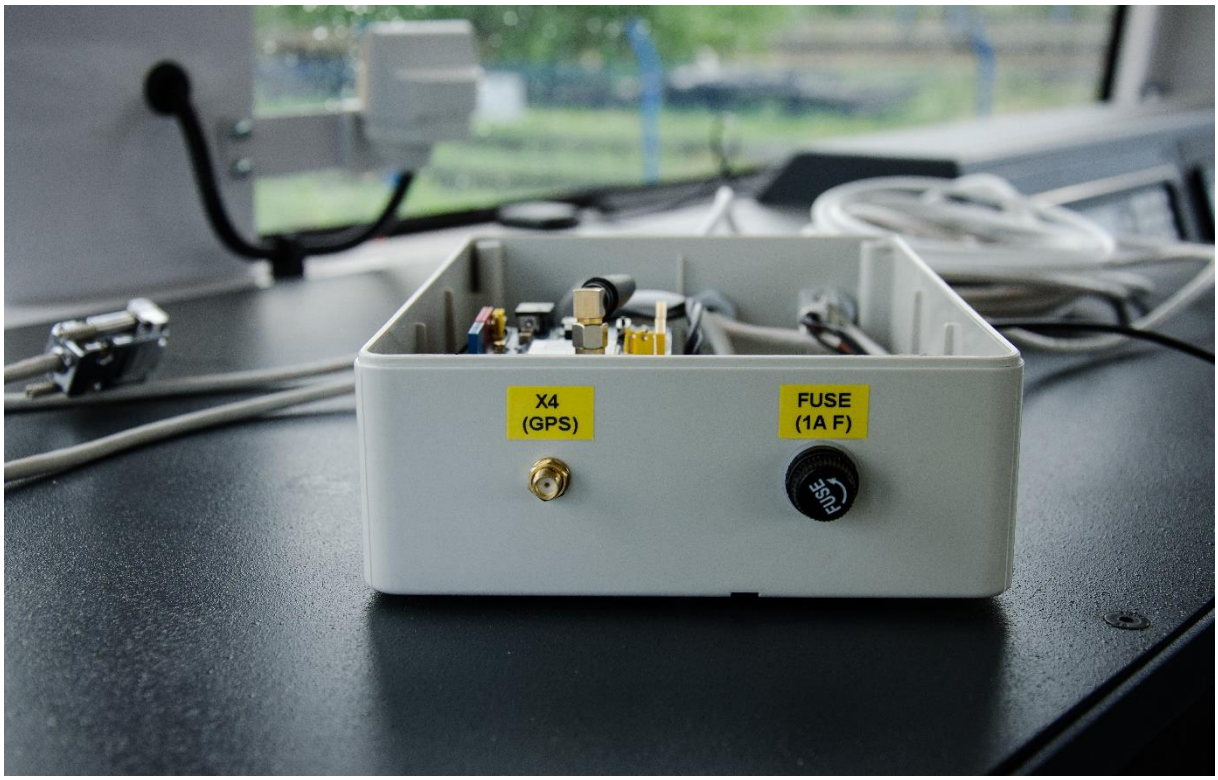


Obrázek 15 Vozidlová část v elektroinstalační krabici

Mimo otvory pro upevnění jednotlivých prvků distančními sloupky, je krabice rovněž opatřena nezbytnými kabelovými průchodkami a otvorem pro pojistkové pouzdro. Propojovací kabely mají délku 3 metry. Kabely pro CAN jsou zakončeny konektory CANON 9M a CANON 9F v krytkách. Napájecí kabel byl prozatím opatřen konektorem do autozásuvky, kterými jsou motorové vozíky vybaveny ve verzi napěťových úrovní 12 a 24 V. V krabici je patrný nevyužitý holý konektor CANON 9, který byl během testování využíván pro připojení CAN analyzátoru UCINT.



Obrázek 16 Průchodky pro kabely linky CAN a napájení



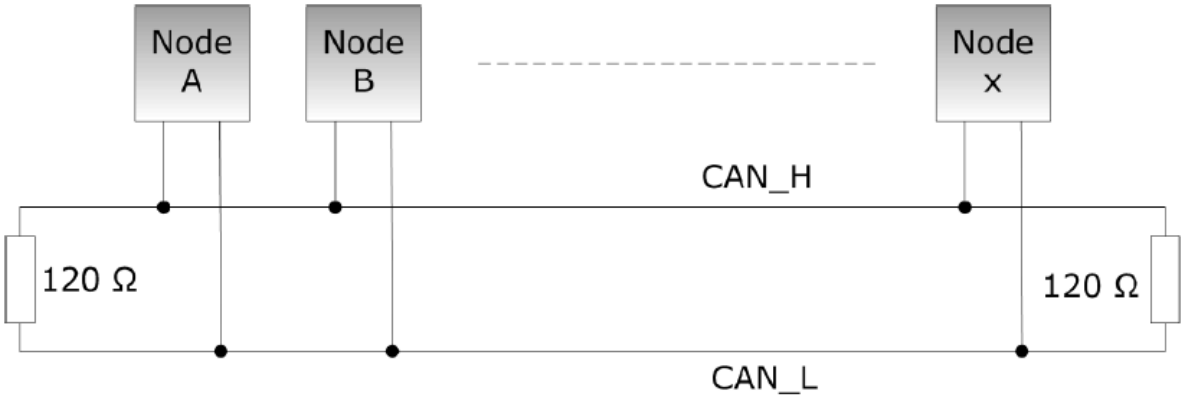
Obrázek 17 Výstup na GPS anténu a pojistkové pouzdro

Tabulka 2 Základní parametry mobilní části

Parametr	Hodnota/vybavení
Rozsah napájecího napětí	10–30 V DC
Proudový odběr (při napájení 24 V)	200 mA
Jištění	Rychlá trubičková pojistka 1 A
Vstupy/výstupy	1x CAN J1939 (250 kbit/s), bez zakončovacího rezistoru 120 Ω
	2x CANON 9 konektor (MALE, FEMALE), zapojené piny 2 – CANL, 7 – CANH, 3 – GND
	1x externí GPS anténa (SMA konektor) 1575,42 MHz (2,7–5,0 V)

4 FORMÁT CAN ZPRÁV

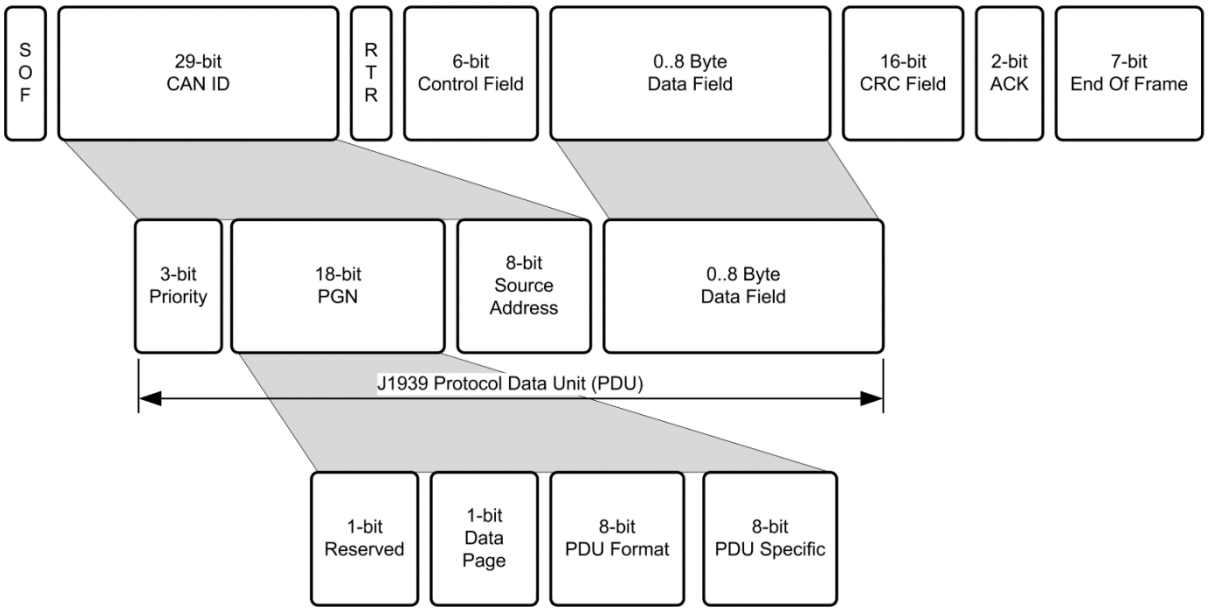
Zdrojem dat z vozidla je sběrnice CAN, a proto je vhodné zmínit její charakteristické vlastnosti. Sběrnice CAN je dvouvodičová sériová sběrnice. Stav na sběrnici určuje velikost rozdílového napětí mezi vodiči nesoucími označení CAN_H a CAN_L. Napěťové úrovně na sběrnici jsou definovány rozdílovým napětím označovaným jako recessive (2 V) a dominant (0 V). Využíván je metalický kroucený pár pro potlačení rušení se zakončovacím rezistorem 120 Ω pro eliminaci odrazů na vedení.



Obrázek 18 Topologie sběrnice CAN, zdroj: [6]

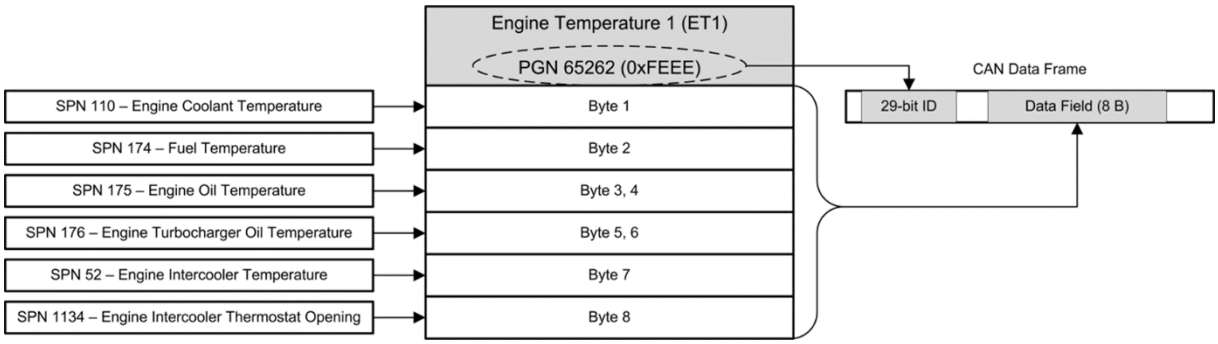
Sběrnice vozidla využívá přenosový protokol SAE J1939, který blíže specifikuje vlastnosti sběrnice, a tak je následující popis vztažen přímo k tomuto protokolu využívaném v motorových univerzálních vozících, nikoliv obecněji v rámci širší variability sběrnice CAN. Protokol SAE J1939 je obecně využíván ve vozidlech se vznětovými motory vyskytujících se na silničních, železničních i vodních cestách. Používá se nejen pro komunikaci s pohonnou jednotkou, ale i s dalšími prvky souvisejících s pohybem vozidla a bezpečnostní relevancí.

Komunikace probíhá v režimu multi-master, peer to peer nebo broadcast. Přenosová rychlost je 250 kb/s. Jednotlivé zprávy definuje 29 bitový identifikátor dle standardu CAN 2.0B. Datová část zprávy má velikost 8 bajtů. Je-li nutné přenášet delší informace, je využit transportní protokol a odesílání dat probíhá po paketech. Vytvořená mobilní část však přijímá pouze zprávy typu broadcast a nepodporuje přijímání multipaketů. Také nerozlišuje význam přijímaných dat. Převod zpráv do fyzikálního rozměru provádí až webová aplikace na serveru.



Obrázek 19 Formát CAN zpráv na sběrnici, zdroj: [6]

CAN řadič Ardunina DUE umožňuje nastavit 7 různých rozsahů identifikátorů zpráv pro filtraci přijímaných dat na úrovni řadiče. Výhoda spočívá v tom, že v případě doručení nechtěné zprávy není zprávou zatěžován samotný procesor. V případě doručení zprávy, která odpovídá jednomu z požadovaných rozsahů identifikátorů je z identifikátoru vyjmuta hodnota PGN a samotná data. PGN (Parameter Group Number) definuje typ a datový obsah zprávy v rámci protokolu J1939. Zbývající části zprávy jsou bez dalšího užitku zahozeny.



Obrázek 20 Význam PGN, zdroj: [7]

PGN 65262 Engine Temperature 1 - ET1

Transmission Repetition Rate:	1 s		
Data Length:	8		
Extended Data Page:	0		
Data Page:	0		
PDU Format:	254		
PDU Specific:	238	PGN Supporting Information:	
Default Priority:	6		
Parameter Group Number:	65262	(0xFEEE)	

Start Position	Length	Parameter Name	SPN
1	1 byte	Engine Coolant Temperature	110
2	1 byte	Engine Fuel Temperature 1	174
3-4	2 bytes	Engine Oil Temperature 1	175
5-6	2 bytes	Engine Turbocharger Oil Temperature	176
7	1 byte	Engine Intercooler Temperature	52
8	1 byte	Engine Intercooler Thermostat Opening	1134

Obrázek 21 Příklad obsahu zprávy protokolu J1939, zdroj: [8]

SPN 110 Engine Coolant Temperature

Temperature of liquid found in engine cooling system.

Data Length:	1 byte		
Resolution:	1 deg C/bit, -40 deg C offset		
Data Range:	-40 to 210 deg C	Operational Range:	same as data range
Type:	Measured		

Supporting information:

PGN	65262

Obrázek 22 Příklad struktury signálu (SPN) protokolu J1939, zdroj: [8]

V rámci jedné CAN zprávy je přenášeno více veličin. Samotné signály přijímané z řídícího systému trakčního pohonu mají velikost 2 bitů, 1 bajt, nebo 2 bajty. Vozidlová část systému vzdálené diagnostiky přijímá pouze zprávy z řídícího systému trakčního motoru a ECU spalovacího motoru. Přijímáno je celkem 18 CAN zpráv.

5 SOFTWARE MOBILNÍ ČÁSTI

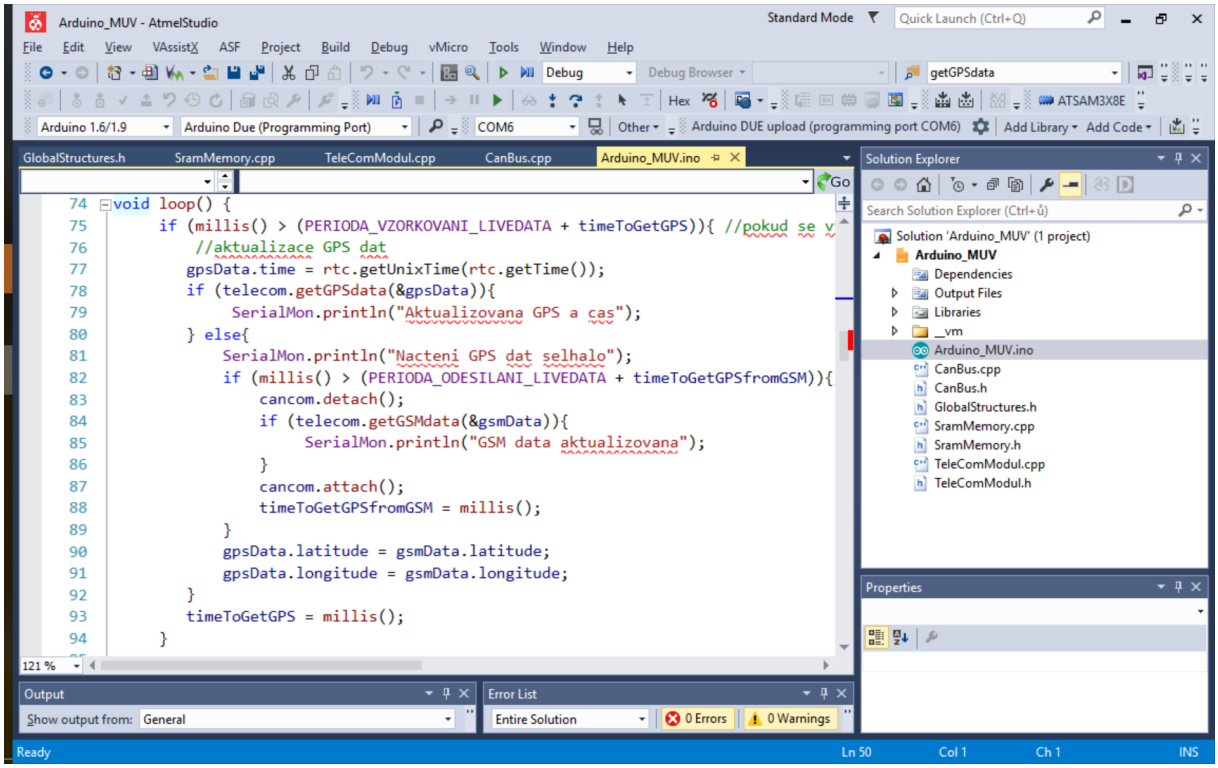
Projekt Arduino nabízí pro svůj hardware zároveň softwarovou podporu. Základním nástrojem k naprogramování funkce je program Arduino IDE. Aplikaci lze stáhnout na stránkách projektu. Využívá vlastní programovací jazyk Wiring, který však vychází z jazyka C/C++. Wiring je podle některých zdrojů označován jako framework jazyka C++, což by nakonec podpořil i fakt, že lze využívat klasickou C/C++ syntaxi. Prostředí Arduina IDE je jednoduché, ale pro složitější programy založené na více souborech je vhodnější používat plnohodnotnější řešení z nabídky programovacích prostředí. Jedním z kritérii výběru programovacích prostředí je nutná podpora Arduina. Jedním z takových je Atmel Studio, které je založené na oblíbeném Microsoft Visual Studiu. Po instalaci několika doplňků lze úspěšně používat Atmel Studio a primární soubor s příponou „.ino“ rozšířit o další vlastní knihovny s příponami „.cpp“ a hlavičkovými soubory „.h“. Mimo to lze samozřejmě využívat také externí knihovny, jako je tomu u Arduina IDE, i když zde to je přeci jen trochu složitější. Ne vždy chování programu odpovídá obvyklým zvyklostem.

V prostředí programu Atmel Studio byl vytvořen projekt s hlavním souborem *Arduino_MUV.ino*, třemi programovými moduly *CanBus.cpp*, *SramMemory.cpp* a *TeleComModul.cpp* pro zpřehlednění kódu a dalšími knihovnami třetích stran. Dále je ke každé knihovně jeden hlavičkový soubor a nadto soubor *GlobalStructures.h* s deklarací globálních struktur a definicí globálních konstant. Všechny tyto soubory jsou přítomny v projektu a byly vytvořeny v rámci této práce. Využívány jsou však také externí knihovny převzaté ze známého serveru GitHub.com s celosvětově největší komunitou vývojářů. V následující tabule je uveden přehled použitých programových modulů v projektu.

Tabulka 3 Přehled použitých programových modulů mobilní části

Programový modul	zdroj	Použití
GlobalStructures	vlastní realizace	Globální proměnné a struktury
CanBus	vlastní realizace	Zpracování zpráv ze sběrnice CAN
TeleComModul	vlastní realizace	Komunikace s modulem SIM808
SramMemory	vlastní realizace	Zápis/čtení externí paměti SRAM
ArduinoJson	převzatá	Zpracování formátu JSON
DS3231	převzatá	Zápis času do RTC a jeho čtení
due_can	převzatá	Zpracování zpráv ze sběrnice CAN
SRAM_23LC	převzatá	Zápis/čtení externí paměti SRAM
TinyGSM	převzatá	Komunikace s modulem SIM808
Timezone	převzatá	Převod času z UTC na místní čas

Poslední uvedená knihovna *Timezone* zůstala bez užitku vzhledem k její nahraditelné funkci na straně serveru, ale v programu je ponechána včetně nastaveného místního času s přechody standardní/letní čas (SEČ, SELČ) jako alternativní možnost řešení. Funkce realizuje změnu času z UTC na místní čas. Stejná funkce je defaultně realizována na straně serveru při předávání času ve formátu UNIX, bez nutnosti nastavování. Místní čas je odvozen od polohy serveru.



Obrázek 23 Projekt Arduino_MUV v programu Atmel Studio

Hlavní soubor *Arduino_MUV.ino* obsahuje funkci *setup()* a *loop()*. Funkce *setup()* realizuje počáteční nastavení po zapnutí napájení a funkce loop() představuje „nekonečnou smyčku“, která po vykonání funkce *setup()* běží až do odpojení zařízení od napájení. Funkci *loop()* ještě mohou přerušovat příchozí zprávy ze sběrnice CAN. Zprávy ze sběrnice CAN jsou přijímány na základě přerušení ve funkci *printFrame()*. Z funkcí *setup()*, *loop()* a *printFrame()* jsou pak volány ostatní funkce.

Obsluhu externích knihoven zajišťuje několik knihoven interních, které rozšiřují hlavní soubor s funkcemi *setup()* a *loop()*. Každá z interních knihoven obsahuje stejnojmennou třídu s veřejnými a privátními funkcemi. Veřejné funkce lze volat z libovolných souborů, ve kterých je daná knihovna includovaná. Privátní funkce nejsou dostupné zvenčí, pouze v rámci funkcí obsažených v dané knihovně.

5.1 Globální struktury, globální proměnné a jejich použití

Pro předávání parametrů mezi funkcemi slouží proměnné, které lze buď předávat při volání funkce, případně přes její návratovou hodnotu, nebo lze využívat globální proměnné. Využívání globálních proměnných se obecně příliš nedoporučuje, protože může docházet k jejich nechtěnému přepisování a zároveň není na první pohled příliš patrné, které proměnné jsou v kterých funkcí čteny a přepisovány. Někdy však nezbývá, než globální proměnné využít, ostatně jinak by jejich existence nebyla ničím opodstatněna.

Například funkce *printFrame()* volaná na základě přerušení od příchozích zpráv na CAN lince se nachází ve třídě *CanBus*. Skrze vstupní hodnoty lze do této funkce předat pouze data z linky CAN, nikoliv proměnné z hlavního souboru *Arduino_MUV.ino*, protože z něj není volána. Z tohoto důvodu tak vznikly globální proměnné. Zároveň není příliš žádoucí mít vytvořenou spoustu samostatných proměnných. Řad a proměnných má určitou souvislost s jinými, a tak je možné je shromažďovat do struktur a přistupovat k nim objektově. Struktury v rámci jazyka C/C++ mohou obsahovat proměnné různých datových typů.

```
typedef struct
{
	boolean  error;
	byte     dataRequest;
	uint16_t positionSRAMwrite;
	uint16_t positionSRAMread;
	uint16_t countSRAMwrite;
	uint16_t countSRAMread;
	uint16_t totalSamplesAmount;
	size_t   dataLength;
	uint16_t dataCapacity;
} InfoProcess_t;

typedef struct
{
	float   latitude;
	float   longitude;
	int16_t altitude;
	float   speed;
	time_t  time;
} GpsData_t;

extern InfoProcess_t process;
extern GpsData_t gpsData;
```

Zdrojový kód 1 Deklarace globálních struktur a globálních proměnných v GlobalStructures

V souboru *GlobalStructures* jsou deklarovány globální proměnné *process* a *gpsData*. Proměnná *process* struktury *InfoProcess_t* je definovaná v soboru *Arduino_MUV.ino*, již bez klíčového slova *extern*. Obsahuje všechny potřebné pomocné proměnné, které jsou výchozí pro rozhodování programu o ukládání a čtení dat do/z paměti SRAM a odesílání dat na server. Proměnná *gpsData* struktury *GpsData_t* nese informace o GPS poloze a aktuálním času. Řeší stav, kdy je třeba získávat data ze třídy *TeleComModul* ve třídě *CanBus* během volání přerušení a zároveň se tyto knihovny nemohou křížově includovat. Data v proměnné *gpsData* jsou tak periodicky „občerstvovaná“ ve smyčce *loop()* s voláním funkce ze třídy *TeleComModul* a funkce *CanBus::printFrame()* vždy získá poslední uložená data, pokud zrovna dochází k ukládání dat na SRAM se stanovenou periodou jedné sekundy.

Soubor *GlobalStructures* obsahuje i další struktury, které již nemají charakter globálních proměnných. Pokud vytváříme struktury a předáme je jako parametr volaných funkcí v knihovnách, předáváme spolu se samotnými daty název typu struktury, ale nikoliv strukturu samotnou. Pokud daná knihovna strukturu s daným názvem nezná, nezná zároveň ani formát předávaných dat, proto je vhodné struktury deklarovat globálně. Jako příklad zde uvádím strukturu, jejíž proměnná obsahuje vždy kompletní balík dat, která jsou jako jeden vzorek zasílaná na server, ukládaná do paměti SRAM, nebo vyčítaná z paměti SRAM a zasílaná na server. Zároveň je to příklad struktury s obsahem pole a vnořené struktury.

```
typedef struct {
	uint16_t pgn;
	uint8_t data[8];
	uint16_t expire;
	int32_t update;
} CANdata_t;

typedef struct {
	CANdata_t canData[POCET_VSECH_PRIJIMANYCH_PGN];
	float latitude;
	float longitude;
	uint16_t altitude;
	uint8_t speed;
	time_t time;
} LiveData_t;
```

Zdrojový kód 2 Deklarace globálních struktur lokálních proměnných v GlobalStructures

Struktura *CANdata_t* obsahuje jednu CAN zprávu definovanou svým PGN a hodnotou v proměnné *data* o velikosti 8 bajtů. Struktura dále obsahuje časový údaj v milisekundách, po který je zpráva platná od chvíle jejího přijetí (proměnná *expire*). Položka *update* obsahuje

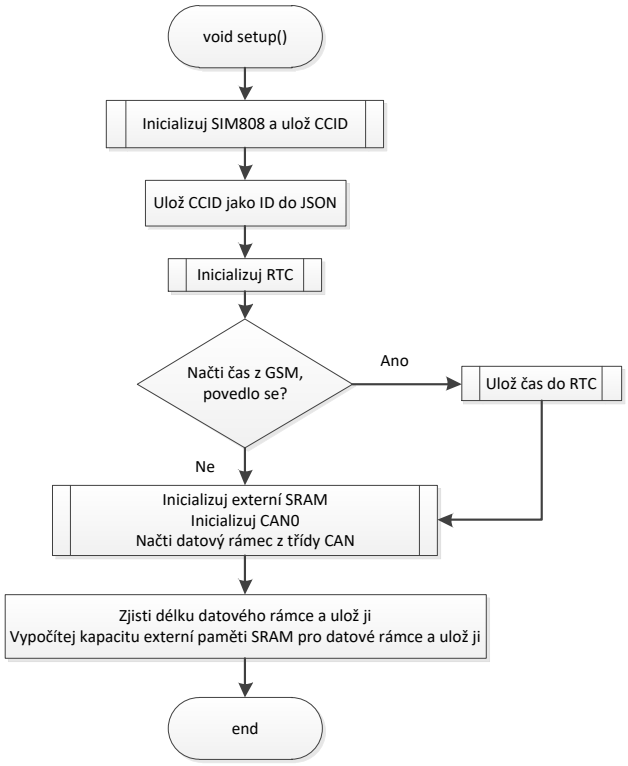
časovou značku z funkce *millis()*. Funkce *millis()* vrací počet milisekund od počátku spuštění Arduina. Při uložení tohoto údaje lze pak s odstupem času zjistit dobu, která uběhla od uložení hodnoty. Na základě časové značky posledního uložení a času expirace lze rozhodnout, zda je daná zpráva ještě platná, nebo nedošlo v definovaném čase k její aktualizaci, a tak je prohlášená za neplatnou, což v daném případě znamená přepsání všech bajtů zprávy (proměnná data) na hodnotu 0xFF. Tato funkce je pak vykonávaná během vykonávání programu a bude zmiňována dále.

Struktura *CANdata_t* je vnořena do struktury *LiveData_t* jako pole přijímaných CAN zpráv. Aktuální vzorek dat má vždy jednu informaci o zeměpisné šířce, délce, nadmořské výšce, informaci o rychlosti a zaznamenaný čas.

5.2 Funkce setup()

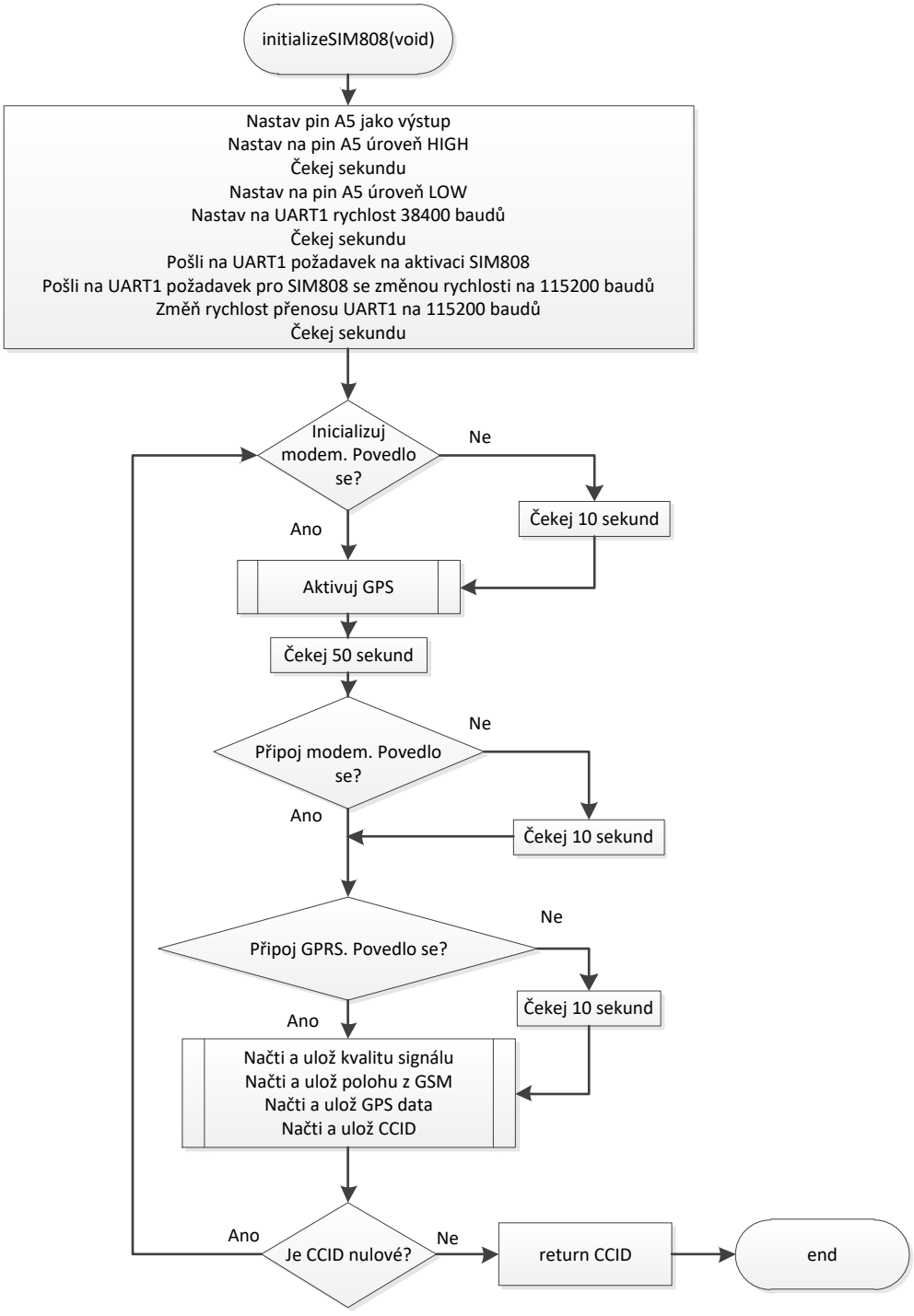
Funkce *setup()* realizuje počáteční nastavení po spuštění systému. Probíhá zde inicializace všech součástí systému. Z GSM dat je také zjišťován aktuální čas UTC, který je předán modulu reálného času. Ten pak aktuální čas poskytuje pro každý ukládaný vzorek dat.

Postup vykonávání instrukcí objasňuje následující vývojový digram funkce *setup()*. Popisy jednotlivých funkcí systému budou dále doprovázeny vývojovými diagramy pro lepší pochopení dané problematiky.



Vývojový diagram 1 Funkce setup()

Nejzajímavější částí funkce *setup()* je inicializace modulu SIM808, kterou zde také uvádím. Popis funkce následuje pod vývojovým diagramem této funkce.



Vývojový diagram 2 Funkce initializeSIM808()

Funkce nejdříve mění stavovou hodnotu na pinu A5. Účelem je přestavení modulu do aktivního stavu. Dále probíhá, možná na první pohled trochu nelogické, nastavení rychlosti pro komunikaci se SIM808 na rychlost 38400 baudů a následně její změna na hodnotu 115200 baudů. Zařízení by totiž mělo samo poznat rychlost komunikace a té se přizpůsobit. Bohužel

tato teorie byla v praxi ověřena jako nefunkční, alespoň v souvislosti s dodaným firmwarem v modulu SIM808. Probíhalo dlouhé hledání chyby, až se podařilo s využití programu pro zasílání AT příkazů přímo do modulu SIM808 naleznout komunikační rychlost, na kterou modul reaguje. Po zaslání požadavku na změnu přenosové rychlosti AT příkazem již modul SIM808 odpovídá požadovanou rychlostí, a pak ještě zbývá změnit komunikační rychlost na straně Arduina.

Během komunikace s GSM modulem si lze všimnout řady vložených zpoždění. Zpoždění tam nejsou náhodou. Některá byla obsažena již přímo v příkladu ke knihovně *TinyGSM*, další pak byla doplněna na základě zkoušek. Tím největším zpožděním je čekání po zapnutí GPS. Zjišťování polohy těsně po zapnutí GPS modulu trvá běžně do 1 minuty. Při špatné dostupnosti signálu trvá čekání na platná GPS data ještě déle, nebo případně nejsou informace dostupné vůbec a v takovém případě je možné alespoň přibližnou (nepřesnou) informaci o poloze získat z GSM, jak je rozebráno dále ve funkci *loop()*. Pokud bychom vynechali zpoždění po zapnutí GPS, byla vždy u prvních vzorků brána informace o poloze z GSM.

Při zasílání zpráv z vozidla na server se vozidlo identifikuje svým ICCID (Integrated Circuit Card Identifier). To je unikátní identifikátor SIM karty, který karta získá při svém vyrobení. Výhodou je, že není jakkoliv nutné zasahovat do programu mobilní části při nasazení vzdálené diagnostiky na nové vozidlo. Pouze se identifikátor SIM karty přidá pomocí webové aplikace na server jako ID vozidla, když je vozidlo vkládáno v nastavení.

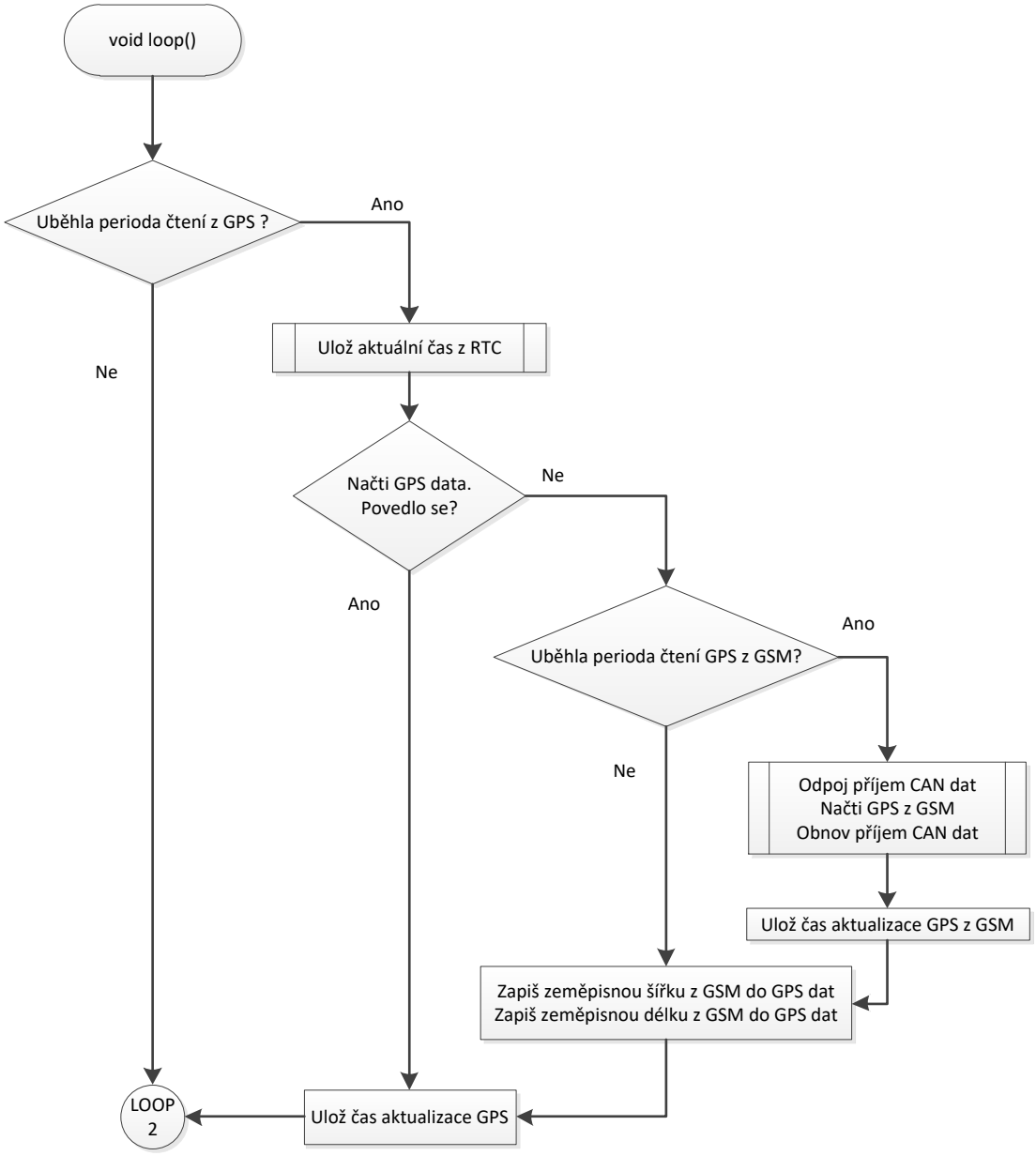
5.3 Funkce loop()

Funkce nekonečné smyčky, *loop()*, plní dvě hlavní úlohy. Aktualizuje data z GPS a čas a zároveň iniciuje odesílání dat na server. Perioda aktualizování GPS dat a času je shodná s periodou vzorkování dat, které jsou jako celek s touto periodou ukládána do externí paměti SRAM.

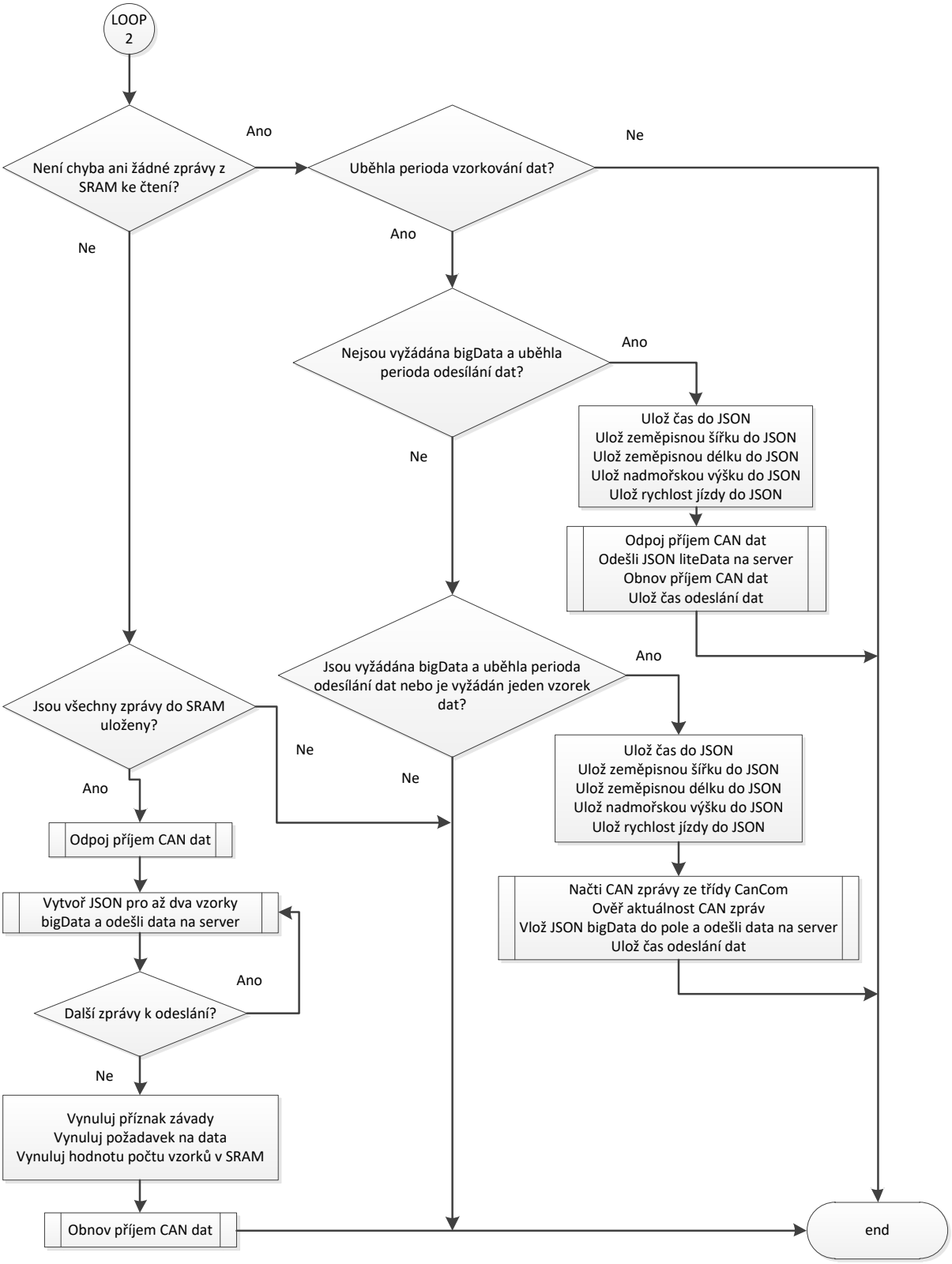
GPS data jsou v místech s dostatečným dohledem satelitů brána z GPS modulu s připojením GPS antény na frekvenci 1575,42 MHz. Ne vždy je však k dispozici signál, alespoň ze 4 družic, abychom dostali informaci nejen o zeměpisné šířce, výšce, ale také informaci o nadmořské výšce a rychlosti jízdy. Všechny tyto parametry jsou odesílány na server. Pro případy, kdy nejsou k dispozici žádná GPS data je možné zeměpisnou šířku a výšku získávat alternativním způsobem z GSM. Z rozmístění anténních vysílačů je možné alespoň přibližně určit informaci o zeměpisné šířce a výšce, kterou nám následně BTS po vyžádání zašle. Nevýhodou zjišťování polohy z GSM je nejen nepřesnost, neúplnost dat (chybí informace

o nadmořské výšce a rychlosti), ale také pomalá odezva systému. Zatímco informaci o poloze z GPS modul SIM808 poskytne takřka okamžitě, při získávání polohy z GSM dochází k prodlevě přibližně 3 sekundy čekáním na odpověď.

