

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-104378-92671

**GPS SYSTÉM PRE KYNOLÓGICKÉ ZÁCHRANNÉ
ZLOŽKY**

DIPLOMOVÁ PRÁCA

2024

Bc. Tomáš Vago

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-104378-92671

**GPS SYSTÉM PRE KYNOLOGICKÉ ZÁCHRANNÉ
ZLOŽKY**
DIPLOMOVÁ PRÁCA

Študijný program: Aplikovaná mechatronika a elektromobilita

Názov študijného odboru: Informatika

Školiace pracovisko: Ústav informatiky a matematiky

Vedúci záverečnej práce: Ing. Peter Čapák, PhD.

Bratislava 2024

Bc. Tomáš Vago



ZADANIE DIPLOMOVEJ PRÁCE

Študent: **Bc. Tomáš Vago**
ID študenta: 92671
Študijný program: aplikovaná mechatronika a elektromobilita
Študijný odbor: kybernetika
Vedúci práce: Ing. Peter Čapák, PhD.
Vedúci pracoviska: Ing. Ján Cigánek, PhD.

Názov práce: **GPS systém pre kynologické záchranné zložky**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Cieľom práce je vytvoriť systém sledovania polohy psovodov a psov pri pátracích akciách.
Úlohy:

1. Oboznámte sa s dostupnými riešeniami sledovania polohy osôb a zvierat pomocou GNSS.
2. Vyberte vhodný HW pre umiestnenie na zvieratá a osoby.
3. Realizujte vhodné riešenie serverovej časti systému.
4. Realizujte klientskú časť systému.
5. Overte funkčnosť.

Termín odovzdania diplomovej práce: 10. 05. 2024
Dátum schválenia zadania diplomovej práce: 28. 02. 2024
Zadanie diplomovej práce schválil: prof. Ing. Vladimír Kutiš, PhD. – garant študijného programu

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná mechatronika a elektromobilita
Autor:	Bc. Tomáš Vago
Diplomová práca:	GPS systém pre kynologické záchranné zložky
Vedúci záverečnej práce:	Ing. Peter Čapák, PhD.
Miesto a rok predloženia práce:	Bratislava 2024

Táto diplomová práca sa zaobráva vývojom a implementáciou GPS systému pre kynologické záchranné jednotky, s cieľom zlepšiť koordináciu a efektivitu vyhľadávacích operácií. V súčasnosti používané systémy čelia rôznym výzvam, najmä cenovou dostupnosťou a zložitosťou používania. Na základe analýzy existujúcich riešení bola navrhnutá a vyvinutá nová platforma, ktorá kombinuje pokročilé technológie GNSS (Globálny navigačný satelitný systém) a moderné metódy spracovania dát, čím poskytuje presnejšie a rýchlejšie sledovanie polohy v reálnom čase.

Systém bol testovaný v rôznych prostrediach a situáciách, čo potvrdilo jeho schopnosť zvýšiť presnosť a znižovať časy reakcie pri pátracích akciách. Táto práca prispieva k lepšiemu porozumeniu možností integrácie moderných technológií do kynologických záchranných operácií a predstavuje dôležitý krok k zvyšovaniu bezpečnosti a účinnosti záchranných tímov. Výsledky a metodológie uvedené v tejto práci ponúkajú východisko pre ďalší výskum a rozvoj v oblasti aplikovaných záchranných technológií.

Kľúčové slová: GPS, IoT, mobilná aplikácia, ReactNative, SpringBoot

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Mechatronics and Electromobility
Author:	Bc. Tomáš Vago
Master's thesis:	GPS system for canine rescue services
Supervisor:	Ing. Peter Čapák, PhD.
Place and year of submission:	Bratislava 2024

This thesis focuses on the development and implementation of a GPS system for canine rescue units aimed at improving coordination and efficiency in search operations. Current systems face various challenges, including inadequate accuracy and delays in tracking the movements of rescue teams and their canine companions. Based on an analysis of existing solutions, a new platform was designed and developed that combines advanced GNSS (Global Navigation Satellite System) technologies and modern data processing methods, providing more accurate and faster real-time location tracking.

The system was tested in various environments and situations, confirming its ability to increase accuracy and reduce response times during search missions. This work contributes to a better understanding of the possibilities of integrating modern technologies into canine rescue operations and represents an important step towards enhancing the safety and effectiveness of rescue teams. The results and methodologies presented in this thesis offer a foundation for further research and development in the field of applied rescue technologies.

Keywords: GPS, IoT, mobile application, ReactNative, SpringBoot

Podakovanie

Toto podakovanie by som chcel venovať všetkým tým, ktorým vďačím za to, že z boxeristu, ktorý mal školu na poslednom mieste, sa stal niekto, kto do dotiahol až sem. Bolo by ich ľahké spomenúť všetkých.

Ďalší, komu by som chcel špeciálne podakovať je môj učiteľ matematiky zo základnej školy Štefan Róža, ktorý kým žil, dával študentom viac, ako len matematické vedomosti. Učil nás správnym hodnotám. Pán učiteľ, budem Vám vďačný nie len za to, že ste ma na základnej škole učili matematiku, vďaka ktorej som sa dokázal popasovať aj s tou vysokoškolskou. Ďakujem Vám najmä za to, že ste ukazovali všetkým svojim študentom, ako byť správnym človekom.

Podakovanie patrí aj veľa pedagógom z našej univerzity. Všetkým tým, ktorí uprednostňujú ľudskosť a spravodlivosť počas ich výučby a tak pôsobia na študentov podobným vplyvom, ako pán spomenutý v odseku vyššie.

A v neposlednom rade podakovanie patrí aj môjmu vedúcemu práce za trpezlivosť a pomoc. Spomínam na veľa zábavy a zážitkov, ktoré sme na predmetoch s Vami zažili.

Obsah

Úvod	1
1 Teoretická časť diplomovej práce	2
1.1 Úvod do sveta webových aplikácií, mikroprocesorov a senzorov	2
1.1.1 Mikroprocesory v mobilných aplikáciách	2
1.1.2 Mikroprocesory v IoT a embedded systémoch	2
1.1.2.1 Porovnanie vlastností a využitie	3
1.1.2.2 Mikroprocesory pre špeciálne použitia	3
1.1.2.3 Aplikačné scenáre a budúci vývoj	4
1.1.2.4 ESP32: podrobný popis a využitie	4
1.1.3 Senzory v mobilných aplikáciách	5
1.1.3.1 GPS Senzory: podrobný popis a využitie	6
1.1.3.2 GPS modul NEO-M8	7
1.2 Dostupné technológie pre vývoj webových aplikácií	8
1.2.1 Serverová časť - backend	8
1.2.2 Užívateľská časť - frontend	12
1.2.3 Dostupné frameworky	15
1.2.4 Expo: Platforma pre vývoj React Native aplikácií	16
1.3 Haversinova funkcia : význam a využitie v geografických výpočtoch	17
1.4 Úvod do problematiky záchranárskej kynológie s použitím gps	18
1.4.1 Prepátravanie pomocou rojníc	19
1.4.2 Sektorové prehľadávanie	19
2 Ciele práce	21
2.1 Oboznámenie sa s dostupnými riešeniami sledovania polohy osôb a zvierat pomocou GNSS.	21
2.2 Zvolenie vhodného HW pre umiestnenie na zvieratá a osoby.	22
2.3 Realizácia vhodného riešenia serverovej časti systému.	22
2.4 Realizácia klientskej časti systému.	22
2.5 Overenie funkčnosti	22
3 Praktická časť	24
3.1 Funkcionálne požiadavky	24
3.2 Nefunkcionálne požiadavky	24
3.3 Kód mikročipu ESP32	25

3.4	Prepojenie mikročipu ESP32 a gps modulu NEO M8N	27
3.5	Návrh a popis architektúry databázy	28
3.6	Autorizácia a autentifikcia užívateľa	31
3.7	Prihlásenie užívateľa a generovanie JWT tokenu	34
3.8	Vytváranie novej akcie na strane servera	35
3.9	Získavanie akcií pre konkrétneho užívateľa	36
3.9.1	Hlavný komponent frontendovej časti	38
3.10	Spresnenie lokalizácie pomocou akcelerometra	39
3.11	Parsovanie JWT tokenu	40
3.12	Controllery na strane užívateľa	41
3.13	Získavanie polohy na strane užívateľa	45
3.14	Pomocné služby na frontendovej časti	47
3.15	Ukážka hlavných častí mobilného systému	50
Záver		54
Zoznam použitej literatúry		55

Zoznam obrázkov a tabuliek

Obrázok 1	Ktorý mikroprocesor zvoliť?	3
Obrázok 2	Procesor ESP32	5
Obrázok 3	senzor získavania polohu NEOM8	7
Obrázok 4	GPS obojok garmin TT15X	21
Obrázok 5	Obojok DOG GPS X30TB	23
Obrázok 6	Zapojenie ESP32	27
Obrázok 7	GPS obojok pre psa	28
Obrázok 8	ER diagram serverovej časti	30
Obrázok 9	Screen pre prihlásenie	50
Obrázok 10	Screen pre zmenu profilových údajov	51
Obrázok 11	Postup vytvárania nových akcii a sektorov	52
Obrázok 12	Priradovanie sektorov a zobrazovanie sektorov na akcii	53

Úvod

V posledných rokoch integrácia moderných technológií do záchranných operácií výrazne zvýšila ich efektivitu. Medzi tieto technológie patrí GPS, ktorá zohráva kľúčovú úlohu, najmä v koordinácii kynologických jednotiek zapojených do pátracích operácií. Napriek pokrokom súčasné systémy stále čelia výzvam sledovania psovodov v teréne a efektívneho riadenia psovodov a psov počas pátracích akcií. Táto diplomová práca si kladie za cieľ riešiť tieto problémy vývojom GPS sledovacieho systému určeného pre kynologické záchranné jednotky. Systém je navrhnutý tak, aby zlepšil operačnú koordináciu a časy reakcie, čím sa zvyšujú šance na úspešné záchrany. Práca je štruktúrovaná do teoretických základov, návrhu a implementácie systému, nasledované testovaním a hodnotením vyvinutého systému.

1 Teoretická časť diplomovej práce

1.1 Úvod do sveta webových aplikácií, mikroprocesorov a senzorov

V súčasnej ére digitálnej transformácie mobilné aplikácie prenikajú do každej sféry nášho života, či už ide o zdravotníctvo, vzdelávanie, priemysel alebo osobné použitie. Jednou z kľúčových technológií, ktoré umožňujú mobilným aplikáciám byť tak univerzálne užitočnými, sú mikroprocesory a senzory. Tieto komponenty rozširujú funkčnosť mobilných zariadení tým, že im umožňujú zbierať a spracovať údaje z ich okolia v reálnom čase.

