FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Systém pro sledování vozidel a zaznamenávání knihy jízd pomocí GPS lokátorů
Student:	Matěj Jehlička
Vedoucí:	Ing. Pavel Kubalík, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

1) Prozkoumejte existující řešení.
2) Pomocí metod softwarového inženýrství navrhněte a naprogramujte aplikaci pro sledování vozidel v reálném čase, která bude nasbíraná data ukládat a dále zpracovávat.
3) Řešení se bude skládat ze serverové části a uživatelské části představované lokátorem (tyto dvě zařízení budou mezi sebou komunikovat přes síť Internet)
4) Navržené řešení zrealizujte a řádně otestujte.
5) Požadavky:
 - GPS lokátor bude postaven na platformě ESP32
 - lokátor bude odesílat data na server pomocí GPRS/EDGE
 - server bude umožňovat zobrazení pozice lokátorů v reálném čase
 - server bude umožňovat připojení několika lokátorů
 - server bude zaznamenávat polohu a metadata z lokátorů
 - uživatelské rozhraní bude realizováno formou webové aplikace.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 14. února 2021 v Praze.

ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F8

Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Systém pro sledování vozidel

a zaznamenávání knihy jízd pomocí GPS lokátoru

Matěj Jehlička

květen 2021

Vedoucí práce: Ing. Pavel Kubalík, Ph.D.

Poděkování / Prohlášení

Chtěl bych poděkovat svému vedoucímu práce, Ing. Pavlu Kubalíkovi, Ph.D., za čas a cenné rady, kterých se mi při konzultacích dostalo.

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. 5. 2021

.....

Abstrakt / Abstract

Tato bakalářská práce se zabývá návrhem a implementací systému pro sledování vozidel. Popisuje a vytváří otevřené řešení, které cílí na nízké provozní a pořizovací náklady a je snadno rozšiřitelné. V implementaci využívá výhod moderního jazyka Scala, který kombinuje funkcionální a objektový přístup. Při přenosu aktuálních poloh z lokátoru využívá protokol MQTT, pro ostatní komunikaci pak HTTP. Dále popisuje ostatní použité technologie a způsob nasazení pomocí nástroje Docker. Výsledná implementace je uživatelsky otestována. Nakonec je u ní také změřena náročnost na množství přenesených dat.

Klíčová slova: lokátor, GPS, lokace vozidel, SIM808, ESP32, protokol, Docker

This bachelor thesis deals with designing and creating a vehicle tracking system. It describes a solution that has minimal initial and operating costs. The final solution is open and prepared for an extension. A modern language Scala is used in the implementation, combining object-oriented and functional programming. For transferring position data between the backend server and the tracker, the MQTT protocol is used. Any other communication is transmitted by HTTP protocol. The thesis describes necessary technologies for implementation and deployment, such as Docker. The final implementation is user-tested, and the total amount of transferred data is measured.

Keywords: tracker, GPS, vehicles locations, SIM808, ESP32, protocol, Docker

Title translation: A vehicle tracking system (with a recording of journey logs using GPS)

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Sledování vozidel pomocí GPS a sdílení nasbíraných dat se začalo objevovat ještě dříve, než přišel fenomén internetu věcí. Zprvu šlo o takzvané pasivní sledování. Sledovací zařízení bylo umístěno do vozidla a po celou dobu jeho provozu zaznamenávalo potřebné informace. Po návratu bylo zařízení z vozidla vyjmuto a data stažena, případně dále zpracována. Z dnešního pohledu, kdy není internetové připojení výsadou, naopak, s lehkou nadsázkou lze říci, že je nutnou podmínkou pro přežití v dnešní společnosti, zdá se tento přístup až neuvěřitelný. Veškeré informace chceme mít hned.

Logickým důsledkem tohoto vývoje je i přechod na sledování aktivní. Informace o vozidlech jsou přenášeny v reálném čase. Uživatelé si je mohou zobrazit z kteréhokoliv místa na Zemi a uspokojit tak svou touhu po okamžitých informacích.

Tato práce se zabývá analýzou a implementací právě takového systému. V první části (kapitole 3) popisuje již existující komerční, ale i otevřená řešení, která mezi sebou porovnává. Další část (kapitola 4) se zabývá analýzou a návrhem celého systému. Zde jsou mimo jiné popsány i důležité technologie klíčové pro implementaci. Analýza je podpořena diagramy v notaci UML, které zlepšují její pochopení. Způsob, jak byly poznatky z analýzy použity v implementaci, popisuje kapitola 5. Jelikož je systém komplexní, nepopisuje kapitola všechny implementační detaily, ale zaměřuje se na kritické části systému. I zde je prostý text podpořen diagramy. V závěru práce (kapitole 6) jsou popsány naměřené parametry výsledného řešení, dále výsledky testování a závěrečné zhodnocení v kapitole 7.

Kapitola 2
Cíl práce

Cílem této práce je navrhnout a implementovat systém pro sledování vozidel jako celek. Vytvořit otevřený *backend* poskytující aplikační rozhraní a referenční implementaci *lokátoru* se zaměřením na nízké pořizovací i provozní náklady. Díky tomu bude systém využitelný i pro osobní použití.

Serverový backend bude plně kompatibilní s velice rozšířeným jednodeskovým počítačem RaspberryPi. Dále bude poskytovat aplikační rozhraní, přes které bude přijímat informace od *lokátorů* a následně je poskytovat pro další zpracování. Veškerá komunikace bude prováděna přes protokol HTTP a MQTT.

Uživatelské rozhraní tedy nebude závislé na interní implementaci *backendového serveru*, ale pouze na jeho endpointech a struktuře dotazů, resp. odpovědí. Implementované řešení *uživatelského rozhraní* lze tedy chápat také jako referenční a může být rozšířeno, resp. ochuzeno, o funkčnosti dle potřeby.

K implementaci lokátorů bude využita platforma ESP32 společně s modulem SIM808, který umožňuje přístup k síti internet (pomocí technologie GPRS) a k aktuální GPS poloze. Výstupem práce bude řádně otestovaný funkční prototyp *lokátoru*, *uživatelské rozhraní* a *serverový backend*.

Kapitola 3
Existující řešení

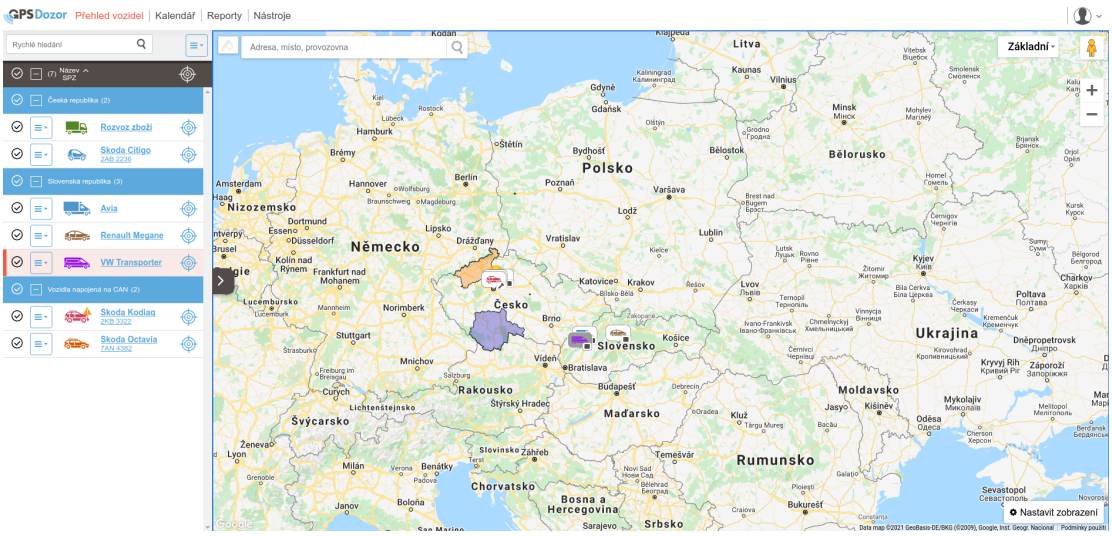
Do následujícího porovnání nebyly kvůli vysokému počtu zahrnuty všechny služby, produkty a systémy pro sledování vozidel. Vybrané produkty odpovídají svou specifikací a nabízenými funkcionalitami cíli tohoto projektu. Všechny porovnávané produkty (až na jeden) jsou komerční. U každého produktu je vybrána taková služba, která svou specifikací odpovídá rozsahu funkcionalit tohoto projektu. Dále je u všech řešení uvedena pořizovací (pokud existuje) cena a měsíční náklady na provoz, společně s klíčovými vlastnostmi každého produktu.

GPSDozor

GPSDozor [53] je český produkt, který nabízí kompletně hotové řešení, včetně *sledovacích zařízení*. Ta mohou být napájena integrovanou baterií, nebo přímo z vozidla – autobaterií. Klientská přístupnost je velice přívětivá. Produkt se snaží zachovat jednoduchost ovládání bez zbytečných funkcí, které uživatelé nepotřebují.

Umožňuje přístup k datům přes webové rozhraní i mobilní aplikaci. Mimo sledování aktuální polohy umožňuje zaznamenávat knihu jízd, řazení vozů do kategorií nebo odesílat upozornění, pokud některé z vozidel opustí předem definovanou oblast, tzv. geofence. Další funkce, jako například sledování čerpání pohonných hmot, jsou dostupné také. Výrobce dále uvádí zpoždění zobrazení aktuální polohy 5–10 minut.

Takto kompletní služba je u firmy *TLV s.r.o. – GPS Dozor* zpoplatněna částkou 2 390 Kč za sledovací zařízení s měsíčním paušálním poplatkem 200 Kč.[1]



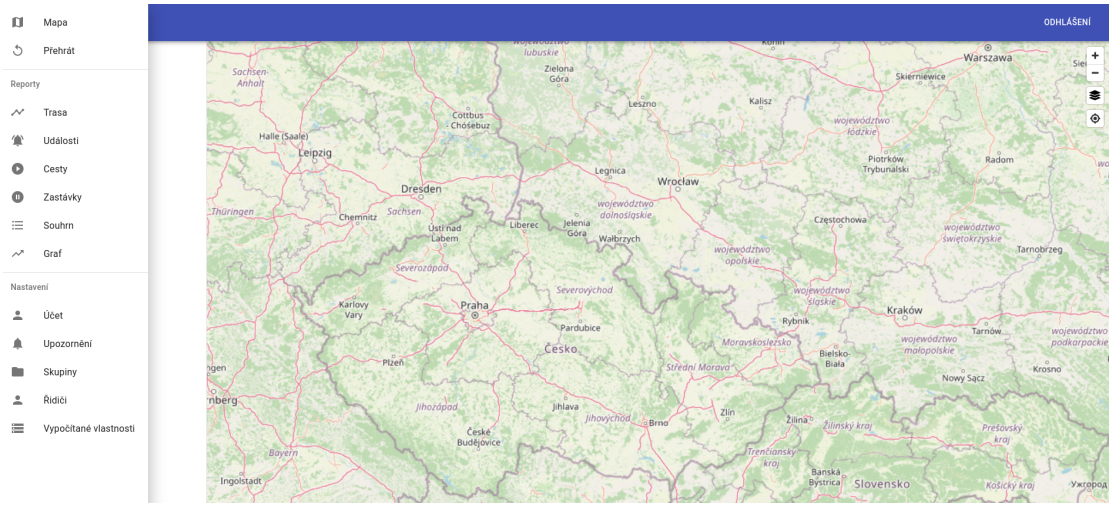
Obrázek 3.1. Uživatelské rozhraní aplikace GPS Dozor

[1] Uvedená cena je za produkt, který svou specifikací odpovídá zařízení a funkcionalitám vytvořeným v tomto projektu. Web produktu: `https://www.gpsdozor.cz/lokator-basic`

Traccar

Traccar [55] je projekt s otevřenými zdrojovými kódy a širokou podporou hardwaru. Funguje na třech základních vrstvách: sledovací zařízení (v tomto případě i chytrý telefon), řídící server, klientská aplikace (webová, mobilní Android/iOS). Služba nevyvíjí specifické sledovací zařízení, ale podporuje již hotová řešení. Tím zaručuje dostatečnou variabilitu a umožňuje vybrat zařízení dle finančních možností, či jiných preferencí, zákazníka. Případně umožňuje integrovat vlastní řešení. Nabízené funkcionality jsou mimo sledování polohy v reálném čase také například zaznamenávání knihy jízd a geofence s možností nastavení upozornění.

Jelikož jde o otevřený projekt, je možné celé řešení hostovat na vlastním hardwaru (serveru) a nepředávat tak citlivé informace třetím stranám. Zároveň je možné zakoupit si cloudový server pro 5 zařízení za 9,95 $[2]. Měsíční paušál je tedy srovnatelný s měsíčním paušálem pro jeden lokátor u produktu GPSDozor. Pořizovací cena poté závisí na vybraném typu lokátoru.



Obrázek 3.2. Uživatelské rozhraní aplikace Traccar

GPSWOX

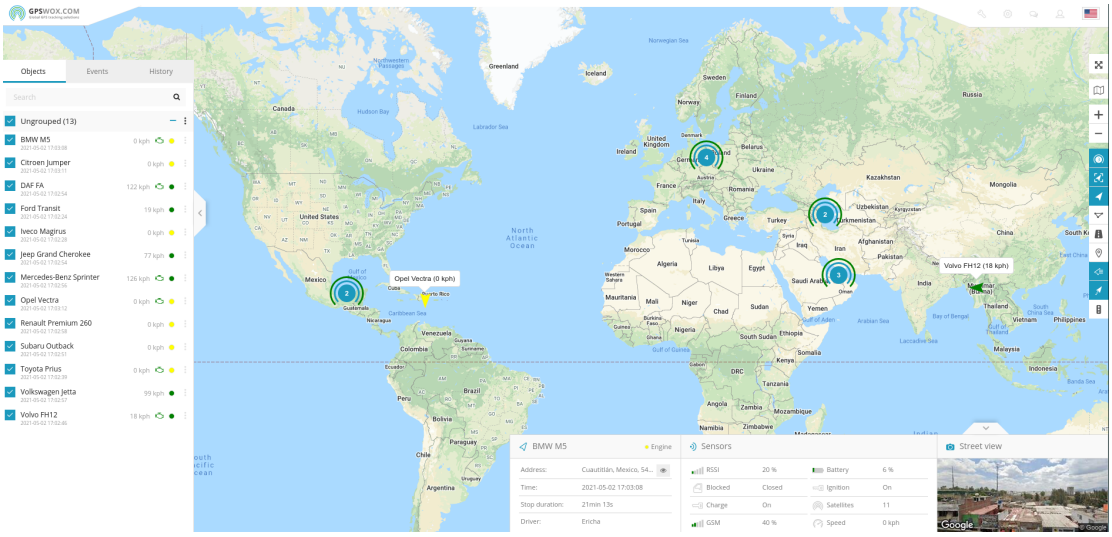
Stejně jako předchozí i tato služba umožňuje uživatelům zobrazovat data ve webovém prohlížeči nebo v mobilní aplikaci. Oproti prvnímu popisovanému řešení (GSPDozor), GPSWOX [25] nenabízí *lokátory*, ale podporuje sadu již hotových řešení od několika výrobců. Některá se připojují přímo do diagnostické sběrnice CAN, jiná fungují na bateriový provoz s interním akumulátorem a jsou nezávislá na napájení z vozidla.

Další podpůrné funkce, jako zaznamenávání knihy jízd a dělení vozidel do flotil, jsou dostupné také. Nechybí ani funkcionalita geofence s možností nastavení notifikace při jejich překročení.

Zajímavou doplňující funkcionalitou, kterou předchozí zmiňovaná řešení nemají, je možnost sledování mobilních telefonů. Do sledovaného mobilního telefonu je nainstalována aplikace, která komunikuje se serverem a umožňuje tak jeho lokalizaci. Tuto aplikaci lze zdarma stáhnout na stránkách produktu.

GPSWOX využívá tedy stejný přístup jako projekt Traccar. Uživatel si pouze předplatí pronájem serveru, na který lokátor odesílá data. Pořizovací cena tak záleží na

[2] 217.51 Kč k 12. 4. 2021



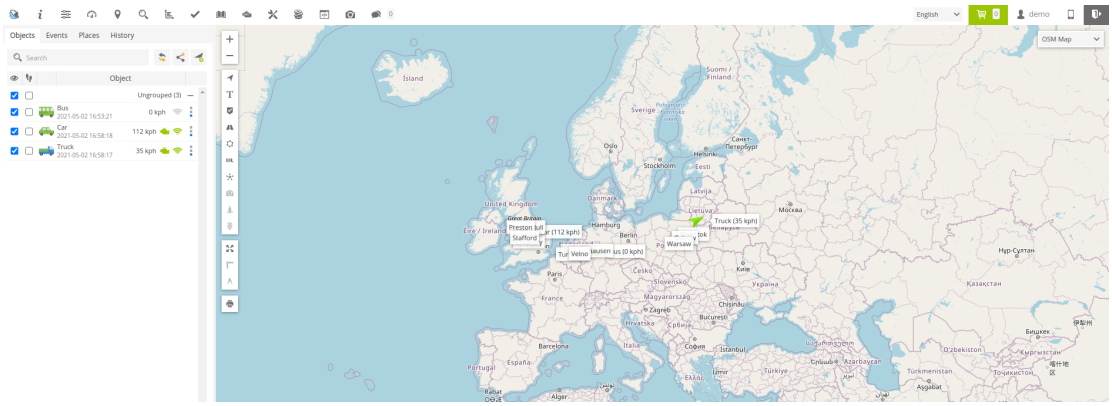
Obrázek 3.3. Uživatelské rozhraní aplikace GPSWOX

typu potřebného lokátoru. Měsíční provoz poté uživatele stojí 9,97 €[3] měsíčně s možností připojení až 5 lokátorů. Pro uživatele, kteří chtějí sledovat pouze jedno zařízení, GPSWOX nabízí možnost předplatit sledování 1 lokátoru za 2,99 $[4] měsíčně.

GPS-server.net

Řešení pro sledování vozidel od společnosti GPS-server.net [24] nabízí stejné možnosti jako GPSWOX a Traccar. Uživatel si kupuje pouze zdroje sledovacího serveru. V tomto případě s roční periodou. Rozdílem proti ostatním je jen možnost zakoupit si zdroje pro sledování pouze jediného lokátoru. Tento pronájem stojí 15 €[5] ročně. Sledování 5 lokátorů je poté zpoplatněno 60 €[6] ročně.

Jde tedy o nejlevnější z porovnávaných řešení, co se týče měsíčních poplatků. Měsíční poplatky pro sledování pěti lokátorů činí po rozpočítání roční platby 5 €[7]. Za tuto cenu nabízí srovnatelné funkcionality jako konkurenční řešení. Uživatelské rozhraní je dostupné jak ve formě webové, tak mobilní aplikace. Umožňuje vytvářet geofence, odesílat upozornění a generovat reporty o zaznamenaném pohybu.



Obrázek 3.4. Uživatelské rozhraní aplikace GPS-server.net

[3] 259,47 Kč k 12. 4. 2021
[4] 65,36 Kč k 12. 4. 2021
[5] 390,38 Kč k 12. 4. 2021
[6] 1561,5 Kč k 12. 4. 2021
[7] 130,13 Kč k 12. 4. 2021

Kapitola 4
Analýza a návrh

Tato kapitola popisuje analýzu technologií potřebných pro sledování vozidel pomocí GPS, dále návrh architektury – rozdělení řešení na více logických celků. Pro každý popisuje volbu implementačního jazyka, frameworku, resp. sady knihoven. Popisuje také způsob ukládání dat. Jsou zde zahrnuty také popisy klíčových protokolů a technologií.

Pro lepší ilustraci a pochopení některých popisovaných problémů jsou uváděny diagramy. Všechny jsou zakresleny v notaci UML, není-li uvedeno jinak. Diagramy v této kapitole nemusejí přesně reflektovat konečnou implementaci. Jde pouze o návrh (analýzu) problémů, které popisují.

4.1 Cílová skupina

Cílovou skupinou jsou především jednotlivci. Ti, kteří si chtějí spravovat data o své poloze sami a nechtějí vkládat důvěru do technologií třetích stran. Dále také ti, kteří chtějí mít ucelené řešení s nízkými pořizovacími náklady. Těmi mohou být kromě jednotlivců i malé firmy s vozovým parkem čítajícím jednotky až desítky vozů.

Velcí přepravci a provozovatelé vozidel zajisté ocení, že se o administraci systému nebudou muset starat sami, a využijí některou z komerčních technologií. Ty většinou poskytují i větší integraci, jako např. plánování trasy. Nejsou tedy primární cílovou skupinou tohoto projektu.

4.2 Funkční a nefunkční požadavky

Funkčními požadavky jsou zde myšleny takové vlastnosti softwaru, které popisují jeho požadované chování. Popisují jeho funkcionality. Oproti nim stojí požadavky *nefunkční*, tedy takové, které popisují omezení designu aplikace.

4.2.1 Funkční požadavky

- **F1 – Systém bude umožňovat správu uživatelů a jejich rolí.**
  Každá role bude mít přesně definované operace, které může provádět.
- **F2 – Systém bude obsahovat následující role: User, Reader, Editor a Admin.**
  Každý uživatel může mít přiřazenu žádnou, ale i několik rolí. Role budou organizovány hierarchicky, a to následovně:
  - User – přihlášený uživatel může zobrazit pouze aktuální pozice vozidel a změnit své přístupové heslo
  - Reader – může zobrazit seznam vozidel, lokátorů a uživatelů + vše, co je umožněno roli User
  - Editor – může editovat vozidla a lokátory + vše, co je umožněno roli Reader
  - Admin – může editovat uživatele + vše, co je umožněno roli Editor

- **F3 – Systém bude poskytovat aplikační rozhraní pro práci se záznamy.**
  Aplikační rozhraní bude umožňovat přidávat, upravovat a odstraňovat záznamy všech entit v rámci mezí stanovených rolemi.
- **F4 – Systém bude sledovat vykonané trasy jednotlivých vozidel.**
  Trasy bude evidovat a umožňovat jejich zobrazení.
- **F5 – Sledovací zařízení bude konfigurovatelné přes webové rozhraní.**
- **F6 – Vozidla půjdou rozřazovat do flotil.**
  Vozidlo může, ale nemusí být součástí flotily. Může ale také být součástí více flotil. Umožní tedy rozdělit vozidla do logický celků.
- **F7 – Systém bude umožňovat exportovat trasy vozidel ve formátu GPX.**
  Vygenerovaný soubor bude možné stáhnout do zařízení uživatele přes uživatelské rozhraní.
- **F8 – Systém bude poskytovat webové uživatelské rozhraní.**
  V uživatelském rozhraní bude umožněna práce se záznamy, společně se zobrazením aktuální polohy vozidel.

4.2.2 Nefunkční požadavky

- **N1 – Server bude možné provozovat na jednodeskovém počítači RaspberryPi.**
  Součástí budou Dockerfile resp. Docker compose file pro přípravu a nastartování požadovaného Docker Image.
- **N2 – Aplikační rozhraní bude interagovat přes protokol HTTP.**
- **N3 – Lokátor bude postaven na platformě ESP32.**
- **N4 – Lokátor bude přistupovat do sítě internet přes služby mobilního operátora.**
- **N5 – Lokátor bude se serverem komunikovat prostřednictví protokolu MQTT.**
- **N6 – Veškerá veřejná komunikace bude šifrována protokolem TLS.**
  Veškerá komunikace, která bude probíhat přes síť internet, bude šifrována, aby bylo zajištěno soukromí přenášených dat.
- **N7 – Lokátor bude měřit aktuální GPS pozici a odesílat ji na backendový server.**

4.3 Případy užití

Případy užití popisují interakci uživatele se systémem, ale i interakci jednotlivých částí systému mezi sebou. Každý případ užití je popsán scénářem, který přesně definuje kroky, které se mají vykonat společně s jejich pořadím. Ve scénáři figurují tzv. Aktéři (části systému, nebo uživatelé, kteří v případu užití vykonávají nějakou činnost). Níže popsané případy užití zachycuje diagram na obrázku 4.1.

- **UC1 – Přihlášení do systému**
  Aktéři: User, Systém
  Scénář:
  1. User vyplní přihlašovací formulář a odešle ho.
  2. Systém uživatele přihlásí, nebo jeho požadavek odmítne s udáním důvodu.

- **UC2 – Zobrazení aktuální polohy vozidel**
  Aktéři: User, Systém
  Scénář:
  1. Systém zobrazí seznam vozidel.
  2. User vybere vozidlo, u něhož chce zobrazovat aktuální polohu, případně zruší výběr vozidla, u kterého polohu sledovat nadále nechce.
  3. Systém začne zobrazovat aktuální polohu vybraného vozidla, případně přestane zobrazovat aktuální polohu vybraného vozidla.
  4. Zpět na bod 2, nebo konec.

- **UC3 – Změno vlastního hesla**
  Aktéři: User, Systém
  Scénář:
  1. Systém zobrazí formulář pro změnu hesla.
  2. User vyplní formulář pro změnu hesla.
  3. Systém uloží nové heslo, nebo požadavek odmítne s udáním důvodu.

