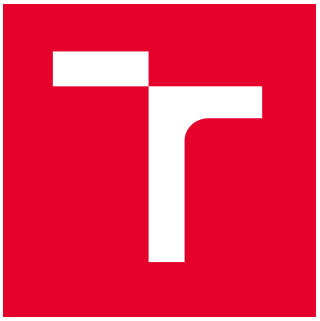
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2024 Bc. Jana Lázničková



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

ŘÍDICÍ A VIZUALIZAČNÍ SOFTWARE PRO PROPAGAČNÍ ROBOTICKÉ VOZÍTKO

CONTROL SYSTEM AND VISUALIZATION FOR A PROMOTIONAL ROBOTIC VEHICLE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jana Lázničková**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Miroslav Jirgl, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Studentka: Bc. Jana Lázničková ***ID:*** 220997

Ročník: 2 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Řídicí a vizualizační software pro propagační robotické vozítko

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je implementace řídicího a vizualizačního software pro funkční vzorek robotického vozítka určeného pro propagační účely zadávající firmy.

1. Popište stručně koncepci robotického vozítka se zaměřením na komunikační rozhraní a možnost výměny dat.
2. Popište cíle a způsoby využití vozítka.
3. Navrhněte koncepci řídicího a vizualizačního software.
4. Vytvořte vizualizaci.
5. Navrhněte a realizujte způsob pro možnost přidání virtuálních vozítek.
6. Implementujte software pro řízení a optimalizaci pohybu vozítek.
7. Software otestujte a prezentujte výsledky.

DOPORUČENÁ LITERATURA:

SICILIANO, B. a O. KHATIB. Handbook of Robotics [online]. Springer-Verlag Berlin Heidelberg, 2008 [cit. 2023-09-11]. ISBN 978-3-540-30301-5. Dostupné z: https://www.academia.edu/26665877/Handbook_Springer_of_Robotics

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 15.5.2024

Vedoucí práce: Ing. Miroslav Jirgl, Ph.D.

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

Abstrakt

Práce je zaměřena na softwarovou část autonomního robotického vozítka, určeného pro propagační účely firmy B:TECH. Jelikož se jedná o test proveditelnosti, výsledek práce slouží pouze jako idea vhodného návrhu softwaru pro robotické vozítko s možností využití tohoto řídícího softwaru v reálném provozu. Jsou zde uvedeny možnosti výměny dat mezi robotickým vozítkem a nadřazeným řídícím systémem. Navrhnutá koncepce popisuje řídící software zajišťující kooperaci mezi reálnými a simulovanými vozítky v provozu. Dále je navrhnuta platforma pro vizualizační aplikaci, která slouží k monitorování a ke kontrole pohybu vozítek a přepravovaných zásilek. Každé vozítko v aplikaci je doplněno o svůj aktuální status.

Klíčová slova

Robotické vozítko, komunikace, .NET MAUI, Secure Storage, vizualizace

Abstract

The thesis focuses on the software part of an autonomous robotic vehicle, designed for promotional purposes of the B:TECH company. As this is a feasibility test, the result of the work serves only as an idea of a suitable software design for the robotic vehicle with the possibility of using this control software in real operation. The possibilities of data exchange between the robotic vehicle and the master control system are presented. The proposed concept describes the control software ensuring cooperation between real and simulated vehicles in operation. Furthermore, a platform for a visualization application is designed to monitor and control the movement of the vehicles and transported shipments. Each vehicle in the application is updated with its current status.

Keywords

Robotic vehicle, communication, .NET MAUI, Secure Storage, visualization

Bibliografická citace

LÁZNIČKOVÁ, Jana. Řídicí a vizualizační software pro propagační robotické vozítko. Brno, 2024. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/160032. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Miroslav Jirgl.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	*Bc. Jana Lázničková*
VUT ID studenta:	*220997*
Typ práce:	*Diplomová práce*
Akademický rok:	*2023/24*
Téma závěrečné práce:	*Řídicí a vizualizační software pro propagační robotické vozítko*

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: 15. května 2024 ----------

podpis autora

Poděkování

Děkuji svému vedoucímu práce Ing. Miroslavu Jirglovi, Ph.D. za odbornou pomoc při zpracování mé diplomová práce. Dále bych ráda poděkovala konzultantovi Ing. Jiřímu Klimešovi za podnětné rady při tvorbě mé práce.

V Brně dne: 15. května 2024 ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

Úvod

Zadaní této diplomové práce bylo vytvořeno ve spolupráci se společností B:TECH, která má hlavní sídlo v Havlíčkově Brodě. Práce spadá do odvětví průmyslové automatizace, a to konkrétně do oblasti spojené s řízením a robotikou. Cílem této práce je provést studii proveditelnosti zaměřenou na ověření funkčnosti softwaru. Dále lze získat zkušenosti o provozu této aplikace v reálném prostředí, zhodnotit její stabilitu při běžném používání a posoudit její použitelnost v praxi. Na základě získaných zkušeností je možné vytvořit optimální nabídku pro zákazníka, která bude odpovídat jeho specifickým požadavkům.

V dnešní době je nezbytně nutné, aby byl celý proces výroby či nákupu určitého typu spotřebního zboží co nejrychlejší. Snahou je veškerou lidskou činnost nahradit činností automatických zařízení. Tímto směrem se v poslední době začala ubírat i oblast transportu materiálu po výrobních halách, kde lidskou činnost začaly do určité míry nahrazovat robotická vozítka. Tato robotická vozítka umožňují dle své velikosti přepravovat různé typy materiálu ze startovacího bodu A do koncového bodu B. Jsou tedy schopna autonomně plnit nejrůznější úkoly dle svého předdefinovaného řídícího softwaru. Nutností logické části robotických vozítek je důsledná propracovanost pohybu vozítka v prostoru s výskytem dalších autonomních vozítek a jiných překážek. Dále je nutností ošetřit dle jakých způsobů se budou vozítka po hale pohybovat a také je zapotřebí ošetřit reakce na vnější vlivy působící na vozítko. Z těchto poznatků vyplývají důležité body této práce.

Práce je založena na seznámení se s koncepcí pro návrh logické části robotických vozítek. Tato část má za úkol výměnu dat mezi reálným modelem vozítka, popřípadě simulovaným a řídícím softwarem. Dále je nutné vytvořit vizualizační software, který je dalším bodem práce. Tento vizualizační software monitoruje pohyb vozítka a zobrazuje základní informace o něm. Dále je umožněno monitorovat jednotlivé zásilky, které jsou přepravovány vozítky. Následně je nutné uvažovat práci s virtuálními vozítky ve vizualizačním softwaru pro možnou kooperaci s ostatními vozítky. Závěrem práce je otestování a prezentování dosažených výsledků.

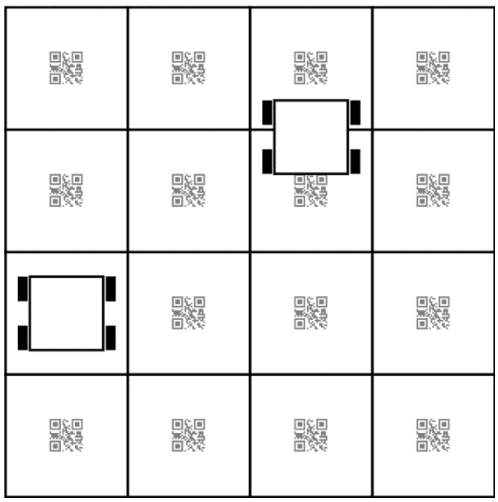
Tato diplomová práce je členěna do sedmi základních kapitol. V první kapitole je představena koncepce robotického vozítka s možným využitím robotických vozítek v praxi. Následující kapitola pojednává o teoretickém návrhu funkce aplikace se zaměřením na vizualizační a řídící software. Třetí kapitola je zaměřena na výběr vhodného komunikačního protokolu spojeného s výměnou dat a návrhem struktury předávané zprávy. Následně navazuje kapitola popisující konkrétní framework, který byl vybrán pro realizaci této práce. Jsou zde popsány funkce, které budou využívány pro tvorbu aplikace. V páté kapitole je popsána tvorba vizualizace v konkrétním frameworku. Dále je kapitola zaměřena na realizaci řídícího softwaru v podobě komunikace, řízení a plánování tras. V šesté kapitole jsou prezentovány dosažené výsledky a poslední kapitola nabízí souhrnné zhodnocení diplomové práce.

1. ROBOTICKÉ VOZÍTKO

Následující kapitola popisuje základní koncepci robotického vozítka se zaměřením na komunikační rozhraní. V kapitole 1.1 je popsán princip získávání dat a všeobecný princip komunikace mezi robotickým vozítkem a řídícím softwarem. Kapitola 1.2 je zaměřena na využití této úlohy a také na obecném využití robotických vozítek v praxi.

1.1 Koncepce

Vozítko bude mít za úkol pohybovat se v uzavřeném prostoru za účelem přepravy materiálu. V této aplikaci budou přepravovaným materiálem zásilky. Pohyb vozítka bude zajištěn pomocí snímaných QR (Quick Response) kódu připevněných na podlaze. Data z načtených QR kódů budou odesílána z jednotlivých robotických vozítek do řídícího softwaru. Tato odeslaná data budou obsahovat údaje o aktuální poloze neboli QR kód, na kterém se aktuálně vozítko nachází. Vozítko bude kromě údajů o poloze odesílat i data o svém aktuálním stavu (stav baterie, proudový odběr motorů atd.), která budou následně zpracovávána pro provoz.



Obrázek 1.1 Princip úlohy

Data mezi vozítkem a řídícím software budou odesílána přes rozhraní Wi-Fi, jelikož je nutné odesílat data bezdrátově. Data budou odesílána ve formě paketů obsahující vždy určitou informaci. Princip komunikace bude založen na dotazování se konkrétních vozítek, které po obdržení dotazu odešlou požadovaná data. Výběr konkrétního komunikačního protokolu je popsán v kapitole 3.5. Vozítka budou v pozici slave a řídící software v pozici master. Master bude dostávat data od vozítek, která budou použita pro řízení a vizualizaci. Konkrétní princip řízení a vizualizace je blíže popsán v kapitole 2. Cílem řízení je pouze zadávání povelů pro pohyb vozítka jako je například zastav, jeď doleva. Přímo mechanický pohyb vozítka v podobě řízení motorů, není součástí práce.

V reálném provozu bude více takových vozítek, proto je cílem řídícího softwaru zajistit bezpečnost, kooperaci s ostatními vozítky a co nejrychlejší a nejefektivnější přepravu zásilek do požadovaného místa určení. V této práci bude řízeno pouze jedno reálně vytvořené vozítko a ostatní vozítka budou simulovaná. Tato vozítka budou základem pro vizualizační rozhraní, kde bude uživateli umožněno monitorovat pohyb vozítek a také sledovat aktuální informace o převáženém nákladu.

1.2 Využití

V dnešní době jsou autonomní robotická vozítka využívána v aplikacích, které mají do určité míry usnadnit nebo plně nahradit lidskou činnost. To znamená, že člověk se v rámci používání robotických vozítek nachází pouze v pozici kontroly či údržby. Lze tedy říci, že robotická vozítka jsou plně soběstačná a dokážou se dle svého vnitřně implementovaného softwaru samostatně rozhodovat a plnit úkoly zadané člověkem. Obecné využití robotického vozítka může být v typických aplikacích jako jsou například přepravní mechanismy v průmyslu, v oblasti vzdělávání a výzkumu, nebo bezpečnostních aplikacích při kontrole určitých prostorů. V poslední době jsou hojně využívány i v domácnostech v podobě například robotických vysavačů či různých pomocníků.

Dané autonomní robotické vozítko, navržené pro propagační účely může sloužit jako zkušební platforma pro vývoj a testování nových technologií v oblasti autonomních vozidel a softwaru pro řízení. Úloha je využitelná v širokém rozsahu aplikací týkajících se přepravy materiálů a kontroly nad přepravou. Dané vozítko bude tedy plnit požadavky řídícího softwaru a vykonávat svoji činnost v podobě přepravy zásilek. Nezbytnou součástí je kolaborace vozítek s lidmi, a tudíž je vozítko vybaveno bezpečnostními prvky minimalizující následky či pravděpodobnost kolize. O realizaci výše uvedených požadavků se stará nadřazený systém v podobě řídícího a vizualizačního softwaru.

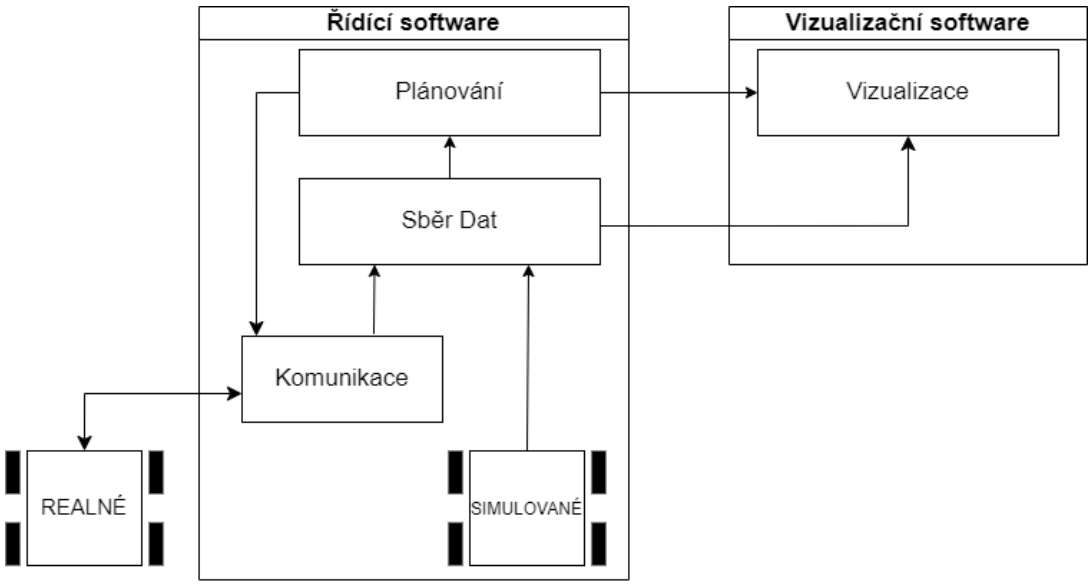
Řídící software je navržen tak, aby byl univerzální pro aplikace týkající se přepravy zásilek s ohledem na požadavky zákazníka. Software není pevně spjat s konkrétním typem robotického vozítka, ale při změně implementace lze změnit typ řízeného prvku (robotické vozítko). Vizualizační software je hlavním nástrojem pro monitorování dané úlohy. Jedná se o variabilní nástroj, který umožňuje monitorování vozítek a přepravovaných zásilek. Tento software je v podobě mobilní či desktopové aplikace spustitelné na různých typech chytrých zařízení. Obsluha se tedy nemusí nacházet v bezprostřední blízkosti manipulačního prostoru, ale může úlohu monitorovat odkudkoliv. Jediným požadavkem je to, aby bylo ovládací zařízení (mobilní telefon, PC atd.) připojeno ke stejné síti ve které se nacházejí robotická vozítka.

2. KONCEPT ŘÍDÍCÍHO A VIZUALIZAČNÍHO SOFTWARU

Tato kapitola je zaměřena na návrh vizualizačního a řídícího softwaru dané aplikace. V kapitole 2.1 je popsáno propojení řídícího a vizualizačního softwaru s jejich klíčovými funkcemi. Kapitola 2.2 pojednává o teoretickém řešení řídícího algoritmu robotického vozítka. V kapitole 2.3 je blíže specifikován návrh vizualizačního uživatelského rozhraní s popisem jednotlivých logických řešení obrazovek aplikace.

2.1 Propojení řídícího a vizualizačního softwaru

Celá aplikace bude rozdělena do dvou logických celků, kde každý bude mít na starosti jednotlivé úlohy. Znázornění propojení obou softwarů je na obrázku 2.1.



Obrázek 2.1 Návrh funkce aplikace

První částí, kterou bude zajišťovat řídící software, je komunikace s reálným vozítkem. Řídící systém bude ve spojení s reálným vozítkem prostřednictvím komunikačního rozhraní, přičemž bude získávat data o aktuální poloze vozítka a stavu vozítka. Data o poloze budou následně využívány k plánování optimální trasy vozítka a následným pokynům určující pohyb. Tyto pokyny budou posílány vozítku zpět přes komunikační rozhraní. Dále řídící software bude umožňovat vytvořit simulované vozítko, které bude mít obdobné chování jako reálné. S tímto typem vozítka však nebude nutná komunikace přes komunikační rozhraní. Komunikace bude probíhat uvnitř softwaru.

Ve vizualizačním softwaru poté probíhá zpracování dat ve vizuální podobě. Jednotlivé koncepty těchto softwaru jsou popsány níže.

2.2 Řídící SW

2.2.1 Řízení vozítka

Princip pohybu vozítka bude založen na snímání QR kódů. QR kódy budou rozmístěny rovnoměrně do šachovnicového uspořádání, které bude nutné nadefinovat. Vozítko bude vybaveno kamerou, která tyto QR kódy bude snímat. Data z kamery ve vhodném formátu bude vozítko odesílat řídícímu softwaru pro další zpracování. V QR kódech bude zakódováno unikátní číslo sloužící k identifikaci aktuální polohy vozítka. Znát polohu vozítka je nezbytné pro plánování trasy a přesunu vozítka na další QR kód.

Jelikož budou QR kódy rozmístěny jako šachovnicové pole, bude se vozítko pohybovat dle maticového systému. Daná aplikace bude mít šachovnicové pole o velikosti 8 x 8 QR kódů, tudíž celkem bude obsahovat 64 odlišných QR kódů. Zpráva od vozítka bude vždy obsahovat řádek a sloupec daného QR kódu, ze kterého lze při znalosti rozmístění kódů určit danou polohu vozítka.

Způsob řízení je založen na přesunu vozítka z bodu A do bodu B (přes více QR kódů). Tato trasa bude plánována dle specifikací zásilky a jejího startovacího a cílového bodu. Detailnější popis plánování trasy je popsán v podkapitole 2.2.2.

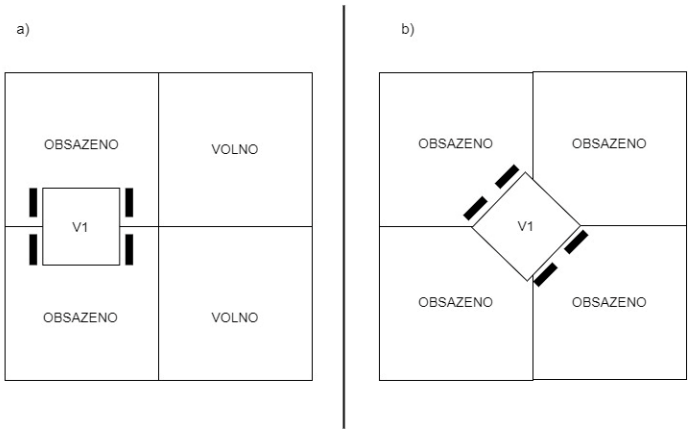
Vozítko se bude pohybovat pouze přímým směrem. Kdyby se vozítko pohybovalo i úhlopříčným směrem, došlo by při přesunu vozítka k zabrání 4 polí s QR kódy naráz. Tím by se zmenšila možnost využívat tyto pole jinými vozítky. Při pohybu přímým směrem, vozítko obsadí pouze dvě pole. Rozdíl situací je znázorněn na obrázku č. 2.2. Na tomto obrázku je patrné, že pohyb přímým směrem je efektivnější a jednodušší.

Kdy a jak vozítko dostane příkaz pro přesun na vedlejší QR kód je popsáno v podkapitole 2.2.2 plánování trasy.

Kromě informací o poloze bude vozítko posílat i data o náhlých událostech. Náhlou událostí je například ztráta polohy vozítka, kdy při přesunu vozítko nenalezne QR kód nebo je z bezpečnostních důvodů vozítko zastaveno (výskyt jiného vozítka či osob na trase). V případě, kdy vozítko ztratí QR kód bude sebelokalizováno. Při sebelokalizaci vozítko projede vzdálenost o velikosti dvou čtverců (polí) od své aktuální pozice a snaží se nalézt nejbližší QR kód. Pokud nalezne jakýkoli QR kód, nahlásí svou aktuální polohu řídícímu softwaru, kterou si software uloží jako výchozí pozici pro následné plánování trasy. Pokud vozítkem QR kód není nalezen, je pohyb vozítka zastaven a je přivolána obsluha. indikací ve vizualizační aplikaci. Pro snazší nalezení vozítka je k tomuto oznámení přiložena i poslední známá pozice. Pokud dojde k této situaci řídící software obsadí od vozítka dva čtverce, na každou stranu, aby nedošlo ke kolizi s ostatními vozítky. A ostatní vozítka nemohou při pohybu tyto pole využívat do doby vyřešení situace.

Jelikož je vozítko napájeno z baterie, může dojít při provozu k vybití baterie vozítka. Vozítko hlásí stav baterie průběžně a řídící software vyhodnocuje kdy má vozítku dát povel pro přesun vozítka do dokovací stanice. Pokud bude napětí baterie příliš nízké

a vozítko by nebylo schopno dokončit trasu, mohlo by být nahrazeno jiným nejbližším volným vozítkem.



Obrázek 2.2 Možné pohyby vozítek

2.2.2 Možnosti přidání vozítek

Nedílnou součástí řídícího softwaru a následného řízení je vytvoření a přidání vozítek do aplikace. Vozítka v aplikaci budou moci být přidávána jako reálná nebo jako simulovaná. Základem pro přidání vozítek bude uživatelská obrazovka umožňující přidání reálného či simulovaného vozítka. Pro přidání bude obrazovka obsahovat tlačítka rozlišující reálné a simulované vozítko. Uživatel při přidání vozítka bude moci měnit jeho přednastavené vlastnosti týkající se nastavení IP adresy, portu komunikace a identifikačního ID vozítka. Primárně budou vozítkům přidělovány vlastnosti dle jejich typu a pořadí přidání do aplikace.

Nastavená vozítka budou moci být ukládána například do kolekce. Kolekce funguje jako kontejner, do kterého může být uložen libovolný počet prvků stejného datového typu. Prvky v kolekci jsou indexovány, což znamená, že lze přistupovat k jednotlivým prvkům pomocí jejich pozice v seznamu. Kolekce také umožňuje notifikovat změny, avšak neumožnuje zapamatování si přidaných vozítek během chodu aplikace. Tato varianta bude moci být použita, pokud uživatel bude chtít vždy po spuštění aplikaci přidat vozítka opětovně. Další možností je přidávání vozítek a jejich vlastností do databáze. Kde přidaná vozítka během chodu aplikace jsou uchována po celou dobu, dokud nejsou data z databáze úmyslně odstraněna.

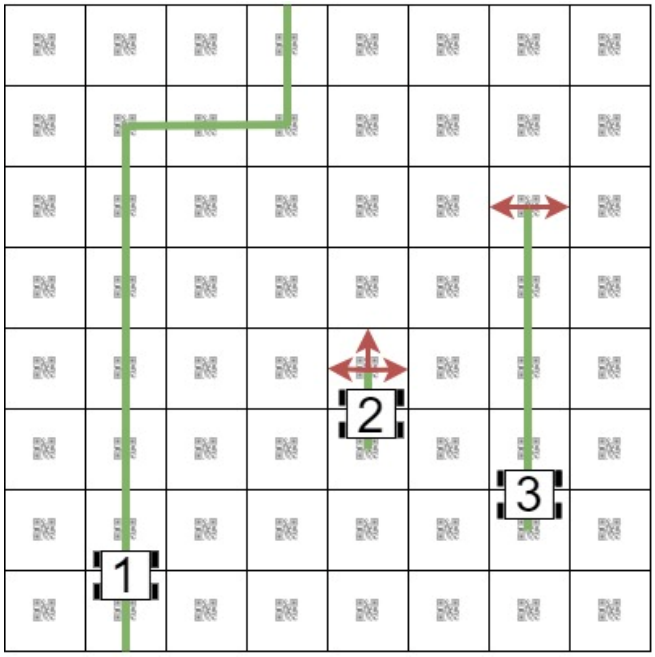
Při přidání a uložení vozítek budou zároveň ve vizualizační aplikaci zobrazeny symboly reprezentující konkrétní vozítko a jeho aktuální polohu. Budou rozlišovány dvě výchozí polohy symbolů vozítek na mapě. Při přidání reálného vozítka bude symbol vozítka zobrazen na mapě v místě odpovídající jeho aktuální reálné poloze. Naopak při přidání simulovaného vozítka bude jeho symbol umístěn do rohu mapy, kde budou tato vozítka řazena vedle sebe.

2.2.3 Plánování trasy

Při přesunu vozítka z bodu A do bodu B je nutné naplánovat optimální trasu. Optimalizace trasy spočívá v hledání neobsazených QR kódů tak, aby trasa byla co nejkratší a nejrychlejší.