1.1.1 Mikroprocesory v mobilných aplikáciách

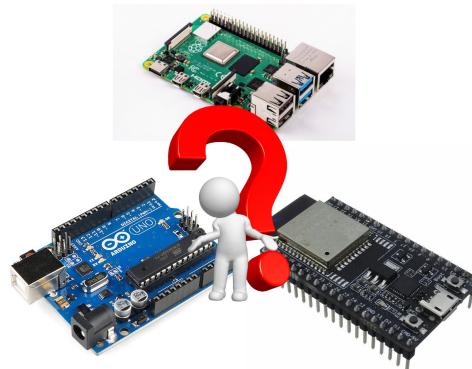
Mikroprocesory sú mozgom každého smartfónu, kde spracúvajú príkazy a vykonávajú programy. Sú zodpovedné za výkon aplikácií, ich multitasking a celkovú odozvu systému. Vývojári aplikácií ich využívajú na spracovanie komplexných algoritmov, od grafických reprezentácií po umeleckú inteligenciu a strojové učenie, čo umožňuje aplikáciám ponúkať personalizované a interaktívne užívateľské rozhrania.

1.1.2 Mikroprocesory v IoT a embedded systémoch

V tejto časti sa budeme podrobne zaoberať mikroprocesormi podobnými ESP32, ktoré sa využívajú v rámci aplikácií Internetu vecí (IoT) a embedded systémov. Mikroprocesory sú základom mnohých moderných technologických riešení, umožňujúc inteligentné správanie, komunikáciu s internetom a interakciu s fyzickým svetom prostredníctvom senzorov a aktuátorov.

- **Arduino platforma**, známa svojou užívateľsky prívetivou vývojovou prostredím a bohatou komunitou, zahrňa viaceré mikrokontroléry ako Arduino Uno, Mega a Nano. Tieto dosky sú ideálne pre začiatočníkov a vzdelávacie projekty vďaka svojej jednoduchosti a rozsiahlemu množstvu dostupných rozšírení a knižníc. Arduino je často využívané v hobby elektronike, prototypovaní a vzdelávacích projektoch zameraných na základy programovania a elektroniky [1].
- **Raspberry Pi**, hoci primárne jednodoskový počítač, sa často používa v embedded projektoch a IoT aplikáciách. Ponúka plnohodnotné operačné systémy ako Linux a Windows IoT, ktoré umožňujú beh komplexných aplikácií a spracovanie veľkých objemov dát. Raspberry Pi je vhodné pre náročnejšie výpočtové úlohy, serverové aplikácie v IoT, ako sú systémy pre správu obsahu a analýzu dát v reálnom čase [2].

- **Particle Photon a Electron**, rozširujú koncept IoT integráciou s Particle Cloud, čo umožňuje jednoduché a bezpečné pripojenie zariadení k internetu a ich správu na diaľku. Tieto mikrokontroléry sú ideálne pre komerčné IoT aplikácie, ktoré vyžadujú spoľahlivé cloudové služby, ako sú monitorovanie zariadení, zber dát a ovládanie zariadení v rozsiahlych sietach [3].



Obr. 1: Ktorý mikroprocesor zvolit?

1.1.2.1 Porovnanie vlastností a využitie

Každý z týchto mikroprocesorov má unikátnu vlastnosť, ktoré ho predurčujú pre rôzne typy projektov:

- **ESP32** je optimálny pre energeticky efektívne aplikácie vyžadujúce komplexnú sieťovú konektivitu.
- **Raspberry Pi** poskytuje výpočtový výkon potrebný pre náročné aplikácie a dátové analýzy.
- **Particle Photon a Electron** sú určené pre projekty, kde je kľúčová integrácia so cloudovými službami a diaľkové riadenie zariadení.
- **Arduino** je vhodné pre edukačné účely a projekty, ktoré nevyžadujú vysoký výpočtový výkon.

1.1.2.2 Mikroprocesory pre špeciálne použitia

- **Texas Instruments MSP430**: tento mikrokontrolér je známy svojou extrémne nízkou spotrebou energie, čo ho robí ideálnym pre batériou napájané aplikácie, ako sú senzorické uzly v IoT sietach.

- **STMicroelectronics STM32:** ide o rodinu mikrokontrolérov na báze ARM Cortex-M ponúka vysoký výkon a bohaté periférne možnosti, vrátane USB, CAN a Ethernet, čo ju robí vhodnou pre priemyselné aplikácie vyžadujúce robustné spojenia.

1.1.2.3 Aplikačné scenáre a budúci vývoj

Z uvedených mikroprocesorov môžeme vytvoriť rôzne typy zariadení a systémov zahŕňajúce:

- **Smart home zariadenia:** využitie ESP32 alebo Particle Photon pre riadenie osvetlenia, bezpečnostných systémov a teploty v domácnostiach.
- **Wearable technológie:** Arduino a MSP430 sú ideálne pre vývoj nositeľných zariadení zameraných na monitorovanie zdravotného stavu alebo fitness aktivity.
- **Priemyselné monitorovanie a kontrola:** STM32 a Raspberry Pi poskytujú potrebný výpočtový výkon pre spracovanie dát z mnohých senzorov v reálnom čase a môžu riadiť zložité priemyselné stroje a systémy.

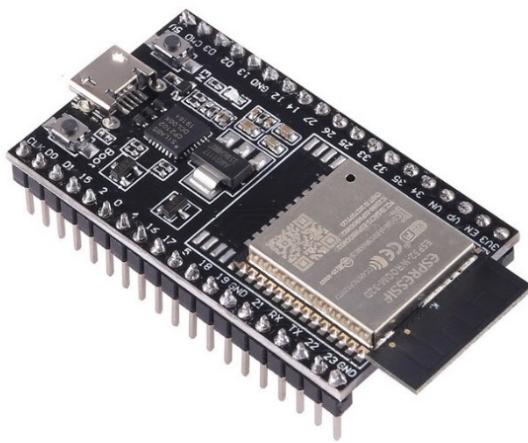
Vzhľadom na rýchly rozvoj technológií a neustále sa zvyšujúce požiadavky na inteligentné zariadenia sa predpokladá, že budúci vývoj mikroprocesorov bude zahŕňať ešte väčšie zvýšenie výkonu, zníženie spotreby energie a lepšiu integráciu s AI a strojovým učením. To umožní ešte sofistikovanejšie a autonómnejšie systémy, čím sa otvoria nové možnosti ich aplikácií v rôznych oblastiach ľudskej činnosti.

Pochopenie týchto technológií a ich potenciálnych aplikácií je kľúčové pre každého, kto sa zaoberá dizajnom a implementáciou moderných elektronických systémov a zariadení.

1.1.2.4 ESP32: podrobný popis a využitie

ESP32 je výkonný mikrokontrolér vyvinutý spoločnosťou Espressif Systems, ktorý sa vyznačuje svojou všeobecnosťou a integrovanými funkciami pre konektivitu. Tento čip je navrhnutý s dôrazom na IoT (Internet vecí) aplikácie, poskytujúc bohatú súpravu funkcií vrátane Wi-Fi, Bluetooth a širokú škálu periférnych rozhraní. Obsahuje dvojjadrový procesor Tensilica Xtensa LX6, ktorý pracuje na frekvencii až 240 MHz. Tento čip ponúka tiež až 520 KB interného SRAM a integrovanú podporu pre Wi-Fi a Bluetooth (Classic a BLE). ESP32 podporuje množstvo I/O rozhraní vrátane UART, SPI, I2C, a PWM, čo umožňuje vývojárom flexibilne pripojiť rôzne senzory a aktuátory. S integrovaným Wi-Fi a Bluetooth, ESP32 umožňuje ľahké pripojenie k internetu a komunikáciu medzi zariadeniami. Toto je ideálne pre aplikácie smart home a prenos dát medzi zariadeniami v reálnom

čase. ESP32 je navrhnutý pre minimalizáciu spotreby energie, s režimami hlbokého spánku a moduláciu výkonu, čo je klúčové pre batériou poháňané zariadenia v IoT. Mikrokontrolér je podporovaný rozsiahloou vývojovou komunitou a espressif poskytuje SDK s nástrojmi a príkladmi pre ľahký vývoj aplikácií. Takisto je oblúbenou voľbou pre vývoj inteligentných zariadení v domácnostiach, ako sú termostaty, svetelné systémy a bezpečnostné kamery. Vďaka svojej nízkej spotrebe energie a kompaktnej veľkosti, ESP32 je ideálny pre integrovanie do nositeľnej elektroniky, ako sú fitness náramky a inteligentné hodinky. Robustnosť a viacúčelová konektivita čipu umožňujú jeho použitie v priemyselných IoT aplikáciách, kde je potrebná spoľahlivá a bezpečná komunikácia. Aj keď ESP32 ponúka mnoho výhod, jeho použitie môže byť obmedzené v extrémne náročných priemyselných aplikáciách, kde môžu byť požiadavky na procesorový výkon a pamäťové kapacity vyššie, než dokáže ESP32 poskytnúť. Navyše, komplexnosť integrovaného vývojového prostredia a nástrojov môže predstavovať výzvu pre nových vývojárov. Pokračujúce vylepšenia v ESP32 a jeho nástupcov by mohli zahŕňať zlepšenia v oblasti bezpečnosti, zvýšenie výkonu a vďaka svojej nízkej spotrebe energie a kompaktnej veľkosti. Tento čip je ideálny pre integrovanie do nositeľnej elektroniky, ako sú fitness náramky a inteligentné hodinky alebo aj psie obojky [4].



Obr. 2: Procesor ESP32

1.1.3 Senzory v mobilných aplikáciách

Senzory v mobilných zariadeniach otvárajú dvere k širokej škále aplikácií, ktoré reagujú na prostredie užívateľa. Patria sem:

- **GPS senzory:** umožňujú aplikáciám poskytovať navigačné služby a lokalizačné

služby, čo je kritické pre mapové a cestovné aplikácie.

- **Akcelerometre a gyroskopy:** tieto senzory merajú pohyb a orientáciu zariadenia, čo je využívané v hrách, aplikáciách pre fitness a zdravie, a tiež na vylepšenie užívateľského rozhrania pri prechádzaní medzi aplikáciami.
- **Senzory svetla a blízkosti:** regulujú jas obrazovky a vypínajú dotykovú obrazovku počas telefonovania, keď je telefón pri uchu, čím šetria batériu a zlepšujú používateľský komfort. kritické pre mapové a cestovné aplikácie.
- **Biometrické senzory:** Odtlačky prstov a skenery tváre zlepšujú bezpečnosť zariadení, umožňujúc bezpečnejšie overovanie a prístup k citlivým údajom a operáciám.

Tieto technológie, integrované do mobilných aplikácií, transformujú spôsob, akým interagujeme so zariadeniami a ako zariadenia interagujú s okolitým svetom. Umožňujú mobilným aplikáciám byť nápomocnými, adaptačnými a oveľa inteligentnejšími. Ako mikroprocesory a senzory pokračujú v rozvoji, môžeme očakávať ďalšie inovácie v mobilných aplikáciách, ktoré budú ešte viac intuitívne a spojené s reálnym svetom.

1.1.3.1 GPS Senzory: podrobný popis a využitie

GPS (Global Positioning System) senzory sú zariadenia, ktoré umožňujú presné určovanie polohy pomocou signálov prijatých od satelitov obiehajúcich okolo Zeme. Sú neoddeliteľnou súčasťou mnohých moderných technológií, od navigačných systémov v autách až po sledovacie zariadenia v smartfónoch a športových hodinkách. GPS senzor pracuje na princípe trianguácie, kde zariadenie vypočíta svoju polohu analýzou signálov z viacerých GPS satelitov. Každý satelit vysiela informácie o svojej pozícii a čase. GPS prijímač používa tieto informácie na určenie vzdialenosť od každého satelitu a na vypočítanie presnej polohy užívateľa.