- **UC4 – Zobrazení seznamu uživatelů**
  Aktéři: Reader, Systém
  Scénář:
  1. Reader MŮŽE vyplnit vyhledávací pole.
  2. Systém zobrazí seznam uživatelů podle případného uživatelského vstupu, a to podle výsledků fulltextového vyhledávání ve jménech a usernamech uživatelů, jinak zobrazí všechny uživatele.

- **UC5 – Zobrazení seznamu vozidel**
  Aktéři: Reader, Systém
  Scénář:
  1. Reader MŮŽE vyplnit vyhledávací pole.
  2. Systém zobrazí seznam vozidel podle případného uživatelského vstupu, a to podle výsledků fulltextového vyhledávání v názvech vozidel, jinak zobrazí všechna vozidla.

- **UC6 – Zobrazení seznamu lokátorů**
  Aktéři: Reader, Systém
  Scénář:
  1. Reader MŮŽE vyplnit vyhledávací pole.
  2. Systém zobrazí seznam lokátorů podle případného uživatelského vstupu, a to podle výsledků fulltextového vyhledávání v názvech vozidel, jinak zobrazí všechna vozidla.

- **UC7 – Zobrazení seznamu tras vozidel**
  Aktéři: Reader, Systém
  Scénář:
  1. Reader vybere vozidlo, u kterého chce zobrazit trasy.
  2. Systém zobrazí seznam vybraného vozidla.

- **UC8 – Vygenerování konfiguračního souboru pro lokátor**
  Aktéři: Reader, Systém
  Scénář:
  1. Reader vybere lokátor, pro který chce vygenerovat konfigurační soubor.
  2. Reader klikne na tlačítko vygenerovat konfigurační soubor.
  3. Systém vygeneruje konfigurační soubor a umožní jeho stažení.

- **UC9 – Exportování trasy vozidla**
  Aktéři: Reader, Systém
  Scénář:
  1. Reader vybere vozidlo, pro které chce vygenerovat záznam trasy.
  2. Reader klikne na tlačítko pro vygenerování souboru se záznamem trasy.
  3. Systém vygeneruje soubor se záznamem trasy a umožní jeho stažení.

- **UC10 – Úprava vozidel**
  Aktéři: Editor, Systém
  Scénář:
  1. Editor vybere vozidlo, které chce upravit.
  2. Systém zobrazí formulář s předvyplněnými informacemi o vybraném vozidle.
  3. Editor upraví požadovaná data a potvrdí kliknutím na tlačítko uložit.
  4. Systém uloží změny, nebo odmítne požadavek s udáním důvodu.

- **UC11 – Úprava lokátorů**
  Aktéři: Editor, Systém
  Scénář:
  1. Editor vybere lokátor, který chce upravit.
  2. Systém zobrazí formulář s předvyplněnými informacemi o vybraném lokátoru.
  3. Editor upraví požadovaná data a potvrdí kliknutím na tlačítko uložit.
  4. Systém uloží změny, nebo odmítne požadavek s udáním důvodu.

- **UC12 – Úprava uživatelů**
  Aktéři: Administrator, Systém
  Scénář:
  1. Administrator vybere uživatele, kterého chce upravit.
  2. Systém zobrazí formulář s předvyplněnými informacemi o vybraném uživateli.
  3. Administrator upraví požadovaná data a potvrdí kliknutím na tlačítko uložit.
  4. Systém uloží změny, nebo odmítne požadavek s udáním důvodu.

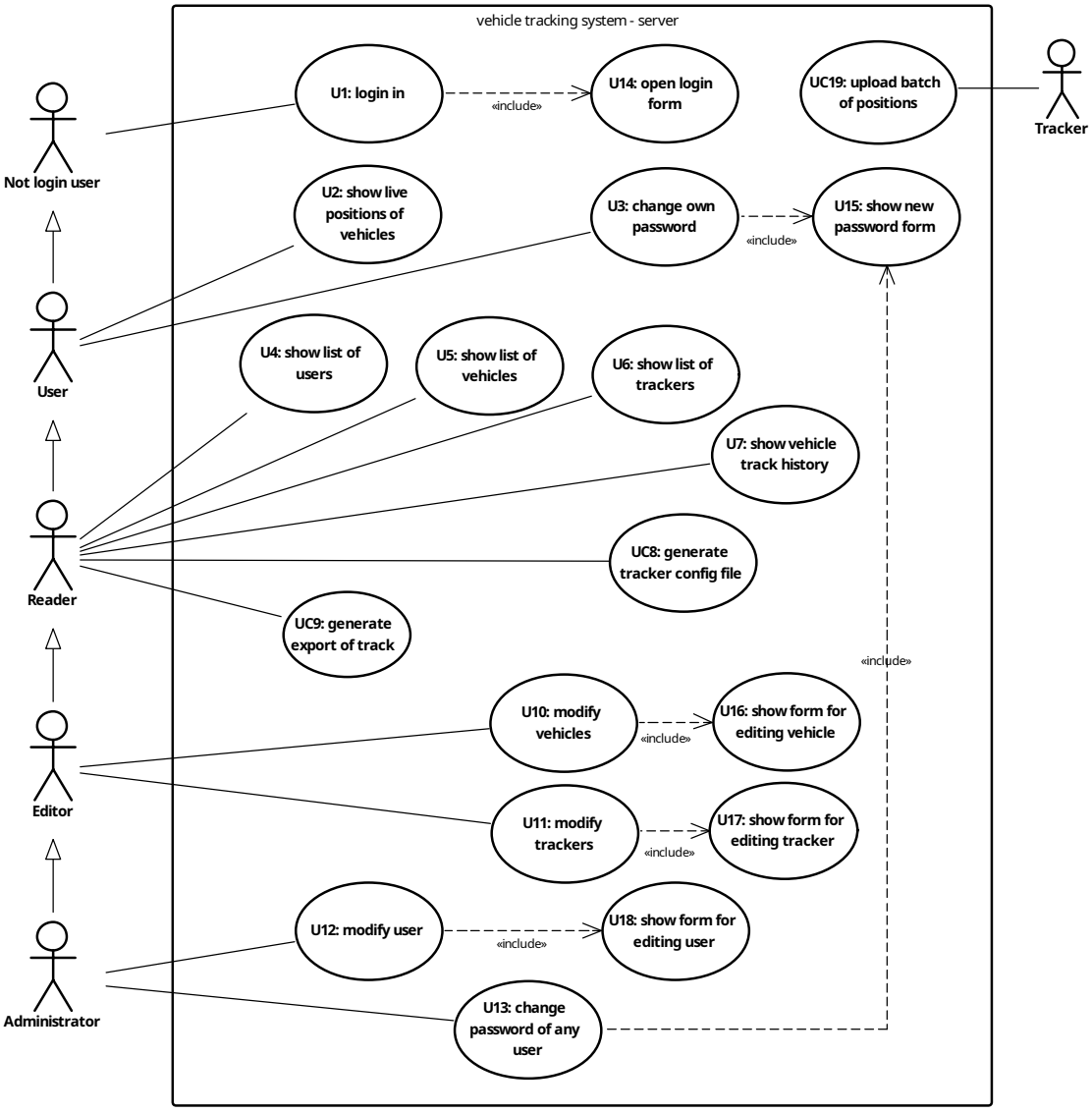
- **UC13 – Změna hesla uživatele**
  Aktéři: Administrator, Systém
  Scénář:
  1. Administrator vybere uživatele, kterému chce změnit heslo, a potvrdí kliknutím na tlačítko změnit heslo.
  2. Systém zobrazí formulář pro změnu hesla.
  3. Administrator vyplní formulář pro změnu hesla.
  4. Systém uloží změny, nebo odmítne požadavek s udáním důvodu.

- **UC14 – Zobrazení přihlašovacího formuláře**
  Aktéři: Systém
  Scénář:
  1. Systém zobrazí přihlašovací formulář.

- **UC15 – Zobrazení formuláře pro změnu hesla**
  Aktéři: Systém
  Scénář:
  1. Systém zobrazí formulář pro změnu hesla.

- **UC16 – Zobrazení editačního formuláře vozidla**
  Aktéři: Systém
  Scénář:
  1. Systém zobrazí editační formulář vozidla.

- **UC17 – Zobrazení editačního formuláře lokátoru**
  Aktéři: Systém
  Scénář:
  1. Systém zobrazí editační formulář lokátoru.
- **UC18 – Zobrazení editačního formuláře uživatele**
  Aktéři: Systém
  Scénář:
  1. Systém zobrazí editační formulář uživatele.
- **UC19 – Odeslání získaných pozic na server**
  Aktéři: Tracker, Systém
  Scénář:
  1. Tracker nasbírá data o aktuální pozici.
  2. Tracker odešle naměřená data na server.
  3. Server uloží přijaté pozice.



Obrázek 4.1. Diagram případů užití

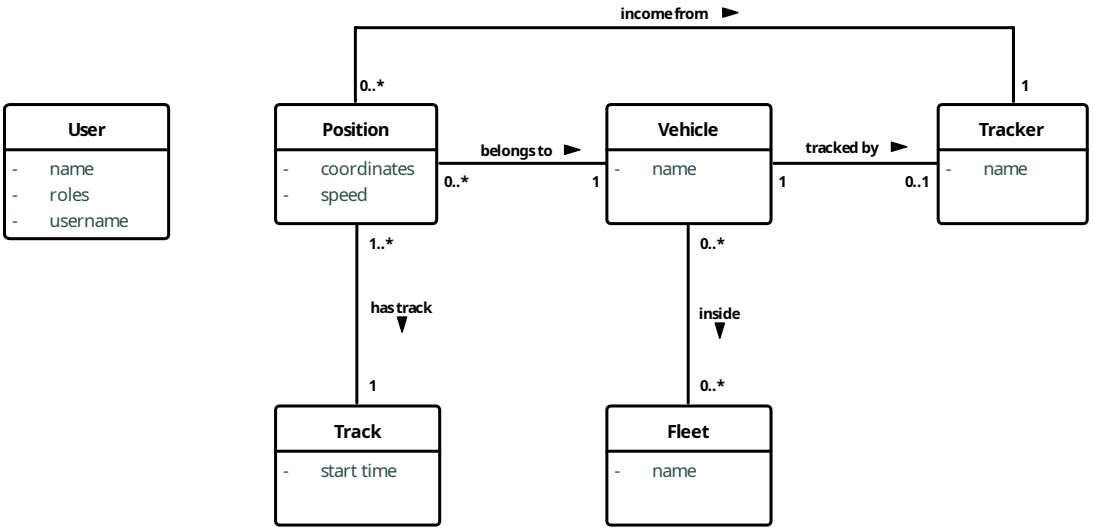
4.4 Doménový model

Doména systému pro sledování vozidel, definována mimo jiné funkčními a nefunkčními požadavky, spojuje několik entit do jednoho logického celku. Diagram doménového modelu je zobrazen na obrázku 4.2. Vyobrazuje všechny entity a vztahy mezi nimi. Separovaná entita `User` bude využita pouze pro autentizaci komunikace a s doménou jako takovou do jisté míry zcela nesouvisí. Proto také nemá vztah s jinými entitami.

Každá pozice – `Position` – musí být nutně vygenerována některým z lokátorů – `Tracker` – (ten ji vytvoří podle naměřených hodnot) a zároveň musí patřit do právě jedné trasy – `Track`. Tím jsou naměřené pozice rozděleny do menších logických celků, kde každá trasa reprezentuje jednu cestu vozidla. Přispívá tak k přehlednosti naměřených pozic.

Vozidlo může mít v jednu chvíli přiřazen maximálně jeden lokátor. Aby bylo možné přiřadit historii pozic ke správnému vozidlu – `Vehicle` – i po odstranění nebo změně lokátoru u vozidla, každá pozice patří právě jednomu vozidlu, jehož lokátor ji vygeneroval.

Konečně flotila – `Fleet` – slouží pro dělení vozidel do logických celků. Jedna flotila může mít více vozidel a současně jedno vozidlo může být součástí více flotil.



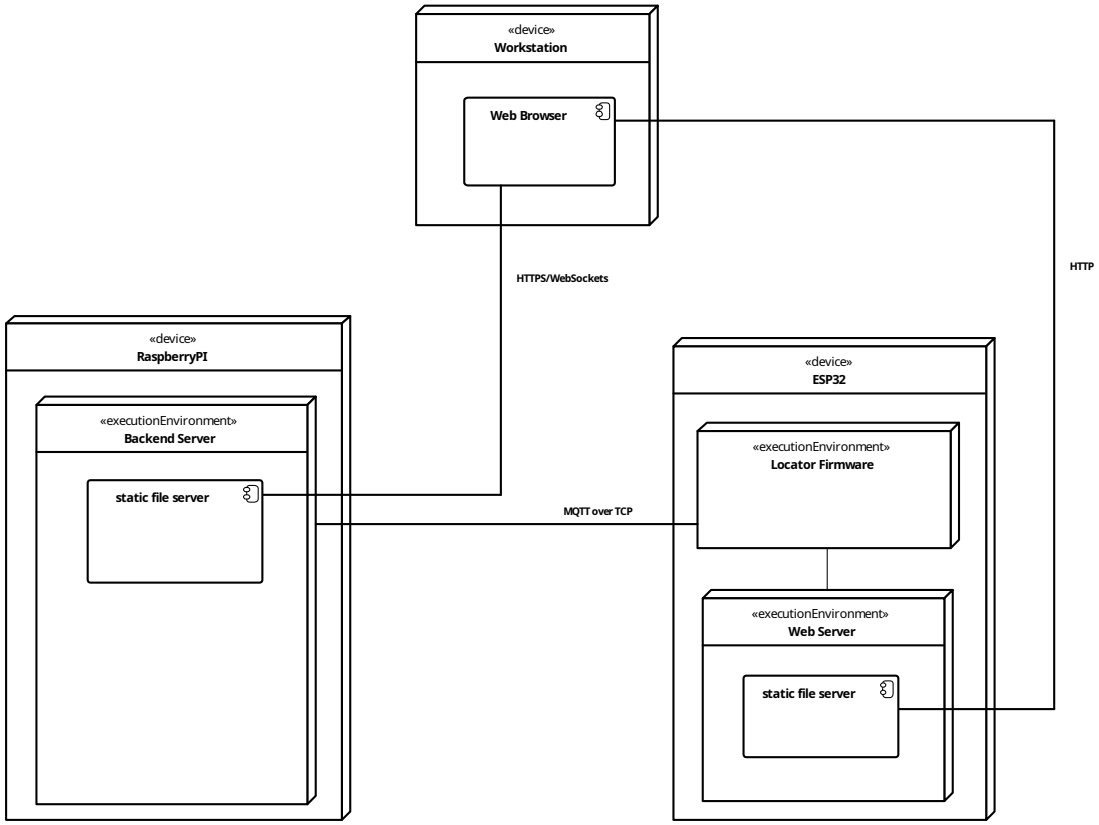
Obrázek 4.2. Doménový model

4.5 Architektura projektu

Systém se bude skládat ze tří hlavních celků – *lokátoru*, *backendového serveru* a *uživatelského rozhraní*. Jelikož jde o tři odlišné kategorie softwaru vyžadující jiné funkcionality, avšak dohromady tvořící plnohodnotný funkční celek, bude nutné vybrat vhodné implementační jazyky a technologie. Schématické znázornění částí systému a jejich vzájemnou interakci popisuje diagram 4.3. Každá část systému bude pak zvláště popsána v samostatné podsekci.

4.5.1 Backendový server

Server bude muset sbírat data z několika *lokátorů* současně a společně s tím vyřizovat dotazy od *uživatelského rozhraní* (případně jinak obsluhovat API), které může být



Obrázek 4.3. Návrh architektury

spuštěno na více zařízeních zároveň. Z toho důvodu bude vhodné použít takový programovací jazyk, který umožňuje asynchronní zpracování a zároveň v něm lze snadno a bezpečně řešit konkurenční přístup k datům. Pro implementaci tedy bude zvolen moderní jazyk Scala, který je díky kombinaci funkcionálního a objektového přístupu pro programování backendových aplikací vhodný. Jeho velkou výhodou je též kompilace do JVM bajtkódu, přeložený program je tedy multiplatformní a lze jej provozovat na různých platformách a operačních systémech, včetně ARM64 použitého v RaspberryPi.

4.5.2 Uživatelské rozhraní

Interakci s uživatelem bude zajišťovat webová aplikace (F8). Díky tomu bude možné k *uživatelskému rozhraní* přistupovat z kteréhokoliv zařízení, které má webový prohlížeč. K implementaci bude zvolen JavaScriptový framework Vue.js. Touto volbou se zajistí kompatibilita mezi zařízeními a jedinou doplňující podmínkou bude podpora JavaScriptu ve webovém prohlížeči. Přístup k datům tedy bude možný jak z desktopu, tak např. mobilního telefonu.

Výhodou využití JavaScriptu je také možnost poskytovat *uživatelské rozhraní* přímo z *backendového serveru*. Sestavený web budou pouze statické soubory. Celá aplikace se stáhne na uživatelské zařízení a veškerá exekuce už poté bude probíhat na něm. Navigaci bude také řeší samotná webová aplikace. Nebude potřeba použít dedikovaný webový server, na kterém by bylo *uživatelské rozhraní* spuštěno. Tím se sníží latence při dotazech na *backendový server* (oproti případu, kdy by byl server na jiném stroji) a zjednoduší nasazení systému.

Uživatelské rozhraní bude získávat data pro zobrazení z *backendového serveru*. Získaná data o pozicích lokátorů budou zobrazována na mapovém podkladu formou usku-

tečněných tras jednotlivých lokátorů a formou jejich aktuální (poslední známé) pozice. Jednotlivé trasy půjdou přes *uživatelské rozhraní* exportovat ve formátu GPX (F7).

4.5.3 Lokátor

Hlavním úkolem *lokátoru* je zaznamenávat aktuální GPS souřadnice a zasílat je na *backendový server* (N7). Dále musí mít kompaktní rozměry, aby nezabíral ve vozidle příliš mnoho místa. Zároveň nebude provádět, až na šifrování komunikace, velké výpočetní úkony, proto bude vhodné pro implementaci zvolit mikrokontrolér, resp. jednodeskový počítač. Nabízí se použití platformy Arduino, existující v mnoha verzích, která je velice populární v komunitě zabývající se IoT.

Rozumnou verzí (co do celkové velikosti) je pro lokátor typ Arduino Nano. Jde o mikrokontrolér s mikroprocesorem ATmega328 a velikostí flash paměti 32 KB [3]. Dále má vyvedeno 22 vstupně výstupních digitálních a 8 vstupně výstupních analogových pinů, které lze programově ovládat, zapisovat/číst z nich data. Nemá žádné další periferie přímo na desce. Funkční požadavek F5 definuje, že sledovací zařízení bude konfigurovatelné přes webové rozhraní. Mikrokotrolér Arduino NANO se tedy nehodí pro implementaci, neboť by bylo nutné připojit externí WiFi modul. Dalším důvodem, proč tento mikrokontrolér není vhodný, je nutnost přenášená data šifrovat. Šifrování komunikace je výpočetně náročný úkon, na který mikroprocesor ATmega328 výkonnostně nestačí. Existují i jiné varianty podobné Arduino Nano s výkonnějšími, 32bitovými mikroprocesory a možností WiFi konektivity [4]. Ani ty ale nejsou vhodné, protože je předvídatelné, že projekt bude využívat několik knihoven a zkompilovaný kód bude větší než 32 KB.

Pro implementaci bude tedy nutné vybrat vhodnější, výkonnější platformu. Konkrétně vývojovou desku DOIT ESP 32 DevKit. Tato vývojová deska disponuje dvoujádrovým procesorem Tensilica Xtensa 32-bit LX6 a pamětí flash o velikosti 4 MB, jak uvádí dokumentace [61]. Obě vlastnosti budou s výhodou využity v následné implementaci. Dále obsahuje přímo na desce WiFi modul, takže pro provoz webového serveru nebude nutné připojení modulu externího. Díky dostatečné kapacitě flash paměti se na ni kromě zkompilovaného kódu vejde také webové rozhraní. Může tedy být uloženo přímo na vývojové desce bez nutnosti připojovat externí paměťové médium. Dvě dostupná procesorová jádra se v implmentaci využijí také, jedno se bude starat o obsluhu AP, webového serveru a poskytování webové aplikace uživatelského rozhraní, zatímco druhé o sbírání a přípravu dat pro odeslání na *backendový server*.

Sledování polohy a vlastní odesílání dat na *backendový server* bude zajišťovat modul WARCAR s integrovaným modulem SIM808 od společnosti SIMCom. Data o aktuálních pozicích budou přenášena protokolem MQTT (pro více informací o protokolu vizte sekce 4.6.1 a 4.6.3) přes síť internet. Připojení k internetu bude zajištěno službou GPRS, již poskytují mobilní operátoři.

Konfigurace bude prováděna přes webové rozhraní. Webový server bude zprovozněn přímo na mikrokontroléru ESP32. Síť, ve které bude webové rozhraní přístupné, bude dostupná přes AP vysílaný ESP32.

4.6 Komunikační protokoly

Ke komunikaci mezi *backendovou částí* systému a *uživatelským rozhraním* bude použit protokol HTTP. Není zde zapotřebí komunikovat obousměrně, jelikož veškerá komunikace bude iniciována uživatelským rozhraním. Jedinou výjimkou je příjem aktuální polohy z lokátorů v reálném čase, kvůli čemuž je potřeba použít ještě jiný, plně duplexní a asynchronní protokol. K této komunikaci tedy bude vhodný protokol Web-

Socket, který navazuje spojení pomocí HTTP dotazu a odpovědi [17], a tak je snadné ho použít v implementaci aplikace, která již protkol HTTP ke komunikaci využívá.

Větší pozornost si zaslouží výběr komunikačního protokolu mezi *lokátorem* a *backendovým serverem*. Z hlediska zachování konzistence aplikačního rozhraní serveru přichází v úvahu využití stejného protokolu jako pro komunikaci s uživatelským rozhraním, tedy HTTP. Při bližším prozkoumání ale vyvstává problém s datovým tokem a výpočetní (tedy i energetickou) náročností. Zařízení bude sice napájeno přímo z vozidla, návrh však počítá s případnou možností rozšířit lokátor o bateriový provoz. Druhým argumentem, proč zvolit jiný komunikační protokol, je omezené množství dat, která lze přenést přes síť internet. Lokátor do ní bude připojen přes služby mobilního operátora, který si zpravidla účtuje poplatky za množství přenesených dat (podrobněji vizte sekci 4.9). Při každém HTTP dotazu se kromě výpočetně drahého navázání spojení (avšak toto spojení je nutné navázat bez ohledu na protokol) zároveň přenese velké množství režijních dat, které lze ušetřit volbou protokolu vhodnějšího.

Z těchto důvodů bude pro komunikaci mezi *lokátorem* a *serverem* zvolen protokol MQTT. Jeho režijní náklady jsou menší. Hlavička zprávy má pevnou velikost pouze 2 B, následovanou názvem kontextu (libovolné nenulové délky) a volitelnou část s přihlašovacími údaji [6]. Zbytek zprávy jsou již data nesoucí relevantní informaci, tzv. payload. Podrobnější porovnání protokolů MQTT a HTTP popisuje sekce 4.6.3. Důsledkem této volby je také fakt, že *backendový server* nebude muset být přístupný z veřejného internetu. Stačí pouze, aby byl veřejně přístupný pouze MQTT broker, který může být hostovaný např. v cloudu. Tím ovšem přichází případný uživatel o jednu z hlavních výhod, které toto řešení bude mít, a sice nesdílení citlivých údajů o své poloze se službami třetích stran.

4.6.1 MQTT

MQTT je binární protokol, který byl vyvinut firmou IBM v roce 1999. Prvotním účelem bylo navrhnout spolehlivý, jednoduchý protokol s nízkými nároky na datový tok, který bude sloužit pro komunikaci se senzory na ropovodu [13].

Vzor publish/subscribe, který protokol implementuje, umožňuje klientu připojit se k brokeru (server) a odebírat zprávy z vybraných kontextů (v originálu topics). Zároveň může zprávy do těchto kontextů také odesílat. Broker zde funguje jako router, neboť ihned poté, co jsou do některého z kontextů nahrána data, je přepošle všem klientům, kteří daný kontext odebírají. Tato vlastnost zajišťuje minimální zpoždění v komunikaci mezi jednotlivými klienty.

Je nutné zdůraznit, že ačkoliv protokol samotný podporuje zabezpečení pomocí přihlašovacího jména a hesla, komunikace mezi klientem a brokerem není nijak šifrována. Přihlašovací údaje jsou posílány ve zprávě jako pole znaků zakódovaných v UTF-8 [6]. Případnému útočníkovi tedy stačí odposlechnout inicializaci spojení mezi klientem a brokerem a získá přihlašovací údaje. Stejně tak může útočník odposlechnout veškerou kominukaci mezi brokerem a klientem. Zodpovědnost šifrovat komunikaci se tedy přenáší na transportní protokol.