Standardně je tak GPS poloha aktualizována každou sekundu a při nedostupnosti GPS dat každých 10 sekund, což je vzhledem k nepřesnosti polohy z GSM vysílačů dostačující perioda. Při čekání na data je však nutné dočasně zakázat příjem CAN zpráv, což platí obecně pro veškerou GSM komunikaci, jinak dochází k selhání přijímaných informací. Dočasné zakázání příjmu CAN zpráv však narušuje nejen příjem dat z CAN linky, ale také periodu ukládání dat do paměti SRAM, kde tak dochází k několikasekundovým výpadkům. Zakázání a povolení příjmu CAN zpráv se provádí pomocí funkcí *cancom.detach()* a *cancom.atach()*.



Vývojový diagram 3 Funkce loop() 1. část



Vývojový diagram 4 Funkce loop() 2. část

Z vývojového diagramu výše je patrné, že data se na server mohou odesílat ve třech různých podobách. Třemi různými cestami ve smyslu pohybu po vývojovém diagramu.

1. V prvním případě se nevyskytují příznaky závad na CAN lince a od klienta nejsou vyžadována data. Na server jsou odesílána jsou tzv. *liteData* bez zpráv CAN. Datový rámec *liteData* obsahuje jen ID vozidla (ICCID), časovou značku, informace o poloze a rychlost jízdy. Tím dává vozidlo také informaci o svém aktivním stavu a plní funkci periodického dotazování s předáním řídících informací mobilní části. Na jejich základě vozidlo zašle, nebo bude periodicky zasílat, požadovaná data ze sběrnice CAN na server.
2. Je-li vyžádán webovým klientem pouze jeden vzorek dat, nebo je-li požadováno trvalé periodické odesílání dat, pak jsou posílána tzv. *bigData* s daty z CAN zpráv. Před odesláním jsou data získána ze třídy *CanBus* a CAN zprávy projdou kontrolou, zda nedošlo k vypršení jejich platnosti. Pokud z nějakého důvodu CAN zpráva nebyla mobilní částí do nastaveného timeoutu přijmuta, vyprší platnost dat dané zprávy ve struktuře *LiveData_t* a do všech datových bajtů zprávy ve struktuře se zapíše hodnota 0xFF jako indikace neplatného obsahu zprávy. Pokud jsou všechny bity u daného signálu nastavené na 1, znamená to v protokolu J1939, že hodnota signálu není dostupná. Data jsou následně přenesena do formátu JSON (kapitola 6) a odeslána na server.
3. Poslední z cest odesílaných dat se uplatní po výskytu zprávy s příznaky závad, nebo pokud ve webové aplikaci byla vyžádána data volbou *Vyžádat více.* V případě reakce na výskyt závady musí nejdříve proběhnout zaplnění ½ externí paměti SRAM daty po výskytu závady a následně je odesílán celý obsah externí paměti SRAM. Při vyžádání dat z webové aplikace je odesláno z vozidla na server posledních 60 vzorků dat. Zde byla uvažována možnost zasílání většího množství vzorků společně na server, čím by bylo ušetřeno množství HTTP hlaviček spolu s informací s ID vozidla, které se musí posílat. Tato myšlenka však v zásadě naráží na dvě omezení. Prvním je kapacita výstupního bufferu modulu SIM808, který má velikost 1460 bajtů. Knihovnou *TinyGSM* je hodnota pro jistotu dostatečného množství paměti ještě omezena na hodnotu 1 kB. Druhé omezení vyplývá z funkce knihovny *ArduinoJson*, která plně neumožňuje dynamické vytváření objektů a jejich plnění do pole. Z těchto důvodů je množství vzorků v poli omezeno na dva. Pole JSON objektů zároveň není možné plnit ve smyčce *loop()*, protože by neustále docházelo k jeho zvětšování, přestože by bylo plněno stejnými hodnotami, protože pole přebírá JSON objekt pouze jako pointer. Proto vznikla další funkce, *createJsonArrayAndSend()*, jejíž proměnné po jejím ukončení

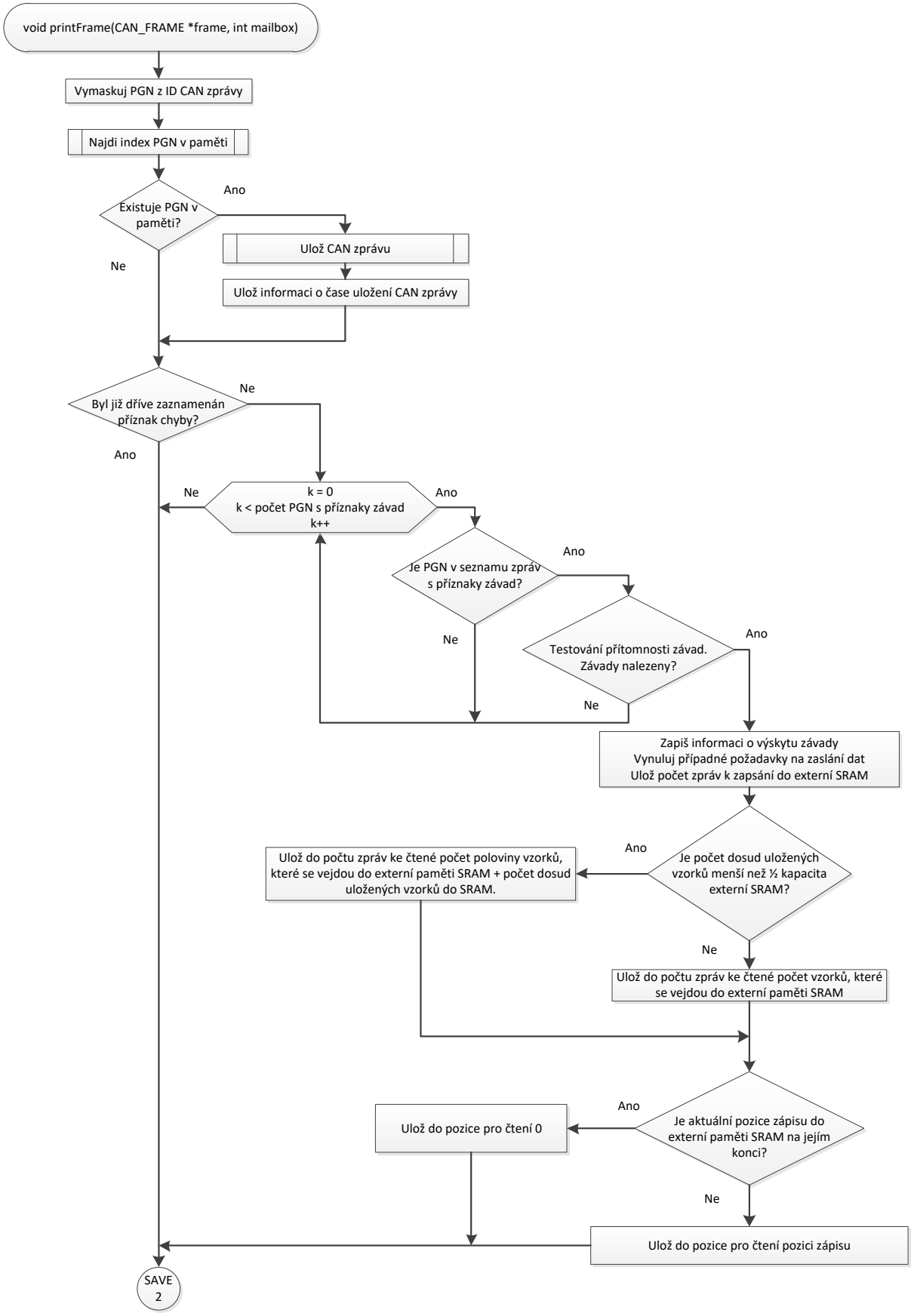
zanikají a nedochází tak k neustálému zvětšování pole JSON objektů. Ve funkci jsou zároveň vždy vyčítána další data z externí paměti SRAM.

5.4 Funkce volaná přerušením – CanBus::printFrame()

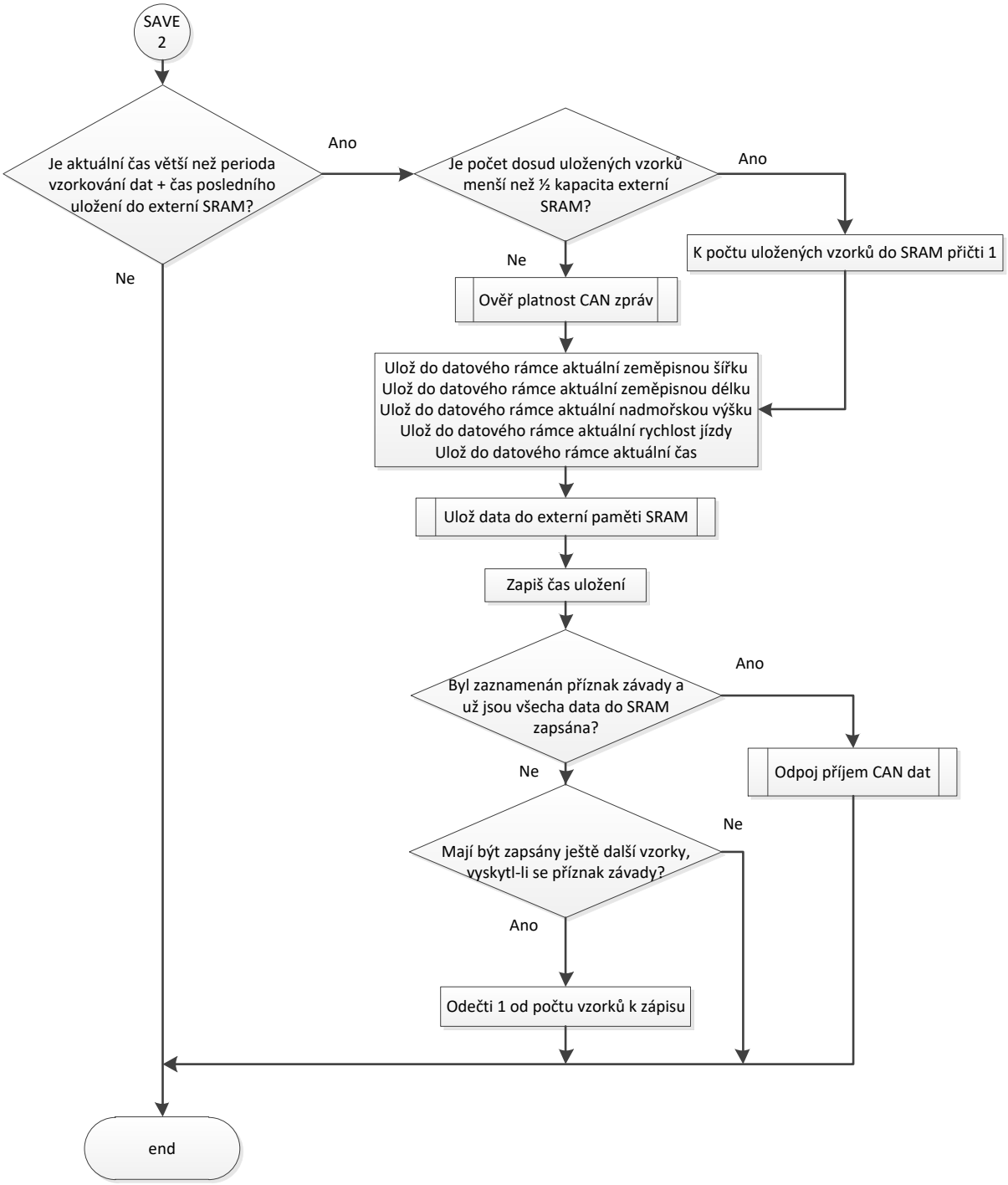
Funkce *printFrame()* je volaná přerušením vyvolaným řadičem CAN při příjmu zprávy. Funkce ukládá data do struktury *LiveData_t* a před uložením kontroluje timeoutu příjmu CAN zpráv.

Přijde-li zpráva s příznaky závad řídícího systému trakčního pohonu s PGN 0xFF10, pak je provedena kontrola na přítomnost závad. V rámci 8 bajtové zprávy je přenášeno celkem 27 příznaků závad od řídícího systému trakčního pohonu. Každý příznak závady je přenášen pomocí dvou bitů. Hodnota bitů 0b01 vyjadřuje přítomnost závady. Zařízení reaguje na přítomnost alespoň jedné závady následujícím způsobem – informace o závadě je zanesena do procesní struktury a je nastaveno množství vzorků, které bude ještě uloženo do externí paměti SRAM a kolik jich bude také následně z paměti vyčteno se zohledněním okrajových podmínek.

Do externí paměti jsou po výskytu závady kontinuálně ukládána data. Po uložení všech požadovaných vzorků od výskytu závady se zakáže příjem CAN zpráv a následuje zasílání dat na server ve smyčce *loop()*, kde se následně obnoví příjem CAN zpráv opětovným provázáním přerušení s funkcí a proces se vrací opět do bezporuchového stavu. Tímto postupem je docíleno stavu, kdy jsou k dispozici data před výskytem závady, během jejího prvního výskytu i po něm a je tak možné sledovat časový vývoj v oblasti výskytu závady po odeslání všech dat z paměti SRAM na server.



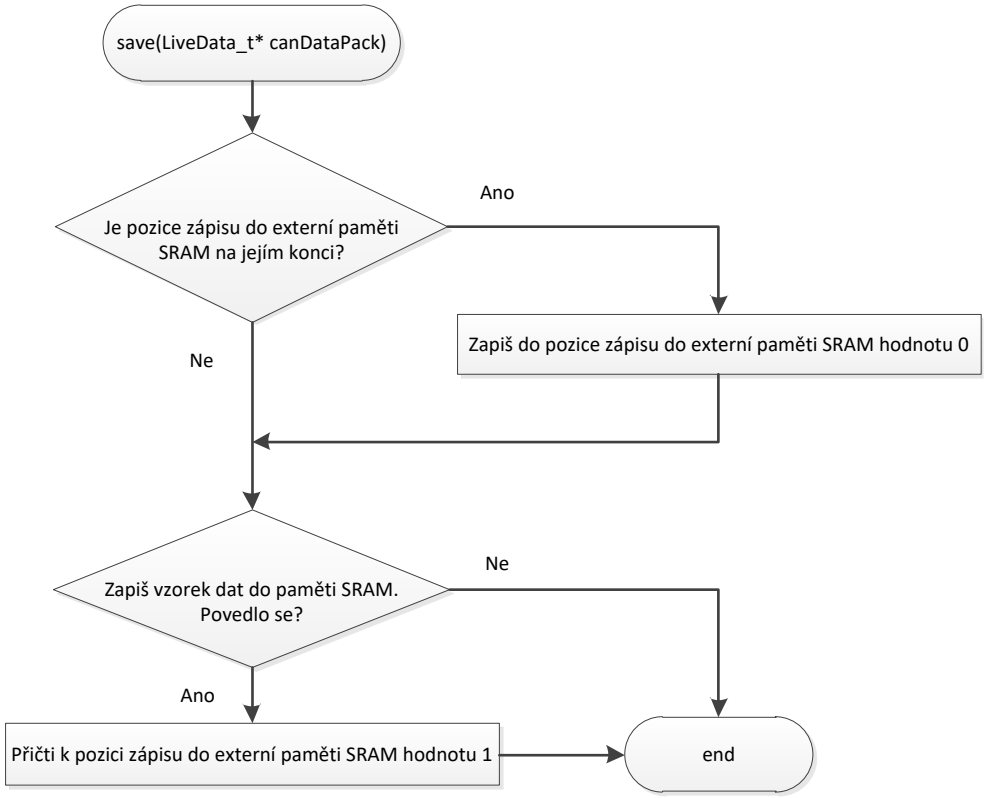
Vývojový diagram 5 Funkce CanBus::printFrame() 1. část



Vývojový diagram 6 Funkce CanBus::printFrame() 2. část

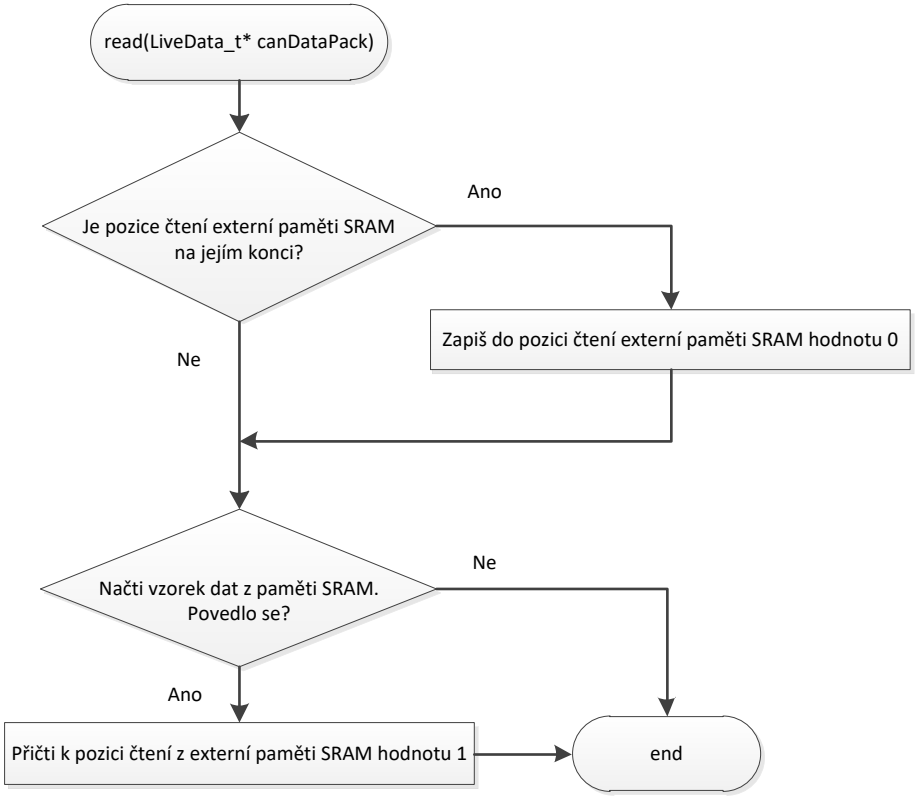
5.5 Funkce SramMemory::write() a read()

Z předchozích funkcí nebyl patrný způsob čtení a ukládání dat do paměti SRAM, která je podpořena knihovnou *SRAM_23LC*. Zde se projeví další z výhod používání struktur. Struktura *LiveData_t* obsahuje všechny potřebné informace a představuje jeden vzorek pro zaslání na server. Každý vzorek se ukládá do externí paměti SRAM tím způsobem, že se obsah datové struktury *LiveData_t* kopíruje do externí paměti SRAM bajt po bajtu.



Vývojový diagram 7 Funkce SramMemory::save()

Obdobný proces probíhá ve funkci read() pro čtení dat z paměti SRAM.



Vývojový diagram 8 Funkce SramMemory::read()

5.6 Testování a ladění chyb

Testování a ladění chyb je neoddělitelnou součástí programování. Syntaktické chyby a některé chyby závislostí dokáží odhalit programy při překladu (kompilaci). Jiné jsou odhalitelné přímo za chodu zařízení. Za účelem usnadnění ladění programu aplikace odesílá na rozhraní UART Arduina DUE (s následným převodem na rozhraní USB – Native Port) ladící hlášky ve formě řetězců. Na straně PC pak stačí pro příjem hlášení použít program typu terminál. Pro odeslání řetězců slouží na Arduinu funkce print() a println(). Funkce println() v porovnání s print() navíc odřádkuje.

Obrázek 24 Sériový monitor programu Arduino IDE

Pro zjišťování délky vykonávání jednotlivých procesů mohou posloužit vstupně/výstupní porty, na kterých lze generovat obdélníkový signál a průběhy mohou být vykreslovány na osciloskop.

Zasílání vozidlových dat na sběrnici CAN je možné simulovat. Využit byl USB-CAN převodník USB2CAN od společnosti CANLAB a převodník UCINT firmy Elbas. Přípravky mají USB konektor pro připojení do PC a obslužnou aplikaci na PC. Kromě vytváření vlastních zpráv a jejich generování na sběrnici CAN je možné s přípravkem UCINT ukládat časové záznamy ze sběrnice a ty následně kdykoliv na sběrnici reprodukovat, což se hodí právě pro případ testování bez přítomnosti vozidla, pouze s připojením k PC, které současně také poskytuje diagnostické pracoviště pro sledování chodu Arduina prostřednictvím monitorovací sériové linky.

6 VOLBA KOMUNIKAČNÍHO PROTOKOLU A FORMÁTU DAT PRO PŘENOS NA WEB

V této části už se pozvolna dostáváme k softwarovému řešení webové aplikace. Pro komunikaci se serverem je tady k dispozici modul SIM808 vybavený technologií GPRS. Pro Arduino je k dispozici řada knihoven, které řeší alespoň tu nejspodnější vrstvu komunikací a přenosových protokolů. Pro komunikaci s řadou komunikačních modulů kombinovatelných s Arduinem se využívají AT příkazy. Zasílání AT příkazů však už zajišťuje knihovna *TinyGSM*, která obsluhuje GPS a má i povědomí o AT příkazech k GPS, právě pro případ využití modulu SIM808, který obsahuje i GPS modul.

Knihovna *TinyGSM* podporuje nad vrstvou TCP především protokoly HTTP a MQTT. Zároveň by mělo být možné využít i zabezpečení SSL/TLS na protokolu HTTPS. HTTP je standardní protokol využívaný v internetových sítích. Naproti tomu protokol MQTT byl přímo navržen pro potřeby IoT, kdy mezi sebou komunikuje více zařízení, která jsou neustále on-line. Existují ale i jiné protokoly (CoAP) a s rozvojem IoT proběhl a probíhá vývoj i v této oblasti na poli komunikačních protokolů s důrazem na minimalizaci přenášených dat, trvalé připojení pro oboustrannou komunikaci, zabezpečení a v neposlední řadě také možnost jednoduché implementace na mikrokontrolery.

HTTP protokol neumožňuje obousměrnou komunikaci. Vychází z filozofie, kdy klient pošle dotaz a server následně zašle odpověď. HTTP hlavičky, které jsou se zprávami vždy odesílány jsou poměrně velké a obsah zpráv je přenášen ve formě řetězců. HTTP protokol má však i několik výhod. Na straně serveru s využitím REST API není nutné instalovat službu, která by vyžadovala, pro ověřovací provoz, nákladný webhosting s vlastním virtuálním serverem (VPS). Spokojit se lze s webhostingem zdarma včetně domény třetího řádu. Další s tím související výhoda spočívá v přímé komunikaci, bez datového překladu, protože i protokol MQTT přenáší data ve formě řetězců a pro datovou úsporu nad rámec HTTP hlaviček by bylo nutné data zpracovávat v dalším kroku. Protokol MQTT má v rámci úspor tři úrovně QoS (Quality of Service), které definují zabezpečení přenesení zpráv s tím související datovou náročnost v podobě potvrzování přijetí zprávy. Odeslání jednou bez potvrzení přijetí (QoS 0), přijetí minimálně jednou (QoS 1) a přijetí právě jednou (QoS 2). Tato problematika vychází z obousměrné komunikace, kdy odesílatel (publisher) rozesílá prostřednictvím prostředníka (broker) několika příjemcům (subscriber) a při úrovni QoS 2 tak s každým příjemce proběhne dvoukolové potvrzování.

Komunikace mezi vozidly pro potřeby vzdálené diagnostiky nedává smysl. Zároveň datové rámce při posílání dat ze sběrnice CAN přesahují velikost hlavičky. Méně příznivá je situace v rámci odpovědi serveru, ale máme zajištěno podle dělení MQTT QoS na nejvyšší úrovni 2, tedy server potvrzuje přijetí a nad to posílá data, která chceme předat vozidlu (vyžádání dat). V otázce volby přenosového protokolu tak bylo rozhodnuto ve prospěch HTTP. V případě reálného nasazení by však bylo žádoucí otázku komunikačních protokolů ještě zvážit. Při větším množství vozidel by již do celkové ceny řešení v rámci provozních nákladů mohly významně promlouvat datové tarify mobilních operátorů, což by mohlo hrát ve prospěch VPS serveru, případně podpůrných služeb třetích stran, které se specializují na podporu IoT.

V rámci přenosového protokolu HTTP je možné volit mezi různými metodami předání dat. Jejich rozdíly se však při použití s Arduinem trochu stírají, protože jejich hlavní diference vyplynou na povrch při komunikaci mezi webovým prohlížečem a serverem. Pro nejpoužívanější metody GET a POST jsou však stanovená doporučení. Data posílaná metodou GET by neměla být jakkoliv ukládána. Měla by pouze vracet výsledek pro klienta. Obsah metody GET je totéž co adresa ve webovém prohlížeči pokud si odmyslíme HTTP hlavičku, se kterou se běžný uživatel nesetká. Naproti tomu metoda POST představuje „zadní vrátka“ při odesílání dat, která se již k ukládání využívají a mělo by jít o unikátní data, u kterých není žádoucí opakované odesílání stejného obsahu. Jde primárně o data přebíraná z webových formulářů.

Budu se tedy držet tohoto pravidla a využiji metodu POST. Při menším množství proměnných lze v rámci metody GET i POST realizovat zápis dat následujícím způsobem.

```
promenna1=hodnota1&promenna2=hodnota2&…
```

Vzhledem ke skutečnosti, že je potřeba přenášet zhruba dvacítku zpráv CAN a další informace s ID vozidla, informace o čase a poloze, bude vhodné se porozhlédnout po lepším řešení umožňující definování datových struktur. Tento požadavek představuje zásadní skutečnost neslučitelnou s metodou GET, která jiné datové konvence nepřipouští. Starším řešením je formát XML. Stejnou funkci v zásadě plní také inovativní formát JSON (JavaScript Object Notation), který je jeho odlehčenou podobou přinášející datovou úsporu. Jde o jednoduchý formát, který je možné jednoduše číst strojově i lidmi, který však nepopisuje typy dat, která přenáší, a tak není vhodný pro přenos binárních dat. Nejefektivnějším způsobem, jak ještě relativně jednoduše přenášet zprávy ze sběrnice CAN s délkou dat jedné zprávy 8 bajtů jako řetězce je převod dat do hexadecimálního formátu. Každý bajt reprezentují v hexadecimálním formátu dva znaky, což při převodu celé zprávy do řetězce představuje 16

znaků, tedy 16 bajtů dat. Další data pak zabere název zprávy (PGN), kde by se však v případě požadavku na další datovou úsporu dala provést s využitím převodní tabulky minimalizace do dvou znaků, které by při posílaném množství zpráv stále byly unikátní. Podpora JSON pro Arduino existuje v podobě knihovny *ArduinoJson*, se kterou pak lze vytvořit například následující obsah (xxxx – skryté hodnoty), který je následně odeslán na server.


```
{"data":[{"ti":1524209366,"la":15.77149,"lo":50.04537,"al":210,"sp":35,"ff10":"0300170404c000f0","ff
12":"4d015e01b4001e05","xxxx":"0000700400000000","xxxx":"7c1b00190000d001","xxxx":"60c67d621313400a"
,"xxxx":"7e7d7dfffffffffff","xxxx":"f0fffffffffffff","xxxx":"3a07c800c8000000","xxxx":"3b07c900cc000
000","xxxx":"01fcfffffffffff","xxxx":"05f0fffffffffff","xxxx":"e0abfa8e10a4ffff","xxxx":"a08c8890c
800d200","xxxx":"81908190ffffffff","xxxx":"ffffffffffffffff","xxxx":"ffffffffffffffff","xxxx":"fffff
fffffffffff","xxxx":"ffffffffffffffff"}],"id":"8942031016282054984","bd":1}
```


Zdrojový kód 3 Data přenášená z vozidla na server uložená ve formátu JSON

Položku *id* reprezentuje unikátní číslo SIM karty, takzvané ICCID, které je zároveň unikátním identifikátorem vozidla pro webovou aplikaci. Položka *bd* říká, zda jsou posílána *bigData*, či nikoliv. Položka *data* je indexované pole a v něm neindexované pole se záznamy jednoho vzorku dat: *ti* – čas UNIX, *la* – zeměpisná šířka (latitude), *lo* – zeměpisná délka (longitude), *al* – nadmořská výška (altitude), *sp* – rychlost (speed) a dále jednotlivá PGN. Tento datový objem se posílá, pokud je vyžádáno trvalé zasílání dat z mobilní části na server, nebo jeden vzorek dat. Při volbě na webu *Vyžádat více*, nebo po zjištění příznaku závady trakčního pohonu se v indexovaném poli posílají dva vzorky dat.

V rámci formátu JSON je klíč („název proměnné“) vždy uveden v uvozovkách. Od hodnoty je oddělen dvojtečkou. Pokud je hodnota řetězec, uvádí se také v uvozovkách. Jde-li o číslo, uvozovky se nepoužívají. Má-li klíč více položek, ke kterým je potřeba přistupovat pomocí indexů[], použijí se hranaté závorky a jako oddělovače položek čárky. Není-li k položkám třeba přistupovat pomocí indexů (mají vlastní klíče), použijí se špičaté závorky.

6.1 Teoretický výpočet spotřeby mobilních dat

Pro výpočet spotřeby přenesených dat vycházím z délky řetězce JSON dat posílaných z mobilní části na server a přijímaných ze serveru mobilní částí včetně zpráv CAN z výše uvedeného řetězce. Je nutné připočítat délky HTTP hlaviček, které jsou do výpočtu zahrnuty ve své přibližné minimální délce potřebné pro přenos.

Množství přenesených dat při odeslání jedné zprávy

$$M_{dat/přenos} = m_{B/znak}(n_{JSON1} + n_{JSON2} + (n_{HTTPheader} * 2)) \tag{6.1}$$

$$M_{dat/přenos} = 1 * (575 + 12 + (50 * 2)) \cong 687\ B \tag{6.2}$$

- Předpokládané množství přenesených dat mezi mobilní částí a serverem s jednou datovou zprávou. $M_{dat/přenos}$ $[B]$
- Počet bajtů potřebných k přenesení jednoho ASCII znaku $m_{B/znak} = 1$ B
- Délka JSON řetězce odesílaná na server $n_{JSON1} = 575$
- Délka JSON řetězce přijímaná ze serveru $n_{JSON2} = 12$
- HTTP hlavička odesílaná spolu se zprávou. Přibližná hodnota počtu znaků. $n_{HTTPheader} = 50$

Spotřeba dat pro jedno vozidlo za měsíc

Pro výpočet celkové měsíční spotřeby uvažuji denní nasazení vozidla 12 hodin a zasílání zprávy z vozidla na server každých deset sekund.

$$M_{dat/měsíc} = n_{dní} * n_{hodin} * n_{minut} * n_{1/min} * M_{dat/přenos} \quad (6.3)$$

$$M_{dat/měsíc} = 30 * 12 * 60 * 6 * 687 = 89 \text{ MB} \quad (6.4)$$

Modelová spotřeba 89 MB za měsíc by přibližně odpovídala stavu, kdy by každých 10 sekund 12 hodin denně byla data odesílána na server včetně CAN zpráv. Tomu odpovídá volba *Vyžádat trvalé zasílání dat* na webu, kde je však perioda o trochu delší (~ 14 s) vlivem připočtení času potřebného k odeslání dat z vozidla na server a přijetí odpovědi ze serveru, což by bylo možné jednoduše přizpůsobit tomuto příkladu.

Samotná hodnota spotřeby 89 MB za měsíc patří rozhodně mezi ty větší v oblasti IoT. Měsíční datové balíčky pro M2M zařízení se pohybují od jednotek MB po několik málo stovek MB, ale ani v tomto případě veřejné ceníky, ani u virtuálních operátorů na „specifickém“ českém mobilním trhu, nenabízí příliš zajímavé ceny, případně jen pro velice malé datové přenosy, ale při porovnání ceny a množství nabízených dat vychází takové tarify nejdráž. Pravděpodobně se vyplatí vyjednání osobních podmínek s mobilními operátory pro rozsáhlejší systémy s velkým množstvím komunikujících zařízení. Pro náš případ zkušebního testování byla využita předplacená datová SIM karta, která aktuálně při nákupu vychází na 200 Kč a následně za měsíční poplatek 100 Kč je poskytováno 500 MB dat.

7 WEBOVÁ APLIKACE

Webová aplikace je napsaná v PHP, celosvětově nejoblíbenějším skriptovacím jazyku na straně serveru. Jazyk je to zároveň interpretovaný, což znamená, že na rozdíl od jazyka C/C++ se zde neprovádí kompilace. Dnes už není mnoho vývojářů, kteří píší aplikace v čistém PHP, které zabírá příliš mnoho času. Na řadu přišly frameworky, které poskytují v základu lepší podporu pro psaní aplikací. Zjednodušují a zpřehledňují psaní kódu a zároveň zvyšují úroveň bezpečnosti a pomáhají s laděním chyb. Webová aplikace využívá český Nette Framework, který je předním frameworkem v PHP na českém a slovenském trhu. Šířen je jako svobodný software.

Instalace, aktualizace frameworku a závislostí se provádí s využitím utility Composer, která však nemá grafické rozhraní. Její ovládání probíhá z příkazového řádku. Webová aplikace ve své kořenové složce obsahuje soubory *composer.json* a *composer.lock*. Soubor *composer.json* definuje požadavky programátora na verzi Nette, jeho součásti a součásti externích autorů. Soubor *composer.lock* je pak aktualizován programem Composer při každé ručně vyžádané aktualizaci. Zároveň aktualizuje soubor *composer.lock* a zapisuje do něj podrobnosti k aktualizaci, které pak využívá při aktualizaci následující a případně zabrání aktualizaci při nekompatibilitě jednotlivých součástí. Před aktualizací je však přesto jistější předchozí verzi zálohovat.

Nette Framework je reprezentantem architektury MVC (Model-view-controller). MVC je návrhová architektura, která odděluje aplikační logiku, grafické rozhraní a řídící logiku. Model zpracovává data a komunikuje s databází. Grafické rozhraní ve formě šablon definuje vizuální podobu zobrazované aplikace. Šablony obsahují HTML tagy, které mají vazbu na CSS styly, které společně definují grafickou podobu a využíván je šablonovací sytém Latte. Do finální vizuální podoby vstupuje na straně klienta nejen CSS, ale také JavaScript, ale o něm až dále, aby v přehledu použitých nástrojů a jazyků nezanikl smysl MVC architektury. Poslední část – controller – v případě Nette reprezentují presentery, které mají za úkol utvářet vazby mezi modelem a šablonami. Právě presentery definují, co se bude dít po volání jednotlivých URL adres. Dokonce i v případě, pokud je volána neexistující stránka dané domény, nebo dojde k výskytu chyby při vykonávání kódu. Jako pomůcka pro programátory slouží prefix jednotlivých funkcí. Standardně se využívá prefix *render*. Například funkce *public function renderChart($id, $from, $to)* z presenteru *DataPresenter* (app/presenters/DataPresenter.php) je přístupná z URL adresy *DOMÉNA/data/chart* (obecně záleží na definovaném způsobu

routování). Funkce *renderChart()* přebírá až tři vstupní parametry a výsledek je vykreslen do šablony *chart.latte* ve složce *app/teplates/Data/chart.latte* která je vložena do neproměnné části souboru *@layout.latte*, která obsahuje statický obsah vzhledem k danému presenteru. Jde o celou HTML stránku mimo obsah, který je vzhledem k podstránkám presenteru neproměnný, až na drobné změny, které je možno definovat podmínkami. Smysl je takový, že při volání odkazu není nutné překreslovat celou webovou stránku, ale pouze dílčí části, což je rychlejší a nepůsobí rušivě. Méně obvyklý je na rozdíl od prefixu *render* prefix *action*, který značí, že stránka nevykresluje data do obvyklé šablony. Jde však pouze pro pomůcku pro programátory, na funkci nemá vliv. O výstupu je rozhodnuto až v samotné funkci. V presenteru může být jedna funkce, která se zpravidla vkládá na samotný začátek ještě před funkci *renderDefault()* a není povinná. Tou je funkce *startup()*, která se vykonává vždy ještě před vykonáním jednotlivých volaných funkcí. Lze využít pro cokoliv, co je potřebné vykonat pokaždé. Hodí se tak nejlépe k ověření uživatele z cookie session.

Presenter přejímá případná data HTTP metod, volá funkce modelu a data potřebná k zobrazení klientovy předá do šablony. Šablona se postará o finální podobu vykreslení na straně serveru. Následně jsou data zaslaná klientovy (webový prohlížeč), který je příjme. Zároveň si stáhne další soubory deklarované v hlavičce stránky. Jde o již zmíněné soubory kaskádových stylů CSS a soubory skriptovacího jazyka JavaScript. Klient zohlední tyto závislosti a vykreslí stránku ve své finální podobě. Zatímco PHP po zaslání dotazu na server pošle klientovy odpověď, ukončí spojení a smaže pracovní data, JavaScript „je živý“ po celou dobu zobrazení stránky v prohlížeči. Může tak případně reagovat na obsluhu uživatele bez zaslání dotazu na server změnou grafické podoby stránky. Nebo například na pozadí komunikovat se serverem, se kterým si bude vyměňovat data bez znovunačtení samotné webové stránky. Například pokud stránka obsahuje chat.

Kromě souborů pro Composer se v kořenové složce webové aplikace nacházejí jednotlivé podsložky. Složka vendor obsahuje soubory frameworku Nette a případně další rozšiřující knihovny. Nejzajímavějšími složkami z uživatelského pohledu programátora dané aplikace jsou složky *www* a *app*. Do složky *www* se umisťují veřejně přístupné soubory. Jde typicky o obrázky, soubory kaskádových stylů CSS a Javascript, které bohužel musí být také přístupné, protože rovněž patří mezi soubory čtené klientem. Složka *app* obsahuje všechny soubory našeho projektu z MVC architektury. Ty nejsou veřejně přístupné. Server vrací pouze zpracovaný výsledek ve formě HTML stránky.

Složka *temp* v kořenovém adresáři obsahuje cache celé aplikace. Na straně serveru se vytvoří pouze jednou. Po aktualizaci souborů přes FTP je nutné smazat složku *temp,* aby se cache znovu vytvořily a provedené změny projevily. Složka *log* obsahuje záznamy o chybách a výjimkách (exceptions) na serveru. Pokud dojde k chybě na testovacím serveru (PC), nezaznamenají se informace o chybách do složky *log,* ale jsou odeslány přímo klientovy. K tomuto účelu Nette obsahuje velice užitečnou součást – Laděnku (Tracy).

K HTML stránkám patří také vizuální podoba, která v dnešní době vyžaduje responzivní web zaručující vhodné zobrazení stránky na různých zařízeních s rozměrově odlišnými displeji. Jelikož nejsem grafik a grafický návrh zabere spoustu času a ještě více, pokud má být web responzivní, rozhodl jsem se použít volně dostupnou HTML šablonu s názvem *Picollo* pocházející od Nathana Browna z texaského Austinu. Šablona využívá oblíbenou JavaScriptovou knihovnu *jQuery*. V Nette šablona vyplňuje především stránky *@layout.latte* a a částečně i šablony jednotlivých stránek prostřednictvím HTML tagů s vazbou na kaskádové styly CSS.