Před začátkem každé trasy je zapotřebí vytyčit a zvolit nejrychlejší cestu do cílového bodu přes takové QR kódy, které nejsou obsazeny žádným jiným vozítkem. Nyní existuje několik variant, jak při navádění vozítka pokračovat:

- **První varianta** – první varianta spočívá v tom, že vozítko projíždí QR kódy dle předem vypočtené trasy do doby, dokud nedojde k obsazení některého z volných QR kódů dané trasy. Následně je nutné trasu přepočítat s ohledem na pohyb ostatních vozítek. Vozítko dostává povel vždy když nahlásí, že detekoval požadovaný QR kód.
- **Druhá varianta** – další varianta je výpočetně náročnější, ale v některých ohledech může být efektivnější. Tato varianta spočívá v tom, že trasa je přepočítávána vždy na každém QR kódů s ohledem na aktuální stav šachovnice.
- **Třetí varianta** – třetí varianta vychází z předpokladu, že vozítko bude dostávat signály pouze o změně směru na určitých QR kódech stanovené trasy. Na těchto pozicích by také docházelo k přepočítávání tras. Znamená to tedy, že vozítko na každém QR kódu své trasy nahlásí svoji pozici, ale povely k řízení bude dostávat jen na těch QR kódech, kde bude nutná změna směru jízdy.



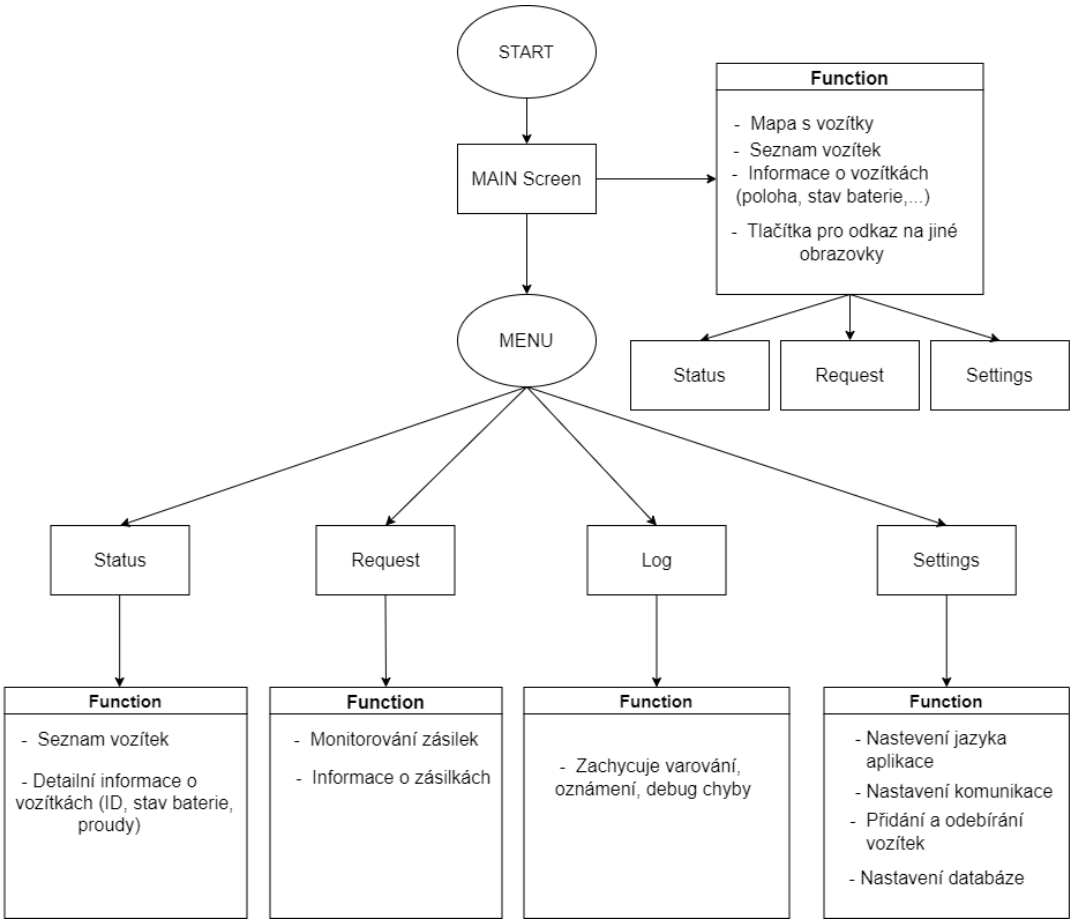
Obrázek 2.3 Možnosti plánování tras

Pokud vozítko dorazí do blízkosti cílového bodu a je cílová pozice obsazena, je nuceno vyčkávat na vedlejší pozici. Vyčkávací pozice musí být taková, aby jiné vozítko v dané pozici cíle mohlo bezproblémově tuto pozici opustit. Vozítka postupně čekají ve frontě v pořadí, v jakém se k cílovému QR kódu přiblížily.

U vozítek by se také mohla řešit jejich priorita podle převážených zásilek, nicméně pro realizaci tohoto řešení by bylo zapotřebí přídavného hardwaru. Proto se toto řešení v řídícím softwaru nebude realizovat. Dále do části řídícího softwaru spadá komunikace s reálným vozítkem a sběr dat. Toto je blíže rozebráno v kapitole 3.

2.3 Vizualizační SW

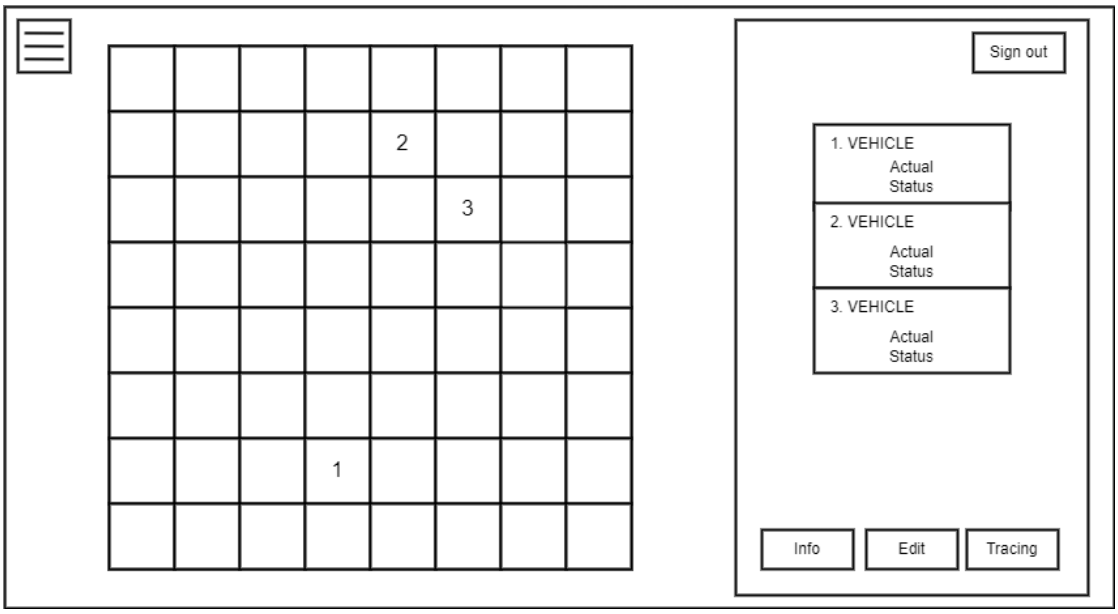
Vizualizační software bude v dané úloze sloužit jako mobilní či desktopová aplikace pro monitorování robotických vozítek a převážených zásilek. Celý vizualizační software bude obsahovat několik obrazovek. Vývojový diagram reprezentující propojení jednotlivých obrazovek je na obrázku 2.4.



Obrázek 2.4 Blokové schéma – funkce jednotlivých obrazovek vizualizace

Výchozí obrazovkou bude hlavní („Home Page“) obrazovka, která je na obrázku č. 2.5. Základem hlavní obrazovky bude šachovnicová mapa s pohybujícími se vozítky. Mapa bude vybavena dvěma vstupními a třemi výstupními pozicemi pro manipulaci se zásilkami. Také zde budou označena místa, kde mají vozítka dokovací stanici a údržbu. Pokud se změní stav vozítka, tato změna bude signalizována změnou barvy vozítka na mapě. Změnou stavu vozítka se rozumí, zda je robotické vozítko volné nebo obsazené. Pokud vozítko bude volné, bude mít zelenou barvu. Modrou barvou bude označeno vozítko, které bude obsazené. Pokud vozítko bude v errorovém stavu bude červené.

Dále bude v pravé části tabulka s vozítky a jejich základním statusem v podobě stavu (aktivní/neaktivní), informací o nákladu či stavu baterie. V dolní části hlavní obrazovky se bude nacházet několik tlačítek. Jedním z tlačítek bude tlačítko Edit. Toto tlačítko bude odkazovat na stránku, kde bude možné přidat či odebrat vozítka. Dále zde bude tlačítko Info, které bude odkazovat na stránku kde budou blíže specifikovány vlastnosti vozítek. Poslední tlačítko bude sloužit pro monitorování čekajících zásilek na vstupních pozicích pro přepravu.



Obrázek 2.5 Návrh obrazovky – Home Page

V levém horním rohu hlavní obrazovky bude navigační tlačítko v podobě rozbalovacího Menu. Toto rozbalovací Menu bude sloužit pro přesun mezi vedlejšími obrazovkami a hlavní obrazovkou. Seznam a funkce jednotlivých vedlejších obrazovek jsou následující:

- **Status** – na této obrazovce budou uživatelé schopni zobrazit kompletní seznam všech používaných vozítek v aplikaci. Každé jednotlivé vozítko bude prezentováno s podrobnými informacemi o jeho aktuálním stavu. Mezi tyto informace bude patřit například stav baterie, který poskytne uživateli přehled o dostupném energetickém zdroji vozítka. Kromě toho bude možné zjistit, zda

je vozidlo v daný okamžik aktivní nebo neaktivní. Tato informace umožní uživatelům rychle identifikovat vozidla, která jsou momentálně v provozu, a tak lépe plánovat a koordinovat jejich použití. Celkově tedy obrazovka Status poskytne uživatelům ucelený přehled o aktuálním stavu všech používaných vozítek v aplikaci, což jim usnadní efektivní správu a monitorování.

- **Request** – na obrazovce Request bude uživatel schopen prohlížet seznam obou vstupních pozic, které obsahují čekající zásilky. Zásilky budou zobrazeny ve frontě typu FIFO (First In, First Out), což znamená, že zásilky budou odbavovány podle pořadí, v jakém byly přijaty. Každá zásilka v seznamu bude identifikována svým jedinečným identifikačním číslem a bude obsahovat informace o cílové pozici a době, po kterou se nachází ve frontě. Jakmile bude zásilka odbavena a odebrána ze seznamu, zmizí ze seznamu čekajících zásilek. Tato obrazovka umožní uživatelům sledovat a spravovat příchozí zásilky a zajistí, že budou odbaveny v pořadí, v jakém byly přijaty. Tímto způsobem bude uživatelům poskytnuta efektivní a přehledná správa čekajících zásilek v aplikaci.
- **Log** – tato obrazovka bude sloužit pouze k informativnímu zobrazení, které bude obsahovat důležité informace o různých oznámeních spojených s provozem aplikace. Zde uživatel bude moci sledovat různé události a upozornění týkajících se vozítek. Varování se bude týkat například nenalezení QR kódu vozítkem**.** Jendou z doplňkových informací bude čas, kdy se událost stala. To umožní uživatelům sledovat chronologii událostí. Tímto způsobem bude uživateli umožněno sledovat chod aplikace a reagovat tak na významné události a tím zajistit plynulý chod aplikace.
- **Settings** – v této obrazovce uživatel bude mít možnost provádět různá nastavení. První možností nastavení bude výběr jazyka, kde uživatel bude moci vybírat mezi českým a anglickým jazykem pro rozhraní aplikace. Dále bude moci uživatel upravovat nastavení komunikace a spravovat seznam vozítek, které budou k dispozici v aplikaci. Mezi další možnosti bude patřit přidávání či odebírání vozítek z aplikace. Kromě toho se na obrazovce bude nacházet možnost připojen k databázi. Připojení k databázi bude však podmíněno autentizací uživatele, aby byla zaručena bezpečnost dat v aplikaci. Pro odhlášení uživatele bude sloužit tlačítko Sign out, které se bude nacházet na hlavní obrazovce aplikace.

3. KOMUNIKACE A SBĚR DAT

V této kapitole jsou popsány vlastnosti a použití jednotlivých komunikačních protokolů s cílem zvolit protokol vhodný pro přenos dat mezi reálným vozítkem a řídícím softwarem v dané aplikaci. V první části kapitoly jsou podrobně rozebrány transportní protokoly, jak pomocí těchto protokolů probíhá komunikace a také jsou zmíněny jejich hlavní výhody a nevýhody. Dále jsou zde popsány komunikační protokoly Modbus a MQTT a bezdrátová komunikace Wi-Fi. Závěr této kapitoly pojednává o návrhu konkrétního komunikačního protokolu, s ohledem na vlastnosti, které jsou nezbytné pro konkrétní aplikaci.

3.1 Transportní protokoly

3.1.1 Vlastnosti

Transportní protokoly se nacházejí na transportní vrstvě ISO/OSI (International Organization for Standardization/Open Systems Interconnection) modelu. Tato vrstva umožňuje přenos dat mezi aplikační vrstvou a vrstvou síťovou. Jelikož síťová vrstva umožňuje přenos dat bez záruky ztráty či poškození paketu, je cílem transportních protokolů zajistit správnost a kvalitu posílaných dat dle požadavků aplikační vrstvy. Požadavky na data se mohou u různých aplikací lišit tím, že v některých aplikacích je důležitá správnost dat a jinde například rychlost přenosu dat. [1] Dále transportní vrstva umožňuje následující služby:

- **Ustavení spojení a multiplexing** – Pro přenos dat je nezbytné zahájit spojení mezi komunikujícími stranami, kde si obě strany musí nastavit parametry komunikačního kanálu tzv. třícestný handshaking. Třícestný handshaking definuje tři kroky (datagramy) pro nastavení spolehlivé komunikace. Po nastavení potřebných parametrů může být zahájen vlastní přenos dat [2]. Zahájit spojení můžou obě komunikující strany i s více stranami naráz v jeden okamžik. Poslané pakety jsou pak multiplexovány do jednoho datového toku a při přijmu zpětně demultiplexovány. [1]
- **Pomalý start a kontrola zahlcení** – Při zahájení odesílání dat, je rychlost přenosu nastavena na nízkou, aby nedocházelo k přehlcení používané sítě. Pokud síť není zahlcena, zvýší se rychlost odesílání dat, aby byla síť využívána co nejefektivněji. Vyšší rychlost odesílání je využívána po takovou dobu, dokud není detekováno přehlcení sítě. Při detekci přehlcení je rychlost odesílání opět snížena, nežli se síť znovu uvolní. [1]
- **Mechanismy řízení toku dat** – Mechanismy pojednávají také o zahlcení, ale ze strany příjemce. Pokud je příjemce zahlcen velkým objemem dat a není

schopen tyto data zpracovat, dochází ke ztrátě paketů a tím k zahlcení sítě. Odesílatel na to zareaguje snížením rychlosti odesílání dat. [1]

- **Služby pro zajištění důvěryhodnosti přenosu** – Pokud dojde při odesílání dat po síti k poškození dat kvůli defektům sítě, tato služba zajistí jejich následnou nápravu. Všechna odesílána data v podobě paketů jsou číslována, aby příjemce mohl rozpoznat, zda došlo k defektům sítě. Jestliže je zjištěn defekt, pak odesílatel zajistí nápravu poškozených dat. [1]

Nejčastější dělení transportních protokolů je podle služeb, zda jsou služby spojově orientované (TCP protokol) anebo nespojově orientované (UDP protokol) [1].

3.1.2 TCP

Transmission Control Protocol (TCP) dále již jen TCP je transportní protokol, který poskytuje spojově orientovanou službu. Tato služba definuje komunikaci pomocí spojení mezi odesílatelem a příjemcem, kde komunikace probíhá ve třech krocích. Prvním krokem, pro zahájení komunikace, je navázat spojení. Spojení se navazuje odesíláním datagramu s příznaky SYN (Synchronize) a ACK (Acknowledgement) a musí být udržováno po celou dobu komunikace. V druhém kroku už se mohou být přenášena data, pokud jsou obě strany dohodnuty na parametrech jejich spojení. Po přenosu dat probíhá poslední krok komunikace, a to je přerušení a uvolnění spojení mezi odesílatelem a příjemcem. Pro ukončení komunikace slouží příznaky FIN (Finish) a ACK. [3][4]

Jedná se o velmi rozšířený protokol, který zaručuje vysokou spolehlivost. Je vhodný například pro odesílání e-mailů a prohlížení webových stránek. Bezpečný a spolehlivý přenos je zaručen potvrzováním zpráv, kterým se ověřuje, zda byla přenesena všechna data. Velkou nevýhodou TCP protokolu je, že je velice náročný pro síť a jeho hlavička je obsáhlá. Na následujícím obrázku 3.1 je hlavička TCP protokolu, která obsahuje záhlaví a data. [3][4]

Obrázek 3.1 Struktura TCP [4]

Záhlaví se skládá z následujících polí:

- **Zdrojový port** – je 16bitové pole, které obsahuje adresu portu zdrojového uzlu (odesílatele). Může dosahovat hodnot v rozsahu 0 až 65535. Při navazování spojení je na odesílacím zařízením zdrojový port přidělován náhodně, aby nedošlo ke konfliktu s ostatními spojeními. [5][6][7]
- **Cílový port** – je 16bitové pole, které obsahuje adresu portu cílového uzlu (příjemce). Tento port je důležitý pro směrování dat na správné koncové zařízení. [5][6][7]
- **Pořadové (sekvenční) číslo** – skládá se z 32bitového pole a udává pořadí prvního bytu odeslaného paketu. Podle tohoto čísla se určuje správnost pořadí odeslaných datových paketů. [5][6][7]
- **Potvrzovací číslo** (ACK) – jeho délka je 32 bitů a slouží k potvrzení přijetí předchozího poslaného byte, který byl odeslán. Příjemce tím informuje odesílatele o úspěšném příjmu dat. [5][6][7]
- **Offset dat** – tento offset je 4bitový a udává délku hlavičky TCP pomocí 4 bytových slov v hlavičce. Minimální délka hlavičky je 20 bytů a maximální délka je 60 bytů. Hodnota tohoto offset pole nabývá hodnot od 5 do 15. Jelikož z minima vyplývá (5 x 4 = 20) a z maxima (15 x 4 = 60). [5][6][7]
- **Rezervováno** – Toto pole je vyhrazeno pro pozdější využití a je to 6bitové číslo. [5][6][7]
- **Příznaky** – tyto příznaky slouží jako signalizující bity. Může být až 6 příznaků po 1bitových polích. Prvním příznakem je URG (Urgent pointer). Pokud je tento příznak nastaven na 1 je platný a signalizuje, že v TCP segmentu se nacházejí urgentní data. Druhým příznakem je ACK a jeho funkce spočívá v potvrzování přijatých dat. Další příznak je PSH (Push) a jeho hlavní funkcí je urychlit přenos dat, Pokud je nastaven na 1 data se neukládají do vyrovnávací paměti a přenos se tím urychlí. Čtvrtý příznak je RST (Reset) a resetuje spojení mezi odesílatelem a příjemcem. Dalším příznakem je SYN a umožňuje synchronizaci pořadových čísel. Používá se během inicializace komunikace (třícestný handshaking). Posledním příznakem je FIN a také je jako poslední v pořadí, jelikož ukončuje spojení mezi odesílatelem a příjemcem. [5][6][7]
- **Velikost okna** – je 16bitové pole, které indikuje kolik dat může být odesláno a uloženo do vyrovnávací paměti, dokud odesílatel neobdrží od příjemce potvrzení příjmu zpráv. Lze tím řídit tok dat a zabránit přehlcení sítě. [5][6][7][8]
- **Kontrolní součet** – je v hlavičce TCP představen jako 16bitové pole, které umožňuje příjemci zkontrolovat, zda nedošlo k poškození zprávy během jejího přenosu. Tím lze zvýšit spolehlivost přenosu dat po síti. Před odesíláním zprávy je prováděn kontrolní součet dat, který je uložen do hlavičky TCP. Příjemce po přijetí této zprávy pak provádí porovnání kontrolního součtu

uloženého v hlavičce se součtem, který vypočetl z přijaté zprávy. Pokud se oba součty shodují, zpráva nebyla během přenosu poškozena. [4][5][6][7]

- **Urgentní ukazatel** – tento ukazatel zabírá 16bitové pole v TCP hlavičce a odkazuje na místo, kde se nacházejí naléhavá data, které musí být co nejdříve přijata. Platí pouze, pokud je nastaven příznak URG. [5][6][7]
- **Výplň** – má funkci doplnění TCP hlavičky, aby její velikost byla násobkem 32 bitů. [5][6][7]

3.1.3 UDP

User Datagram Protocol (UDP) dále již jen UDP je transportní protokol, který poskytuje nespojově orientovanou službu. Tato služba definuje přenos dat od odesílatele k příjemci bez vytvoření spojení a zaručení kvality služby. Nevyužívá se zde potvrzování zpráv ani se neřídí tok dat. Je vhodný pro aplikace, které potřebují komunikovat v reálném čase a přenášet data co nejrychleji. Také je vhodný při komunikaci s více klienty. Ztráty dat jsou pak řešeny opakovanou výzvou či snížením kvality. [1][9]

Na obrázku 3.2 je patrné, že stejně jako u TCP protokolu, hlavička UDP obsahuje zdrojový a cílový port. Zdrojový port je však volitelný, jelikož nedochází k potvrzování přijaté zprávy. Pokud se zdrojový port nepoužívá, pak je nastaven na 0. Dalším polem UDP hlavičky je délka a označuje délku paketu a dat v bytech. Tento parametr je již povinný a umožňuje příjemce informovat o velkosti dat, které má očekávat. Dalším nepovinným parametrem je kontrolní součet. Funkce kontrolního součtu je již popsána v podkapitole 3.1.2. [1][9]

+	bity 0 - 15	16 - 31
0	zdrojový port	cílový port
32	délka	kontrolní součet
64	data	

Obrázek 3.2 Struktura UDP [9]

V následující tabulce 3.1 jsou porovnány klíčové vlastnosti výše uvedených transportních protokolů.