Jak již bylo zmíněno, velkou výhodou tohoto protokolu je možnost přenést více zpráv v jenom spojení s minimem režijní informace. Není tedy nutné před odesíláním každé jedné zprávy inicializovat spojení mezi klientem a brokerem. Na začátku komunikace stanoví klient timeout, maximální doba mezi dvěma *PINGREQ* pakety. Pokud je tato doba překročena, broker uzavře spojení, pro případnou další komunikaci je nutné spojení vytvořit znovu. Ve specifikaci protokolu je tento mechanismus nazývaný jako KeepAlive [6]. Jeho důsledkem je i schopnost přestát krátkodobý výpadek spojení.

Další klíčovou vlastností protokolu je možnost nastavit u každého spojení kvalitu služby (QoS). Toto nastavení popisuje garanci doručení dat z klienta na server. Protokol rozlišuje tři úrovně. QoS 0 – zpráva je doručena nejvýše jednou (není zaručeno její doručení), QoS 1 – zpráva je doručena alespoň jednou (zaručeno doručení, ale může docházet k duplikacím) a konečně QoS 2 – zpráva je doručena právě jednou [6].

4.6.2 HTTP

„Protokol HTTP byl navržen, aby byl jednoduchý a lidsky čitelný.“ ([38], překlad autor). Z toho důvodu je protokol textový – informace nejsou kódovány binárně, ale jako strukturovaný text. Každý HTTP dotaz obsahuje hlavičku, která je opět tvořena textem a není definována její přesná délka. Může totiž obsahovat několik nepovinných částí, které blíže popisují a dodávají kontext (metadata) tělu dotazu. Tento koncept umožňuje libovolné rozšíření dotazu o vlastní hlavičky, a tím zlepšení škálovatelnosti protokolu.

Důležitou charakteristikou je bezstavovost protokolu. Jednotlivé dotazy nemají mezi sebou žádnou vazbu. Neznamená to ale, že by protokol nebyl schopný udržovat relaci s klientem. K tomu slouží hlavička *Cookies*. V ní lze předávat kontext komunikace a metadata pro pro jednoznačné rozlišení relací [18]. Příkladem může být e-shop, kde v této hlavičce lze předávat např. identifikátor přihlášeného uživatele.

Pro zlepšení výkonnosti protokolu není nutné navazovat pro každý dotaz explicitní spojení. Tím se sníží množství přenesených dat o množství nutné pro vytvoření spojení mezi komunikujícími subjekty. Jedno spojení je využito pro více dotazů. Ve verzi HTTP/1.1 lze konfigurovat maximální délku a množství dotazů přenesených přes jedno takové spojení [18]. Novější verze tuto konfiguraci pomocí hlaviček neumožňuje, nicméně je již implicitně implementována přímo v protokolu HTTP/2 [9].

4.6.3 MQTT vs. HTTP

Pro lepší znázornění náročnosti jednotlivých protokolů na datový tok, potažmo na výpočetní výkon, prezentuje tato sekce porovnání hlavní myšlenky obou protokolů (MQTT a HTTP) společně s experimentálním měření T. Yokotaniho a Y. Sasakiho [58].

Již z popisu protokolu MQTT, resp. HTTP v sekci 4.6.1, resp. 4.6.2, lze vyčíst zásadní rozdíl ve velikosti hlaviček zpráv (resp. dotazů) jednotlivých protokolů. První zmíněný protokol v ní obsahuje pouze několik příznaků, které charakterizují obsah a typ zprávy. Ty zabírají 2 bajty. Dále se posílá název kontextu, kódování jako UTF-8 řetězec, do kterého je daná zpráva odesílaná. Tato informace je však už užitečnou informací a není nutná pro správné fungování protokolu. Nelze ji tedy považovat za režijní informaci.

Oproti tomu druhý zmiňovaný protokol tuto velikost 2 bajtů přesáhne už u prvních dvou znaků hlavičky. Ty bude hlavička obsahovat zaručeně. Každý HTTP dotaz totiž jako první uvádí svůj typ, kde nejkratší možný název typu obsahuje tři znaky – GET. Typ zprávy je následován cílem dotazu (nejčastěji URL). Pro porovnání protokolů lze cíl chápat jako kontext u MQTT. Lze nahlédnout, že při odesílání většího množství dat bude rozdíl podílů užitečných a režijních dat protokolů HTTP a MQTT stále větší.

Toto tvrzení podporuje měření T. Yokotaniho a Y. Sasakiho [58], kde autoři mimo jiné měřili množství přenesených režijních dat obou protokolů. V každém protokolu odesílali prázdné zprávy do topicu o velikosti jednoho bajtu z různého počtu klientů (10, 100, 1 000). Měřením dochází k závěru, že protokol HTTP, za zmíněných podmínek, přenese více než dvakrát tolik režijních dat než protokol MQTT.

4.6.4 WebSocket

Komunikační protokol WebSocket využívá pro svůj přenos transportní protokol TCP. Inicializační handshake pro navázaní spojení je tvořen HTTP dotazem s odpovědí. Klient nejprve odešle na server HTTP GET dotaz s hlavičkou `Upgrade` a `Connection`, verzí WebSocket protokolu a náhodným 16bajtovým klíčem zakódovaným v Base64 pro identifikaci daného spojení. Server na něj odpoví status kódem `101 Switching Protocols` a hlavičkami `Upgrade`, `Connection`, `Sec-WebSocket-Protocol` a `Sec-WebSocket-Accept` [17]. Ukázku takové inicializace nabízí přímo RFC6455 [17] a je vypsána níže:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Hodnota hlavičky `Sec-WebSocket-Accept`, zakódovaná v Base64 a odeslaná v odpovědi serveru, je vypočtený SHA-1 haš řetězce, který vznikl spojením hodnoty hlavičky `Sec-WebSocket-Key`, odeslané klientem, a UUID definovaným ve specifikaci protokolu WebSocket. Tímto mechanismem se ověří, že obě strany komunikace podporují protokol WebSockets, jinak by neznaly UUID a vypočtené haše by se neshodovaly.

Po této inicializaci lze přes otevřené spojení obousměrně odesílat data. Protokol WebSocket odesílá data jako sekvenci rámců. Každý rámec, který přenáší pouze payload a nikoliv hlavičku, musí být maskován logickou operací XOR náhodnými 32 bity. Tento mechanismus implementuje protokol proto, aby se předešlo ukládání přenášených rámců do cache [17].

4.7 Kódování zpráv

Popisované protokoly HTTP a MQTT definují způsob výměny dat mezi dvěma zařízeními. Neříkají nic o struktuře přenášené informace. Zprávy tedy musejí být strukturovány zvlášť. I zde je potřeba volit pro dvě použití, komunikaci *backendu* s *uživatelským rozhraním* a komunikaci *backendu* s *lokátorem*.

Mezi notoricky známé strukturování těl HTTP dotazů patří formáty XML a JSON. S ohledem na vybraný implementační jazyk *uživatelského rozhraní*, JavaScript framework Vue.js, je jistě výhodnější využít pro kódování zpráv formát JSON, který je výchozím formátem serializace objektů právě pro jazyk JavaScript. Tuto volbu podporuje i fakt, že přenášená data mezi *uživatelským rozhraním* a *backendovým server* nebudou vyžadovat příliš složitou strukturalizaci. Půjde v zásadě o jednoduché objekty, případně pole takových objektů.

Ačkoliv formát JSON má menší vyjadřovací schopnosti, není totiž tolik rozšiřitelný, vyniká v jednoduchosti, minimu režijní informace a rychlosti parsování [41]. Tyto vlastnosti dělají z formátu JSON dobrého kandidáta i při výměně dat mezi *sledovacím zařízením* a *backendovým serverem*. Stále se ale jedná o textový, lidsky čitelný formát. Zajisté tedy nese jistou míru redundantní informace. Například následující JSON dokument s vynecháním bílých znaků má velikost 62 B.

```
{
   "name"      : "John",
   "lastname"  : "Doe",
   "age"       : 35,
   "lastLogin" : 933026400
}
```

Stejná data zakódovaná v binárním Protobuf mají velikost 19 B.

```
syntax = "proto2";

package protocol;

message Message {
  required string name     = 1;
  required string lastname = 2;
  required int64 age       = 3;
  optional int64 lastLogin = 4;
}
```

Přestože toto srovnání není reprezentativním vzorkem, porovnává pouze jeden náhodný vstup, lze pozorovat, že zápis čísel v textovém formátu přináší značnou redundanci přenášené informace. Podrobnější popis, proč je druhé kódování úspornější, nabízí sekce 4.7.1, popisující způsob kódování Protobuf. Aby bylo srovnání spravedlivé, je důležité zdůraznit, že pro správné fungování Protobuf musejí obě strany komunikace znát strukturu zprávy předem. Formát JSON je možné dekódovat, i pokud je struktura zprávy neznámá. Je ale k diskusi, k čemu jsou data, která systém nezpracovává v žádném kontextu a mají tak pro něj nulovou vypovídající hodnotu.

4.7.1 Protobuf

Serializační mechanismus Protobuf, od společnosti Google, Inc. je jazykově neutrální a na rozdíl od strukturovaní pomocí XML nebo JSON přináší lepší kompresi přenášené informace [21]. Na úkor toho ale není zakódovaná zpráva lidsky čitelná. I přístup ke kódování je zcela odlišný. Uživatel nejprve definuje strukturu protokolu, jak mají zprávy vypadat. Tato definice se provádí pomocí dvojic klíč–hodnota, např. `optional string name = 1` obalených direktivou `message`. Velice jednoduchá zpráva pak může vypadat například takto:

```
message BasicMessage {
    optional string name = 1;
    optional int32  age  = 2;
}
```

První část klíče (`optional`/`required`) říká, zda je odesílaná položka ve zprávě vyžadována, či nikoliv. Při serializaci zprávy nehraje žádnou roli. Je zde pouze proto, aby

vytvořená zpráva byla před serializací validována, zda obsahuje vše vyžadované protokolem. Druhá část, `string`, určuje datový typ (protokol rozlišuje šest typů) a konečně poslední, `name`, určuje název položky, který může být libovolný a také nehraje při serializaci žádnou roli. Je zde pouze jako identifikátor použitý při definici struktury zprávy a jednoduší přistupování k jednotlivým položkám ve zdrojovém kódu. Výslednou, takto definovanou zprávu poté uživatel přeloží kompilátorem `protoc` do požadovaného programovacího jazyka. Kompilátor vygeneruje potřebnou implementaci zprávy, společně se serializační, resp. deserializační rutinou. Příklad podporovaných jazyků, do kterých lze zprávu přeložit, jsou C, Go a Scala [21].

Protokol rozřazuje datové typy do šesti skupin. Každá skupina využívá pro své datové typy jiný typ kódování. Výčet, které typy jsou jak kódovány, je popsán v tabulce 4.1.

typ	enkódování	datový typ
0	Varint[1]	int32, int64, uint32, uint64, sint32, sint64, bool, enum
1	64-bit	fixed64, sfixed64, double
2	Length-delimited	string, bytes, embedded messages, packed repeated fields
3	Start group	groups (deprecated)
4	End group	groups (deprecated)
5	32-bit	fixed32, sfixed32, float

Tabulka 4.1. Datové typy Protobuf [22]

V tabulce 4.1 lze pozorovat, že Protobuf kóduje celá čísla (typ 0) kódováním Varint. Toto kódování umožňuje, aby celá čísla zabírala méně místa v paměti. Podrobný popis toho, jak toto kódovaní funguje, lze nalézt na stránce 19. Pokud je potřeba zakódovat celé číslo se znaménkem, je výhodnější využít `sint64` nebo `sint32` datový typ. Na tyto datové typy je nejprve použito ZigZag kódování, kterým jsou kódovaná čísla převedena na datový typ bez znaménka, na který je následně aplikováno Varint kódování. Tento přístup je vhodnější než ukládat záporná čísla jako `int64`, resp. `int32`, neboť reprezentace takového čísla v tomto formátu by měla vždy 10 bajtů, a tudíž by byla méně efektivní [20]. Všechny ostatní typy čísel jsou uloženy ve fixní délce 64 bitů, resp. 32 bitů v little-endian[2].

Specificky jsou kódována pole znaků (řetězce). Jejich hodnota je zakódovaná jako délka v bajtech (zakódovaná kódováním Varint), následovaná takto definovaným počtem bajtů (znaky řetězce jsou kódovány jako UTF8). Stejným způsobem (typ 2 v tabulce 4.1) jsou kódovány vnořené zprávy a opakovaná pole.

ZigZag

Kódování ZigZag převede záporná čísla na kladná, přičemž pro tento převod využije aritmetický posun[3] (`>>`, `<<`) a logickou operaci XOR (`^`) [20].

```
(n << 1) ^ (n >> 31) pro sint32
(n << 1) ^ (n >> 63) pro sint64
```

[1] pro více informací o Base 128 Varints vizte stranu 19

[2] Little-endian je takové řazení bitů, kde LSB je uložen na nejnižší adresu. Ostatní bity jsou postupně ukládány až po MSB, který bude na nejvyšší adrese paměťového bloku, na kterém jsou takto seřazená data uložena.

[3] aritmetický posuv vpravo duplikuje MSB a LSB se zahazuje, zatímco aritmetický posuv vlevo doplňuje na místo LSB 0 a MSB se zahazuje

Base 128 Varints

„Varints are a method of serializing integers using one or more bytes. Smaller numbers take a smaller number of bytes“ [22] (volně přeloženo jako „Varints jsou způsob, jakým serializovat celá čísla za použití jednoho nebo více bajtů. Menší čísla zabírají menší množství bajtů“).

Každý bajt kromě posledního má ve Varint nastavený MSB[4] na 1. Tento příznak indikuje, zda další bajt v bajtstreamu patří stále k číslu kódovanému pomocí Varint, nebo je již součástí jiného logického celku. Zbylých (spodních) 7 bitů každého bajtu je použito pro zakódování samotného čísla ve dvojkovém doplňku po 7bitových skupinách od nejnižších 7 bitů po nejvyšší [22].

Pro lepší názornost, jak toto kódování funguje, je zde uveden příklad zakódování/dekódování čísla 277. Číslo 277 má ve dvojkové soustavě zápis 100010101_2.

```
rozdělíme 100010101 do 7bitových skupin (do nejvyšší doplníme zleva 0)
000 0010 001 0101
následně tyto skupiny prohodíme (kóduje se od LSB)
001 0101 000 0010
nakonec přidáme před první skupinu jako nejvyšší bit 1 a před druhou 0
1001 0101 0000 0010
výsledkem jsou dva bajty reprezentující číslo 277 jako Varint
```

4.8 Zabezpečení a autorizace

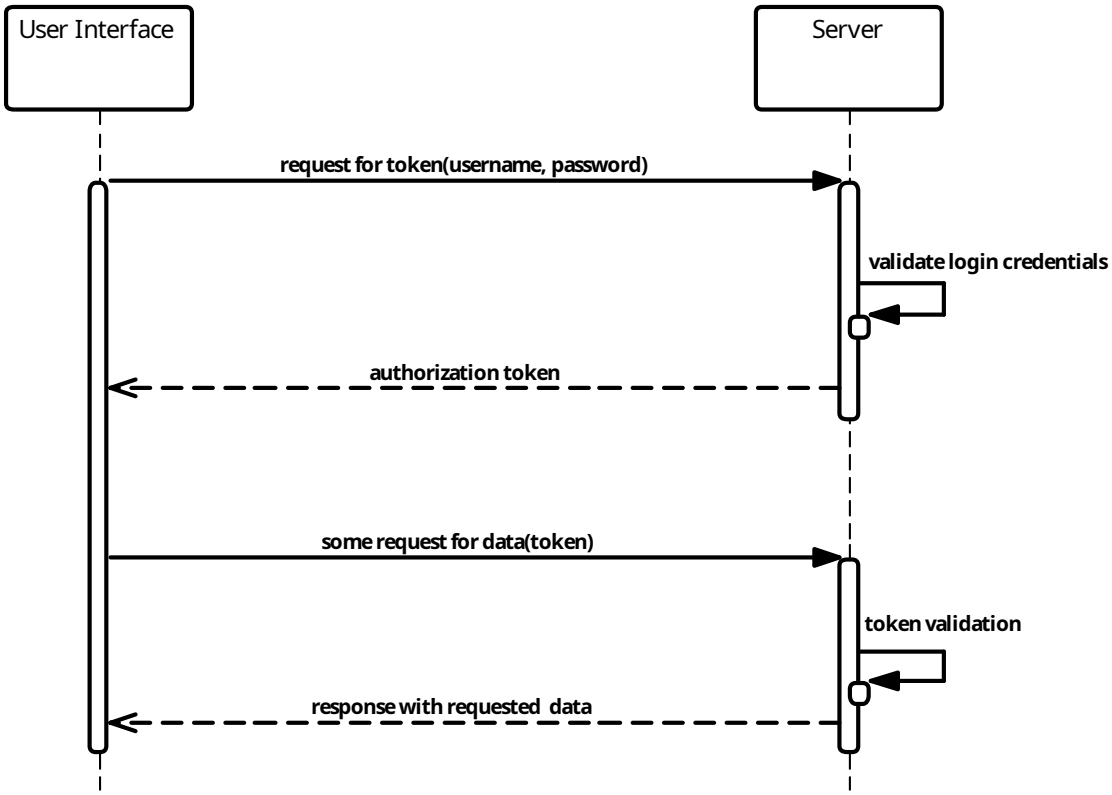
Veškerá komunikace bude zabezpečena na úrovni páté (relační) vrstvy ISO/OSI[5] modelu, za pomoci protokolu TLS. Tento protokol se řadí mezi kryptografické, zabezpečující komunikaci mezi koncovými uživateli. V této práci bude využit pro šifrování komunikace mezi *backendovým serverem* a *lokátorem*, stejně pak mezi *backendovým serverem* a *uživatelským rozhraním*. Pro tato spojení bude využito asymetrického šifrování bez využití klientského certifikátu.

Popsaný mechanismus zajistí bezpečnost komunikace, nikoliv však autentizaci klienta (uživatele). Aby bylo zajištěno zabezpečení dat a nebyla poskytnuta neoprávněnému uživateli, bude nutné, aby se každý uživatel nejprve autentizoval uživatelským jménem a heslem. Poté obdrží JWT (pro více informací o JWT tokenech vizte sekci 4.8.2) token s předem definovanou platností. Tento token poté bude muset klient odeslat s každým dotazem na server. Vystavený token bude podepsán klíčem, který je znám pouze *backendovému serveru*. Pokud by byl token modifikován, server by tuto změnu detekoval, neboť by neseděl podpis dat, a protože klíč je tajný, podpis nemůže být podvrhnut. Návrh celého mechanizmus autentizace a autorizovaného dotazu je znázorněn na obrázku 4.4.

Ukládání uživatelských hesel

Další krok ke zvýšení zabezpečení systému je hašovaní ukládaných uživatelských hesel. Při úniku dat z databáze poté nebude útočník schopen tato hesla přečíst a zneužít. Nicméně nadále může využít útoku hrubou silou, např. s použitím rainbow tables nebo slovníkového útoku. Navíc, pokud prolomí jedno heslo, které používá více uživatelů, bude (z vlastnosti hašovací funkce plyne, že bude mít stejný haš) mít potenciálně přístup k více účtům.

[4] Most Significant Bit (česky nejvýznamnější bit)
[5] normalizovaný v ČSN ISO/IEC 7498-3 [29]



Obrázek 4.4. Sekvenční diagram návrhu autorizování a autentizace

Tento nedostatek lze vyřešit přidáním náhodné sekvence bytů k uživatelským heslům, tzv. sůl (anglicky salt), a poté, na takto upravené heslo, použít hašovací funkci. Útočník už nemůže použít útok pomocí rainbow tables, neboť hašované řetězce mají v sobě náhodnou část. Sůl je nutné si ukládat stejně tak jako výsledný haš, jinak by při validaci hesla (např. při přihlášení uživatele) nebylo možné vypočíst haš, který je uložen v databázi.

Konkrétní možností, jak tento mechanismus implementovat, je využití funkce PBKDF2. Ta na základně vstupního hesla, soli a počtu iterací vypočítá klíč předem definované délky. Při dostatečném počtu iterací vznikne klíč, který je odolný jak proti útoku hrubou silou, tak i proti útoku metodou rainbow tables [37].

4.8.1 TLS

TLS je kryptografický protokol, který zajišťuje šifrované end-to-end spojení. Komunikace je šifrována a dešifrována na koncových zařízeních. Kdokoliv jiný komunikaci odchytí, není schopen ji dešifrovat bez znalosti klíčů vyměněných během inicializace spojení. Pokud by byla odposlechnuta inicializace, ani tak není možné komunikaci dešifrovat. Jedna část klíče je totiž odeslána šifrovaně a může ji dešifrovat pouze zařízení, pro které byla určena. Mechanismus, kterým je toto zabezpečení zajištěno, popisuje následující text.

Protokol kombinuje symetrickou a asymetrickou kryptografii, a tím zajišťuje kromě vysokého zabezpečení také dostatečný výkon. Po dohodnutí obou komunikujících stran na použitých šifrách a verzích protokolu (kde je využita asymetrická kryptografie – serverový, resp. klientský certifikát) je všechna komunikace šifrována kryptografií symetrickou. Celá komunikace včetně úvodního nastavení spojení může fungovat i v režimu pouze s využitím symetrické kryptografie, kde využívá pro výměnu šifrovacích klíčů

algoritmu Diffie–Hellman [14]. Tento režim nebude v implementaci použit, a proto zde není více popisován.

Před začátkem šifrované komunikace si zařízení musejí vyměnit šifrovací klíč, kterým budou šifrovat následnou komunikaci. Klient nejprve odešle na server *ClientHello* zprávu s náhodně vygenerovaným *client random*. Mimo náhodné byty odešle klient také svou verzi protokolu a šifry, které podporuje. Tím inicializuje komunikaci. Server odpoví *ServerHello* zprávou společně s náhodně vygenerovaným *server random* a vybraným typem šifry ze seznamu poskytnutého klientem. Následně odešle zprávu *Certificate* s veřejným certifikátem a hned poté *ServerHelloDone* (tím indikuje klientu, že další komunikace bude pokračovat podle domluvených mechanismů). Tento certifikát musí být podepsán některou certifikační autoritou[6], nesmí být expirovaný a revokovaný. Všechny tyto podmínky klient ověří. Pokud je certifikát neplatný, klient ukončí se serverem spojení, jinak pokračuje následujícím. Vygeneruje tzv. *premaster secret*, které zašifruje veřejným klíčem serveru (derivovaného z certifikátu) a zašifrovaná data odešle serveru, tzv. *ClientKeyExchange*. Jediný, kdo tato data může rozšifrovat, je server, protože on jediný má privátní klíč. S jeho pomocí dešifruje příchozí zprávu. Poté klient i server vypočítají ze známých dat, *server random*, *client random* a *premaster secret*, pomocí stejného algoritmu *session id*. Poté, co klient vypočítá *session id*, pošle serveru *ChangeCipherSpec* zprávu. Toto je předposlední zpráva inicializace ze strany klienta a indikuje serveru, že veškeré další odeslané zprávy budou již šifrovány dohodnutým způsobem. Konečně klient odešle serveru *Finish* zprávu, která je již zašifrována pomocí *session id*. Server dešifruje *Finish* zprávu od klienta. Pokud se mu to nepodaří, ukončí spojení. Jinak také odešle *Finish* zprávu zašifrovanou klíčem *session id*. [14]

Pro lepší znázornění komunikace je celý tento proces navázání spojení zobrazen na obrázku 4.5.

4.8.2 JWT

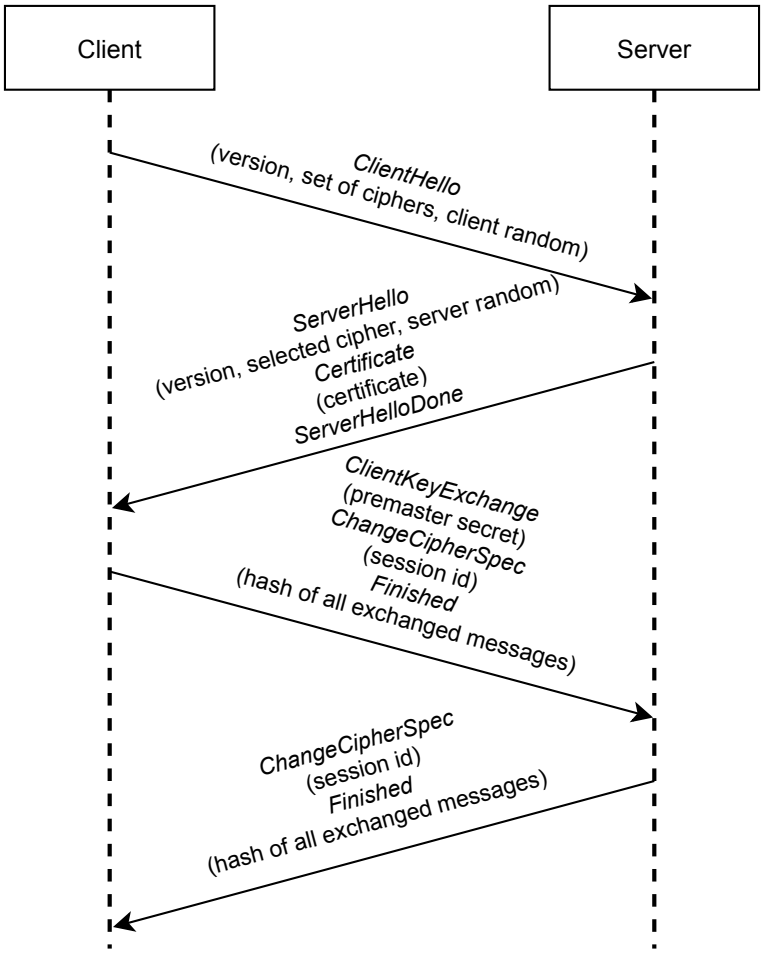
JSON Web Token (JWT) je kompaktní zápis přenášené informace, kódovaný v Base64. Payload JWT je množina claims, reprezentována jako JSON dokument. Může obsahovat nula a více dvojic klíč–hodnota. Některé klíče určuje přímo specifikace. Jsou jimi `iss` (Vydavatel), `sub` (Předmět), `aud` (Příjemce), `exp` (Čas expirace), `nbf` (Neplatný před), `iat` (Čas vytvoření), `jti` (JWT ID) a žádný z nich není povinný. Protože je tato množina libovolně rozšiřitelná, JWT tak umožňuje kódovat libovolná data.