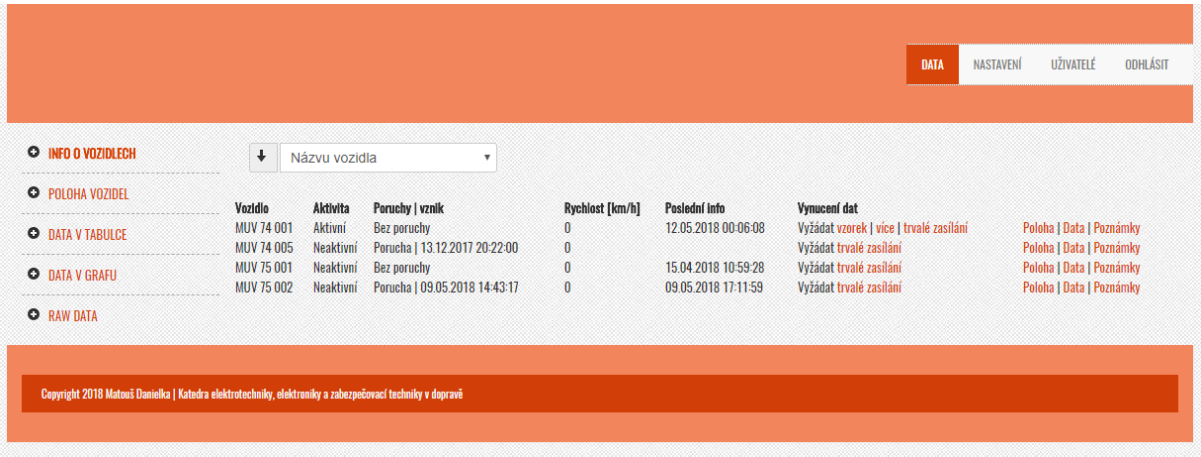
7.1 Webová aplikace především z uživatelského pohledu

Webová aplikace obsahuje řadu stránek, které jsou si často svými prvky blízké. Proto podobu stránek formou průvodce rozeberu z uživatelského pohledu a v dalších kapitolách „vypíchnu“ technické řešení jednotlivých prvků a podrobněji rozeberu technicky zajímavější části.

Po načtení domovské stránky webu se v případě návštěvy nepřihlášeného uživatele zobrazí přihlašovací formulář. Je-li uživatel přihlášený, je rovnou přesměrován dále. Přihlášení je nezbytnou podmínkou pro zobrazení dalšího obsahu webových stránek. Po přihlášení je uživatel přesměrován na výpis vozidel se základními informacemi o vozidlech. Tato stránka zároveň do určité míry slouží jako rozcestník k jednotlivým vozidlům. Pokud uživatel vystupuje s rolí administrátora, zobrazí se mu stránka *Info o vozidlech*.

Vystupuje-li jako běžný uživatel, zobrazí se mu stránka *Info o vozidlech* také, ale v souvislosti s omezenými uživatelskými pravomocemi se nebudou v pravém hlavním menu nahoře zobrazovat položky *Nastavení* a *Uživatelé*, kde je možné měnit přijímaná data, mazat přijatá data. Dále přidávat, měnit a mazat uživatelské účty. Položky pravého menu, pokud vyloučíme možnost odhlášení, jsou zároveň rozděleny do jednotlivých presenterů, které mají vlastní šablonu *@layout.latte* s odlišným menu po levé straně stránky.

7.1.1 Stránka Info o vozidlech



Obrázek 25 Webová stránka s výpisem vozidel

Stránka *Info o vozidlech* zobrazuje všechna vozidla do počtu 20 vozidel. Při překročení tohoto objemu dochází k aktivaci stránkování a mezi vozidly lze pak listovat. Formulářové pole nad výpisem vozidel definuje parametr, podle kterého jsou vozidla seřazena. Na výběr je řazení podle názvu vozidel, poslední aktivity, podla času přidání a času vzniku poruchy po předchozím bezporuchovém stavu.

Neuběhlo-li od poslední komunikace s vozidlem více než pět minut, pak je vozidlo ve výpisu označeno jako aktivní a v takovém případě se také zobrazuje nenulová rychlost jízdy, pokud bylo vozidlo současně s poslední informaci v pohybu. Čas přijetí poslední informace poskytuje sloupec *Poslední info*. Informace o poruše uvádí přítomnost poruchy pohonu vozidla a čas jejího vzniku. Jde vždy o čas výskytu první poruchy po anulaci té předešlé. Vysvětlení filozofie anulace poruch je uvedeno o několik řádků níže. Sloupec *Vynucení dat* nabízí vždy možnost trvalého zasílání dat z daného vozidla na server. Je-li vozidlo aktivní, pak jsou u vozidla zobrazeny další dvě možnosti, v rámci kterých je možné zaslat pouze jeden vzorek dat, nebo po sobě jdoucí skupinu více vzorků.

Údaje na stránce *Info o vozidlech* se aktualizují každých 10 sekund. Vozidlo data na server posílá po 10 sekundách od přijetí odpovědi od předchozího zaslání dat na server, což reálně představuje čas do 15 sekund, během kterého se vozidlo na serveru opět zahlásí s aktuálními informacemi. Výjimkou je stav po detekci příznaku závady během čekání na dokončení ukládání do externí paměti SRAM. Vozidlo se v takovém případě odmlčí na přibližně 3,5 minuty, v závislosti na množství ukládaných CAN zpráv a velikosti externí paměti SRAM. Následně bez prodlení mezi zasíláním jednotlivých dvojic vzorků setrvale odesílá data až do kompletního odeslání obsahu externí paměti SRAM.

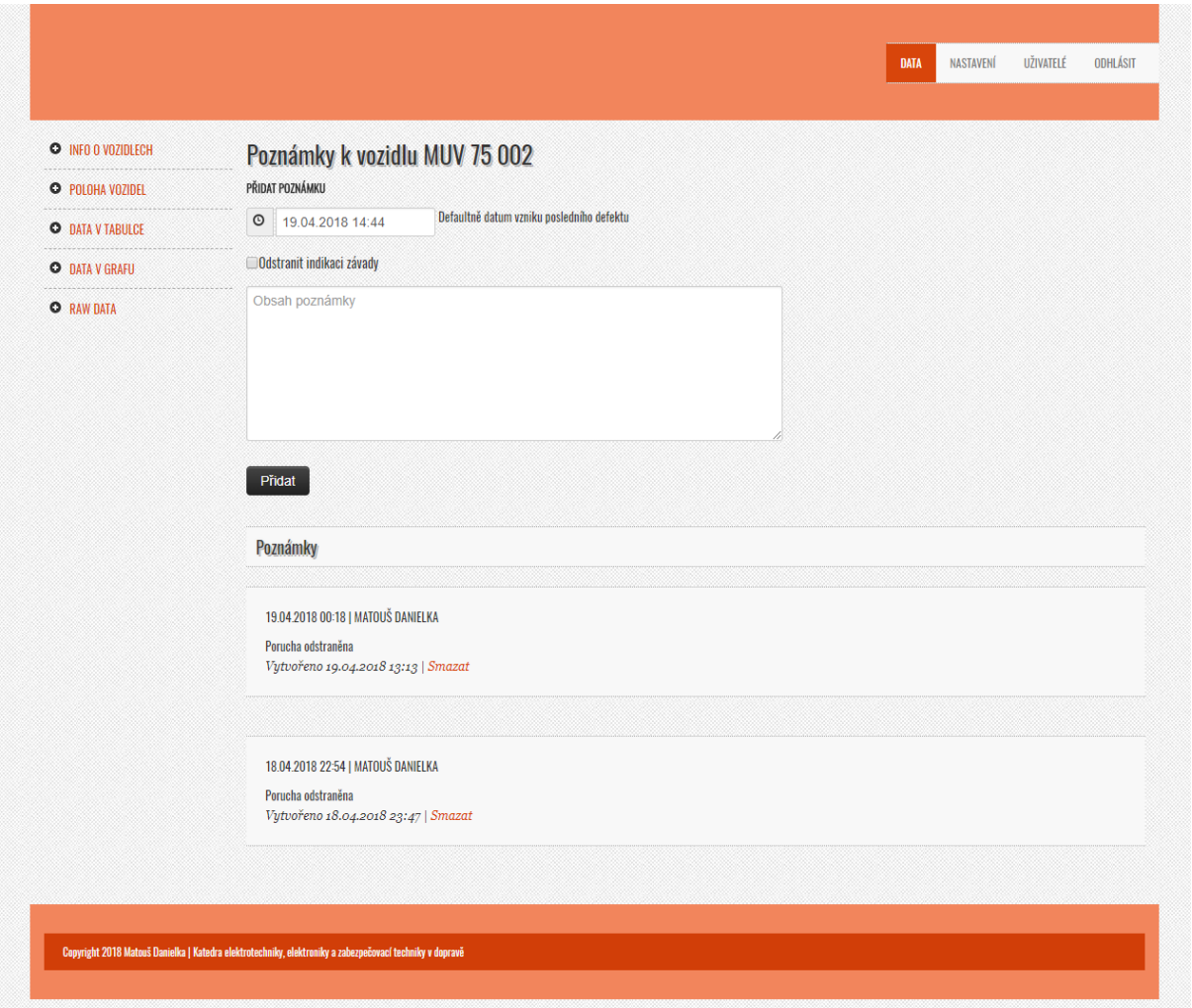
Na základě volby *Vyžádat vzorek* na stránce *Info o vozidlech* odešle aktivní vozidlo na server jeden (aktuální) vzorek dat. Informace u vozidla v pravém sloupci tabulky se přepne po zaznamenání požadavku do databáze MySQL. Nově je zobrazována informace *Vyžádán vzorek.* Spolu s touto informací je zobrazena možnost *Zrušit zaslání*, která zůstává zobrazena do doby přijetí kýženého vzorku dat. Následně se menu vrací do původní podoby s nabídkou tří možností odesílání dat. Zrušení zaslání dat vyvolá stejnou změnu v menu, ale není zaručeno, že vzorek skutečně nebude odeslán. Vozidlo si již mohlo požadavek vyzvednout v odpovědi na poslední data, která odeslalo na server.

Další volba *Odeslat více* vyvolá odeslání posledních 60 vzorků z externí paměti SRAM. Po nastavení požadavku na odeslání dat se rovněž u vozidla změní obsah posledního sloupce. Zobrazována je informace *Vyžádán balík dat* a opět s možností zrušení zaslání. Po odeslání první dvojice vzorků dat z vozidla na server je potvrzeno spuštění zasílání 60 vzorků dat a menu se vrací do původní podoby, avšak další data se ještě v následujícím čase budou z vozidla na server zasílat do kompletního odeslání všech 60 vzorků.

Volba *Vyžádat trvalé zasílání*, která je dostupná i po dobu neaktivity vozidla, aktivuje trvalé zasílání CAN zpráv z vozidla na server. Data jsou zasílána s prodlevou 10 sekund od přijetí odpovědi ze serveru v reakci na odeslání předcházejících dat, stejně jako v defaultním režimu bez vyžádaných dat. Po aktivaci tvrvalého zasílání se u vozidla ve sloupci *Vynucení dat* zobrazuje informace *Trvalé zasílání dat* s možností zrušení zasílání. Tato volba se automaticky nijak neruší, pouze je přerušována v reakci na výskyt příznaku závady řídícího systému trakčního pohonu. Po odeslání dat z vozidla na server v režimu závady se opět testuje přítomnost příznaku závady a pokud se již závada nevyskytuje, pak se vozidlo vrací k trvalému periodickému zasílání dat.

Poslední pravý sloupec na straně *Info o vozidlech* odkazuje na další stránky. Kliknutím na odkaz *Poloha* dojde k zobrazení polohy vozidla na mapě. Prostřední volba *Data* přesměruje na stránku *Data v tabulce* a již zobrazí data pro konkrétní vozidlo v defaultně zadaném časovém intervalu, který je týden do minulosti a hodina do budoucnosti. Poslední volba *Poznámky* je pak přístupná pouze z této nabídky a odkazuje na stránku, kde lze dlouhodobě vést dokumentaci závad a oprav ke konkrétnímu vozidlu a resetovat stav vozidla na „bez poruchy“ (kvitování poruchy).

7.1.2 Stránka poznámky k vozidlu



Obrázek 26 Webová stránka s dokumentací k vozidlu

Na stránce *Poznámky* lze dlouhodobě vést dokumentaci závad a oprav ke konkrétnímu vozidlu a resetovat stav vozidla na „bez poruchy“ (kvitování poruchy). Formulář s přidáním poznámky má standardně dvě položky – obsah pozvánky a datum, ke kterému se váže. Podle tohoto údaje jsou zároveň seřazeny jednotlivé poznámky na stránce pod formulářem, u kterých je pak uvedeno také datum jejich vytvoření a její autor. Tyto informace se k poznámce doplňují implicitně po odeslání webového formuláře. Při větším počtu poznámek než 10 dochází k aktivaci stránkování.

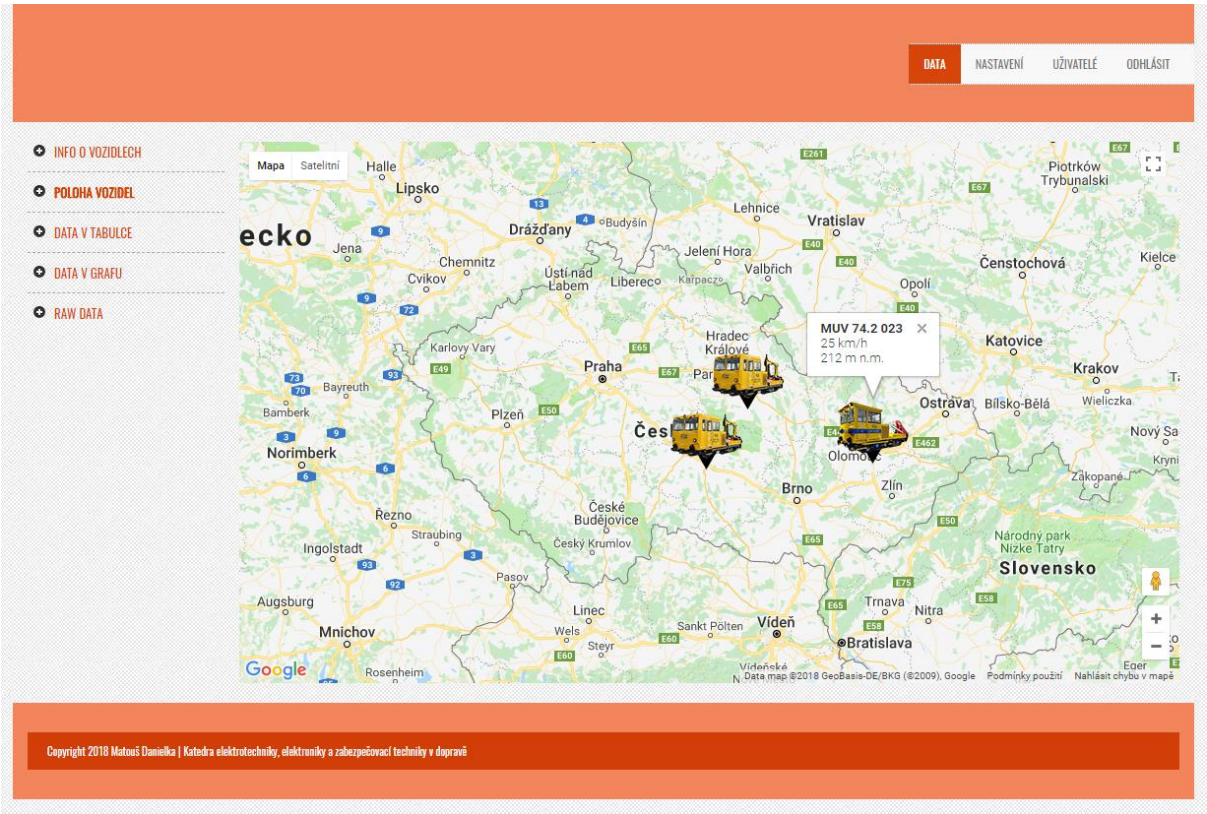
Pokud se u vozidla nevyskytla závada, pak se položka s časovým údajem ve formuláři defaultně nastaví na aktuální čas. Vyskytla-li se závada, pak defaultní hodnota představuje čas výskytu prvního příznaku poruchy trakčního pohonu do doby anulace závady. Informace o této skutečnosti je navíc doplněna informativním popisem napravo od pole s časovým údajem, jako

je zobrazeno na obrázku výše. Odstranit indikaci závady lze provést zaškrtnutím checkboxu s popisem *Odstranit indikaci závady*.

Toto řešení jako celek vychází z následující filozofie – po zaznamenání závady obsluha zjistí její souvislosti, zajistí opravu a po provedení opravy k vozidlu uloží záznam s podrobnostmi a zároveň odstraní indikaci závady. Pokud z nějakého důvodu není tento postup vhodný, není nutné se ho držet. Po nabytých zkušenostech z praktického chování vozidla se jeví jako lepší varianta vytvoření stránky s historií závad (pro každé vozidlo), do které by se automaticky generovaly závady, bez ruční anulace závad, avšak v rámci řešení se nabízí celá řada možností uchopení implementační logiky.

7.1.3 Stránka Poloha vozidel

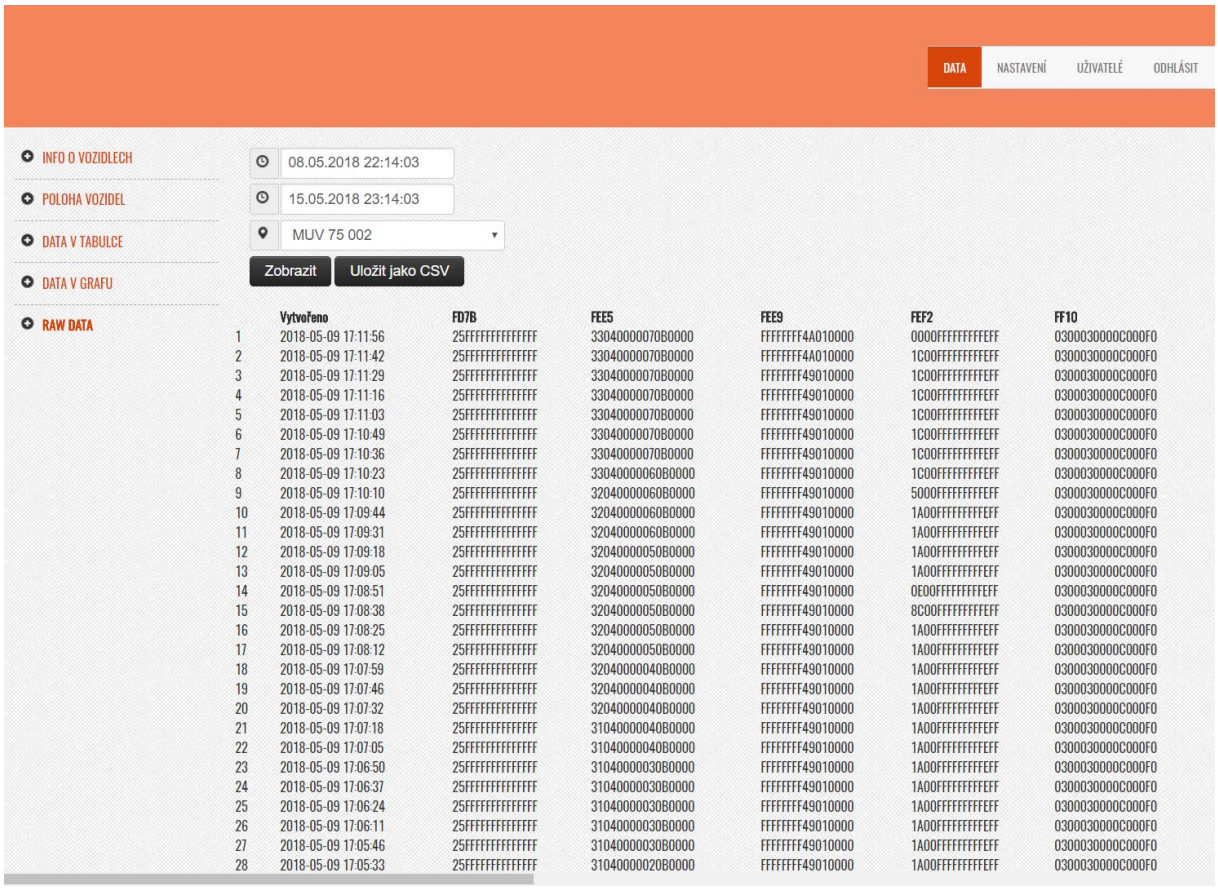
Na stránce *Poloha vozidel* je zobrazena mapaGoogle se zobrazenou mapou České republiky a přilehlým okolím. V mapě jsou zaneseny body s ikonkami vozidel. Po kliknutí na vozidlo se zobrazí jeho název a informace o jeho aktivitě. Je-li vozidlo aktivní dle výše definované charakteristiky, je navíc zobrazována poslední zaznamenaná rychlost jízdy a nadmořská výška. V případě využití přímého odkazu *Poloha* u konkrétních vozidel ze stránky *Info o vozidlech* se zobrazí přiblížená mapa zachycující detailnější polohu vozidla. V obou případech jsou však na mapě zakreslena všechna vozidla, po mapě se lze standardně pohybovat a přepínat mezi základní a satelitní fotomapou. Mapa je aktualizována každých 10 sekund.



Obrázek 27 Mapa s polohou vozidel

7.1.4 Stránky s tabulkami dat

Vzhledem k lepší tematické návaznosti v levém menu přeskočíme na stránku *Raw data*. Zde jsou zobrazena surová CAN data ve standardním hexadecimálním formátu s délkou dat 8 bajtů. Před zobrazením je nutné vybrat vozidlo a časový interval, ze kterého mají data pocházet. Data je možné nejen zobrazit na stránce, ale také vyexportovat do formátu CSV. Vzhledem k množství CAN zpráv se záznamy nevejdou na standardní šířku stránky, nicméně jsou zobrazena v celkem přehledné formě. Z důvodu větší variability zde nebylo stránkování žádným způsobem aplikováno, avšak podle potřeby lze měnit časový interval zpráv podle času vzniku datových rámců (řádků) a tím lze nepřímo omezovat množství zobrazených záznamů na stránce podle potřeby.



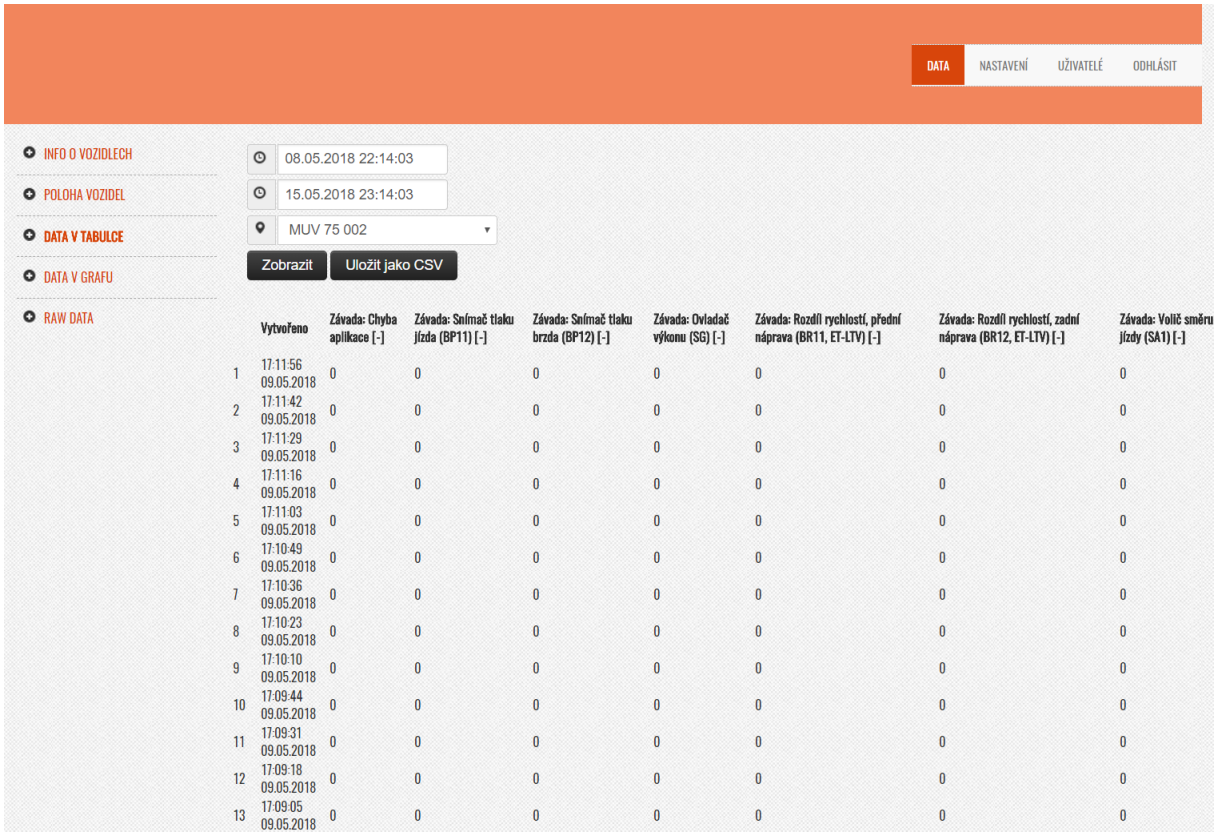
	Vytvořeno	FD7B	FEE5	FEE9	FEF2	FF10
1	2018-05-09 17:11:56	25FFFFFFFFFFFF	33040000070B0000	FFFFFFFF4A010000	0000FFFFFFFFEFF	0300030000C000F0
2	2018-05-09 17:11:42	25FFFFFFFFFFFF	33040000070B0000	FFFFFFFF4A010000	1C00FFFFFFFFEFF	0300030000C000F0
3	2018-05-09 17:11:29	25FFFFFFFFFFFF	33040000070B0000	FFFFFFFF49010000	1C00FFFFFFFFEFF	0300030000C000F0
4	2018-05-09 17:11:16	25FFFFFFFFFFFF	33040000070B0000	FFFFFFFF49010000	1C00FFFFFFFFEFF	0300030000C000F0
5	2018-05-09 17:11:03	25FFFFFFFFFFFF	33040000070B0000	FFFFFFFF49010000	1C00FFFFFFFFEFF	0300030000C000F0
6	2018-05-09 17:10:49	25FFFFFFFFFFFF	33040000070B0000	FFFFFFFF49010000	1C00FFFFFFFFEFF	0300030000C000F0
7	2018-05-09 17:10:36	25FFFFFFFFFFFF	33040000070B0000	FFFFFFFF49010000	1C00FFFFFFFFEFF	0300030000C000F0
8	2018-05-09 17:10:23	25FFFFFFFFFFFF	33040000060B0000	FFFFFFFF49010000	1C00FFFFFFFFEFF	0300030000C000F0
9	2018-05-09 17:10:10	25FFFFFFFFFFFF	32040000060B0000	FFFFFFFF49010000	5000FFFFFFFFEFF	0300030000C000F0
10	2018-05-09 17:09:44	25FFFFFFFFFFFF	32040000060B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
11	2018-05-09 17:09:31	25FFFFFFFFFFFF	32040000060B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
12	2018-05-09 17:09:18	25FFFFFFFFFFFF	32040000050B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
13	2018-05-09 17:09:05	25FFFFFFFFFFFF	32040000050B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
14	2018-05-09 17:08:51	25FFFFFFFFFFFF	32040000050B0000	FFFFFFFF49010000	0E00FFFFFFFFEFF	0300030000C000F0
15	2018-05-09 17:08:38	25FFFFFFFFFFFF	32040000050B0000	FFFFFFFF49010000	8C00FFFFFFFFEFF	0300030000C000F0
16	2018-05-09 17:08:25	25FFFFFFFFFFFF	32040000050B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
17	2018-05-09 17:08:12	25FFFFFFFFFFFF	32040000050B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
18	2018-05-09 17:07:59	25FFFFFFFFFFFF	32040000040B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
19	2018-05-09 17:07:46	25FFFFFFFFFFFF	32040000040B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
20	2018-05-09 17:07:32	25FFFFFFFFFFFF	32040000040B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
21	2018-05-09 17:07:18	25FFFFFFFFFFFF	31040000040B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
22	2018-05-09 17:07:05	25FFFFFFFFFFFF	31040000040B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
23	2018-05-09 17:06:50	25FFFFFFFFFFFF	31040000030B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
24	2018-05-09 17:06:37	25FFFFFFFFFFFF	31040000030B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
25	2018-05-09 17:06:24	25FFFFFFFFFFFF	31040000030B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
26	2018-05-09 17:06:11	25FFFFFFFFFFFF	31040000030B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
27	2018-05-09 17:05:46	25FFFFFFFFFFFF	31040000030B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0
28	2018-05-09 17:05:33	25FFFFFFFFFFFF	31040000020B0000	FFFFFFFF49010000	1A00FFFFFFFFEFF	0300030000C000F0

Obrázek 28 Webová stránka s výpisem CAN zpráv v surovém formátu

Do prvního sloupce tabulky jsou automaticky generovaná čísla řádků. Druhý uvádí zaznamenaný čas zpráv na straně mobilní části a dále už jsou sloupce CAN zpráv s PGN v hlavičce.

Stránka *Data v tabulce* pak zobrazuje k daným časovým záznamům hodnoty jednotlivých fyzikálních veličin, nebo jde o dvoustavové informace. Vizuální podoba stránky je totožná se stránkou se zprávami CAN.

Data ukládaná do CSV mají podstatě stejnou koncepci jako data zobrazovaná v tabulce na webu. První řádek tvoří hlavička zpráv s názvem zpráv a fyzikální jednotkou v případě signálů. U surových CAN zpráv obsahuje hlavička nejdříve popis zprávy a v závorce PGN. Hodnoty jsou v CSV oddělovány středníky. Desetinná místa odděluje tečka, tak jak hodnoty přichází z databáze včetně počtu desetinných míst.



	Vytvořeno	Závada: Chyba aplikace [-]	Závada: Snímač tlaku jízda (BP11) [-]	Závada: Snímač tlaku brzda (BP12) [-]	Závada: Ovladač výkonu (SG) [-]	Závada: Rozdíl rychlostí, přední náprava (BR11, ET-LTV) [-]	Závada: Rozdíl rychlostí, zadní náprava (BR12, ET-LTV) [-]	Závada: Volič směru jízdy (SA1) [-]
1	17:11:56 09.05.2018	0	0	0	0	0	0	0
2	17:11:42 09.05.2018	0	0	0	0	0	0	0
3	17:11:29 09.05.2018	0	0	0	0	0	0	0
4	17:11:16 09.05.2018	0	0	0	0	0	0	0
5	17:11:03 09.05.2018	0	0	0	0	0	0	0
6	17:10:49 09.05.2018	0	0	0	0	0	0	0
7	17:10:36 09.05.2018	0	0	0	0	0	0	0
8	17:10:23 09.05.2018	0	0	0	0	0	0	0
9	17:10:10 09.05.2018	0	0	0	0	0	0	0
10	17:09:44 09.05.2018	0	0	0	0	0	0	0
11	17:09:31 09.05.2018	0	0	0	0	0	0	0
12	17:09:18 09.05.2018	0	0	0	0	0	0	0
13	17:09:05 09.05.2018	0	0	0	0	0	0	0

Obrázek 29 Webová stránka s výpisem signálů

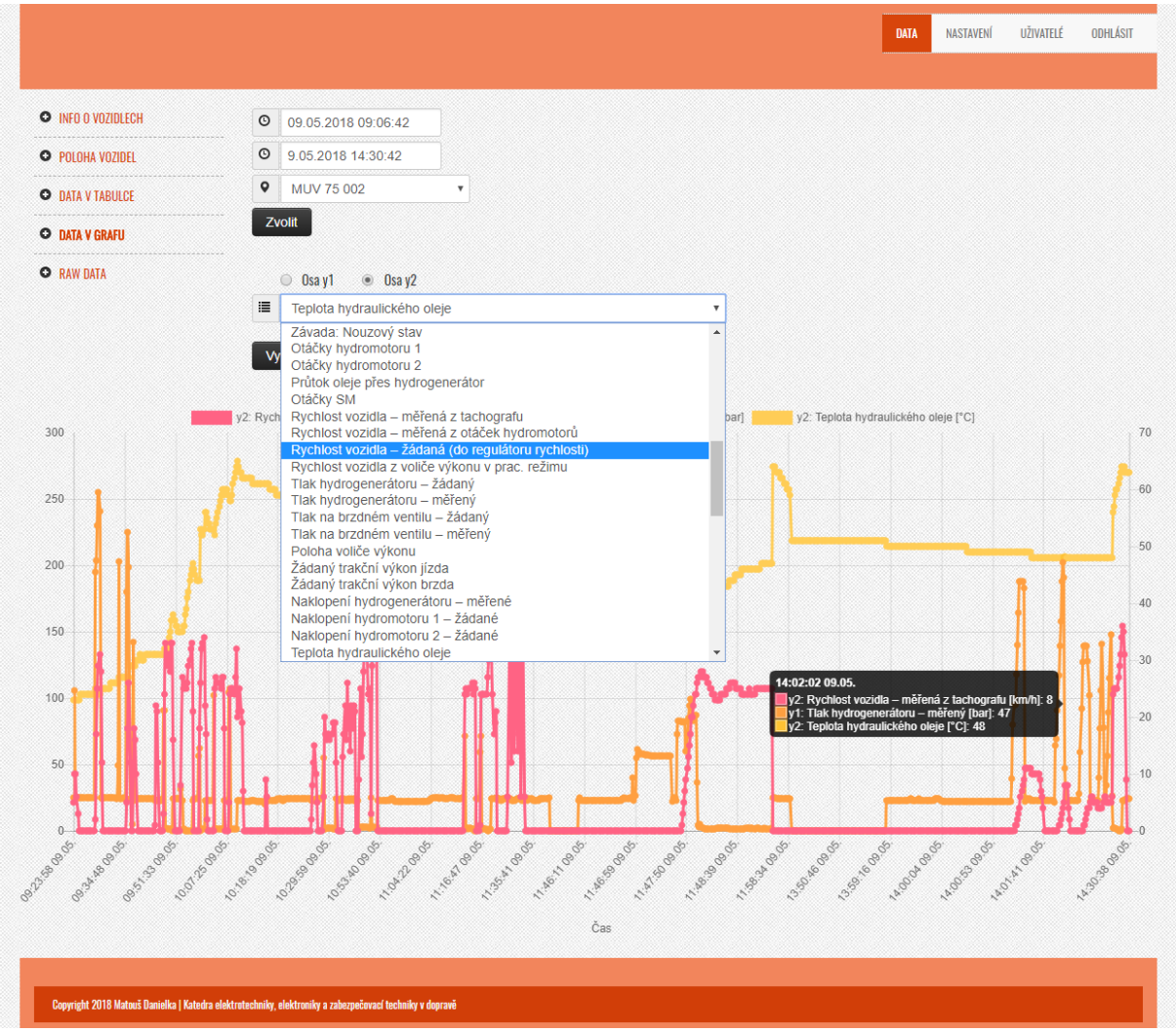
Závada: Nouzový stav [-]	Otáčky hydromotoru 1 [1/min]	Otáčky hydromotoru 2 [1/min]	Průtok oleje přes hydrogenerátor [lit/min]	Otáčky SM [1/min]	Rychlost vozidla – měřená z tachografu [km/h]	Rychlost vozidla – měřená z otáček hydromotorů [km/h]
0	0	0	0	0	0	0
0	0	0	5	800	0	0
0	0	0	5	799	0	0
0	0	0	5	799	0	0
0	0	0	5	799	0	0
0	41	41	5	799	1	1
0	191	191	5	798	5	5
0	375	366	5	800	9	9
0	175	175	64	919	4	4
0	0	0	5	799	0	0
0	0	0	5	800	0	0
0	358	366	5	800	9	9
0	491	500	5	800	13	13
0	525	525	9	843	13	13

Obrázek 30 Webová stránka s výpisem signálů v detailnějším pohledu

7.1.5 Stránka Data v grafu

Obdobný formulářový přístup přináší také stránka *Data v grafu*. Zde již nefiguruje možnost ukládání do CSV, protože by tato možnost byla duplicitní se stránkou *Data v tabulce*. Po vyplnění časového intervalu dat a volbě vozidla potvrzené kliknutím Zvolit se zobrazí pole grafu s časovou osou, pokud existují k vozidlu v daném časovém intervalu záznamy, jinak se zobrazí poznámka informující o opačné skutečnosti.

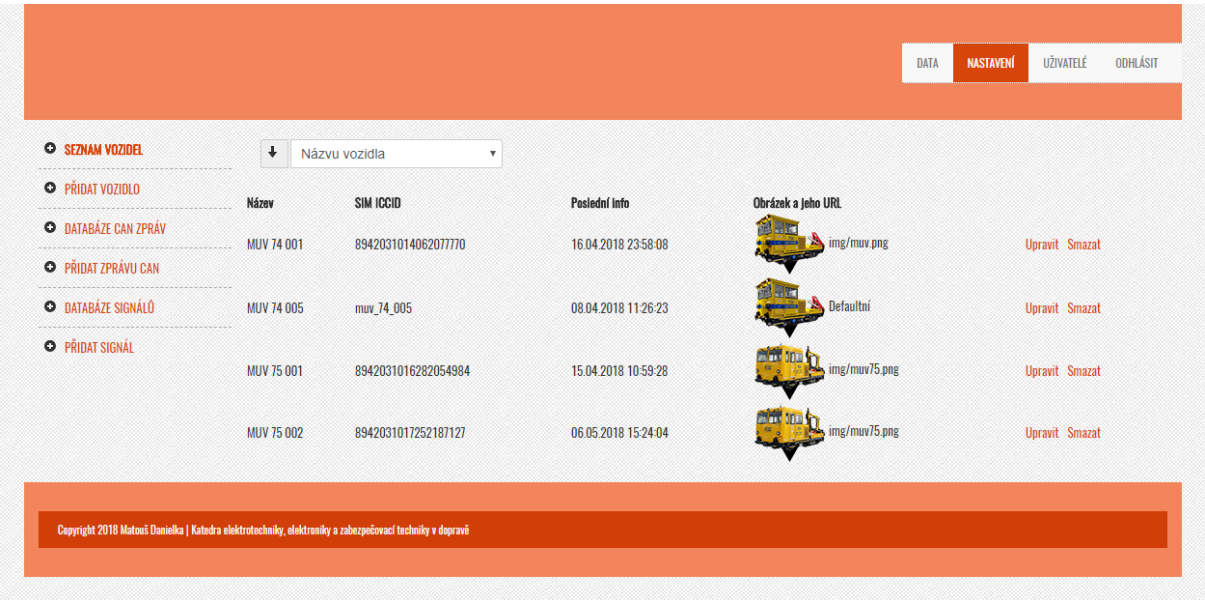
Se zobrazením pole grafu se pod dosavadním formulářem zobrazí nabídka s volbou vertikální osy a výběrem signálu, který se po vybrání okamžitě promítají do grafu. Rozsah osy Y je volen automaticky podle největší/nejmenší hodnoty v rámci všech zobrazovaných průběhů pro danou osu Y. Jednotlivé signály lze do grafu postupně přidávat a v opačném sledu odmazávat. Při pohybu kurzoru nad grafem ve směru časové osy se zobrazují konkrétní hodnoty signálů pro daný časový okamžik.



Obrázek 31 Webová stránka s grafem

7.1.6 Skupina stránek Nastavení

Záložka *Nastavení* obsahuje několik stránek pro správu vozidel, CAN zpráv a signálů. U vozidel je možné jejich přidávání, úprava a mazání. Přehled vozidel na straně *Seznam vozidel* je taktéž doplněn o základní informace o stavu vozidel podobně jako v uživatelské části *Data*, ale hodnoty již nejsou periodicky obnovovány. Výpis je doplněn o seřazení vozidel podle parametrů. Zároveň je obsaženo stránkování s počtem 10 vozidel na stránku.



Obrázek 32 Webová sekce s nastavením vozidel a CAN zpráv

Pro každé nové vozidlo je třeba vyplnit dva údaje povinné a případně i jeden nepovinný. Povinným údajem je identifikátor SIM karty ICCID a dále název vozidla. Nepovinný údaj je relativní adresa obrázku. Kořenovou složkou je složka *www*, jako jediná přístupná ze strany klienta, kde jsou umístěny všechny soubory využívané na straně klienta – obrázky, CSS styly, JavaScript. Naopak sem nepatří soubory Nette Frameworku (vyjma 3 souborů) včetně našich souborů PHP (modely, presentery) i šablonovacího systému Latte, které mají zůstat skryty. Obrázky se samozřejmě mohou ukládat do složky *www* i libovolných podsložek a k nim lze nastavit cestu i pro ikonky vozidel. Není-li adresa obrázku vyplněna, automaticky se použije defaultní ikonka vozidla MUV 74 (img/muv.png). Kromě této ikonky jsem vytvořil ještě ikonku vozidla MUV 75. Nepředpokládám využití více typů vozidel, a proto také není nabízeno přímé vkládání obrázků.

Kromě přidávání vozidel je možné dané údaje upravovat kromě ICCID, které je zároveň unikátním identifikátorem vozidel v databázi. Je-li z nějakého důvodu nutné změnit SIM kartu,

je třeba vozidlo smazat a vytvořit znovu s novým ICCID. Pro případ náhodného smazání kliknutím na *Smazat*, je tato volba pojištěna nutným potvrzením smazání. Po kliknutí na *Smazat*, se objeví vyskakovací okno s potvrzením této volby. Smazaná vozidla odstraní informace o vozidlu, avšak nedojde k okamžitému odstranění dat k vozidlu v databázi. Ty pak postupně odmazává cron spolu s ostatními starými daty ostatních vozidel.