Tabulka 3.1 Hlavní rozdíly transportních protokolů [10][11]

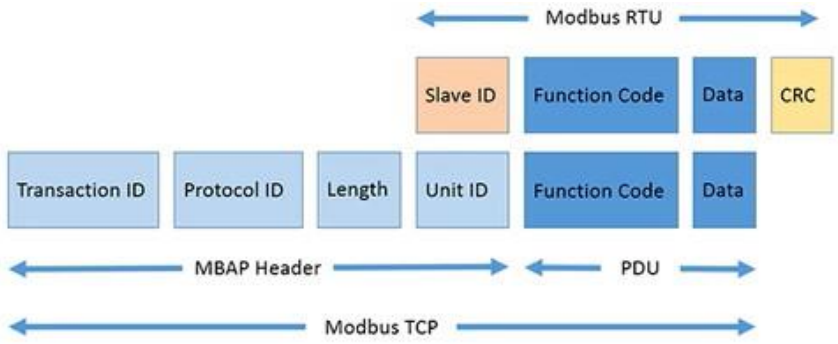
Vlastnosti	TCP	UDP
Typ připojení	Před přenosem dat je nutné navázat spojení	K zahájení/ukončení spojení není potřeba připojení
Rychlost	Pomalý	Rychlý
Přenos dat	Pokud se ztratí paket lze znovu odeslat data	Pokud se ztratí paket nelze obnovit ztracená data
Doručení	Je zaručeno	Není zaručeno
Potvrzení přijetí	Potvrzuje, data posílána v určitém pořadí	Nepotvrzuje, data přijímána v libovolném pořadí
Multicast (data poslána více příjemcům)	Neumožňuje	Umožňuje
Použití	E-mail, SMS, prohlížení webových stránek	Online přenos, videochat

3.2 Modbus

3.2.1 Úvod

Modbus byl vytvořen společností Modicon v roce 1979, ale dnes ho spravuje organizace Modbus-IDA. Je to otevřený komunikační protokol určený pro přenos dat, který se nachází v aplikační vrstvě v ISO/OSI modelu. Pracuje na principu server – klient (master – slave). [12] Existuje několik typů Modbus protokolu:

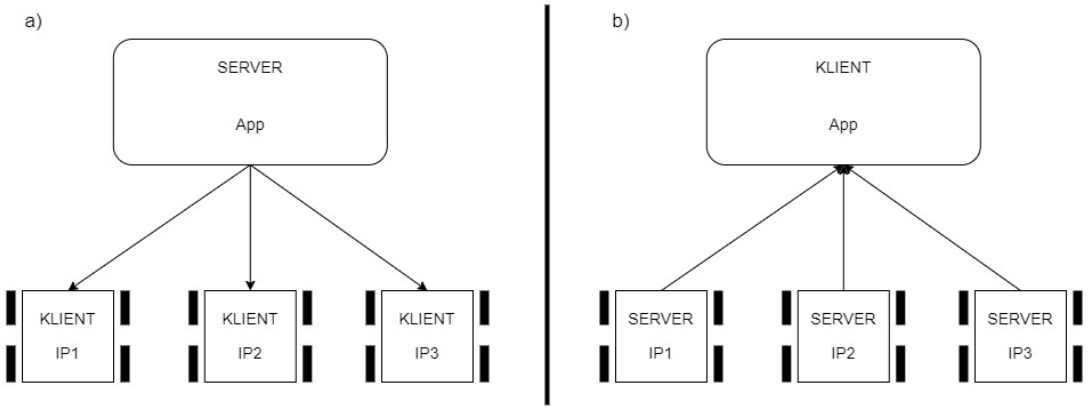
- **Modbus RTU** (Remote Terminal Unit) – komunikuje na rozhraní sériové linky RS485, RS422 a RS232 [13]. Používá binární kódování a zabezpečení dat je zaručeno tím, že provádí kontrolní součet CRC (Cyclic Redundancy Check) v rámci chybně přenesených dat či zašumění v komunikačním kanálu. [14]
- **Modbus TCP** – tento protokol navazuje na Modbus RTU a přenos dat probíhá prostřednictvím TCP/IP v síti Ethernet [15]. Formát zprávy má stejný základ (PDU – Protokol data unit) jako RTU, ale navíc obsahuje MBAP (Modbus Application Protocol) hlavičku, jak lze vidět na obrázku č. 3.3. a neobsahuje CRC. Podrobně popsaná problematika tohoto protokolu je v následující podkapitole 3.2.2.
- **Modbus ASCII** – Zprávy používají znaky ASCII (American Standard Code for Information Interchange), a proto jsou čitelnější než zprávy z Modbus RTU, avšak jsou méně efektivní. Zde se používá kontrola chyb LRC (Longitudinal Redundancy Check). Režimy RTU a ASCII jsou navzájem nekompatibilní, jelikož přenášejí rozdílný formát dat. Z tohoto důvodu nemůžou komunikovat mezi sebou dvě zařízení, které mají rozdílnou konfiguraci. [16]



Obrázek 3.3 Modbus – formát zprávy [17]

3.2.2 Modbus TCP

V případě použití komunikačního protokolu Modbus TCP do této aplikace jsou na obrázku č. 3.4 uvedeny dvě možnosti komunikačního spojení mezi aplikací a vozítky. V prvním případě za a) je server aplikace a klient jsou jednotlivá vozítka. V druhém případě za b) je naopak aplikace klient a vozítka jsou brány jako server.



Obrázek 3.4 Komunikace mezi aplikací a vozítkem

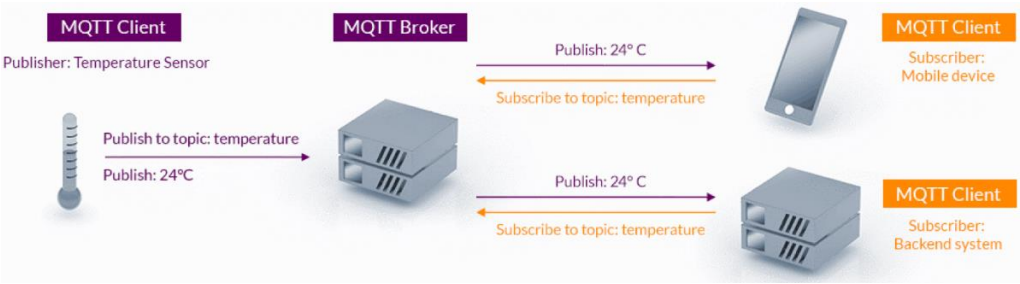
Porovnání jednotlivých variant propojení aplikace s vozítky:

- **1. varianta za a)** – v tomto případě jsou vozítka v pozici klient, tudíž má aplikace kontrolu nad komunikací. Aplikace si vybírá jednotlivé klienty, se kterými chce zrovna komunikovat. Nevýhodou je však to, že lze komunikovat vždy s jedním klientem a klienti nemůžu odesílat požadavky na komunikaci po celou dobu, jen když je o to aplikace požádá. Pro každé vozítko by musel být vytvořený port, aby se mohlo vozítko připojit, standartní port je 502.
- **2. varianta za b)** – při použití této varianty jsou vozítka servery, tudíž při komunikaci s aplikací nemá aplikace kontrolu nad tím, s kým bude komunikovat. Výhodou je, že vozítka se sama dotazují aplikace, takže mohou odesílat data kdykoliv je to nutné. Pokud však nastane situace, že bude komunikovat více vozítek s aplikací najednou vzniká problém tzv.

„úzkého hrdla“. Při vzniku úzkého hrdla je omezen tok dat do aplikace a tím snížena efektivita a výkon celé komunikace. Nicméně servery mohou komunikovat pouze s aplikací nikoli s ostatními vozítky.

3.3 MQTT

MQTT je zkratka pro Message Queuing Telemetry Trans-port. Protokol MQTT využívá pro přenos dat TCP. Je ve většině případů používán pro přenos dat v IoT (Internet of Things) aplikacích. Komunikace je založena na přenosu dat mezi Publisher a Subscriber prostřednictvím centrálního bodu Broker. Architektura MQTT je na obrázku 3.5 s konkrétním případem přenosu informace o teplotě. [18]



Obrázek 3.5 Architektura MQTT[19]

Popis jednotlivých komunikačních uzlů je následující:

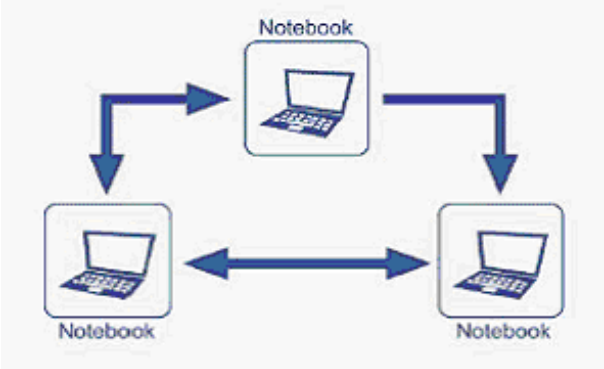
- **MQTT Client** (Publisher nebo Subscriber) – jakékoliv zařízení může být zároveň Subscriber (odběratelské zařízení) i Publisher (publikující zařízení), tedy jedno konkrétní zařízení může odesílat zprávy Brokeru a také je od Brokeru přijímat. Jedno odběratelské zařízení může přijímat zprávy od více publikujících zařízení vzhledem k jejich „předplatnému“. MQTT Client může být například senzor měřící konkrétní veličinu nebo chytrá aplikace, která tuto měřenou veličinu zpracovává. [19]
- **MQTT Broker** – je sprostředkovatel mezi publikujícím zařízením a odběratelským zařízením. Přijímá data od publikujícího zařízení (Publisher), která následně filtruje podle „předplatného“, které mají odběratelé (Subcriber). Tato data jsou následně odesílá vhodným odběratelům. Broker může obsluhovat velké množství připojených publikujících zařízení nebo odběratelů. [19]

Výhoda MQTT protokolu je, že umožnuje velkému množství připojených zařízení komunikovat mezi sebou. Avšak tím, že je připojeno více zařízení, stává se MQTT Broker úzkým hrdlem, tedy tento protokol nepodporuje horizontální škálování (vhodné rozšíření výpočetní kapacity). Také s touto nevýhodou souvisí fakt, že MQTT protokol není schopen pracovat v reálném čase. Tedy není vhodný pro aplikace, které vyžadují vysokou časovou přesnost a potřebují data zpracovávat bez zpoždění. [20]

3.4 Wi-Fi

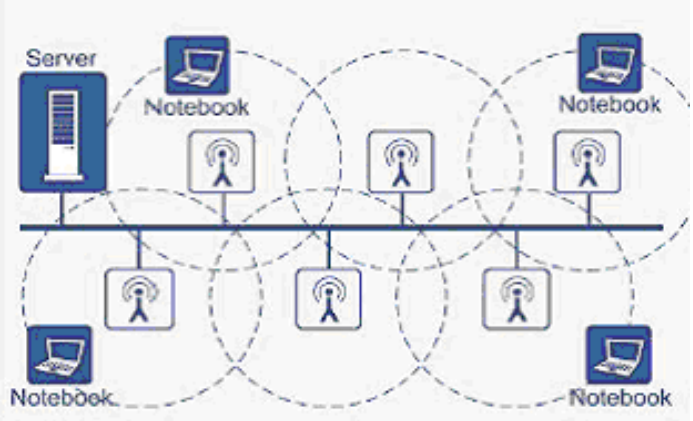
Wi-Fi síť je zkratka pro Wireless Fidelity a jedná se o nejrozšířenější veřejný bezdrátový standart IEEE 802.11. Tato bezdrátová síť je definovaná v nelicencovaném kmitočtovém pásmu 2,4 GHz a 5 GHz. [21][22] Existují dva hlavní typy struktur bezdrátové sítě, jak probíhá komunikace mezi jednotlivými stanicemi:

- **Ad-hoc síť**– tyto sítě jsou navrženy tak, aby stanice mezi sebou byly schopné komunikovat přímo. Pro společnou komunikaci nepotřebují žádné prostředníky, avšak komunikující stanice musí být v blízkosti svého rádiového přenosu. Pokud by tedy komunikovaly stanice mezi sebou ve větší vzdálenosti, tak by tato varianta nemohla být použita. Tato struktura bezdrátové sítě se většinou používá pro nárazové přenosy dat mezi blízkými komunikačními zařízeními. V dnešní době většina uživatelů tuto variantu nepoužívá a volí jednodušší cestu, z pohledu konfigurace sítě a využívá pro přenos dat přes přenosné médium typu USB disk a další. [23]



Obrázek 3.6 Struktura bezdrátové sítě Ad-hoc [24]

- **Infrastrukturní síť** – tyto sítě mají předem stanovenou infrastrukturu, kde spojovací člen se nazývá přístupový bod (Access point – AP). Přístupový bod plní funkci spojení mezi drátovou a bezdrátovou sítí, a je považován za tzv. most. Tento přístupový bod umožnuje komunikovat s více stanicemi najednou. Pokud tedy chce stanice komunikovat s jinou stanicí musí se nejprve připojit do sítě s přístupovým bodem. Komunikace je tedy dvojfázová, nejprve stanice data odesílá přístupovému bodu a poté přístupový bod odesílá data cílové stanici. Komunikovat v této síti může každá stanice, která se připojí k přístupovému bodu a je v blízkosti jeho pokrytí. Výhodou přístupového bodu je to, že pokud se stanice nachází v úsporném režimu, přístupový bod data uchová a odešle je až tehdy, když se z tohoto režimu stanice probudí. Tento postup má za následek šetření baterie. Připojení do této sítě je mnohem jednodušší než do sítě Ad-hoc, jelikož zde stačí pouze vložit síťovou kartu do zařízení a připojit se do konkrétní sítě. [23]



Obrázek 3.7 Struktura bezdrátové sítě – Infrastrukturní síť [24]

3.5 Vybraný protokol

3.5.1 Požadavky

Při výběru vhodného komunikačního protokolu zadavatel vyčlenil výše uvedené protokoly a přenosové médium, které by mohly být případně použity pro komunikaci. Dále pro tuto aplikaci upřesnil požadavky pro přenos dat mezi robotickým vozítkem a řídícím softwarem. Komunikační protokol by měl splňoval následující požadavky:

- Bezdrátové připojení
- AP (Access point)
- Přenos v reálném čase
- Připojení více zařízení
- Broadcast
- Potvrzování zpráv

Z těchto uvedených požadavků na komunikační protokol, je možné vybrat vhodný protokol pro přenos dat v této aplikaci s konkrétními fakty v podobě výhod a nevýhod, kterými již zmiňované protokoly disponují. Protokoly jsou porovnány na základě požadavků této aplikace.

Prvním požadavkem na protokol je bezdrátový přenos dat. Tento přenos je nezbytný, jelikož se vozítka budou pohybovat po hale (místnosti) a řídící software nemusí být v blízkosti vozítek. Navíc vozítka při pohybu nemůžou mít na sobě připevněný vodič pro přenos těchto dat. Z tohoto důvodu může být z výše zmíněných protokolů vyloučen Modbus RTU a Modbus ASCII, jelikož oba mají pro přenos dat fyzické médium v podobě datového vodiče.

Dalším požadavkem je přenos dat v reálném čase. Tento požadavek je důležitý pro rychlou a bezprostřední manipulaci s vozítky s ohledem na bezpečnost. Systém si nemůže dovolit dlouhou časovou latenci. Tento požadavek by po správném nastavení splňovaly

zbylé komunikační protokoly Modbus TCP, MQTT, TCP a UDP. Protokoly Modbus TCP a MQTT mají úskalí v tom, že cyklicky posílají data a jelikož komunikace probíhá jako Master – slave. Slave zařízení by u Modbus TCP nemělo možnost vynutit si komunikaci. Při použití protokolu MQTT by toto bylo, avšak tento protokol postrádá možnost testovat činnost zařízení. Použití TCP protokolu je nevhodné, jelikož budou posílány pouze krátké pakety, a navíc je časově náročné opětovné zřízení spojení.

Třetí požadavek je připojení více komunikačních zařízení do jedné sítě, z důvodu rozsáhlosti aplikace s robotickými vozítky. V aplikaci je nutné, aby mohla s řídícím softwarem komunikovat všechna robotická vozítka. Zde by bylo možné použít MQTT a Modbus TCP, ale tyto protokoly nesplňují výše uvedený požadavek.

Následně je důležité, aby byl umožněn broadcast. To znamená, aby bylo možné poslat zprávu více zařízením najednou. Tento požadavek splňuje protokol UDP a byl také vybrán pro následnou realizaci komunikace.

3.6 Koncept komunikace

Po dohodě se zadavatelem komunikace mezi robotickým vozítkem a řídícím softwarem bude probíhat přes Wi-Fi s infrastrukturním typem sítě. Pro aplikaci bude použit UDP protokol a komunikace bude typu peer to peer mezi řídícím softwarem a vozítky. Při komunikaci budou data přenášena v podobě paketu. Vzhled tohoto paketu se bude lišit v závislosti na směru toku dat.

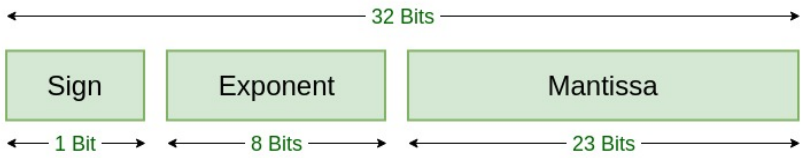
Data odesílaná vozítku budou obsahovat dvě hlavní části. První část bude definovat číslo vozítka o velikosti 2 byte a druhá část bude obsahovat pohybové příkazy, popřípadě predikovaný QR kód.

Data přijatá od vozítka budou mít obecnou strukturu, která se bude skládat ze tří hlavních částí. První část paketu bude obsahovat informace o typu zprávy a bude mít velikost 1 bytu. Druhá část paketu bude udávat číslo vozítka o velikosti 2 bytů a tato část bude datového typu word. Poslední část paketu bude zahrnovat podrobné informace, které se týkají daného typu zprávy. Bližší popis těchto částí je níže.

Některé z informací, které popisují typ zprávy budou datového typu float (4 byty). Jedná se o datový typ s pohyblivou desetinnou čárkou, který umožňuje reprezentovat reálná čísla. Tato reprezentace podléhá normě IEEE 754 a rozděluje číslo do tří složek:

- **Mantisa** – je 23bitové číslo, které představuje všechny platné číslice v převáděném čísle. Tedy všechny číslice, které mají první číslovku nenulovou společně se všemi dalšími čísli. Mantisa reprezentuje reálné či komplexní číslo.
- **Exponent** – je 8bitové číslo představující posun řádu desetinného čísla tedy exponent (mocninu) převáděného čísla. Toto číslo je na rozdíl od mantisy celočíselné.

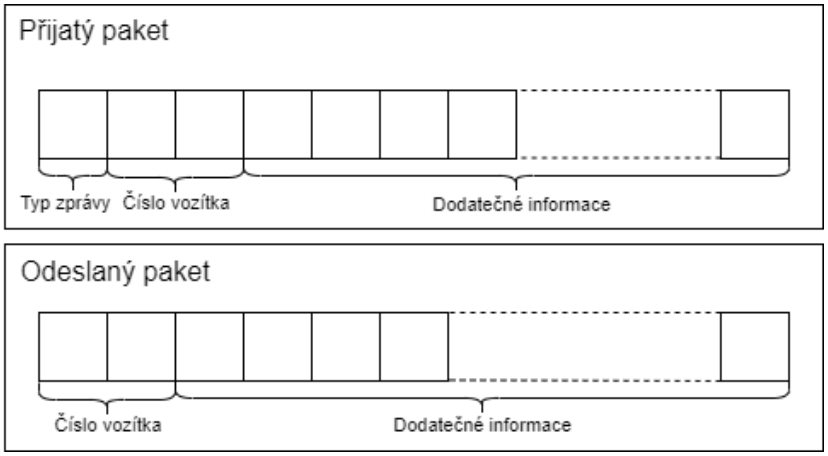
- **Znaménko** – je 1bitové číslo představující znaménko. Pokud je reálné číslo kladné je v této části reprezentováno binární hodnotou 0. Pokud se jedná o záporné číslo je binární hodnota 1. [25]



Obrázek 3.8 Reprezentace datového typu float

Nebude nutné data v paketu šifrovat, jelikož se přenášená data budou v binární soustavě a také data nebudou přenášena přes veřejnou síť. Na obrázku 3.9 je možné vidět obecnou strukturu paketů zprávy.

Pokud od vozítka nepřijde stavová zpráva do stanoveného času, bude prohlášeno za nečinné a bude přivolána obsluha.



Obrázek 3.9 Obecná struktura paketů

První část přijatého paketu bude tedy obecně rozdělovat typy jednotlivých zpráv. Budou rozlišovány následující typy zpráv:

- **0x01** – tento typ zprávy bude obsahovat informace o obecném stavu vozítka.
- **0x02** – druhá zpráva bude obsahovat informace o aktuálním nalezeném QR kódu společně s polohou vozítka vůči QR kódu.

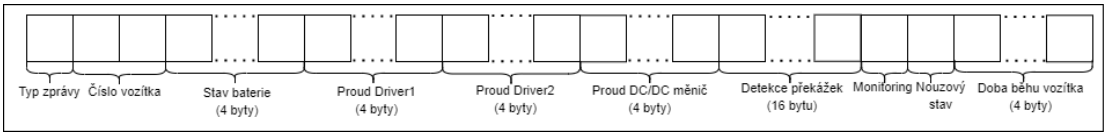
Druhá část přijatého paketu a první část odeslaného paketu bude obsahovat dva byty, které budou určovat konkrétní číslo vozítka jakému je paket zasílán nebo jaké vozítko odeslalo zprávu řídícímu systému. Příklad čísla podvozku může být následující:

- **0x0001**– paket tedy bude odeslán nebo přijímán vozítkem, jehož číslo je 1.
- **0x0010**– paket s tímto označením vozítka bude odpovídat vozítku s číslem 2.
- **0x0007** – paket s tímto označením vozítka bude odpovídat vozítku s číslem 7.
- **0x000A** – paket s tímto označením vozítka bude odpovídat vozítku číslo 10.

Poslední část obou paketů jsou dodatečné informace, kde je obsaženo tělo zprávy. Konkrétní popis jednotlivých paketů je následující.

3.6.1 Přijatá data – typ zprávy 0x01

Tato stavová zpráva bude obsahovat dodatečné informace o vozítku týkající se stavu napětí na baterii, proudů, detekce překážek, obsazenost či aktivnost vozítka, informace o nouzovém stavu a celkovou dobu běhu vozítka. Na obrázku 3.10 se nachází struktura daného typu zprávy.



Obrázek 3.10 Struktura paketu typu zprávy 0x01

Informace o stavu napětí baterie bude datového typu float, což značí 4 byty. Příklad některých variant, které mohou nastat jsou popsány níže:

- **0x00000000**– vozítko má napětí na baterii 0 V
- **0x0AD7233C** – vozítko má napětí na baterii 0,01 V
- **0x0AD7A33C** – vozítko má napětí na baterii 0,02 V
- **0x0000803F** – vozítko má napětí na baterii 1 V
- **0x00004041** – vozítko má napětí na baterii 12,00 V

Další tři informace týkající se proudů, které procházejí jednotlivými drivery a DC/DC měničem budou mít dohromady 12 bytů (4 byty pro každý proud zařízením). Bude se jednat o datový typ float a možné varianty budou následující:

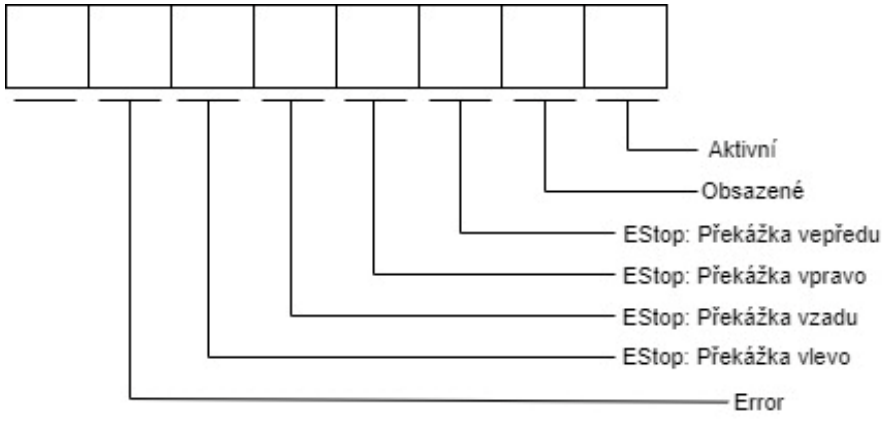
- **0x00000000** – proud procházející konkrétním zařízením je 0 A
- **0xEC51383D** – proud procházející konkrétním zařízením je 0,045 A
- **0x9A99993E** – proud procházející konkrétním zařízením je 0,3 A
- **0x0000003F** – proud procházející konkrétním zařízením je 0,5 A

Informace o detekci překážek budou typu float a jejich celková velikost bude 16 bytů. Tedy data z jednoho senzoru budou mít velikost 4 byty. Robotické vozítko obsahuje celkem čtyři senzory vzdálenosti detekující překážku. Zde jsou vypsány některé z možností vzdáleností:

- **0x0000FA44; 0xCDCC4842; 0x91793044; 0x0000AF43 –** nedetekována překážka (max detekovaná vzdálenost 2000 mm) přední senzor; překážka detekována v 50,2 mm pravý senzor; překážka detekována v 705,9 mm zadní senzor; překážka detekována v 350 mm levý senzor
- **0x331311644; 0x00004843; 0x9A99F742; 0x00007A44 –** překážka detekována v 600,3 mm přední senzor; překážka detekována v 200 mm pravý senzor; překážka detekována v 123,8 mm zadní senzor; překážka detekována v 1000 mm levý senzor

Jaký senzor detekoval překážku je zřejmé z pořadí dat v obdržené zprávě. Pořadí senzorů reprezentováno v datech bude: přední senzor, pravý senzor, zadní senzor a levý senzor.

Další byte zprávy je Monitoring. Tento byte obsahuje informace o obsazenosti, aktivnosti a nouzovém zastavení vozítka při detekci překážky budou mít velikost 1 bytu. První bit označuje, zda je vozítko aktivní či neaktivní, druhý bit označuje obsazenost vozítka v rámci převážené zásilky. Další 4 bity značí EStop zastavení při detekci překážky vždy s konkrétním číslem senzoru. Poslední bit označuje Errorový stav vozítka, kde jednotlivé příčiny vzniku Erroru označuje další byte popsaný níže. Jednotlivý bity této informace jsou na obrázku 3.11.



Obrázek 3.11 Rozdělení Monitoring na jednotlivé bity

Informace o nouzovém stavu vozítka budou mít velikost také 1 bytu a mohou nastat například tyto situace.

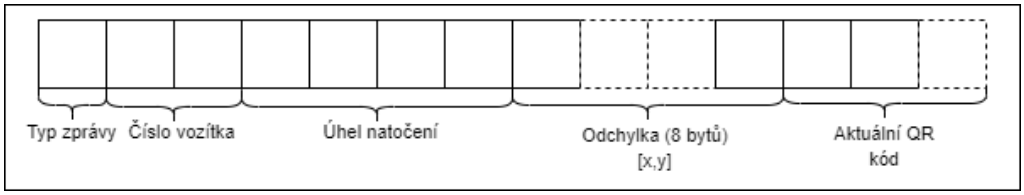
- **0** – vozítko se nenachází v nouzovém stavu
- **1** – nouzové zastavení – vybitá baterie
- **2** – nouzové zastavení – ztráta QR kódu

Poslední dodatečná informace, kterou paket bude obsahovat je doba po jakou bylo vozítko v provozu. Tato informace bude mít velikost 4 byty typu int a bude udávána v sekundách.