Medzi jeho kľúčové vlastnosti radíme:

- **Presnosť:** moderné GPS senzory môžu poskytnúť veľmi presné určenie polohy, často do niekolkých metrov. Vysoká presnosť je zabezpečená kombináciou signálov z viacerých satelitov a pokročilých algoritmov na spracovanie dát.
- **Rýchlosť Pripojenia:** GPS senzory sa líšia v rýchlosti, s akou sú schopné nájsť a udržať spojenie so satelitmi. Tento parameter je kritický v dynamických aplikáciách, ako je napríklad navigácia v reálnom čase.

- **Spotreba Energie:** napriek svojej pokročilej funkcionalite, mnohé GPS senzory sú navrhnuté tak, aby minimalizovali spotrebu energie, čo je dôležité pre batériou poháňané zariadenia.
- **Integrácia s inými Technológiami:** GPS senzory sa často kombinujú s inými senzormi a technológiemi, ako sú akcelerometre, gyroskopy alebo dokonca Wi-Fi a mobilné dátové siete na zlepšenie presnosti a spoľahlivosti určovania polohy.

1.1.3.2 GPS modul NEO-M8

Modul NEO-M8 od spoločnosti u-blox je vysoko výkonný GPS modul, ktorý sa vyznačuje vynikajúcou presnosťou, rýchlosťou a spoľahlivosťou pri určovaní polohy. Je navrhnutý tak, aby poskytoval presné a spoľahlivé navigačné dátá v rôznych aplikáciách, od osobných navigačných zariadení až po pokročilé autonómne systémy. Tento modul podporuje viaceré GNSS (Global Navigation Satellite Systems) vrátane GPS, GLONASS, Galileo a BeiDou, čo znamená, že môže prijímať signály z viacerých satelitných systémov naraz. Táto schopnosť zvyšuje presnosť a znižuje čas potrebný na prvotné získanie polohy (Time To First Fix - TTFF). Modul môže tiež pracovať v režime A-GNSS (Assisted GNSS) pre ešte rýchlejšie a presnejšie určenie polohy. Jeho veľkou prednosťou je vysoká presnosť.



Obr. 3: senzor získavania polohy NEOM8

NEO-M8 ponúka vynikajúcu navigačnú presnosť aj v náročných prostrediac, ako sú mestské kaňony alebo husté lesy, vďaka pokročilému spracovaniu signálov a schopnosti využívať viacero GNSS systémov súčasne. Modul je taktiež navrhnutý s ohľadom na energetickú efektivitu, čo je ideálne pre batériou poháňané zariadenia v mobilných aplikáciach alebo v IoT (Internet of Things) zariadeniach. Veľmi zásadnou klúčovou vlastnosťou tohto modulu je, že obsahuje pokročilé technológie na potlačenie rušenia a zabezpečenie vysokého výkonu v prostrediac s vysokým stupňom rušenia, ako sú mestské oblasti a komplexné priemyselné prostredia a preto nájde uplatnenie v širokom spektre aplikácií:

- **Automobilový priemysel:** od navigačných systémov až po pokročilé bezpečnostné systémy.
- **Drony a autonómne vozidlá:** pre presné určovanie polohy a navigáciu.
- **IoT aplikácie:** Napríklad v sledovacích zariadeniach pre logistiku a asset management.
- **Osobné navigačné zariadenia:** ako súčasť outdoorových a fitness zariadení.

1.2 Dostupné technológie pre vývoj webových aplikácií

V dnešnom svete, keď počítačové technológie napredujú krokmi, aké si ľudstvo pred pár rokmi ani nepredstavovalo, je veľmi dôležité vybrať pre každý projekt správne technológie, ktoré umožnia bezproblémovú implementáciu a zároveň budú mať doстатok kompatibilitu medzi sebou. V tejto časti sa zameriame na popísanie jednotlivých technológií, ktoré sú dostupné, porovnáme si ich jednotlivé výhody a nevýhody a popíšeme dôvody, prečo sme sa rozhodli využívať nami zvolené technológie. V našej práci budeme rozlišovať dva veľmi dôležité pojmy:

1.2.1 Serverová časť - backend

Serverová časť, ktorá sa často nazýva aj backend, je najdôležitejšia časť akejkoľvek zložitejšej mobilnej aplikácie, ktorá sa stará o spracovanie a správu dát, ich ukladanie a rovnako aj zabezpečenie. Táto časť aplikácie je nasadená na serveri (apache, aws, google) a pre koncového užívateľa je neviditeľná. Medzi najčastejšie používané programovacie jazyky, v ktorých sa píše backend patria:

- **Java** je veľmi rozšírený programovací jazyk a zároveň je to softvérová platforma, ktorá umožňuje viacplatformový vývoj. Pôvodne bol vyvinutý spoločnosťou Sun Microsystems, no dnes je však pod správou spoločnosti Oracle. Hlavným princípom Javy je podľa anglickej skratky WORA (write once, run anywhere), čo v preklade znamená "napíš raz, spusti kdekoľvek, že kód môžeme spúštať na akomkoľvek zariadení, ktoré podporuje samotnú Javu bez potreby opäťovnej komplikácie [5]. Balíček tejto platformy v sebe takisto zahŕňa JVM (java virtual machine), ktorá je nevyhnutnou súčasťou pre spusťenie skompilovaného bajtkódu. Preto považujeme túto nezávislosť ako jednu z najhlavnejších výhod Javy, pretože je zodpovedná za možnosť developerov

implementovať softvér na jednom systéme a následne ho nasadzovať bez akýchkoľvek zásahov medzi rôzne platformy. Java je medzi developermi známa najmä širokým spektrom štandardných knižníc, vďaka ktorým je možné pokryť väčšinu programátorských potrieb ako napríklad bezpečná komunikácia pomocou sieti, dispomuje rôznymi grafickými uživatelskými rozhraniami, kde môžu developeri spravovať nasadzovanie aplikácie, či prístup k databázam. Štandardom v dnešnom programátorskom svete sú objektovo orientované princípy, ktoré zjednodušujú programovanie a z hľadiska architektúry napomáhajú ku udržiavaniu komplexnosti a zároveň jednoduchosti väčších, ale aj menších aplikácií. Pri použití Javy máme v našom projekte možnosť implementácie viacerých vlákien, čím sa môže zvýšiť efektívnosť aplikácie. Rovnako podporuje rozsiahle možnosti pri investigácii problémov v kóde a zároveň prehľadný spôsob spracovávania chýb, čo je neodmysliteľnou súčasťou každej rozsiahlej mobilnej, či webovej aplikácie [6]. Java zohráva dôležitú úlohu aj pre mobilnú platformu Android, ktorý je jedným z najpoužívanejších mobilných operačných systémov a teda je vhodná na vývoj rôznych mobilných aplikácií pre Android. Tu by sme sa zastavili ako pri prvej nevýhode. Naša práca je zameraná na vývoj multiplatformovej aplikácie, preto sme sa rozhodli využiť Javu len ako backendovú technológiu. Jedným z ďalších dôvodov, prečo sme sa rozhodli vybrať Javu ako backend programovací jazyk bol aj fakt, že vďaka veľkému množstvu vzdelávacích zdrojov, či rôznych fór má pre developerov pomerne rozsiahlu podporu v prípade riešenia problémov a stáva sa tým pádom vcelku jednoduchá. Jej veľká obľúbenosť či už na akademickej pôde, alebo priamo v IT priemysle zabezpečuje aj široký výber vývojárskych prostredí, medzi ktoré patria napríklad Eclipse, IntelliJ IDEA alebo Netbeans, ktoré sa rovnako podieľajú na uľahčení programovania, ladenia a testovania samotných Java aplikácií [7]. Napriek faktu, že Java ponúka veľké množstvo výhod, v dnešnej dobe čelí rovnako aj kritike, ktorú môžeme spojiť s jej nevýhodami. Ako hlavnú určite spomenieme pomalší výkon, ak ju porovnávame napríklad s jazykom C alebo C++. Ďalšou nevýhodou, ktorú treba spomenúť je jej stručnosť, ktorá v niektorých prípadoch môže viesť k dlhším kódom v prípade, že sa jedná o jednoduchšie úlohy. Napriek týmto drobnostiam však vývoj samotnej Javy smeruje k lepšiemu a zavedenie funkcií ako lamda výrazy alebo streamy navyšujú jej efektívnosť.

- **Node.js** je výkonné open-source a zároveň cross-platform prostredie, ktoré umožňuje developerom spúštať JavaScript kód mimo webového prehliadača.

NodeJS bol vyvinutý najmä kvôli zlepšeniu výkonnosti, udržiavateľnosti a škálovateľnosti webových aplikácií. Toto prostredie využíva jadro JavaScript V8, ktoré používa napríklad aj Google Chrome a zabezpečuje využívanie viacerých úloh efektívne na základe neblokovanej architektúry, ktorá využíva riadenie pomocou udalostí tzv. eventov. Jednou z kľúčových funkcií Node.js je, že dokáže používať asynchronné input/output operácie [8]. To znamená, že ak serverová časť (v tomto prípade aplikácia napísaná v Node.js) vykonáva úlohy ako napríklad čítanie z databázy, alebo získavanie dát z inej aplikácie (pomocou REST API), nie je tým nijak blokované hlavné vlákno. Namiesto toho pokračuje vo všetkých ostatných operáciach, ku ktorým ma dostupné dátá a v momente, keď je všetko pripravené, vráti sa k úlohám, ktoré nemohol vykonať kvôli externým dátam. Tento prístup nám umožňuje obsluhovať viacero úloh naraz efektívnejšie a jeho využitie je vhodné pri aplikáciach, ktoré vykreslujú veľa dát v reálnom čase s nízkou latenciou. Node.js rozširuje možnosti JavaScriptu tým, že umožňuje serverové programovanie a tvorbu serverových aplikácií. Tým sa otvárajú dvere front-end vývojárom na vytváranie úplných aplikácií výlučne pomocou JavaScriptu, jazyka, ktorý už dobre poznajú. Navyše, Node.js podporuje najnovšie funkcie JavaScriptu a umožňuje vývojárom ľahko si vybrať, ktorú verziu ECMAScriptu chcú používať, len prostredníctvom aktualizácie svojej verzie Node.js. Celý ekosystém okolo tohto jazyka je bohatý na množstvo knižníc a balíkov, ktoré sú spravované jednoducho pomocou NPM (správca node balíkov). Node.js podporujú rôzne operačné systémy, ako napríklad Windows, macOS a Linux, čo z neho robí vysoko dostupný vývojarsky doplnok na rôznych platformách. Rovnako však ako pri Java, aj tu máme pri veľkom množstve výhod aj určité nevýhody, ktoré je treba zvážiť pri niektorých projektoch. Napriek tomu, že Node dokáže pracovať asynchronne dokáže nezávisle od seba vykonávať viaceré operácie) funguje stále len na jednom vlákne. V prípade, že by sme na serverovej časti potrebovali vykonávať náročné matematické operácie, ktoré by sme vedeli rozdeliť medzi viaceré vlákna pri použití iného jazyka, môže byť pomerne náročný pre CPU. Asynchronnosť, ktorá je jeho obrovskou výhodou býva pri vývoji aj značnou nevýhodou nakoľko sa spoľieha na takzvané callbacky alebo aj spätne volania, čo v niektorých prípadoch vedie k zlej čitateľnosti a udržiavateľnosti kódu. Rovnako má nižšiu výkonnosť pri používaní relačných databáz. Zatiaľ, čo veľmi dobre funguje pri používaní noSQL databáz ako napríklad mongoDB, v našom prípade potrebujeme použiť

SQL databázu, pri ktorej sú výkon a operácie s databázou menej optimalizované. Poslednou nevýhodou Node.js, ktorú v tejto časti spomenieme je jeho nestabilita. Kedže ide o pomerne nový jazyk, ktorý sa neustále vyvíja a v jeho implementácii nastávajú zásadné zmeny, je potrebné udržiavať samotný node a všetky jeho balíčky na čo najnovších stabilných verziách [9].