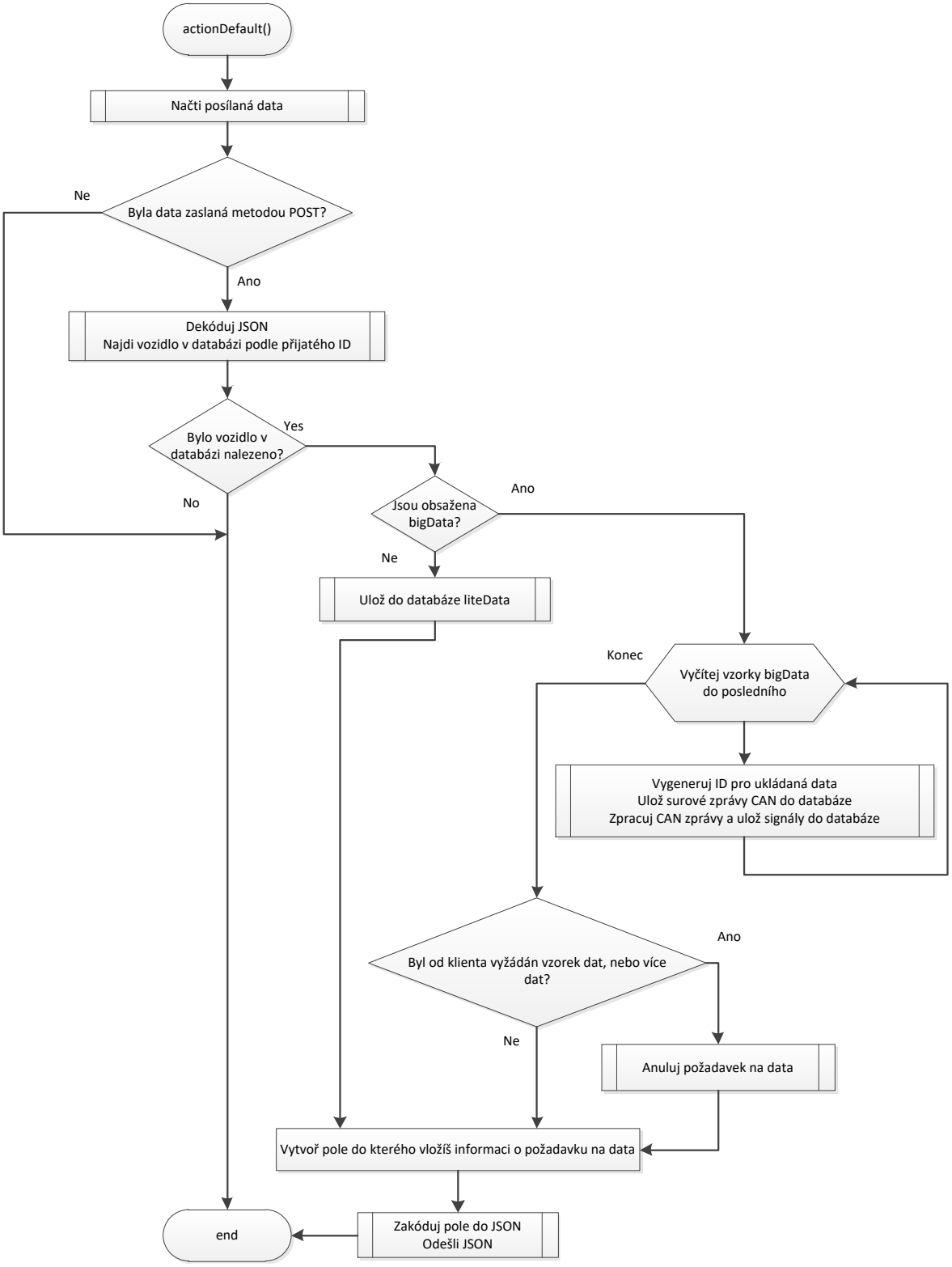
Další stránky v kategorii Nastavení spravují CAN zprávy a signály. Přidání CAN zpráv a signálů vkládá do tabulek MySQL nejen řádky s těmito informacemi (types_of_can_messages a types_of_processed_messages), ale také sloupce do jednotlivých tabulek dat (can_raw_data, processed_data). Při mazání zpráv a signálů jsou analogicky odstraňovány i sloupce těchto tabulek.

Další stránky již neobsahují příliš zajímavého, a tak je jejich popis vynechán.

7.2 Příjem dat z mobilní části – REST API

V části popisující volbu přenosového protokolu a formátu dat byla popsána podoba dat odesílaných z vozidla na server. Teď se podíváme na to, co se s nimi děje na straně serveru. Pro příjem dat jsem vytvořil *ImportPresenter*, který má jedinou funkci *actionDefault()* pro příjem dat. Navíc ještě obsahuje funkci *actionCron(),* jejíž úkolem je mazání starých dat z databáze se záznamy staršími jednoho měsíce. Server volá tuto funkci každý den ve 3:00, protože s velkou zátěží se obvykle přesouvají do časových poloh, kdy servery nezatěžují samotní uživatelé.

Podíváme se ale podrobněji na funkci *actionDefault()*. Nejdříve ověřuje vstupní data s ohledem na metodu příjmu dat (POST) a platnost ID vozidla a následně volá funkce ze třídy *MainModel*, která dále zpracovává data. Po zpracování dat na straně serveru se do vozidla vrací odpověď s velice krátkou informací ve formátu JSON, ve které je uvedeno, zda klient vyžaduje zaslání dat v jedné ze třech podob. V tomto případě presenter neposílá data pro finální vykreslení do šablony, ale volá metodu, která se postará o zaslání JSON dat zpět klientovi místo vykreslované šablony.



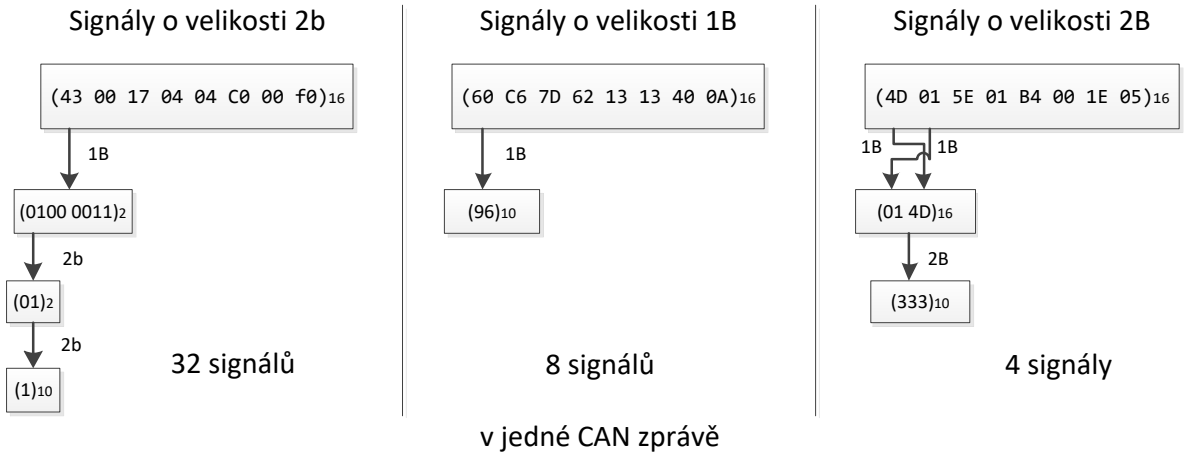
Vývojový diagram 9 Funkce ImportPresenter:: actionDefault()

Z volaných funkcí je nejzajímavější funkce *processData()*, která převádí data z CAN zpráv do jednotlivých signálů definovaných fyzikální jednotkou, nebo na dvoustavové veličiny.

7.3 Funkce processData()

Funkce realizuje překlad CAN zpráv na jednotlivé signály podle PGN. Zprávy CAN jsou podle komunikačního protokolu v dokumentu [1] převedeny na jednotlivé signály. Na serveru jsou data přijata ve formátu JSON. Nette obsahuje třídu pro manipulaci s formátem JSON, která nám z formátu JSON vytvoří pole a podle formátu JSON se k jednotlivým položkám přistoupí názvem klíče a nebo indexem[]. Do funkce *processData()* už vstupuje pole, nikoliv JSON a pouze jeden vzorek dat.

Samotný princip zpracování dat je nastíněn v následujícím obrázku. Na vstupu je vždy 8 bajtů dat v hexadecimálním formátu, které jsou postupně převáděny do výsledné podoby v desítkové soustavě. Dvoubajtová informace má podle protokolu J1939 zleva nejdříve spodní bajt a poté horní. Pro převod do desítkové soustavy je tak nutné dané dva bajty prohodit.



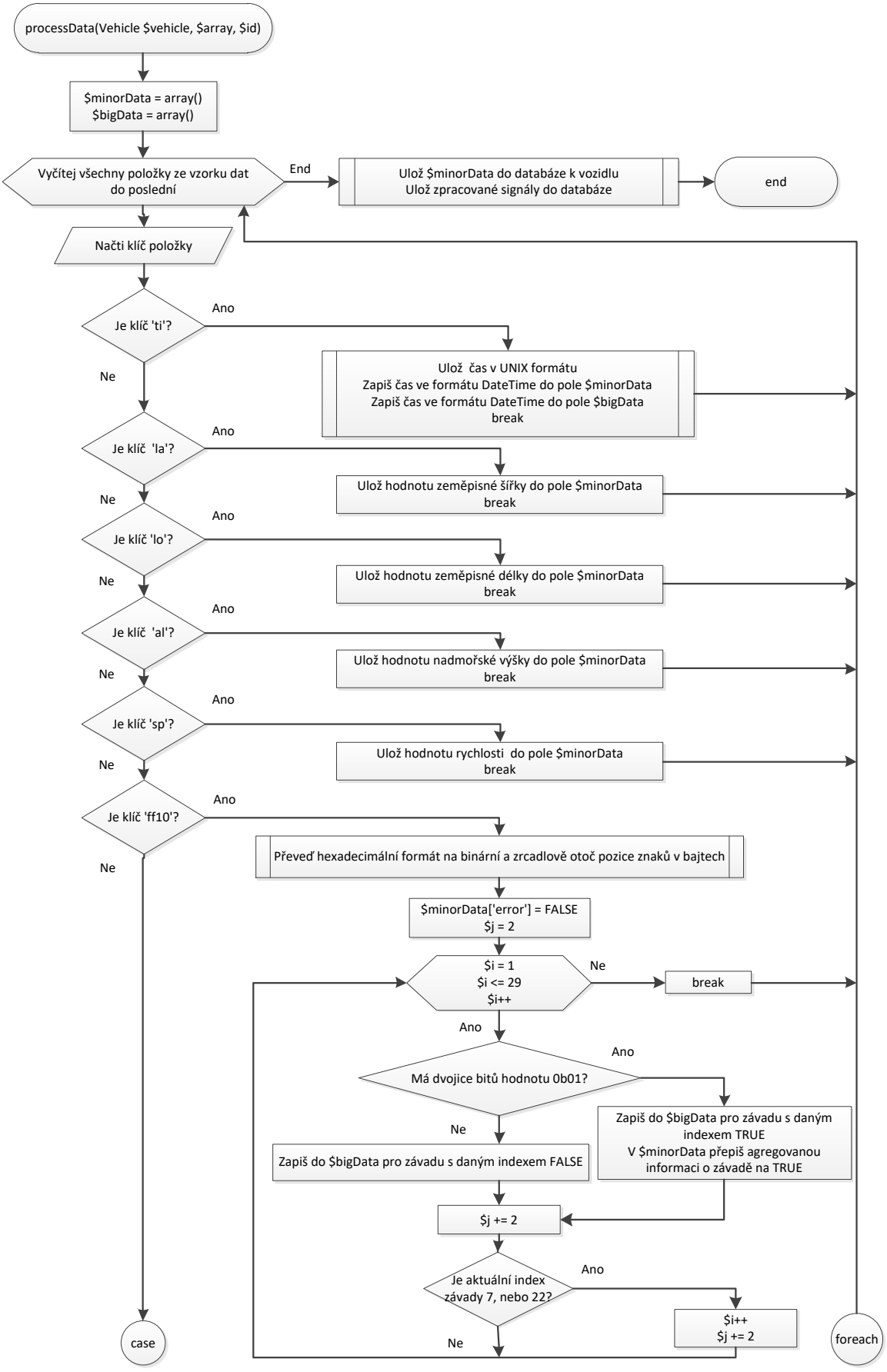
Obrázek 33 Převod CAN zprávy na signály o velikosti 2b, 1B a 2B

Převodem do desítkové soustavy ještě nemusí být získána reálná fyzikální hodnota. Získaná *RAW* hodnota je na fyzikální hodnotu *FYZ* převedena podle následujícího vztahu s parametry přepočtu *resolution* a *offset*.

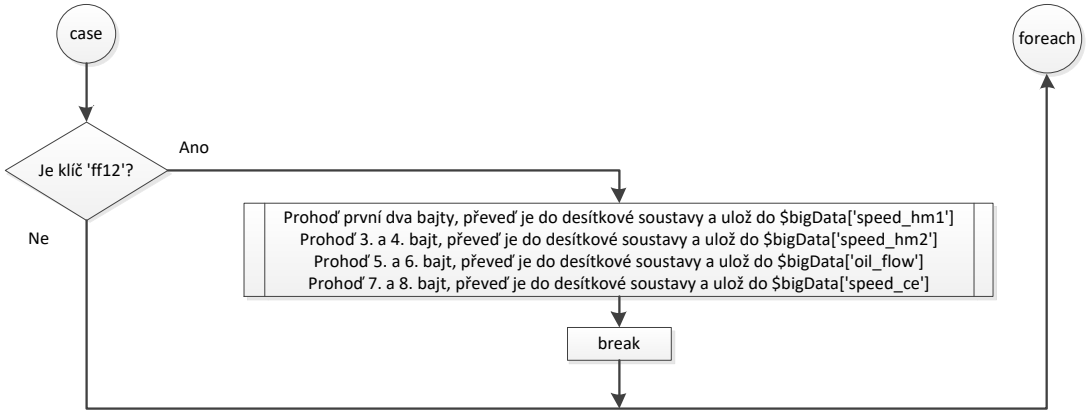
$$FYZ = RAW * resolution + offset \tag{7.1}$$

Dvoustavové informace jsou protokolem J1939 standardně přenášeny pomocí dvou bitů a každé z hodnot odpovídá konkrétní stav:

- 0b00 – příznak závady nenahozen
- 0b01 – příznak závady nahozen
- 0b10 – error
- 0b11 – not available



Vývojový diagram 10 Funkce MainModel::processData() 1. část



Vývojový diagram 11 Funkce MainModel::processData() 2. část

Funkce *processData()* využívá tři další pomocné funkce, jejichž vývojové diagramy jsou dostupné v přílohách. Do těchto funkcí byly vloženy části kódu, které by se jinak ve funkci *processData()* opakovaly. Funkce *formatConversionToBin()* převádí hexadecimální číslo v řetězci na číslo ve dvojkové soustavě. Funkce *formatConversionToBinAndTurn()* plní stejnou úlohu, ale navíc otáčí pozice bitů v jednotlivých bajtech, pro jednodušší zpracování v dalším kroku. Využity jsou u zpráv, ve kterých jsou přenášeny informace s délkou dvou bitů. Funkce *threeStatesConversion()* pak převádí dvoubitovou hodnotu na v zásadě dvoustavovou informaci. Avšak v případě chyby ve sledovaném zařízení bude signál nabývat hodnoty 2 (0b10), což je poslední třetí stav, který naznačuje název funkce.

Do databáze MySQL lze i takovou hodnotu běžně uložit, přesto že do dané proměnné jsou běžně ukládané pouze dva stavy (0, 1). Databáze totiž proměnnou typu *boolean* nezná, resp. zná, ale nepoužívá ji jako datový typ sloupců tabulek. Nejmenší datový typ dostupný v databázi má velikost jednoho bajtu a lze mu nastavit omezení na jeden znak. Jde konkrétně o datový typ *tinyint(1)*.

Pro snadnější přidávání nových zpráv a modifikaci stávajících zpráv v seznamu (PGN, definice přenášených signálů ve zprávě, převody signálů na fyzikální rozměr) by v budoucnu bylo výhodnější vytvořit databázi zpráv a signálů uloženou v MySQL databázi na místo stávajícího řešení „na tvrdo v kódu“. Stávající řešení neumožňuje uživateli správu zpráv a signálů (tj. úpravu, přidávání, mazání zpráv a signálů). Je třeba modifikovat kód. Nové řešení by umožnilo uživateli správu zpráv a signálů skrz webové rozhraní. V ideálním případě by tak bylo možné pro všechny datové typy a pozice ve zprávě v rámci protokolu J1939 určit jejich podobu prostřednictvím webového formuláře, avšak současná podoba má výhodu rychlejšího vykonávání, které v menší míře zatěžuje server.

7.4 Testovací server ve vlastním počítači

Vytvářenou webovou aplikaci není nutné pro odzkoušení nutné nasazovat na server resp. takový server si lze nasimulovat v počítači s obvyklými prostředky na straně serveru. Nejjednodušším řešením je využít balík Xampp, který obsahuje všechny potřebné prostředky a není nutné ho před použitím dále konfigurovat, pouze ho stačí nainstalovat. Obsahuje softwarový server Apache HTTP Server a databázi MySQL v aktuální podobě MariaDB a další doplňky. Víc součástí ale není třeba. Po spuštění těchto součástí přes XAMPP Control Panel stačí soubory s obsahem webu vložit do nastavené kořenové složky serveru, případně si tuto složku změnit v konfiguračním souboru Apache. Posledním krokem je pak naplnění databáze tabulkami a konfigurace přístupu k databázi ve složce *app/config/*. Nastavení je zde asi nejvhodnější definovat pomocí tří souborů. Soubor *config.neon* obsahuje obecně platná nastavení. Soubor *config.development.neon* obsahuje nastavení pro vývojové prostředí v PC a *config.production.neon* pak pro produkční nasazení na serveru. Ve které situaci se daný soubor využije je definováno v souboru app/bootstrap.php.

```
parameters:

database:
        dsn: 'mysql:host=127.0.0.1;dbname=muv'
        user: root
        password:
        options:
               lazy: true
```

Zdrojový kód 4 Soubor app/bootstrap.php

Po úspěšném nastavení softwarového serveru už stačí jen zadat do webového prohlížeče adresu *localhost*, nebo odpovídající IP adresu 127.0.0.1 vlastního počítače a mělo by dojít k zobrazení vytvořené kořenové stránky s voláním základního presenteru *HomepagePresenter* a funkce *public function renderDefault()*, kterou by měl každý presenter obsahovat.

7.5 Obsah databáze MySQL

Databáze obsahuje celkem 7 tabulek. Tabulku uživatelů, tabulku vozidel a tabulku poznámek k vozidlům. Dále tabulku typů CAN zpráv a tabulku typů zpracovaných signálů a nakonec i tabulky se surovými CAN zprávami a zpracovanými signály.

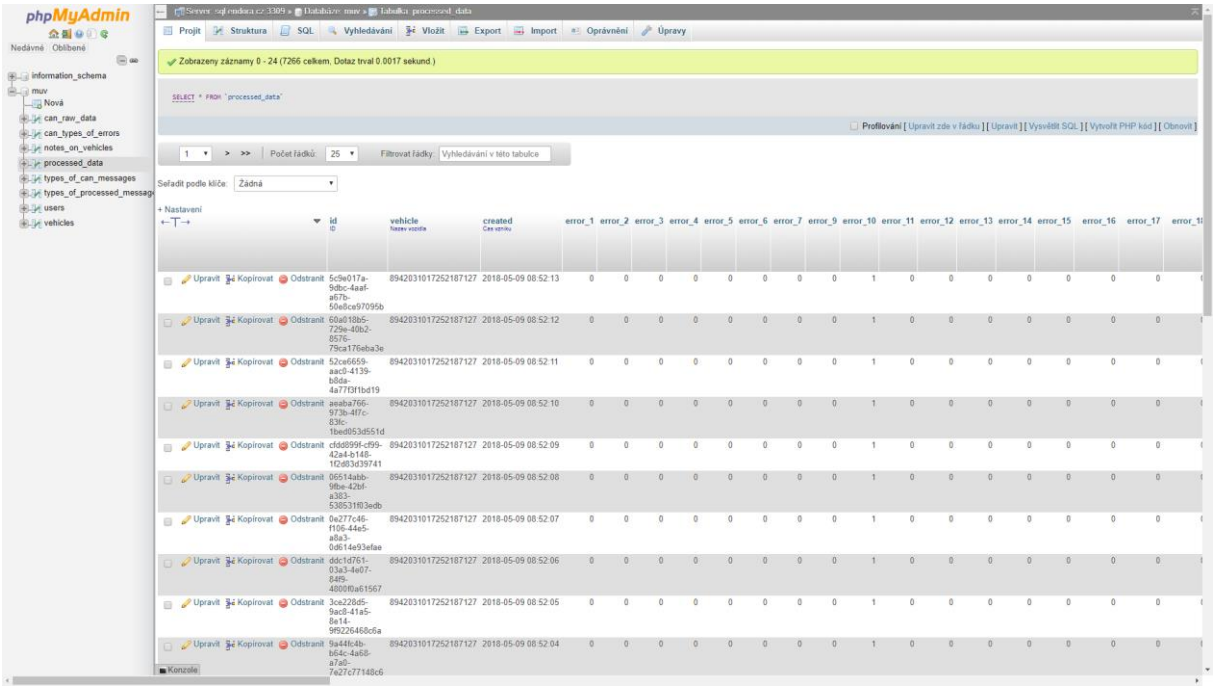
```
CREATE TABLE IF NOT EXISTS `users` (
       `id` varchar(36) NOT NULL PRIMARY KEY,
       `username` varchar(255) NOT NULL,
```

```
	`password_hash` varchar(255) NOT NULL,
	`role` varchar(255) NOT NULL,
	`first_name` varchar(255) DEFAULT NULL,
	`last_name` varchar(255) DEFAULT NULL,
	`email` varchar(255) NOT NULL,
	`phone` bigint(20) DEFAULT NULL,
	`created_on` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
	`updated_on` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP,
	`is_deleted` boolean DEFAULT NULL,
	`deleted_on` datetime DEFAULT NULL,
	UNIQUE KEY `user_email` (`email`),
	UNIQUE KEY `user_username` (`username`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `vehicles` (
	`id` varchar(255) NOT NULL PRIMARY KEY COMMENT 'ID vozidla',
	`name` varchar(255) DEFAULT NULL COMMENT 'Nazev vozidla',
	`last_info` datetime DEFAULT NULL COMMENT 'Cas prijeti posledni informace z vozidla',
	`is_defect` boolean DEFAULT 0 COMMENT 'Poukazuje na vyskyt poruchy',
	`defect_time` datetime NOT NULL COMMENT 'Cas vzniku posledni poruchy',
	`latitude` float(10,6) NOT NULL COMMENT 'Zemepisna sirka',
	`longitude` float(10,6) NOT NULL COMMENT 'Zemepisna delka',
	`altitude` int(4) NOT NULL COMMENT 'Nadmorska vyska',
	`speed` float(5,2) NOT NULL COMMENT 'Rychlost km/h',
	`image_url` varchar(255) DEFAULT NULL COMMENT 'URL ikonky',
	`are_data_requested` tinyint(1) DEFAULT 0 COMMENT 'Vynuceni odeslani dat na server',
	`created_on` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT 'Vytvoreno',
	`updated_on` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP COMMENT 'Aktualizovano'
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `notes_on_vehicles` (
	`id` varchar(36) NOT NULL PRIMARY KEY COMMENT 'ID',
	`vehicle` varchar(36) NOT NULL COMMENT 'Vehicle ID',
	`author` varchar(36) NOT NULL COMMENT 'User ID',
	`date` datetime NOT NULL COMMENT 'Cas vzniku',
	`note` text CHARACTER SET utf8 COLLATE utf8_czech_ci NOT NULL COMMENT 'Poznamka',
	`created_on` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT 'Vytvoreno',
	`updated_on` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP COMMENT 'Aktualizovano',
	CONSTRAINT `package_vehicles` FOREIGN KEY (`vehicle`) REFERENCES `vehicles` (`id`),
	CONSTRAINT `package_users` FOREIGN KEY (`author`) REFERENCES `users` (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `types_of_can_messages` (
	`name` varchar(255) NOT NULL PRIMARY KEY COMMENT 'Název zpravy',
	`title` varchar(255) NOT NULL,
	`created_on` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT 'Vytvoreno',
	`updated_on` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP COMMENT 'Aktualizovano'
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

```
CREATE TABLE IF NOT EXISTS `types_of_processed_messages` (
	`id` varchar(255) NOT NULL PRIMARY KEY COMMENT 'ID zpravy',
	`can` varchar(255) NOT NULL COMMENT 'Zdrojova CAN zprava',
	`name` varchar(255) NOT NULL COMMENT 'Nazev zpravy',
	`scale` varchar(255) NOT NULL COMMENT 'Jednotka merene veliciny',
	`created_on` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT 'Vytvoreno',
	`updated_on` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP COMMENT 'Aktualizovano'
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Zdrojový kód 5 Založení tabulek v databázi MySQL

V definici tabulek nejsou zastoupeny tabulky se surovými a zpracovanými daty (can_raw_data, processed_data). Databáze CAN zpráv v době psaní tohoto textu čítala 18 zpráv (datový typ varchar(16)), v kterých bylo zastoupeno celkem 80 signálů různých datových typů. Tabulky tak obsahují sloupce s těmito položkami a samozřejmě nutné položky jako ID záznamu, ID vozidla, čas vzniku a podobně. Zároveň je možné položky v databázi měnit s požadavkem na přidání CAN zprávy a nového signálu, případně jejich odstraňování.



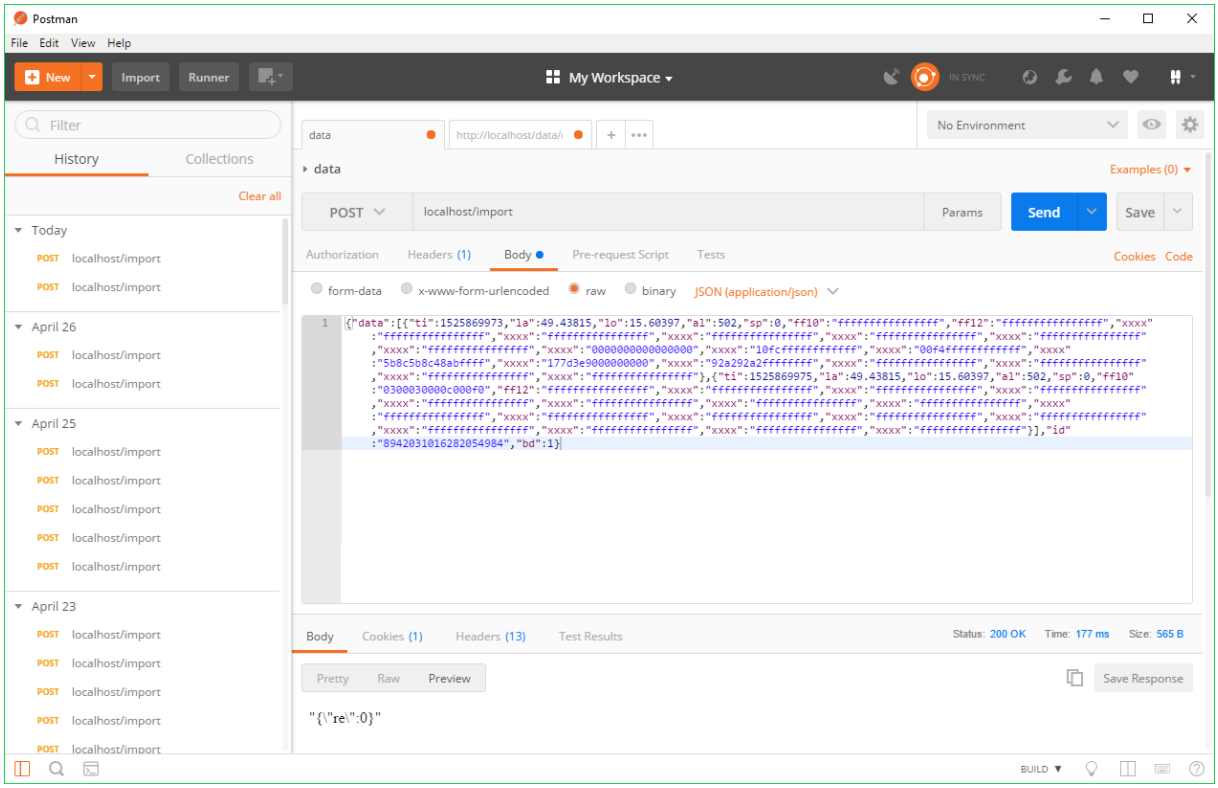
Obrázek 34 Tabulka zpracovaných dat v databázi MySQL

7.6 Testování s programem Postman

Testování komunikace mobilní části se serverem lze samozřejmě provádět pohledem na odpověď serveru na straně Arduina a v úspěšném případě sledovat uložená data na serveru. Nejdříve je však žádoucí ověřit správnou funkci API. V případě HTTP metody GET stačí zadat adresu do webového prohlížeče. Pokud používáme metodu POST v souvislosti s webovými

formulářů, není také nic jednoduššího, než modelová data do formulářů zadat a odeslat na server. Pokud však používáme metodu POST a nevyužíváme webové formuláře, nadto ještě předáváme data ve formátu JSON, s pouhým webovým prohlížečem nevystačíme.

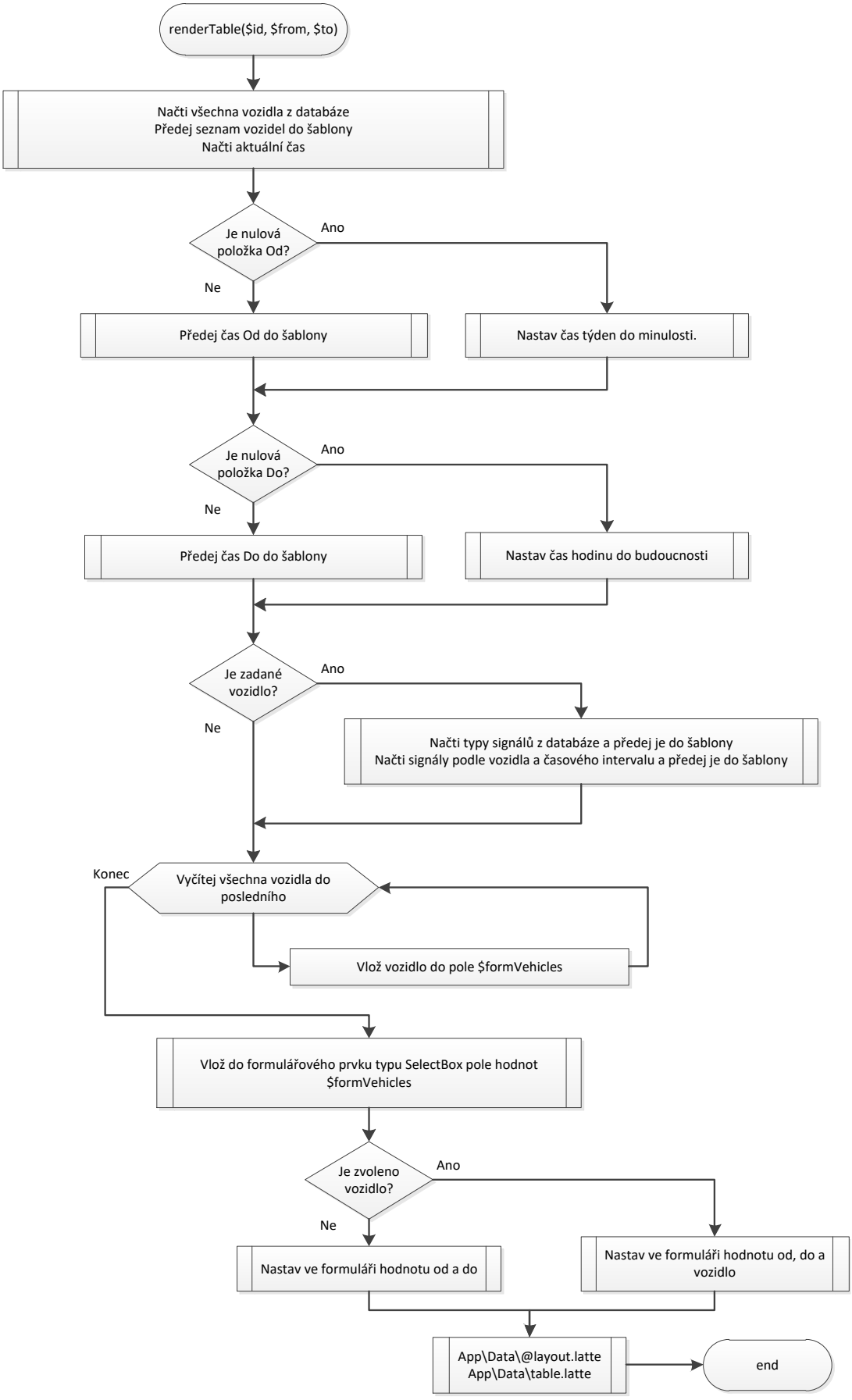
V takovéto situaci se dobrý nástrojem stává například program Postman, kde zadáme URL, vybereme metodu (POST) a do těla zprávy se zvoleným formátem JSON vložíme obsah. Následně stačí už jen iniciovat odeslání tlačítkem *Send*. Pod zadanými daty lze pak procházet odpověď serveru od HTTP hlavičky po plnohodnotné grafické zobrazené webové stránky, pokud samozřejmě server vrací webovou stránku s HTML.



Obrázek 35 Screen programu Postman se zkušebními daty ve formátu JSON

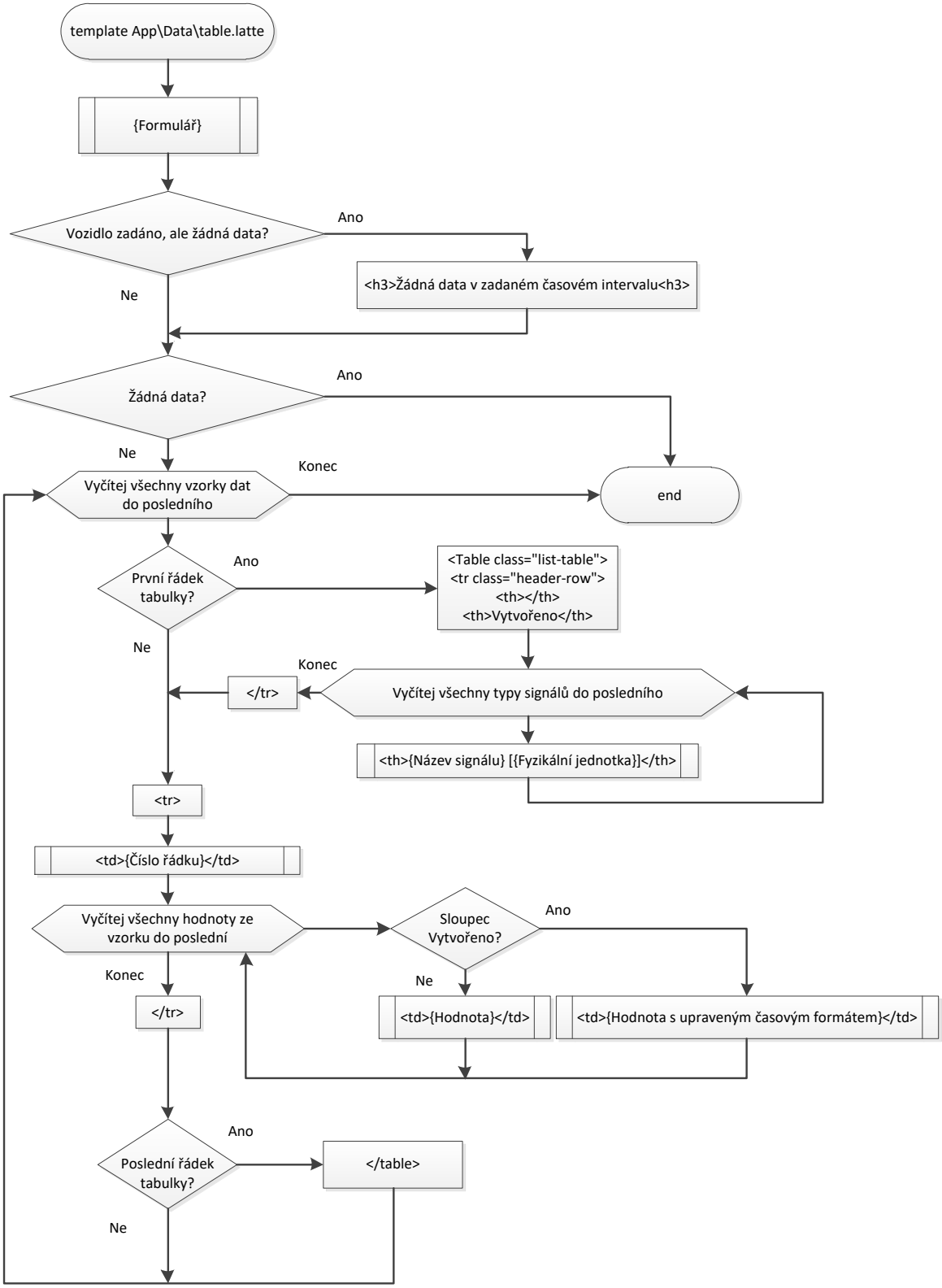
7.7 Vykreslování dat do tabulek

Jako příklad poslouží stránka *Data v tabulce* s funkcí *renderTable()* v presenteru. Funkce převezme případná vstupní data GET dotazu, načte potřebná data z databáze. Potřebné proměnné pro vykreslení do šablony předá šabloně a v posledním kroku naplní daty *tableForm*. *TableForm* je název formuláře na stránce s tabulkou signálů.



Zdrojový kód 6 Funkce DataPresenter::renderTable()

Šablona se pak postará především o vykreslení formuláře tabulky dat s využitím skriptovacího jazyka HTML.



Zdrojový kód 7 Šablona ke stránce s tabulkou signálů

Formulář *tableForm* má v presenteru svou vlastní funkci, která je z vnějšku nepřístupná (protected function). Ve funkci jsou definovány jednotlivé prvky formuláře a pravidla (omezení) omezující vstupní hodnoty. Framework Nette má velice dobře zpracované ošetření vstupních hodnot formulářů (zabezpečení), které bývalo častým nešvarem před masivnějším nasazením frameworků. Samotná pravidla definovaná pro jednotlivé pole formuláře se promítají nejen do vyhodnocení na straně serveru, ale také do JavaScriptu na straně klienta.

```
protected function createComponentTableForm(){
        $form = new UI\Form;
        $form->addText('dateFrom', '')
                ->setRequired('Vyplňte prosím čas')
                ->setAttribute('class', 'span2')
                ->setAttribute('placeholder', 'Od data')
                ->setAttribute('id', 'dateFrom')
                ->addRule(Form::MAX_LENGTH, 'Příliš dlouhé', 19);
        $form->addText('dateTo', '')
                ->setRequired('Vyplňte prosím čas')
                ->setAttribute('class', 'span2')
                ->setAttribute('placeholder', 'Do data')
                ->setAttribute('id', 'dateTo')
                ->addRule(Form::MAX_LENGTH, 'Příliš dlouhé', 19);
        $form->addSubmit('insert', 'Zobrazit')
                ->setAttribute('class', 'btn btn-inverse')
                ->onClick[] = [$this, 'selectTableFormSucceeded'];
        $form->addSubmit('save', 'Uložit jako CSV')
                ->setAttribute('class', 'btn btn-inverse')
                ->onClick[] = [$this, 'selectTableFormToCsvSucceeded'];
        return $form;
}
```

Zdrojový kód 8 Definice podoby formuláře tableForm v presenteru Data

Do šablony lze předat celý formulář makrem {control tableForm}, jak je uvedeno ve vývojovém diagramu, nebo dílčí části zvlášť. Výhoda prvního spočívá v jednoduchosti, v druhém případě pak vyniká možnost detailního nastavení grafického stylu. Protože jsem využil HTML šablonu, která si s jednoduchým předáním formuláře nevystačí, pokud je žádoucí udržet estetický a jednotný grafický styl, neobejdu bez druhé možností nazývané jako low-level formuláře. V šabloně je tedy reálně makro {control tableForm} nahrazeno následujícím obsahem.

```
{form tableForm}
       <div class="input-prepend">
             <span class="add-on"><i class="icon-time" title="Od"></i></span>
             {input dateFrom}<span class="err-info">{inputError dateFrom}</span>
```

```
        </div>

        <div class="input-prepend">
              <span class="add-on"><i class="icon-time" title="Do"></i></span>
              {input dateTo}<span class="err-info">{inputError dateTo}</span>
        </div>

        <div class="input-prepend">
              <span class="add-on"><i class="icon-map-marker"></i></span>
              {input vehicle}<span class="err-info">{inputError vehicle}</span>
        </div>
        <div class="row">
              {input insert}
              {input save}
        </div>
{/form}
```

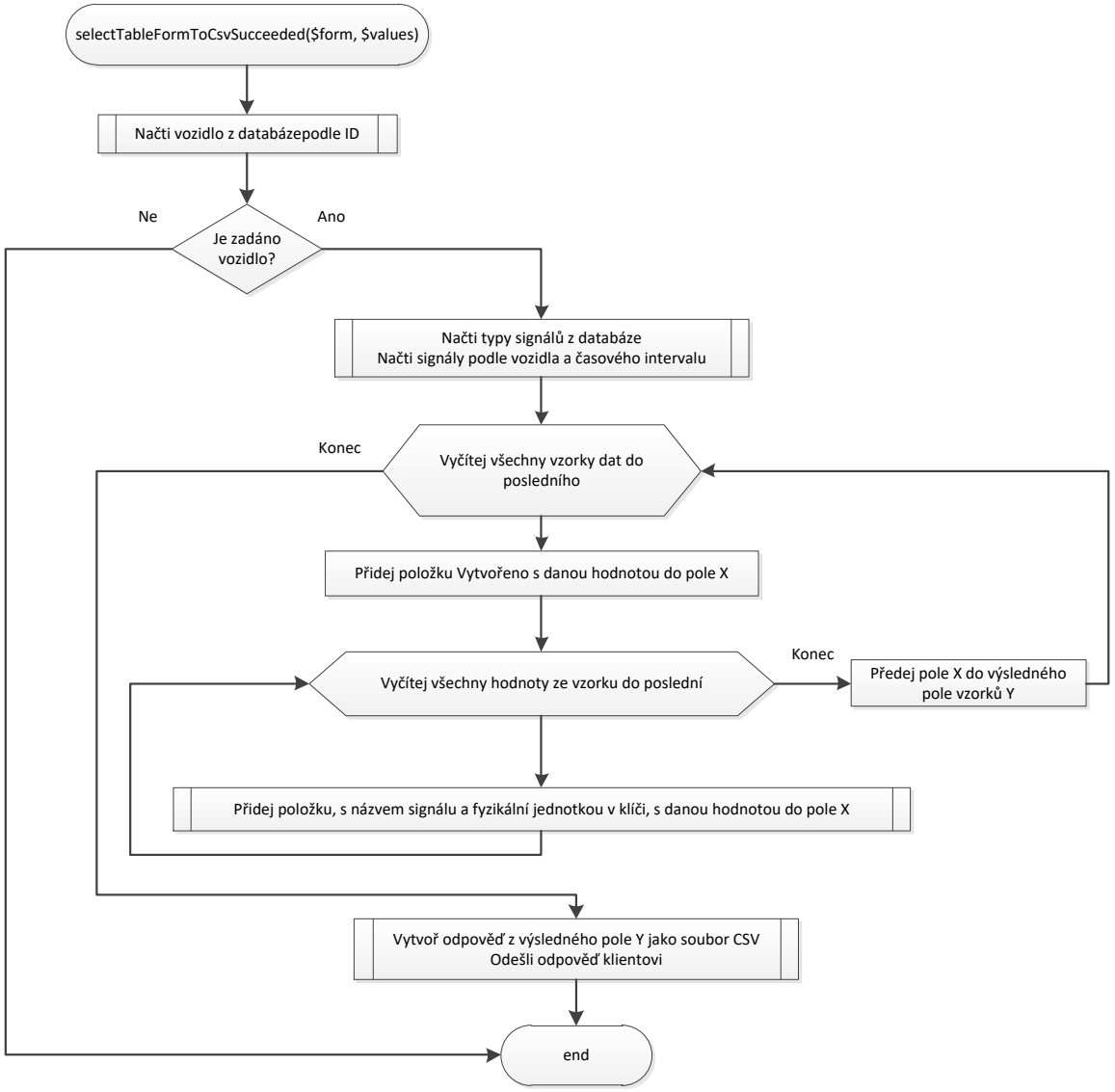
Zdrojový kód 9 Low-level formulář tableForm v šabloně

7.8 Export do CSV

Základní technické řešení obstarává komponenta *Nette CSV response* od Vladislava Hejdy, která se nabízela jako vhodné řešení. Protože její poslední aktualizace proběhla v roce 2014, není v současnosti možné komponentu automaticky doinstalovat programem Composer do frameworku Nette z důvodu nekompatibility. Soubor jsem tak přidal do složky *app/model* a opravil daný zdroj nekompatibility. Nakonec stačilo nahradit závislost na zastaralé třídě *Nette\Object* traitou *Nette\SmartObject*.