- **0x00000000** – vozítko bylo v provozu po dobu 0 s
- **0x00000001** – vozítko bylo v provozu po dobu 1 s
- **0x00000009** – vozítko bylo v provozu po dobu 9 s

3.6.2 Přijatá data – typ zprávy 0x02

Takto označená stavová zpráva ponese informace týkající se QR kódu, které vozítko detekovalo. Bude se jednat o informace, které budou popisovat nejprve jaký QR kód byl načten, dále jaký je úhel natočení vozítka vůči QR kódu. Tato informace bude udávaná v rozsahu $\pm 180°$. Poslední informace se bude týkat odchylky, která je mezi QR kódem a středem vozítka. Struktura dané zprávy je znázorněna na obrázku 3.12.



Obrázek 3.12 Struktura paketu typu zprávy 0x02

První informace nese data o úhlu natočení vozítka. Bude typu float, takže bude mít velikost 4 bytů. Mohou nastat například tyto varianty:

- **0x00000000** – vozítko je vůči QR kódu natočeno o 0 °
- **0x0000A0C0** – vozítko je vůči QR kódu natočeno o –5 °
- **0x0000A040** – vozítko je vůči QR kódu natočeno o 5 °
- **0x66669EC2** – vozítko je vůči QR kódu natočeno o – 79,2 °
- **0x66669E42** – vozítko je vůči QR kódu natočeno o 79,2 °
- **0x00003443** – vozítko je vůči QR kódu natočeno o 180 °
- **0xCDCC34C2** – vozítko je vůči QR kódu natočeno o – 45,2 °
- **0xCDCC3442** – vozítko je vůči QR kódu natočeno o 45,2 °

Druhá informace týkající se odchylky bude typu float a celkově bude mít 8 bytů. Z nichž první 4 byty budou odpovídat odchylce v ose x a poslední 4 byty odchylce v ose y. Odchylka bude udávána v mm a přijatá zpráva bude obsahovat například tyto odchylky:

- **0x66668E42; 0xCDCCB0C1 –** střed vozítka je odchýlen od středu QR kódu v ose x o 71,2 mm; střed vozítka je odchýlen od středu QR kódu v ose y o 22,1 mm
- **0x00000000; 0xCDCC2441 –** střed vozítka je odchýlen od středu QR kódu v ose x o 0 mm; střed vozítka je odchýlen od středu QR kódu v ose y o 10,3 mm
- **0x0000C0C1; 0x9A9991C0 –** střed vozítka je odchýlen od středu QR kódu v ose x o – 24 mm; střed vozítka je odchýlen od středu QR kódu v ose y o – 4,55 mm

Jaká odchylka přišla v dané ose je zřejmé z pořadí dat v obdržené zprávě. Pořadí os reprezentováno v datech bude: osa x, osa y.

Poslední informace týkající se QR kódu, na kterém se vozítko nachází nebude mít pevně danou velikost. V dané aplikaci je použita informace o QR kódu o velikosti 2 byte, ale data uložená v QR kódu mohou mít různou velikost, tedy při změně dat v QR kódu by aplikace nemohla být flexibilní. V dané aplikaci může být informace následující:

- **0x0000** – Žádná detekce QR kódu
- **0x0101** – vozítko se nachází na QR kódu [1, 1]
- **0x0102** – vozítko se nachází na QR kódu [1, 2]

- **0x0103** – vozítko se nachází na QR kódu [1, 3]
- **0x0104** – vozítko se nachází na QR kódu [1, 4]
- **0x0105** – vozítko se nachází na QR kódu [1, 5]
- **0x0106** – vozítko se nachází na QR kódu [1, 6]
- **0x0107** – vozítko se nachází na QR kódu [1, 7]
- **0x0108** – vozítko se nachází na QR kódu [1, 8]
- **0x0201** – vozítko se nachází na QR kódu [2, 1]

První byte bude reprezentovat sloupec maticového rozložení QR kódu a druhý byte bude reprezentovat řádek. Zde je uveden pouze výčet několika z celkových 65 možností, které mohou nastat (aplikace obsahuje 64 QR kódů a variantu nenalezení žádného z kódu).

3.6.3 Odeslaná data

Daný typ zprávy bude obsahovat informace, které se týkají pohybových příkazů, které má dané vozítko vykonat. Struktura dané zprávy je na obrázku 3.13. Příkazy bude vozítko přijímat od řídícího systému. Tyto příkazy budou mít velikost 1 bytu. Poslední byte zprávy bude představovat predikovaný QR kód, který bude sloužit pro jednu z možných variant plánování tras a řízení vozítka. Struktura zprávy je na obrázku 3.13.

Obrázek 3.13 Struktura paketu odesílané zprávy

Pohybové příkazy, které mohou přijít od nadřazeného řídícího software robotickému vozítku mohou být následující:

- **0x01** – vozítka dostane příkaz zastavit a nevykonávat žádný pohyb
- **0x02** – vozítko dostane příkaz jet dopředu
- **0x04** – vozítko dostane příkaz jet doprava
- **0x08** – vozítko dostane příkaz jet dozadu
- **0x10** – vozítko dostane příkaz jet doleva

Predikce QR kódu bude mít stejnou strukturu jako když nadřazený software přijme od vozítka informaci o aktuální poloze. Tato struktura je popsána výše v podkapitole 3.6.2.

4. NET MAUI

V následující kapitole je popsán framework, který je využíván pro tvorbu řídícího a vizualizačního softwaru této práce. V kapitole 4.1 je představení tohoto frameworku. Kapitola 4.2 pojednává o architektuře .NET MAUI. V následující kapitole 4.3 je podrobně popsána struktura MVVM (Model – View – ViewModel), která je součástí .NET MAUI a je využívána v této aplikaci. Dále jsou zde uvedeny výhody a nevýhody této struktury.

4.1 Základní popis

Pro tvorbu vizualizačního a řídícího SW bylo vybráno vývojové prostředí Visual Studio Cummunity 2022 s platformou .NET MAUI (dále již jen MAUI). MAUI je zkratka pro Multi-platform App UI a jedná se o relativně novou technologii pro tvorbu aplikací. Tento framework vznikl ze systému Xamarin a je následníkem Xamarin.Forms a vyvinula ho společnost Microsoft. Podobně jako Xamarin je aplikace tvořena ve dvou jazycích. Logická stránka v jazyce C# a grafická stránka v jazyce XAML. [26]

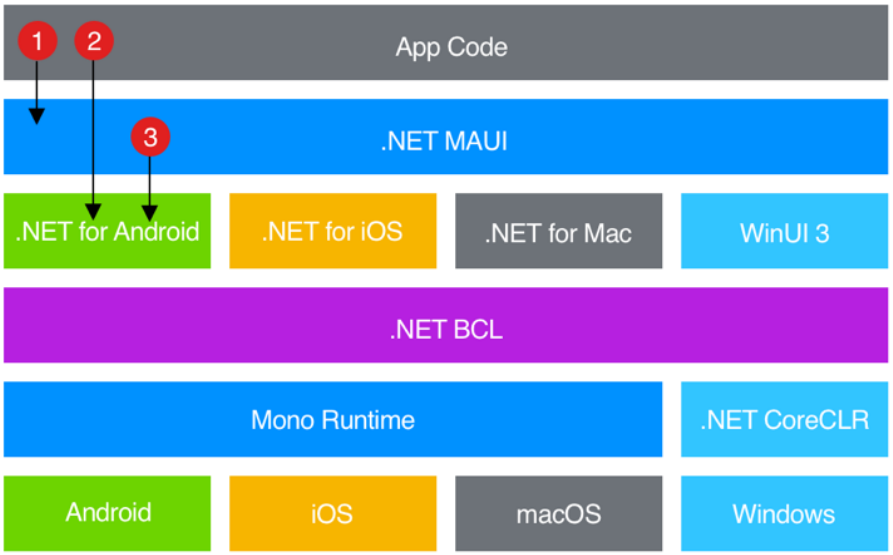
MAUI umožňuje vytvářet multiplatformní aplikace pouze v rámci jednoho kódu. V předchozích verzích .NET bylo již možné vytvářet aplikace, které jsou spustitelné na různých platformách, ale vždy bylo nutné napsat unikátní kód pro konkrétní platformu (Android, iOS atd.). MAUI má nespornou výhodu v tom, že sjednocuje rozhraní pro různé platformy do jednoho rozhraní. Stačí tedy pouze vytvořit jeden kód, který je následně spustitelný na platformách jako je Android, iOS, Windows a macOS. Je to umožněno tím, že MAUI při kompilaci SW v jazyce C#, vytvoří IL (Intermediate language), což je kód příslušného jazyka daného operačního systému. [26]

4.2 Architektura

V sekci App Code se vytváří kód dané aplikace, která běží na rozhraní .NET MAUI. Na obrázku 4.1 má první vrstva přímý přístup ke konkrétním platformám (.NET for Android, .NET for iOS…) nebo k nim může přistupovat přes sdílené rozraní .NET MAUI. Pro každou platformu je tedy různě definováno uživatelské rozhraní, proto se zde nachází .NET BCL (Base Class Library). Tato knihovna vytváří pro jednotlivé platformy společnou business logiku a má funkci tzv. IL (Intermediate Language). [26][27]

Mono Runtime je překladač, který převede kód v IL jazyce do nativního kódu konkrétního operačního systému. Tedy IL slouží jako zprostředkovatel mezi C# jazykem a nativním kódem dané platformy. Mono Runtime překladač využívají operační systémy Android, iOS a MacOS. U operačního systému Windows není zapotřebí Mono Runtime, jelikož jazyk, který se využívá u .NET BCL je stejný jako u operačního systému Windows. Tedy aplikace pro Windows používají .NET CoreCLR (Core Common Language Runtime). Pro tvorbu nativních komponent je pro operační systém MacOS

použit Mac Catalyst a u Windows se využívá knihovna WinUI 3 (Windows UI Library). Aplikace pro Android a iOS se kompilují obdobným způsobem jako při kompilaci v systému Xamarin. Základní rozdíly mezi Xamarin a MAUI jsou uvedeny v tabulce 4.1. [26][27]



Obrázek 4.1 Architektura .NET MAUI [26]

Tabulka 4.1 Základní rozdíly mezi Xamarin.Forms a .NET MAUI [28][29]

Rozdíly	**Xamarin**	**.NET MAUI**
Struktura projektu	Pro každou platformu samostatný projekt	Jeden projekt pro různé platformy
Podporované platformy pro Windows	UWP (Universal Windows Platform)	WinUI
Architektura	.NET Framework	.NET 6,7,8
Uchování obrázků	Rozlišení obrázků specifické pro každou platformu	Obrázky ve formátu svg jsou společné pro všechny platformy
Grafické rozhraní API	Žádná grafika k dispozici	Grafické režimy: paint, winding, blend
	Neumožňuje vlastní kreslení	Kreslení grafiky je možné
Více oken	Nepodporuje	Lze zároveň otevřít více oken na Androidu, iOS, ...
Hot Reload XAML (= Hot reload umožňuje zobrazit výsledek změn při běhu aplikace.	Nepodporuje	Podporuje

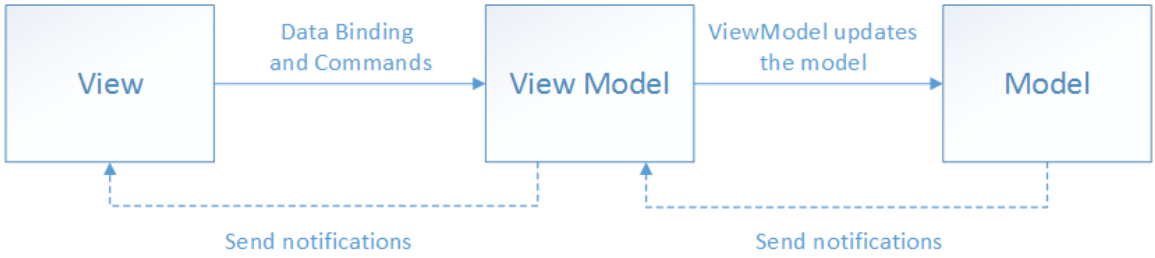
4.2.1 Konkrétní příklad

Pro vytvoření tlačítka .NET MAUI zprostředkovává zobrazení tlačítka na .NET platformách. Pro každou platformu je toto zobrazení specifické a tlačítka jsou přizpůsobena vzhledem k operačnímu systému. Následuje .NET BCL, který obsahuje jediný popis „Button“ shodný pro všechny operační systémy. Následně je tento „Button“ pomocí Mono Runtimu nebo .NET CoreCLR spuštěn a vykonán na daném operačním systému. Operační systém poté tlačítko zobrazí pro interakci s uživatelem.

4.3 MVVM

Při upravování aplikací v .NET MAUI dochází k problémům, pokud je aplikace příliš rozsáhla. Tyto problémy mohou vznikat při propojování uživatelského rozhraní (XAML) s vytvářením funkcionality tohoto rozhraní. Tyto problémy řeší MVVM. [30]

MVVM je zkratka pro Model – View – ViewModel a odděluje business logiku od uživatelského rozhraní. Použitím MVVM se zjednodušuje údržba, vývoj aplikace a je vhodným nástrojem pro testování vytvářené aplikace. Hlavním cílem je zlepšit spolupráci designerů s vývojáři a umožnit jim snazší spolupráci při vývoji aplikace. MVVM se skládá ze tří hlavních složek uvedených na obrázku 4.2, kde každá složka zastupuje určitou funkci. Funkce jednotlivých složek je popsána níže. [30]



Obrázek 4.2 Struktura MVVM [30]

4.3.1 View

View je prezentační vrstva této struktury a všeobecně se stará o vzhled vytvářené aplikace. Jedná se o základ struktury uživatelského rozhraní bez business logiky. Do uživatelského rozhraní patří všechny vizuální prvky jako jsou tlačítka, textová pole, ikony atd. Je vždy psán v XAML, ale má omezený code – behind (kód na pozadí) v jazyce C#. Code – behind je použit tehdy, pokud není možné v XAML nadefinovat určité chování vizualizačního prvku (animace). View je svázaný s ViewModelem vazbami. Tyto vazby umožňují View reagovat událostmi na požadavek uživatele pomocí vazeb (Binding). View nezná data, která jsou obsažena v Modelu a ani s nimi nemůže pracovat. [30][31][32]

4.3.2 ViewModel

ViewModel je spojovací vrstvou mezi View a Modelem. Na rozdíl od View je zde obsažena prezentační logika, kde se implementují příkazy, načítání či ukládání dat. Těmito příkazy může být například filtrování či řazení dat. Tyto příkazy definují funkcionalitu, kterou View uživateli nabízí prostřednictvím ovládacích prvků. ViewModel předává data z Modelu do View v požadovaném formátu. Pokud se změní data v Modelu, ViewModel tyto data aktualizuje a předá je View. Proto detekci změn musí být implementovány ve ViewModelu rozhraní jako INotifyCollectionChanged a INotifyPropertyChanged. [30][31][32]

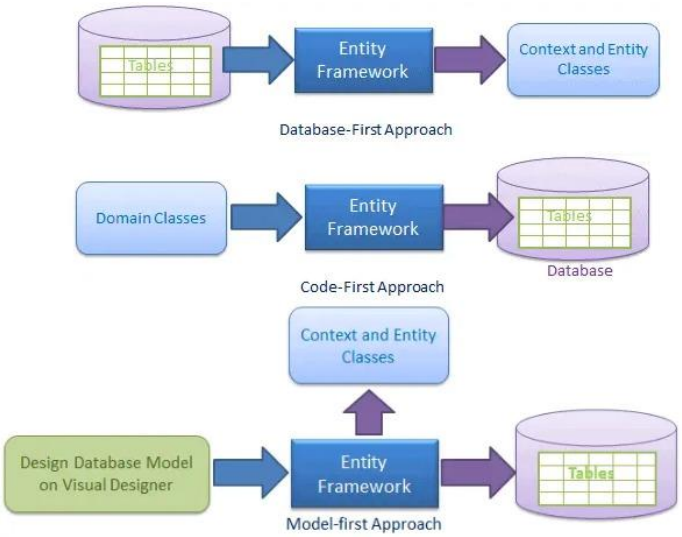
- **INotifyCollectionChanged** – je rozhraní, které patří do kolekce dat zvanou ObservableCollection, která je neustále sledována. Toto rozhraní, poskytuje aktuální informace o tom, zda došlo ke změně dat v obsahu této kolekce. Pod změnou dat si lze představit přidání nebo odebrání položek a také aktualizaci celého seznamu. Tím je zajištěna aktuálnost zobrazení, o které se stará View. [33]
- **INotifyPropertyChanged** – aby bylo dosaženo oboustranné komunikace mezi View a ViewModel musí se implementovat do tříd toto rozhraní, které dokáže vyvolat událost PropertyChange, vždy když se změní vlastnost třídy. Pokud se vlastnost třídy nezmění, není nutno tuto událost vyvolávat. Pokud, je rozhraní INotifyPropertyChanged implementováno pomocí BindableObject, je možné sledovat změny i u vlastností, které jsou navázané pomocí Bindingu. Tato informace o změně pak také jako INotifyCollectionChanged upozorní View, aby se aktualizovalo zobrazení. Těmito změnami mohou být například změny barev, či textových polí. [30][34]

4.3.3 Model

V Modelu jsou obsažena data, která jsou potřeba pro správný chod aplikace a business logika. Tyto data mohou být uložena v databázi, se kterými Model komunikuje prostřednictvím Entity Framework. Model komunikuje pouze s ViewModelem a k View nemá žádný přístup. [30]

- **Entity Framework** – jedná se o nástroj umožňující vývojářům pracovat s daty, které jsou uloženy v databázi pomocí objektů tříd. Je to rámec, který je podporován společností Microsoft a je vhodný pro .NET aplikace. Eliminuje se tím rozsáhlost kódu, která by byla zapotřebí pro přístup dat. Existují tři hlavní přístupy pro práci s daty. [35] První přístup je „Database First“, druhý přístup je „Code First“ a poslední přístup je „Model First“. Hlavní rozdíl mezi těmito přístupy je ve vytváření objektového modelu. Přístup Database First znamená, že je nejdříve navržena databáze a pomocí Entity Frameworku je vygenerován model, který se mapuje na navrženou databázi. Code First se liší

tím, že nejdříve se začíná kódem, kde je model definován třídami a následně je pomocí Entity Frameworku vygenerována databáze, která odpovídá příslušnému modelu. Poslední variantou přístupu je Model First, kde se vytváří třídy a databáze za pomoci navrženého modelu. Tento navržený model je již tvořen vazbami, a tudíž Entity Framework pomocí modelu vytváří příslušné třídy a databázi. [35] [36]



Obrázek 4.3 Hlavní přístupy pro práci s daty [35]

4.3.4 Výhody a nevýhody

Hlavní výhodou je, že použití MVVM struktury umožňuje ochránit kód aplikace. To je zaručeno tím, že ViewModel brání provádění velkých změn v kódu, který je napsán v Modelu. Další výhodou je, že lze vytvářet Unit testy pro ViewModel a Model, aniž by se do těchto testů zahrnovalo View. Dále pokud je uživatelské rozhraní implementováno v XAML nebo C#, lze ho upravovat, bez změny kódu ve ViewModelu nebo Modelu. Poslední výhodou je oddělení práce pro designéry a vývojáře, což umožňuje souběžnou práci a efektivnější vývoj. Vývojáři pracují na ViewModelu a Modelu a designéři se zaměřují na návrh uživatelské rozhraní ve View. [30][32]

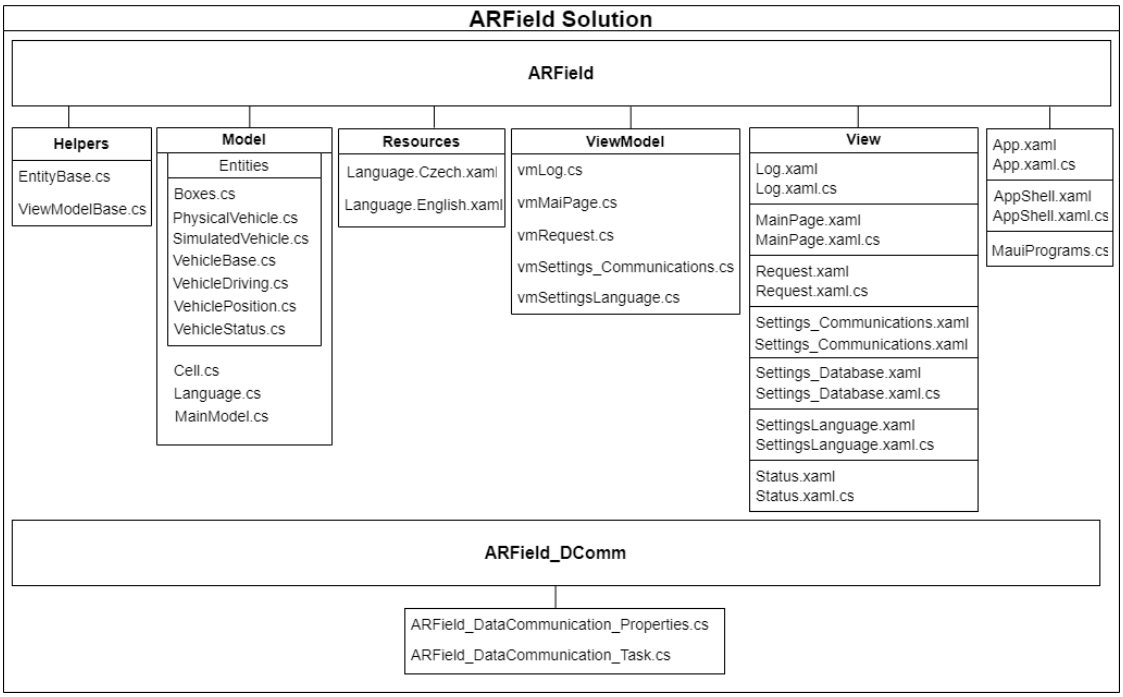
Hlavní nevýhodou je složitost. Implementace MVVM struktury se nevyplatí, když se vytváří jednoduchá aplikace. Další nevýhodou může být ladění kódu, jelikož se zde používá deklarativní datová vazba, kvůli které je obtížné zjistit, kde nastala chyba. [30][32]

5. REALIZACE VIZUALIZACE A ŘÍDÍCÍHO SOFTWARU

V této kapitole je popsána realizace vizualizačního a řídícího softwaru jejichž tvorba vychází z konceptu, který je popsán v kapitole 2. Softwary jsou vytvořeny ve vývojovém prostředí Visual Studio 2022 Community. V kapitole 5.1 je uvedena konkrétní struktura celého projektu. Následující kapitola 5.2 popisuje tvorbu vizualizačních obrazovek, které jsou navrženy a popsány v kapitole 2.3. Poslední kapitola 5.3 je zaměřena na řídící software, kde je popsána realizace komunikace mezi robotickým vozítkem a nadřazeným softwarem. Dále je zde popsáno řízení reálných i simulovaných vozítek v podobě plánování tras.

5.1 Rozdělení projektu

Níže popsané rozdělení specifikuje hlavní vytvořené části celého programu. Ostatní části zde nebyly zahrnuty, jelikož se jedná o soubory, které se zabývají barvami, fonty a styly celé aplikace. Tyto nezmíněné soubory jsou interně vytvořeny při založení projektu. Celé řešení se skládá ze dvou hlavních projektů ARField (Autonomous Robotic Field) a ARField_DComm. Na obrázku 5.1 je graficky znázorněna struktura celého projektu rozdělena na jednotlivé soubory. V podkapitolách 5.1.1 a 5.1.2 jsou blíže specifikovány tyto jednotlivé soubory daných projektů.



Obrázek 5.1 Struktura projektu

5.1.1 ARFiled

V tomto projektu je zahrnuta veškerá logická část aplikace, která je realizována pomocí architektury MVVM (Model – View – ViewModel). Tato architektura umožňuje oddělení zobrazení, logiky aplikace a dat, což usnadňuje tvorbu a údržbu aplikace. Projekt ARField je zaměřen na správu uživatelského rozhraní a interakce s uživatelem. Struktura projektu ARField zahrnuje několik hlavních složek, přičemž první z nich je Helpers. Tato složka obsahuje pomocné třídy, které slouží k usnadnění některých operací v rámci aplikace. Mezi tyto třídy patří:

- **EntityBase.cs –** tato třída slouží jako základ pro ostatní třídy (Modely a Entity), jelikož umožňuje sledování změn v aplikaci.
- **ViewModelBase.cs –** tato třída slouží jako základ pro všechny ViewModely v aplikaci a stejně jako EntityBase umožňuje sledování změn.