- **Python** je jedným z najuznávanejších programovacích jazykov známy najmä jeho jasnou syntaxou, ľahkou čitateľnosťou a všetstrannosťou. Vďaka týmto vlastnostiam je veľmi oblúbený medzi developermi a stáva sa vo veľa prípadoch jasnou voľbou pre vývoj backendu webovej aplikácie. Python ponúka viaceré programovacie paradigmy ako napríklad objektovo orientované, či funkcionálne programovanie. Ako sme už spomenuli, jeho syntax je veľmi čitateľná, vďaka čomu je možné písat zdrojové súbory na menej riadkov kódu. Podobne ako spomenutý node.js, aj python disponuje naozaj bohatým spektrom knižníc a balíčkov, ktoré developerom uľahčujú prácu a vo veľkej miere zjednodušujú menšie operácie. Jeho frameworky ako napríklad Flask alebo Django disponujú efektívou architektúrou, ktorá tiež napomáha k vytváraniu backendu client-server aplikácií. Výhodou pythonu je aj jeho komplexnosť, kedže okrem vývoja backendu rôznych aplikácií sa da využiť aj na vývoj umelej inteligencie, či strojové učenie. Samozrejme, asi ako každý jazyk, ani python nie je dokonalý a preto si spomenieme aj nejaké jeho nevýhody. Nakoľko python sa radí medzi interpretované jazyky, jeho výkon v porovnaní s jazykmi, ktoré sú kompliované ako napríklad Java alebo C++ je jeho výkon nižší, čo môže mať vplyv najmä na aplikácie, ktoré potrebujú procesovanie dát v reálnom čase. Nakoľko python obsahuje zámok nazývaný global interpreter lock, ktorý zakazuje pythonu spúšťanie viacerých vlákien naraz, má nižší výkon pri viac vláknových aplikáciach a tým sa stáva menej vhodným pre aplikácie, pre ktoré je súbežnosť viacerých operácií súčasne klúčová. Jednoduchosť a flexibilita, ktorú sme spomenuli medzi výhodami, sa spája aj s nevýhodou, že python je náročnejší na výkon hardvéru, pretože jeho jednoduchosť z hľadiska kódu vplýva na vyššie nároky alokovanej pamäte v porovnaní napríklad s jazykom C. Napriek týmto všetkým spomenutým faktom je python aj nadalej veľmi oblúbený backandový jazyk v rôznych menších firmách či startupoch, ktoré ocenia rýchly vývoj a jednoduché možnosti nasadzovania [10].

1.2.2 Užívateľská časť - frontend

Pojem frontend môžeme vysvetliť ako klientská časť mobilnej/webovej aplikácie, ktorá je zodpovedná za koncový výstup, ktorý uvidia užívatelia priamo vo svojich mobilných zariadeniach, alebo pri webových aplikáciach vo svojom webovom rozhraní. V tejto časti aplikácie môžu užívatelia iteragovať s aplikáciou v podobe rôznych tlačidiel, vstupných textových polí alebo obrázkov. Vývoj frontendu je zamieraný na celkové užívateľské rozhranie, čo znamená, že developeri sa musia sústreďiť okrem funkcií, s ktorými užívateľ iteraguje a pomocou ktorých komunikuje so serverovou časťou aj na dizajn či intuitívnosť aplikácie. Ďalším veľmi dôležitým aspektom frontendu je takzvaná responzivita alebo prispôsobivosť všetkých dizajnov, čo v preklade znamená, že frontend musí byť schopný prispôsobiť sa rôznym veľkosťiam obrazoviek našich mobilných zariadení. Rovnako dôležitou časťou je správna optimalizácia a rýchlosť frontendu, aby mal konečný užívateľ z aplikácie čo najlepší zážitok v podobe rýchlo načítaných stránok. Frontend sa skladá z jeho troch základných programovacích jazykov, ktorými sú:

- **HTML** alebo aj HyperText Markup language je základnou zložkou každého frontendu či už webovej alebo mobilnej aplikácie. Jeho využitie je dôležité pri štruktúrovaní obsahu na internete. Každá webová aplikácia a zároveň niektoré mobilné aplikácie, ktoré navštívite sú tvorené pomocou HTML. Tento jazyk sa stará o to, aby obsah webovej či mobilnej aplikácie ako obrázky či text boli správne naformátované. Kód HTML využíva takzvané tagy ako napríklad `<head>`, `<div>`, `<body>`, `` v prípade webových aplikácií a napríklad `<view>`, `<text>`, `<area>` v prípade mobilných aplikácií. Jeho neoddeliteľnou súčasťou je aj CSS, ktoré samostatne rozoberieme v ďalšom odstavci. Medzi základné výhody využívania HTML patrí určite jednoduchosť a radí sa medzi najjednoduchšie jazyky vôbec. Je rovnako aj univerzálny, nakoľko ho dokáže čítať každý webový prehliadač, či mobilné zariadenie. Aj pri HTML prichádzajú niektoré nevýhody, ktoré by sme radi spomenuli. HTML disponuje pomerne limitovanou funkcionalistou. Nakoľko ide výlučne o statický jazyk, nie je možné žiadne dynamické generovanie výsunu bez použitia ďalšieho doplnkového jazyka, akým je napríklad JavaScript, ktorý rovnako rozvinieme podrobnejšie. Rovnako si treba pri vývoji HTML dať pozor aj na jeho ďalšiu nevýhodu a tou je nekonzistentnosť. HTML obsah sa môže v rôznych webových prehliadačoch zobrazovať rozlične, preto je dôležitý podrobné testovanie. Jazyk

HTML rovnako, ako iné moderné programovacie jazyky sa neustále vyvíja. Jeho najnovším štandardom je HTML5. Tento štandard obsahuje rozličné doplnky umožňujúce prácu s multimediálnym, či grafickým obsahom priamo v prehliadači zariadenia, v ktorom aplikáciu zorbazujeme bez ďalších doplnkov. HTML5 disponuje aj viacerými API službami, ktoré ulahčujú a zároveň umožňujú developerom ukladať údaje bez pripojenia internetu alebo spúštať rôzne úlohy efektívne na pozadí aplikácie. Znalosť a použitie HTML je preto klúčové pri tvorbe webových alebo mobilných aplikácií [11].

- **CSS** alebo aj cascading style je hlavná technológia, alebo môžme povedať aj programovací jazyk, ktorý sa používa na štýlovanie a vytváranie dizajnu webových a mobilných aplikácií. Developeri pomocou CSS ovládajú rozloženie jednotlivých objektov, farby, font písma a všetky podstatné vizuálne zmeny, ktoré sú dôležité pre pútavosť a dobrú orientovanosť v aplikáciach. Vďaka CSS je možné meniť dizajn aplikácií bez potreby zasahovať priamo do HTML kódu a teda oddeliť prezentačnú časť aplikácie od jej obsahu. CSS na základe identifikátorov vyberá jednotlivé časti z HTML a aplikuje na ne zvolené štýly. Medzi hlavné výhody CSS patrí najmä jeho konzistentnosť, vďaka ktorej môžu developeri zabezpečiť jednotný štýl vo viacerých častiach aplikácií. Tým, že sa prepojí jeden CSS súbor s viacerými HTML súbormi sa vytvorí konzistentný štýl naprieč viacerými komponentami. Rovnako tým, že používame tento CSS štýl pre viaceré časti aplikácie, zobrazovacie zariadenie si dokáže tieto štýly uložiť do svojej medzipamäte, namiesto toho, aby opäťovne tieto dátá načítaval a tým urýchli chod aplikácie. Najnovšie verzie CSS ponúkajú funkcie ako napríklad mriežku alebo flexbox, ktoré zjednodušujú pokročilé nastavenia uloženia jednotlivých častí a zároveň pracujú absolutne responzívne, čo znamená, že sa prispособia veľkosti obrazovky zariadenia. CSS ako základný jazyk je pomerne jednoduchý, no na zvládnutie týchto pokročilejších funkcií je potrebné viac učenia a praxe. Rovnako ako pri HTML aj pri CSS je potreba dať si pozor na rôzne zariadenia, ktoré môžu jednotlivé štýly vykresliť rozlične. Táto nevýhoda si vyžaduje mimoriadne úsilie, aby bola zabezpečená konzistentnosť medzi jednotlivými najčastejšie využívanými zariadeniami. V prípade väčších a rozsiahlych aplikácií je nevyhnutný dôraz na dobré navrhnutie štýlov, pretože ich údržba sa môže stať zložitá a zároveň nepraktická. Príchodom najnovšej verzie CSS3 bolo developerom umožnené množstvo možností akými sú napríklad rôzne animácie a prechody [12].

- **JavaScript** je programovací jazyk, ktorý je používaný pri tvorbe webových a mobilných aplikácií na vytváranie dynamických a interaktívnych častí. Vďaka javascriptu vie klientská časť komunikovať s tou serverovou pomocou viacerých technológií ako sú sockety alebo rest, ktoré sú obe zároveň použité aj v našej práci. Pôvodne bol tento jazyk vyvinutý kvôli programovaniu frontendových častí aplikácie, no v dnešnej dobe je pomerne rozvinutý a je možné vďaka nemu implementovať aj serverovú časť aplikácie (backend), najmä prostredníctvom už spomenutého node.js. Medzi jeho základné vlastnosti a zároveň výhody, ktoré by sme chceli spomenúť patrí určite jeho schopnosť vytvárať rôzne zložité komponenty, akými sú napríklad interaktívne mapy, animovaná grafika a akékoľvek dynamické formuláre. Vďaka javascriptu vieme reagovať v prostredí frontendu na jednotlivé vstupy užívateľov (rôzne kliknutia či textové vstupy) bez potreby opäťovného načítavania stránky. Ďalšou jeho výhodou je samozrejme prototypová dedičnosť, čo znamená, že objekty môžu dediť vlastnosti iných objektov podobne ako v Java. Rovnako ako v Node.js, je možné používať asynchronné programovanie, vďaka čomu sa môžu viaceré zložité operácie vykonávať priamo na strane klienta súčasťne. Javascript je podporovaný takmer každým moderným zariadením, takže jeho použiteľnosť je naozaj takmer bezbariérová. Rovnako ako predošlé jazyky, aj tento disponuje rôznymi frameworkami, ktoré uľahčujú niektoré štátia vývoja (napríklad ReactNative, React, Angular, Vue.js). Vďaka týmto frameworkom môžeme vytvárať rôzne užívateľské rozhrania, automatizovať potrebné dynamické komponenty a zároveň pomerne jednoducho testovať pracovné postupy. Ani Javascript však nie je dokonalý a preto by sme radi spomenuli zopár nevýhod, ktoré tento jazyk má. Tou najhlavnejšou je bezpečnostné riziko, keďže celý javascript sa nachádza na strane klienta, je dôležité implementovať na frontende len funkcionality, ktoré nijak neohrozujú aplikáciu zvonka. Ďalej napriek tomu, že javascript sa radí vo všeobecnosti, medzi výkonné a rýchle jazyky, v prípade komplikovaných, zložitých alebo zle optimalizovaných kódov môžu spomalovať chod webovej stránky a to najmä na mobilných zariadeniach pri spracovávaní veľkého množstva dát. Napriek tomuto sa však javascript neustále vyvíja a príjma nové funkcie, ktorého udržujú v popredí vývoja webových a mobilných aplikácií. Schopnosť javascriptu komunikovať a spolupracovať s viacerými technológiami ho robí nadalej najpoužívanejším jazykom pre vývoj v tejto oblasti[13].