Data ukládaná do CSV mají podstatě stejnou strukturu jako data zobrazovaná v tabulce na webu. První řádek tvoří hlavička zpráv s názvem zpráv a fyzikální jednotkou. Hodnoty jsou odděleny středníky. Desetinná místa jsou aktuálně oddělovaná tečkou, tak jak hodnoty přichází z databáze.

Podívejme se ale na související předání dat komponentě *Nette CSV response* a odeslání finálního souboru webovému klientovi. V předchozí kapitole bylo představeno konkrétní řešení formuláře. Z kódu v presenteru je patrné, která funkce bude volána po kliknutí na tlačítko Uložit jako CSV. Tou je funkce *selectTableFormToCsvSucceeded()*. Komponentě Nette CSV vyžaduje minimální množství předávaných informací. Předáváno je pouze pole vzorků dat. Hlavička CSV souboru je vytvářená z klíčů k hodnotám, což v případě signálů s požadavkem popisné řetězce představuje opravdu dlouhé klíče, ale řešení je to funkční i s diakritickými znaky.



Vývojový diagram 12 Funkce DataPresenter:: selectTableFormToCsvSucceeded ()

7.9 Vykreslování vozidel do mapy

Stránka Poloha vozidel obsahuje Google mapu s polohou vozidel. Pro vývojáře webových aplikací připravil Google Maps API a k němu dokumentaci na stránkách *developers.google.com/maps/documentation/*. Mapa se do stránky negeneruje na straně serveru, ale až na straně klienta (v internetovém prohlížeči) a aktivně reaguje na požadavky uživatele do zavření záložky webového prohlížeče. Většinu práce tak vykonává JavaScript.

Pro pochopení generování stránky je třeba nejdříve začít v presenteru, kde jsou na straně serveru získávány závislosti předávané šabloně a následně klientovy. Funkce *renderMap()* v presenteru je však v tomto naprosto prázdná, protože data o vozidlech pro mapu si vyžádá až JavaScript. Po vykonání funkce v presenteru jsou data vykreslená do šablony *map.latte*. Šabloně nejsou předávána žádná dat z presenteru, a tak nemáme k dispozici žádné proměnné.

```
{block title}
	<title>Mapa | Data</title>
{/block}

{block content}
<div data-url="{link //location}" id="address"></div>
<div id="map"></div>
<script async defer
	src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDYbGT0yG2rUWUBk4QiQzUBGyfLG4ZxCUM&callback=initMap">
</script>
{/block}
```

Zdrojový kód 10 Šablona map.latte

Tag div s id="address" má za úkol předat absolutní adresu URL, z relativní adresy, pro správnou funkci JavaScriptu, aby bylo možné nasadit aplikaci na libovolnou doménu a nebyla tím ovlivněna její správná funkce.

V šabloně je v bloku *title* nastaven název webové stránky, který se objeví v záložce webové stránky. Déle je v bloku *content* nastaven vlastní obsah stránky. Oddíl <*div*> s atributem id="address" předává absolutní URL adresa stránky v atributu *data-url*. Odkazovanou webovou stránku si následně zavolá JavaScript ze souboru googleMap.js, jež se přidává do HTML hlavičky závislostí v souboru základní šablony @layout.latte. JavaScriptem je volána funkce *actionLocation()* z presenteru *Data* a ta předá informace pro jednotlivá vozidla vykreslovaná do mapy, ale o tom až dále. Šablona obsahuje také oddíl s atributem id="map" s vnořeným krátkých JavaScriptem obsahujícím odkaz na Google Maps API s vygenerovaným klíčem k naší mapě. Tím je řečeno, kam bude Google Maps API vykreslovat mapu. Google Maps API si prostřednictvím tohoto kratičkého scriptu může do stránky zároveň naincludovat vlastní externí JavaScriptové soubory závislostí, které jsou potřebné pro zobrazení mapy.

Já se ale teď zaměřím na mnou vytvořený JavaScriptový soubor, který předává Google Maps API informace o vozidle pro vykreslení na mapě. Ostatní záležitosti jíž zajišťuje samotné Google Maps API autonomně (pohyby po mapě, zoomování, přepínání na satelitní mapu,…). Můj JavaScriptový soubor googleMap.js volá předávanou webovou adresu, která obsahuje REST API a server odpovídá prostřednictvím funkce *actionLocation()* z presenteru *Data*. Funkce vrací data o všech vozidlech ve formátu JSON. Jelikož funkce neobsahuje žádné vstupní parametry, můžeme její URL zadat do prohlížeče a dostaneme stejnou následující odpověď. Nicméně danou adresu nevede žádný veřejný odkaz a při volání dané adresy nesmíme

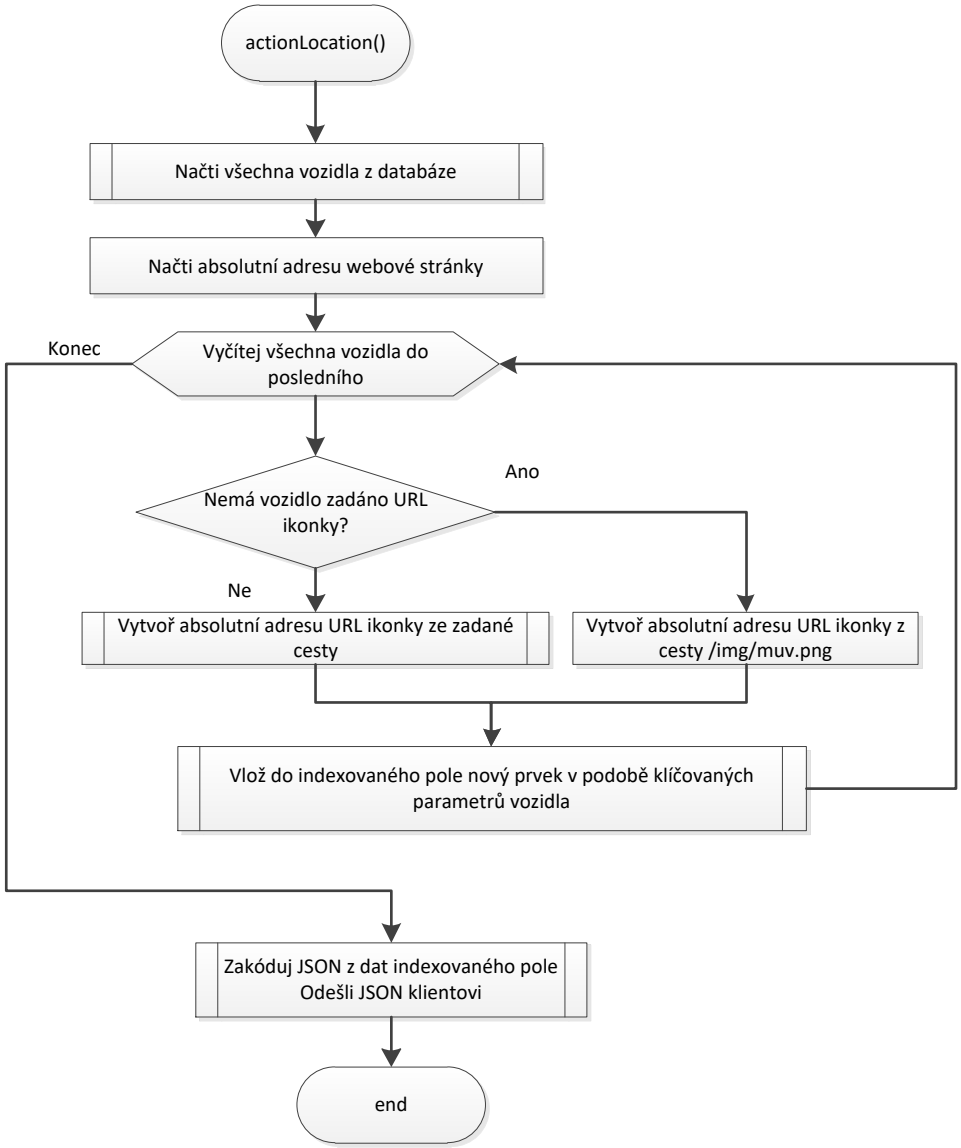
zapomenout v presenteru na funkci *startup()*, která ověřuje přihlášení, takže bez přihlášení uživatele proběhne prosté přesměrování na úvodní stránku s přihlašovacím formulářem.

```
"[{\"vehicle\":\"89420310172521871 27\",\"title\":\"MUV 75   002\",\"speed\":0.04,\"lat\":49.41831,\"l
ong\":15.59913,\"alt\":494,\"url\":\"http://sledovani-muv.mzf.cz/img/muv75.png\",\"act\":false},
{\"vehicle\":\"8942031016282054984\",\"title\":\"MUV   75   001\",\"speed\":0.0,\"lat\":49.85261,\"lon
g\":16.05335,\"alt\":388,\"url\":\"http://sledovani-muv.mzf.cz/img/muv75.png\",\"act\":false},
{\"vehicle\":\"8942031014062077770\",\"title\":\"MUV   74   001\",\"speed\":0.0,\"lat\":0.0,\"long\":0
.0,\"alt\":0,\"url\":\"http://sledovani-muv.mzf.cz/img/muv.png\",\"act\":false},
{\"vehicle\":\"muv_74_005\",\"title\":\"MUV   74.2   023\",\"speed\":25.0,\"lat\":49.45647812,\"long\"
:17.45022964,\"alt\":212,\"url\":\"http://sledovani-muv.mzf.cz/img/muv.png\",\"act\":true}]"
```

Zdrojový kód 11 Data o vozidle ve formátu JSON

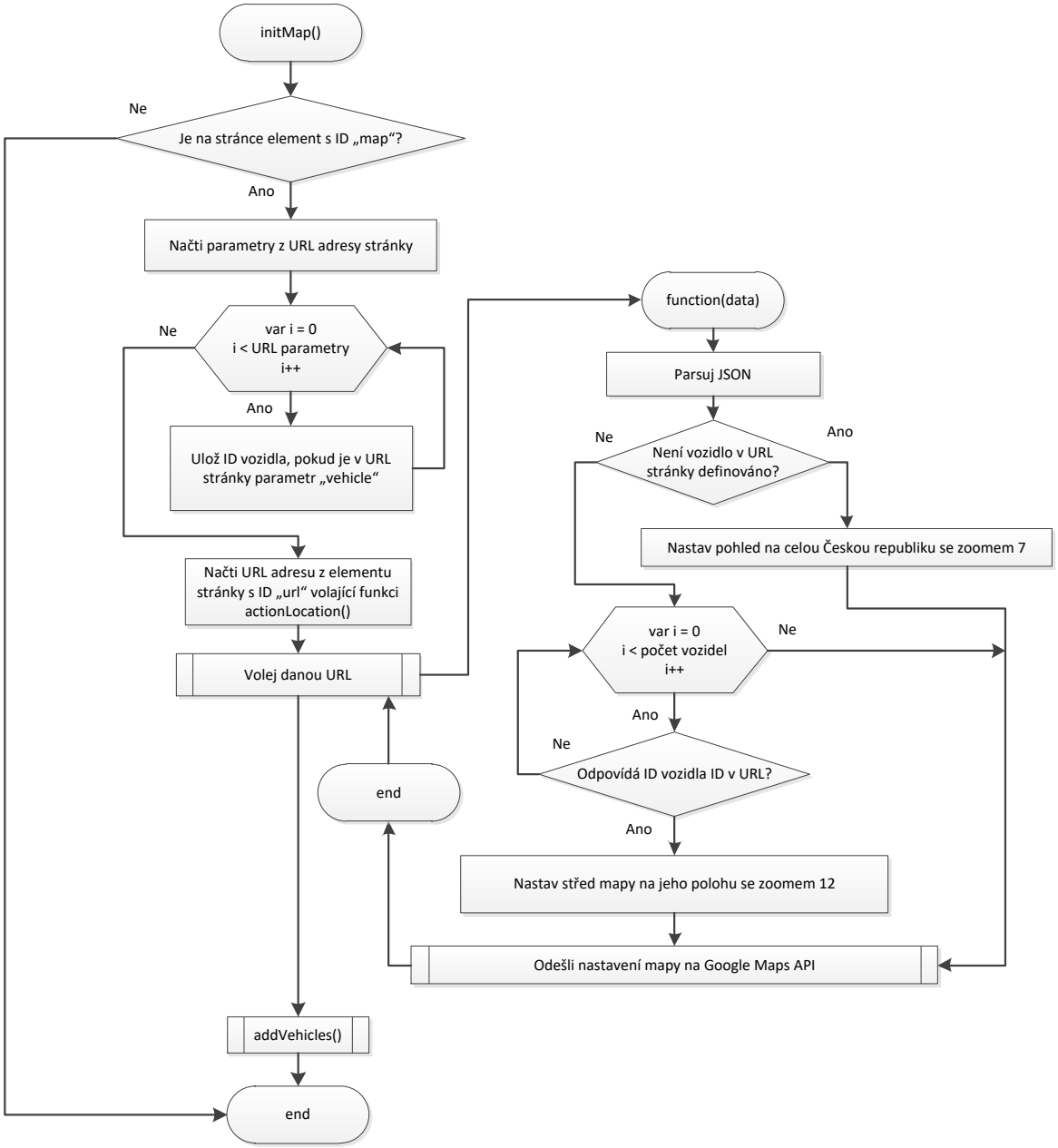
- vehicle – ID vozidla (SIM ICCID)
- title – název vozidla
- speed – rychlost vozidla
- lat – zeměpisná šířka
- long – zeměpisná délka
- alt – nadmořská výška
- url – absolutní URL adresa ikonky vozidla
- act – aktivní/neaktivní vozidlo

Funkce *actionLocation()* funguje následovně. Vybere vozidla z databáze s dalšími informacemi, vygeneruje absolutní adresy k ikonám vozidel a spolu s dalšími parametry předá data ve formátu JSON webovému klientovi. To je patrné z následujícího vývojového diagramu.



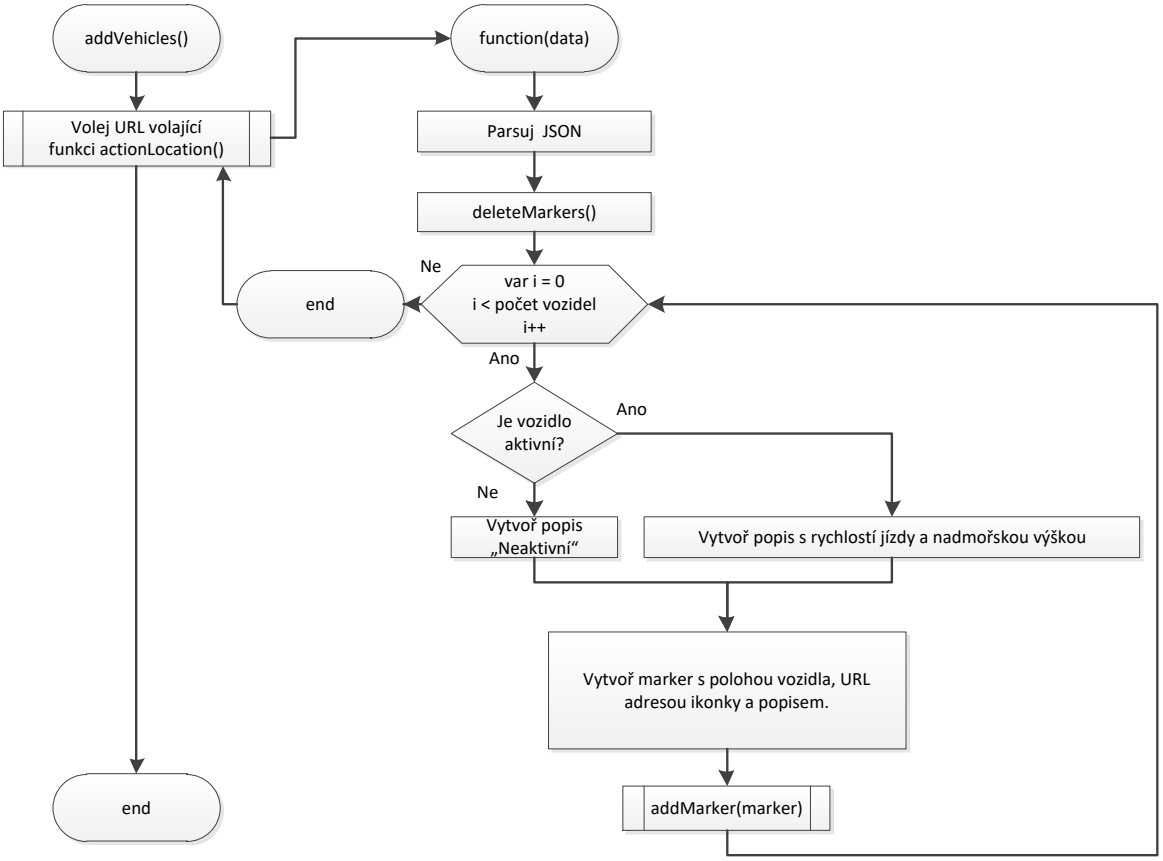
Vývojový diagram 13 Funkce DataPresenter::actionLocation()

Dále už se podíváme i na samotný JavaScriptový soubor, ke kterému vede z kořenové složky frameworku následující cesta *www/js/googleMap.js*. Funkce jsou v některých případech ve funkcích vnořeny do volání metody GET. Tomu je také mírně přizpůsobena podoba vývojových diagramů.



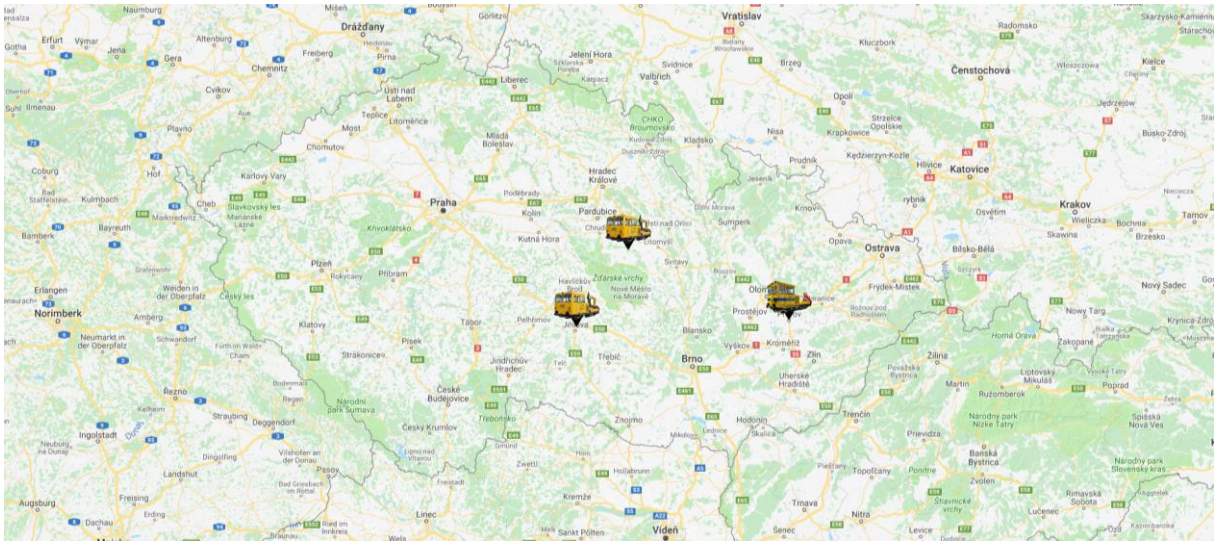
Vývojový diagram 14 JavaScript k mapě Google, funkce initMap()

Funkce *initMap()* nejdříve z odkazu stránky zjišťuje, zda nebyl vytvořen požadavek na konkrétní vozidlo a je na server vyslán GET dotaz s voláním výše rozebrané funkce *actionLocation()*. Pokud nebylo vyžádáno konkrétní vozidlo, zobrazí se mapa s předdefinovaným zoomem (7) a středovou polohou. Nastavení konkrétními hodnotami ve vývojovém diagramu představuje zobrazení České republiky a jejího blízkého okolí. S požadavkem na konkrétní vozidlo je střed mapy v poloze vozidla a zoomem 12. V závěru je volána funkce *addVehicles()*, která je popsána níže.



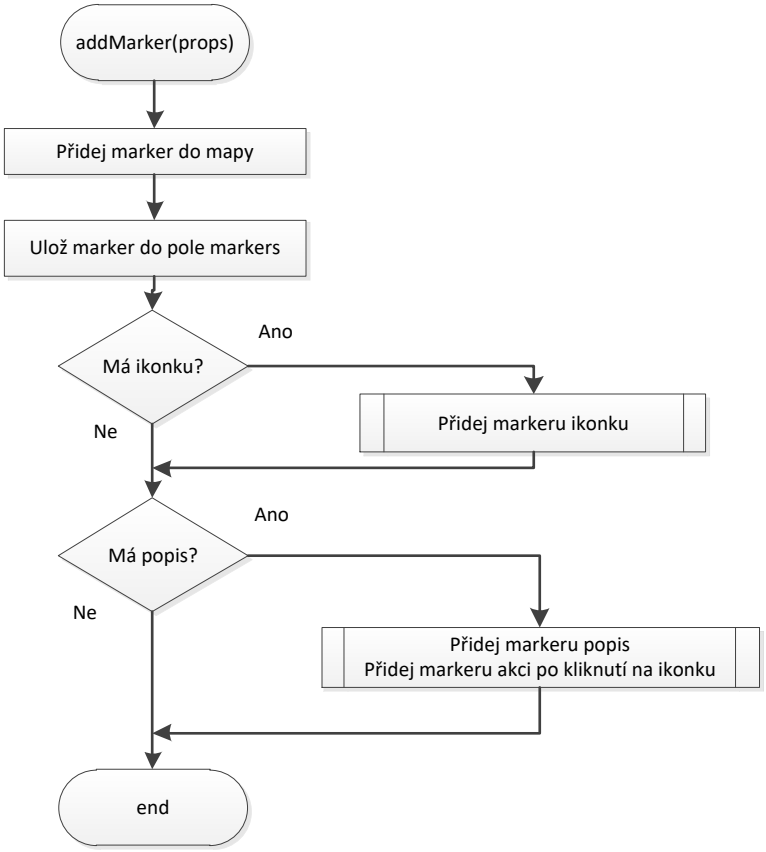
Vývojový diagram 15 JavaScript k mapě Google, funkce addVehicles()

Funkce *addVehicles()* přidává do mapy jednotlivé markery. Markery jsou jednotlivé vyznačené body na mapě. Funkce předává odkazy na vlastní ikonky markerů a dále informace, které se zobrazí kliknutím na ikonku. Samotná vlastnost zobrazení daných informací po kliknutí je také předávána jako požadavek na dané chování prvku.



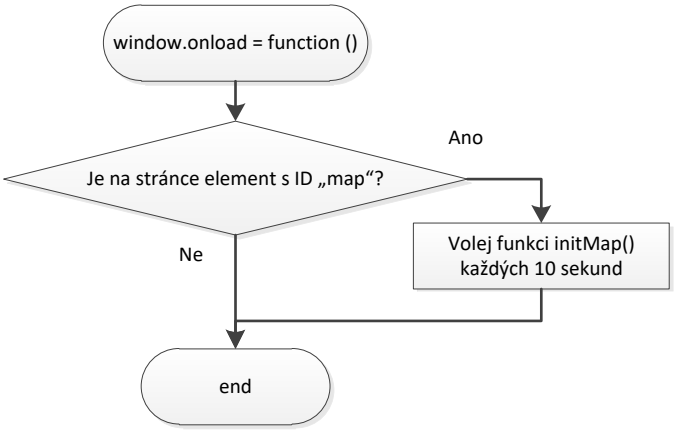
Obrázek 36 Google mapa se třemi markery

V posledním kroku utváření dat markera je volána funkce *addMarker()*, která předává data s informacemi o vozidle spolu s absolutním odkazem na ikonku do Google Maps API.

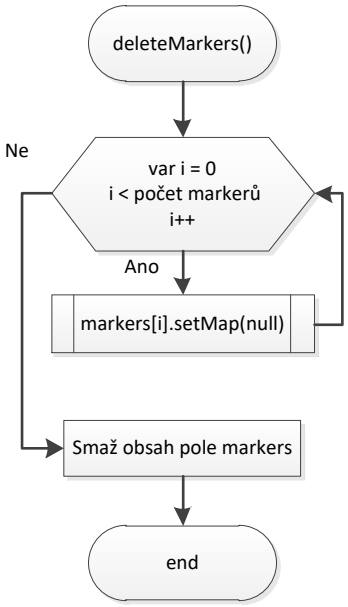


Vývojový diagram 16 JavaScript k mapě Google, funkce addMarker()

Dále je nadefinováno pravidelné volání funkce *initMap()* s periodou 10 sekund. Pohled na mapu se se změnou polohy vozidel neposouvá a bublina s údaji o stavu, rychlosti a nadmořské výšce vozidla při překreslení zmizí. Ve funkci *addVehicles()* se před přidáním nových markerů (ikon označujících polohu vozidel v mapě) musí původní markery odstranit jednotlivě, pokud nemá dojít k přegenerování celé mapy. V tomto případě jde o vlastnost Google Maps API. Proto byla v JavaScriptu deklarována globální proměnná *markers[]*, která v sobě uchovává seznam právě zobrazených markerů (vozidel) v mapě. Ve funkci *addMarker()* jsou do tohoto seznamu ukládány jednotlivé markery a následně ve funkci *deleteMarkers()* dochází k odeslání NULL hodnot markerů do Google Maps API za účelem vymazání vozidla z mapy.



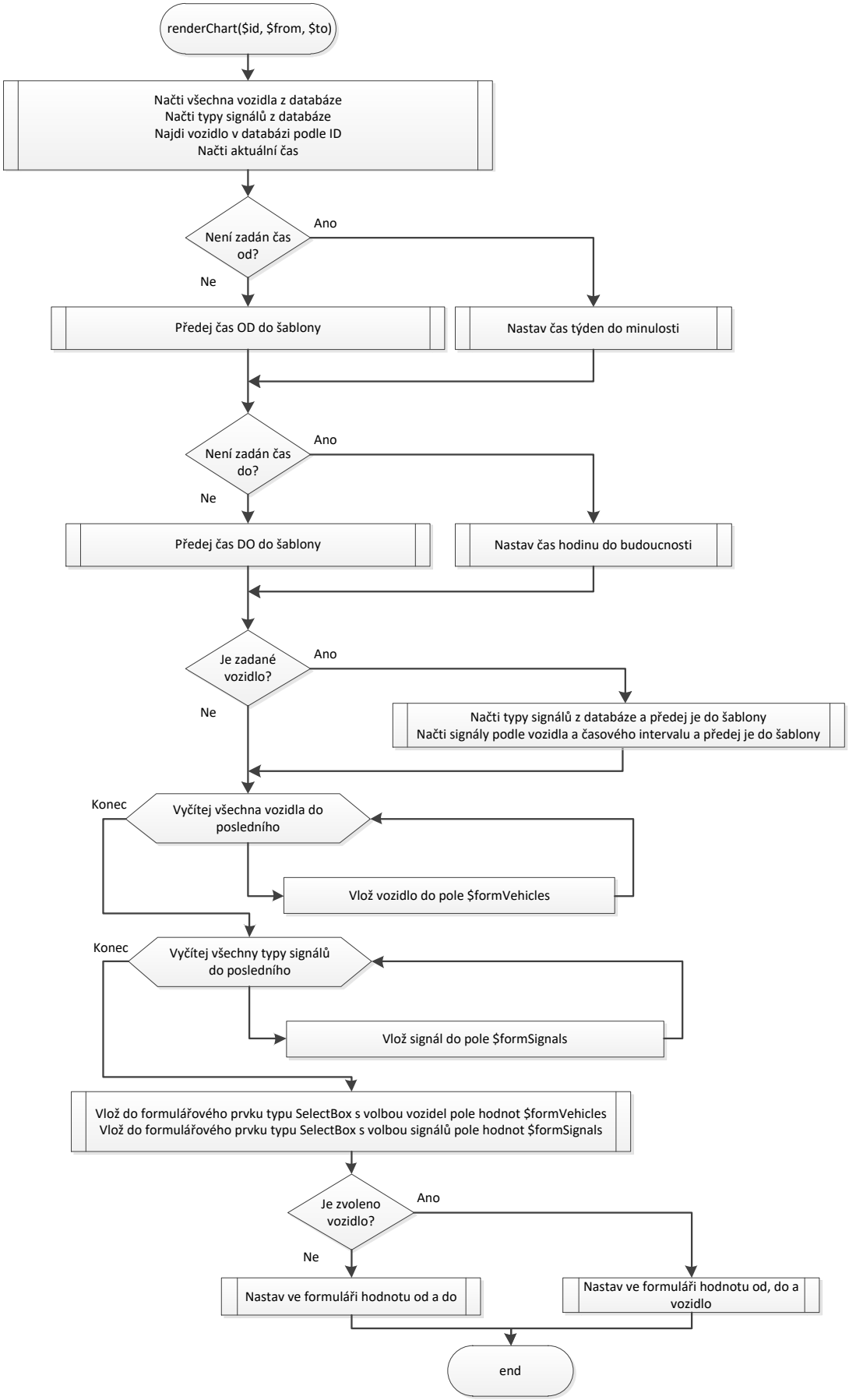
Vývojový diagram 17 JavaScript k mapě Google, naplnění windows.onload



Vývojový diagram 18 JavaScript k mapě Google, funkce deleteMarkers()

7.10 Vykreslování dat do grafu

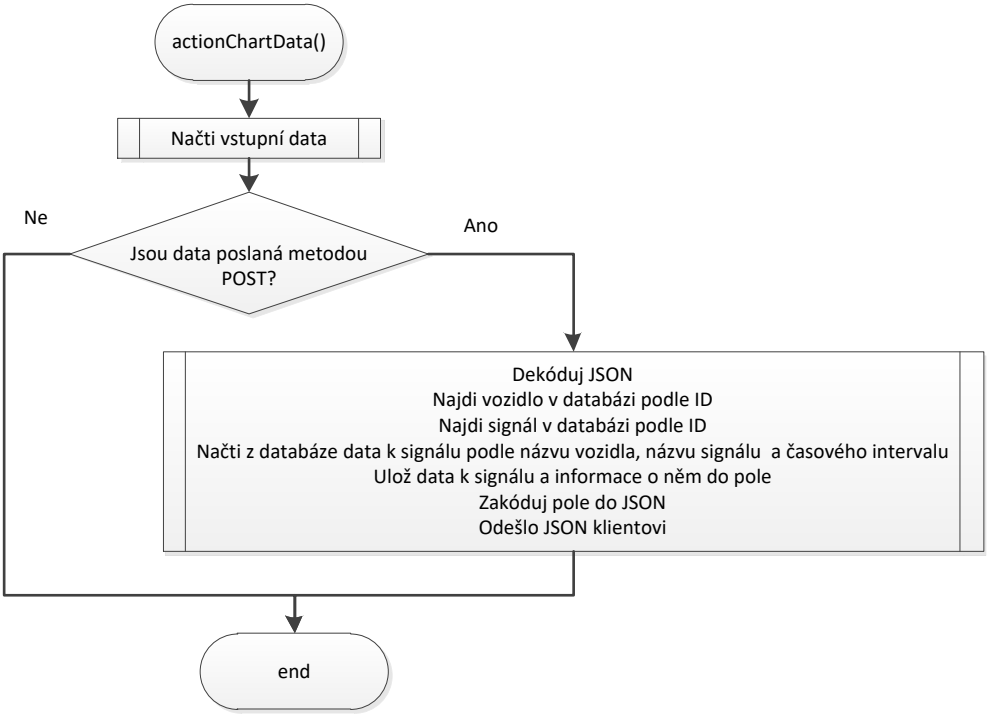
Jednotlivé signály lze vykreslit do grafu na stránce Data v grafu. Nejdříve je nutné vybrat časový interval dat a vozidlo. Poté je zobrazen graf s již zvoleným časovým intervalem na ose X. V dalším kroku lze pak přidávat jednotlivé signály. K dispozici jsou dvě osy Y, mezi kterými lze přepínat během vkládání jednotlivých signálů. Rozsahy na osách Y se volí automaticky podle vkládaných dat. Jednotlivé signály pak lze skrývat kliknutím na popis daného signálu v legendě grafu, nebo odmazávat v opačném pořadí, než byly jednotlivé signály přidávány. Nejdříve se tedy opět chronologicky podíváme na funkci v presenteru, která vybraná data dále podstupuje šabloně.



Vývojový diagram 19 Funkce DataPresenter::renderChart()

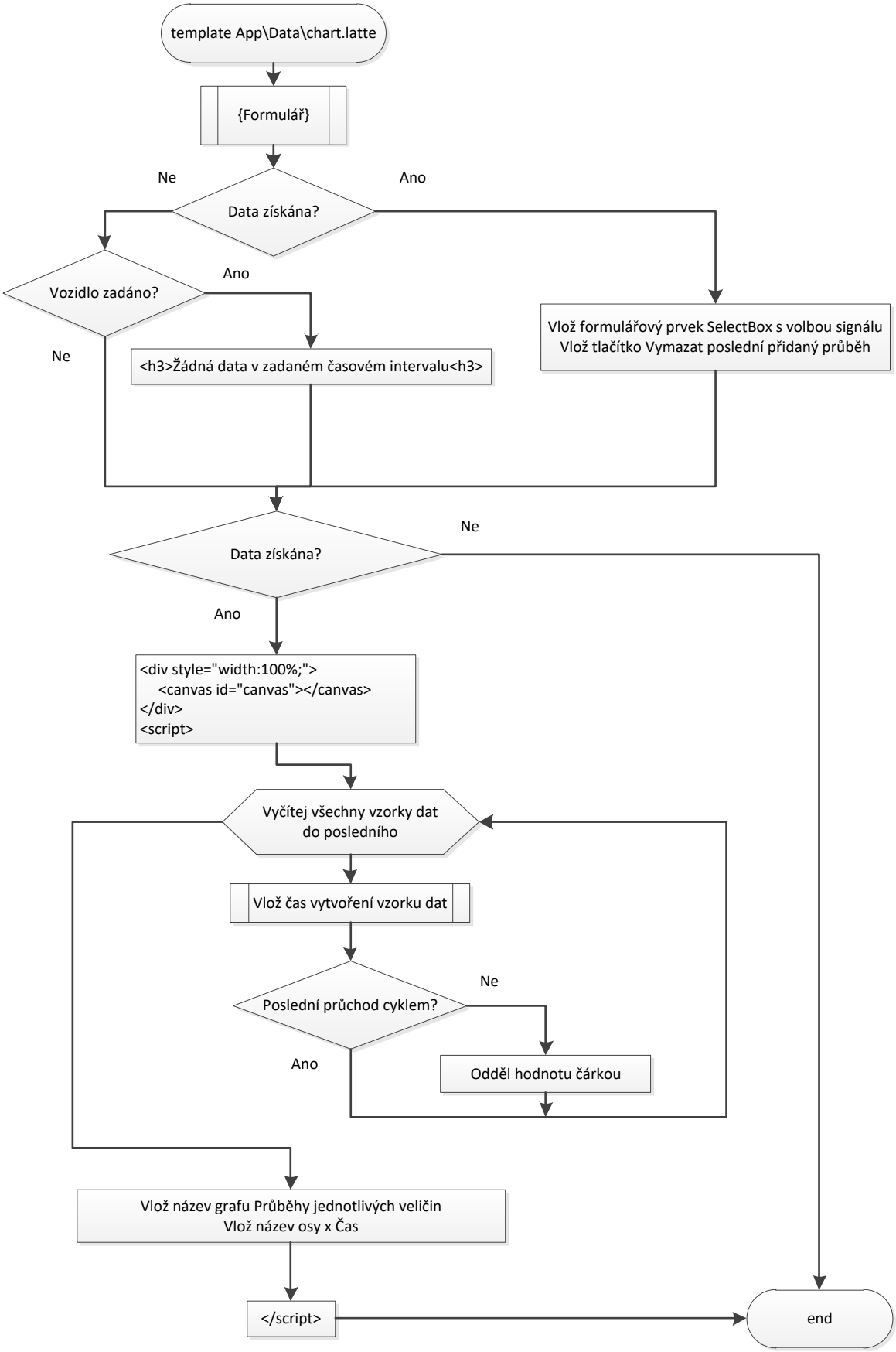
Z funkce v presenteru jsou potřebná data předána šabloně. V šabloně je situace složitější a pro zobrazení ve vývojovém diagramu značně nepříznivá. V šabloně jsou jako obvykle data generovaná do stránky. Součástí šablony je kromě HTML také JavaScript, který je v tomto případě nutné mít i v šabloně, jinak by nebylo možné používat rozšíření využívající moderní prvek pro dynamické vykreslování bitmap – canvas, který přišel s HTML5 (2014). Samotný HTML5 canvas by však pro jednoduché zobrazení nestačil. Nabízí se několik zajímavých podpůrných řešení pro zobrazení grafů. Mne nejvíce zaujal projekt Chart.js ze stránek www.chartjs.org, který, pro zajímavost, čítá skoro 20 tisíc řádků kódu v jazyku JavaScript a je samozřejmě dostupný jako svobodný software.

Zobrazení průběhů v grafu na základě volby signálů ve formuláři, je jako obvykle při použití JavaScriptu doprovázeno spoluprací se serverem „zadními vrátky“ pomocí REST API s datovým formátem JSON. Data tentokrát zajišťuje funkce *actionChartData()*.