Další hlavní složkou tohoto projektu je Model. Tato složka obsahuje soubory, které vytváří datový model aplikace. Níže jsou popsány tyto soubory, které jsou rozděleny na Entity a Modely.

- **Cell.cs –** jedná se o třídu, která popisuje vlastnosti polí na mapě. Mezi tyto vlastnosti patří řádek a sloupec, ve kterém se pole nachází a také definuje minimální a maximální souřadnice daného pole. Tato třída slouží pro následnou detekci pole, na které klikl uživatel.
- **Language.cs** – zde jsou definovány vlastnosti, které popisují konkrétní jazyk aplikace, jako je například název jazyka, kultura apod.
- **MainModel.cs –** je třída, která představuje hlavní datový model aplikace. Obsahuje logiku a data potřebná pro správu vozítek, front zásilek a další informace nutné pro běh aplikace.
- **Boxes.cs –** v této entitě jsou vytvořeny základní vlastnosti zásilek. Mezi tyto vlastnosti patří například číslo zásilky nebo destinace, ve které má být zásilka odbavena.
- **PhysicalVehicle.cs** – tato třída představuje reálné fyzické vozítko, která obsahuje vlastnosti pro nastavení komunikace a správu stavu vozítka.
- **SimulatedVehicle.cs** – tato třída pouze slouží pro rozlišení simulovaného vozítka od reálného.
- Dále aplikace obsahuje Modely VehicleBase.cs, VehicleDriving.cs, VehiclePosition.cs, VehicleStatus.cs, které jsou blíže popsány v podkapitole 5.3.1.

Třetí hlavní složkou projektu ARField jsou Resourses, kde je zmíněna pouze ta část, která obsahuje Languages.

- **Language.Czech.xaml –** tento XML soubor obsahuje definice textových řetězců pro různé části uživatelského rozhraní aplikace ARField, které mají být přeloženy do českého jazyka. K jednotlivým textovým řetězcům se přistupuje přes unikátní klíče.

- **Language.English.xaml –** tento soubor má obdobnou funkci jako Language.Czech.xaml, avšak obsahuje textové řetězce, které mají být přeloženy do anglického jazyka.

Jelikož je v projektu dodržována MVVM struktura, jsou zde soubory dále rozděleny do ViewModels a Views. ViewModel propojuje Model s View a obsahuje tyto soubory:

- **vmLog.cs –** tento ViewModel slouží pro logování v aplikaci. Obsahuje metody pro získání seznamu logovacích záznamů a jejich zpracování.
- **vmMainPage.cs –** zde je definován ViewModel pro hlavní stránku aplikace, kde se propojuje datový model s uživatelským rozhraním.
- **vmRequest.cs –** tato třída kromě propojení Modelu s View obsahuje i metody pro odebírání a přidávání položek do fronty A a B.
- **vmSettings_Communications.cs –** tato třída obsahuje metodu, která slouží pro přihlášení uživatele. Dále je zde vytvořeno oznámení pro uživatele zda byl úspěšně přihlášen. Také se zde vytvářejí příkazy pro přidávání a odebírání vozítek z kolekce.
- **vmSettingsLanguage.cs –** tato třída obsahuje konstruktor, který inicializuje kolekci jazyků z hlavního objektu aplikace (App).
- **vmStatus.cs –** tento ViewModel je vytvořen pro zobrazení stavu vozítek v aplikaci. Je zde vytvořena vlastnost SelectedVehicle, která reprezentuje vybrané vozítko uživatelem ze seznamu.

Ve View se nachází následující soubory, které se starají o vzhledovou část aplikace. Každý tento soubor ještě obsahuje code-behind k propojení View a jeho ViewModelu s příponou .xaml.cs.

- **Log.xaml** – V tomto souboru je vytvořen vzhled stránky pro zobrazení logovacích záznamů v aplikaci.V code-behindu stránky je obsažen konvertor, který převádí typy zpráv dle odpovídající ikony.
- **MainPage.xaml –** zde je vytvořen vzhled pro hlavní stránku Home Page. V code-behindu této stránky jsou události pro interakci s uživatelským rozhraním, jako jsou stisknutí tlačítka nebo kliknutí na obrazovku. Stránka obsahuje také kód pro kreslení grafického zobrazení pole, které reprezentuje mapu. Pro kreslení mapy je použita třída MapDrawable, která implementuje rozhraní IDrawable a obsahuje metodu Draw, která je volána pro vykreslení mapy na plátno (canvas). Dále je zde pro vykreslení vozítka použita třída VehicleDrawable a nakonec pro vykreslování obsazených polí je zde třída FieldDrawable.
- **Request.xaml** – tato stránka vytváří vzhled se seznamem jednotlivých zásilek, které mají být odbaveny. V code-behindu této stránky je pouze propojení této stránky s jejím vytvořeným ViewModelem. Toto propojení umožňuje manipulaci s daty.

- **Settings_Communications.xaml –** zde je vytvořen vzhled stránky s autentizaci uživatele a nastavení komunikace. Stejně jako u stránky Request je v code-behindu pouze propojení přes BindingContext s ViewModelem.
- **Settings_Database.xaml** – tato stránka vytváří vzhled pro základní nastavení databáze.
- **SettingsLanguage.xaml –** toto je stránka pro výběr jazyka aplikace. Kromě propojení stránky s ViewModelem v code-behindu je zde vytvořena část logiky pro výběr jazyka.
- **Status.xaml –** zde je vytvořen vzhled stránky se seznamem vozítek, které jsou dostupné v aplikaci.

Další částí projektu jsou soubory App.xaml, AppShell.xaml a MauiProgram.cs, kde jejich funkce v projektu je následující:

- **App.xaml –** ve vzhledové části stránky jsou zdroje, ve kterých jsou definovány barvy, styly a jazyky aplikace. V code-behindu se nastavuje výchozí heslo pro přihlášení. Dále zde probíhá načtení jazyků a výchozí nastavení celé aplikace.
- **AppShell.xaml –** na této stránce se vytváří menu pro aplikaci.
- **MauiProgram.cs –** tento kód obsahuje konfigurace služeb, ViewModelů inicializace pro vytvoření aplikace.

5.1.2 ARField_DComm

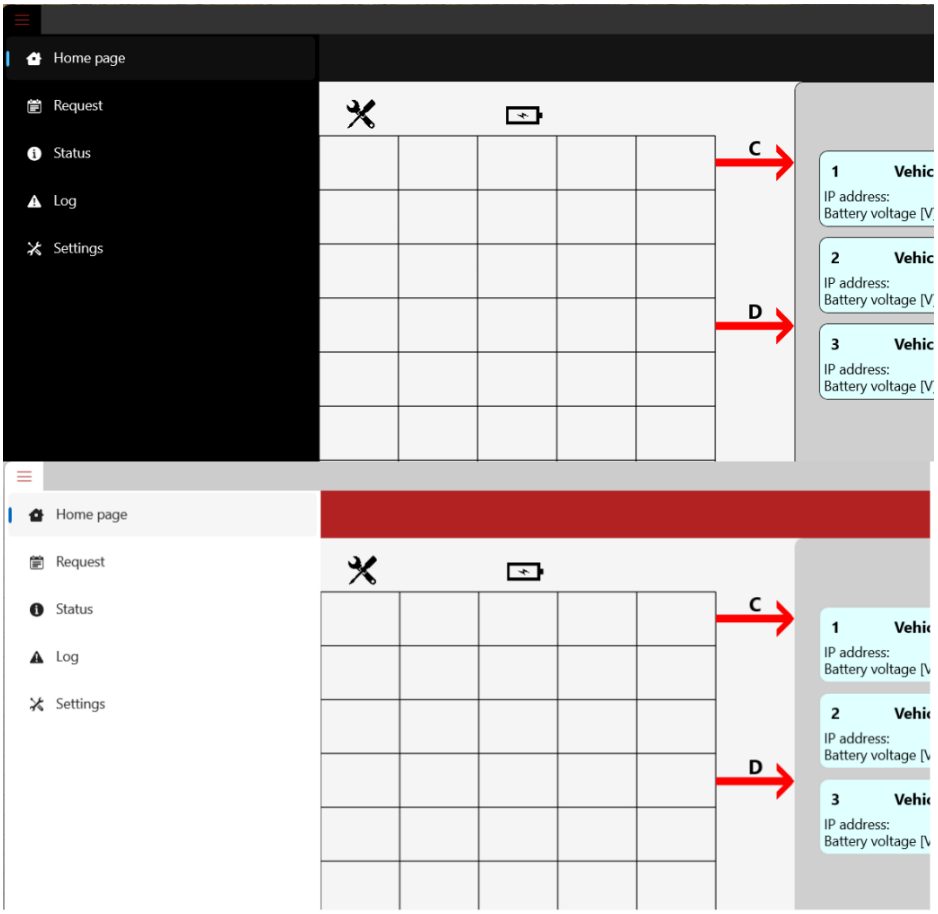
Druhý projekt celého řešení je ARField_DComm, který slouží pro komunikaci. Předává si data s projektem ARField. Tento projekt se skládá z následujících souborů:

- **ARField_DataCommunication_Properties.cs –** tato třída poskytuje základní infrastrukturu pro komunikaci s daty v aplikaci. Definuje stavy komunikace a zprávy, které mohou být během komunikace obdrženy.
- **ARField_DataCommunication_Task.cs –** tato třída umožňuje navázat a udržovat spojení s dalším zařízením přes síť. V této aplikaci je to navazování spojení s Raspberry Pi 4.

5.2 Uživatelské rozhraní

Realizace uživatelského rozhraní vychází z návrhu konceptu, který je popsán v kapitole 2.3. Uživatelské rozhraní vizualizačního softwaru je vytvořeno v podobě desktopové či mobilní aplikace, která se po prvním spuštění automaticky nainstaluje do zařízení. Aplikace se automaticky přizpůsobuje podle systémového nastavení zařízení. Na obrázku 5.2 je možné vidět porovnání aplikace při nastavení světlého či tmavého režimu v systému. Pro možnost nastavit různé vlastnosti pro světlý a tmavý režim aplikace byl použit prvek AppThemeBinding. Tento prvek umožňuje dynamické přizpůsobení vzhledu aplikace podle aktuálního nastavení.

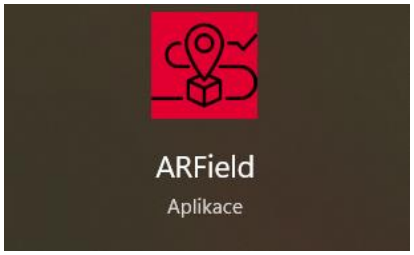
Základem aplikace je hlavní stránka, ze které se lze navigovat na vedlejší stánky pomocí tlačítek. Každá vytvořená obrazovka obsahuje navigační panel. Hlavní navigační panel je v podobě rozbalovacího Menu, kde lze přecházet mezi jednotlivými obrazovkami.



Obrázek 5.2 Automatické přizpůsobení aplikace dle zvoleného systémového nastavení

5.2.1 Ikona aplikace

V .NET MAUI je umožněna změna ikony aplikace. Pro vytvoření ikony aplikace je zapotřebí do složky AppIcon vložit .svg upravený soubor. V této konkrétní aplikace byl jako podklad použit červený čtverec o rozměrech 50 x 50 px. Samotná ikona byla stažena z volně dostupného zdroje s příponou .png [37]. Tento soubor .png byl poté převeden do formátu .svg. Následně bylo zapotřebí této ikoně změnit velikost na 30 x 30, aby ji bylo možné použít a odpovídala tak velikosti ikony operačního systému. Pro vytvoření ikony bylo nutné do hlavní části projektu naimportovat cestu těchto dvou obrázků .svg a nastavit je jako MauiIcon. Výsledná realizovaná ikona aplikace je na obrázku 5.3.



Obrázek 5.3 Ikona aplikace ARField

5.2.2 Rozbalovací Menu

Rozbalovací Menu obsahuje vždy název stránky společně s ikonou. Tyto ikony byly do aplikace přidány přes vlastnost Glyph, kam se předává jejich Unicode. Na webové stránce https://andreinitescu.github.io/IconFont2Code/ lze nalézt jednotlivé Unicody vybraných ikon, které jsou následně vkládány do aplikace. Celé Menu je tvořeno v AppShell, kde vzhledová stránka je psána v XML. Vlastnost rozbalovacího Menu je nastavena na Flyout. Tato vlastnost umožňuje skrýt celé Menu a po kliknutí na ikonu nabídky se toto Menu rozbalí.

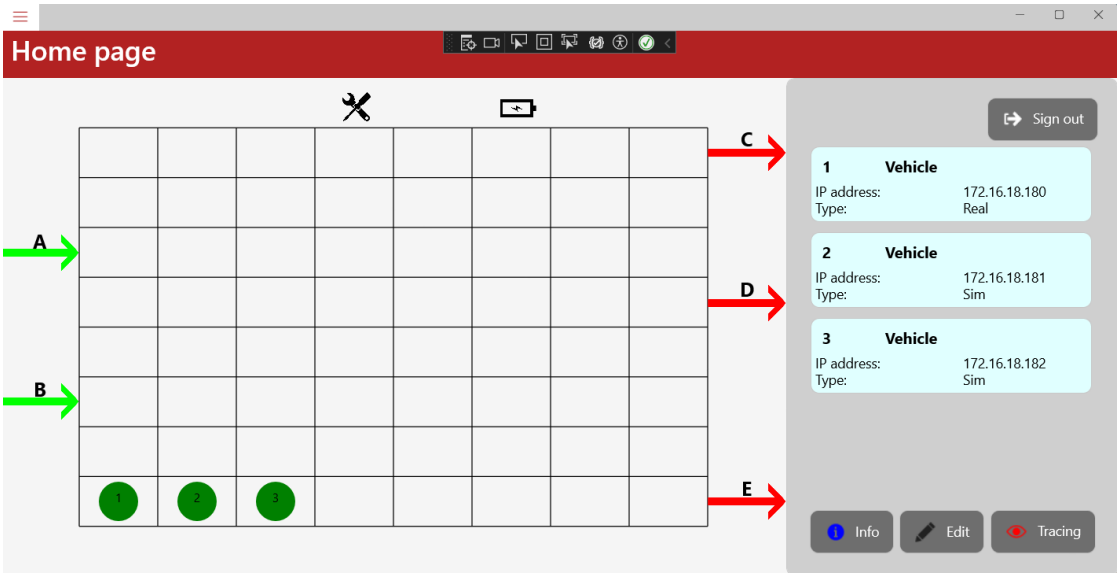
Pro vytvoření vzhledu rozbalovacího menu je použito klíčové slovo ShellContent, které je definováno vlastnostmi Title, ContentTemplate a Route. Title označuje titulek dané stránky. V této aplikaci je titulek dynamický, což znamená, že umožňuje aktualizaci titulku definovaný v jiné části aplikace. Proto je zde název titulku dostupný pod klíčovým slovem. To je využíváno při změně jazyku aplikace, která je popsána níže. ContentTemplate je vlastnost, která odkazuje na vzhled nějakého vytvořeného View nebo obsahu, který má být zobrazen. V uvedeném kódu níže ContentTemplate definuje, jak bude vypadat vzhled stránky Mainpage.xaml. Poslední vlastnost Route označuje cestu, kde se tento obsah a vzhled nachází, takže slouží jako prostředek pro navigaci na konkrétní stránku. Dále přes ShellContent.Icon se vkládá ikona, která se zobrazuje vedle titulku. Barva této ikony je nastavená tak, že se dynamicky mění v závislosti na systémovém nastavení zařízení. Pokud má aplikace světlý motiv, ikona je tmavě šedivá a pokud má aplikace tmavý motiv, ikona se zobrazuje jako bílá. Tímto způsobem je zajištěno, že ikona bude vždy viditelná. Níže je ukázka kódu, který definuje část vzhledu rozbalovacího menu.

```
<ShellContent
    Title="{DynamicResource strShellMainPage}"
    ContentTemplate="{DataTemplate views:MainPage}"
    Route="MainPage">
    <ShellContent.Icon>
        <FontImageSource FontFamily="AwesomeIcons"
                         Glyph="&#xe00d;"
                         Color="{AppThemeBinding  Light={StaticResource
Gray900}, Dark={StaticResource White}}" />
    </ShellContent.Icon>
</ShellContent>
```

5.2.3 Home Page

Hlavní obrazovka vizualizační aplikace MainPage.xaml je rozdělena pomocí mřížky (Gridu). Grid umožňuje rozdělit určitou oblast na libovolný počet řádků a sloupců. Pro účely této aplikace je hlavní obrazovka rozdělena na dva samostatné sloupce. První sloupec zobrazuje mapu a jeho velikost je definována hvězdičkou. Tím je vytvořen sloupec o velikosti, která je závislá na délce informací v druhém sloupci. Pokud se tedy změní velikost vizualizačního okna nebo množství obsahu ve druhém sloupci, dojde k znovu vykreslení mapy s novým měřítkem odpovídající aktuální velikosti vizualizačního okna.

Tato mapa zprostředkovává jednoduchý přehled o poloze vozítek v reálném prostředí a čase. Uživateli je zde umožněno monitorovat pohyb vozítek. Mapa je konstruovaná za pomoci vlastní vytvořené metody Draw ve třídě MapDrawable, která implementuje rozhraní IDrawable. Díky tomuto rozhraní je umožněno kreslení objektů do určité oblasti pomocí vstupních parametrů canvas a dirtyRect. Pro nakreslení svislé a vodorovné přímky je použit příkaz canvas.DrawLine, kterému je zadán počáteční a koncový bod dané přímky ve 2D souřadnicovém systému. Vztažný bod [0; 0] se nachází v levém horním rohu vizualizačního okna. Mapa se skládá z vnější a vnitřní oblasti. Vnitřní část je rozdělena na 8x8 čtverců o stejné velikosti a představuje prostor kde se pohybují vozítka. Vnější část mapy slouží pro manipulaci se zásilkami a tvoří rámeček kolem vnitřní mapy. Pro vytvoření mapy je zapotřebí nakreslit několik svislých a vodorovných přímek stejně vzdálených od sebe. Celá mapa obsahuje 10 x10 čtverců. Pro nakreslení stejně vzdálených přímek je nejprve nutné zjistit celkovou šířku a výšku vykreslovacího okna. Tyto informace o velikostech okna poskytují příkazy dirtyRect.Height a dirtyRect.Width. Jelikož se do okna vykreslují jen přímky tvořící vnitřní část mapy, bylo nejprve nutné nalézt vhodnou pozici první přímky. Nejprve byly vytvořeny pomocné proměnné offset_x a offset_y, které představují pozici první přímky, která je posunuta o velikost vnějšího rámečku. Následně byla vypočtena nová velikost oblasti kam lze vykreslovat vnitřní část mapy, která je menší o dvojnásobný offset v ose x a v ose y. Jednotlivé přímky byly vykresleny ve for cyklu.



Obrázek 5.4 Uživatelské rozhraní – obrazovka „Home Page“

Vnější rámeček sloužící jako manipulační prostor, obsahuje dále označení servisní a dobíjecí stanice pro vozítka. Jednotlivé prostory pro naložení a vyložení zásilek jsou označeny šipkami určující směr toku zásilek, jsou zde dvě startovací a tři cílové pozice. Tyto šipky a doplňkové ikony byly vybrány na webové stránce https://andreinitescu.github.io/IconFont2Code/ kde je zapotřebí mít již stažené fonty jako například icofont nebo barlow. Do aplikace jsou v části .xaml přidány pomocí příkazu <Image>, který umožnuje dle vybraného fontu následně zobrazit ikony a nastavit jejich pozici a velikost.

V aplikace je realizována detekce kliknutí uživatele na mapu. Toto může být využito například k získání bližších informací o zásilkách. Aby bylo možné detekovat kliknutí uživatele na obrazovku je pro tuto akci použita metoda TapGestureRecognize, která udává přesné souřadnice uživatelského kliknutí. Do vytvořeného Listu Cell jsou postupně ukládána všechna data čítající souřadnice každého z polí mapy. List tedy obsahuje vždy položku se 4 souřadnicemi. Tyto souřadnice vymezují vždy jeden čtverec z mapy. V tomto případě je ukládáno 100 položek, jelikož velikost celého prostoru mapy je 10x10. Jedna položka obsahuje souřadnice xmin, xmax, ymin, ymax a další dvě číslice popisující číslo řádku a sloupce pro maticové použití. Následně je využit System.Linq, který definuje metodu Where. Tato metoda vždy když uživatel klikne na určité políčko nacházející se na mapě, prohledá celý List a vrátí konkrétní položku odpovídající danému políčku. Hledání v Listu je založeno na porovnávání souřadnic jednotlivých políček se souřadnicemi, na které kliknul uživatel. Touto metodou je možné i měnit barvy políček, jelikož aplikace od vozítka dostává informace o poloze v podobě maticového zápisu s konkrétním řádkem a sloupcem matice (mapy).

Výše popsaný princip popisuje chování dané části aplikace, která je napsaná v podobě code-behind stránky. Aby bylo možné tuto logiku uplatnit a použít při zobrazení uživateli,

je nezbytné vytvořit kód v XML. V prvním sloupci, kde se zobrazuje mapa je použito rozložení AbsoluteLayout. Jedná se o typ rozložení, který umožňuje umístit prvky uživatelského rozhraní na konkrétní souřadnice na obrazovce. Pomocí GraphicView.Drawable je pak definován obsah, který má být vykreslen. V této aplikaci je to přes vytvoření instance MapDrawable.

Ve druhém sloupci obrazovky se nachází seznam vozítek se základními informacemi o nich a tlačítka Info, Edit a Tracing. Seznam vozítek je po spuštění aplikace prázdný, dokud uživatel vozítka nepřidá přes obrazovku Settings communucation. Při stisku tlačítka Info je uživatel odkázán na obrazovku Status. Po stisknutí tlačítka Edit, je uživatel odkázán na stránku Communication, kde je vyzván k autentizaci pro zpřístupnění úprav a nastavení komunikace. Stiskem tlačítka Tracing je uživatel navigován na obrazovku Request. Po přepnutí se na zmíněné obrazovky je možné šipkou zpět vrátit se opět na Home Page obrazovku. Pro odhlášení uživatele je v pravém horním rohu tlačítko Sign out. Detekce odhlášení a přihlášení je uživateli oznámeno pomocí Toastu.

Pro zobrazení seznamu vozítek bylo zapotřebí použít CollectionView. CollectionView umožňuje zobrazit seznam jednotlivých vozítek. Přes vlastnost IteamSource je vybrán zdroj dat, který má být zobrazen. Pomocí kolekce CollectionView.ItemTemplate je definováno, jak mají být jednotlivé položky v seznamu uživateli zobrazeny. Tato kolekce pro zobrazení obsahu jednotlivých položek používá vlastnost DataTemplate, kde se nastavuje atribut x:DataType. Tento atribut specifikuje typ dat, který je vázán na jednotlivé položky. V této aplikace je pro typ dat použit VehicleBase.cs. Pro každou položku v seznamu je vytvořen vizuální rámeček přes Frame. Aby se jednotlivé položky ze seznamu řadily pod sebe je v této kolekci použit VerticalStackLayout, kde jednotlivé texty jsou vypisovány v Gridech přes Label. Texty je potřeba nabindovat, aby byly vázány na vlastnosti třídy VehicleBase.cs.

Tlačítka jsou v uživatelském rozhraní vykreslovány přes Button, kde pozice tohoto tlačítka je definována pomocí Gridu. V Button lze nastavit události, které se mají vykonat po stisknutí tlačítka. Definice těchto událostí je vytvořena v code-behindu stránky. Například při stisknutí tlačítka Info je navigace na jinou stránku provedena přes příkaz Navigation.PushAsync.

Pro vykreslení vozítek na mapě slouží třída VehicleDrawable, která je vytvořená v code-behindu stránky. Má vlastnost BindableProperty která umožňuje jiným částem kódu přistupovat k informacím o vozítkách a aktualizovat je, pokud se změní. Například, když se vozítko pohne nebo změní svůj stav, BindableProperty zajistí, že tyto změny se promítnou do vizuálního zobrazení vozítka na obrazovce. Vozítka jsou vykreslovány přes canvas.FillCircle, kde souřadnice x a y jsou univerzálně vypočteny. Výpočet je založen na obdobném principu jako vykreslování mapy, kde nyní se vypočítávají středy jednotlivých polí mapy. Do tohoto výpočtu se předává aktuální poloha vozítka. Také je zde řešena změna barvy vozítka dle sledování jeho aktuálního stavu.

Dále je zde vytvořen FieldDrawable, který umožňuje vykreslovat obsazené pole na mapě. Tato metoda obsahuje atribut FieldOccupy, který určuje, které části poje jsou obsazené, dle již vytvořeného předaného pole Occupy přes Binding. Aby mohl být použit Binding, je v této třídě implementován PropertyChanged pro detekování změny pole. Dále v metodě Draw() je vykreslovaní obsazeného pole jako světle červeného obdélníku.