Vlastní token je reprezentován jako sekvence URL bezpečných částí oddělených tečkou. Počet částí záleží na tom, zda je použit JWS (JSON Web Signature) [32] s JWS Compact Serialization, nebo je použit JWE (JSON Web Encryption) [33] s JWE Compact Serialization [31].

Pro implementaci bude vhodnější využít JWS. Data v tokenu nepotřebují být šifrována. O to se postará šifrovací protokol TLS. Nicméně je nutné vědět, že token nebyl podvrhnut. Proto se využije metoda podpisu (JWS) namísto metody šifrování (JWE).

Při použití podpisu je struktura JWT následující. První část obsahuje JOSE hlavičku, opět zapsanou jako JSON dokument a kódovanou v Base64. V ní musí být definován použitý podpisový algoritmus definovaný ve specifikaci RFC7518 [30] s klíčem `alg` a typ tokenu s klíčem `typ`. Tento typ může být použit k indikaci, zda se jedná o JWS nebo JWE [31]. Druhá část tokenu je vlastní payload obsahující claims, opět jako JSON zakódovaný v Base64. Poslední, třetí část, obsahuje HMAC vypočítaný ze zakódované hlavičky a payloadu s využitím algoritmu definovaného v hlavičce (`alg`) s použitím

[6] Certifikační autorita musí být klientu známa.



Obrázek 4.5. Schéma navázaní zabezpečeného spojení protokolu TLS

libovolného[7] klíče [31]. Výsledná podoba JWT tokenu je získána spojením těchto třech částí oddělených tečkou, jak bylo popsáno výše.

4.9 Mobilní internetové připojení

Jak již bylo několikrát zmíněno, *lokátory* budou komunikovat se zbytkem systému přes síť internet. K tomu je nutné zajistit každému z nich internetové připojení. To bude realizováno prostřednictvím služby GPRS dostupné přes mobilní operátory.

V České republice dominují na trhu s mobilními daty tři telekomunikační společnosti – *O2 Czech Republic a.s.*, *T–Mobile Czech Republic, a.s.*, *Vodafone Czech Republic a.s.* V nabídce každé lze nalézt nepaušální datové tarify. Ty lze zřídit prostým zakoupením předplacené karty a dobitím kreditu. Tato možnost také bude využita při realizaci *lokátoru*. Službu lze kdykoliv zrušit a poskytuje výhodné cenové podmínky.

Tabulka 4.2 zobrazuje přehled tarifů od výše zmíněných telekomunikačních společností vhodných pro sledovací systém. Zahrnuje pouze nabídky přímo určené pro IoT zařízení, označované také jako M2M[8] tarify. Oproti běžným tarifům umožňují tyto zakoupení menšího objemu dat, a tím snižují paušální náklady. Uváděné ceny jsou platné k 1. 5. 2021 s uzavřením smlouvy o jejich odběru na dobu neurčitou.

[7] klíč musí být zarovnán na velikost bloku použité hašovací funkce
[8] machine to machine

společnost	objem dat [MB]	cena [Kč]
O2	1	37,51
	10	55,66
	20	61,71
	50	67,76
	100	110,11
T–Mobile	neomezeně	18,15 + 21,02 za MB
Vodafone	100	49

Tabulka 4.2. Nabídka mobilního internetového připojení

Zmíněné služby jsou platné pouze na území České republiky. Aby bylo možné využívat *lokátor* po celém světě, je nutné využít služby jiného poskytovatele, jako je třeba Hologram, Inc. Tato společnost poskytuje mobilní připojení pro IoT zařízení téměř po celém světě [28]. Poplatky za datový přenos účtuje stejně jako čeští operátoři, tedy za množství přenesených dat.

Společnost Things Mobile Srl nabízí v jednom ze svých produktů (Unlimited [52]) zcela odlišný přístup k účtování poplatků. Ty nejsou účtovány za množství přenesených dat, ale je omezena rychlost datového toku na 32 kbps. Za tuto službu zaplatí zákazník měsíční poplatek ve výši 30 €[9] [52] a je dostupná ve všech zemích Evropy [48].

4.10 Ukládání dat

Pro ukládání dat bude zvolena relační databáze. Schéma entit a vztahů mezi nimi je pevně stanoveno (vizte 4.4). Databázové schéma tak přesně reflektuje doménový model. Pokud by byla struktura volnější a vztahy mezi entitami nebyly jednoznačně určeny, připadalo by v úvahu přemýšlet, zda nezvolit pro implementaci nerelační databázi. Nicméně to není tento případ.

Z mnoha dostupných databázových relačních enginů bude pro implementaci vybrán H2. Výhodou toho enginu je možnost provozu jak v embedded, tak standalone módu. Může být tedy použit přímo bez nutnosti nasazovat databázový server. Je napsán v jazyce Java a má podporu JDBC API, v případě potřeby tak lze změnit databázový stroj i engine změnou JDBC připojení. Další vlastností výhodnou pro implementaci je kompatibilita s několika dialekty SQL dotazovacího jazyka [26].

4.11 Nasazení systému

V této sekci se systém dělí pouze na dvě části, *serverový backend* a *lokátor*. Vyplývá tak z analýzy architektury systému. V sekci 4.5.2 jsou popsány důvody, proč je možné a výhodné mít součástí backendové business logiky také webový server. Toto sloučení eliminuje nutnost nasazovat uživatelské rozhraní zvlášť na dedikovaný webový server.

Pro nasazení a provozování systému bude použit virtualizační nástroj Docker, který izoluje jednotlivé aplikace do tzv. kontejnerů. Kontejner je zcela odstíněn od zbytku systému stroje (anglicky host), na kterém je spuštěn. Využívá separátní souborový systém poskytovaný tzv. container image. Na něm musí být uloženy všechny potřebné soubory

[9] 770,4 Kč k 9. 5. 2021

pro běh aplikace, včetně všech závislostí, konfigurací, zkompilovaných binárních souborů apod. Image obsahuje také veškeré systémové (kontejnerové) nastavení, jako jsou např. systémové proměnné [15]. Z toho vyplývá, že nastavení hostitelského systémů, včetně systémových proměnných, nemá na běh kontejnerů žádný vliv.

Pro sestavení Docker kontejneru lze s výhodou využít nástroje Docker compose. Ten za pomoci uživatelsky definovaného strukturovaného souboru vytvoří běhové prostředí pro aplikaci s požadovanými vlastnostmi. Nastaví systémové proměnné, namapuje kontejnerové porty na lokální, případně propojí několik kontejnerů apod. Konfigurační soubor `docker-compose.yml` je společně s popisem diskutován v kapitole 5.

U *lokátoru* nelze mluvit přímo o nasazení. Po zapojení a nahrání firmwaru je zařízení připraveno k použití. Veškerá další nastavení probíhají již jako procesy v aplikaci a jsou řízena přes uživatelské rozhraní.

Kapitola 5
Implementace

Implementace využívá všech poznatků získaných při analýze. V jednotlivých sekcích doplňuje informace, které z teoretické části nebyly zcela patrné, nebo je jejich popis až příliš závislý na konkrétní implementaci. Odkazy na takové sekce jsou v předchozí kapitole zmíněny, aby byla dodržena logická návaznost textu.

Kapitola je rozdělena na několik částí. První část popisuje důležitá fakta a implementační procesy, které jsou společné pro celý systém, případně nezbytné pro jeho propojení. Další sekce poté cílí na konkrétní části systému (rozděleného dle analýzy architektury v sekci 4.5).

5.1 Struktura zpráv

Část analýzy (4.6) v této práci diskutovala, který protokol je pro problematiku této práce vhodný. Implementace tyto poznatky reflektuje a na jejich základě je postavena struktura zpráv odesílaných *lokátorem* přes MQTT protokol na *backendový server*. Sekce také popisuje druhý diskutovaný problém – sdílení informace o nové poloze mezi *backendovým serverem* a *uživatelským rozhraním* s využitím protokolu WebSocket.

Lokátor sbírá vzorky aktuální polohy s frekvencí 4 sekundy (toto je výchozí hodnota, která může být uživatelem změněna). Aby se ušetřil datový tok, jsou jednotlivé vzorky odesílány v balících po 5 (toto je výchozí hodnota, která může být uživatelem změněna), tzv. `Report`. Balík dat obsahuje kromě pozic i další metadata – autorizační token, který *lokátoru* vygeneroval *backendový server*. Tím bude zaručena autentizace, resp. autorizace vůči serveru. Dále obsahuje `sessionId`, náhodných 16 bajtů vygenerovaných při zapnutí lokátoru. Tento identifikátor je poté použit na straně serveru pro identifikaci nové trasy (více v sekci 5.2.4). Poslední dvě informace, které `Report` obsahuje, jsou `vehicleId` (identifikátor vozidla, ke kterému je tracker přiřazen) a `timestamp`, datum a čas sestavení `Reportu`. Pokud by se pozice posílaly ihned po naměření, veškerá tato metadata by se musela odesílat s každou z nich.

V ukázce kódu 5.1 je popsána struktura `Reportu` ve formátu Protobuf ve verzi 2. Položky označeny jako `required` jsou požadovány v každé zprávě a v případných aktualizacích struktury je nelze nikdy vynechat. Z toho důvodu je povinný pouze autorizační token a položky, které identifikují pozici (`TrackerPosition`). Dále je dobré si povšimnout, že indexy položek mají mezi sebou „mezery“. Ty vytvářejí dostatečný prostor pro další rozšíření. Ve struktuře zprávy totiž nelze kvůli zachování zpětné kompatibility jednotlivé položky mazat, stejně tak měnit jejich index. Novou položku tak lze přidat zařazením na kterýkoliv nevyužitý index. Nevyužívanou položku pak nelze nijak modifikovat a obě strany komunikace ji začnou ignorovat. Ze stejného důvodu není vhodné označovat nové položky jako povinné (`required`). Takové položky totiž nelze vynechat ani v případě, že už nejsou potřebné.

Druhým místem, kde je nutné navrhnout strukturu přenášených zpráv, je komunikace mezi *uživatelským rozhraním* a *backendovým serverem* pro přenos dat o aktuální poloze. Z analýzy vyšel jako vhodný protokol WebSocket. Komunikace tak bude asynchronní

Kód 5.1.

```
syntax = "proto2";

package protocol;

message Report {
  required string token              = 1;
  optional bytes sessionId           = 4;
  optional int64 vehicleId           = 7;
  optional int64 timestamp           = 10;
  repeated TrackerPosition positions = 13;
}

message TrackerPosition {
  required int64 timestamp  = 1;
  required double latitude  = 4;
  required double longitude = 7;
  oprional double speed     = 10;
}
```

a případné potvrzování a odpovědi na zprávy je nutné vyřešit na úrovni aplikačního protokolu.

Zprávy o aktuální poloze a žádost o její odebírání jsou přenášeny ve formátu JSON (vizte ukázku kódu 5.2). Tělo zprávy vždy obsahuje její typ, kterým může být jednou z možností: `EMPTY`, `HEARTBEAT`, `TEXT`, `POSITION`, `SUBSCRIBE_POSITIONS`, `UNSUBSCRIBE_POSITIONS`, `ERROR` a `VEHICLE`. Struktura obsahu zprávy je jednoznačně určena jejím typem a veškeré detaily jsou popsány v dokumentaci kódu přiložené na paměťovém médiu. Aby *backendový server* věděl, na které spojení má odeslat která data o nových pozicích, uživatelské rozhraní se musí k jejich odběru nejprve přihlásit. To udělá tak, že odešle zprávu typu `SUBSCRIBE_POSITIONS`, přiloží svůj autorizační token a v těle uvede identifikátor vozidla, jehož pozice chce zobrazovat. Server požadavek validuje. Ověří platnost tokenu a existenci vozidla. Pokud je vše v pořádku, odešle *uživatelskému rozhraní* poslední známou pozici vybraného vozidla. Od této chvíle, pokud server obdrží `Report` s novými pozicemi od odebíraného vozidla, resp. lokátoru, odešle nejaktuálnější z nich na *uživatelské rozhraní*.

Kód 5.2.

```
{
    msgType: string
    token: string (or null)
    payload: JSON
}
```

Struktura dotazů na aplikační rozhraní *backendového serveru* je popsána v dokumentaci kódu přiložené na paměťovém médiu. Zde nebude více rozebírána, z důvodu velkého množství endpointů a struktury, která je snadno pochopitelná, bez nutnosti dalšího popisu.

5.2 Backendový server

Backendový server je implementovaný v programovacím jazyce Scala, který kombinuje funkcionální a objektový přístup. Implementace dodržuje třívrstvou architekturu, kde odděluje business, datovou a prezentační vrstvu. Základem programu je toolkit `scala-server-toolkit` [54] pro programování funkcionálního serveru v jazyce Scala. Ten pomáhá mj. s konfigurací knihovny `http4s` [56], která obsluhuje dotazy přicházející na server. Jelikož jsou všechny dotazy a odpovědi kódovány ve formátu JSON (dle analýzy 4.7), využívá se pro kódování, resp. dekódovaní přenášených objektů do, resp. z tohoto formátu, knihovna `Circe` [12].

V následujících sekcích jsou popsány nejdůležitější části softwaru. Pro lepší pochopení, jak systém funguje, a pro jeho správné využití je kromě implementace popsána i konfigurace a způsoby nasazení. Detailní popis aplikačního rozhraní zde není nutné uvádět, je totiž součástí dokumentace kódu.

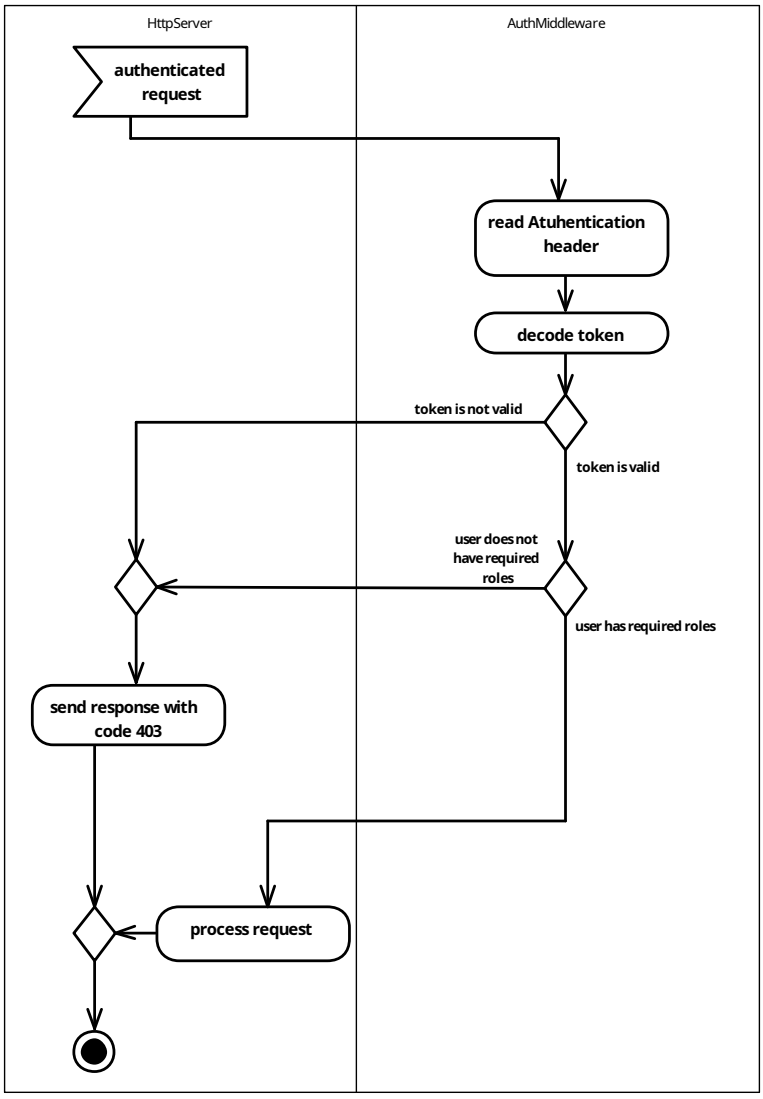
5.2.1 TLS/SSL certifikát

K zabezpečení komunikace je využit protokol TLS. Veškerá komunikace s *backendovým serverem* je tímto protokolem šifrována. Jelikož TLS využívá asymetrické kryptografie, je nutné získat pro *backendový server* certifikát podepsaný důvěryhodnou certifikační autoritou. Pro implementaci byla zvolena autorita Let's Encrypt vydávající TLS certifikáty zdarma s využitím ACME protokolu [7]. Uživatel, který o certifikát žádá, musí prokázat, že je vlastníkem domény, a projít bezpečnostní výzvou. K automatizaci tohoto procesu na straně uživatele slouží nástroj Certbot [16].

5.2.2 Autentizace/autorizace uživatele a lokátoru

Dotazy na server musejí být autorizovány, jinak by data mohl získat kdokoliv. Server ověřuje věrohodnost dotazu na základě JWT tokenu poslaného v hlavičce `Authorization`. Aby klient tento token získal, musí se nejprve autentizovat, na endpoint `/auth` odeslat HTTP dotaz typu POST. V těle zprávy uvede své přihlašovací údaje jako JSON dokument se dvěma hodnotami `username` a `password`. Server jeho požadavek přijme a snaží se ověřit správnost údajů. To znamená, že přijatou hodnotu `password` zahašuje se stejnou solí a počtem iterací, jako bylo heslo uloženo do databáze. Následně pak porovná vypočítanou hodnotu a hodnotu hesla uloženou v databázi. Pokud se hodnoty rovnají, odpoví uživateli se status kódem 200 a v těle odpovědi odešle informace o přihlášeném uživateli spolu s autorizačním tokenem. V případě, že je přijaté heslo nesprávné, nebo uživatel s přijatým `username` neexistuje, *backendový server* v odpovědi vrátí 403, Forbidden.

Získaný token má omezenou platnost a jak již bylo zmíněno, je nutné ho přikládat jako `Authorization` hlavičku při každém HTTP dotazu na *backendový server*. Server tuto hlavičku dekóduje a ověřuje, jestli klient má právo získat data, na která se dotazuje. Mechanismus ověření zajišťuje třída `AuthJwtMiddleware`, která využívá vlastnosti použité knihovny `http4s` nabízející možnosti tzv. middleware. Jde o metody, které jsou provedeny nad dotazem před tím, než je dále zpracováván. `AuthJwtMiddleware` tedy přečte z příchozího dotazu hlavičku `Authorization`, ověří její integritu, požadovaná práva a buď požadavek zamítne s kódem 403, nebo postoupí dotaz dalšímu zpracování. Activity diagram zpracování autentizovaného dotazu s využitím middleware zobrazuje diagram na obrázku 5.1.



Obrázek 5.1. Autentizovaný dotaz

JWT autorizační token

Autorizační token je posílán ve formátu JWT. Aby nemohl být podvrhnut, je pro jeho data spočítán HMAC-SHA256 s užitím tajného klíče známého pouze *backendovému serveru*. Haš je dle specifikace součástí JWT. Pokud by byla data změněna, jejich HMAC-SHA256 by se neshodoval s tím, který je součástí tokenu. *Backendový server* by tedy tuto skutečnost poznal a odpověděl kódem 403.

Obnovení platnosti tokenu

Je na zodpovědnosti uživatele, resp. *uživatelského rozhraní*, aby před vypršením platnosti tokenu požádal o nový. Pro tento účel slouží endpoint `/auth/refresh`, který po přijmutí HTTP GET dotazu s přiloženým autorizačním tokenem v hlavičce `Authorization` vrátí nový token. Všechny tokeny jsou vydávány se stejnou životností. Proto je nutné tuto obnovu provádět opakovaně.

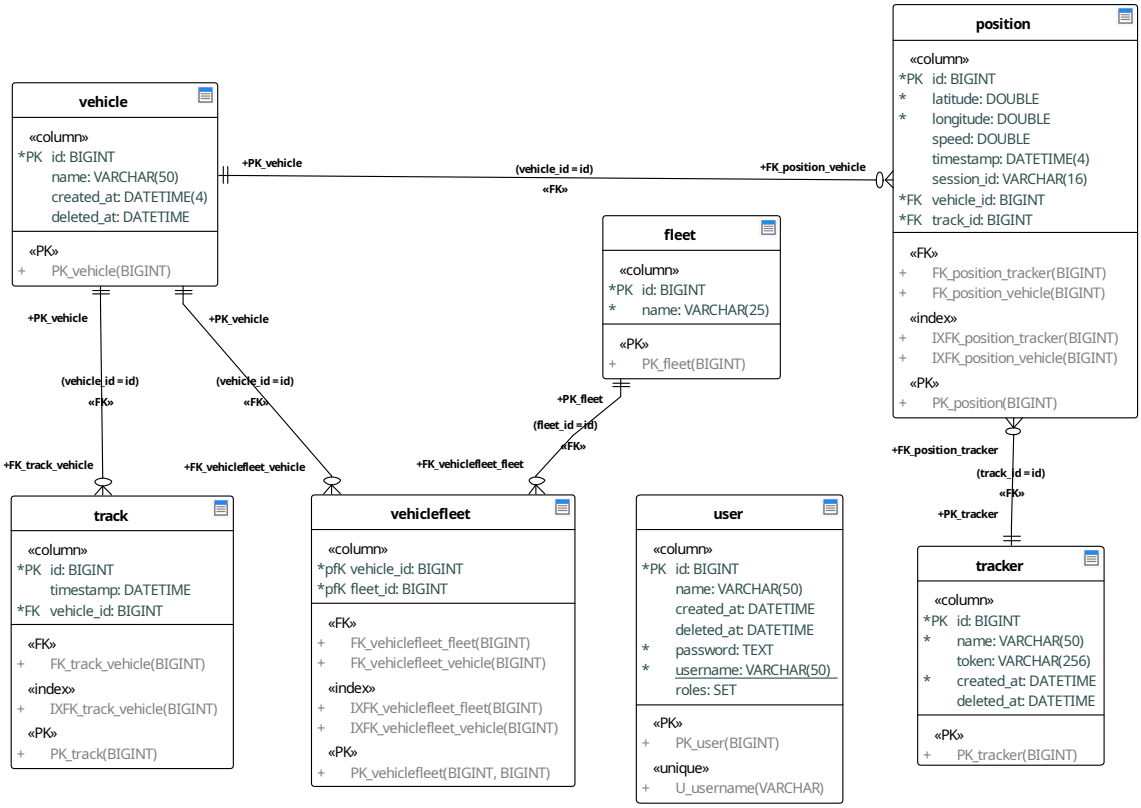
Token není vydáván na dobu neurčitou z důvodů bezpečnosti. Pokud uživatel neprovede prodloužení, nebude aktivní, bude po předem definované době (systémová proměnná `JWT_EXPIRATION`) „odhlášen“. Pro další práci se systémem bude nutné uživatele znovu autorizovat.

Naopak tokeny pro lokátory jsou vydávány na dobu neurčitou. Zde tento fakt nijak neovlivní bezpečnost, neboť získání tokenu nahraného v lokátoru vyžaduje fyzický přístup k zařízení a přečtení jeho EEPROM paměti, ve které je uložen. Protože je veškerá komunikace s *backendovým serverem* šifrována, nelze token zjistit ani odposlechnutím komunikace. Ušetří se nutnost token aktualizovat a tím i datový tok. Pokud by i přesto došlo ke kompromitování lokátoru, lze token revokovat na straně serveru.

5.2.3 Databáze

Pokud databáze neexistuje, aplikace ji vytvoří. Následně spustí `create.sql` skript, který vytvoří potřebnou strukturu a závislosti mezi tabulkami. Po inicializaci jsou všechny tabulky prázdné.

Server interaguje s databází pomocí JDBC API, který je standardem pro tuto komunikaci v jazyce Java/Scala. Nepřistupuje k němu však přímo, ale přes knihovnu *doobie*. Tato knihovna umožňuje čistě funkcionální přístup k JDBC API, a navíc má rozšířenou podporu pro H2 databázový engine, který byl při analýze zvolen jako vhodný (4.10). Strukturu databáze zachycuje databázový diagram 5.2.