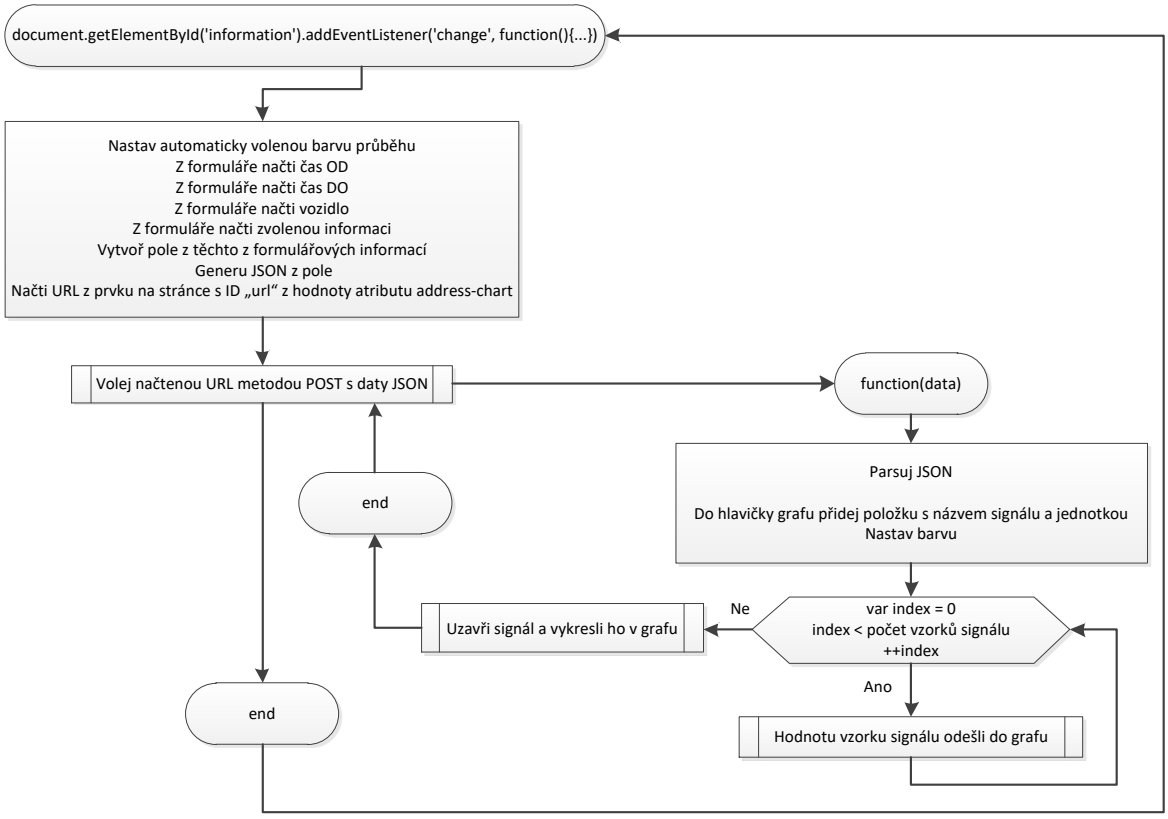
Vývojový diagram 20 Funkce DataPresenter::actionChartData

V následujícím vývojovém diagramu je uvedená logika šablony, která tentokrát zahrnuje i část s kódem JavaScriptu, která se na straně serveru plní daty. JavaScript je samozřejmě aktivní až na straně klienta. Tady jde podstatě pouze o naplnění proměnných, jako by šlo z pohledu JavaScriptu o konstanty zapsané v samotném scriptu. Nejde o soubor se zmiňovanými téměř 20 tisíci stranami, pouze několika málo nezbytnými funkcemi, která jsou s ním propojeny.



Vývojový diagram 21 Šablona ke grafu

Následující JavaScriptová funkce vychází z příkladu k pluginu *Chart.js* a byla upravena pro dané podmínky na webu. Funkce vkládá do grafu jednotlivé průběhy. Data přebírá od funkce *actionChartData()*, kterou sama volá s požadavkem na zaslání dat pro konkrétní vozidlo, signál a časový interval vzorků dat. Absolutní URL adresu pro volání funkce *actionChartData()* získává z hodnoty atributu prvku webové stránky.



Vývojový diagram 22 Reakce na změnu prvku s ID information

Z přehledu je vynechán vývojový diagram události *onload*, kterou jsem nijak nemodifikoval a obsahuje pouze dva řádky kódu pro inicializaci a stejně tak funkce reagující na stisknutí tlačítka *Vymazat poslední přidaný průběh*.

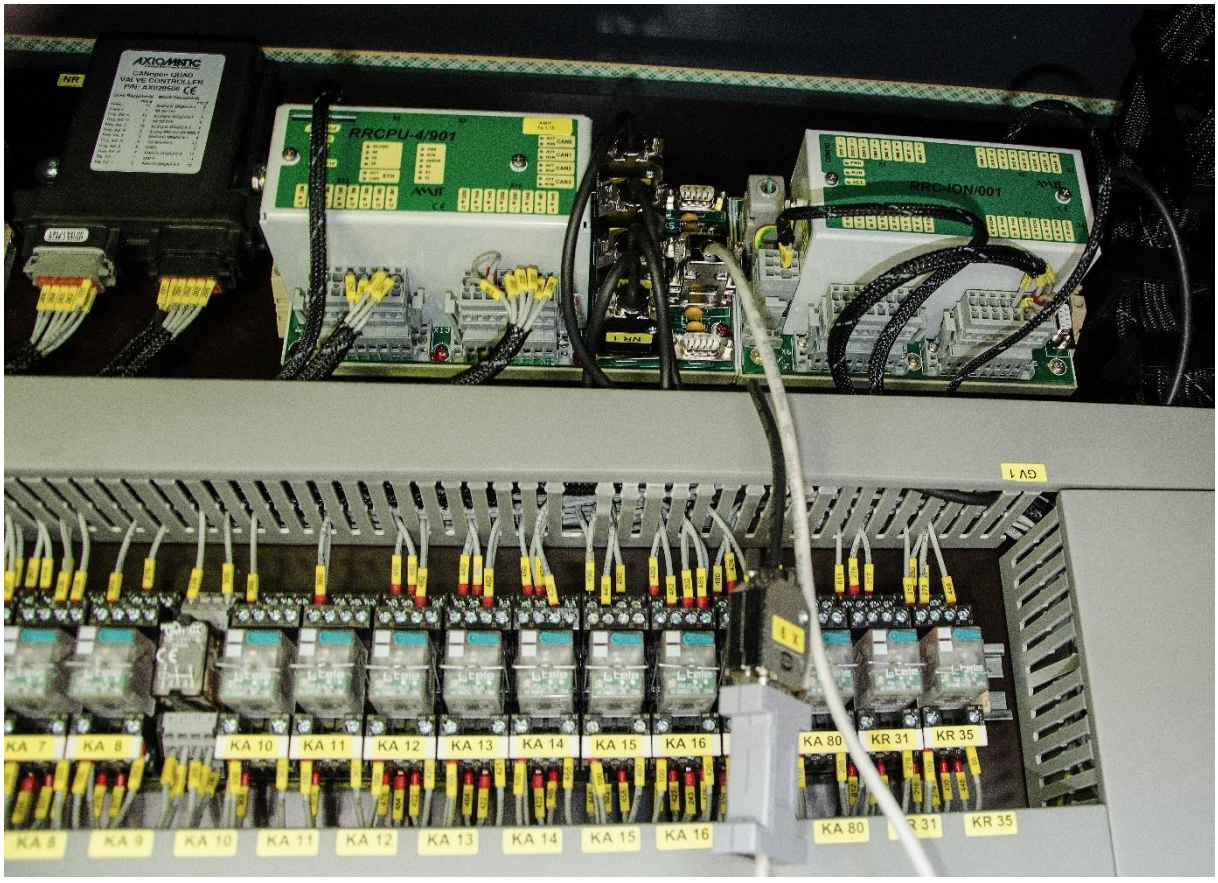
8 ZKUŠEBNÍ NASAZENÍ DO PROVOZU

V závěru dokončování této práce bylo zařízení během jednoho dne zkušebně testováno na vozidle MUV 75 002 na vlečce v Jihlavě, kterou společnost CZ LOKO standardně využívá pro testování svých vozidel.

Obrázek 37 Vozidla MUV 75

Mobilní zařízení bylo umístěno do rozvaděče vozidla. Po odpojení jednoho konektoru sběrnice CAN z řídícího počítače Amit bylo mobilní zařízení zapojeno do série. Jeden z konektorů mobilní části byl připojen do počítače Amit a druhý na odpojený konektor. Napájení bylo zajištěno připojením do konektoru autozásuvky s napájecím napětím 24 V. Následně byla umístěna anténa GPS k oknu, mimo rozvaděč vozidla. Arduino bylo ještě připojeno k počítači pro sledování ladících zpráv. Spolu se sledováním ladících zpráv bylo samozřejmě také sledováno zasílání dat z prostředí webové aplikace a na chytrém telefonu byla obsluhována webová aplikace.

Obrázek 38 Zkušební spojení mobilní části s vozidlem MUV 75



Obrázek 39 Připojení mobilní části na sběrnici CAN

Obrázek 40 Prostřední z antén je anténa pro GPS vytvořeného systému vzdálené diagnostiky

Obrázek 41 Pohled na testovací zapojení ve vozidle MUV 75

GSM anténa modulu SIM808 je umístěna přímo na tomto modulu. Z důvodu této skutečnosti bylo provedeno měření přijímaného výkonu v závislosti na umístění antény ve vozidle s ohledem na ověření možnosti ponechání antény přímo v rozvaděči spolu s mobilním zařízením. Testováno bylo umístění antény uprostřed stanoviště obsluhy, déle v rozvaděči s otevřenými dvířky a po zavření dvířek. Informace o přijímaném výkonu získávaná z GSM se může pohybovat v intervalu od 2 do 30. Pro jednotlivé hodnoty existuje tabulka [23] s odpovídajícími hodnotami v dBm a slovním hodnocením kvality signálu. Hodnoty 2-9 představují nedostatečný signál, 10-14 dostatečný, 15-19 dobrý a >19 výborný. Během tohoto experimentálního testování byly naměřeny následující hodnoty.

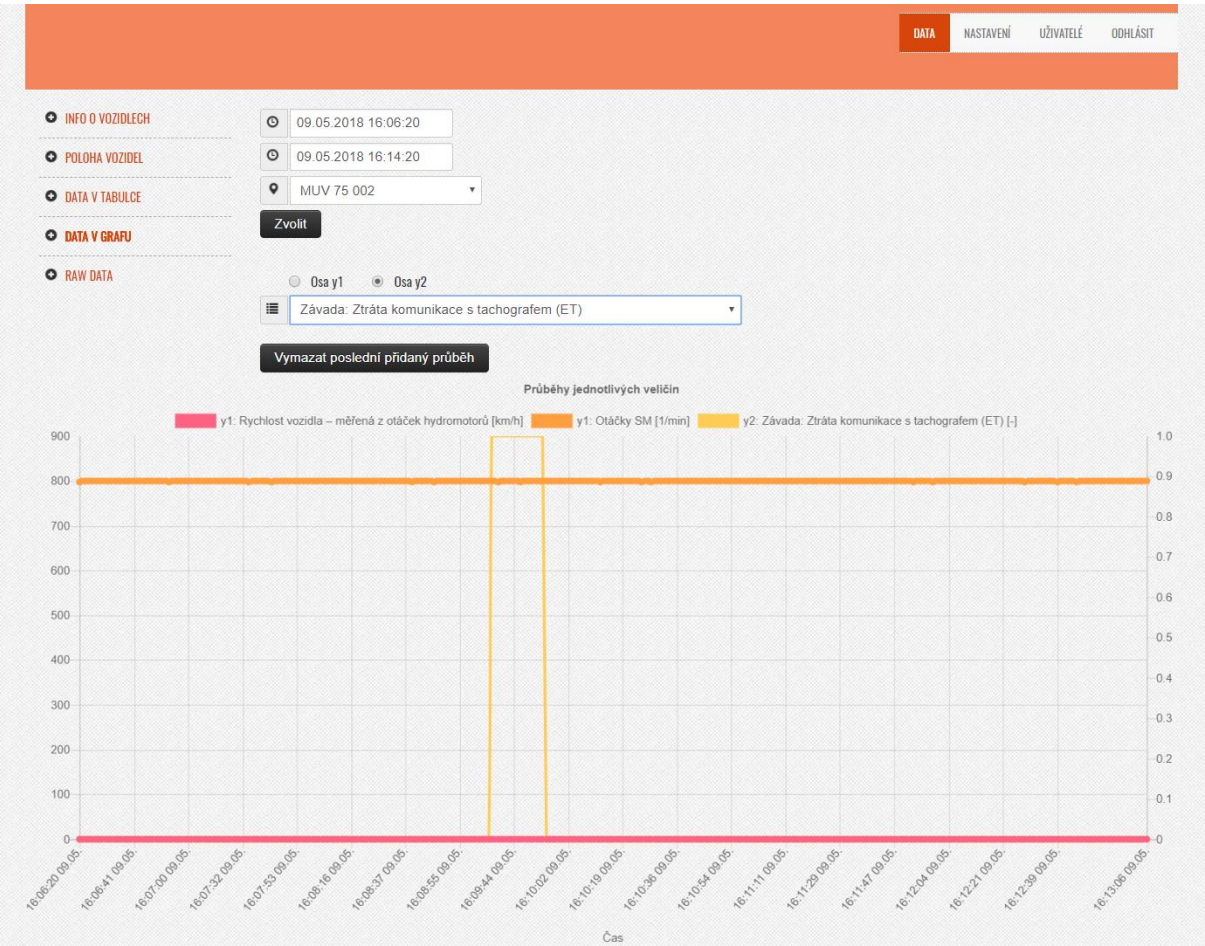
Tabulka 4 Výkon přijímaný z GSM s různým umístěním antény ve vozidle MUV 75

Umístění	Přijímaný výkon [-]	Přijímaný výkon [dBm]
Volně mimo rozvaděč	27	-59
V otevřeném rozvaděči	20	-73
V uzavřeném rozvaděči	15	-83

Měření bylo realizováno v Jihlavě, v místě s výborným GSM signálem, což je také patrné z hodnoty přijímaného výkonu při umístění antény mimo rozvaděč vozidla. S umístěním antény do rozvaděče byl zaznamenán pokles až o 24 dBm na hodnotu -83 dBm. Pro hůře situované oblasti z hlediska úrovně přijímaného výkonu tak existuje důvodné podezření, že by signál nemusel být dostatečný. Hodnota 95 dBm je podle tabulky již nedostatečný přijímaný výkon. Bylo by tedy vhodné umístit GSM anténu mimo samotný rozvaděč vozidla.

Během testování se několikrát ukázalo jako nedostatečné nepájené propojení modulů v rámci sendvičové koncepce. Nahodile docházelo v krátkých okamžicích k chybovému přenosu dat, který se projevil v čase čtení z externí paměti SRAM, ale chybový přenos mohl vzniknout již při zápisu. Konstrukční řešení s touto nectností by však pro výsledný výrobek nepřipadalo v úvahu. Jednotlivé moduly by musely být nahrazeny diskrétními součástkami umístěnými na společnou DPS.

Webová aplikace fungovala korektně. Zobrazovala polohu vozidla a rychlost jízdy z GPS. Úspěšně bylo otestováno jednorázové vyčítání dat i jejich trvalé periodické zasílání. V reakci na uměle vyvolanou závadu byla také její přítomnost správně detekována a data odeslána na server, což dokládá následující zobrazení webové stránky. Při déle trvajícím provozu s vyčítáním dat z externí paměti SRAM pomocí sběrnice SPI s frekvencí 14 MHz docházelo k již míněným výpadkům.



Obrázek 42 Data na webu s uměle vyvolanou závadou na straně vozidla

Testování probíhalo jeden den od brzkých dopoledních hodin do pozdního odpoledne. Během toho času se podařilo na server nalogovat velké množství reálných dat. V kuse byly na server logovány až zhruba hodinové záznamy dat se zprávami CAN.

9 ZÁVĚR

Na základě zadání byla vytvořena hardwarová vozidlová část a softwarové řešení vozidlové části. Dále došlo k vytvoření webové aplikace, která byla následně zkušebně nasazena na server a společně testována s vozidlovou částí jako jeden celek. Podařilo se sestavit funkční prototyp vzdálené diagnostiky vozidla, na základě kterého byla načerpána řada zkušeností již během návrhu systému a posléze během testování, které se také prolínalo s přijímáním dílčích změn softwarové části na základě zjištění. Vytvořený systém by tak mohl být považován za zdařilý prototyp pro vytvoření finálního produktu.

Komerční systémy vzdálené diagnostiky pak více pracují se vstupními daty a nabízí komplexnější řešení s přesahem například do účetnictví (např. vedení knihy jízd). Standardně nabízí výpočty ujetých vzdáleností, měření spotřeby vozidel a podobně. S daty párují nejen vozidla, ale také řidiče, pro které je ve vozidle připravena čtečka jednoho z typů identifikátorů. Při vydání se cestou tímto směrem by bylo nejdříve nutné stanovit si veškeré požadavky na systém a zohlednit všechny aspekty možných podob řešení systému. Špatný počáteční návrh může přinést nepřekonatelné problémy v budoucnosti, které nakonec mohou vyústit do vývoje nového systému zase od nuly.

Seznam použité literatury

1. MAŠEK, Zdeněk. *Komunikace s diagnostickým displejem SM na MUV75*. Pardubice, 2018.
2. MAŠEK, Zdeněk. *Komunikační protokol SAE J1939*. In: . Univerzita Pardubice DFJP, KEEZ, s. 79.
3. SELECKÝ, Matúš. *Arduino: Uživatelská příručka*. Brno: Computer Press, 2016. ISBN 9788025148495.
4. HEROUT, Pavel. Učebnice jazyka C. 3. vyd., upr. České Budějovice: Kopp, 1994. ISBN 80-85828-21-9.
5. HEROUT, Pavel. Učebnice jazyka C. 2. díl. České Budějovice: Kopp, 1999. ISBN 80-85828-50-2.
6. Voss, W. *A Comprehensive Guide To Controller Area Network*. Copperhill Media Corporation, pp. 164, 2005, ISBN 0976511606.
7. VOSS, Wilfried. *A comprehensible guide to J1939*. Greenfield, Mass: Copperhill Technologies Corporation, 2008. ISBN 09-765-1163-0.
8. SAE *J1939-71 Vehicle Application Layer*. SAE International.
9. ODELL, Den a Ondřej BAŠE. *JavaScript: průvodce programováním ajaxových aplikací*. Brno: Computer Press, 2010. ISBN 978-80-251-2733-9.
10. *Programování v jazyku C/C++* [online]. Praha: Petr Bílek [cit. 2018-05-10]. Dostupné z: https://www.sallyx.org/sally/c/
11. TUTORIÁL – UŽÍVÁNÍ HODIN REÁLNÉHO ČASU DS1307 A DS3231 S ARDUINEM. *Arduino.cz* [online]. 2015 [cit. 2018-05-10]. Dostupné z: https://arduino.cz/tutorial-uzivani-hodin-realneho-casu-ds1307-a-ds3231-s-arduinem/
12. PROGRAMOVÁNÍ WEBOVÝCH ROZHRANÍ PRO ARDUINO. *Arduino.cz* [online]. 2016 [cit. 2018-05-10]. Dostupné z: https://arduino.cz/programovani-webovych-rozhrani-pro-arduino/
13. Elecrow: 32u4 with A7 GPRS/GSM/GPS Board. *Arduino* [online]. 2017 [cit. 2018-05-10]. Dostupné z: https://forum.arduino.cc/index.php?topic=465057.0
14. TinyGSM [online]. [cit. 2018-05-10]. Dostupné z: https://github.com/vshymanskyy/TinyGSM
15. SIM808 GPS/GPRS/GSM Shield SKU: TEL0097. *DFRobot* [online]. [cit. 2018-05-10]. Dostupné z: https://www.dfrobot.com/wiki/index.php/SIM808_GPS/GPRS/GSM_Shield_SKU:_TEL0097
16. *GSM/GPRS/GPS Shield (B)* [online]. Waveshare Electronics [cit. 2018-05-10]. Dostupné z: https://www.waveshare.com/wiki/GSM/GPRS/GPS_Shield_(B)
17. MALÝ, Martin. Protokol MQTT: komunikační standard pro IoT. *Root.cz* [online]. 2016 [cit. 2018-05-10]. ISSN 1212-8309. Dostupné z: https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/
18. JAFFEY, Toby. MQTT and CoAP, IoT Protocols. *Eclipse* [online]. 2014 [cit. 2018-05-10]. Dostupné z: https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php
19. *Arduino Forum* [online]. Arduino [cit. 2018-05-10]. Dostupné z: https://forum.arduino.cc/
20. *Nette Framework* [online]. Arduino [cit. 2018-05-10]. Dostupné z: https://nette.org/

21. *Google Maps JavaScript API Tutorial* [online]. In: . Traversy Media, 2017 [cit. 2018-05-10]. Dostupné z: https://www.youtube.com/watch?v=Zxf1mnP5zcw
22. KHAN, Mudassar. *Google Maps V3: Delete all markers - Remove all markers - Clear all markers* [online]. 30. srpna 2014 [cit. 2018-05-10]. Dostupné z: https://www.aspsnippets.com/Articles/Google-Maps-V3-Delete-Remove-Clear-all-markers.aspx
23. *AT+CSQ – Signal quality* [online]. M2MSupport.net [cit. 2018-05-10]. Dostupné z: http://m2msupport.net/m2msupport/atcsq-signal-quality/
24. KARLSEN, Henning. *Library: DS3231* [online]. Rinky-Dink Electronics, **2014** [cit. 2018-05-10]. Dostupné z: http://www.rinkydinkelectronics.com/library.php?id=74
25. *ArduinoJson* [online]. [cit. 2018-05-10]. Dostupné z: https://arduinojson.org/
26. CHRISTENSEN, Jack. *Timezone* [online]. 2012 [cit. 2018-05-10]. Dostupné z: https://github.com/JChristensen/Timezone
27. KIDDER, Collin. *Due_can* [online]. [cit. 2018-05-10]. Dostupné z: https://github.com/collin80/due_can
28. MATTAIR, Justin. *SRAM_23LC* [online]. [cit. 2018-05-10]. Dostupné z: https://github.com/mattairtech/SRAM_23LC
29. JAHODA, Bohumil. JSON. *Je čas* [online]. 2016 [cit. 2018-05-10]. Dostupné z: http://jecas.cz/json

Seznam zkratek a značek

API – Application Programming Interface – programové rozhraní

CAN – Controller Area Network – sběrnice

Cookie – v HTTP informace ukládané do webového prohlížeče, které se zasílají na server spolu s dotazy

CSS – Cascading Style Sheets – jazyk pro popis způsobu zobrazení HTML

DPS – deska plošných spojů

GNSS – Global Navigation Satellite Systém – globální navigační satelitní systém (obecně)

GPS – Global Positioning Systém – globální polohovací systém (název typu GNSS)

HTML – HyperText Markup Language – značkovací jazyk pro tvorbu webových stránek

IoT – Internet of Things – internet věcí

JSON – JavaScript Object Notation – standard formátového zápisu dat

MQTT – Message Queuing Telemetry Transport – komunikační protokol

MySQL – databázový systém pro web komunikující jazykem SQL

PGN – Parameter Group Number – číslo skupiny parametrů

PHP – Hypertext Preprocessor – skriptovací programovací jazyk

QoS – Quality of Service – způsob řízení datových toků

REST – Representational State Transfer – architektura komunikačního rozhraní

Routování – směrování, v souvislosti s Nette, změna podoby URL při stejné podobě presenterů

SEČ – středoevropský čas

SELČ – středoevropský letní čas

Session – v HTTP cookie obsahující identifikaci uživatele

Shield – deska – v souvislosti s Arduinem kompatibilní pro snadné připojení k Arduinu

SQL – Structured Query Language – strukturovaný dotazovací jazyk pro relační databáze

VPS – Virtual Private Server – virtuální server

Seznam obrázků

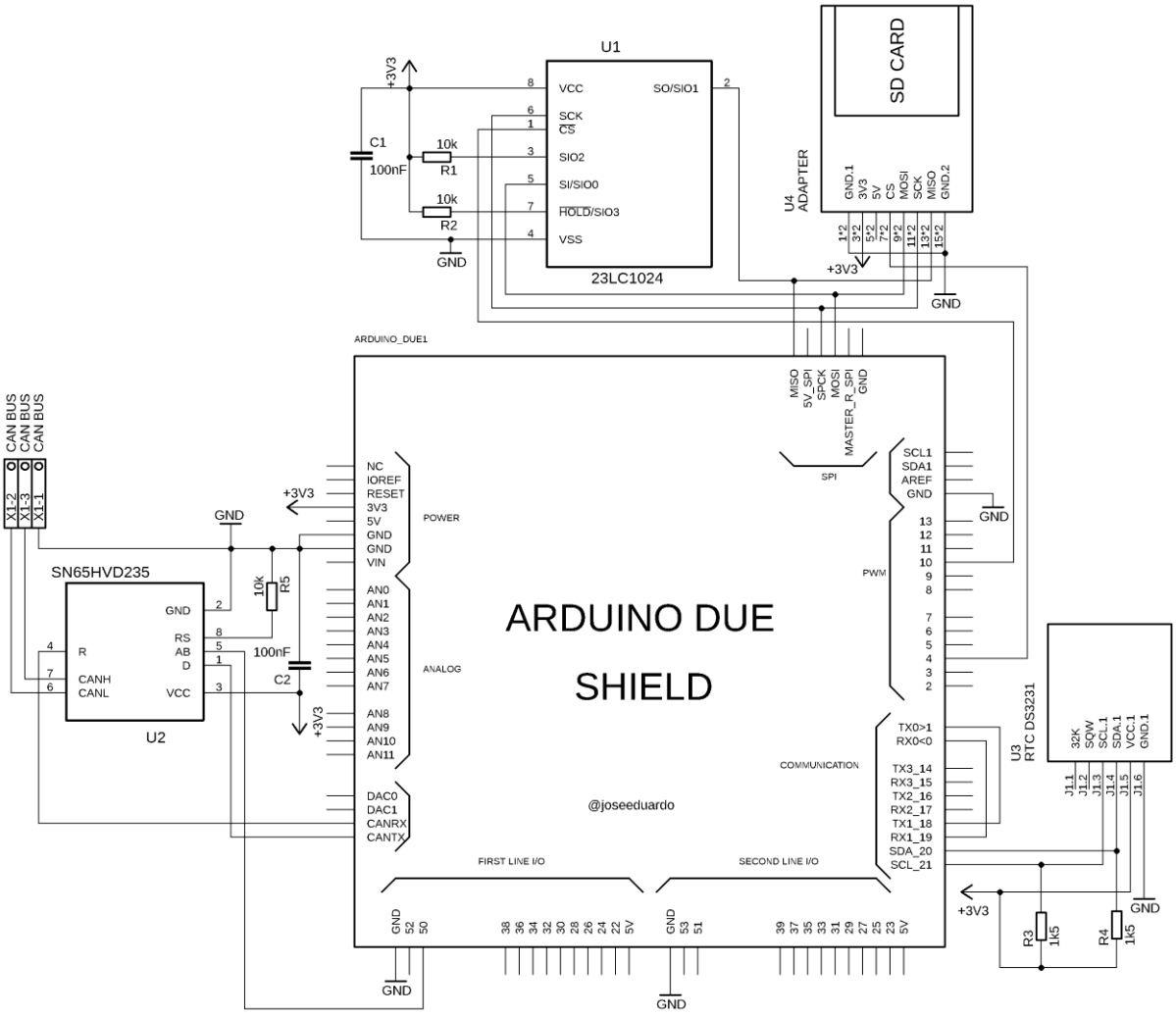
Seznam tabulek

Seznam zdrojových kódů

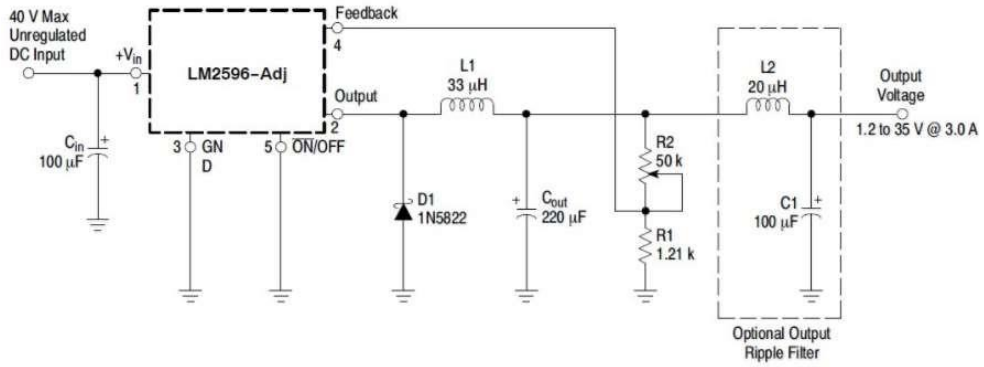
Seznam vývojových diagramů

Seznam příloh

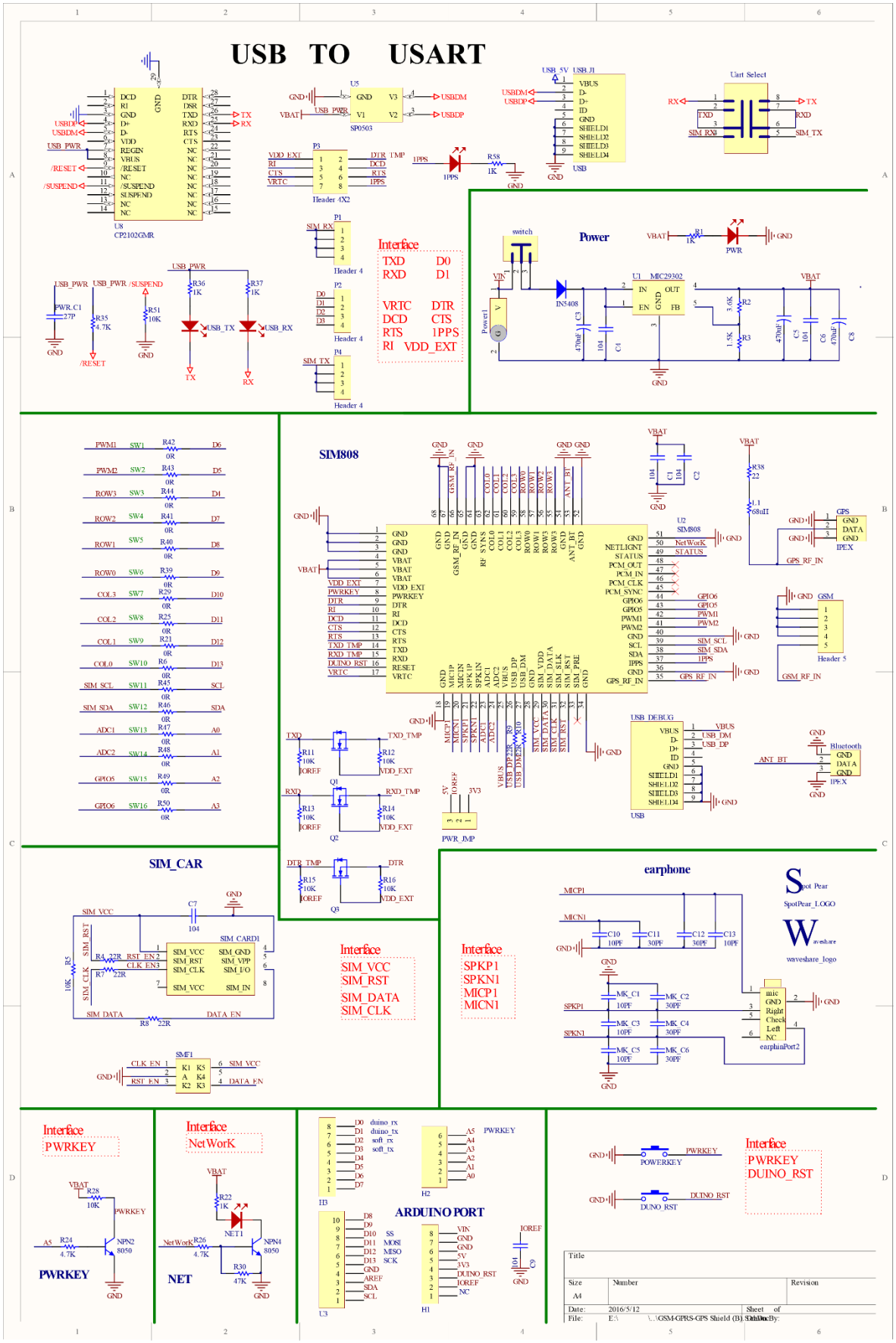
Příloha A Schéma navržené a vyrobené DPS



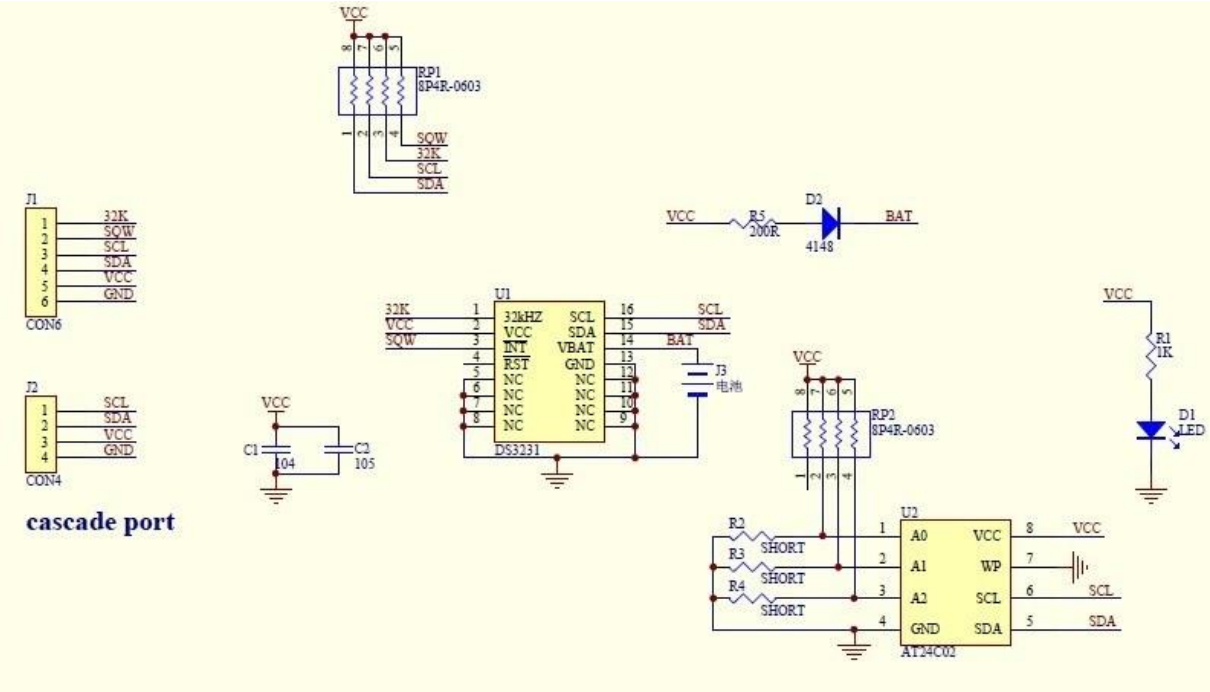
Příloha B Schéma DC-DC step-down měniče s LM2956



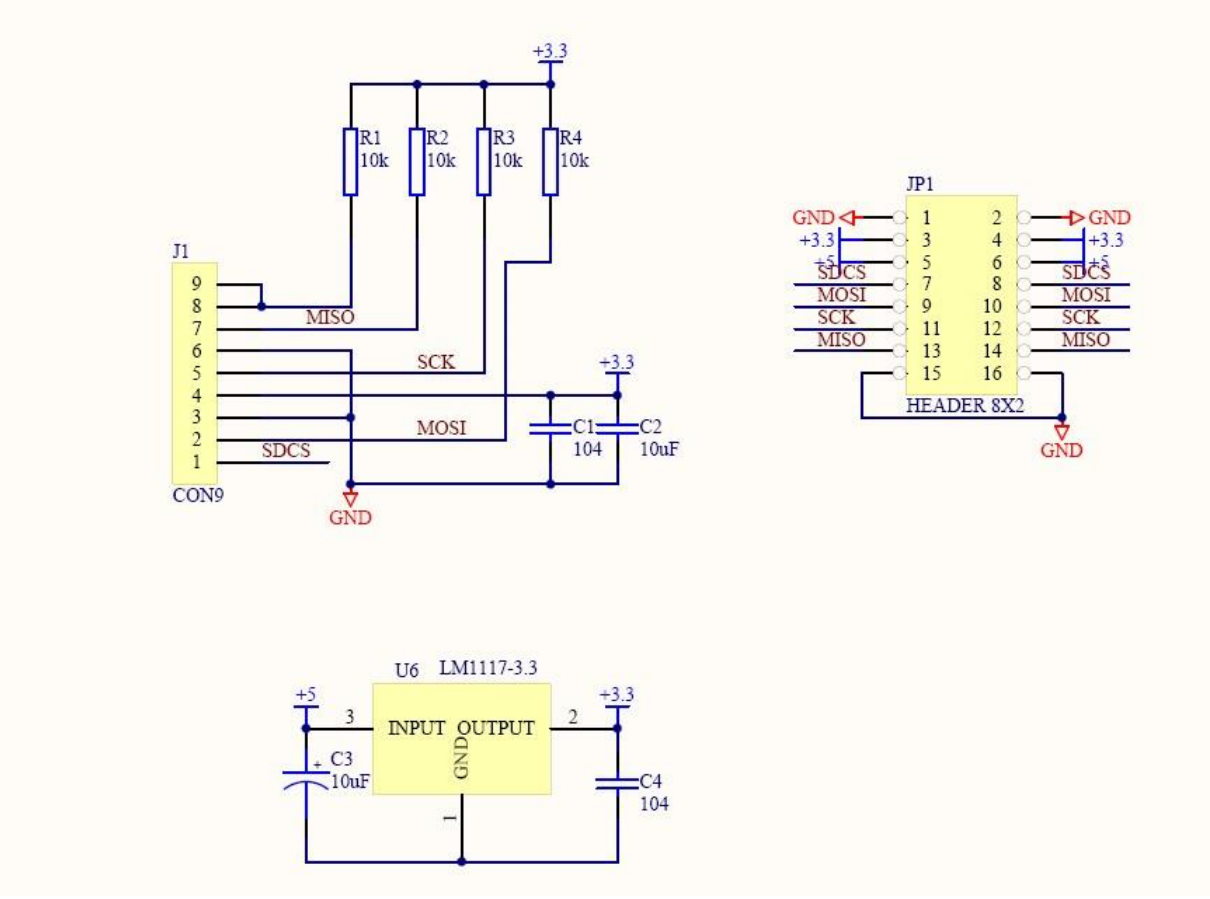
Příloha C Schéma obvodu SIM808



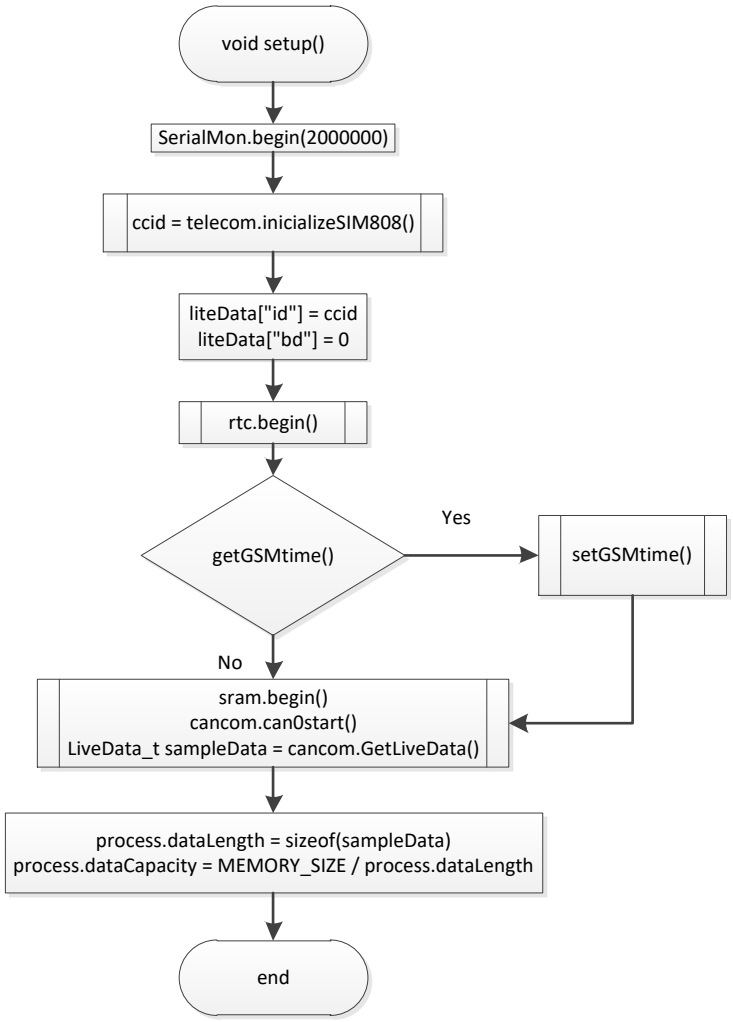
Příloha D Schéma obvodu RTC (DS3231 a AT24C32)