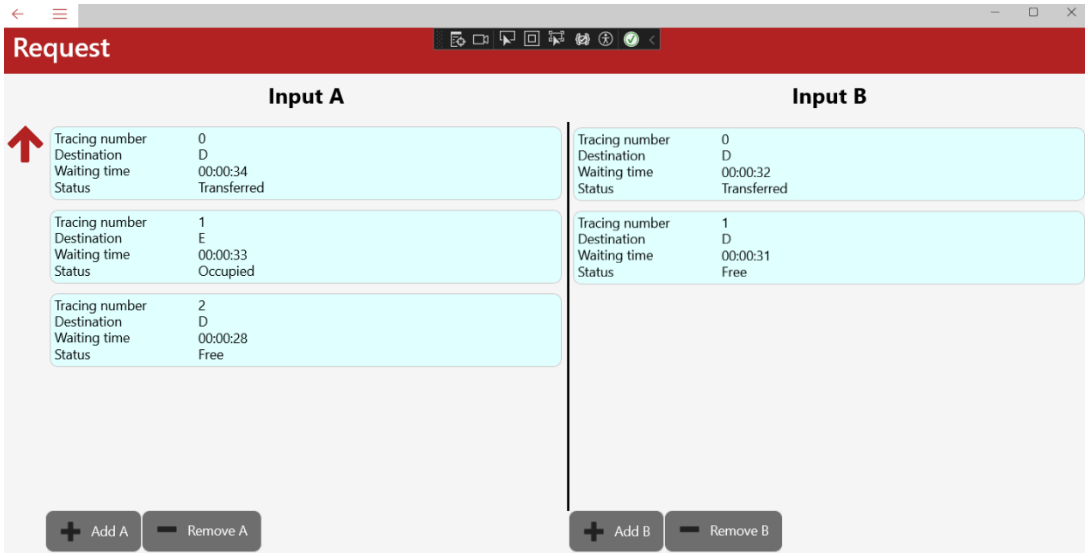
5.2.4 Request

Na této obrazovce je v uživatelském rozhraní možnost monitorovat jednotlivé zásilky. Tyto zásilky čekají ve frontě na odbavení na vstupech A a B. Zásilky jsou řazeny chronologicky za sebou v pořadí, v jakém přijeli na vstup. Tedy první zásilka v seznamu shora je zásilka, která se ve frontě nachází nejdéle. Zásilky jsou definovány svým identifikačním číslem, stavem a destinací na kterou mají být dopraveny. Každá zásilka obsahuje také čas čekání, který je počítán od doby zobrazení se ve frontě. Pokud je zásilka odbavena, automaticky změní svůj stav na Transferred a zůstane ve frontě pro zkoumání historie. V dané aplikaci je přidávání zásilek do fronty řešeno pomocí tlačítek což reprezentuje v reálném provozu skenování příchozích zásilek. Jsou zde přidány i tlačítka pro odstranění zásilky, která zde reprezentují situaci, kdy například obsluha odebere zásilku ručně.

Celá obrazovka je rozdělena jedním Gridem na čtyři sloupce a tři řádky. V prvním sloupci je vykreslena přes Image červená šipka, která určuje směr fronty. V prvním řádku jsou vykresleny nadpisy pro rozdělení jednotlivých front přes Label. Zde je nastaven FontSize na Medium a FontAttributes na Bold. Dále je nastaven HorizontalTextAligment, který určuje, jak má být vykreslovaný text zarovnán uvnitř vizuálního prvku (Label) a nakonec VerticalOptions je použit pro vertikální umístění prvku v rámci kontejneru, zde je kontejnerem Grid. Jednotlivé položky fronty jsou vykreslovány ve ScrollView, aby bylo možné v případě více položek srolovat až na konec fronty. Pro zobrazení jednotlivých položek ve frontě je použit CollectionView, kde zdrojem dat je AppModel.BoxA pro jednu frontu a AppModel.BoxB pro druhou frontu. Typ dat, který má být navázán je v DataTemplate nastaven na Entitu Boxes.cs. Pro řazení položek pod sebe je použit VerticalStackLayout. Vzhled stránky Request je na obrázku 5.5.

V MainModelu jsou vytvořené fronty BoxA a BoxB jako ObservableCollection. Ve ViewModelu jsou vytvořené příkazy pro jednotlivá tlačítka. Při přidávání položky do kolekce se rozlišuje, zda je kolekce prázdná či už nějakou položku obsahuje. Pokud kolekce neobsahuje položku a klikne se na tlačítko Add, přidá se zásilka s identifikačním číslem 0. Pokud uživatel klikne na tlačítko a kolekce již obsahuje nějakou položku tak se nejdříve vypočítá maximální hodnota identifikačního čísla (ID) ze všech položek v kolekci. Poté se do kolekce přidá nová položka s ID o 1 vyšším, než je maximální ID v kolekci. Destinace je položkám přiřazována přes náhodný generátor. Při odebíraná položky tlačítkem Remove se nejprve kontroluje, zda kolekce obsahuje alespoň jeden prvek. Pokud neobsahuje, neprovede se žádná akce. Když kolekce není prázdná a není

vybraná položka, která se má smazat, odstraní se vždy první položka v kolekci. Pokud je vybrána v kolekci položka vymaže se tato položka. Při automatickém odebrání položky (vozítko odebralo zásilku) se tato kolekce chová jako FIFO fronta, jelikož se položky z kolekce odebírají v pořadí, kterém se objevily v kolekci. Dále je v Modelu Boxes.cs spuštěn Task. Task spouští asynchronní úlohu, která běží na pozadí a umožňuje pokračovat hlavnímu vláknu, aniž by bylo nutné čekat na dokončení úlohy. V této úloze je použit Task pro měření času, který definuje čekání zásilky ve frontě.



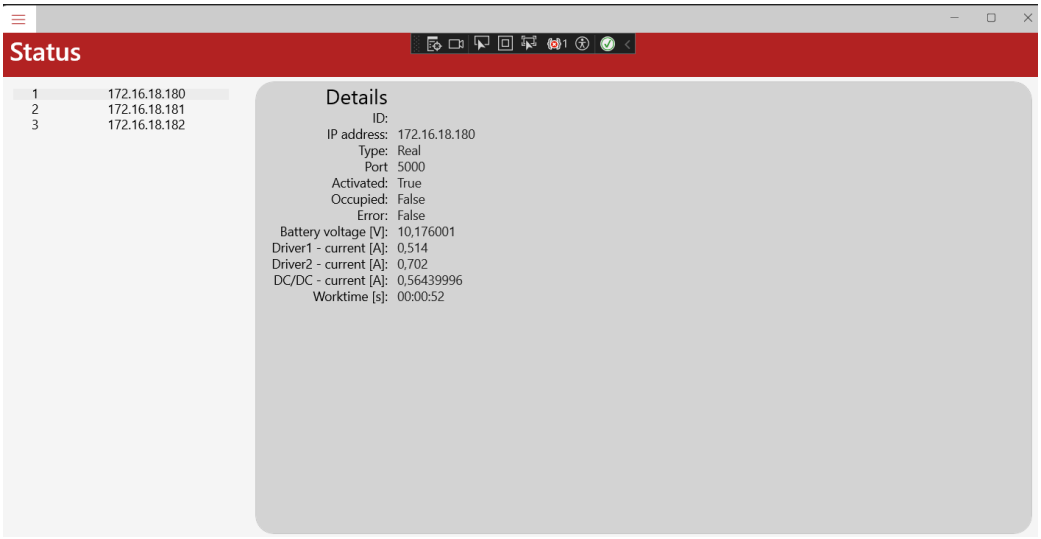
Obrázek 5.5 Uživatelské rozhraní – obrazovka „Request“

5.2.5 Status

Tato obrazovka slouží uživateli pro zobrazení všech dostupných informací týkajících se vozítka. Stránka je rozdělená na dva sloupce přes Grid. V první sloupci je zobrazen seznam vozítek. Pro zobrazení seznamu vozítek je použit CollectionView jako u Home Page obrazovky. Zde je navíc umožněn výběr vozítka ze seznamu. Pro realizaci výběru bylo zapotřebí v CollectionView nastavit SelectedItem. Ten je nastaven přes Binding na SelectedVehicle, který je vytvořen ve ViewModelu této stránky. Dále je zde nastaven SelectionMode na Single, aby bylo možné vybrat pouze jednu položku ze seznamu, u které se mají zobrazit informace.

V druhém sloupci je vytvořen rámeček přes Frame. V tomto rámečku se zobrazují bližší data o vybrané položce ze seznamu. Přes vlastnost CornerRadius jsou rámečku nastavené zaobleného rohy. Dále je nastaven Margin, který umožňuje definovat vnější okraje okolo rámečku. V aplikaci je Margin nastaven na hodnoty Margin = 15,5,15,10. První číslo určuje, jaký má být okraj od horní strany, druhé číslo nastavuje okraj od pravé strany. Další třetí číslo definuje vzdálenost objektu od spodní strany. Poslední číslo nastavuje okraj od levé strany. Pro vykreslení jednotlivých informací pod sebe je použit VerticalStackLayouts. Do tohoto kontejneru je přidán Grid, který kontejner rozděluje na

dva sloupce. Do prvního sloupce se přes Label vypisuje text, který je přes Dynamic Resource odkázán na klíč, který je uložen v Language.Czech.xaml a Language.English.xaml. Funkce klíčů, která slouží pro přeložení textu je popsána níže v podkapitole Settings. Do druhého sloupce je také přes Label vypsán odpovídající text přes Binding. Binding je zde použit pro navázání dat na vlastnost objektu zvolené položky.



Obrázek 5.6 Uživatelské rozhraní – obrazovka „Status“

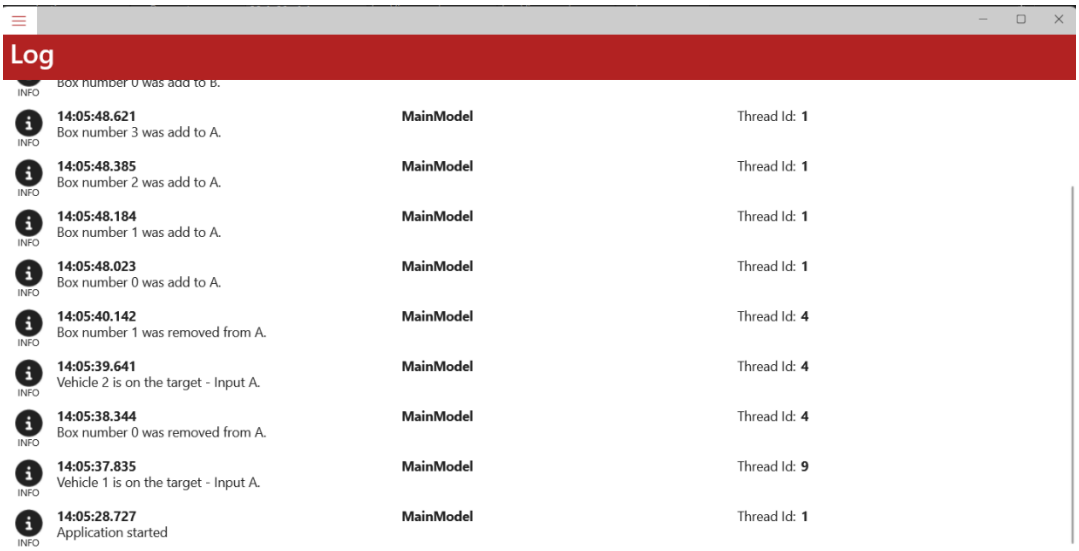
5.2.6 Log

Tato obrazovka slouží uživateli k zobrazení a zaznamenání událostí, stavů a chyb v běžícím programu. Je strukturována pomocí tabulkového rozložení Grid. V této struktuře je použito CollectionView k zobrazení kolekce záznamů. Každý záznam je zobrazen ve dvousloupcovém rozložení. V prvním sloupci je ikona odpovídající typu zprávy a ve druhém sloupci je obsah záznamu. Obsah záznamu je zobrazen v kontejneru VerticalStackLayout, který je rozdělen na tři sloupce.

První část záznamu obsahuje čas, kdy byl záznam pořízen spolu s obsahem zprávy. Tato zpráva je vykreslována přes Label, kde se používá vlastnost LineBreakMode. Tato vlastnost určuje, jak se má s textem zacházet, když je příliš dlouhý. Zde je vlastnost nastavena na WordWrap což znamená, že se text bude zalamovat na konci řádku a bude pokračovat na novém řádku. Další částí záznamu je zdroj, odkud je záznamu pořízen. A poslední část záznamu definuje identifikátor vlákna. Vzhled této obrazovky je zobrazen na obrázku 5.7.

V Code-behindu této stránky je implementováno propojení na ViewModel. Ve ViewModelu jsou uchována data, která mají být zobrazena. Dále je zde vytvořen konvertor TypeGlyphConverter, který přebírá typy zpráv jako například, info, error a převádí je na jejich příslušnou ikonu pomocí specifických Unicode přes Glyph.

Ve ViewModelu je vytvořena veškerá logika pro tuto obrazovku. Jsou zde implementovány dvě třídy. V první třídě je zajištěna práce s daty, které mají být zobrazena na této obrazovce. Jsou zde založeny dvě kolekce (ObservableCollection), první kolekce je AllEntries a zde jsou uchovávány veškeré záznamy a další kolekcí je Entries, kam se ukládají konkrétní záznamy. Pomocí metody LoadData() jsou do kolekce AllEntries uložena data, která jsou poté seřazena podle času a uložena do kolekce Entries. Dále je zde implementována asynchronní metoda GetLogList(), která získává seznam záznamů. Další implementovanou třídou je LogEntry. Tato třída reprezentuje jeden záznam. Konstruktor přijímá souvislý text, který obsahuje jednotlivé informace o záznamu rozdělené svislou čarou. Tento řetězec obsahuje informace o čase, logovací úroveň (info, error, …), ID vlákna, zdroj a samotnou zprávu. Tento text je rozprasován a dále testován. Nejprve se testuje podmínky, zda počet částí v řetězci je roven 6, pak se předpokládá, že jsou v textu obsaženy úplné informace. Pokud je počet částí roven 5, pak chybí informace o zdroji logování a atribut Source je nastaven na prázdný řetězec. Pokud není splněna ani jedna z podmínek, atribut Message je reprezentován prázdným řetězcem.

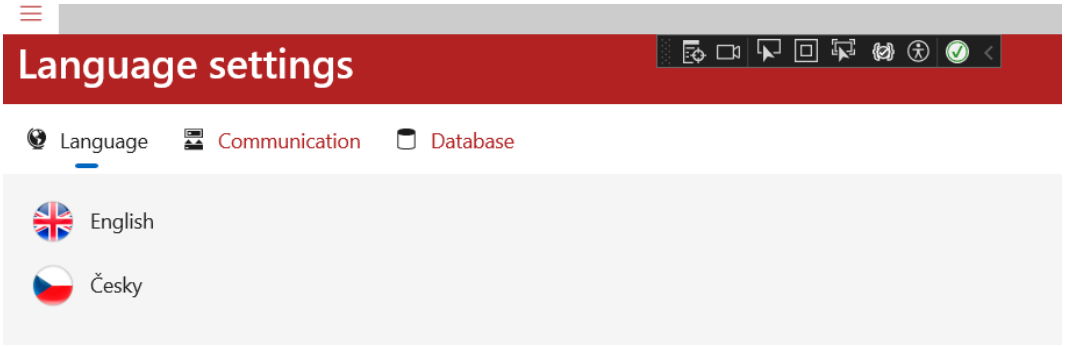


Obrázek 5.7 Uživatelské rozhraní – obrazovka „Log“

5.2.7 Settings

Na obrázku č. 5.8 je znázorněna obrazovka „Settings“. Tato obrazovka má své navigační podmenu, které odkazuje na stránku Language, Comunications a Database. Toto podmenu je vytvořeno pomocí klíčového slova FlyoutItem v .xaml kódu stránky. Mezi těmito obrazovkami se lze libovolně přepínat. Na obrazovce Language se nastavuje jazyk celé aplikace. Aplikace se nastavuje do českého jazyka kliknutím na tlačítko s českou vlajkou. Pokud je zvolen jazyk čeština, pak je veškerý text napsaný v aplikaci, včetně rozbalovacího menu, převeden do českého jazyka. To stejné platí při změně na anglický jazyk.

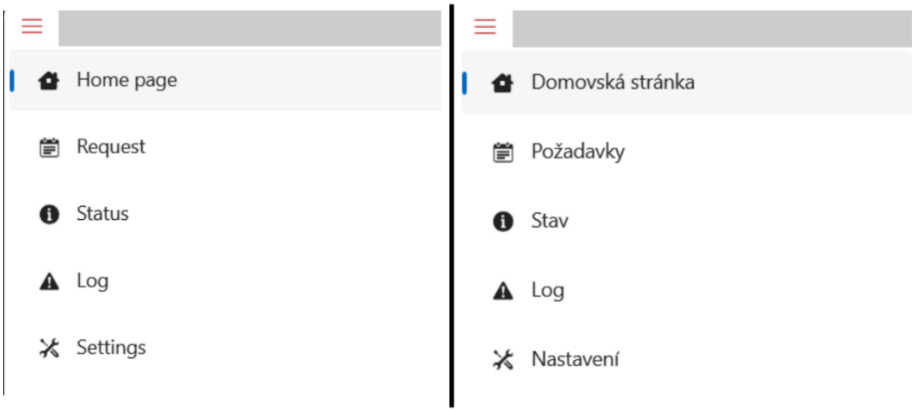
Pro vzhled stránky je vybráno rozložení CollectionView. Toto rozložení umožňuje zobrazit položky ze seznamu, kde jsou uložené jazyky. Pomocí SelectionMode je nastaveno, že lze vybrat pouze jednu položku ze seznamu. Položky jsou uspořádány tak, že jedna položka zabírá vždy jeden celý řádek a položky jsou řazeny vertikálně pod sebe. Toto je umožněno nastavenými vlastnostmi jako jsou Span, Orientation v GridItemsLayout. Pro zobrazení jednotlivých položek, které obsahují vlajku společně s názvem příslušného jazyka je použita definice CollectionView.ItemTemplate. V této sekci je nejprve nastavena příslušná šířka jednotlivých sloupců (vlajka, text), kde první sloupec má pevnou šířka 50 a druhý je typu hvězdička a zabírá tak zbytek dostupného prostoru. U obrázku je použit Binding s vazbou na IconName a ta obsahuje odkaz na to, kde se daný obrázek nachází. Pro text je použit obdobný postup jako u obrázku. Je zde také použit Binding s vazbou na LanguageName. Příslušné obrázky společně názvem jazyka jsou uloženy v Listu AppLanguages. Vzhled této stránky je na obrázku 5.8.



Obrázek 5.8 Uživatelské rozhraní – obrazovka „Settings – Language“

Konkrétní slova, která je nutné při změně nastavení jazyka přeložit se nacházejí v souborech Language.Czech.xaml a Language.English.xaml. V těchto souborech se nacházejí klíčová slova, pod kterými jsou uloženy překlady. Tyto klíčová slova jsou používána v .xaml kódu a slouží jako odkaz na jednotlivé překlady. Použití těchto klíčových slov v DynamicResource zajišťuje snadnou změnu a aktualizaci textu za běhu aplikace, aniž by bylo nutné měnit samotný .xaml kód. V části App.xaml.cs jsou inicializovány dostupné jazyky aplikace, které jsou ukládány do Listu AppLanguages. Pokud uživatel vybere některý z jazyků, provede se obslužná rutina Language_SelectionChanged, která detekuje, že došlo ke změně jazyka. V této rutině se volá metoda SetLanguage, která nastavuje jazyk aplikace na základě výběru uživatele. Tato metoda nejdříve kontroluje, zda se vybraný jazyk liší od aktuálně uloženého jazyka, pokud ano, dojde k aktualizaci tohoto jazyka v uživatelových preferencích. Dále tato metoda pracuje s ResourseDictionary, ve kterém jsou uloženy lokalizační zdroje. Tyto zdroje slouží pro aktualizaci obsahu aplikace dle vybraného jazyka. Na obrázku 5.9 je znázorněna funkčnost na rozbalovacím Menu této aplikace. Při spuštění aplikace dojde k automatickému nastavení jazyka aplikace dle systémového nastavené zařízení. Tedy

pokud je v systému zařízení nastaven anglický jazyk, aplikace se při spuštění automaticky přeloží do anglického jazyka.



Obrázek 5.9 Uživatelské rozhraní – rozbalovací Menu

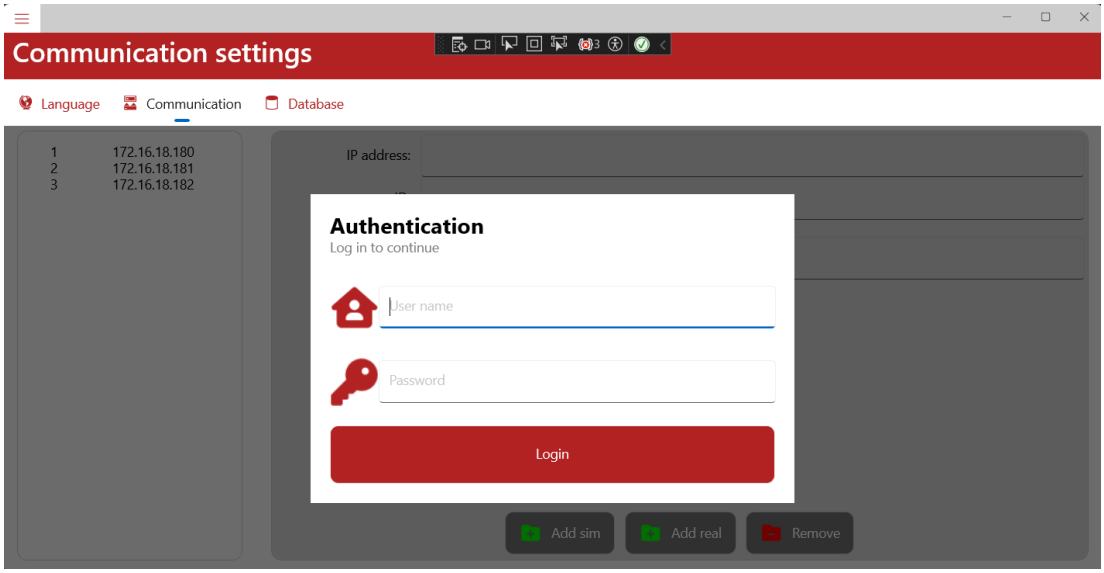
V podmenu Communication je vytvořena ovládací část pro nastavení komunikace nadřazeného řídícího softwaru s robotickým vozítkem. Toto podmenu je přístupné až po autentizaci uživatele. Autentizace spočívá v zadání správného uživatelského jména a hesla. Po přihlášení je uživateli posláno oznámení o úspěšném přihlášení a je mu zpřístupněna a zobrazena obrazovka pro nastavení komunikace. Pokud uživatel zadá špatné heslo vypíše se mu informace, že zadal špatné heslo.