Obrázek 5.2. Diagram schématu databáze

5.2.4 Rozpoznání nové trasy

Pro lepší uživatelský zážitek jsou zaznamenaná data rozdělena do tras. Trasa je posloupnost po sobě jdoucích pozic, chronologicky seřazených dle času a data naměření, které odeslal jeden lokátor během jedné jízdy vozidla. Začátek jízdy vozidla je definován jako pozice, která má jiné `sessionId` než pozice, která jí bezprostředně předchází, pokud jsou seřazeny podle data měření a zároveň časový rozdíl mezi nimi je alespoň 5 minut (toto je výchozí hodnota, která může být uživatelem změněna). Konec jízdy

je definován jako poslední naměřená pozice před začátkem nové jízdy. Tato definice mechanismu je prevencí před vytvářením falešných tras při náhlém vypnutí napájení *lokátoru* (může nastat např. při nechtěném vypnutí motoru vozidla), výpadku spojení se serverem nebo dlouhém stání.

Pokud by se měřil pouze časový úsek mezi dvěma příchozími zprávami na server, mohlo by také dojít k vytvoření falešné trasy. *Lokátor* by při výpadku internetového připojení odeslal balík naměřených pozic se zpožděním. Vygenerování falešné trasy by se u zpožděného odesílání dalo vyřešit porovnáním časových razítek jednotlivých pozic (kdy byly naměřeny). Problém tohoto řešení nastane v případě, stojí-li vozidlo delší dobu na jednom místě (tudíž nejen neodesílá, ale ani neměří aktuální polohu). Časový rozdíl poslední odeslané pozice před tím, než vozidlo zastavilo, a pozice, která byla naměřena jako první při opětovném pohybu, by byl příliš velký. Takové počínání by bylo vyhodnoceno jako nová trasa. Ve skutečnosti ale mohlo vozidlo např. pouze stát v koloně.

Jelikož při rozpoznávání trasy je nutné vždy získat poslední známou pozici vozidla (a zároveň se na tuto pozici dotazuje uživatelské rozhraní), není vhodné se pokaždé dotazovat databáze. Proto jsou poslední známé pozice vozidel uloženy v mezipaměti[1], implementované třídou `CaffeineAtomicCache` s využitím knihovny `ScalaCache` [10]. Cache je implementována způsobem umožňujícím atomickou úpravu hodnoty. Data v ní tak zůstávají konzistentní s daty uloženými v databázi.

5.2.5 Zpracování pozic z lokátoru

Lokátor odesílá na server balíky pozic (zpráva `Report`). Pozice jsou serverem uloženy do databáze. Pokud navíc některý z klientů odebírá informaci o aktuální pozici vozidla, které data odeslalo, je tomuto klientu odeslána nejaktuálnější pozice. Sekvenční diagram zachycující tento proces je uveden na obrázku 5.3.

5.2.6 Diagram tříd

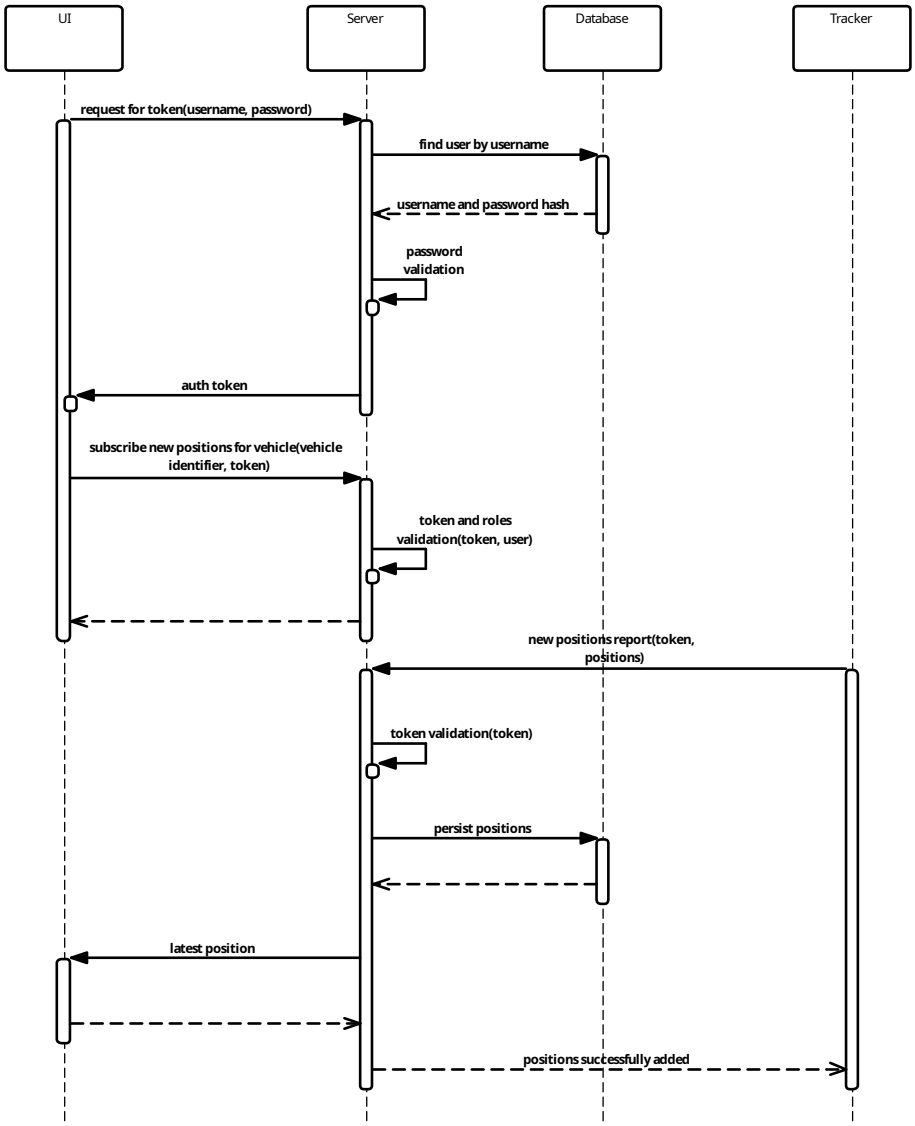
Jelikož je *backendový server* napsán v objektově-funkcionálním jazyce, není možné jeho strukturu zcela přesně zachytit jako diagram tříd, který je určen pro objektové programovací jazyky. Nicméně pro znázornění vztahů mezi objekty, resp. třídami, je dostačující. Diagram přiložený jako příloha C tedy zachycuje závislost jednotlivých tříd, ne tak už jejich vnitřní strukturu.

5.2.7 Konfigurace a nasazení

Konfigurace serveru se provádí konfiguračním souborem `reference.conf` ve formátu HOCON umístěným v cestě `./src/main/resources`. Většina parametrů je jako výchozí namapována na systémové proměnné. Tím je usnadněna konfigurace při nasazení pomocí Docker compose. Parametry, které jsou v konfiguračním souboru nastaveny hodnotou, nikoliv jako systémová proměnná, indikují uživateli, že po jejich přenastavení nemusí systém pracovat korektně. Timeouty spojení, počty pokusů pro znovupřipojení apod. jsou nastaveny tak, aby byl zvolen rozumný kompromis mezi stabilitou a rychlostí.

Parametry jsou rozděleny do logických celků, uvedených jejich jménem následovaným dvojicí složených závorek, uvnitř kterých se nacházejí jednotlivé parametry zapsané jako dvojice klíč–hodnota. Přehled systémových proměnných s vysvětlením, co znamenají, je uveden níže. Ostatní konfigurační parametry nejsou pro nasazení důležité. Z toho důvodu zde nebudou popisovány. Jejich popis je dostupný v dokumentaci kódu.

[1] anglicky cache



Obrázek 5.3. Zpracování nových pozic

Jak bylo analyzováno, *backendový server* lze nasadit jako Docker kontejner. To však není nutností a systém může běžet jako klasický proces v hostitelském systému. Tato varianta je však méně vhodná. Nezaručuje například automatický restart aplikace po pádu (resetování zařízení) apod. Pokud je to ale nutné a využití Dockeru nepřichází v úvahu, lze server zkompilovat a spustit příkazem `sbt run` (poté je vhodnější konfigurovat server přímo v konfiguračním souboru, nikoliv za pomoci systémových proměnných). Aby se server spustil, musí být na zařízení nainstalován nástroj *sbt* a Java v minimální verzi 11.

Přehled systémových proměnných

- JWT_SECRET – klíč pro podpis vystavovaných JWT tokenů symetrickou šifrou
- JWT_EXPIRATION – délka platnosti vydaného JWT tokenu v sekundách
- DB_JDBC_URL – connection string k databázi
- DB_USERNAME – uživatelské jméno pro přístup do databáze
- DB_PASSWORD – heslo pro přístup do databáze
- MQTT_HOST – adresa MQTT brokeru

- MQTT_PORT – port MQTT brokeru
- MQTT_USER – uživatelské jméno pro přihlášení k MQTT brokeru
- MQTT_PASSWORD – heslo pro přihlášení k MQTT brokeru
- MQTT_TOPIC – název kontextu, ze kterého jsou čteny pozice
- MQTT_SSL – pokud je true, komunikace s MQTT brokerem je šifrována (TLS)
- MQTT_SUBSCRIBER_NAME – jméno serveru prezentované MQTT brokeru

Deploy script

Pro zjednodušení celého procesu nasazení je k dispozici bash skript `deploy.sh`, který agreguje všechny níže popsané postupy. Ty jsou zde rozepsány pro lepší pochopení, co lze (a jak) na serveru nakonfigurovat.

Sestavení spustitelného archivu (fat JAR)

Projekt obsahuje rozšíření pro *sbt* s názvem `sbt-assembly` [27], který umožňuje vytvořit z aplikace tzv. fat JAR – archiv, který obsahuje všechny potřebné závislosti (knihovny) nutné pro běh programu. Lze jej vytvořit následujícím příkazem, který stáhne všechny závislosti, zkompiluje kód a sestaví požadovaný archiv.

```
sbt assembly
```

Výsledkem této operace je soubor, který může být spuštěný na JVM a nepotřebuje pro svůj běh žádné další závislosti. Lze ho také spustit jako systémový proces příkazem

```
java -jar <cesta_k_fat_jar>
```

Avšak hlavním důvodem, proč je tento archiv vytvářen, je jeho použití v Docker image. Důvodem, proč není z projektu rovnou vygenerován Docker image, ale je nejprve zabalen do fat JAR, je nízký výpočetní výkon počítače RaspberryPi, na který je server primárně nasazován. Kompilace kódu a vytvoření Docker image by tak trvalo neúměrně dlouho (řádově několik minut). image totiž nemůže být vytvořen na lokálním, výkonnějším, počítači a následně přenesen na RaspberryPi kvůli rozdílným architekturám procesorů. To ale neplatí o samotném kódu (fat JAR), který je kompilován do Java bajtkódu, neboť, jak již bylo několikrát zmíněno, ten může být spuštěn na libovolné architektuře procesoru, tedy na libovolném zařízení s JVM.

Nasazení jako Docker kontejner

Zde popsaný postup je nutné provádět na zařízení, na které má být *backendový server* nasazen. Opět z důvodů možných rozdílů v architekturách procesorů.

Než je možné spustit Docker kontejner, je nutné vytvořit pro něj image. Soubor `Dockerfile` popisuje postup, jak se takový image má vytvořit a co vše má obsahovat. Následujícím příkazem

```
docker build . -t cz.cvut.fit/tracker-server:1.0
```

spuštěným z adresáře, ve kterém se nachází `Dockerfile`, dojde k vytvoření Docker image s tagem `cz.cvut.fit/tracker-server`. Dále už jen chybí nastavit pro tento image běhové prostředí a spustit ho jako Docker kontejner. K tomu slouží soubor `docker-compose-with-mqtt.yml` popisující, jak má běhové prostředí vypadat. Jeho výpis ukazuje kód 5.3.

Před spuštěním serveru je nutné správně nastavit několik proměnných. Jde především o cesty k souborům uloženým na hostitelském filesystému. U serveru je nutné nastavit cestu ke složce s databází – `<abs_cesta_k_db_slozce>:/DB`. Dále cestu ke složce s *uživatelským rozhraním* – `<abs_cesta_k_ui_slozce>:/front`.

U MQTT brokeru je pak nutné nastavit cesty k certifikátům a klíčům, konfiguraci a souboru pro logování. Struktura zůstává stejná. Před dvojtečkou je uvedena cesta k požadovanému souboru na hostitelském filesystému, za ní pak cesta v Docker kontejneru (s touto cestou počítá server). Pokud není nutné společně se serverem nasazovat MQTT broker, použije se `docker-compose.yml` (jeho výpis ukazuje kód 5.4), kde je pak nutné kromě zmiňovaného změnit také `MQTT_HOST`, případně `MQTT_PORT`.

Jsou-li všechny systémové proměnné správně nastaveny, lze vše spustit jedním příkazem

```
docker-compose up -d
```

Nástroj Docker compose vytvoří kontejnery a systém je poté připraven k použití (dostupný přes `http://localhost:8085`).

Kód 5.4.

```
# docker-compose.yml
version: "3"

services:
  server:
    image: cz.cvut.fit/tracker-server:1.0
    ports:
      - 8085:8080
    environment:
      - JWT_SECRET=mH@6@*6dp*p^8Xhegecv
      - JWT_EXPIRATION=900
      - DB_JDBC_URL=jdbc:h2:<CHANGE ME>;
                    INIT=RUNSCRIPT FROM './create.sql';
                    MODE=MySQL
      - DB_USERNAME=
      - DB_PASSWORD=
      - MQTT_HOST=<CHANGE ME>
      - MQTT_PORT=8883
      - MQTT_TOPIC=tracker/positions
      - MQTT_SSL=true
      - MQTT_SUBSCRIBER_NAME=tracker-rpi
    volumes:
      - <CHANGE ME>:/DB
      - <CHANGE ME>:/front
    restart: always
```

Nasazení jako Docker kontejner bez využití fat JAR

Pokud má být server nasazen na zařízení se stejnou architekturou jako zařízení, na kterém se provádí kompilace, lze celý postup zjednodušit. Pro tyto účely je totiž do projektu přidáno další rozšíření pro *sbt* s názvem `sbt-native-packager` [49]. To umožňuje přímo z *sbt* vygenerovat Docker image příkazem

```
sbt docker:publishLocal
```

Kód 5.3.

```
# docker-compose-with-mqtt.yml
version: "3"

services:
  server:
    image: cz.cvut.fit/tracker-server:1.0
    depends_on:
      - mqtt
    links:
      - "mqtt:broker"
    ports:
      - 8085:8080
    environment:
      - JWT_SECRET=mH@6@*6dp*p^8Xhegecv
      - JWT_EXPIRATION=900
      - DB_JDBC_URL=jdbc:h2:<CHANGE ME>;
                    INIT=RUNSCRIPT FROM './create.sql';
                    MODE=MySQL
      - DB_USERNAME=
      - DB_PASSWORD=
      - MQTT_HOST=broker
      - MQTT_PORT=8883
      - MQTT_TOPIC=tracker/positions
      - MQTT_SSL=true
      - MQTT_SUBSCRIBER_NAME=tracker-rpi
    volumes:
      - <CHANGE ME>:/DB
      - <CHANGE ME>:/front
    restart: always
  mqtt:
    image: eclipse-mosquitto:2-openssl
    ports:
      - 8883:8883
    volumes:
      - <CHANGE ME>:/mosquitto/certs/chain.pem
      - <CHANGE ME>:/mosquitto/certs/cert.pem
      - <CHANGE ME>:/mosquitto/certs/privkey.pem
      - <CHANGE ME>:/mosquitto/config/mosquitto.conf
      - <CHANGE ME>:/mosquitto/data
      - <CHANGE ME>:/mosquitto/log
    restart: always
```

Vygenerovaný image bude mít tag `cz.cvut.fit/tracker-server`. Spuštění kontejneru s tímto image může být taktéž provedeno pomocí Docker compose jak samostatně, tak společně s MQTT brokerem.

5.3 Uživatelské rozhraní

V implementaci uživatelského rozhraní je použit JavaScriptový framework Vue.js. Je tomu především z důvodu dobré kompatibility frameworku mezi různými typy zařízení (vizte analýzu, sekce 4.5.2). Webová aplikace *uživatelského rozhraní* je implementována jako tzv. jednostránková. Klientské zařízení si stáhne celý JavaScriptový zdrojový kód stránky při prvním přístupu a veškeré další operace jsou již prováděny na lokálním zařízení. Prvotní inicializace (stažení souborů aplikace) tak může být pomalejší. Další interakce se stránkou je poté zpravidla rychlejší. Navigaci si provádí aplikace samotná a není tak nutné odesílat požadavky na server a čekat na jeho odpověď. Využívá k tomu knihovnu `vue-route` [60], která je součástí Vue.js framework.

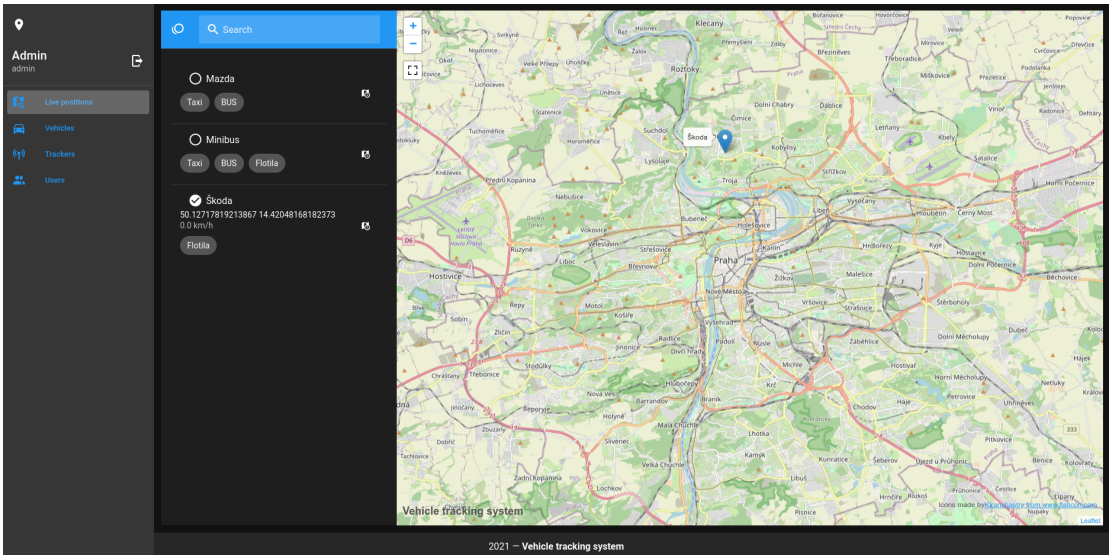
5.3.1 Komunikace s backendovým serverem

K získání dat pro zobrazení je nutné interagovat s aplikačním rozhraním *backendového serveru*. Pro tento účel je v aplikaci využita knihovna `axios` [5]. Přes tuto knihovnu jsou posílány dotazy na server a očekávané odpovědi jsou zpracovávány asynchronně. Získávání dat od *backendového serveru* tedy není blokující.

Některé takto získané informace musejí být perzistentní. Z toho důvodu je součástí webové aplikace knihovna `Vuex`. Ta ukládá stav aplikace na zařízení klienta. Takto uložených informací je minium a týkají se především přihlášeného uživatele (login session) a nastavení vzhledu, resp. zda mají být zobrazovány nápovědy. Aplikace má tyto informace k dispozici i v případě, že byla zavřena a opětovně otevřena na stejném zařízení.

5.3.2 Zobrazení aktuální/historie polohy vozidel

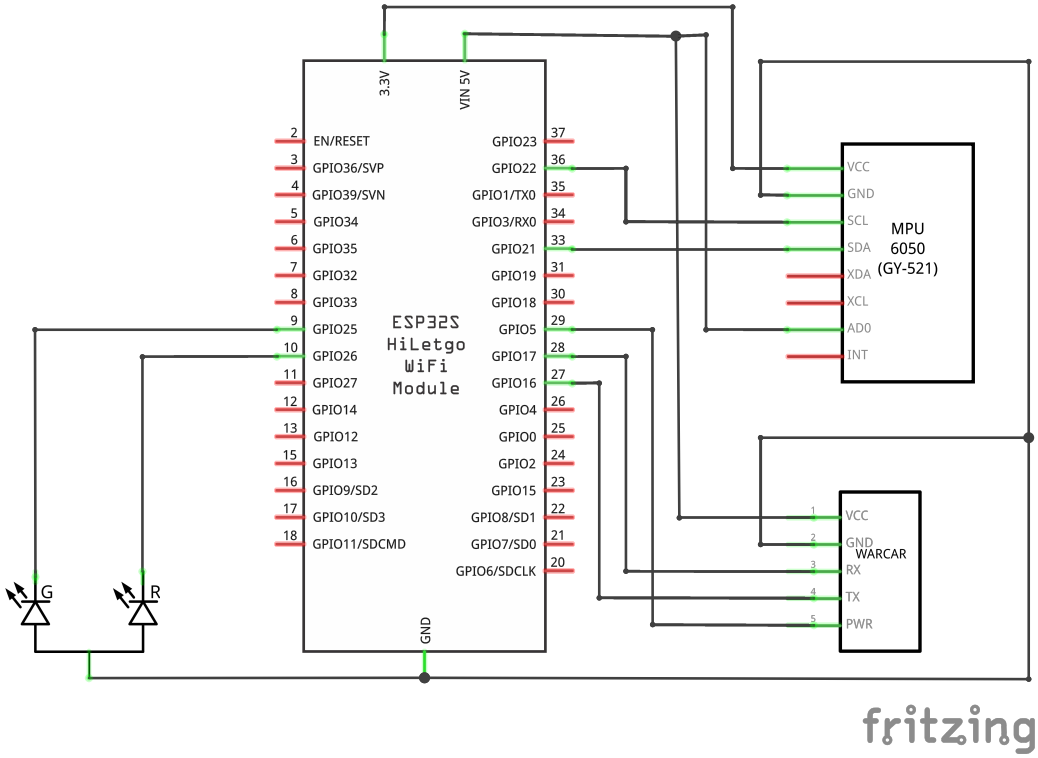
Aktuální pozice jsou přímo zobrazovány do mapového opensource podkladu OpenStreetMap přes JavaScriptovou knihovnu `Leaflet` [1]. Uživatel tak ihned vidí, kde se vozidla nacházejí přímo na mapě. Historie polohy vozidla je zobrazována jako posloupnost naměřených pozic spojených úsečkou. Tím vynikne trasa vozidla zobrazená jako čára na mapovém podkladu. Navíc lze v místech, ve kterých byla naměřená pozice, zobrazit její GPS polohu, čas měření a okamžitou rychlost vozidla v tento čas.



Obrázek 5.4. Screenshot *uživatelského rozhraní*

5.4 Lokátor

Zařízení je sestaveno z řídícího mikrokontroléru ESP32 a modulu WARCAR (obsahujícího čip SIM808) určeného pro sledování polohy a připojení k internetu (GPRS). Komunikace mezi modulem WARCAR a mikrokotrolérem probíhá přes sériovou linku. Dále je součástí *lokátoru* také akcelerometr, který slouží pro rozpoznání, zda vozidlo stojí, nebo je v pohybu. Této informace je využito při filtrování nepřesně změřených poloh. Informace získané z akcelerometru zároveň umožňují snížit datový tok, protože pozice nejsou v případě, že vozidlo stojí, odesílány. Schéma zapojení *lokátoru* je znázorněno na obrázku 5.5.



Obrázek 5.5. Schéma zapojení lokátoru

Firmware *lokátoru* je napsán v jazyce C++. Využívá se v něm asynchronních úloh a veškeré funkcionality nejsou blokující. Obsluhu asynchronních úloh zajišťuje knihovna `Tasker` [34]. Díky mikrokontroléru ESP32, na kterém je postaven, může firmware *lokátoru* využívat pro svůj běh dvě procesorová jádra. Na jednom jádře je spuštěno uživatelské rozhraní a WiFi management, na druhém poté zbytek kódu. Ten se stará o načítání dat o aktuální poloze a jejich odesílání na server přes protokol MQTT. Komunikace s MQTT je zajištěna pomocní knihovny `PubSubClient` [42]. Knihovna umožňuje odesílat zprávy pouze s QoS na úrovni 0 – bez potvrzení a zaručení doručení. Tento nedostatek je kompenzován nastavením KeepAlive intervalu na 15 sekund. Ztráta spojení je tak detekována nejpozději po 15 sekundách.

Asynchronní přístup s sebou přináší problém synchronizace vláken. Ta je nutná, pokud více vláken přistupuje k jednomu zdroji. Zde to je především sériová linka propojující ESP32 a WARCAR modul. Každé čtení a zápis z ní musejí být synchronizovány. K tomu je použito synchronizačního primitiva implementačního jazyka C++ mutex.