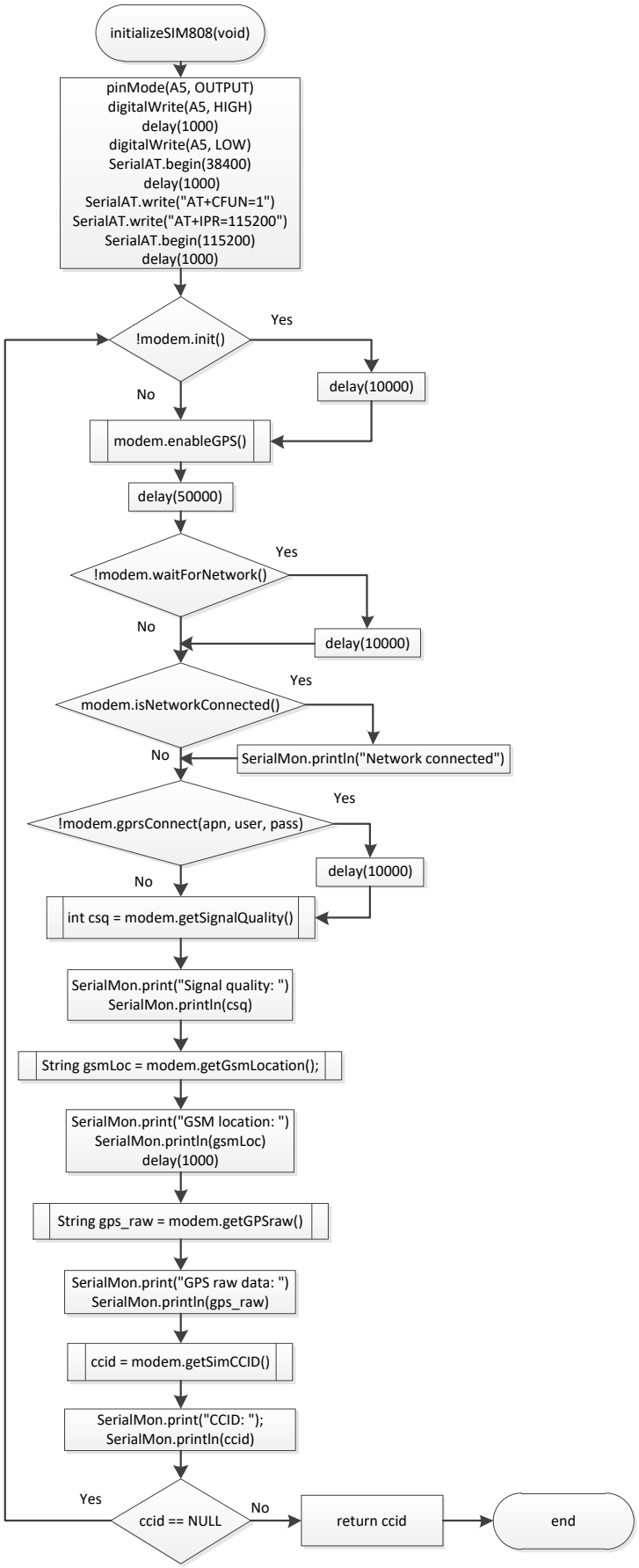
Příloha E Schéma čtečky SD karet



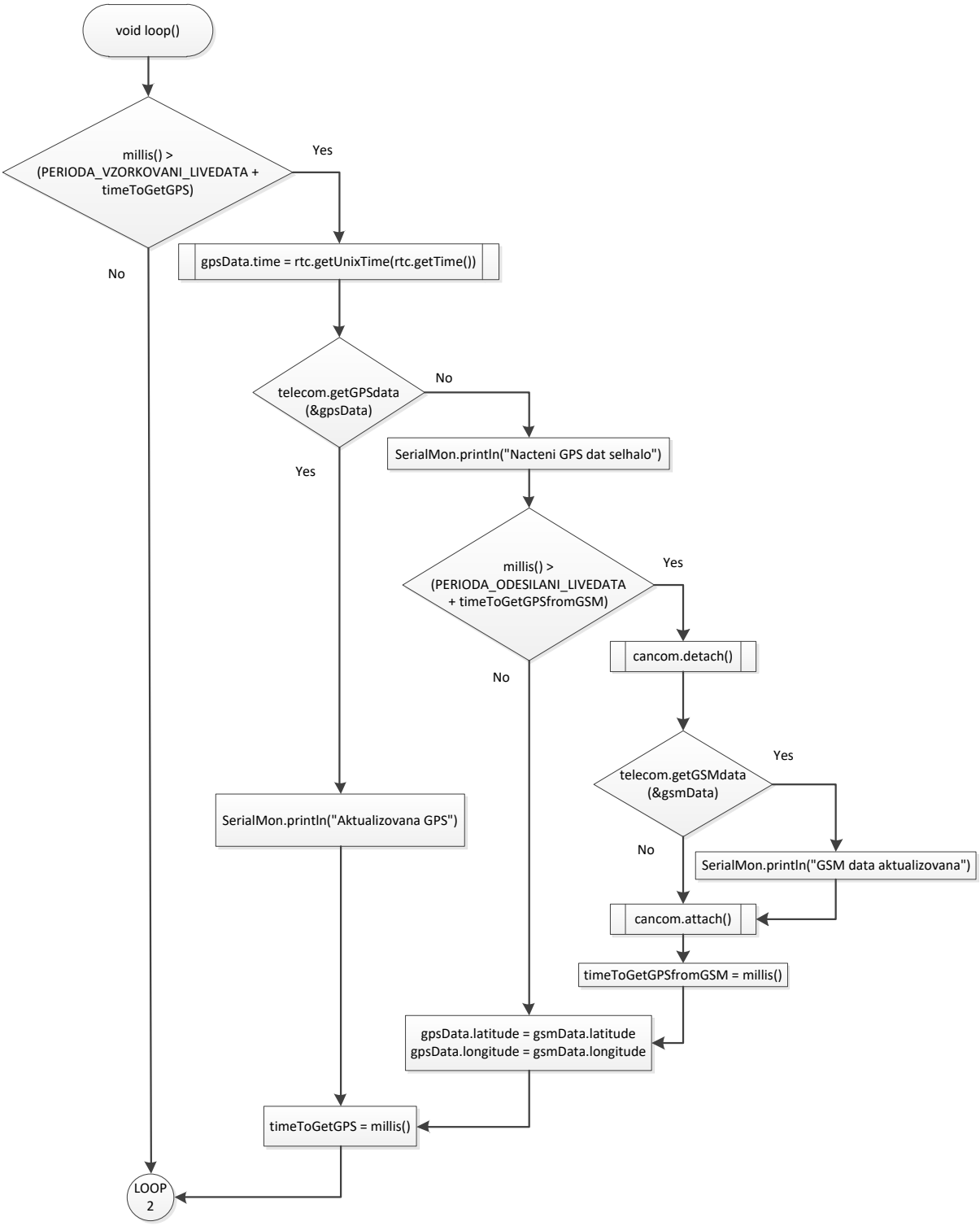
Příloha F Mobilní část, funkce setup()

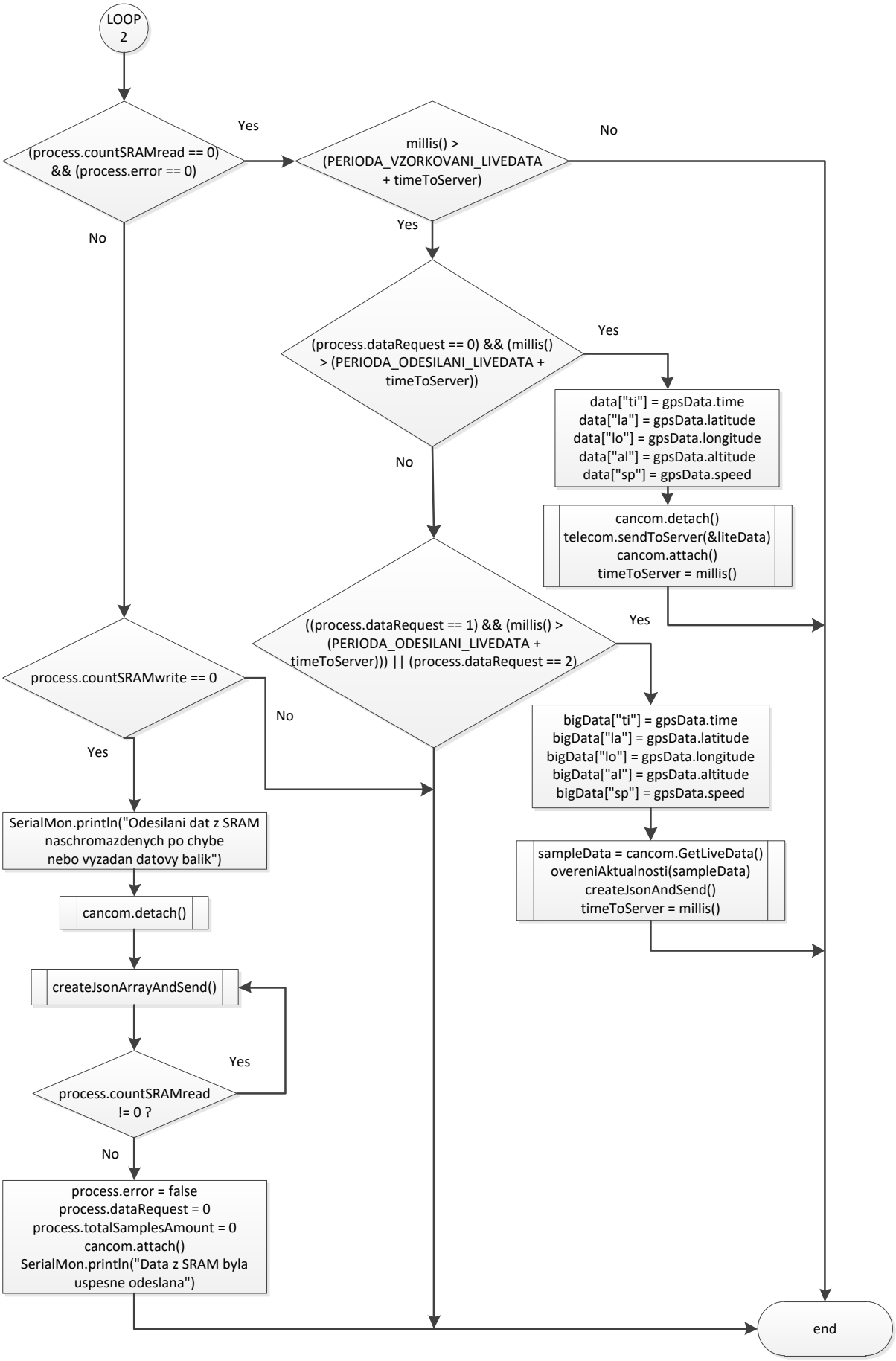


Příloha G Mobilní část, TeleComModul::initializeSIM808(void)

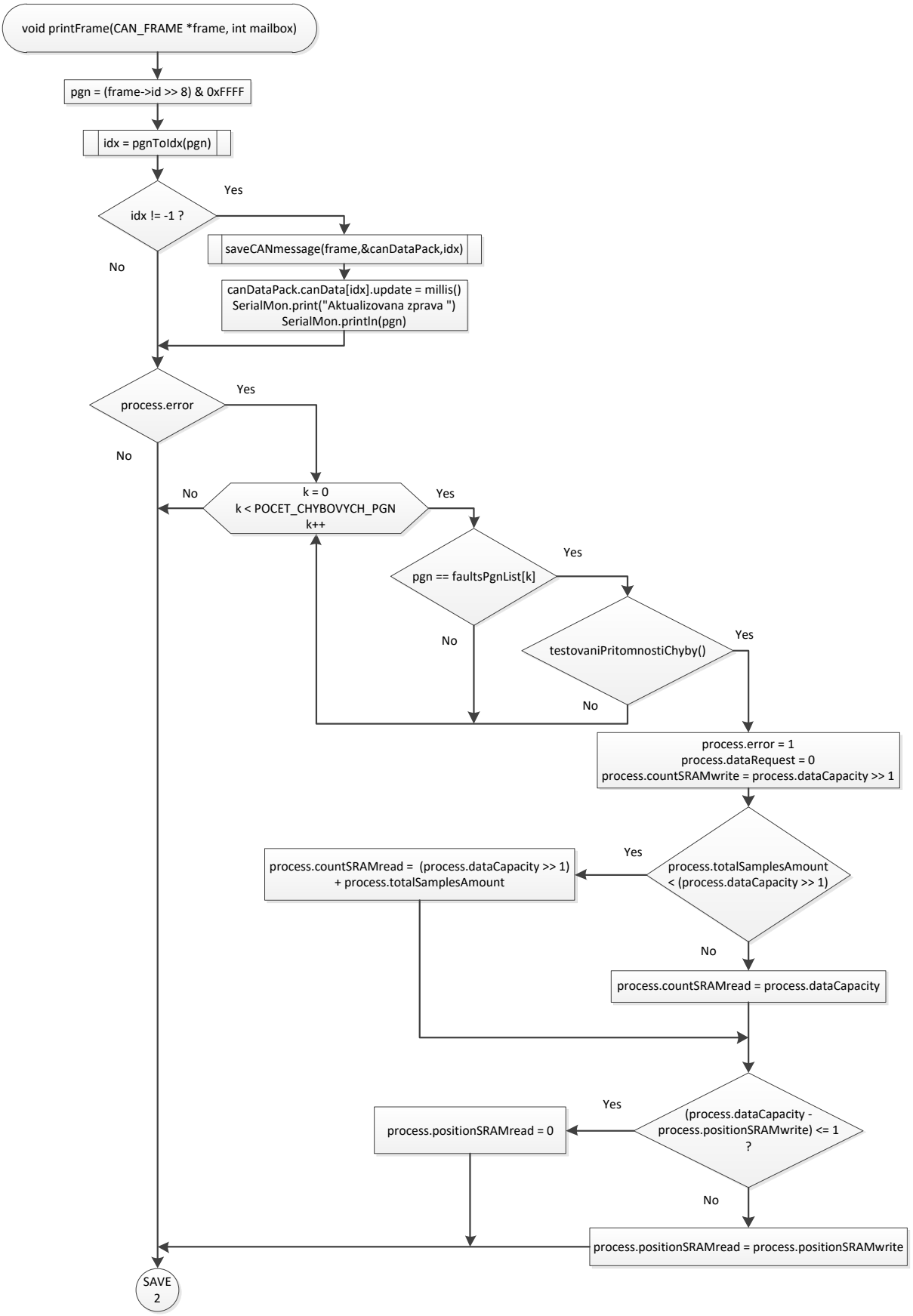


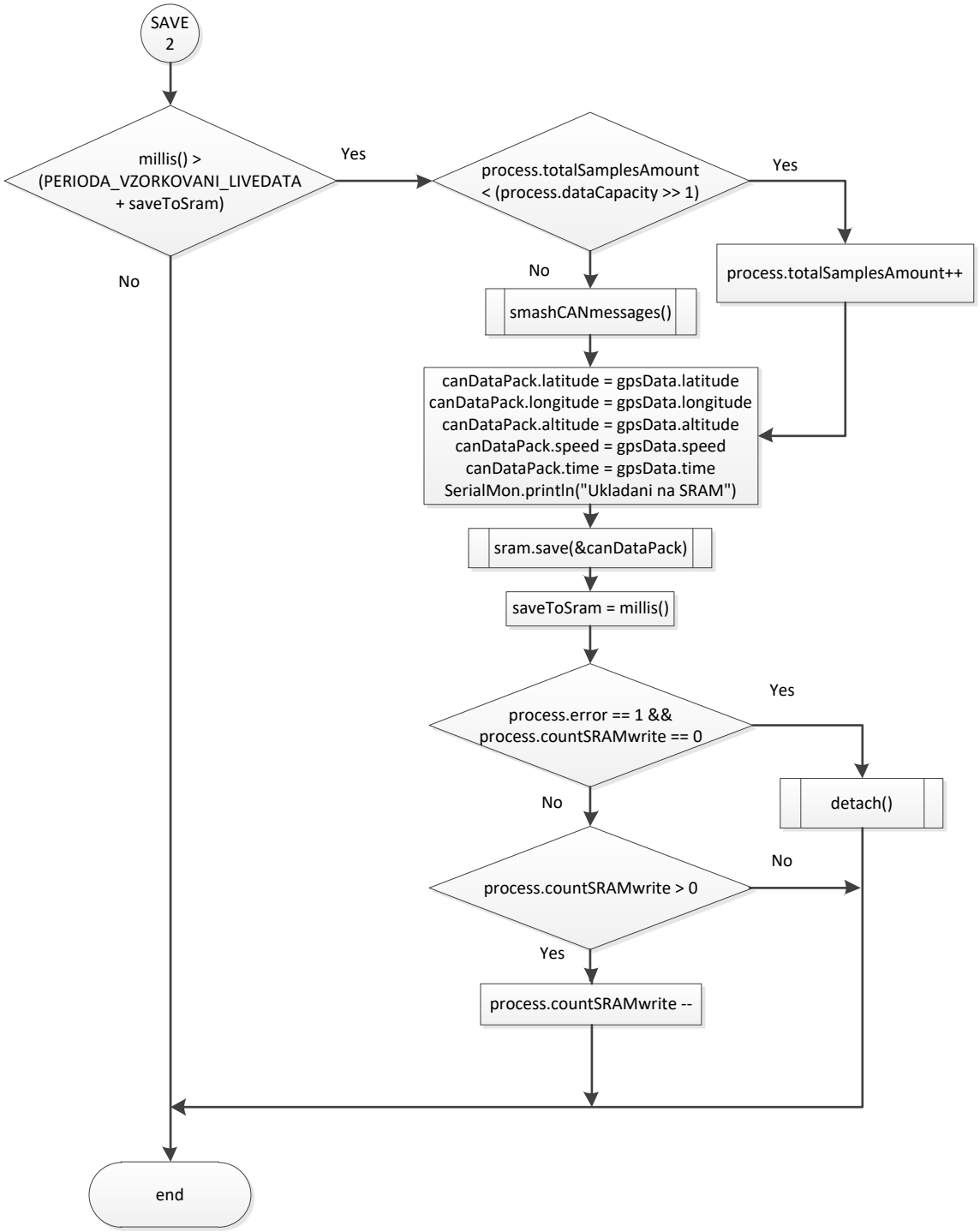
Příloha H Vozidlová část, funkce loop()



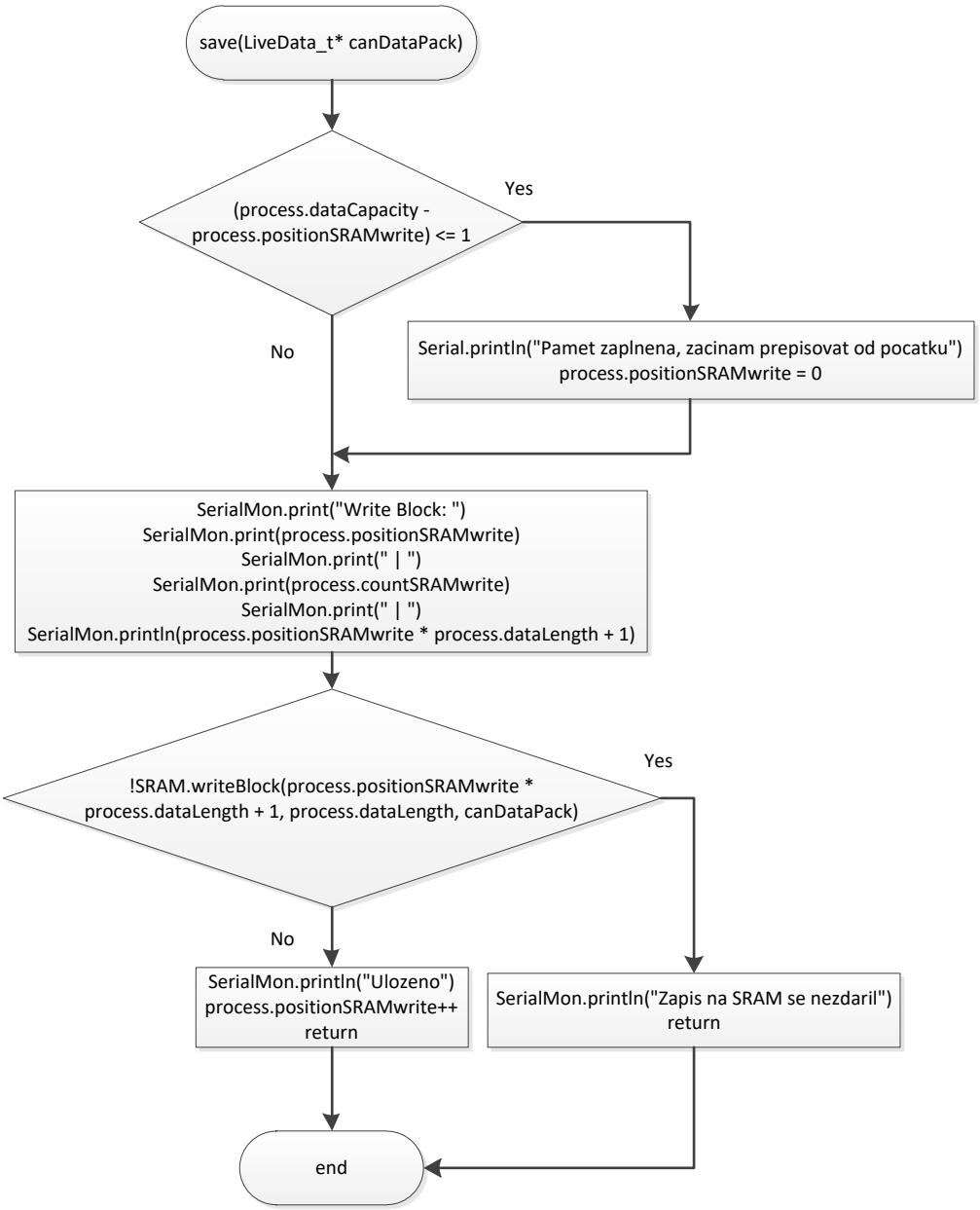

LOOP 2
(process.countSRAMread == 0) && (process.error == 0)
Yes
No
millis() > (PERIODA_VZORKOVANI_LIVEDATA + timeToServer)
Yes
No
(process.dataRequest == 0) && (millis() > (PERIODA_ODESILANI_LIVEDATA + timeToServer))
Yes
No
data["ti"] = gpsData.time
data["la"] = gpsData.latitude
data["lo"] = gpsData.longitude
data["al"] = gpsData.altitude
data["sp"] = gpsData.speed
cancom.detach()
telecom.sendToServer(&liteData)
cancom.attach()
timeToServer = millis()
((process.dataRequest == 1) && (millis() > (PERIODA_ODESILANI_LIVEDATA + timeToServer))) || (process.dataRequest == 2)
Yes
No
bigData["ti"] = gpsData.time
bigData["la"] = gpsData.latitude
bigData["lo"] = gpsData.longitude
bigData["al"] = gpsData.altitude
bigData["sp"] = gpsData.speed
sampleData = cancom.GetLiveData()
overeniAktualnosti(sampleData)
createJsonAndSend()
timeToServer = millis()
process.countSRAMwrite == 0
Yes
No
SerialMon.println("Odesilani dat z SRAM naschromazdenych po chybe nebo vyzadan datovy balik")
cancom.detach()
createJsonArrayAndSend()
process.countSRAMread != 0 ?
Yes
No
process.error = false
process.dataRequest = 0
process.totalSamplesAmount = 0
cancom.attach()
SerialMon.println("Data z SRAM byla uspesne odeslana")
end

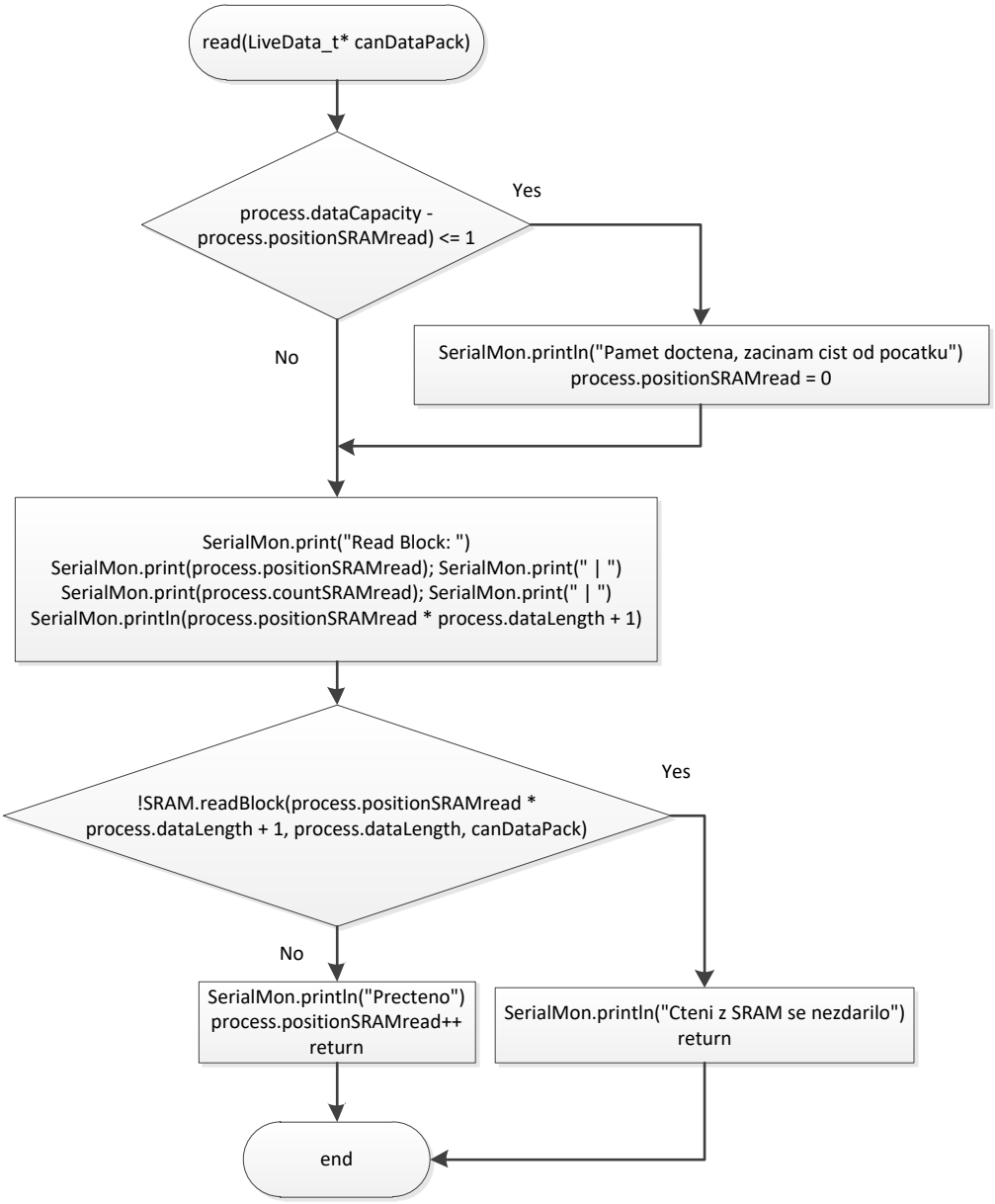
Příloha I Vozidlová část, CanBus::printFrame()



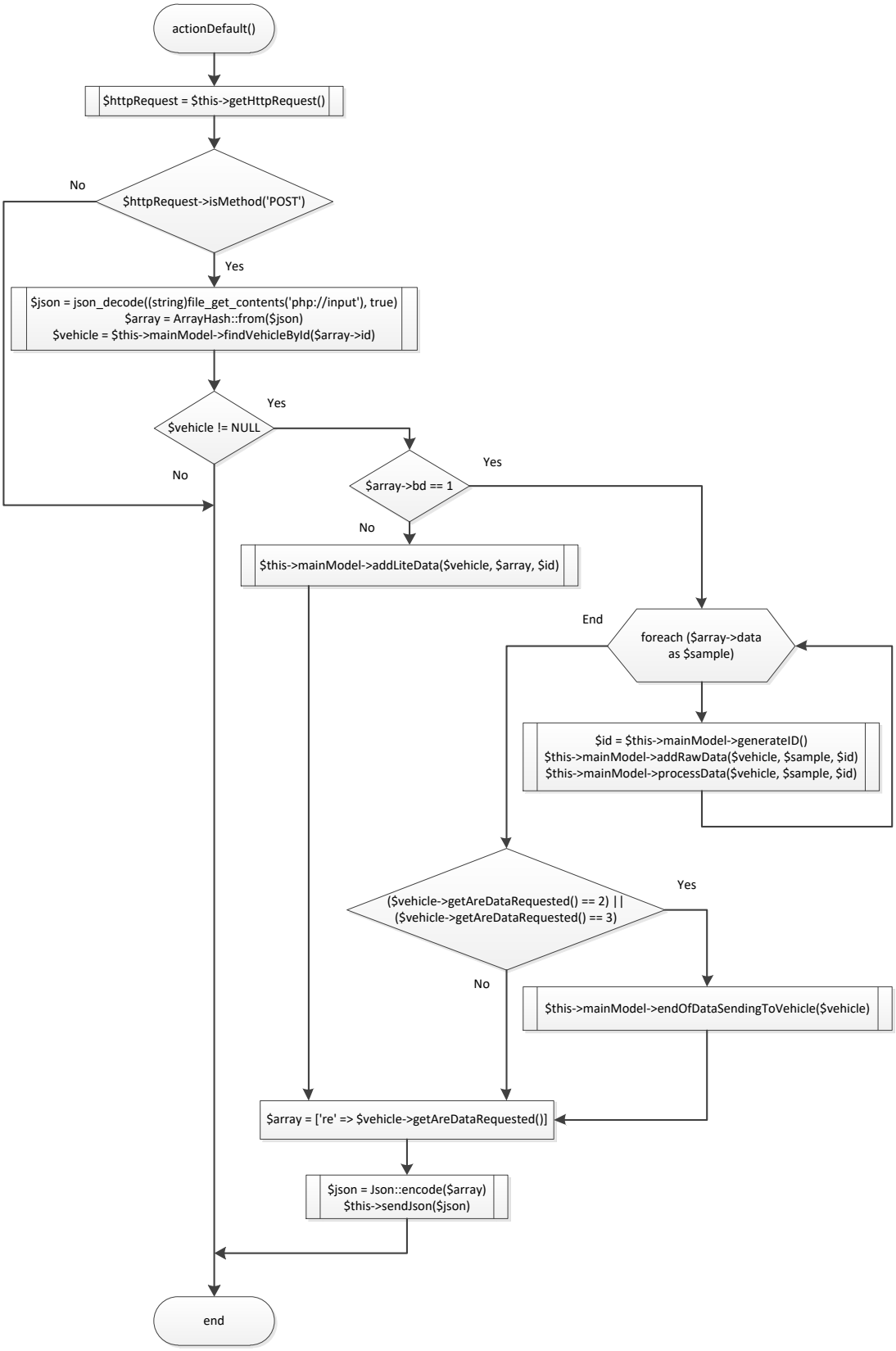

SAVE 2
millis() > (PERIODA_VZORKOVANI_LIVEDATA + saveToSram)
Yes
No
process.totalSamplesAmount < (process.dataCapacity >> 1)
Yes
process.totalSamplesAmount++
No
smashCANmessages()
canDataPack.latitude = gpsData.latitude
canDataPack.longitude = gpsData.longitude
canDataPack.altitude = gpsData.altitude
canDataPack.speed = gpsData.speed
canDataPack.time = gpsData.time
SerialMon.println("Ukladani na SRAM")
sram.save(&canDataPack)
saveToSram = millis()
process.error == 1 && process.countSRAMwrite == 0
Yes
detach()
No
process.countSRAMwrite > 0
No
Yes
process.countSRAMwrite --
end

Příloha J Webová aplikace, SramMemory::save() a read()

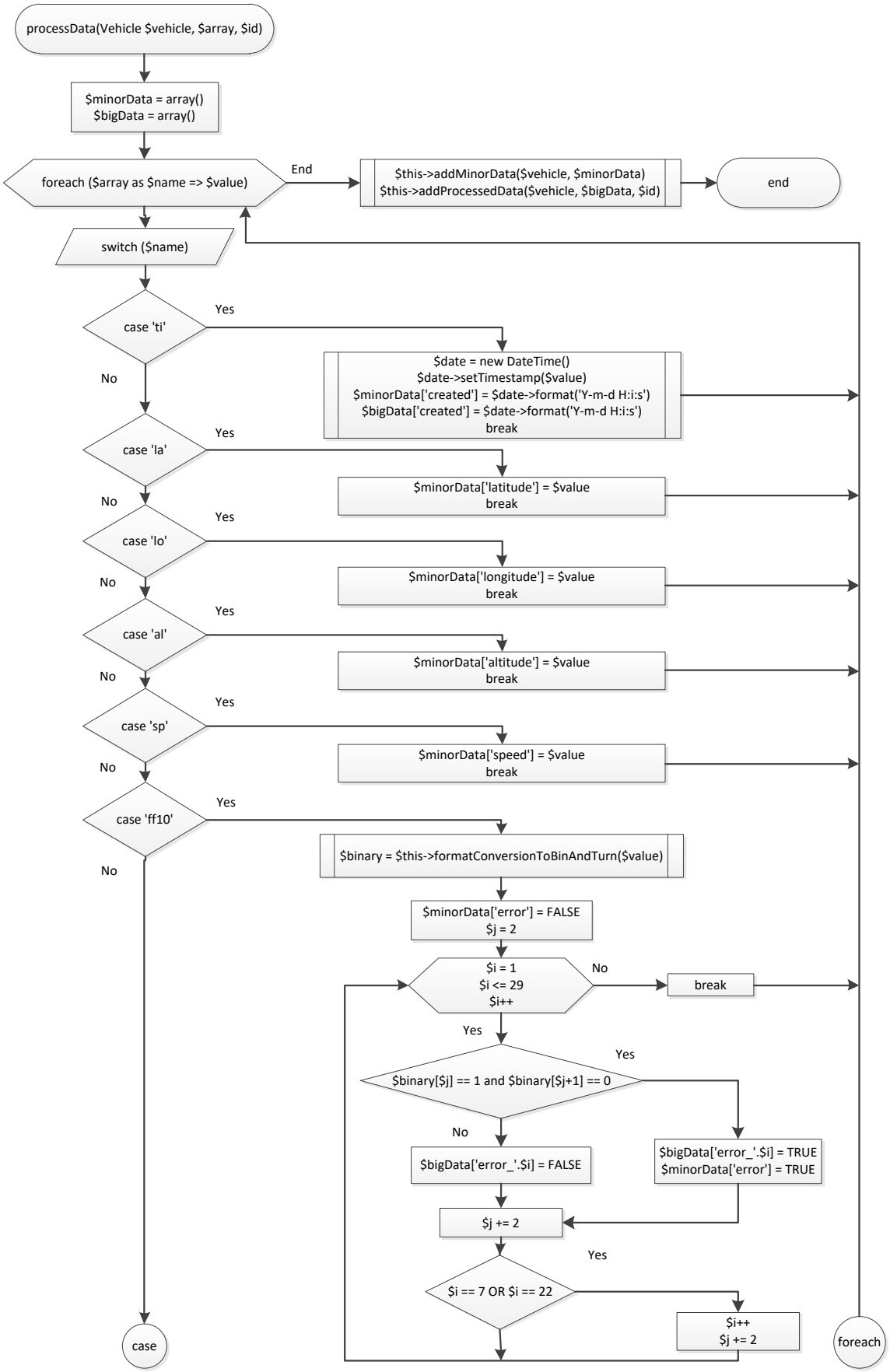


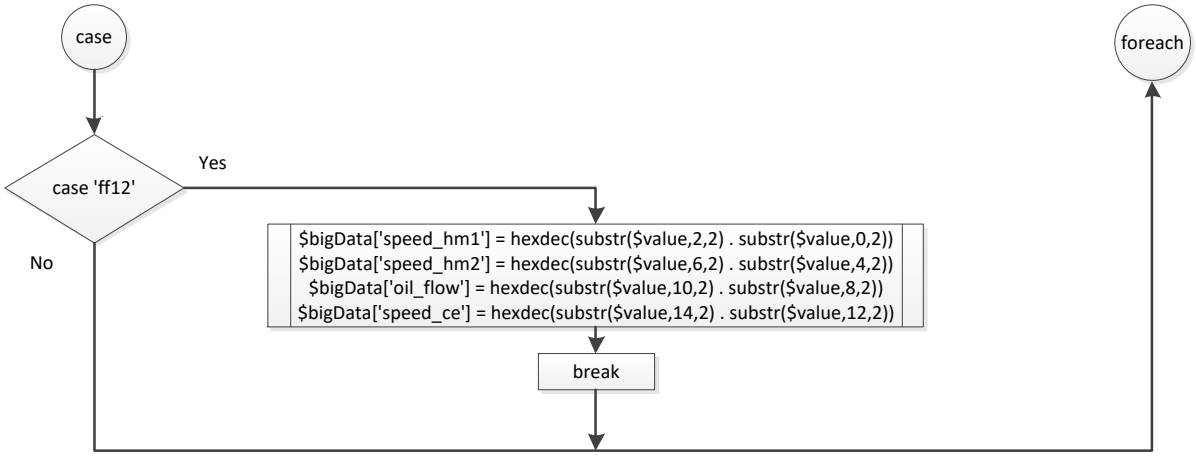

read(LiveData_t* canDataPack)
process.dataCapacity - process.positionSRAMread) <= 1
Yes
No
SerialMon.println("Pamet doctena, zacinam cist od pocatku")
process.positionSRAMread = 0
SerialMon.print("Read Block: ")
SerialMon.print(process.positionSRAMread); SerialMon.print(" | ")
SerialMon.print(process.countSRAMread); SerialMon.print(" | ")
SerialMon.println(process.positionSRAMread * process.dataLength + 1)
!SRAM.readBlock(process.positionSRAMread * process.dataLength + 1, process.dataLength, canDataPack)
Yes
No
SerialMon.println("Precteno")
process.positionSRAMread++
return
SerialMon.println("Cteni z SRAM se nezdarilo")
return
end

Příloha K Webová aplikace, ImportPresenter:: actionDefault()

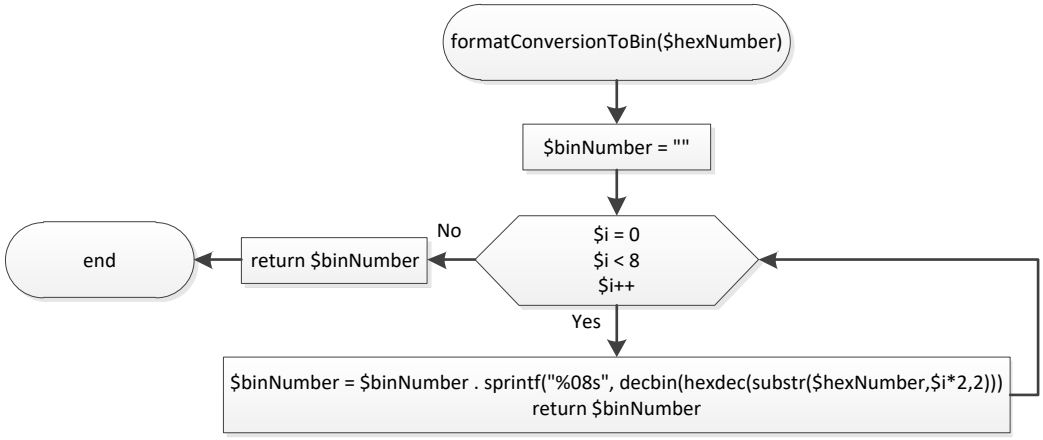


Příloha L Webová aplikace, MainModel::processData()

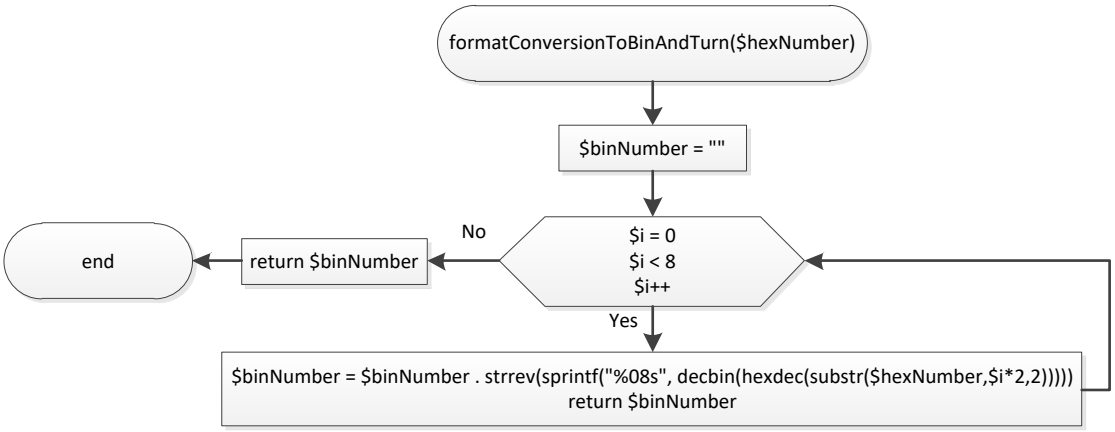




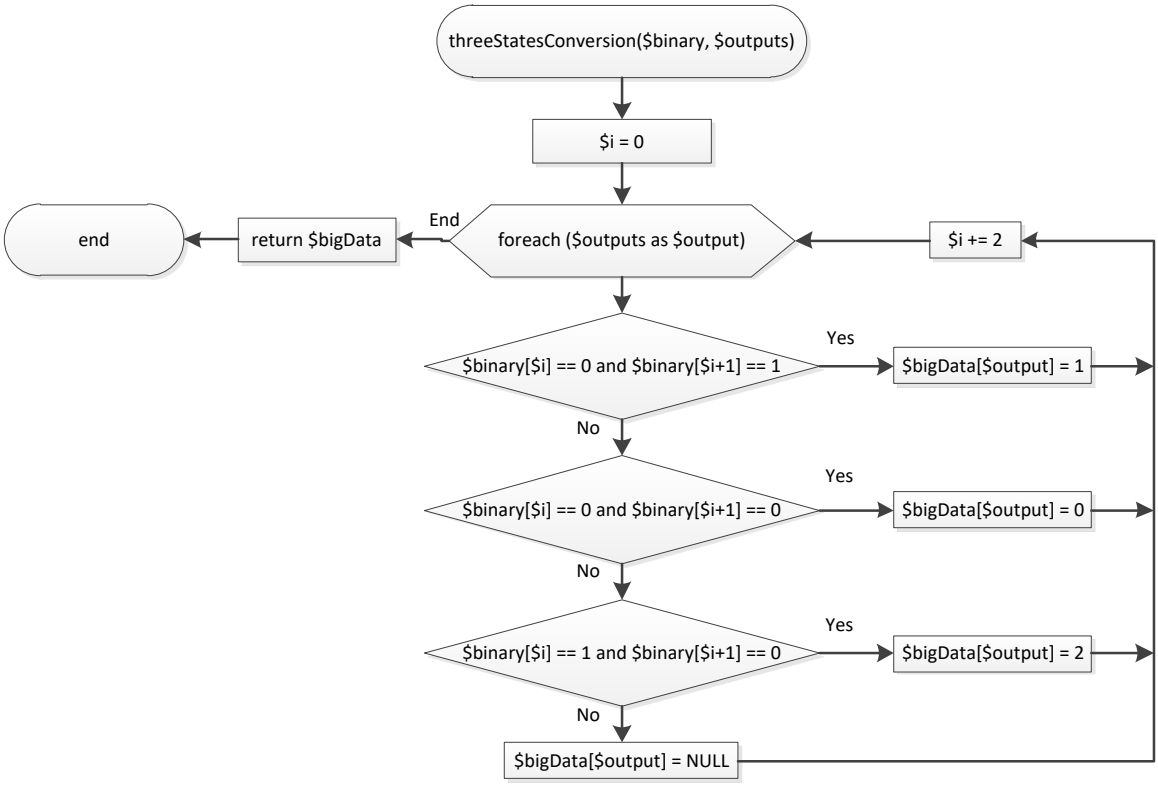
Příloha M Webová aplikace, MainModel::formatConversionToBin()