1.2.3 Dostupné frameworky

V tejto kapitole si rozoberieme dostupné frameworky, ktoré bolo možné využiť v našej práci.

- **SpringBoot** je rozšírená verzia základného frameworku Spring, ktorý je písaný v Java. Developerom uľahčuje základnú konfiguráciu potrebných nastavení ako pripojenie na databázu, bezpečnostné filtre a podobne. Rovnako dokáže automatizované spravovať jednotlivé závislosti, ktoré pôvodne vyžadovali manuálne nastavenie. SpringBoot je vo všeobecnosti navrhnutý tak, aby bol developerovi umožnený čo najrýchlejší vývoj samotnej serverovej časti bez potreby zdĺhavých nastavovaní na začiatku projektu. Taktiež je možné využiť predvolené nastavenia pri inicializácii springboot projektu, čo znamená automatické predkonfikurovanie určitých závislostí, čím sa opäť urýchluje uvedenie aplikácie do prevádzky. Ďalej prišiel SpringBoot s možnosťou zabudovaných HTTP serverov ako napríklad Tomcat alebo Jetty, ktoré môžeme jednoducho využívať na testovanie jednotlivých aplikácií a nie je potrebné nasadzovanie na externý server [14]. Springboot ma výbornú predprípravu na využívanie mikroslužieb a poskytuje základné funkcie akými sú napríklad už spomenuté vstavané servery a jednoduchý prístup k rozhraniam príkazového riadka . Napriek tomu je dôležité zhodnotiť využitie SpringBootu podľa požiadavok aplikácie. Je potrebné počítať s vyšším zdrojom pamäte a s dostatočnou senioritou v Java, ktorá je nevyhnutná pre správny návrh a konfiguráciu aplikácie [15].
- **Hibernate** je technológia, ktorá priamo súvisí s vývojom v SpringBoote a preto si ju spomenieme ako nasledovnú. Ide o prominentný rámec objektovo relačného programovania (ORM). Píše sa rovnako v jazyku Java a priamo zjednodušuje všetky interakcie, ktoré serverová časť aplikácie využíva pre komunikáciu s databázou. Je využívaný najmä kvôli jeho schopnosti mapovať jednotlivé entitné triedy v Java na konkrétné tabuľky v databáze. Ďalej sa stará o transakcie s databázou, čím optimalizuje zložité operácie nad databázou vďaka veľmi intuitívnomu objektovo orientovanému prístupu [16].
- **ReactNative** je veľmi moderný framework, ktorý vyvinula spoločnosť Facebook za účelom developmentu v oblasti mobilných aplikácií. ReactNative je jedinečný v tom, že vďaka nemu developer dokáže využívať mobilnú aplikáciu multiplatformo, čo znamená, že ho využíva súčasne pre iOS aj Android. Toto všetko využitím jedného zdrojového kódu. ReactNative je založený na archi-

tektúre Reactu, ktorá fuguje využívaním komponentov, vďaka ktorým môže vytvárať rôzne používateľské rozhrania. Oblúbeným sa stáva aj vďaka podpore obrovskej komunity a celkového rozsiahleho ekosystému, vďaka ktorému má developer k dispozícii prístup k množstvu balíčkov a knižníc, rôznych nástrojov a rámcov. Rovnako v prípade akýchkoľvek problémov a bugov v týchto balíčkov táto komunita pomáha riešiť všetko v čo najrýchlejšom možnom čase [17]. Výkon reactNative samozrejme nie je na takej úrovni, ako aplikácie, ktoré sú plne natívne, napriek tomu je kvalita výkonu na veľmi podobnej úrovni vďaka schopnosti priamo pracovať s jednotlivými natívnymi komponentami. Nepotrebuje pre tento účel používať webové rozhranie. Rovnako vývoj je veľmi pohodlný vďaka funkcie hot reloading, ktorá aplikuje všetky aktuálne zmeny v kóde do spustenej služby, bez potreby opäťovnej komplikácie kódu (technicky presne). Ak by sme však v našej aplikácii potrebovali využívať zložité animácie, prípadne komplikované gestá, reactNative môže reactNative tahat za kratší koniec v porovnaní s natívnymi aplikáciami [18]. ReactNative je tou najsprávnejšou volbou pre programy, ktorých je základ rýchly spôsob vývoja. Natívne aplikácie sa dodnes považujú za efektívnejšie, avšak je dôležité zhodnotiť, či potrebné kompromisy stoja za zváženie.

1.2.4 Expo: Platforma pre vývoj React Native aplikácií

Expo je open-source platforma, ktorá umožňuje vývojárom rýchlo vytvárať a testovať aplikácie React Native bez potreby konfigurácie natívnych vývojových prostredí. Je navrhnutá tak, aby zjednodušila proces vývoja mobilných aplikácií tým, že poskytuje súbor nástrojov a služieb, ktoré automatizujú mnohé náročné aspekty vývoja. Rovnako umožňuje vývojárom začať vytvárať aplikácie bez komplikovaného nastavovania natívnych SDK. Vývojári môžu jednoducho nainštalovať Expo CLI a začať vytvárať nové projekty [19]. Má funkcie ako hot reloading a umožňuje vývojárom zdieľať náhľady aplikácií s klientmi alebo tímom prostredníctvom QR kódov, čo značne urýchluje iteratívny vývoj. Platforma Expo ponúka bohatú knižnicu predpripravených komponentov a prístup k rozhraniám API pre kameru, polohu, senzory a ďalšie hardvérové funkcie zariadenia. Veľmi veľkou výhodou je mobilná aplikácia Expo Go, ktorá umožňuje vývojárom testovať svoje aplikácie priamo na mobilných zariadeniach iOS a Android bez potreby vydávania cez obchody s aplikáciami.

Kedže sa neustále inovuje a pridáva nové funkcie, aby zlepšilo svoju platformu. S nástupom technológií ako EAS (Expo Application Services), ktoré umožňujú väčšiu

flexibilitu pri správe a vydávaní aplikácií, sa Expo stáva ešte viac flexibilným a mocným nástrojom pre vývojárov aplikácií. Umožňujú taktiež efektívnejšiu prácu na projektoch a zjednodušujú procesy ako aktualizácie aplikácií, správu konfigurácií a nasadenie nových verzií. Pravidelné aktualizácie zvyšujú jeho schopnosti a zlepšujú celkový výkon. Využitím platformy Expo prichádzajú aj určité obmedzenia, medzi ktoré by sme zaradili:

- **Vlastné natívne moduly:** aj keď Expo ponúka širokú škálu API, nemusí z krabice podporovať všetky natívne moduly. Vývojári, ktorí potrebujú špecifické natívne funkcie, ktoré Expo priamo nepodporuje, by mohli musieť prejsť z Expo na čistý React Native projekt, čo zahŕňa správu natívneho kódu [20].
- **Výkon a veľkosť aplikácie:** aplikácie vytvorené s Expo môžu byť väčšie a mierne pomalšie v porovnaní s tými, ktoré sú postavené na čistom React Native vďaka dodatočným knižniciam a vrstvám abstrakcie.

1.3 Haversinova funckia : význam a využitie v geografických výpočtoch

Haversinova funkcia predstavuje klúčový nástroj v sférickej trigonometrii, najmä pri výpočte vzdialenosí na sfére, ako je Zem. Je to matematický vzorec, ktorý efektívne rieši problémy spojené s výpočtom ortodromických (najkratších) vzdialenosí medzi dvoma bodmi na zemskom povrchu na základe ich zemepisných súradníc [21].

Vzdialenosť d medzi dvoma bodmi na povrchu sféry, s ich zemepisnými šírkami ϕ_1, ϕ_2 a dĺžkami λ_1, λ_2 , možno vypočítať pomocou Haversinovho vzorca:

$$d = 2r \arcsin \left(\sqrt{\text{hav}(\phi_2 - \phi_1) + \cos(\phi_1) \cos(\phi_2) \text{hav}(\lambda_2 - \lambda_1)} \right)$$

kde funkcia $\text{hav}(\theta)$ je definovaná ako:

$$\text{hav}(\theta) = \sin^2 \left(\frac{\theta}{2} \right)$$

Haversinova funkcia sa široko využíva v navigačných systémoch, ako sú GPS, letecká a námorná navigácia, kde presnosť a efektivita sú nevyhnutné. Táto funkcia je tiež neoceniteľná v geografických informačných systémoch (GIS) a aplikáciách vyžadujúcich presné výpočty geografických dát [22]. Vďaka svojej schopnosti presne

a efektívne vypočítať vzdialenosť medzi bodmi na sférickom povrchu je Haversinova funkcia nenaRADiteľným nástrojom v mnohých technických a vedeckých disciplínach. Jej aplikácie v moderných technológiách, od navigácie po mapovanie a analýzy, svedčia o jej dôležitosti a všestrannosti v digitálnej dobe.

1.4 Úvod do problematiky záchranárskej kynológie s použitím gps

V súčasnom dynamickom prostredí záchranárskych operácií, kde každá sekunda môže znamenať rozdiel medzi životom a smrťou, sa stáva GPS technológia neoceniteľným nástrojom pre psovodov záchranárov. Títo špecialisti, pracujúci často v náročných a nepredvídateľných terénoch, sa spoliehajú na GPS nielen na zlepšenie efektivity svojich vyhľadávacích misií, ale aj na zvýšenie bezpečnosti a koordinácie svojich tímov.

GPS zariadenia používané psovodmi záchranármami umožňujú presné sledovanie polohy vyhľadávacích psov aj v najnáročnejších podmienkach. Tieto zariadenia sú obvykle pripojené na obojky psov, kde zaznamenávajú ich presnú trasu pohybu. Vďaka tomu môžu záchranári v reálnom čase vidieť, kde sa pes nachádza, kam smeruje, a či nezablúdil príliš ďaleko od svojho vodiča alebo z bezpečnej zóny. S použitím GPS je koordinácia medzi rôznymi členmi záchranárskych tímov podstatne jednoduchšia. Vodiči psov môžu efektívne plánovať vyhľadávacie trasy a zabezpečiť, že sa ich zvieratá nepohybujú v oblastiach, ktoré už boli prehľadané, alebo sú známe svojou nebezpečnosťou. Taktiež, v prípade potreby rýchlej evakuácie alebo posilnenia tímu na určitom mieste, môžu byť presné GPS údaje klíčové pre rýchle a koordinované konanie.