Pokud některá část kódu bude chtít přistupovat k sériové lince, musí počkat na zámek (mutex). Poté má k ní zaručen výhradní přístup. Po ukončení práce se sériovou linkou musí být zámek znovu odemčen.

Ruční uzamykání a odemykání zámku může vést velice snadno k deadlocku, např. při předčasném návratu způsobeném výjimkou. Proto je vhodnější použít mechanismus RAII, konkrétně například wrapper `lock_guard`, který počká na uvolnění zámku, poté ho zamkne a pokud vykonávaný kód opustí lexikální scope, ve kterém byl zamčen, automaticky jej odemkne. Ukázku užití tohoto mechanismu ukazuje následují část kódu implementace:

```
std::lock_guard<std::recursive_mutex> lg(gsm_mutex);
return modem.isNetworkConnected() && modem.isGprsConnected();
```

V ukázce není využit standardní `mutex`, nýbrž `recursive_mutex`. V implementaci je využit z důvodu, chtějí-li se volat funkce jím chráněné navzájem, zároveň má ale vlákno exkluzivní přístup k sériové lince a je tedy zachován základní princip mutexu.

Aby bylo spojení zabezpečené a nemohlo dojít k odposlechnutí přenášených dat, je komunikace zabezpečena pomocí TLS. Knihovna `TinyGSM` [47] použitá pro interakci s modulem WARCAR sice umožňuje navazovat zabezpečená spojení sama o sobě, nicméně se zmiňovaným modulem nefunguje. Z toho důvodu je šifrování zajištěno knihovnou `SSLClient` [23].

Pro zjednodušení implementace a práce se zdrojovými kódy *lokátoru* je použit nástroj PlatformIO [45]. S jeho pomocí lze celý projekt sestavit příkazem `pio run`. Závislosti na knihovnách jsou popsány přímo v konfiguračním souboru nástroje PlatformIO – `platformio.ini`. Nástroj se postará také o upload zkompilovaného kódu do mikrokontroléru. Společně s binárními soubory zkompilovaného kódu lze nahrát také obraz souborového systému SPIFFS, který je využit pro uložení statických souborů uživatelského rozhraní lokátoru.

Natavení *lokátoru* lze měnit přes uživatelské rozhraní a je ukládáno do EEPROM paměti mikrokontroléru pomocí knihovny `Preferences`, která je součástí Arduino frameworku pro ESP32.

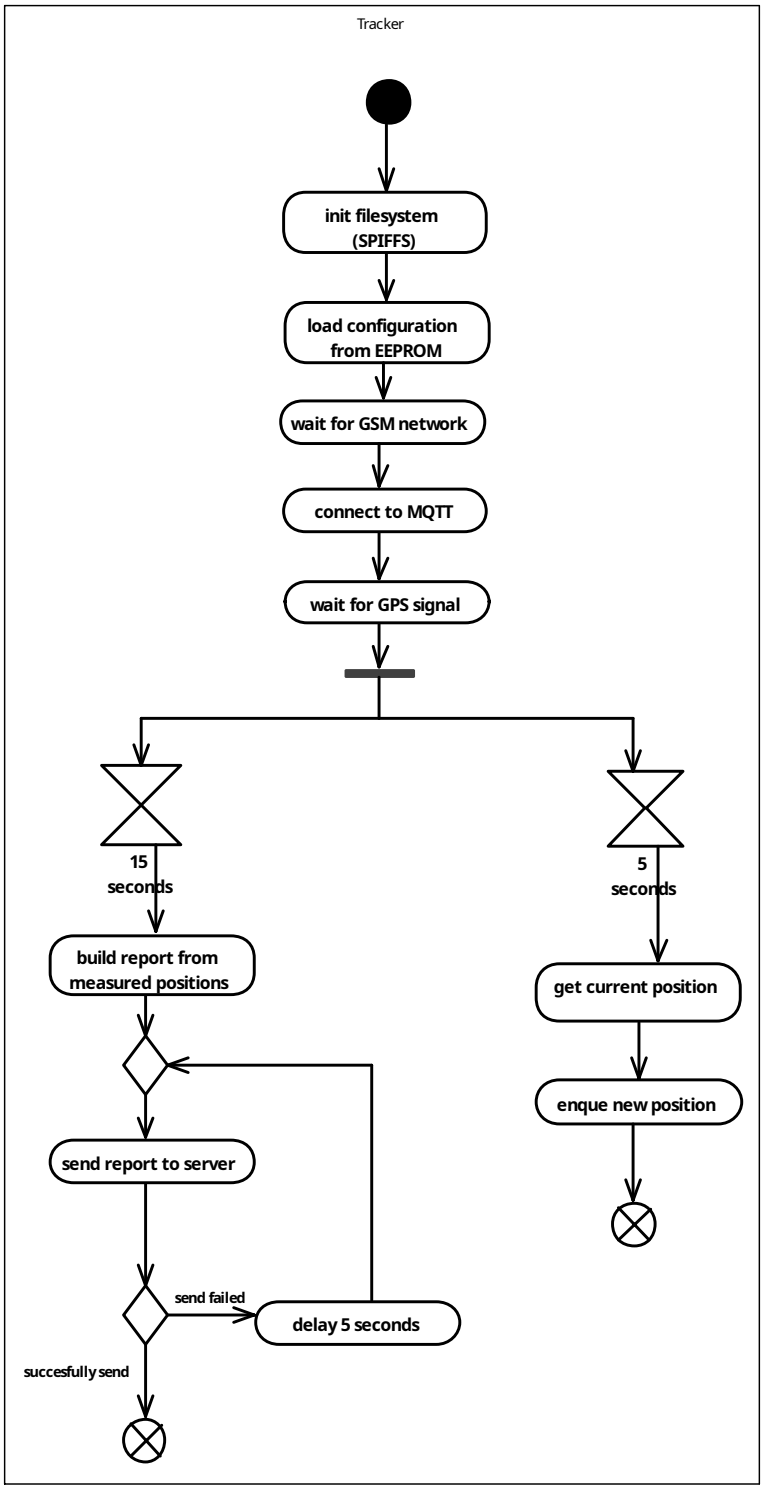
5.4.1 Zpracování naměřených pozic

Zařízení *lokátoru* měří svou aktuální pozici v předem nastavených intervalech. Po předem definovaném intervalu (který je násobkem intervalů měření aktuální pozice) jsou všechny naměřené pozice zabaleny do jedné zprávy (`Report`) a odeslány na *backendový server*. Toto odeslání se nemusí podařit. Pokud tuto skutečnost lokátor indikuje, zařadí neodeslanou zprávu (`Report`) do fronty pro znovuodeslání. Zpráva je odesílána do té doby, dokud není doručena na *backendový server*. Všechny naměřené pozici i zprávy `Report`, které se nepodařilo odeslat jsou uloženy v paměti RAM. Pokud tedy dojde k vypnutí napájení, všechny tyto pozice budou smazány.

Implementovaný mechanismus znovuodesílání (popsaný diagramem aktivit na obrázku 5.6) nemusí vždy odeslat pozice v pořadí, ve kterém byly naměřeny. Každá pozice má ale u sebe časové razítko, kdy byla pořízena. Na *backendovém erveru* není poté problém tyto pozice zpracovávat ve správném pořadí.

5.4.2 Filtrace nepřesných pozic

Získání dat o aktuální pozici zajišťuje již zmíněný modul WARCAR. Jelikož přesnost GPS přijímače není stoprocentní, měří nepřesnou lokaci ve stavu, kdy setrvává ve stejné geografické poloze. Tyto polohy se poté uživateli v mapě jeví jako shluk náhodných



Obrázek 5.6. Diagram aktivity měření a odesílání aktuální pozice

pohybů okolo skutečné geografické polohy *lokátoru*. Jako první přicházelo při implementaci v úvahu filtrovat pozice, během kterých měl *lokátor* nízkou okamžitou rychlost (do 4 km/h). Takové řešení má však mnoho nevýhod. Například pokud by bylo nutné sledovat objekty (např. osoby), které se pohybují pomaleji než takto nastavená hranice, zařízení by nefungovalo korektně. Všechny tyto pozice při nízkých rychlostech by byly filtrovány. Jelikož je systém zaměřený převážně na sledování vozidel, nevypadá tento

nedostatek jako závažný. Nicméně opak je pravdou. Pokud se bude vozidlo pohybovat např. v koloně, jeho rychlost bude velice nízká. Tento pohyb by takto implementovaný *lokátor* také vyfiltroval a nezaznamenal. Proto bylo od tohoto řešení ustoupeno.

Vhodnějším odstraněním problému nepřesných pozic při nízkých rychlostech je přidat do zařízení akcelerometr. Díky němu lze zjistit, zda se objekt skutečně pohybuje (má určitou akceleraci – zrychlení), nebo stojí na místě. Tento způsob filtrování byl použit i v implementaci *lokátoru* popisovaného v této práci. Pokud naměřená okamžitá rychlost je nižší než předem definovaná mez (empiricky určena na 4 km/h, nicméně lze ji v konfiguraci změnit), ověří se, zda je *lokátor* skutečně v pohybu (má zrychlení). Pokud ano, je pozice zpracována běžným způsobem, jinak je vyfiltrována (ignorována) a neprobíhá její další zpracování.

Přijímač nevykazuje chyby pouze při nízkých rychlostech. V rychlostech vyšších je však jejich pravděpodobnost menší. Aby se ale i zde filtrovaly zjevně nepřesné pozice, je mezi posledním naměřeným a nově naměřeným vzorkem vypočítána vzdálenost. Při výpočtu se zanedbává zakřivení Země a výpočet se provádí pouze v rovině. Jelikož vzdálenosti nepřesáhnou desítky (maximálně jednotky stovek metrů), bude nepřesnost minimální. V implementaci je určena rozhodující vzdálenost (v metrech) vztahem

$$\frac{250}{3.6} \times 4 \times 1.25 \approx 348m$$

Hodnota vychází z faktu, že zákon č. 361/2000 Sb., o silničním provozu v §18, v odstavci 3 říká: *„Řidič motorového vozidla o maximální přípustné hmotnosti nepřevyšující 3 500 kg a autobusu smí jet mimo obec rychlostí nejvýše 90 km.h^{-1}; na silnici pro motorová vozidla rychlostí nejvýše 110 km.h^{-1} a na dálnici rychlostí nejvýše 130 km.h^{-1}. Řidič jiného motorového vozidla smí jet rychlostí nejvýše 80 km.h^{-1} “* [62]. Dále pak z rychlosti vzorkování aktuální polohy, která je ve výchozím stavu nastavena na 4 sekundy. Ačkoliv zákon definuje maximální povolenou rychlost na dálnici 130 km/h, byla rozhodující hodnota vypočítána z rychlosti 250 km/h. Tím se zaručí dostatečná tolerance pro případnou jízdu v jiných zemích než České republice, kde toto omezení nemusí být platné. Konečný výsledek je ještě zvýšen o jednu čtvrtinu pro případ, že by měření nového vzorku bylo zpožděno.

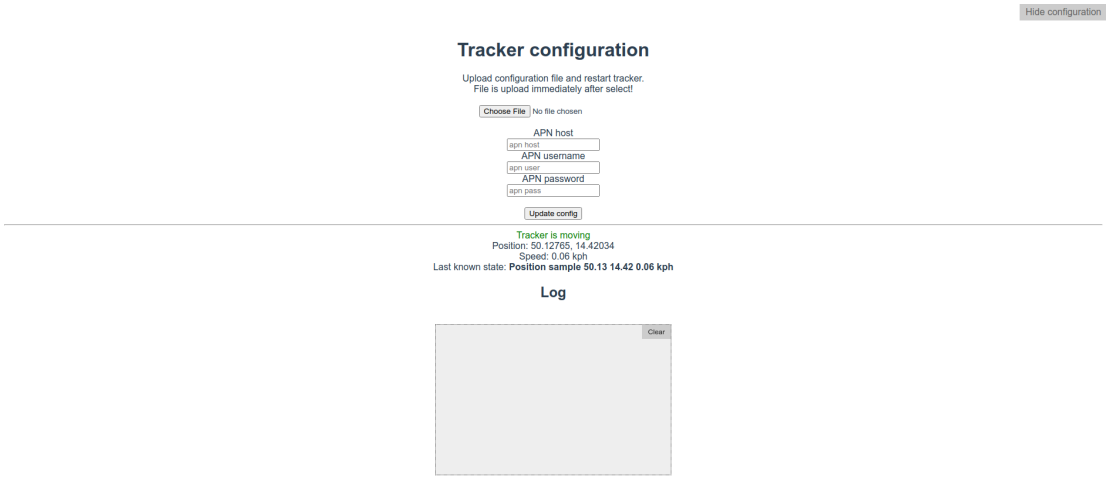
5.4.3 Uživatelské rozhraní lokátoru

Uživatelské rozhraní *lokátoru* je napsáno ve frameworku Vue.js, stejně jako *uživatelské rozhraní* celého systému. Uživatel přes něj může měnit nastavení a zároveň nahrát konfigurační soubor vygenerovaný *backendovým serverem*. Konfigurace v něm uložená zajistí spárování zařízení s konkrétním vozidlem v systému. Tento soubor spáruje zařízení se záznamem v aplikaci. Data jsou mezi mikrokontrolérem a webovým rozhraním přenášena protokolem WebSocket.

Lokátor odesílá informace o svém aktuálním stavu do svého webového rozhraní. V něm lze poté záznam těchto činností procházet. Součástí toho záznamu jsou také aktuální souřadnice a aktuální rychlost. Ukázku webového rozhraní *lokátoru* zobrazuje obrázek 5.7.

5.4.4 Konfigurace

Jak nastínila předchozí sekce, *lokátor* se konfiguruje přes uživatelské rozhraní. Rozhraní je přístupné po připojení na WiFi s SSID `tracker?` (otazník nahrazen MAC adresou ESP32), kterou *lokátor vysílá*, a po zadání adresy `tracker.local` do adresního řádku webového prohlížeče. WiFi je chráněna heslem, které je chráněna heslem, které lze nastavit pouze při kompilaci kódu. Aby bylo zvýšeno zabezpečení *lokátoru*, AP se po dvou



Obrázek 5.7. Screenshot uživatelského rozhraní lokátoru

minutách běhu automaticky vypne. Zároveň je ale nutné během této doby provést veškerou potřebnou konfiguraci a změny v nastavení. V případě potřeby další konfigurace je nutné *lokátor* restartovat. WiFi síť bude opět dostupná po dobu dvou minut.

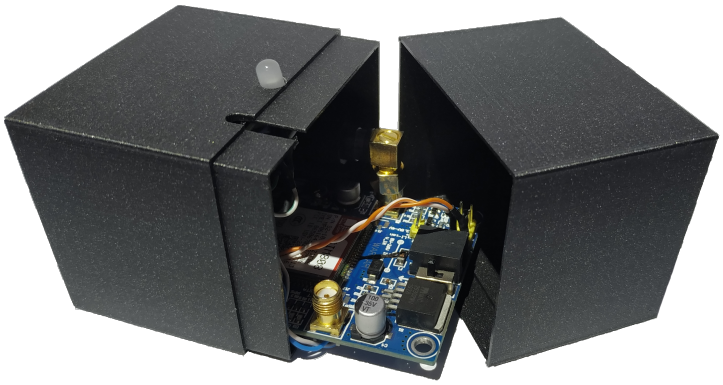
Nastavení tedy není nutné provádět při kompilaci a na zařízení není nutné nahrávat nový kód při každé její změně. Je tedy uživatelsky přívětivější. Konfiguraci také není zapotřebí vyplňovat ručně. V *uživatelském rozhraní* je možné pro každý lokátor stáhnout konfigurační soubor, vygenerovaný *backendovým serverem*, ve formátu JSON. Ten obsahuje informace nutné pro identifikaci odeslaného balíku poloh na *backendový server*. Současně pak konfiguraci připojení k MQTT serveru. Ta je identická s konfigurací připojení *backendového serveru*, včetně případného uživatelského jména a hesla. Tato vlastnost jednak zjednoduší konfiguraci a oprostí uživatele od nutnosti zadávat přihlašovací údaje. Ale také zlepší dostupnost celé aplikace. MQTT broker načítá konfiguraci a společně s ní i databázi uživatelů při svém startu. Pokud by měly být přihlašovací údaje unikátní pro každý *lokátor*, musel by se MQTT broker při každém vytvoření lokátoru restartovat. Během této doby by aplikace nepřijímala žádná data.

Lokátor zaznamenává ve výchozí konfiguraci aktuální pozici každé 4 sekundy. Každých 20 sekund (ideálně tedy 5 pozic) je odešle ve zprávě `Report` na *backendový server*. Zpoždění v zobrazení aktuální pozice je tedy 20 sekund. I toto nastavení však lze změnit ve webovém rozhraní.

Konfigurace APN je ponechána plně na uživateli. Není součástí konfiguračního souboru. Je tak především proto, aby bylo možné vyměnit poskytovatele internetového připojení bez nutnosti nahrávat nový konfigurační soubor, kterému by předcházelo několik změn přes *uživatelské rozhraní* a celý proces by byl příliš zdlouhavý.

5.4.5 Výroba

Při vývoji *lokátoru* bylo pro zapojení všech částí dohromady použito nepájivé kontaktní pole. Aby bylo možné finální verzi zařízení umístit do vozidla a nehrozilo jeho poškození, byla pro lokátor navržena krabička, jejíž model je uložen na přiloženém paměťovém médiu. Obrázek 5.8 ukazuje výslednou podobu zařízení vloženého právě v této krabičce, která byla vytištěna na 3D tiskárně.



Obrázek 5.8. Výsledné sledovací zařízení umístěné ve vytisknuté krabičce

5.4.6 Nasazení

Po správném zapojení lokátoru (dle schématu na obrázku 5.5) je nutné, aby do paměti ESP32 byl kromě binárních souborů nahrán také obraz filesystému SPIFFS se zkompilovaným uživatelským rozhraním. Pro všechny tyto operace je možné využít nástroj PlatformIO.

Filesystém SPIFFS umožňuje použít maximálně 32znakovou délku cesty k souboru. Proto je nutné, aby zkompilované soubory webového rozhraní měly dostatečně krátký název. Toto nastavení lze provést v konfiguraci sestavení v souboru `vue.config.js`. Aby bylo dosaženo rychlejšího načítání a menšího vytížení mikrokontroléru ESP32, jsou výsledné JavaScriptové soubory ještě zkomprimovány GNU zip (gzip) algoritmem.

Pro správné načítání webového rozhraní musí být zachována následující adresářová struktura. Veškeré soubory uživatelského rozhraní musejí být dostupná na cestě `/w`. Následující výpis popisuje adresářovou strukturu, kde mají být soubory webové aplikace uloženy v PlatformIO projektu:

```
<project_root>/
|- data/
|  |- w/
|  |  |- <ui_files>
```

Pro sestavení webového rozhraní ze zdrojových kódů, zkompilování firmwaru a nahrání binárních souborů do paměti ESP32 je připraven skript s názvem `build-and-upload.sh`. Ten s využitím již zmíněného nástroje PlatformIO připraví mikrokontrolér ESP32 pro použití jako *lokátor*. Zkrácený výpis operací, resp. příkazů, které jsou skriptem provedeny, ukazuje výpis kódu 5.5.

5.5 Pořizovací náklady

Jelikož systém byl navržena a realizován tak, aby jeho pořizovací a provozní náklady byly co nejnižší, jsou v této sekci popsány ceny všech součástí systému. Tabulka 5.1 popisuje ceny jednotlivých součástí (všechny ceny jsou platné k 10. 4. 2021). Reprezentuje tedy pořizovací náklady za *backendový server* a jeden *lokátor*. V celkové ceně

Kód 5.5.

```
yarn build && \
cd dist && \
gzip -9 css/* && \
gzip -9 js/* && \
cd .. && \
cp -r dist/* <project_root>/data/w && \
cd <project_root> && \
pio run --target uploadfs && \
pio run --target upload
```

nejsou zahrnuty náklady na 3D tisk, neboť ten je implementačním detailem při výrobě *lokátoru* a lze ho nahradit.

Provozní náklady závisí především na volbě poskytovatele mobilního internetového připojení. Z toho důvodu zde není uváděna a je čistě v režii uživatele. Cena elektrické energie spotřebované *backendovým serverem* bude oproti ceně za připojení zanedbatelná.

název produktu	cena v Kč	obchod
RaspberryPi 4	1 949	Alza.cz a.s.
DOIT ESP 32 DevKit	105	Aliexpress.com
WARCAR	360	Aliexpress.com
akcelerometr GY-521	30	Aliexpress.com
celkem za server	1 949	
celkem za lokátor	495	
celkem	2 444	

Tabulka 5.1. Seznam cen součástí použitých v implementaci

5.6 Možná vylepšení

Při implementaci *lokátoru* není využito uspávání jednotlivých komponent. Kvůli tomu není zařízení vhodné pro bateriový provoz. Pokud by měl být *lokátor* o tuto funkcionalitu rozšířen, bylo by nutné jednotlivé moduly v době nečinnosti uspávat, a tím snížit energetickou náročnost zařízení.

Baterii by bylo možné použít také pro odeslání neodeslaných pozic v případě, že by došlo k náhlému výpadku napájení. Jak bylo popsáno v sekci 5.4.1, pozice a neodeslané zprávy jsou uloženy v paměti, která se po odpojení napájení smaže. Baterie by tedy napájela zařízení pouze po dobu, než by se odeslal všechna data na *backendový server*, poté by se *lokátor* vypnul.

Dalším vylepšením by mohla být možnost nastavení lokátoru pro sledování pomaleji se pohybujících objektů, např. osob. Pro takovou funkcionalitu by bylo nutné změnit nastavení filtrování naměřených pozic, jelikož při pomalejším pohybu bude docházet k většímu množství chybně naměřených pozic. K vyřešení toho problému by bylo možné použít data z akcelerometru a naměřené GPS pozice. Na základě těchto hodnot poté odstraňovat nepřesné pozice pomocí Kalmanova filtru [8].

Kapitola 6
Experimentální měření vlastností lokátoru

V této kapitole je popsáno měření přenesených dat a spotřeby energie finální implementace *lokátoru*, resp. finální verze posílané zprávy `Report` (ukázka kódu 5.1). Měření datového toku nebylo prováděno přímo na zařízení, ale odesílání bylo simulováno na počítači. Jelikož kódování Protobuf je deterministické, je jedno, na jakém zařízení kódování proběhne. Stejně tak odesílání dat na MQTT broker.

Každá zpráva `Report` obsahovala stejný autentizační `token`, stejné `sessionId` a `vehicleId`. Všechny zprávy byly šifrovány protokolem TLS a odesílány do stejného MQTT kontextu (`sledovac/experiment`) na stejný MQTT broker. Datový tok byl odchycen nástrojem Wireshark [51]. Aby bylo možné lépe nahlédnou do takto odchycené šifrované komunikace, testovací skripty ukládaly použité šifrovací klíče.

Naměřená data jsou rozdělena do tří částí. První z nich je inicializace, která reprezentuje množství dat, přenesených při inicializaci všech užitých protokolů (TCP, TLS a MQTT). Třetí část reprezentuje přenesené množství dat všech protokolů při ukončení spojení. Druhá část reprezentuje veškerou ostatní komunikaci. Užitečnými daty jsou myšleny velikosti serializovaných zpráv. Z těchto nasbíraných dat je vypočítána efektivita přenosu definována následujícím vztahem:

$$\frac{uzitecna\ data}{celkem\ preneseno} \times 100$$

Efektivita přenosu tedy vyjadřuje, kolik procent komunikace neslo užitečnou informaci. Její doplněk poté vyjadřuje procentuální množství režijních dat v komunikaci.

Spotřeba elektrické energie byla měřena za běžného provozu lokátoru, kdy odesílal informace o své poloze na *backendový server*.

6.1 Přenos jedné zprávy přes jedno TCP spojení

V tabulce 6.1 je popsáno množství přenesených dat během jednoho TCP spojení, tzn. pro každé odeslání jedné zprávy `Report` se vytvořilo nové spojení. Jednotlivé zprávy se lišily pouze počtem pozic, které obsahovaly. Záznam dešifrované komunikace jedné odeslané zprávy s deseti pozicemi je na obrázku 6.1.

počet pozic	1	5	10	15	20
inicializace [B]	4 821	4 821	4 821	4 821	4 821
payload [B]	366	506	681	856	1 031
ukončení spojení [B]	415	415	415	415	415
celkem přeneseno [B]	5 602	5 742	5 917	6 092	6 267
z toho užitečná data [B]	245	385	560	735	910
efektivita přenosu [%]	4,31	6,70	9,46	12,07	14,52

Tabulka 6.1. Přenesená data MQTT

Dle výsledků měření popsaných v tabulce 6.1 lze dojít k následujícím závěrům. S rostoucím počtem zpráv roste efektivita přenosu. Jestliže bude *lokátor* sbírat pozice a odesílat je po pěti na *backendový server* (odeslání se provede každých 20 sekund), odešle za 24 hodin (86 400 sekund) 4 320 zpráv o velikosti 5 742 B. To platí v případě, že by pro každou zprávu bylo nutné vytvářet nové TCP spojení. Celkem by se tedy přeneslo 24 805 440 bajtů, což je přibližně 24 MB.