Při tvorbě této stránky bylo zapotřebí znepřístupnit nastavení komunikace, dokud uživatel není přihlášen. Pro tuto tvorbu je použito rozložení Grid, ve kterém se vykreslují dva Gridy. V prvním Gridu je vykreslen vzhled stránky pro nastavení komunikace a ve druhém Gridu je vzhled okna pro přihlášení uživatele. Pomocí podmínky u druhého Gridu, která je nastavena přes IsVisible je umožněno tímto Gridem překrýt Grid nad ním. Tvorba vzhledu jednotlivých částí (Gridů) je následující:

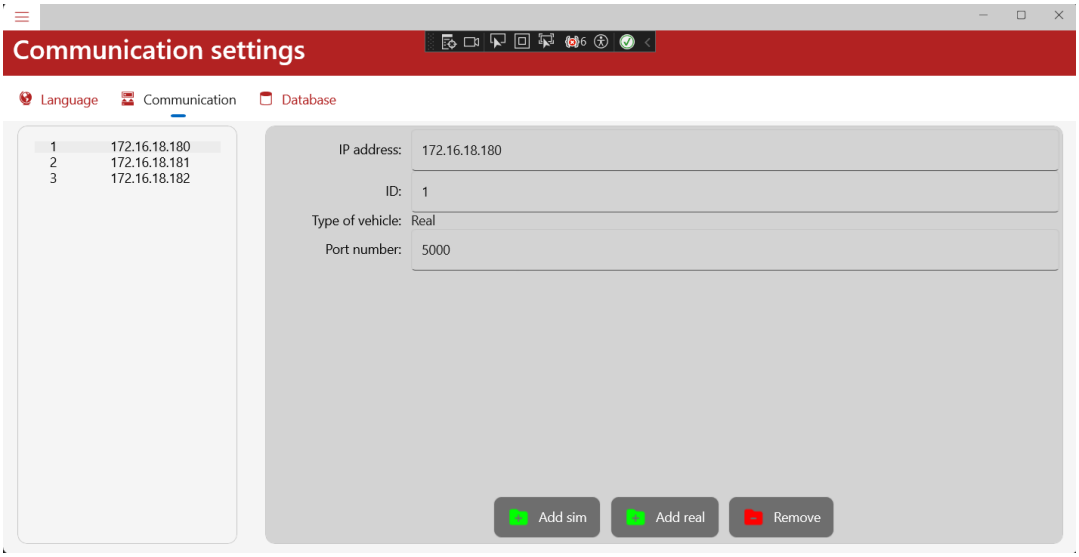
- **Autentizace** – je vytvořena pomocí Gridu s podmínkou, která je Bindovaná na AppModel.LoggedRequest. Tato podmínka říká, že tento Grid bude viditelný, pokud je požadavek na přihlášení True. Dále je v tomto Gridu nastavené tmavé pozadí pro skrytí Gridu, kde se nastavuje komunikace. Poté je do Gridu vykreslen VerticalStackLayout s bílou barvou, ve kterém je vykresleno okno s přihlášením. Pro vykreslení ikonky a textového pole byl do VerticalStackLayoutu přidán Grid, který kontejner rozděluje na dva sloupce. Do prvního sloupce se vykresluje ikona přes Image a do druhého sloupce je pomocí Entry vloženo zadávací pole. Vlastnost Entry umožňuje uživateli zadávat znaky z klávesnice. Tlačítko login je vložen pomocí vlastnosti Button, který je navázán na příkaz login ve ViewModelu této stránky vmSettings_Communication.cs. V příkazu je spouštěna metoda LoginAsync(), kde dochází k porovnávání zadaného hesla a jména s uloženým heslem a uživatelským jménem. Pokud se shodují, je přes příkaz

Toast uživatel upozorněn, že se úspěšně přihlásil. Dále se zde také nastavuje proměnná LoggedRequested na False čímž se tento Grid zneviditelní a zpřístupní se Grid s nastavením komunikace. Pod tlačítkem Login je dále vložen Label, který je také definován podmínkou IsVisible, tentokrát je však navázán na podmínku AppModel.BadAuth, která signalizuje, že proběhla neplatná autentizace, pokud je True. Tím se vypíše pro uživatele pod tlačítko Login hláška, že zadal špatné údaje. Pro uložení hesla bylo použito lokální uložiště SecureStorage. Při spuštění aplikace se v code-behindu stránky App.xaml kontroluje, zda je v úložišti SecureStorage uloženo uživatelské jméno a heslo. Pokud není, nastaví se vždy výchozí uživatelské jméno a heslo.

- **Nastavení komunikace –** Na stránce nastavení komunikace lze provádět změny IP adresy, portu a ID vozítka. Dále lze přidávat vozítko dle typu (simulované/reálné). Tato stránka obsahuje dva sloupce. V prvním sloupci se nachází seznam vozítek. Seznam vozítek v prvním sloupci je zobrazen přes CollectionView, kde je umožněn výběr vozítka jako u obrazovky Status. V druhém sloupci jsou po vybrání položky ze seznamu zobrazeny informace o položce, které lze upravovat. Upravování jednotlivých informací je umožněno přes Entry. Také se zde nacházejí tlačítka Add Real, Add Sim a Remove. Přes tyto tlačítka lze přidávat či odebírat vozítka. Funkce tlačítek je vytvořena přes Commandy, které jsou definovány ve ViewModelu vmSettings_Communication.cs. Při přidávání reálného či simulovaného vozítka do kolekce (ObservableCollection) je kontrolováno, zda je kolekce prázdná. Pokud kolekce neobsahuje žádné vozítko, přidá se vozítko s nulovým identifikačním číslem a defaultním nastavením portu a IP adresy. Pokud je vozítko přidáváno do kolekce, která již nějaká vozítka obsahuje, tak je nejdříve vypočítána maximální hodnota identifikačního čísla (ID) ze všech vozítek v kolekci. Následně se získá poslední IP adresa vozítka z kolekce AppModel.Vehicles a inkrementuje se poslední bajt (třetí bajt) této IP adresy, aby se vytvořila nová IP adresa pro nové vozítko. Poté je do kolekce přidáno vozítko s ID o 1 vyšším, než je maximální hodnota ID v kolekci a nově vypočtenou IP adresou. Vozítko lze z kolekce odebrat tlačítkem Remove. Nejprve se kontroluje, zda kolekce obsahuje alespoň jeden prvek. Pokud neobsahuje, neprovede se žádná akce. Když kolekce obsahuje již nějaká vozítka a žádné vozítko není uživatelem vybráno, odstraní se vždy první vozítko v kolekci. V modelu VehicleStatus.cs je vytvořen Task pro měření času. Tento čas definuje, jak dlouho je simulované vozítko v provozu. Tento čas se spustí každému vozítku, které je přidáno do kolekce. U reálného vozítka je tento čas odesílán řídícímu systému v paketu zprávy. Vozítko si totiž samo měří svoji dobu běhu.

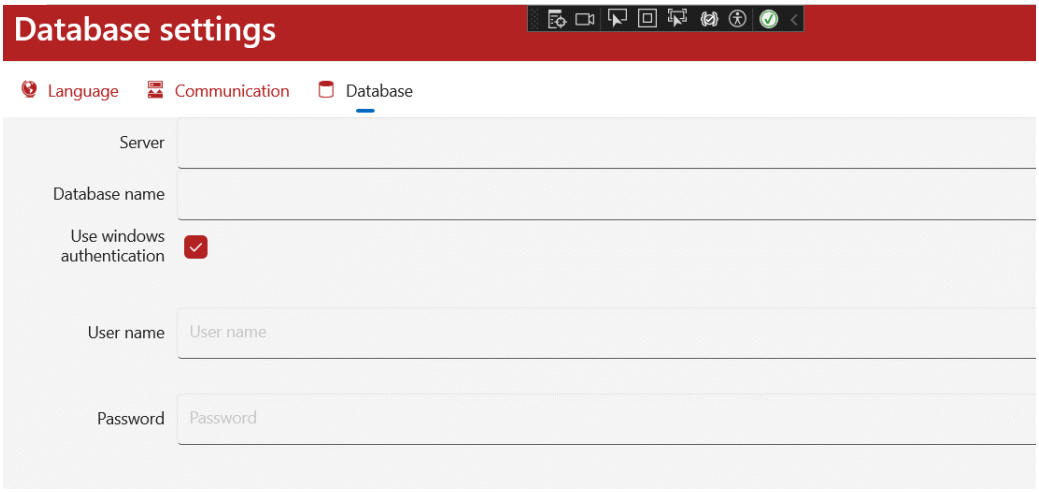


Obrázek 5.10 Uživatelské rozhraní – Autentizace



Obrázek 5.11 Uživatelské rozhraní – obrazovka „Settings – Communication“

Podmenu Database slouží pouze jako příprava pro potenciální rozšíření této práce o možnost ukládání informací o vozítkách do databáze. Po domluvě se zadavatelem je zde vytvořena část pro nastavení databáze. Zde mohou být nastaveny parametry týkající se serveru, ke kterému se uživatel chce připojit. Dále je umožněno nastavit název databáze a přihlásit se pomocí Windows autentifikace.



Obrázek 5.12 Uživatelské rozhraní – obrazovka „Settings – Database“

5.3 Řídící SW

5.3.1 Komunikace

Komunikace probíhá pouze mezi řídícím softwarem a reálným vozítkem. Aby bylo možné sjednotit vlastnosti obou typů vozítek, je v aplikaci vytvořena tzv. bázová třída (Base class) VehicleBase.cs. Vlastnosti vozítek obsahují informace, týkající se ID vozítka, stavu baterie, aktuálně načteného QR kódu, …). U simulovaného vozítka jsou tyto vlastnosti generovány interně v aplikaci a u reálného vozítka jsou posílány ve formě paketů. Jednotlivé pakety, jak již bylo řečeno v kapitole 3.6 jsou dle jejich obsahu rozděleny podle typu zprávy. Pro každý typ paketu byla vytvořena samostatná třída:

- VehicleStatus.cs – je třída, která definuje informace týkající se zprávy 0x01. Jedná se o informace typu ID vozítka, stav baterie, proud protékající drivery, vzdálenost detekovaného předmětu a další.
- VehiclePosition.cs – tato třída definuje informace, které popisují zprávu typu 0x02. Tyto informace jsou – aktuálně načtený QR kód, úhel natočení vozítka vůči QR kódu a odchylka středu vozítka od středu QR kódu v x a y souřadnicích.
- VehicleDriving.cs – je poslední třída, která definuje vlastnosti odesílaného paketu. Jedná se o informace ve tvaru příkazů pro pohyb vozítka (Jeď dopředu, dozadu, atd.).

Poté bázová třída VehicleBase obsahuje metody – ParseMessage a GetMessage pro dané zprávy dle směru toku dat.

Pokud je řídící software v pozici příjemce zpráv, tak je zpráva obdržena ve formě pole bytů. Tato zpráva je následně pomocí veřejné metody ParseMessage rozdělena na jednotlivé byty. Rozdělení těchto bytů podle příslušného počtu je definováno podle

velikosti vytvořených proměnných, které jsou definovány v této metodě. Takže pokud je první definovaná proměnná ID vozítka a má velikost dva byty, pak metoda z celé obdržené zprávy od reálného vozítka přečte první dva byty a hodnotu uloží do proměnné VehicleID.

Pokud je řídící software v pozici odesílatele zprávy, tak je zapotřebí složit pole bytů, které má být odesláno. O tvorbu tohoto pole dat se stará metoda GetMessage. Tato metoda pracuje tak, že nejprve se vytvoří pole o příslušné velikosti a do něho se ukládají jednotlivé informace v přesném pořadí za sebou. Níže je uveden výsek z definice obou metod v podobě zdrojového kódu.

```
private void ParseMessage(byte[] message)
 {
     byte MessageType = message[0];
     ushort mVehicleID = BitConverter.ToUInt16(message, 1);
     if(mVehicleID == VehicleID)
     {
         if(MessageType == 0x02)
         {
             Position.ActQR = BitConverter.ToUInt16(message, 3);
             Position.RotationAngle =BitConverter.ToSingle(message, 7);
             Position.QRDeviation = BitConverter.ToSingle(message, 11);
             Position.ActQR
Encoding.ASCII.GetString(message.Skip(15).ToArray());
         }
         else if(MessageType == 0x01)
         {
             Status.BatteryVoltage = BitConverter.ToSingle(message, 3);
             Status.CurrentDriver1 = BitConverter.ToSingle(message, 7);
             Status.CurrentDriver2= BitConverter.ToSingle(message, 11);
         }
     }
 }

public byte[] GetMessage()
 {
     byte[] buffer = new byte[6];
     byte[] IdBuff = BitConverter.GetBytes(VehicleID);
     byte[] QRBuff = BitConverter.GetBytes(QRPrediction);
     buffer[0] = IdBuff[1];
     buffer[1] = IdBuff[0];
     buffer[2] = 0;
     if (Drive.Stop) buffer[2] |= 0x01;
 }
```

Komunikační protokol je asynchronní a je rozdělen na dvě části – ARField_DataCommunication_Task a ARField_DataCommunication_Properties.cs. Properies skript obsahuje pouze definice vlastností a událostí. Je zde implementováno několik tříd. První třída ARField_DataCommunication zajišťuje komunikaci mezi aplikací a reálným vozítkem pomocí síťového soketu. Tato třída obsahuje vlastnosti pro sledování stavu komunikace, nastavení adresy, portu atd. Dále obsahuje události pro zpracování přijatých dat a zpráv. Další třída ARField_Comm_Message obsahuje informace o zprávě, která může být generována v různých částech programu. Definuje se

zde o jaký typ zprávy se jedná, zda tato zpráva je errorová, stavová nebo zda se jedná o přenášená data. Třída ARField_Comm_DataRx reprezentuje událost, která oznamuje příjem zpráv. Do vlastnosti Data se ukládá samotný obsah přijaté zprávy, jelikož jsou Data typu object, může být přijatá zpráva libovolného datového typu.

Druhý skript Task rozšiřuje první skript o implementaci metod třídy ARField_DataCommunication. Tyto metody slouží pro připojení, odpojení a komunikaci přes síťový socket. Skript obsahuje následující metody:

- **Connect()** – tato metoda slouží k navázání spojení a spouští zde hlavní metodu Communicator. Je zde také použit token, který sleduje žádosti o zrušení operace prováděné v metodě Communicator().
- **Disconnect()** – je metoda, která by v případě rozšíření mohla sloužit pro odpojení komunikace s reálným vozítkem.
- **Comunicator()** – je hlavní metoda pro komunikaci s reálným vozítkem. Tato metoda je spouštěna jako Task a obsahuje logiku pro navázání spojení jako server. Čeká se zde na příchozí data ze soketu a zpracovává je.
- **SendBytes()** – tato metoda slouží k odeslání pole bajtů pomocí komunikačního socketu. Používá asynchronní operaci SendAsync(), která odesílá data asynchronně a neblokuje hlavní vlákno.
- **Receive()** – metoda pro příjem dat ze socketu. Čeká na příchozí data a poté je zpracuje a vrátí jako pole bajtů. Používá asynchronní operaci ReceiveAsync(), která čeká na příjem dat ze socketu a následně je zpracuje.

V PhysicalVehicle.cs je dále vytvořena instance na ARField_DataCommunication pro komunikaci s vozítkem. Nastavuje se zde adresa zařízení a port, přes který je prováděna komunikace. Dále je zde definována obsluha události DataReceived, která se spustí, když jsou data přijata. V této obsluze je zpracování přijatých dat prováděno voláním metody ParseMessage, která data rozprasuje na potřebné byty. Dále je zde zahájen Task, který periodicky odesílá data pomocí metody SendData, ve kterých jsou přenášeny pohybové příkazy.

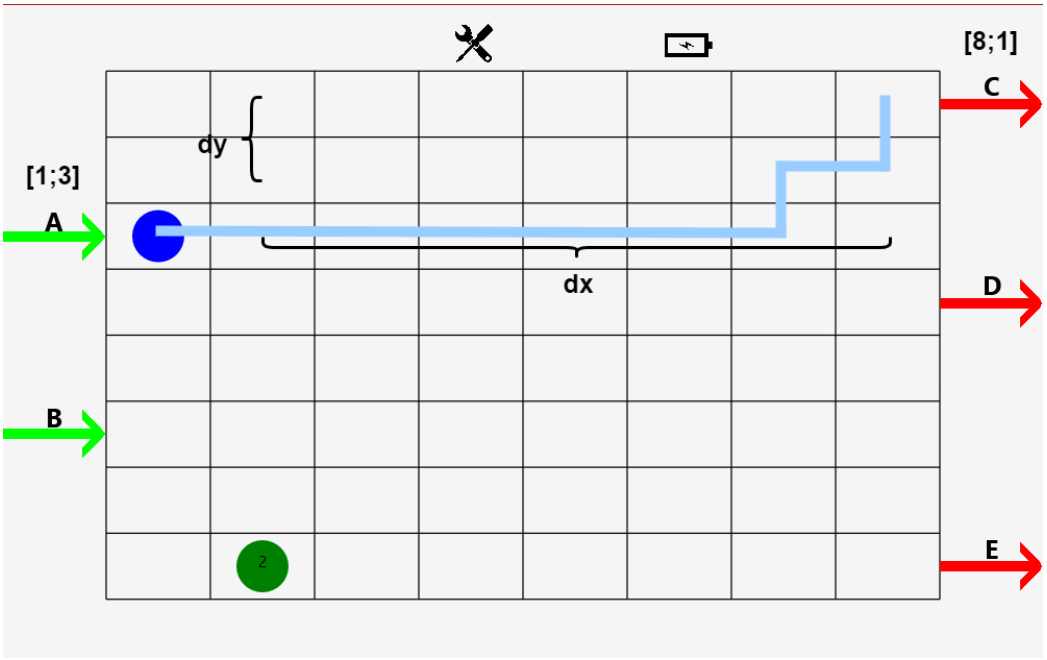
Pro komunikaci a výměnu dat mezi zařízeními je důležité, aby obě zařízení byla připojena ke stejné síti. Komunikace probíhala na privátní síti vytvořené pro tyto účely. Jako router je použit TP-Link a to konkrétně model kapesní Wi-Fi router TL-WR810N. Řídící software komunikuje přes tuto síť s reálným vozítkem přes Raspberry Pi 4. Toto zařízení má statickou adresu a proto v komunikačním protokolu je pevně nastavená tato IP adresa.

5.3.2 Výpočet optimální trasy

Pro nejrychlejší a nejefektivnější převoz zásilek je nutné vypočítat nejkratší cestu, kterou vozítko musí projet. K výpočtu je potřeba znát, v jaké pozici se vozítko aktuálně nachází a jaká je jeho cílová pozice. Poté je možné ze znalosti těchto dvou bodů vypočítat rozdíl jednotlivých souřadnic těchto bodů dx a dy. Princip výpočtu je založen na porovnání

velikostí těchto rozdílů. Dokud jeden z těchto rozdílů je větší než druhý, vozítko se pohybuje tímto směrem. Tento výpočetní algoritmus se tedy snaží kompenzovat větší rozdíl vozítka od cílového bodu. Na obrázku 5.13 je znázorněn pohyb vozítka z počáteční pozice (bod A) do cílové pozice (bod C).

Tento algoritmus je v kódu vytvořen přes tři podmínky. První podmínkou je situace, kdy dy rozdíl je nulový. Poté se vozítko pohybuje pouze horizontálně k cílovému bodu (po ose x). Tato podmínka se vykonává cyklicky do doby, dokud se aktuální pozice vozítka nerovná požadované pozici. V této smyčce se také v každém kroku rozlišuje, zda je aktuální hodnota dx menší nebo větší než nula. Přičemž když je dx menší, než nula je vozítko naváděno v ose x směrem doleva. Pokud je dx větší než nula, tak je pohyb naopak směrem doprava. Druhá podmínka je obdobná a je splněna tehdy pokud je dx rozdíl nulový. Tehdy se vozítko pohybuje vertikálně k cílovému bodu. Tato podmínka se také vykonává cyklicky do doby, dokud se aktuální pozice vozítka nerovná požadované pozici. Dále se v této smyčce rozlišuje, zda je rozdíl dy menší než nula. Poté se vozítko pohybuje v ose y směrem nahoru. Pokud je dy větší než nula, tak je pohyb naopak směrem dolu. Poslední podmínka je splněna, pokud rozdíl dx ani dy není nulový. Poté se vykonává cyklus, který je dokončen až když je aktuální pozice vozítka rovna cílové pozici. V této smyčce probíhá porovnávání jednotlivých velikostí rozdílů dx a dy. Pokud je absolutní hodnota dy větší než absolutní hodnota dx, pak se vozítko pohybuje nahoru či dolu vzhledem k porovnání y-nové souřadnice vozítka s y-ovou souřadnicí cílového bodu. V opačném případě, když je rozdíl dx větší než dy, tak se vozítko pohybuje doleva či doprava v porovnání x-ových souřadnic. Všechny vypočtené body trasy jsou uloženy do listu goto_Pos.



Obrázek 5.13 Pohyb vozítka

Pro výpočet trasy slouží vytvořená metoda GoTo(), která se nachází v souboru VehicleBase.cs. V této metodě je kromě algoritmu pro výpočet tras vytvořen Task, který umožňuje samotný pohyb vozítka na obrazovce aplikace. V tomto Tasku se nachází for smyčka, která postupně prochází všechny body uložené v listu goto_Pos. Podle toho, kde se vozítko nachází ve srovnání s každou cílovou pozicí, přiřadí hodnoty do čtyř směrových příkazů: GoAhead, GoBack, GoLeft, GoRight. Takže například pokud je souřadnice y vozidla menší než souřadnice y cílové pozice, GoAhead dostane hodnotu true, což znamená, že vozidlo se musí pohnout vpřed. Dále se zde rozděluje, zda se jedná o simulované či reálné vozítko. Pokud je vozítko simulované, definuje se proměnná step, která určuje, jakou vzdálenost vozítko urazí při každém kroku simulace. Smyčka for se opakuje čtyřikrát, aby bylo docíleno, že se vozítko ve vizualizaci posune z jednoho bodu na druhý. Vozítko se v tomto případě posouvá o 0,25 vzdálenosti, kterou potřebuje urazit mezi jedním a druhým bodem vypočtené trajektorie. V každém kroku se čeká krátkou pauzu, která reprezentuje plynulé posouvání vozítka po mapě. Poté se pomocí podmínek kontroluje, kterým směrem má vozítko jet, a podle toho se upravuje jeho pozice. Vždy po přesunu na jiný kód zde vozítko čeká po dobu 1 s, což reprezentuje načítání QR kódu a odesílání dat. Během jednoho pohybového cyklu jsou generovány události, , které se týkají začátku pohybu (StartMoving), ukončení pohybu (StopMoving) a dosažení cíle (OnTarget). Když se jedná o reálné vozítko, nejprve se kontroluje, zda je vozítko připojeno k síti. To se provádí pomocí vlastnosti comm.IsConnected třídy PhysicalVehicle. Následně se podle aktuální pozice vozítka, která byla získána z přijaté zprávy a cílového bodu nastavuje směr pohybu. Pokud je například x-ová souřadnice cílového bod větší, než aktuální sloupec QR kódu vozítko dostane příkaz, aby jelo doprava. Poté následuje vykreslování pohybu vozítka na mapě obdobným způsobem jako u simulovaného vozítka. Nakonec jako u simulovaného vozítka se testuje, zda vozítko dosáhlo cílové pozice. Pokud ano, vyvolá se událost OnTarget jinak se vyvolá událost StopMoving, která zastaví pohyb vozítka.

5.3.3 Řízení vozítek

Základem pro převoz zásilek z jednotlivých vstupních pozic je nalezení první volné zásilky ve frontě A a ve frontě B pomocí metody Where. Pokud je nalezena alespoň jedna volná krabice, pokračuje se ve vyhledávání volného vozítka v kolekci Vehicles. Když je nalezeno volné vozidlo, rozhoduje se, která zásilka bude jako první odbavena dle doby čekání ve vstupní frontě. Poté zásilka změní svůj stav z Free na Occupied a vozítku je přiřazen klon této zásilky pomocí metody Clone(). Pokud vozítku není přidělena žádná zásilka má status Free. Jinak je vozítko ve stavu Occupied a přesouvá se na předem stanovenou pozici podle toho jaký vstup má vozítku přidělená zásilka. Pohyb je vykonáván pomocí metody GoTo(), která je popsána výše. Pokud nejsou k dispozici žádné volné krabice nebo vozidla, čeká kód na krátkou dobu pomocí metody Task.Delay(300), než se znovu zkontroluje dostupnost.

Dále je vozítko řízeno dle událostí. Když je vozítko přidáno do kolekce vozítek, nastavuje se odběratel událostí pro toto vozítko, který reaguje na události související s jeho pohybem. Každé vozítko má několik možných událostí během svého pohybu, včetně začátku pohybu, zastavení pohybu a dosažení cíle. Každá z těchto událostí vyvolá jiné akce, které ovlivňují stav a chování vozítka. Když vozítko dosáhne svého cíle, což znamená, že se aktuální pozice vozítka rovnají požadovaným kontroluje se, zda se nachází na jednom z cílových bodů.

Prvním cílovým bodem je, že se vozítko nachází na vstupní pozici, kde má naložit zásilku. Když se vozítko na tomto bodě nachází, nastává proces naložení zásilky a aktualizace stavu vozítka na Occupied. Dále podle toho jakou má zásilka přidělenou destinaci ji vozítko přesune do cílové destinace. V průběhu pohybu vozítka jsou také aktualizovány informace o obsazenosti polí na mapě. Když vozítko začne nebo zastaví pohyb, označí se pole, která vozítko opouští nebo na která se přesouvá, jako obsazená nebo volná. V aplikaci jsou obsazené políčka vybarvena světle červenou barvou. Když vozítko potřebuje na pole, které je obsazené, tak čeká do doby dokud není požadované pole volné. Mapa aktuálně obsazených polí je realizována přes boolovské dvojrozměrné pole, které obsahuje 64 položek reprezentující matici 8 x 8 polí. Do proměnných Originx/y a Targetx/y jsou uložené souřadnice. Origin představují aktuální pozici vozítka a Target následující souřadnice pole na, které se má vozítko přesunout. Pokud je detekován nějaký směr pohybu, cílové souřadnice vozidla jsou aktualizovány podle tohoto směru pohybu. Tedy pokud například vozítko jede dopředu, jeho cílová y-nová souřadnice se snižuje o jedničku. Poté je vyvolána událost StartMoving, což označuje začátek pohybu vozidla a počáteční souřadnice Origin jsou v každé iteraci smyčky aktualizovány na jeho aktuální polohu. Když je vyvolána událost StartMoving, jsou aktualizovány informace o obsazenosti polí na mapě. Příznaky obsazení na políčkách, ze kterých vozidlo odjíždí a na která míří, jsou nastaveny na hodnotu true. A když je vyvolána událost StopMoving, příznaky obsazení na těchto políčkách jsou aktualizovány tak, aby odpovídaly stavu vozítka, které zastavilo. To znamená, že pozice, ve které se vozítko nachází je nastavena na true a ze které vyjelo je nastaven na false.