Okrem logistických výhod, GPS technológia zvyšuje aj bezpečnosť samotných záchranárskych psov a ich vodičov. V prípade, že pes alebo vodič uviazne alebo sa ocitne v nebezpečenstve, umožňuje GPS rýchle zameranie ich polohy a okamžitú reakciu záchranných tímov. Využitie GPS technológie v teréne záchranařmi psovodmi tak predstavuje neoddeliteľnú súčasť moderných záchranných operácií. Nie lenže značne zvyšuje šance na úspešné nájdenie osôb v núdzi, ale zároveň poskytuje cenné informácie pre plánovanie a realizáciu záchranných misií. Tieto systémy, neustále sa vyvíjajúce a zdokonaľujúce, sú príkladom toho, ako moderná technológia môže slúžiť ľudstvu v kritických momentoch.

1.4.1 Prepátravanie pomocou rojníc

Prehľadávanie pomocou rojnice je efektívna metóda v záchrannárskej kynológii, kde sa využíva koordinovaný pohyb skupiny psovodov a ich psov pri záchranných operáciách. Tento prístup je inšpirovaný rojovými správami v prírode, ako sú roje hmyzu alebo hejná vtákov. Na začiatku pátracej akcie sa psovodi s ich psami postavia do formácie, otočení smerom na veľký sektor (napríklad 500x3500 metrov). Po štartie patracej akcie vyšlú psovodi všetkých svojich psov prepátravať spoločne priestor pred sebou. Táto metóda je špecifická svojimi základnými charakteristikami, ktorými sú:

- **Tímova koordinácia:** celý tím psovodov s ich psami sa rozmiestni do formácie, ktorá pripomína roj alebo mriežku. Táto formácia umožňuje systematické a súčasné prehľadávanie veľkých oblastí.
- **Efektivita:** rozloženie tímu do rojnice zvyšuje pokrytie oblasti pri prehľadávaní, čo vedie k rýchlejšiemu a účinnejšiemu vyhľadávaniu osôb alebo objektov v teréne.
- **Komunikácia a spolupráca:** pri prehľadávaní v rojnicovej formácii je klúčová vzájomná komunikácia medzi psovodmi. Umožňuje to koordinovať pohyby, rýchlo reagovať na nálezy a efektívne sa prispôsobovať meniacim sa podmienkam v teréne.
- **Flexibilita:** rojnicové prehľadávanie je flexibilné a môže byť prispôsobené podla špecifických potrieb mise. Formáciu je možné upraviť v závislosti od terénu, počasia a ďalších faktorov, ktoré môžu ovplyvniť prehľadávanie.

Hoci rojnice predstavuje efektívnu metódu v záchranných operáciách kynológie, vyžaduje si vyššiu úroveň koordinácie a tréningu. Správna príprava a organizácia sú klúčové pre úspešné využitie tejto techniky v praxi. Záchranné tímy by mali zvážiť všetky aspekty tejto metódy vrátane potreby pravidelného školenia a možných nárokov na logistiku, aby maximálne využili jej potenciál pri záchrane životov.

1.4.2 Sektorové prehľadávanie

Sektorové prehľadávanie je strategická metóda používaná v kynológii pri záchranných a vyhľadávacích operáciách, kde je terén rozdelený na menšie, spravovateľné oblasti alebo sektory. Každý sektor je potom systematicky prehľadaný jedným alebo

viacerými tímami psovodov s ich trénovanými psami. Tento prístup umožňuje detailné a organizované prehľadávanie rozsiahlych oblastí. V dnešnej dobe je to najpoužívanejšia metóda organizácie USAR (Urban search and rescue), ktorá najmä vďaka jej efektívnosti vie omnoho viac využívať potenciál a talent záchranárskych psov, čím urýchli a uľahčí celkový priebeh a náročnosť záchranárskej operácie. Pre porovnanie uvedieme aj pri tomto spôsobe zopár základných charakteristík:

- **Plánovanie a rozdelenie oblasti:** pred zahájením operácie sa vykoná dôkladné plánovanie, kde sa určí, ako bude daná oblasť rozdelená na sektory. Každý sektor je priradený konkrétnemu tímu, ktorý zodpovedá za jeho kompletné prehľadanie.
- **Systematickosť:** záchranárske jednotky systematicky prehľadávajú svoje pridelené sektory, čo zvyšuje pravdepodobnosť nálezu a minimalizuje riziko prehliadnutia dôležitých indícii alebo stôp.
- **Dôkladnosť:** vďaka menším prideleným oblastiam môžu byť sektory prehľadané dôkladnejšie a detailnejšie, čo je klúčové v situáciach, kde každý dôkaz môže byť rozhodujúci.

Sektorové pátranie robí efektívnejšie voči tomu rojnicovému najmä fakt, že každý sektor je prehľadaný raz, čo znižuje duplicitu a zabezpečuje, že úsilie a zdroje sú vynaložené efektívne. Preto je zásadnou technikou v kynológii, ktorá umožňuje záchranárskym tímom efektívne a systematicky pokryť veľké plochy pri vyhľadávacích operáciách. Aj keď si vyžaduje dôkladné plánovanie a koordináciu, prínosy tejto metódy v podobe zvýšenej efektivity a pokrytie sú neoceniteľné, najmä v náročných záchranných situáciach, kde každý detail môže pomôcť lokalizovať stratené osoby alebo objekty.

Jeho hlavnou nevýhodou je cenová nedostupnosť. Väčšina dostupných zariadení, ktoré sú v dnešnej dobe nevyhnutné pre taktiku sektorového prepátravania sú finančne príliš nákladné a nakoľko vrámci SR pôsobia jednotlivé kynologické kluby len na dobrovoľníckej báze, stretnú sa s takýmito zapožičanými zariadeniami len na krátky čas. Táto nedostupnosť spôsobuje, že pri ostrých akciách kynológovia nedispomnujú dostatočnými skúsenostami a vedomotami, aby mohli GPS zariadenia používať maximálne efektívne a správne.

2 Ciele práce

V tejto časti si popíšeme jednotlivé body, ktoré patria medzi ciele diplomovej práce. Týmito bodmi budeme postupovať pri implementácii a vývoji nového GPS systému, ktorý bude nápnomocný pri lokalizácii psovodov a ich psov v teréne počas ostrých pátracích akcií a tréningov.

2.1 Oboznámenie sa s dostupnými riešeniami sledovania polohy osôb a zvierat pomocou GNSS.

Cieľom je oboznámiť sa a analyzovať existujúce dostupné riešenia pre sledovanie polohy osôb a zvierat. Súčasťou tohto bodu bude porovnanie dostupných obojkov na trhu.



Obr. 4: GPS obojok garmin TT15X

Rovnako aj pri tomto obojku môžeme hovoriť o vysokej kvalite, ktorú ľahko prekonáť, no napriek tomu všetky možnosti, ktoré obojok ponúka, nie sú pre kynológov v tréningu a na pátracích akciach nevyhnutné. Jeho cena podobne ako pri predošom zariadení je pre slovenské týmy príliš vysoká. Pokročilé funkcie vyžadujú určité technické znalosti a zručnosti pre ich správne nastavenie a použitie, ktoré psovodi kvôli jeho nedostupnosti nemajú.

2.2 Zvolenie vhodného HW pre umiestnenie na zvieratá a osoby.

Ďalším cieľom je výber vhodného hardwaru, pomocou ktorého bude možné vytvoriť zariadenie na sledovanie osôb a zvierat v teréne. Vhodné zariadenie by malo byť cenovo dostupné a disponovať vlastným dostatočným zdrojom energie pre použitie v teréne. Toto zariadenie musí byť zároveň schopné ukladať súradnice z niekoľko hodinových aktivít na vlastnú pamäť.

2.3 Realizácia vhodného riešenia serverovej časti systému.

Cieľom číslo tri je realizácia serverovej časti systému. Serverová časť musí byť schopná autorizovať užívateľa pomocou mena a hesla, spracovavať údaje o polohe osôb a zvierat. Serverová časť bude implementovaná pomocou zvolených frameworkov, ktoré sú spomenuté v teoretickej časti tejto práce.

2.4 Realizácia klientskej časti systému.

Štvrtým cieľom je realizovať klientskú časť systému. Táto časť musí byť intuitívna a jednoduchá, aby psovodom nasadeným v pátracích akciách uľahčoval ich výkon a zároveň nezaťažoval samotným používaním jednotlivých psovodov. V klientskej časti bude mať užívateľ na výber viaceré možnosti zobrazenia, bude môcť vidieť na mape svoju polohu a sektor ktorý má prepátrávať.

2.5 Overenie funkčnosti

Piatym cieľom tejto práce je overenie funkčnosti navrhnutého systému. Psovodi dostanú nás systém, ktorý počas tréningu otestujú.

- **DOG GPS X30TB** je špecializované sledovacie zariadenie navrhnuté pre použitie na domácich miláčikoch, najmä na psíkoch. Toto zariadenie kombinuje GPS technológiu s modernými bezdrôtovými komunikačnými schopnosťami, ako je Bluetooth, aby poskytlo majiteľom presnú lokalizáciu ich zvierat. Často je vybavené aj odolným puzdrom, ktoré chráni elektroniku pred vodou, prachom a nárazmi. Zariadenia ako DOG GPS X30TB sa začali vyvíjať s rozšírením používania GPS technológie v osobných elektronických zariadeniach. Postupný pokrok v miniaturizácii a energetickú efektivitu umožnil vývoj praktických GPS trackerov špeciálne určených pre domáce zvieratá. Tieto zariadenia boli postupne doplnené o dodatočné funkcie ako monitormovanie aktivít a zdravotný monitoring, čím sa stali multifunkčnými nástrojmi pre majiteľov zvierat.



Obr. 5: Obojok DOG GPS X30TB

Obojok DOG GPS X30TB nie je napriek jeho vysokej kvalite najlepším riešením pre situácie, v ktorých sa psovodi v ostrých pátracích často vyskytujú. Ich multifunkcionalita, ktorá by mala byť ich bonusom je vo veľa prípadoch nie len zbytočná, ale dokonca aj naškodu. Psovodi sa v týchto nie úplne intuitívnych užívateľských rozhraniach s veľkým množstvom funkcií často zle vyznajú a nedokážu využiť tú najžiadanejšiu - gps sledovanie. Rovnako cena jedného takéhoto obojku, ktorý dokonca priamo nie je určený a vyvinutý na sektorové prepátravanie, sa pohybuje v sumách celého ročného rozpočtu niektorých menších záchranárskych kynologických združení.