Počet odeslaných pozic v jedné zprávě se však musí volit obezřetně. Čím větší odeslaný balík bude, tím větší bude zpoždění zobrazení aktuální polohy v *uživatelském rozhraní*.

Protocol	Length	Info
TCP	76	45933 → 8883 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=346653547 TSecr=0 WS=128
TCP	76	8883 → 45933 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1420 SACK_PERM=1 TSval=2389950932 TSecr=346653547 WS=128
TCP	68	45933 → 8883 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=346653574 TSecr=2389950932
TLSv1.2	291	Client Hello
TCP	68	8883 → 45933 [ACK] Seq=1 Ack=224 Win=65024 Len=0 TSval=2389950962 TSecr=346653574
TLSv1.2	1476	Server Hello
TCP	68	45933 → 8883 [ACK] Seq=224 Ack=1409 Win=64128 Len=0 TSval=346653630 TSecr=2389950979
TLSv1.2	1476	Certificate [TCP segment of a reassembled PDU]
TCP	68	45933 → 8883 [ACK] Seq=224 Ack=2817 Win=64128 Len=0 TSval=346653631 TSecr=2389950979
TLSv1.2	106	Server Key Exchange, Server Hello Done
TCP	68	45933 → 8883 [ACK] Seq=224 Ack=2855 Win=64128 Len=0 TSval=346653631 TSecr=2389950979
TLSv1.2	161	Client Key Exchange, Change Cipher Spec, Finished
TCP	68	8883 → 45933 [ACK] Seq=2855 Ack=317 Win=65024 Len=0 TSval=2389951023 TSecr=346653633
TLSv1.2	310	New Session Ticket, Change Cipher Spec, Finished
TCP	68	45933 → 8883 [ACK] Seq=317 Ack=3097 Win=64128 Len=0 TSval=346653668 TSecr=2389951025
MQTT	136	Connect Command
TCP	68	8883 → 45933 [ACK] Seq=3097 Ack=385 Win=65024 Len=0 TSval=2389951055 TSecr=346653669
MQTT	101	Connect Ack
TCP	68	45933 → 8883 [ACK] Seq=385 Ack=3130 Win=64128 Len=0 TSval=346653701 TSecr=2389951056
MQTT	681	Publish Message [sledovac/experiment]
TCP	68	8883 → 45933 [ACK] Seq=3130 Ack=998 Win=64512 Len=0 TSval=2389953115 TSecr=346655718
TCP	68	45933 → 8883 [FIN, ACK] Seq=998 Ack=3130 Win=64128 Len=0 TSval=346656948 TSecr=2389953115
TLSv1.2	99	Alert (Level: Warning, Description: Close Notify)
TCP	56	45933 → 8883 [RST] Seq=999 Win=0 Len=0
TCP	68	8883 → 45933 [FIN, ACK] Seq=3161 Ack=999 Win=64512 Len=0 TSval=2389954347 TSecr=346656948
TCP	56	45933 → 8883 [RST] Seq=999 Win=0 Len=0

Obrázek 6.1. Výpis dešifrované komunikace mezi klientem a MQTT brokerem

6.2 Přenos více zpráv přes jedno TCP spojení

Tato sekce prezentuje výsledky měření, kdy během jednoho TCP spojení bylo odesláno několik zpráv obsahujících 5 pozic. Tento počet je nastaven jako výchozí maximální počet pozic jedné zprávy odeslané *lokátorem* (vizte 5.4.4). Velikost jedné zakódované zprávy `Report` byla 385 B. Přes jedno spojení bylo odesláno 10, 100, 1 000 a 4 000 zpráv. Více zpráv nebylo měřeno, neboť *lokátor*, který by byl zapnut 24 hodin, by odeslal maximálně 4 320 zpráv `Report`. Mezi odesíláním jednotlivých zpráv bylo vloženo zpoždění 100 milisekund, aby byl protokol TCP donucen potvrdit každou zprávu zvlášť. Nakonec byl porovnán celkový počet přenesených dat a počet užitečných dat (součet velikostí přenesených zpráv).

Naměřené hodnoty v tabulce 6.2 nezahrnují MQTT Ping Pong zprávy pro udržení spojení (KeepAlive). Měření totiž probíhalo rychleji, než ve skutečnosti bude *lokátor* data odesílat. K naměřenému množství přenesených dat by tedy bylo nutné připočíst 5 760 (pokud by *lokátor* byl v provozu 24 hodin) 334B Ping Pong zpráv. Poté by množství přenesených dat odpovídalo skutečnému datovému toku.

počet zpráv	10	100	1 000	4 000
inicializace [B]	4 821	4 821	4 821	4 821
payload [B]	5 749	57 332	579 168	2 294 164
ukončení spojení [B]	415	415	415	415
celkem přeneseno [B]	10 976	62 568	580 034	2 299 400
z toho užitečná data [B]	3 850	38 500	385 000	1 540 000
efektivita přenosu [%]	35,30	61,53	66,47	66,97

Tabulka 6.2. Množství přenesených dat protokolem MQTT

6.3 Doba trvání TCP spojení

Pro určení, jak často se bude navazovat nové TCP spojení, bylo nutné změřit jeho průměrnou délka. Měření probíhalo v klidovém stavu, kdy byl *lokátor* po celou dobu měření na stejném místě. Lokátor byl zapnut v běžném režimu, tedy byl připojen k MQTT brokeru a měřil aktuální pozici. Jelikož nebyl v pohybu, neodesílal naměřené pozice na server. Proto musel každých patnáct sekund odeslat Ping zprávu, aby bylo spojení udrženo. Tím se zároveň zjistilo, zda je TCP spojení aktivní, či nikoliv. Naměřené hodnoty od momentu připojení lokátoru k MQTT brokeru do momentu ukončení toho TCP spojení popisuje tabulka 6.3.

začátek spojení	konec spojení	délka spojení
14:38:04	16:59:50	2h 21m 46s
22:02:12	11:41:16	10h 20m 56s
11:33:52	19:50:56	8h 17m 4s
00:21:01	10:20:50	9h 59m 49s

Tabulka 6.3. Množství přenesených dat protokolem MQTT

6.4 Závěr měření přenesených dat

Měření spotřeby dat je shrnuto v tabulce 6.4. Varianta 1 v tabulce popisuje případ, kdy pro přenos během 24 hodin (4 320 zpráv) bylo použito pouze jedno TCP spojení. Varianta 2 popisuje případ, kdy byly pro přenos těchto zpráv (se stejným obsahem jako v předchozí variantě) použity čtyři TCP spojení. Konečně varianta 3, která popisuje případ, kdy pro každou ze zpráv bylo vytvořeno nové TCP spojení. Pozorováním zmíněné tabulky lze zjistit, že se zvyšujícím se počtem navázaných spojení klesá efektivita přenosu.

	varianta 1	varianta 2	varianta 3
inicializace [B]	4 821	19 284	20 826 720
payload [B]	2 479 340	2 479 340	2 185 920
ukončení spojení [B]	415	1 660	1 792 800
ping pong [B]	1 923 840	1 923 840	0
celkem přeneseno [B]	2 484 576	2 500 284	24 805 440
celkem přeneseno s ping pong [B]	4 403 180	4 424 124	24 805 440
z toho užitečná data [B]	1 663 200	1 663 220	1 663 200
efektivita přenosu [%]	66,94	66,52	6,70
efektivita přenosu s ping pong [%]	37,77	37,59	6,70

Tabulka 6.4. Spotřeba dat při celodenním provozu *lokátoru*

6.5 Měření spotřeby elektrické energie

Měření spotřeby elektrické energie bylo prováděno za pomoci ampérmetru při běžném provozu lokátoru, kdy odesílal informace o své aktuální poloze na *backendový server*. Data o aktuálně odebíraném proudu byla odečítána každých 5 sekund. Celkem bylo naměřeno 30 vzorků. Výsledky tohoto měření popisuje tabulka 6.5. Naměřené hodnoty pouze ilustrují přibližný odběr elektrické energie. Pro zjištění skutečné spotřeby (např. pro potřeby výpočtu doby běhu při zapojení baterie) by bylo nutné změřit více hodnot a vhodně je agregovat.

#	proud [mA]	#	proud [mA]
1	154,5	16	160,4
2	146,8	17	152,3
3	159,6	18	145,5
4	147,9	19	151,3
5	165,7	20	141,4
6	150,5	21	149,4
7	139,6	22	145,4
8	156,7	23	166,9
9	141,7	24	142,7
10	169,9	25	164,2
11	154,3	26	145,3
12	144,5	27	156,8
13	151,2	28	156,5
14	157,8	29	162,8
15	161,1	30	161,0

Tabulka 6.5. Odebíraný proud

Kapitola 7
Testování

7.1 Automatické testování

Backendový server lze automaticky otestovat spuštěním jednotkových testů příkazem `sbt test`. Testy pokrývají kritické funkcionality, jako jsou HTTP dotazy a odpovědi na ně, interní logika pro operace s daty apod. Těmito testy lze zajistit, že aplikační rozhraní se úpravou kódu nezměnilo a bude nadále možné komunikovat s aplikacemi, které ho využívají (např. *uživatelské rozhraní*). Testy lze také spouštět automatiky např. při nahrání nové revize do verzovacího systému Git, a tím ověřit, že aktualizovaný kód funguje správně.

7.2 Uživatelské testování

Výsledný software a hardware byl podroben uživatelskému testování. Každý uživatel obdržel scénář (příloha D), podle kterého postupoval, a jednoduchou uživatelskou příručku k aplikaci (příloha E). Nakonec své kroky zhodnotil v dotazníku, jehož výsledky jsou prezentovány v tabulce 7.1. Během celého testování byla přítomna osoba, která jednotlivé kroky testovaných subjektů pozorovala a zaznamenávala.

Celkem byla aplikace testována pěti subjekty – třemi muži (ve věku 20, 22 a 22 let) se základním, středoškolským a vysokoškolským vzděláním (M1, M2, M3) a dvěma ženami (ve věku 23 a 23 let) s vysokoškolským vzděláním (Z1, Z2). Přepis dotazníku, který subjekty vyplnily, zobrazuje tabulka 7.1.

Seznam dotazovaných aspektů

1. Pomohla Vám nápověda, která se zobrazila při první návštěvě stránky (tzv. onboarding), k získání většího přehledu, jak se aplikace ovládá?
2. Která část scénáře Vám dělala největší potíže. Jak dlouho a kde jste hledali, než jste daný bod splnili?
3. Hodnotíte procházení programem jako intuitivní (nemuseli jste hledat potřebné ovládací prvky)?
4. Byla pro Vás velikost všech prvků adekvátní? Hodnoťte velikost ikon, písma, řezy písma, použité fonty apod.
5. Přidání nového uživatele.
6. Úprava stávajícího uživatele.
7. Změna vlastního hesla.
8. Odstranění uživatele.
9. Přidání nového lokátoru.
10. Úprava stávajícího lokátoru.
11. Vygenerování konfiguračního souboru.
12. Nahrání konfigurace do fyzického lokátoru.
13. Odstranění lokátoru.

14. Přidání nového vozidla.
15. Úprava stávajícího vozidla.
16. Úprava flotil vozidla.
17. Odstranění vozidla.

č. otázky	M1	M2	M3	Z1	Z2	modus	průměr
1	NE	ANO	NE	NE	ANO	–	–
2	1	2	3	4	2	2	2,4
3	ANO	ANO	ANO	AN0	ANO	–	–
4	2	1	1	3	2	2	1,8
5	2	2	1	2	1	2	1,6
6	1	3	3	2	1	2	2
7	2	1	1	2	2	2	1,8
8	1	1	1	5	2	1	2
9	2	1	2	2	1	2	1,6
10	1	2	2	3	3	2	2,2
11	2	3	3	3	2	3	2,6
12	3	1	2	3	1	2	2
13	1	1	1	5	2	1	2
14	1	1	2	2	1	1	1,4
15	1	1	2	3	1	1	1,6
16	1	1	1	1	2	1	1,2
17	1	2	3	4	1	2	2,2
průměr	1,53	1,53	1,87	2,93	1,60	–	–

Tabulka 7.1. Výsledky dotazníku uživatelského testování

Uživatelé, kteří systém testovali, jej zhodnotili následovně:

- Celkový dojem mám dobrý, po pochopení základnách funkcí je užívání aplikace jednoduché.
- Aplikace mi přišla velmi intuitivní. Uživatelskou příručku bych ale více rozepsal. Hlavně bych se zaměřil více na konfiguraci lokátoru.
- Lehce složitější uživatelské prostředí, ale dá se to zvládnout – nevím, zda by byl systém vhodný pro běžnou populaci.
- Přidal bych možnost stažení více tras najednou jako jednu spojenou.

Z pozorování uživatelů při testování a výsledků dotazníku, kde mohli uvést návrhy na vylepšení, vyšly najevo některé nedostatky systému, které snižují jeho uživatelskou přívětivost. Nebyl odhalen žádný závažný problém, který by bránil ve využívání aplikace. Bylo nalezeno a opraveno několik detailů, jako například špatné obarvení políček formuláře s validními a nevalidními daty. Takovéto nedostatky byly opraveny.

Problém, který objevili všichni uživatelé, bylo nalezení tlačítka pro přidání nové entity (uživatele, lokátoru, vozidla). Na základě jejich pohybu kurzorem myši po obrazovce a následném dotazování vyplynulo, že tlačítko pro přidání nové entity je vhodné přesunout z horního levého rohu obrazovky do spodního pravého rohu obrazovky. Jelikož šlo pouze o drobnou změnu v designu, byl tento návrh zapracován.

Další problémy, které byly testováním odhaleny, se týkaly přenesení nastavení z *uživatelského rozhraní* do *lokátoru*. Pro uživatele bylo problematické přepínaní připojení

mezi dvěma WiFi sítěmi. K tomuto problému docházelo především z důvodu neúplného pochopení procesu konfigurace lokátoru. V uživatelské příručce byl tento postup popsán stručně. Koresponduje to i s návrhem na vylepšení, který jeden z uživatelů zmínil. Nicméně všichni uživatelé nahrání nové konfigurace *lokátoru* úspěšně provedli. Tento proces tedy zůstal i po testování nezměněný. Ani po diskusi s uživateli nebyla totiž nalezena lepší varianta, která by byla intuitivnější a uživatelsky přívětivější.

Dalším zjištěním plynoucím z uživatelského testování je fakt, že nápovědu, která provází nového uživatele základními procesy v aplikaci (např. přidání nových entit, jejich úprava, zobrazení ujetých tras apod.), využili pouze dva uživatelé. Tato skutečnost může být také důvodem, proč uživatelé mnohdy některé funkcionality v aplikaci hledali příliš dlouho.

Kapitola 8
Závěr

Cílem práce bylo navrhnout a vytvořit hotové řešení pro sledování vozidel v reálném čase s ohledem na snížení pořizovacích a provozních nákladů. Práce klade největší důraz na backendovou část systému. Její software je navržen tak, aby ji bylo možné provozovat na jednodeskovém počítači RaspberryPi. Nasazování aplikace s tím také počítá a součástí projektu je i build skript, s jehož pomocí lze aplikaci snadno a efektivně (rychle) nasadit i na počítač s takto nízkým výkonem.

Důraz byl kladen také na zabezpečení celého systému. Jednou z hlavních předností systému je totiž fakt, že citlivá data, jako jsou historie pohybu, nejsou odesílána ani přeposílána přes servery třetích stran. Dále je veškerá komunikace, která mezi jednotlivými částmi probíhá šifrování.

Práce splňuje vytyčené cíle a z nich plynoucí funkční a nefunkční požadavky. Systém umožňuje uživatelům sledovat aktuální polohu vozidel. Tuto polohu zaznamenává a dává ji k dispozici pro zpětné prohlížení. Dále umožňuje exportování jednotlivých tras pro případ dalšího zpracování. Také poskytuje rozhraní pro správu lokátorů a vozidel jako takových.

Všechny zmíněné operace jsou uživatelům dostupné přes webové rozhraní. To je navrženo tak, aby bylo intuitivní a vyžadovalo minimum informací před začátkem používání systému. Návrh uživatelského rozhraní byl testován uživatelskými testy. Jejich výsledek je v práci reflektován a vychází z něj podněty pro vylepšení Pro případné další rozšíření, bude systém (zdrojové kódy jeho částí) dostupný na adrese: `https://github.com/vehicle-tracking-system`.

Literatura

[1] Agafonkin, V. *Leaflet* [software]. [vid. 15. 4. 2021]. Dostupné na `https://leafletjs.com`.

[2] Alexander, A. *Scala Cookbook*. O'Reilly, 2013. ISBN 978-1-449-33961-6.

[3] Arduino S.r.l. *Arduino Board Nano* [online]. [vid. 20. 4. 2021]. Dostupné na `https://www.arduino.cc/en/pmwiki.php?n=Main/ArduinoBoardNano#techspecs`.

[4] Arduino S.r.l. *Arduino Nano 33 IoT* [online]. [vid. 20. 4. 2021]. Dostupné na `https://store.arduino.cc/arduino-nano-33-iot`.

[5] Axios. *Axios* [software]. [vid. 14. 4. 2021]. Dostupné na `https://github.com/axios/axios`.

[6] Banks, A., E. Briggs, K. Borgendale a R. Gupta. *MQTT Version 5.0 – OASIS Standard* [online]. [vid. 11. 4. 2021]. Dostupné na `https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html`.

[7] Barnes, R., J. Hoffman-Andrews, D. McCarney a J. Kasten. *Automatic Certificate Management Environment (ACME)* [online]. [vid. 22. 4. 2021]. Dostupné na `https://tools.ietf.org/html/rfc8555`.

[8] Barratt, T. S. a S. P. Boyd. Fitting a Kalman Smoother to Data. In: *2020 American Control Conference (ACC)*. 2020. s. 1526–1531. Dostupné na DOI 10.23919/ACC45564.2020.9147485.

[9] Belshe, M., R. Peon a M. Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)* [online]. [vid. 19. 4. 2021]. Dostupné na `https://tools.ietf.org/html/rfc7540#section-8.1.2.2`.

[10] Birchall, Ch. *scalacache* [software]. [vid. 17. 4. 2021]. Dostupné na `https://github.com/cb372/scalacache`.

[11] Blanchon, B. *Mastering ArduinoJson 6*. France: Benoît Blanchon, Antony, 2020.

[12] Brown, T. *Circe* [software]. [vid. 24. 4. 2021]. Dostupné na `https://circe.github.io/circe/`.

[13] Bryce, R., T. Shaw a G. Srivastava. MQTT-G. In: *2018 41st International Conference on Telecommunications and Signal Processing (TSP)*. IEEE, 2018 [vid. 21. 4. 2021]. s. 1–4. ISBN 978-1-5386-4695-3. Dostupné na DOI 10.1109/TSP.2018.8441479. Dostupné na `https://ieeexplore.ieee.org/document/8441479/`.

[14] Dierks, T. a E. Rescorla. *The Transport Layer Security (TLS) Protocol, Version 1.2* [online]. [vid. 20. 4. 2021]. Dostupné na `https://tools.ietf.org/html/rfc5246`.

[15] Docker Inc. *Orientation and setup* [online]. [vid. 20. 4. 2021]. Dostupné na `https://docs.docker.com/get-started/`.

[16] Electronic Frontier Foundation. *Certbot* [software]. [vid. 24. 4. 2021]. Dostupné na `https://certbot.eff.org`.

[17] Fette, I. a A. Melnikov. *The WebSocket Protocol* [online]. [vid. 11. 4. 2021]. Dostupné na `https://tools.ietf.org/html/rfc6455`.

[18] Fielding, R. a J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing* [online]. [vid. 19. 4. 2021]. Dostupné na `https://tools.ietf.org/html/rfc7230`.

[19] Goes, J. A. De a the ZIO Contributors. *ZIO* [software]. [vid. 11. 4. 2021]. Dostupné na `https://zio.dev`.

[20] Google Inc. *Protocol Buffers - documentation* [online]. [vid. 14. 4. 2021]. Dostupné na `https://developers.google.com/protocol-buffers/docs/overview`.

[21] Google Inc. *Protocol Buffers* [online]. [vid. 14. 4. 2021]. Dostupné na `https://developers.google.com/protocol-buffers`.

[22] Google Inc. [online]. [vid. 14. 4. 2021]. Dostupné na `https://developers.google.com/protocol-buffers/docs/encoding#varints`.

[23] govorox. *SSLClient* [software]. [vid. 14. 4. 2021]. Dostupné na `https://github.com/govorox/SSLClient`.

[24] GPS-server.net. *GPS-server.net* [software]. [vid. 5. 3. 2021]. Dostupné na `https://www.gps-server.net`.

[25] GPSWOX Ltd. *GPSWOX* [software]. [vid. 12. 4. 2021]. Dostupné na `https://www.gpswox.com`.

[26] H2. *H2 Database Engine* [software]. [vid. 20. 4. 2021]. Dostupné na `https://www.h2database.com/html/main.html`.

[27] Hale, C. *sbt-assembly* [software]. [vid. 25. 4. 2021]. Dostupné na `https://github.com/sbt/sbt-assembly`.

[28] Hologram, Inc. *Hologram* [online]. [vid. 1. 5. 2021]. Dostupné na `https://www.hologram.io/products/iot-sim-card`.

[29] ISO/IEC 7498-3. *Informační technologie - Propojení otevřených systémů - Základní referenční model: Pojmenování a adresování*. 2 vyd. Praha, 1998.

[30] Jones, M. *JSON Web Algorithms (JWA)* [online]. [vid. 22. 4. 2021]. Dostupné na `https://tools.ietf.org/html/rfc7518`.

[31] Jones, M., J. Bradley a N. Sakimura. *JSON Web Token (JWT)* [online]. [vid. 21. 4. 2021]. Dostupné na `https://tools.ietf.org/html/rfc7519`.

[32] Jones, M., J. Bradley a N. Sakimura. *JSON Web Signature (JWS)* [online]. [vid. 18. 4. 2021]. Dostupné na `https://tools.ietf.org/html/rfc7515`.

[33] Jones, M. a J. Hildebrand. *JSON Web Encryption (JWE)* [online]. [vid. 22. 4. 2021]. Dostupné na `https://tools.ietf.org/html/rfc7516`.

[34] Kolena, J. *Tasker* [software]. [vid. 14. 4. 2021]. Dostupné na `https://github.com/jendakol/bomb-game/blob/0c25175c45a22eec3f1b15a869de26f3343957da/src/Tasker.h`.

[35] Kolena, J. *Alarm garage* [software]. [vid. 14. 4. 2021]. Dostupné na `https://github.com/alarm-garage`.

[36] Kolena, J. *Bomb game* [software]. [vid. 14. 4. 2021]. Dostupné na `https://github.com/jendakol/bomb-game`.

[37] KŘÍŽ, J. *Bezpečnostní analýza Linux Unified Key Setup (LUKS)*. Praha: ČVUT v Praze, 2019. Bakalářská práce. Dostupné na http://hdl.handle.net/10467/83220.

[38] MOZILLA. *An overview of HTTP* [online]. [vid. 19. 4. 2021]. Dostupné na https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

[39] MQTT. *MQTT* [online]. [vid. 11. 4. 2021]. Dostupné na https://mqtt.org/.

[40] NORRIS, R. *doobie* [software]. [vid. 15. 4. 2021]. Dostupné na https://tpolecat.github.io/doobie/.

[41] NURSEITOV, N., M. PAULSON, R. REYNOLDS a C. IZURIETA. Comparison of JSON and XML data interchange formats: A case study. *22nd International Conference on Computer Applications in Industry and Engineering 2009, CAINE*. 2009, ročník 9, č. 1, s. 157–162.

[42] O'LEARY, N. *PubSubClient* [software]. [vid. 14. 4. 2021]. Dostupné na https://github.com/knolleary/pubsubclient.

[43] ODERSKY, M. *The Scala Programming Language* [software]. [vid. 24. 4. 2021]. Dostupné na https://www.scala-lang.org.

[44] OPENSTREETMAP FOUNDATION. *Open Street Map* [software]. [vid. 15. 4. 2021]. Dostupné na https://www.openstreetmap.org.

[45] PLATFORMIO LABS OÜ. *PlatformIO* [software]. [vid. 24. 4. 2021]. Dostupné na https://platformio.org.

[46] RNG, TLS overhead - netsekure. [online]. [vid. 11. 4. 2021]. Dostupné na http://netsekure.org/2010/03/tlsoverhead/.

[47] SHYMANSKYY, V. *TinyGSM* [software]. [vid. 14. 4. 2021]. Dostupné na https://github.com/vshymanskyy/TinyGSM.

[48] SRL, Things Mobile. *Coverage* [online]. [vid. 9. 5. 2021]. Dostupné na https://www.thingsmobile.com/business/coverage/coverage.

[49] SUERETH, J. *sbt-native-packager* [software]. [vid. 25. 4. 2021]. Dostupné na https://github.com/sbt/sbt-native-packager.

[50] THE C++ RESOURCES NETWORK. *C++ Reference* [online]. [vid. 20. 4. 2021]. Dostupné na http://www.cplusplus.com/reference/.