Druhým cílovým bodem je, že se vozítko nachází na výstupní pozici, kde je zásilka vyložena. Zde vozítko změní svůj stav na Unloading a následně na Free. Zásilka se poté stane doručenou Transferred a je z vozítka odebrána. Když se vozítko stane volným, sleduje se stav zásilek ve frontách A a B. Po každé události na cílovém bodě vozítko zkontroluje, zda ve frontách nejsou žádné volné krabice. Pokud nejsou vozítko již nenese žádnou zásilku, vozítko se vrátí na výchozí pozici, aby bylo připraveno na další úkoly. Když vozítko ve frontách nalezne čekající volnou zásilku stane se Occupied a vydá se pro ní a postup je opakován.

6. TESTOVÁNÍ

Následující kapitola pojednává o testování realizované desktopové aplikace. V této kapitole je popsán průběh jednoho z testovacích cyklů, kde vozítka převážejí zásilku a dochází ke kooperaci mezi nimi.

6.1 Trasování

Testování aplikace probíhalo na již vytvořeném reálném vzorku, kde při testování byla vytvořena reálná mapa o rozměrech 4 x 4 QR kódů. Důvodem použití zmenšené verze mapy byly omezené testovací prostory, které poskytl zadavatel. Pro testování byly použity QR kódy [1,1] až [4,4]. Při testování byla v aplikaci taktéž testována simulovaná vozítka a kooperace mezi nimi, kde tato vozítka se pohybovala po mapě o plné velikosti 8 x 8 QR kódů.

Pro testování vozítek bylo zapotřebí vytvoření konkrétních vozítek v simulaci. Vozítka byla do simulace přidána přes tlačítka Add sim a Add real na obrazovce Setting – Communication, kde byl uživatel nejdříve vyzván k autentizaci. Pro úspěšné přihlášení bylo nutné zadat uživatelské jméno „admin“ a heslo „120knedLiku“. Vozítka jsou do seznamu řazena s identifikačním číslem od jedné. Testování probíhalo se dvěma simulovanýma vozítka a s jedním reálným. Při přidání vozítek se na hlavní obrazovce nakreslila vozítka s odpovídajícími souřadnicemi v podobě kruhů. Simulovaná vozítka se na mapě řadila na pozice [1,8] a [2,8] a reálné vozítko bylo umístěno nejprve mimo mapu a po obdržení pozice dle aktuálně načteného QR kódu na pozici [1,1]. Barva vozítka je určena dle jeho aktuálního stavu, kdy po přidání je vozítko vždy ve stavu Free a má zelenou barvu.

Na stránce Request byly manuálně přidány čtyři zásilky do vstupních front A a B přes tlačítka Add A a Add B. Zásilky jsou do fronty řazeny s identifikačním číslem od nuly. Nejprve byly přidány dvě zásilky na vstup A a další dvě na vstup B se stavem zásilky Free. Čas, jak dlouho zásilka čeká ve frontě se počítá od jejího přidání do fronty. Destinace se zásilkám přiřazuje náhodně dle výstupních destinací C, D a E.

První volná zásilka, která čeká ve frontě nejdéle se přiřadila prvnímu volnému vozítku. V tomto případě se jednalo o reálné vozítko, jelikož bylo přidáno do simulace jako první a přiřadila se mu zásilka ze vstupu A. Ihned po přiřazení zásilky k vozítku se změnil stav jak zásilky, tak vozítka na Occupied. Změna stavu vozítka je znázorněna modrou barvou, jelikož je obsazené. Následující dvě volné zásilky byly přiřazeny simulovaným vozítkům. Poslední zásilka čekající ve frontě má status Free, dokud nebude opět volným vozítkem obsazená. Stav jednotlivých zásilek bylo možné sledovat na obrazovce Request.

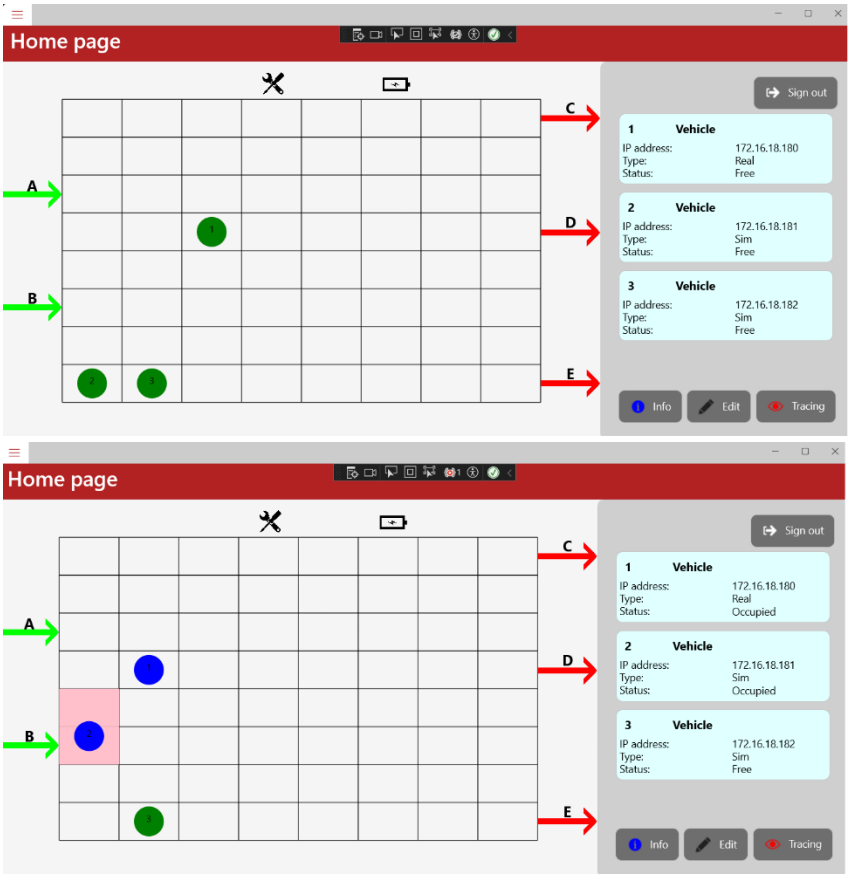
Jakmile se vozítko stalo obsazeným, vyrazilo na vstupní pozici dle zásilky, která mu byla přiřazena. Na mapě také bylo možné sledovat zašedlá políčka, která znázorňovala,

obsazená pole, které si dané vozítko obsadilo pro svůj pohyb. Jednalo se o pole, na kterém se vozítko právě nacházelo a o následující pole, kam mělo naplánováno další pohyb. Naložení zásilky je indikováno změnou stavu vozítka na To Loading a zpět na Occupied což bylo v simulaci detekováno změnou barvy na oranžovou a zpět na modrou. Po naložení zásilky vozítko následně jelo do destinace, kam měla být zásilka odvezena. Mezi tím se i ostatní vozítka dostavila na vstupní pozice. Pokud bylo políčko obsazené, tak vozítko čekalo do doby, než se toto pole uvolnilo a pokračovalo dál po vypočtené trase. U reálného vozítka nemohlo být dosaženo výstupní destinace a odvezení zásilky, jelikož měl omezený počet QR kódu. Toto vozítko dostávalo příkazy postupně dle nahlášení své pozice přes načtený QR kód.

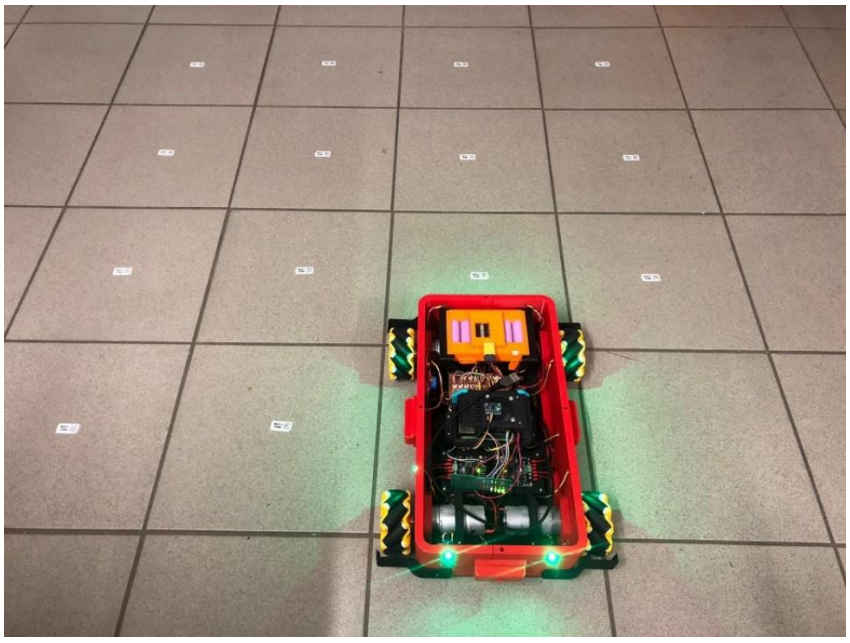
Po dosažení cílové destinace druhého vozítka se změnil status zásilky na Transferred (doručeno) a zastavil se čas čekání. Vozítko změnilo svůj stav na UnLoading a následně na Free. V simulaci toto bylo znázorněno přechodem z oranžové barvy na zelenou. V okamžiku, kdy se vozítko stalo opět Free a bylo prvním volným vozítkem v seznamu byla mu přiřazena poslední a také nejdéle čekající volá zásilka. Z obou se staly Occupied a vozítko opět změnilo barvu na modrou a jelo zpět na vstupní destinaci pro zásilku. Poslední vozítko, které odvezlo zásilku a stalo se opět volným se vrátilo zpět na výchozí pozici, jelikož už ve frontě nebyla žádná volná zásilka pro odbavení. Veškerá oznámení od vozítka, že dojelo do cíle a jakou vezlo zásilku bylo možné sledovat na obrazovce Log.

Při testování dále bylo zjištěno, že v simulaci může nastat varianta, že vozítka jedou proti sobě a jejich trasa vede přes stejný QR kód. Tedy například vozítko č.1 je na pozici [2,2] jede na [2,3] a vozítko č.2 je na pozici [2,3] a jede na [2,2]. V tento moment vznikne tzv. Deadlock a vozítka se zastaví a již nepokračují v pohybu, jelikož obě čekají až se jejich následující pozice uvolní. Tento problém by se částečně dal vyřešit tak, že by obě vozítka odbočily například doprava. Nicméně by pak mohl nastat opět stejný problém, pokud by se na vedlejší pozici nacházelo jiné vozítko. Zadavatel se však ještě nerozhodl, jak se tento problém bude řešit, tudíž bylo zadavateli předloženo pouze teoretické řešení. Všechna vozítka v aplikaci bylo možné vidět na hlavní obrazovce v podobě seznamu a bližší informace o nich na obrazovce Status. U simulovaných vozítek jsou informace o napětí a proudu generovány náhodně a u reálného vozítka jsou tyto informace navázány na data, která řídící software obdrží od reálného vozítka. V simulaci není řešena varianta, že pokud má vozítko slabou baterii, tak je navigováno do dokovací stanice. Nicméně pro sledování stavu baterie slouží status vozítka.

Při testování dále bylo zjištěno, že je potřeba rozšířit komunikaci o diagnostické funkce a handshaking, aby se zvýšila odolnost pro zjištění ztráty spojení.



Obrázek 6.1 Testování vizualizační aplikace



Obrázek 6.2 Testování aplikace na reálném vozítku [38]

7. ZÁVĚR

Cílem diplomové práce bylo vhodně navrhnout a následně realizovat řídící a vizualizační software pro autonomní robotické vozítko. Nejprve proběhlo seznámení se s návrhem robotických vozítek s ohledem na softwarové vlastnosti a použití těchto vozítek v praxi. Poté byl vybrán framework vhodný pro realizaci řídícího a vizualizačního softwaru. Pro danou úlohu bylo vybráno vývojové prostředí Visual studio, které umožňuje vytvořit aplikaci na frameworku .NET MAUI. Tento framework umožňuje vytvářet mobilní a desktopové aplikace. Byl vybrán z důvodu všestrannosti v použití, jelikož při tvorbě aplikací na této platformě není nutné tvořit odlišné kódy pro odlišné operační systémy. Vizualizační software tedy ve výsledku slouží jako uživatelské rozhraní pro zobrazování dat a monitorování pohybu vozítek skrze aplikaci. Řídící software slouží v aplikaci pro přijímání a zpracování dat od vozítek s cílem následného řízení jejich pohybu.

Pro přenos dat byl vybrán protokol UDP, kde komunikačním médiem je v této aplikaci Wi-Fi, jelikož je nutné s vozítky komunikovat bezdrátově. Bylo nutné navrhnout, jak bude vypadat formát dané zprávy, aby byla komunikace přehledná. Proto byl vytvořen formát zprávy ve tvaru, typ zprávy, číslo vozítka a konkrétní zpráva s ohledem na typ zprávy.

Řídící software byl vytvořen dle požadavků zadavatele s ohledem na konkrétní použití aplikace. Tedy aplikace obsahuje reálná i simulovaná vozítka, kterým bylo zapotřebí vhodně plánovat trasy a zajistit tak plynulý pohyb při přepravě zásilek. Dále řídícího softwaru zajišťuje komunikaci s reálným vozítkem nezbytnou pro přijímání dat a odesílání pokynů k řízení. Realizace reálných vozítek z hlediska hardwaru nebyla součástí této práce. Tento software byl testován na již vytvořeném prototypním vzorku, který je součástí jiné diplomové práce [38].

Vizualizační software v aplikaci slouží k monitorování pohybu zásilek a vozítek v reálném čase. Uživateli poskytuje možnost nejen sledovat aktuální polohu vozítek a stav zásilek prostřednictvím vytvořené mapy, ale také spravovat chod aplikace. Jednou z klíčových funkcí je možnost přidávat nová vozítka a zásilky do aplikace, čímž je ovlivněna logistika přepravy. Dále aplikace umožňuje různé nastavení jako například změna jazyka a konfigurace vozítek.

Výsledná forma zhotovení práce byla vytvořena s ohledem na požadavky zadavatele. Práce by však mohla být rozšířena o některá další vylepšení zlepšující funkci celé aplikace. Aplikace by mohla by rozšířena o databázi, kam by se mohla ukládat jednotlivá data týkající se zásilek či vozítek. Dále by mohl být více propracován pohyb vozítek s plánováním tras. Tedy například v aplikace by mohla být použita možnost vyhýbání se vozítek na mapě namísto čekání na uvolnění pozice což by mohlo minimalizovat vzniklé deadlocky. Toto opatření by mohlo zefektivnit a urychlil převoz zásilek.

Literatura

[1] REBOK, Tomáš. Protokoly transportní vrstvy a jejich kategorizace, transportní protokol ARTP. Online. In: . S. 6. Dostupné z: https://www.fi.muni.cz/~xrebok/DOCs/PUBs/Olomouc05_1.pdf. [cit. 2023-12-11].

[2] Handshake. Online. Wikipedia. 2021. Dostupné z: https://cs.wikipedia.org/wiki/Handshake. [cit. 2023-12-11].

[3] NOVOTNÝ, Vojtěch. Úvod do počítačových sítí. Online. In: . S. 194. Dostupné z: https://www.zamekkurim.cz/security/Pocitacove%20site%20-%20Novotny/Pocitacove%20site%20skripta%20-%20Novotny.pdf. [cit. 2023-12-11].

[4] Transmission Control Protocol. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2023. Dostupné z: https://cs.wikipedia.org/wiki/Transmission_Control_Protocol. [cit. 2023-12-11].

[5] Services and Segment structure in TCP. Online. Geeks for geeks. Dostupné z: https://www.geeksforgeeks.org/services-and-segment-structure-in-tcp/. [cit. 2023-12-11].

[6] TCP (Transmission Control Protocol) Segments and Fields. Online. FIBERBIT. Dostupné z: https://fiberbit.com.tw/tcp-transmission-control-protocol-segments-and-fields/. [cit. 2023-12-11].

[7] Transmission control protocol (TCP). Online. Imperva. Dostupné z: https://www.imperva.com/learn/ddos/tcp-transmission-control-protocol/. [cit. 2023-12-11].

[8] Description of Windows TCP features. Online. Microsoft. 2023. Dostupné z: https://learn.microsoft.com/en-us/troubleshoot/windows-server/networking/description-tcp-features. [cit. 2023-12-11].

[9] User Datagram Protocol. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2023. Dostupné z: https://cs.wikipedia.org/wiki/User_Datagram_Protocol. [cit. 2023-12-11].

[10] TCP vs UDP: What’s the Difference and Which Protocol Is Better? Online. Avast. 2023. Dostupné z: https://www.avast.com/c-tcp-vs-udp-difference. [cit. 2023-12-11].

[11] TCP vs. UDP: Jaký je rozdíl? Online. LifeSize. 2017. Dostupné z: https://www.lifesize.com/cs/blog/tcp-vs-udp/. [cit. 2023-12-11].

[12] MODBUS TCP. Online. FCCPS Průmyslové počítače a komunikace průmyslové systémy. Dostupné z: https://www.fccps.cz/modbus-tcp?fbclid=IwAR3WGXkrNmiib2q7M4lfOmR5iDrb5QfClvv86rhif4OmJFe-iVd4dTo_-Hs. [cit. 2023-12-18].

[13] Understanding Modbus Serial and TCP IP. Online. ProSoft technology. 2023. Dostupné z: https://www.prosoft-technology.com/Landing-Pages/Protocol/Modbus-and-Modbus-TCP-

Protocol?fbclid=IwAR0AJdwnZHaFM-Gepb5n_3c02Uyl_Je_zhWGNObwaDQnBVyEOU9lAtWO-QE. [cit. 2023-12-18].
[14] RONEŠOVÁ, Ing. Andrea. Přehled protokolu MODBUS. Online. In: . 2005, s. 20. Dostupné z: Přehled protokolu MODBUS. [cit. 2023-12-18].
[15] MODBUS/TCP. Online. Netio. 2023. Dostupné z: https://www.netio-products.com/cs/slovnik/modbustcp?fbclid=IwAR1OEM7YsAWspr2QNNwTToUmy7ZsDYGMz5IStPLdW4RJEPa1d1A6Oj77Ckg. [cit. 2023-12-18].
[16] What's the difference between Modbus ASCII and Modbus RTU? Online. ProSoft technology. 2019. Dostupné z: https://www.prosoft-technology.com/knowledge-base/Protocols/Modbus/Whats-the-difference-between-Modbus-ASCII-and-Modbus-RTU. [cit. 2023-12-18].
[17] Podrobný popis protokolu Modbus TCP s příklady příkazů. Online. Ipc2U.cz. 2023. Dostupné z: https://ipc2u.cz/blogs/news/podrobny-popis-protokolu-modbus-tcp-s-priklady-prikazu. [cit. 2023-12-18].
[18] Protokol MQTT: komunikační standard pro IoT. Online. Root.cz. 2016. Dostupné z: https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/. [cit. 2023-12-21].
[19] What is MQTT and Why it is Important for the Internet of Things. Online. Akenza.io. 2023. Dostupné z: https://blog.akenza.io/what-is-mqtt?utm_term=&utm_campaign=Dynamic+Search+Campaign+-+EU&utm_source=google&utm_medium=cpc&hsa_acc=7184580909&hsa_cam=19248438136&hsa_grp=144426835677&hsa_ad=641301872801&hsa_src=g&hsa_tgt=dsa-19959388920&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gad_source=1&gclid=CjwKCAiA-P-rBhBEEiwAQEXhHxMGW4_ClkYvrXRX5-g9ah6BIhCvQLZySJ5dj6N6UxsGRjFRiz1PRhoCcE8QAvD_BwE. [cit. 2023-12-21].
[20] Limitations of MQTT. Online. Medium. 2022. Dostupné z: https://blog.iotify.io/limitations-of-mqtt-bd61e8150bca. [cit. 2023-12-21].
[21] KLEMENT, Milan. Technologie bezdrátových sítí základní principy a standardy. Online. In: . Olomouc, 2019, s. 57. Dostupné z: https://www.pdf.upol.cz/fileadmin/userdata/PdF/katedry/ktiv/Studijni_materialy/Klement/2019/TBS_2019_skripta.pdf. [cit. 2023-12-21].
[22] SOCHOR, Tomáš. POČÍTAČOVÉ SÍTĚ 1 URČENO PRO VZDĚLÁVÁNÍ V AKREDITOVANÝCH STUDIJNÍCH PROGRAMECH. Online. In: . Ostrava, 2014. Dostupné z: https://physics.ujep.cz/~jkrejci/ZPP/Materi%E1ly/P7_0_16_Opora.pdf. [cit. 2023-12-21].
[23] ZANDL, Patrick. Bezdrátové sítě WiFi: praktický průvodce. Brno: Computer Press, 2003. ISBN 80-7226-632-2

[24] Bezdrátové technologie. Online. ANTALOVÁ DAGMAR - BIBS. 2011. Dostupné z: https://antalova-bibs.webnode.cz/bezdratove-technologie/. [cit. 2023-12-21].
[25] IEEE Standard 754 Floating Point Numbers. Online. Geeksforgeeks. 2020. Dostupné z: https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/. [cit. 2024-04-06].
[26] What is .NET MAUI? Online. Microsoft. 2023. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui. [cit. 2023-12-11].
[27] MIKEŠ, Tomáš. INTEGRACE DOTVVM DO .NET MAUI. Online, BAKALÁŘSKÁ PRÁCE, vedoucí Ing. JIŘÍ HYNEK, Ph.D. Brno: VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ, 2022. Dostupné z: https://dspace.vutbr.cz/bitstream/handle/11012/211076/final-thesis.pdf?sequence=1&isAllowed=y. [cit. 2023-12-11].
[28] Xamarin Versus .NET MAUI. Online. Syncfusion. 2023. Dostupné z: https://www.syncfusion.com/blogs/post/xamarin-versus-net-maui.aspx. [cit. 2023-12-11].
[29] ŠVAPOR, Bc. Dávid. .NET MAUI MULTIPLATFORM USER INTERFACE FOR KIMAI. Online, DIPLOMOVÁ PRÁCE, vedoucí Ing. DANIEL DOLEJŠKA. Brno: VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ, 2023. Dostupné z: https://dspace.vutbr.cz/bitstream/handle/11012/211934/final-thesis.pdf?sequence=1&isAllowed=y. [cit. 2023-12-11].
[30] Model-View-ViewModel (MVVM). Online. Microsoft. 2022. Dostupné z: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm. [cit. 2023-12-11].
[31] Mvvm: model-view-viewmodel. Online. DotNetportal. 2009. Dostupné z: https://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel. [cit. 2023-12-11].
[32] What Is MVVM Architecture? Online. Builtin. 2023. Dostupné z: https://builtin.com/software-engineering-perspectives/mvvm-architecture?fbclid=IwAR1NxH5FKlSFG9mDy4Nt6c8Y3EJvZlTayTVNzJ7JNDvVUVxkJwjQC5N-ajA. [cit. 2023-12-11].
[33] ObservableCollection<T> Třída. Online. Microsoft. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/api/system.collections.objectmodel.observablecollection-1?view=net-7.0. [cit. 2023-12-11].
[34] Binding mode. Online. Microsoft. 2023. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/data-binding/binding-mode?fbclid=IwAR2SQf6PJ56c-uKQyQPO23EyL7oU1ldb2MqwyLvaqZA8ouu_KvZRmXz8x9s. [cit. 2023-12-11].

[35] What is Entity Framework? Online. Entity Framework Tutorial. 2023. Dostupné z: https://www.entityframeworktutorial.net/entityframework6/what-is-entityframework.aspx#google_vignette. [cit. 2023-12-27].
[36] Database First vs Code First in Entity Framework. Online. Medium. 2017. Dostupné z: https://levelup.gitconnected.com/database-first-vs-code-first-in-entity-framework-fd786fa1dfd6. [cit. 2023-12-27].
[37] Tracking free icon. Online. In: Flaticon. Dostupné z: https://www.flaticon.com/free-icon/tracking_7199862?term=tracking+systems&page=1&position=11&origin=search&related_id=7199862. [cit. 2024-04-21].
[38] KOLÁČNÝ, Michal. Hardwarová platforma pro propagační robotické vozítko [online]. Brno, 2024 [cit. 2024-05-09]. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/160031. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Miroslav Jirgl.

SEZNAM PŘÍLOH

Příloha A - Elektronická příloha

Obsah přílohy CD:
ARField.zip

- Diplomová práce
  - xlazni09.pdf
- Zdrojový soubor – ARField
  - ARField.sln
- Vývojový diagram – drawio
  - vyvoj_diagram_vizualizace.drawio
  - blok_shem_projekt.drawio