- **Garmin TT 15X** je sledovací a výcvikový obojok pre psy, ktorý je navrhnutý tak, aby poskytoval majiteľom psov a psovodom pokročilé sledovacie možnosti a vylepšené tréningové funkcie. Tento obojok je súčasťou rozšíreného ekosystému produktov Garmin pre poľovníctvo a kynológiu, kombinuje vysokú odolnosť, dlhú výdrž batérie a komunikáciu na veľké vzdialenosť. Využíva vylepšený GPS a GLONASS prijímač pre rýchlejšie a presnejšie lokalizovanie polohy psa, čo je kritické v náročných vonkajších prostrediach. Obojok obsahuje pokročilé tréningové možnosti ako stimulácia, zvukové signály a vibrácie, ktoré pomáhajú pri výcviku a riadení správania psa na diaľku. Takisto umožňuje majiteľom a psovodom udržiavať neustálu kontrolu nad pohybom a polohou psa, čo je obzvlášť užitočné pri pátraniach v rozľahlých alebo husto zalesnených oblastiach.

3 Praktická časť

V tejto časti diplomovej práce budeme popisovať jednotlivé postupy z vytvárania systému pre psovodov. Rovnako si popíšeme implementáciu kódu, ktorý sme navrhli, definujeme funkcionálne a nefunkcionálne požiadavky na systém, popíšeme navrhnutý state diagram a uml diagram. Aplikácia zároveň musí byť dostatočne bezpečná, aby sa predišlo zneužitiu dát.

3.1 Funkcionálne požiadavky

Funkcionálne požiadavky definujú konkrétné funkcie alebo správanie, ktoré systém musí vykonať alebo poskytnúť. Nižšie uvedené patria medzi najhlavnejšie:

- **Spracovanie polohy:** systém musí byť schopný spracovávať a zaznamenávať polohu psovodov a ich psov v reálnom čase pomocou GNSS technológií.
- **Hardware pre zvieratá a ľudí:** systém musí podporovať výber a integráciu vhodného hardvéru pre umiestnenie na zvieratá a ľudí.
- **Serverová časť:** musí zahŕňať funkcie pre správu dát, autentifikáciu užívateľov, správu používateľských účtov a zaznamenávanie činností.
- **Klientská časť:** aplikácia musí poskytovať užívateľské rozhranie pre zobrazenie aktuálnej polohy, história pohybu, správu účtov a ďalších relevantných informácií.
- **Overenie funkčnosti:** systém musí podliehať testovaniu na overenie spoloahlivosti, presnosti a efektivity v rôznych prostrediacich a situáciach.

3.2 Nefunkcionálne požiadavky

Nefunkcionálne požiadavky sú zamerané na kvalitu a normy, ktoré náš systém musí splňať:

- **Výkon:** systém musí poskytovať rýchle a presné určovanie polohy s minimálnym oneskorením.
- **Bezpečnosť:** dáta o polohe a osobné informácie musia byť chránené pomocou bezpečnostných protokolov a šifrovania.
- **Spoloahlivosť:** systém musí byť spoloahlivý a musí zaručiť kontinuálnu prevádzku počas záchranných operácií.

- **Rozšíriteľnosť:** systém by mal byť navrhnutý s možnosťou ľahkého rozširovania funkcií alebo integrácie s inými technológiami.
- **Užívateľská prívetivosť:** Aplikácia musí byť intuitívna a jednoducho použiteľná pre všetkých užívateľov, vrátane menej technicky zdatných.
- **Kompatibilita:** Systém musí byť kompatibilný s rôznymi zariadeniami a platformami.

3.3 Kód mikročipu ESP32

V tejto časti si ukážeme a popíšeme kód, ktorý sa nachádza na mikročipe ESP32.

Listing 1: Časť zdrojového kódu implementovaného do ESP32

```

1
2 void loop() {
3     if (SerialBT.available()) {
4         String command = SerialBT.readStringUntil('\n');
5         if (command == "start") {
6             collectingData = true;
7             coordinates = "";
8             SerialBT.println("Starting GPS data collection.");
9         } else if (command == "get coords") {
10            SerialBT.println(coordinates);
11            collectingData = false;
12            SerialBT.println("Stopping GPS data collection.");
13        }
14    }
15
16    if (collectingData) {
17        while (Serial1.available() > 0) {
18            char c = Serial1.read();
19            if (gps.encode(c)) {
20                if (gps.location.isValid()) {
21                    double lat = gps.location.lat();
22                    double lng = gps.location.lng();
23                    coordinates += String(lat, 6) + ", " + String(lng, 6) +
24                    ";" ; }}}} }
```

Funkcia loop() v našom kóde je nekonečná slučka, ktorá sa opakuje počas celej doby činnosti mikročipu ESP32. Táto funkcia je hlavná časť programu pre riadenie GPS a Bluetooth funkčnosti. Táto slučka sa skladá z jednotlivých najdôležitejších častí:

- **Príjmanie inštrukcii cez bluetooth**

- **Príjem príkazu:** Funkcia najprv skontroluje, či boli cez Bluetooth prijaté nejaké dátia. To sa deje pomocou metódy SerialBT.available(), ktorá vráti true, ak sú na vstupe nejaké dátia.
- **Čítanie príkazu:** Ak sú k dispozícii nejaké dátia, prečítajú sa až do nájdenia znaku nového riadku (newline, /n) pomocou SerialBT.readStringUntil('/n'). Prečítaný reťazec sa uloží do premennej command.
- **Príkaz "start":** Ak prijatý príkaz zodpovedá slovu "start", aktivuje sa zber GPS dát nastavením collectingData na true. Zároveň sa vynuluje reťazec coordinates, ktorý ukladá nashromáždené súradnice.
- **Príkaz "get coords":** Ak je prijatý príkaz "get coords", aktuálne uložené súradnice sa odošlú späť cez Bluetooth pomocou SerialBT.println(coordinates). Zber dát možno zastaviť nastavením collectingData na false.

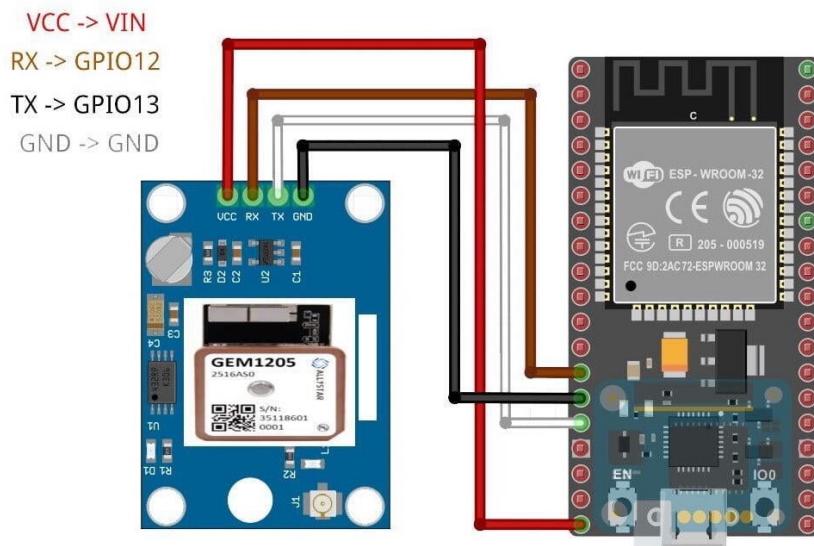
- **Zber a ukladanie gps dát**

- **Kontrola aktivity zberu dát:** Ak je zber dát aktívny (collectingData je true), program pokračuje kontinuálnym čítaním dát z GPS modulu.
- **Čítanie z GPS modulu:** Využíva sa prístupový bod Serial1, kde sú dátia GPS modulu dostupné. Funkcia Serial1.available() zistí, či sú dátia pripravené na čítanie.
- **Dekódovanie GPS dát:** Pre každý prečítaný znak z GPS modulu sa volá gps.encode(c), čo umožňuje knižnici TinyGPS++ spracovať a dekódovať GPS dátia.
- **Ukladanie súradníc :** ak sú aktuálne GPS dátia platné (overenie cez gps.location .isValid()), získajú sa zemepisná šírka (lat) a zemepisná dĺžka (lng) a pridajú sa do reťazca coordinates vo formáte "lat,lng;". Tento formát umožňuje ľahké odoslanie viacerých súradníc naraz.

Vďaka tomuto kódu môže užívateľ na diaľku zapnúť alebo vypnúť zber GPS dát a následne získať zozbierané dátia cez Bluetooth, ktoré následne môže použiť na vyhodnotenie prejdenej a prepátranej trasy.

3.4 Prepojenie mikročipu ESP32 a gps modulu NEO M8N

Pre vytvorenie našeho zariadenia, pomocou ktorého budeme sledovať pohyb zvierat v teréne sme využili už viackrát spomínaný mikročip ESP32 s gps modulom NEOM8N. Zapojenie zariadenia vyzerá nasledovne:

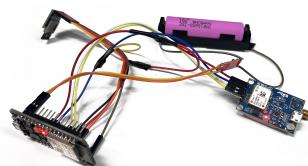


Obr. 6: Zapojenie ESP32

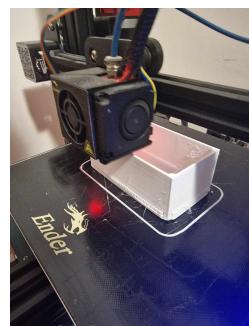
- **VCC** - napájací pin, ktorý slúži na pripojenie k pozitívному napätiu napájacieho zdroja. Pre neoM8N toto je typicky 3.3V alebo 5V, v závislosti od špecifikácie modulu.
- **GND** - zemný pin (Ground), ktorý sa pripája na negatívny pól alebo zem elektrického obvodu. Tento pin je nevyhnutný pre uzavretie elektrického obvodu a správne fungovanie zariadenia.
- **TX (Transmit)** - vysielač, teda pin, cez ktorý modul odosielá dátu. Pri pripojení GPS modulu k mikrokontroléru alebo inej platforme sa TX pin GPS modulu pripája na RX pin prijímacieho zariadenia.
- **RX (Receive)** - prijímač, teda pin, cez ktorý modul prijíma dátu. RX pin na GPS module sa pripája na TX pin odosielacieho zariadenia.

Ďalej sme použili tieto moduly:

- Modul na pamäťovú kartu - tento modul je využitý na ukladanie aktuálnej aktivity psa na pátracej akcii. Nakolko samotný čip ESP32 nedisponuje dostatočou kapacitou pamäte na ukladanie veľkého množstva súradníc, ktoré sú počas aktivity získané, pridali sme externú pamäť, ktorá spolahlivo zabezpečí dostatok pamäte.
- Modul na zapojenie externej batérie - celý zostrojený systém, ktorý bude nosiť pes na obojku je napájaný jedným 3.6V litium iónovým článkom 18650. Tento článok je možné opakovane nabíjať a má dostatočnú kapacitu, aby spolahlivo vydržal nabitý počas pátracej akcie.



(a) kompletné zapojenie ESP32



(b) Obal na obojok

Obr. 7: GPS obojok pre psa

Tento pozapájaný systém sme následne vložili do obalu, ktorý sme namodelovali a vytlačili na 3D tlačiarni.