[51] THE WIRESHARK TEAM. *Wireshark* [software]. [vid. 24. 4. 2021]. Dostupné na https://www.wireshark.org.

[52] THINGS MOBILE SRL. *Unlimited plan* [online]. [vid. 9. 5. 2021]. Dostupné na https://www.thingsmobile.com/business/plans/unlimited-plan.

[53] TLV S.R.O. *GPS Dozor* [software]. [vid. 5. 3. 2021]. Dostupné na https://www.gpsdozor.cz/.

[54] TOKAREV, E. *sst-seed.g8* [software]. [vid. 15. 4. 2021]. Dostupné na https://github.com/avast/sst-seed.g8.

[55] TRACCAR LTD. *Traccar* [software]. [vid. 12. 4. 2021]. Dostupné na https://www.traccar.org/.

[56] TYPELEVEL. *http4s* [software]. [vid. 24. 4. 2021]. Dostupné na https://http4s.org.

[57] WAMPLER, D. a A. PAYNE. *Programming Scala*. Sebastopol: O'Reilly, 2009. ISBN 978-0-596-15595-7.

[58] Yokotani, T. a Y. Sasaki. Comparison with HTTP and MQTT on required network resources for IoT. In: *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*. IEEE, 2016. s. 1–6. ISBN 978-1-5090-0744-8. Dostupné na `http://ieeexplore.ieee.org/document/7814989/`.

[59] You, E. *Vue.js* [software]. [vid. 20. 4. 2021]. Dostupné na `https://vuejs.org`.

[60] You, E. a E. San Martin Morote. *Vue Router* [software]. [vid. 24. 4. 2021]. Dostupné na `https://router.vuejs.org`.

[61] Zerynth. *DOIT Esp32 DevKit v1* [online]. [vid. 20. 4. 2021]. Dostupné na `https://docs.zerynth.com/latest/reference/boards/doit_esp32/docs/`.

[62] Česko. *Úplné znění zákona č. 361/2000 Sb. o provozu na pozemních komunikacích a o změnách některých zákonů (zákon o silničním provozu)*. dvacáté druhé vyd. Praha: Armex Publishing s.r.o., 2020. ISBN 978-80-87451-69-4.

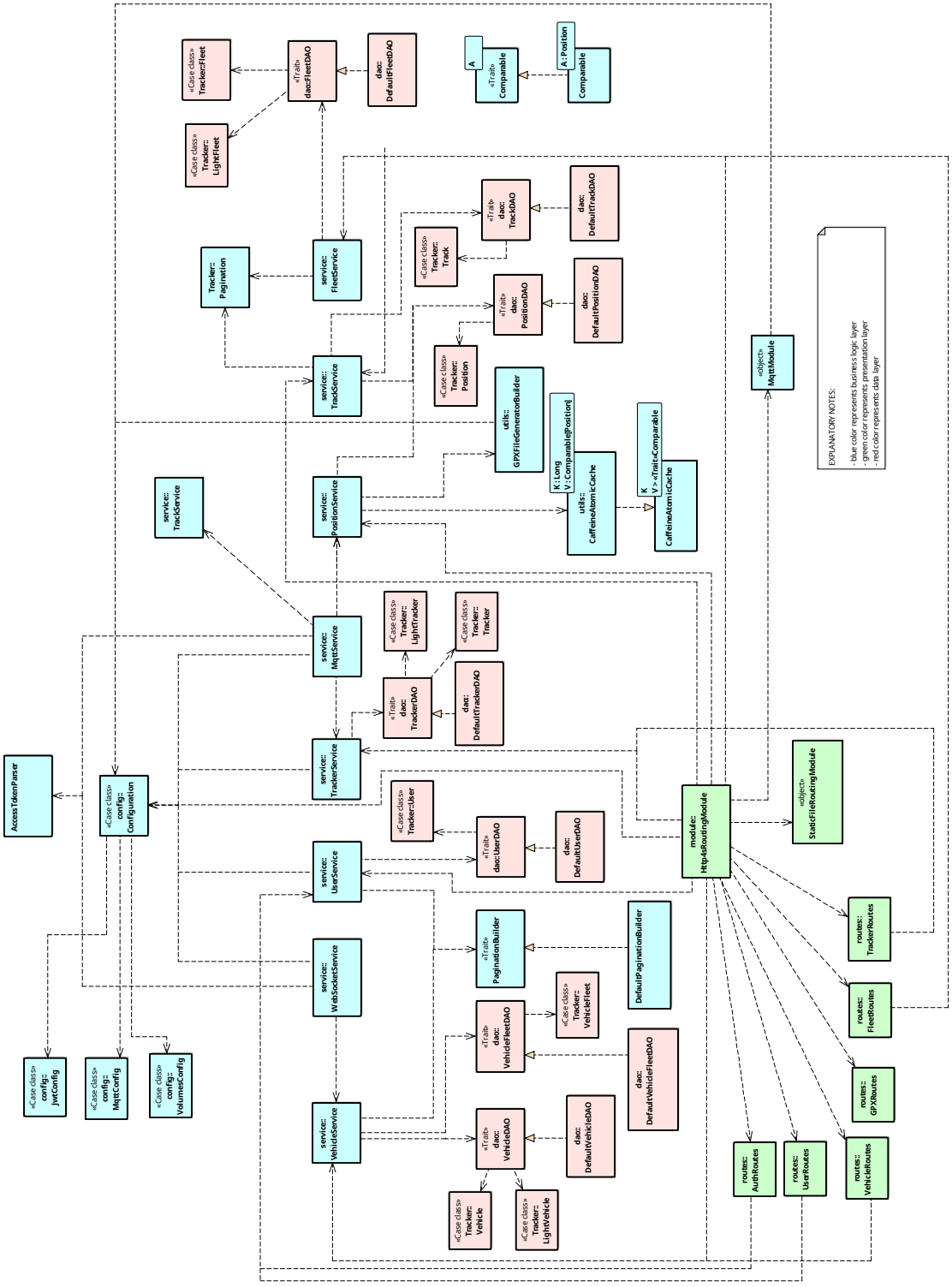
Příloha A
Seznam použitých zkratek

AP	Access Point
EEPROM	Electrically Erasable Programmable Read-Only Memory
GPRS	General Packet Radio Service
GPS	Global Positioning System
GPX	GPS Exchange Format
HMAC	Keyed-hash Message Authentication Code
HOCON	Human-Optimized Config Object Notation
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JAR	Java Archive
JDBC	Java Database Connectivity
JOSE	Javascript Object Signing and Encryption
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
JWT	JSON Web Token
LSB	Least Significant Bit
MAC	Media Access Control
MQTT	Message Queuing Telemetry Transport
MSB	Most Significant Bit
M2M	Machine to Machine
QoS	Quality of Service
RAII	Resource acquisition is initialization
RAM	Random Access Memory
SPIFFS	SPI Flash Filing System
SQL	Structured Query Language
SSID	Service Set Identifier
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	Extensible Markup Language

Příloha B
Obsah přiloženého paměťového média

```
./
|-- doc                                                    dokumentace
|-- front                              zdrojové kódy uživatelského rozhraní
|-- model                                     3D model krabičky pro lokátor
|-- server                               zdrojové kódy backednového serveru
|-- text                                              zdrojový kód textu práce
|-- thesis.pdf                                      text práce ve formátu PDF
|-- tracker                                           zdrojové kódy lokátoru
`-- tracker-front          zdrojové kódy uživatelského rozhraní lokátoru
```

Příloha C
Diagram tříd



Příloha D
Testovací scénář

Uživatelské testování systému pro sledování vozidel

Květen 2021

Pokyny

Na následujících stránkách naleznete popsané postupy operací, které se budou po Vás chtít vykonat. Postupujte systematicky a jednotlivé kroky dělejte postupně, nepřeskakujte. Pokud operaci nebudete schopni provést, oznamte to, případně se doptejte. Tuto možnost se snažte využít v co nejmenší míře. K dispozici máte také uživatelský manuál.

Cílem není Vás zkoušet ze znalostí a schopnosti ovládat počítač. Naopak. Je více než možné, že pokud nějakou operaci nebude možné provést, nebo nebudete vědět, jak ji vykonat, že je na vině chyba v systému.

Po celou dobu testování s Vámi bude osoba, která bude vaše jednotlivé kroky pozorovat a zaznamenávat Váš postup. Na konci tohoto dokumentu naleznete formulář, ve kterém prosím popravdě zhodnoťte dotazované aspekty testovaného programu.

Jméno: .....

Pohlaví: .....

Věk: .....

Nejvyšší dosažené vzdělání: .....

Až budete připraveni, otočte na další stranu

Uživatelské rozhraní

Na počítači před Vámi je spuštěna aplikace pro sledování vozidel.

1. Přihlaste se do aplikace pomocí přihlašovací jména `tester` a hesla `20tester21`.
2. Zobrazte si aktuální pozici vozidla s názvem `Škoda 100`.
3. Zjistěte, jak rychle jelo vozidlo `Škoda 100` při poslední známe poloze.
4. Zobrazte aktuální pozici všech dostupných vozidel.
5. Zobrazte seznam všech vozidel.
6. Přidejte nové vozidlo s libovolným názvem, ale přiřaďte jej do flotily `Testeři`.
7. Zobrazte seznam všech lokátorů (trackerů).
8. Vytvořte nový záznam lokátoru a přiřaďte jej k vozidlu, které jste vytvořili v předchozím kroku. Pojmenujte ho `Sledovač`.
9. Vytvořte nového uživatele s libovolným jménem, uživatelským jménem a heslem. Nový uživatel bude mít všechny dostupné role. Zde napište uživatelské jméno a heslo přidaného uživatele:

Pokud jste dokončili všechny kroky, otočte na další stranu.

Lokátor

Na stole máte připravený lokátor. Zařízení, které komunikuje s prostředím, se kterým jste se již seznámili.

1. Zapněte lokátor (tracker).
2. Zobrazte seznam všech lokátorů.
3. Zobrazte detail Vámi vytvořeného lokátoru.
4. Stáhněte konfigurační soubor.
5. Nahrajte stažený konfigurační soubor do lokátoru.
6. Přesvědčte se v uživatelském rozhraní, že poslední pozice vozidla, ke kterému jste přidali lokátor, byla aktualizována.

Pokud jste dokončili všechny kroky, otočte na další stranu.

Uživatelské rozhraní a lokátor

K následujícím krokům budete potřebovat jak uživatelské rozhraní na počítači, tak lokátor.

1. Odstraňte Vámi vytvořené vozidlo z flotily `Testeři` a přidejte ho do flotily `Minibus`.
2. Změňte své heslo na `tester2021`.
3. Odhlaste se ze systému.
4. Přihlaste se s novým heslem.
5. Představte si, že jste ztratili lokátor a chcete do svého vozidla umístit nový. Nahrajte do lokátoru před Vámi novou konfiguraci.
6. Stáhněte si záznam trasy vozidla `Škoda` ze dne 8. 4. 2021.
7. Ukončete práci se systémem, odhlaste se.

Pokud jste dokončili všechny kroky, otočte na další stranu.

Konec.
Na následující straně začíná dotazník přívětivosti uživatelského rozhraní.

Dotazník

Pomohla Vám nápověda, která se zobrazila při první návštěvě stránky (tzv. onboarding), k získání většího přehledu, jak se aplikace ovládá? ..... Ano Ne

Která část scénáře Vám dělala největší potíže. Jak dlouho a kde jste hledali, než jste daný bod splnili?

---

rozhodně souhlasím →
← rozhodně nesouhlasím

Hodnotíte procházení programem jako intuitivní (nemuseli jste hledat potřebné ovládací prvky) 5 4 3 2 1

Byla pro Vás velikost všech prvků adekvátní? Hodnoťte velikost ikon, písma, řezy písma, použité fonty apod. ... Ano Ne

Pokud chcete, můžete připojit podrobnější komentář k čitelnosti:

Správa uživatelů

Následují otázky hodnoťte na uvedené škále podle toho, jak intuitivní Vám daná operace připadala.

rozhodně souhlasím →
← rozhodně nesouhlasím

Přidání nového uživatele. 5 4 3 2 1

rozhodně souhlasím →
← rozhodně nesouhlasím

Úprava stávajícího uživatele. 5 4 3 2 1

rozhodně souhlasím →
← rozhodně nesouhlasím

Změna vlastního hesla. 5 4 3 2 1

rozhodně souhlasím →
← rozhodně nesouhlasím

Odstranění uživatele. 5 4 3 2 1

Správa lokátorů

Následují otázky hodnoťte na uvedené škále podle toho, jak intuitivní Vám daná operace připadala.

rozhodně souhlasím →
← rozhodně nesouhlasím

Přidání nového lokátoru. 5 4 3 2 1

rozhodně souhlasím →
← rozhodně nesouhlasím

Úprava stávajícího lokátoru. 5 4 3 2 1

rozhodně souhlasím →
← rozhodně nesouhlasím

Vygenerování konfiguračního souboru. 5 4 3 2 1

rozhodně souhlasím →
← rozhodně nesouhlasím

Nahrání konfigurace do fyzického lokátoru. 5 4 3 2 1

rozhodně souhlasím →
← rozhodně nesouhlasím

Odstranění lokátoru. 5 4 3 2 1

Správa vozidel

Následují otázky hodnoťte na uvedené škále podle toho, jak intuitivní Vám daná operace připadala.

rozhodně souhlasím →
← rozhodně nesouhlasím

Přidání nového vozidla. 5 4 3 2 1

rozhodně souhlasím →
← rozhodně nesouhlasím

Úprava stávajícího vozidla. 5 4 3 2 1

rozhodně souhlasím →
← rozhodně nesouhlasím

Úprava flotil vozidla. 5 4 3 2 1

rozhodně souhlasím →
← rozhodně nesouhlasím

Odstranění vozidla. 5 4 3 2 1

Dokážete si představit, že by takovýto systém byl součástí Vaše každodenního života. Měli byste ho nainstalovaný např. ve svém osobním vozidle? .....

rozhodně souhlasím →
← rozhodně nesouhlasím

5	4	3	2	1

Zhodnoťte celkový dojem z aplikace, případně připojte další připomínky.

Příloha E
Uživatelská příručka

Uživatelská příručka

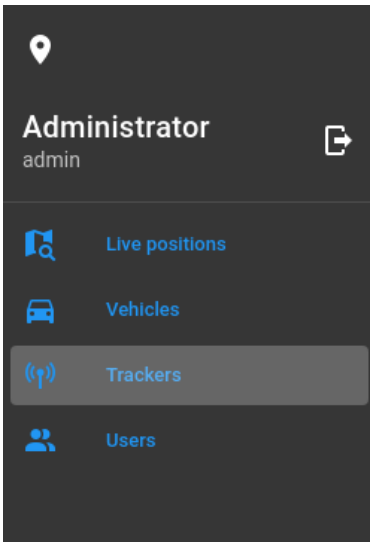
1. Přidání nového lokátoru

Pokud chcete přidat nový lokátor (tracker), musíte jej nejprve vytvořit přes uživatelské rozhraní. V levém menu vyberte možnost *Trackers* (obr. 1). Poté vytvořte nový záznam o lokátoru. Kliknutím vyberte ze seznamu nově přidaný lokátor. Stáhněte konfigurační soubor (obr. 2) kliknutím na tlačítko **DOWNLOAD CONFIG FILE** .

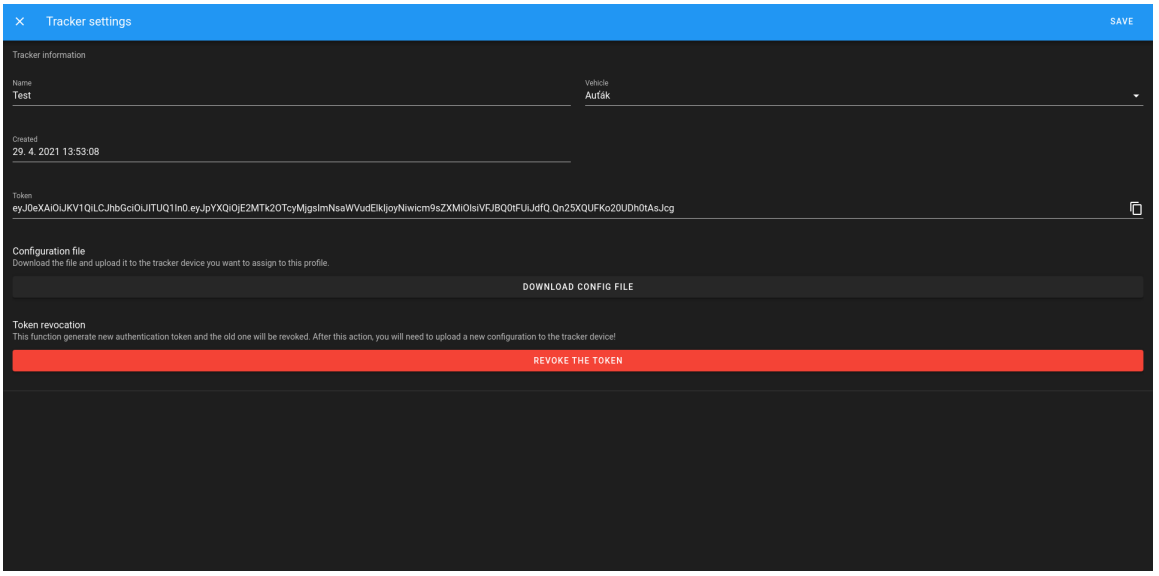
Nyní se připojte k WiFi vysílané Vaším lokátorem (*SSID*: tracker<MAC ADRESA> , *HESLO*: 12345678). Ve webovém prohlížeči přejděte na stránku *tracker.local*. Nahrajte předem stažený konfigurační soubor. Po úspěšném nahrání resetujte lokátor.

Nyní je zařízení spárováno s aplikací a můžete začít sledovat polohu.

UPOZORNĚNÍ: při změně informací o lokátoru (v aplikaci) je nutné provést celou tuto operaci znovu. Autentizační token přestane být platný.



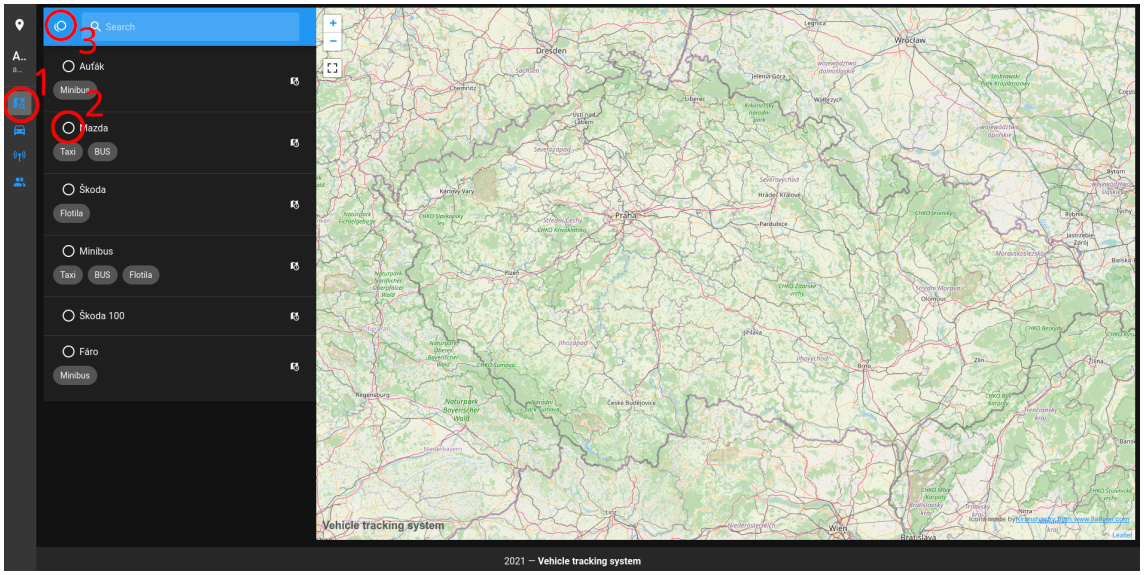
obr. 1



obr. 2

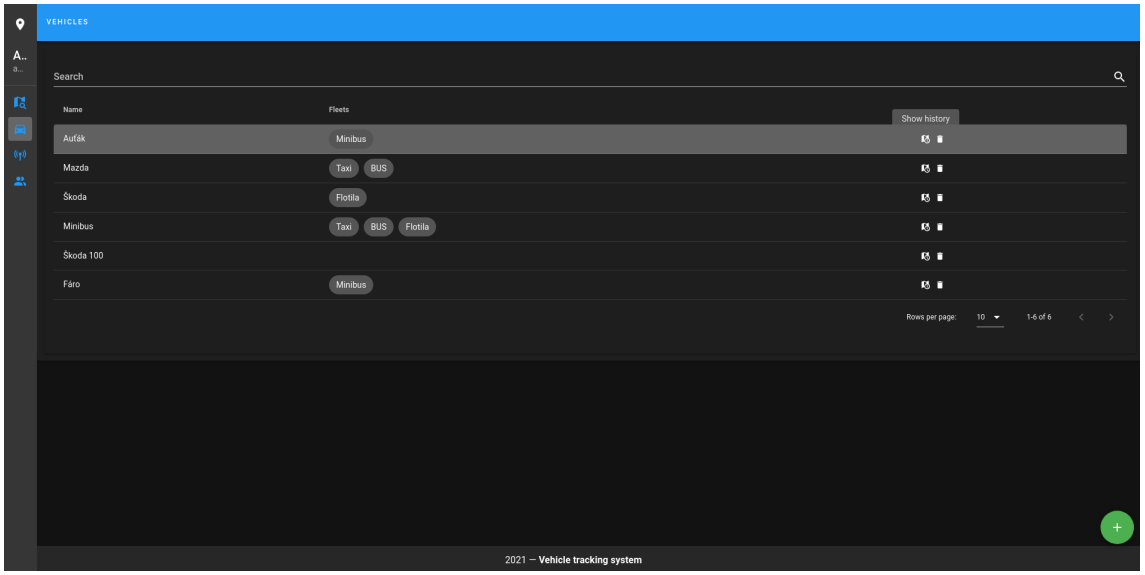
2. Zobrazení aktuálních poloh

1. V levém menu vyberte možnost *Live Positions*.
2. Kliknutím na název vozidla jej vyberte. V mapě se zobrazí jeho poslední známá pozice.
3. Pro zobrazení všech vozidel klikněte na kulaté tlačítko vedle pole pro vyhledávání.



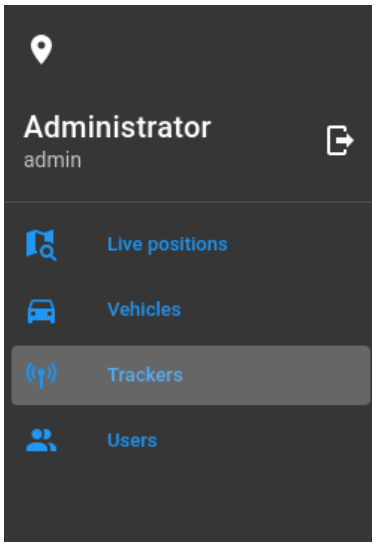
3. Úprava záznamu

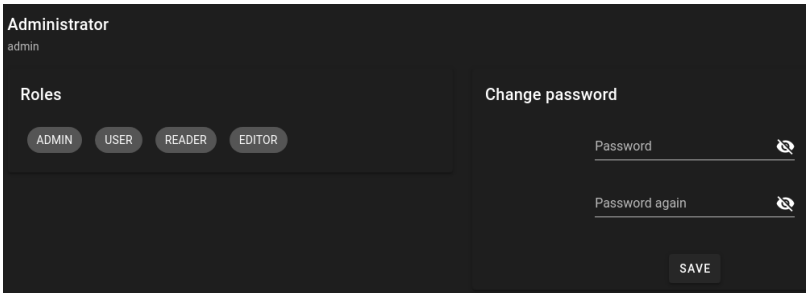
Kliknutím na záznam můžete zobrazit dialog pro úpravu. V levém sloupci jsou pak operace, které lze se záznamem dělat. Najetím kurzorem na jejich ikonu se zobrazí nápověda.



4. Změna uživatelského hesla

V levém menu klikněte na své své jméno (v tomto případě Administrator). Zobrazí se přehled, ve kterém můžete vidět všechny role, které máte přiřazeny. V pravé části obrazovky se nachází formulář pro změnu hesla. Vyplňte jej a klikněte na tlačítko **SAVE**. Vaše přístupové heslo se nyní změnilo. Použijte jej při příštím přihlášení do systému.





5. Lokátor

Před použitím lokátoru je nutné do něj nahrát vygenerovanou konfiguraci. Tu lze stáhnout v uživatelském rozhraní (pro detailnější popis vizte část 1). Po správné konfiguraci se řiďte stavem signalizační LED diody.

Signalizační led dioda

BARVA	TYP BLIKÁNÍ	POPIS
oranžová	rychle	lokátor se připojuje do sítě operátora
	pomalu	čekání na GPS signál
zelená	jedno bliknutí	lokátor změřil aktuální pozici
	dvě bliknutí po sobě	lokátor odeslal pozice na server
červená	rychle	chyba