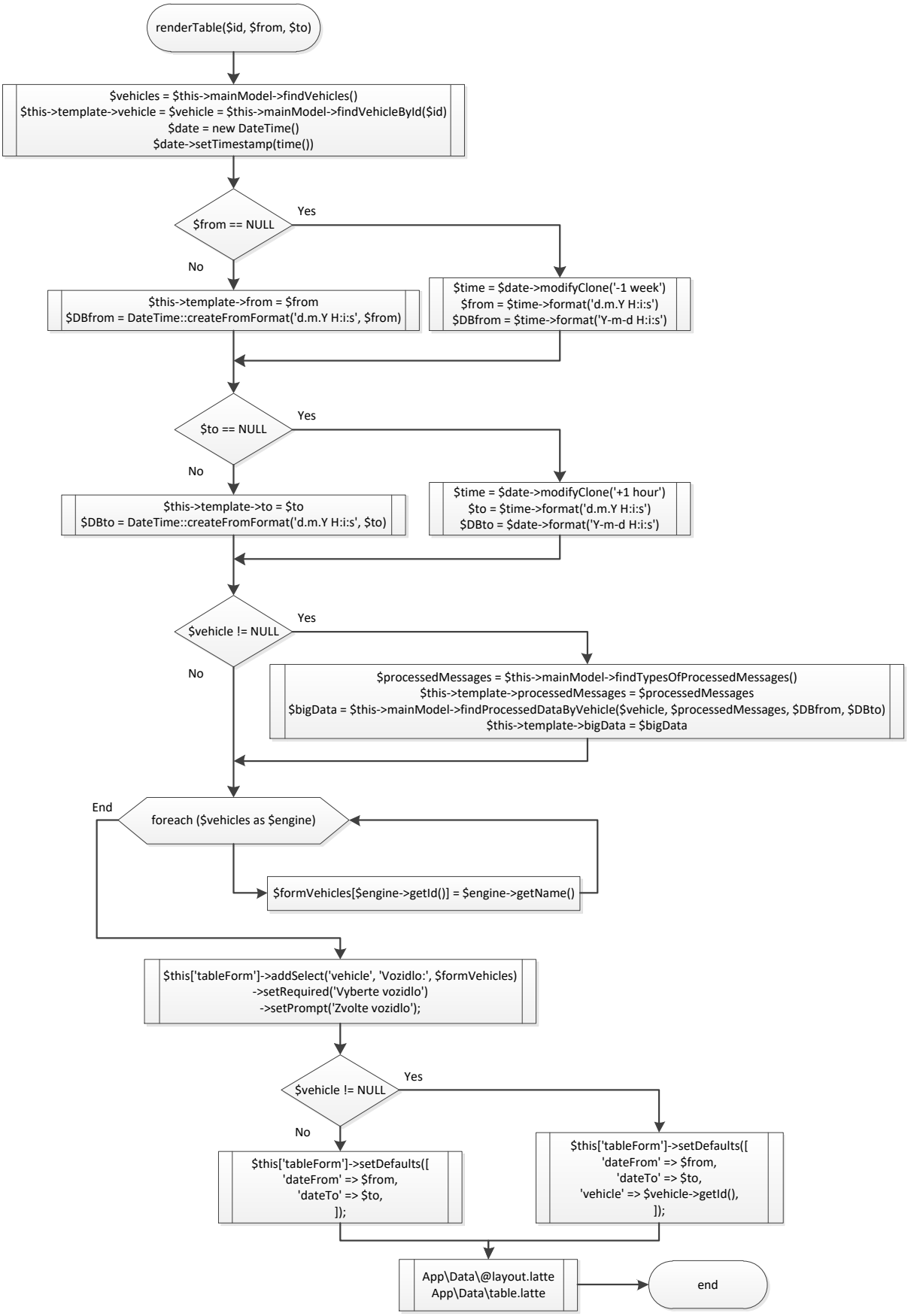
Příloha N Webová aplikace MainModel::formatConversionToBinAndTurn()



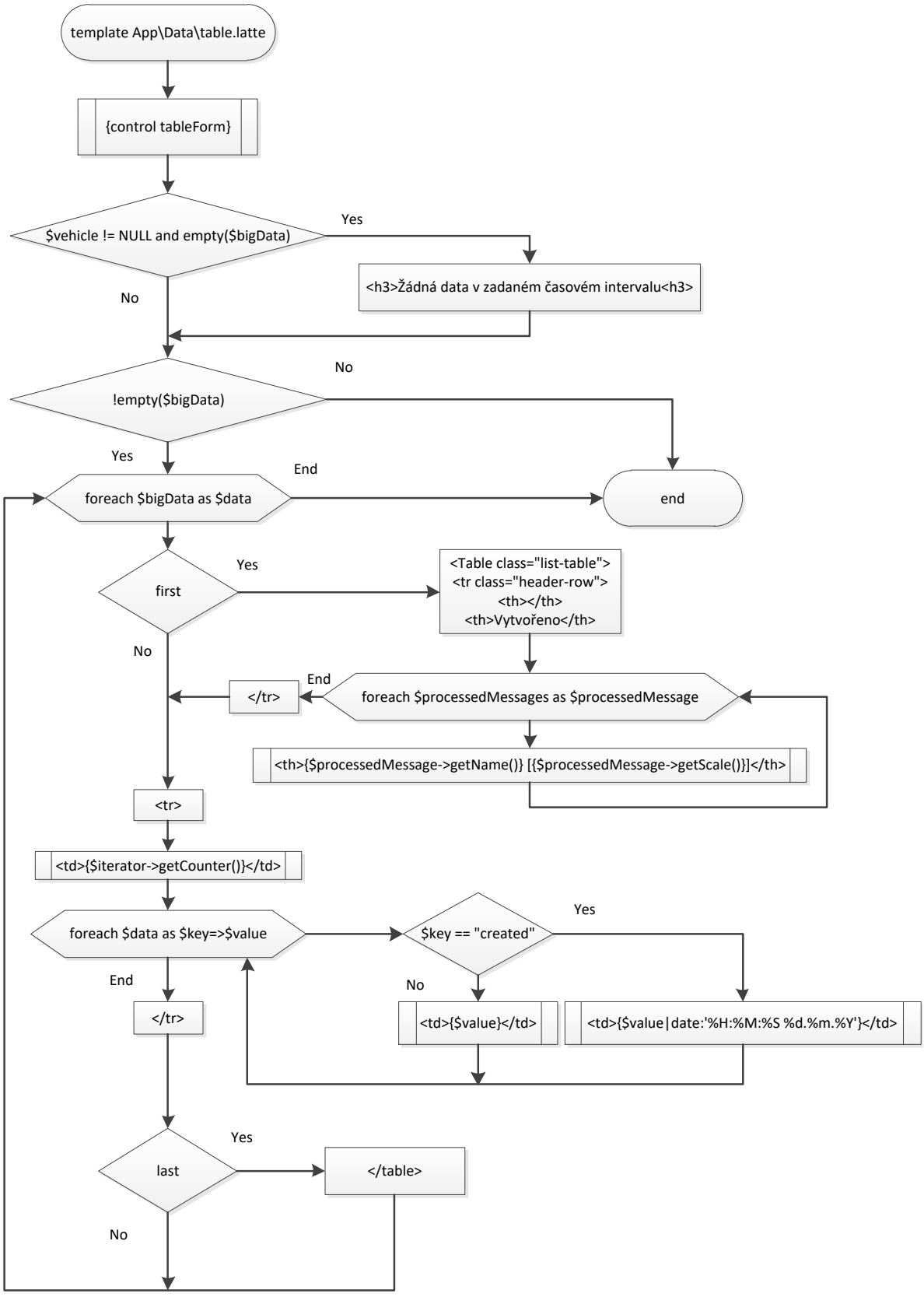
Příloha O Webová aplikace MainModel::threeStatesConversion()



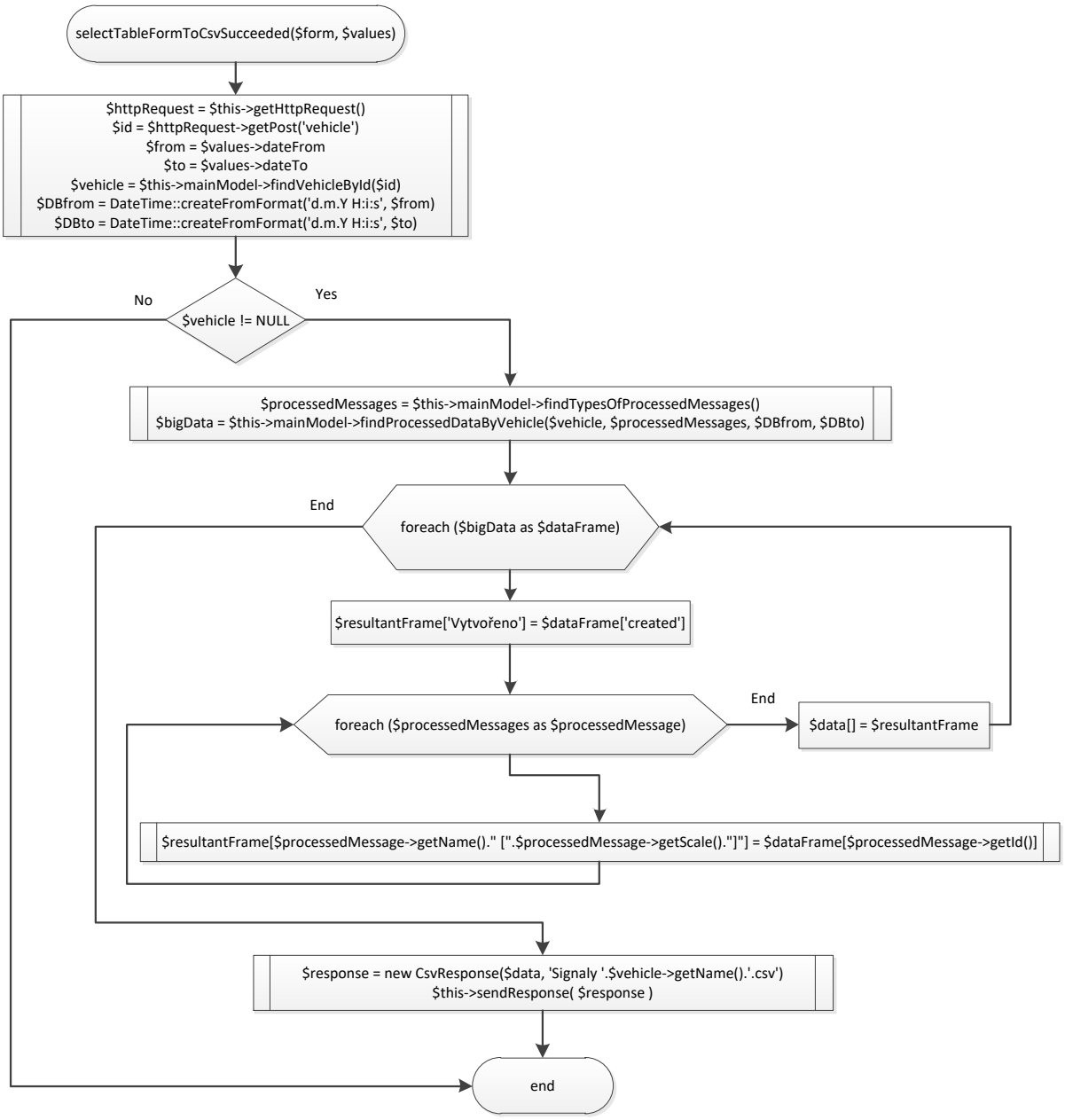
Příloha P Webová aplikace, DataPresenter::renderTable()



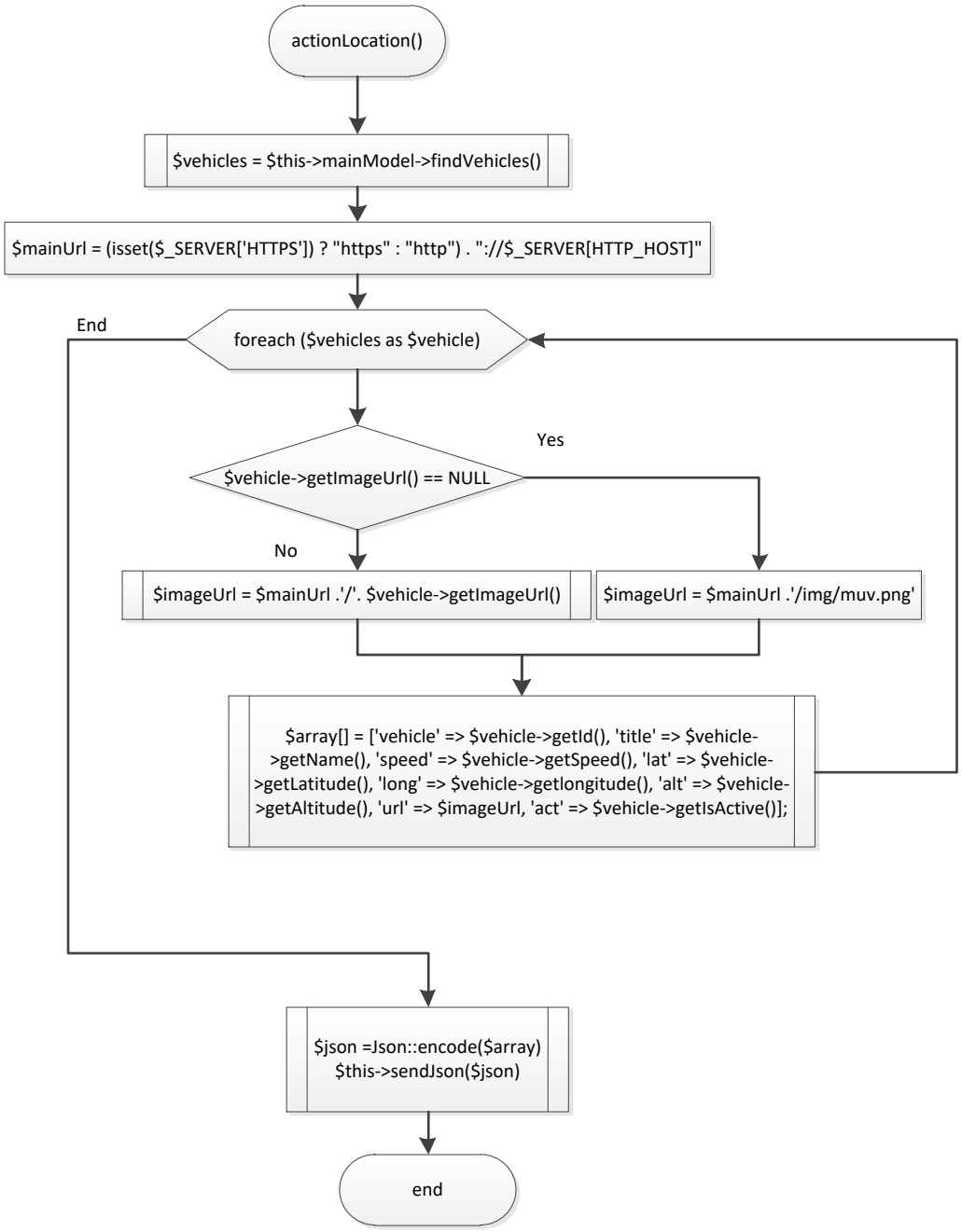
Příloha Q Webová aplikace, šablona ke stránce s tabulkou signálů



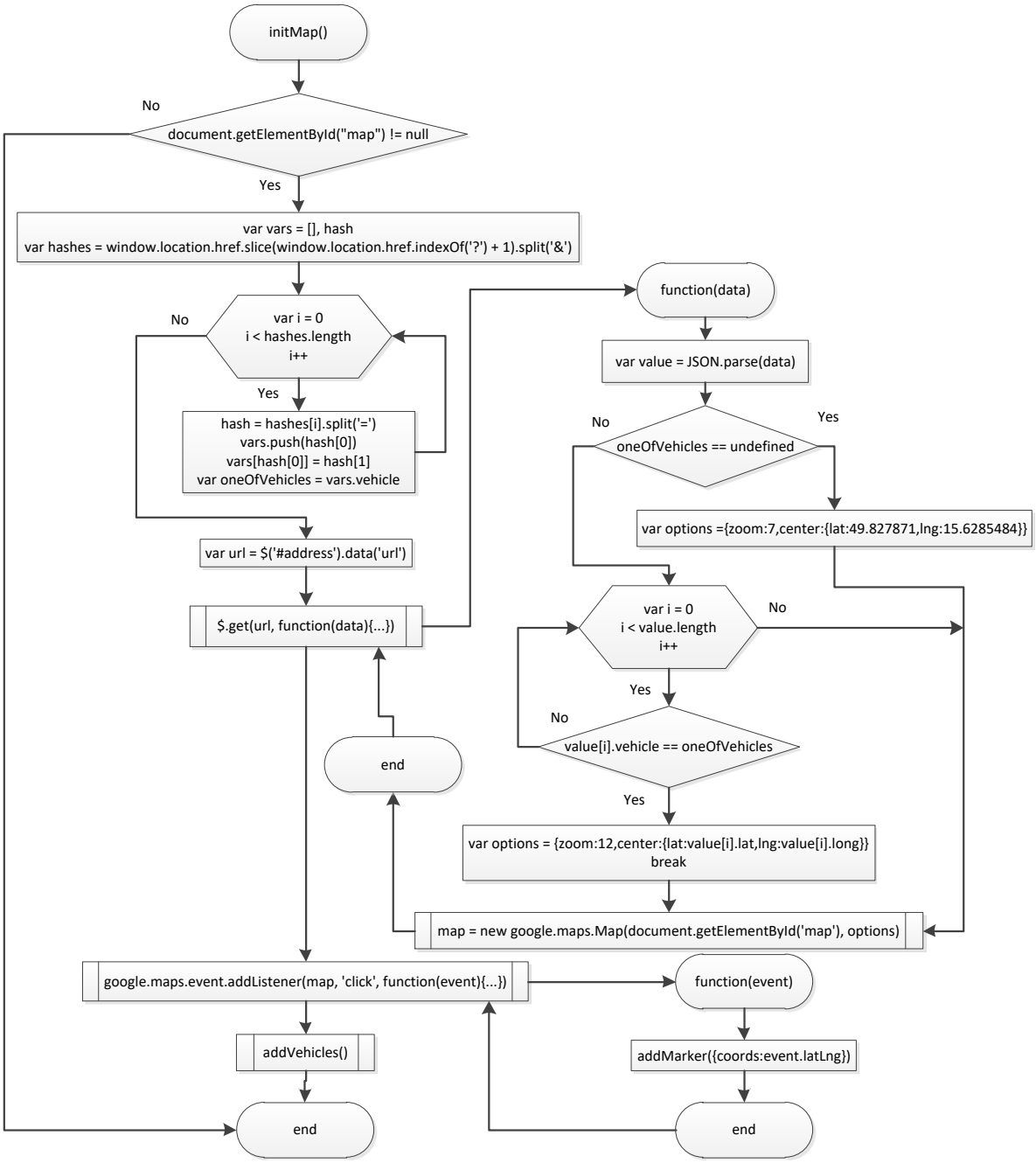
Příloha R Webová aplikace, DataPresenter::selectTableFormToCsvSucceeded()

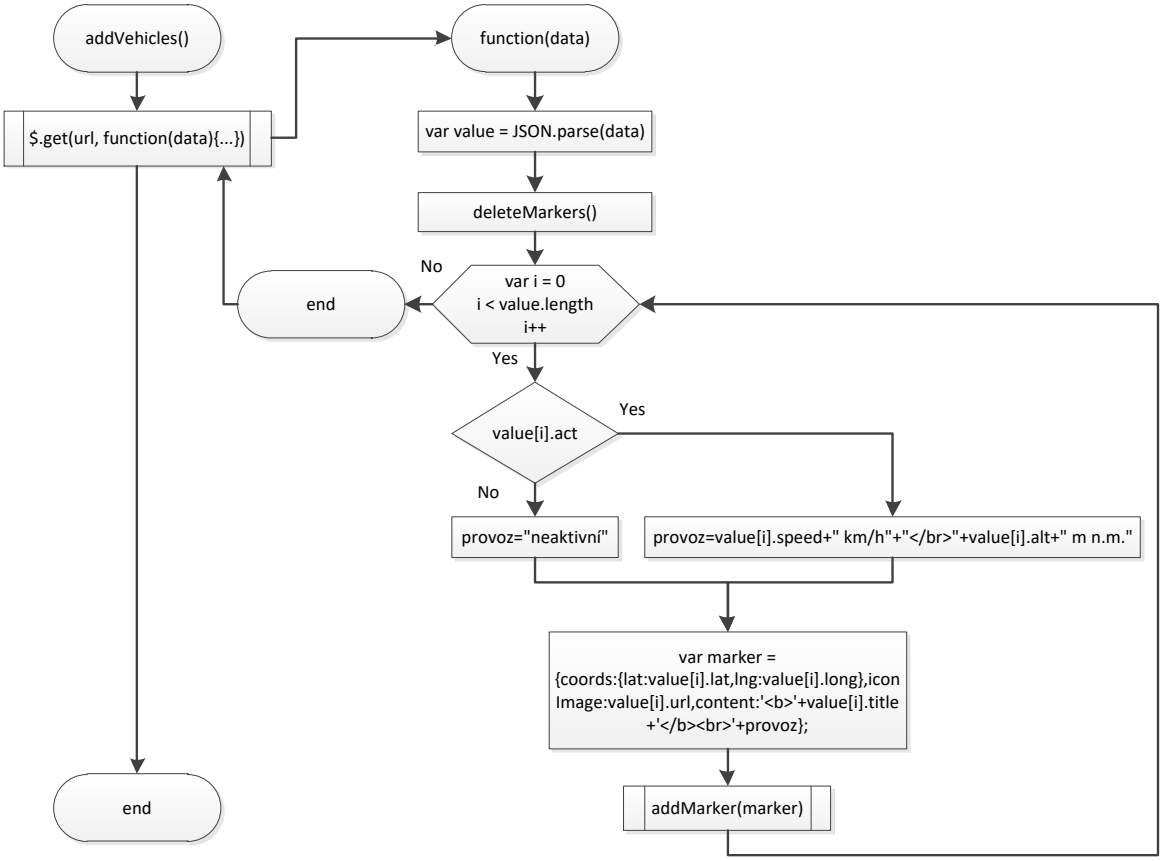


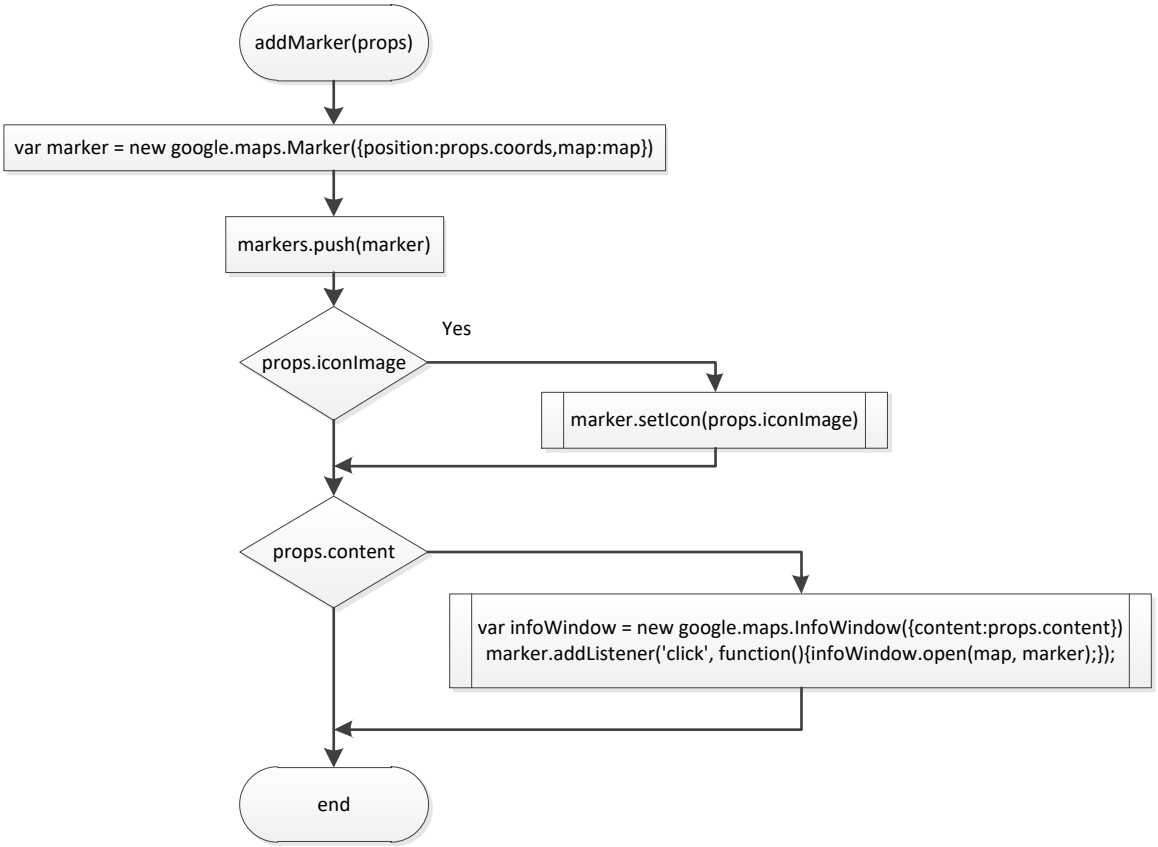
Příloha S Webová aplikace, DataPresenter::actionLocation()

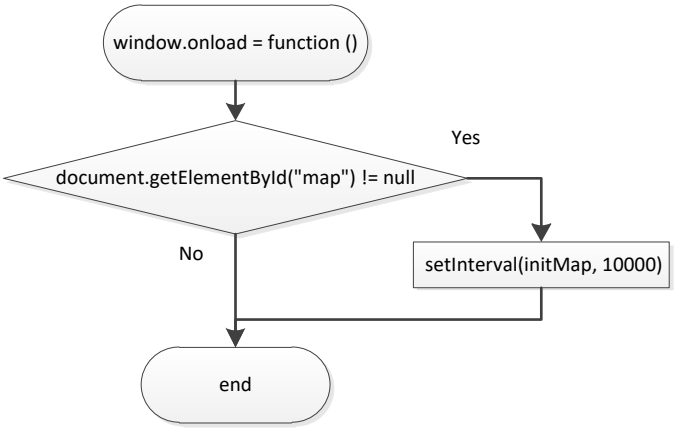


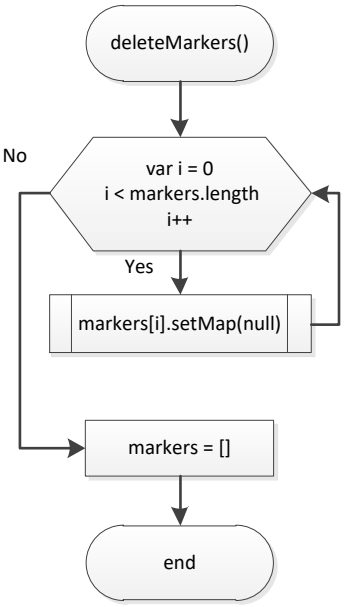
Příloha T Webová aplikace, JavaScript k mapě



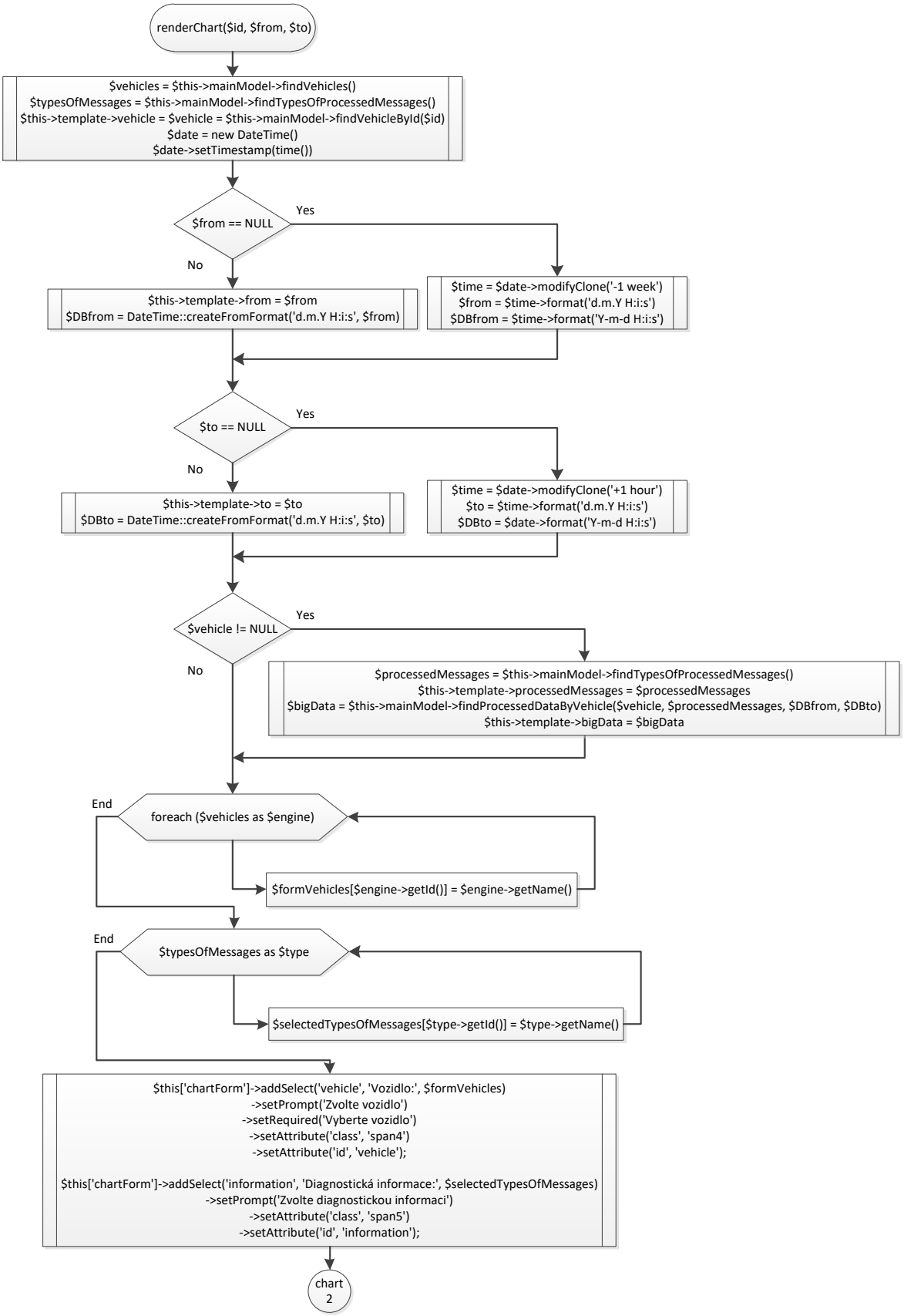

addVehicles()
$.get(url, function(data){...})
end
function(data)
var value = JSON.parse(data)
deleteMarkers()
var i = 0
i < value.length
i++
No
end
Yes
value[i].act
Yes
No
provoz="neaktivní"
provoz=value[i].speed+" km/h"+"</br>"+value[i].alt+" m n.m."
var marker =
{coords:{lat:value[i].lat,lng:value[i].long},icon
Image:value[i].url,content:''+value[i].title
+'
'+provoz};
addMarker(marker)

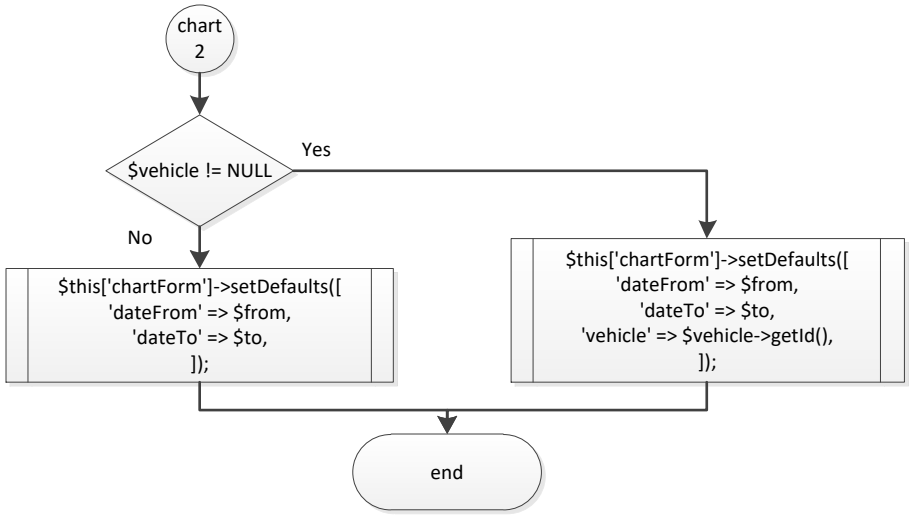

addMarker(props)
var marker = new google.maps.Marker({position:props.coords,map:map})
markers.push(marker)
props.iconImage
Yes
marker.setIcon(props.iconImage)
props.content
var infoWindow = new google.maps.InfoWindow({content:props.content})
marker.addListener('click', function(){infoWindow.open(map, marker);});
end


window.onload = function ()
document.getElementById("map") != null
Yes
No
setInterval(initMap, 10000)
end

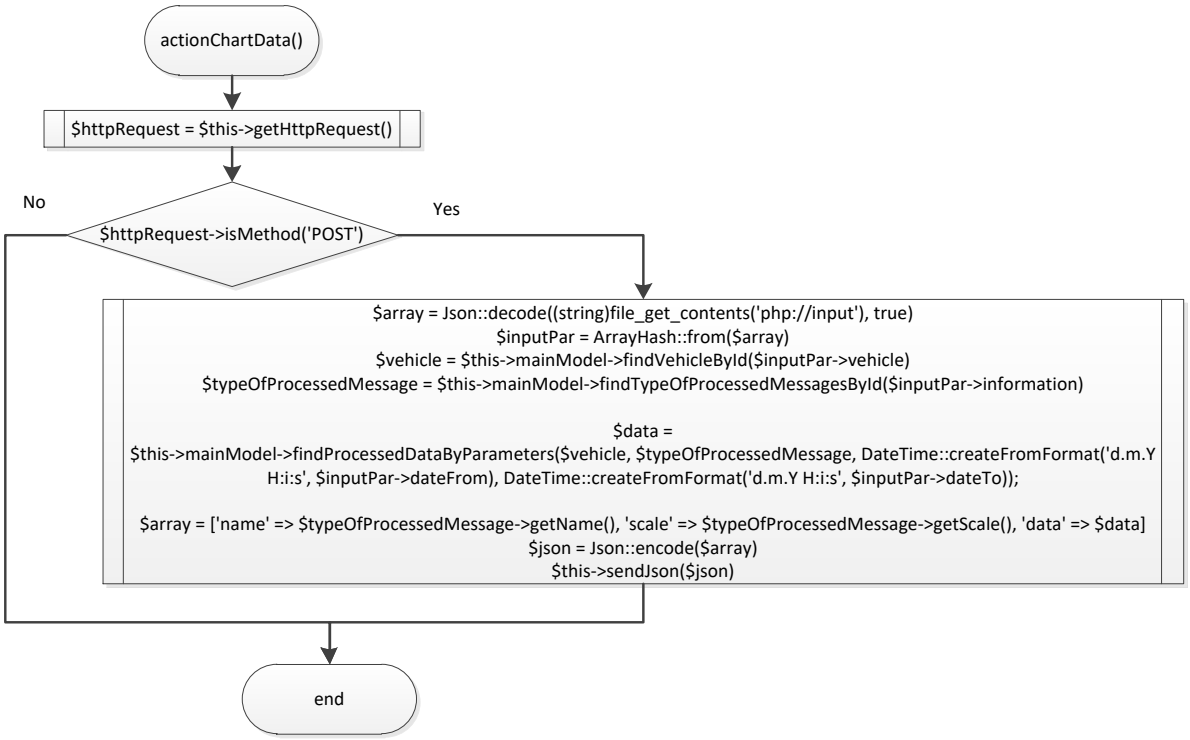

deleteMarkers()
No
var i = 0
i < markers.length
i++
Yes
markers[i].setMap(null)
markers = []
end

Příloha U Webová aplikace, DataPresenter::renderChart()

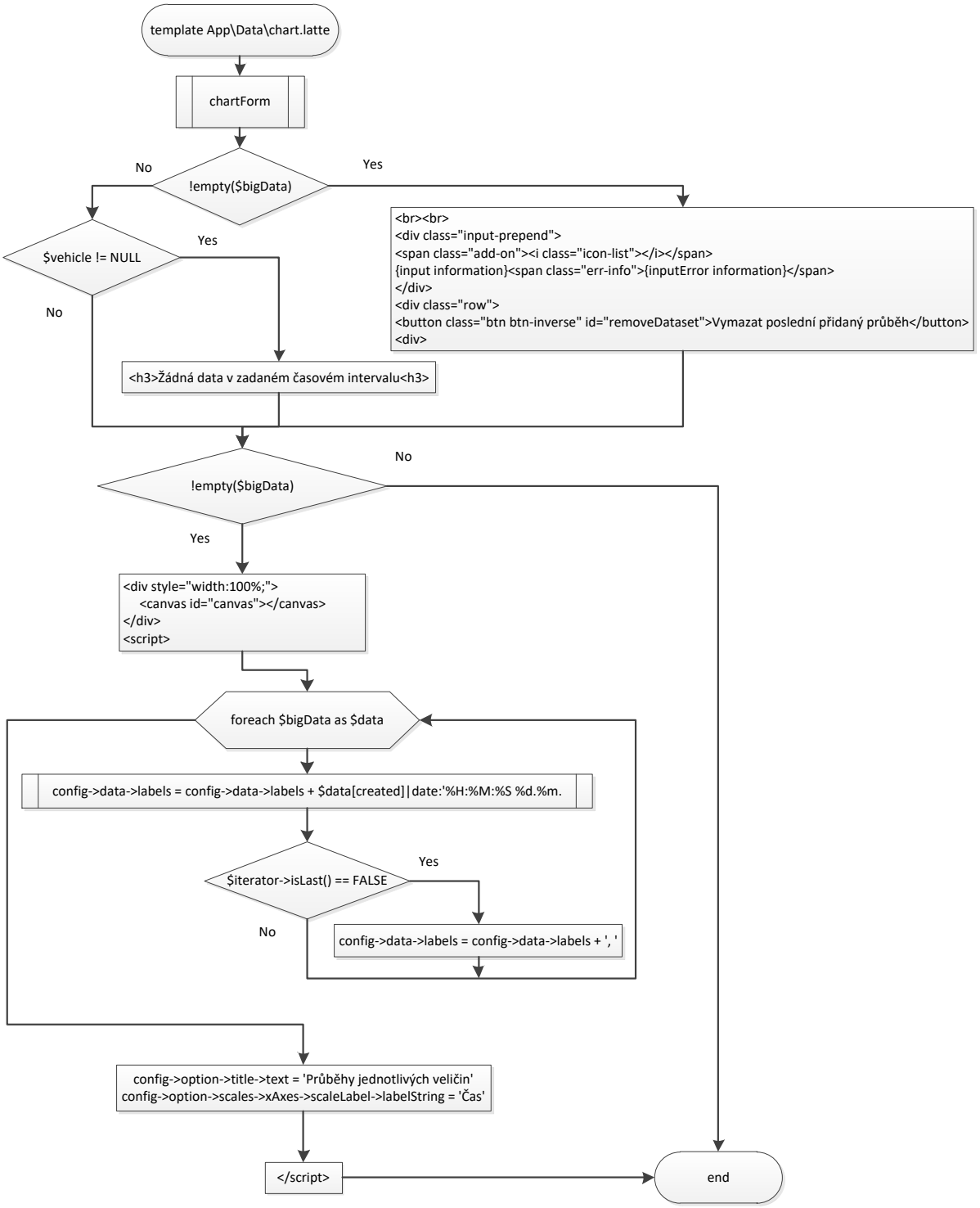




Příloha V Webová aplikace, DataPresenter::actionChartData()



Příloha W Webová aplikace, šablona ke grafu



Příloha X Webová aplikace, JavaScript ke grafu