3.5 Návrh a popis architektúry databázy

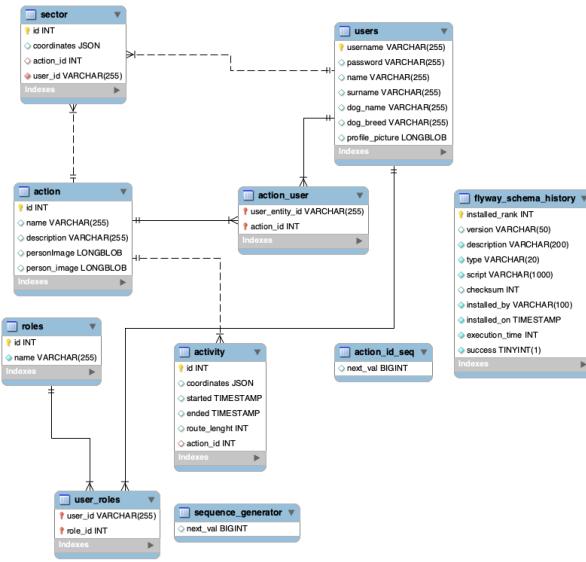
Po úspešnom implementovaní mikročipu ESP32 sme pokračovali implementáciou serverovej časti systému. Na správne fungovanie bol nevyhnutný návrh efektívnej architektúry, ktorá zabezpečí možnosť rôznych rozšírení do projektu a správny chod aplikácie. V databáze sme použili niekoľko N-N relácií, ktoré bolo následne potrebné ošetriť v spring-boot aplikácii kvôli rekurzii atribútov. Túto databázu sme inicializovali pomocou verziuovacieho frameworku flyway, ktorý v spolupráci s frameworkom Hibernate vytvoril všetky

relácie medzi jednotlivými tabuľkami, vytvoril samotné tabuľky a pridal im nami definované atribúty.

Listing 2: Časť sql kódu pre flyway script

```
1 CREATE TABLE roles (
2     id INT AUTO_INCREMENT PRIMARY KEY,
3     name VARCHAR(255) NOT NULL
4 );
5 CREATE TABLE users (
6     username VARCHAR(255) CHARACTER SET utf8mb4 COLLATE
7         utf8mb4_unicode_ci NOT NULL PRIMARY KEY,
8     password VARCHAR(255),
9     name VARCHAR (255),
10    surname VARCHAR (255),
11    dog_name VARCHAR (255),
12    dog_breed VARCHAR (255),
13    profile_picture LONGBLOB
14 )ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci
15 ;
16
17 CREATE TABLE user_roles (
18     user_id VARCHAR(255) CHARACTER SET utf8mb4 COLLATE
19         utf8mb4_unicode_ci NOT NULL ,
20     role_id INT NOT NULL ,
21     PRIMARY KEY (user_id, role_id),
22     FOREIGN KEY (user_id) REFERENCES users(username),
23     FOREIGN KEY (role_id) REFERENCES roles(id)
24 );
```

Vo vyššie uvedenom sql scrite uvádzame príklad použitia flyway scriptu. Konkrétnie v uvedenom príklade vytvárame tri tabuľky: roles, users a user-role, ktoré si podrobne popíšeme po uvedení celého ER (Entity-Relationship) databázového diagramu, ktorý znázorňuje štruktúru databázy a vzťahy medzi tabuľkami.



Obr. 8: ER diagram serverovej časti

- Tabuľka **users**: obsahuje informácie o užívateľoch. Tabuľka obsahuje stĺpce: username (primárny kľúč), password, name, surname, dog name, dog breed, profile picture (obrázok profilu užívateľa).
- Tabuľka **roles**: definuje rôzne role, ktoré môžu užívatelia zaujať. Stĺpce: id (primárny kľúč), name (názov role).
- Tabuľka **user roles**: určuje, ktoré role sú priradené jednotlivým užívateľom. Stĺpce: user id (odkaz na users.username), role id (odkaz na roles.id). Obsahuje zložený primárny kľúč sestavený z user id a role id.
- Tabuľka **action**: Obsahuje informácie o rôznych pátracích akciách, ku ktorým môžu byť priradení psovodi. Stĺpce: id (primárny kľúč), name, description, personImage (obrázok osoby).
- Tabuľka **activity**: sleduje aktivitu, ktorú začne psovod počas konkrétnej pátracej akcie. Je v nej uložená najmä celá prejdená trasa psovoda. Stĺpce: id (primárny kľúč), coordinates (súradnice vo formáte JSON), started (začiatok aktivity), ended (koniec aktivity), route lenght (dlžka trasy), action id (odkaz na action.id).
- Tabuľka **sector**: zaznamenáva informácie o jednotlivých sektoroch, ktoré sú súčasťou pátracej akcie. Stĺpce: id (primárny kľúč), coordinates, action id (odkaz na action.id), user id (odkaz na users.username).

- **Tabuľka action user:** vytvára vzťah medzi užívateľmi a akciami. Stĺpce: user entity id (odkaz na users.username), action id (odkaz na action.id). Obsahuje zložený primárny kľúč z user entity id a action id.

Vzťahy medzi tabuľkami sú reprezentované čiarami s koncovými šípkami, ktoré naznačujú cudzie kľúče a smer vzťahov. Dotted lines symbolizujú možné priradenie viacerých entít na jednu stranu vzťahu (napríklad užívateľ môže mať viacero rôl). Okrem hlavných entít diagram zobrazuje aj pomocné tabuľky ako "flyway schema history", ktorá slúži na sledovanie histórie zmien databázového schématu zavedených pomocou migrácií Flyway, a sequence generator ako potenciálneho generátora sekvenčí pre autoinkrementované hodnoty. Vďaka tomuto diagramu je jednoduchšie lepšie porozumenie a vizualizácia štruktúry databázy, umožňuje lepšiu predstavu o vzťahoch medzi dátami a pomáha pri návrhu a vývoji nášho systému.

3.6 Autorizácia a autentifikácia užívateľa

Tento kód je konfigurácia Spring Security pre serverovú časť aplikácie. Tá využíva JWT (JSON Web Token) pre autentifikáciu a autorizáciu. Kód je organizovaný do Spring konfiguračnej triedy označenej anotáciami @Configuration a @EnableWebSecurity, ktoré signalizujú Springu, aby použil túto triedu pre nastavenie bezpečnostných nastavení.

Listing 3: Autorizácia a autentifikácia užívateľa

```

1  @Configuration
2  @EnableWebSecurity
3  public class SecurityConfig {
4
5      private JwtAuthEntryPoint authEntryPoint;
6
7      private final CustomUserDetailsService userDetailsService;
8
9      @Autowired
10     public SecurityConfig(CustomUserDetailsService
11         userDetailsService,
12         JwtAuthEntryPoint authEntryPoint) {
13             this.userDetailsService = userDetailsService;
14             this.authEntryPoint = authEntryPoint;
15         }

```

```

16  @Bean
17
18      public SecurityFilterChain filterChain(HttpSecurity http)
19          throws Exception {
20
21          http
22              .csrf().disable()
23              .exceptionHandling()
24              .authenticationEntryPoint(authEntryPoint)
25              .and()
26              .sessionManagement()
27              .sessionCreationPolicy(SessionCreationPolicy.
28                  STATELESS)
29              .and()
30              .authorizeRequests()
31              .requestMatchers("/api/auth/**", "/ws").permitAll()
32              .anyRequest().authenticated()
33              .and()
34              .httpBasic();
35
36          http.addFilterBefore(jwtAuthenticationFilter(),
37              UsernamePasswordAuthenticationFilter.class);
38
39      return http.build();
40
41  }
42
43  @Bean
44  AuthenticationManager authenticationManager(
45      AuthenticationConfiguration authenticationConfiguration)
46      throws Exception {
47
48      return authenticationConfiguration.
49          getAuthenticationManager();
50
51  }
52
53  @Bean
54  PasswordEncoder passwordEncoder() {
55
56      return new BCryptPasswordEncoder();
57
58  }
59
60  @Bean
61
62      public JWTAuthenticationFilter jwtAuthenticationFilter() {

```

```

48         return new JWTAuthenticationFilter();
49     }
50 }

```

Vysvetlime si jednotlivé komponenty tejto bezpečnostnej triedy a ich využitie:

- **JwtAuthEntryPoint:** komponent sa používa ako vstupný bod pre spracovanie chýb autentifikácie, zvyčajne poskytujúc odpovede klientovi pri neautorizovaných pokusoch o prístup k zabezpečeným zdrojom.
- **CustomUserDetailsService:** služba používaná Spring Security na načítanie údajov o užívateľovi (ako sú užívateľské meno, heslo a oprávnenia) z aplikácie alebo databázy. V tomto kontexte sa používa pre načítanie a overenie údajov užívateľa počas procesu autentifikácie.
- **SecurityFilterChain:** hlavná metóda filterChain(HttpSecurity http) konfiguruje zabezpečenie aplikácie. Niektoré z nastavení zahrnutých v tejto metóde sú:
- **CSRF Protection Disabled:** Cross-Site Request Forgery (CSRF) ochrana je zakázaná, čo je bežné pre API, ktoré slúži klientom, ktorí nie sú vykreslení na serveri.
- **Session Management:** nastavenie politiky vytvárania session na STATELESS znamená, že aplikácia nevyužíva session pre ukladanie užívateľského stavu.
- **Request Authorization:** nastavenia autorizácie pre HTTP požiadavky, kde niektoré URL cesty sú dostupné bez autentifikácie (/api/auth/**, /ws), zatiaľ čo ostatné požiadavky vyžadujú, aby bol užívateľ autentifikovaný.
- **JWT Authentication Filter:** jwtAuthenticationFilter() je bean, ktorý definuje filter pre každú prichádzajúcu HTTP požiadavku, kde sa kontrolujú JWT tokeny na overenie autenticity užívateľa pred sprístupnením zabezpečených zdrojov.
- **Password Encoder:** passwordEncoder() definuje BCryptPasswordEncoder ako spôsob šifrovania a kontroly hesiel, čo poskytuje bezpečný spôsob na ich uloženie a porovnávanie.

Vďaka tomuto kódu vie frontendová časť bezpečne komunikovať so serverovou. Po prihlásení sa užívateľovi vygeneruje JWT token, ktorý má platnosť 2 hodiny. Po vypršaní jeho platnosti je užívateľ v prípade neaktivity automaticky odhlásený.

3.7 Prihlásenie užívateľa a generovanie JWT tokenu

Listing 4: Autorizácia a autentifikácia užívateľa

```
1  @PostMapping("login")
2  public ResponseEntity<AuthResponseDTO> login(@RequestBody
3      LoginDto loginDto){
4          Authentication authentication = authenticationManager.
5              authenticate(
6                  new UsernamePasswordAuthenticationToken(
7                      loginDto.getUsername(),
8                      loginDto.getPassword()));
9          SecurityContextHolder.getContext().setAuthentication(
10             authentication);
11         String token = jwtGenerator.generateToken(authentication)
12         ;
13         return new ResponseEntity<>(new AuthResponseDTO(token),
14             HttpStatus.OK);
15     }
```

Táto metóda login je anotovaná s @PostMapping("login"), čo znamená, že ide o HTTP POST požiadavku na ceste /login. Metóda sa používa na autentifikáciu užívateľov v Spring Boot aplikácii s použitím JWT (JSON Web Token) autentifikácie. Metóda vykonáva nasledovné dôležité kroky:

- **Prijatie požiadavky:** metóda login očakáva, že dostane požiadavku s telom typu LoginDto, ktoré obsahuje užívateľské meno a heslo, ktoré chce užívateľ použiť na prihlásenie. @RequestBody anotácia indikuje, že parameter loginDto by mal byť naplnený z tela požiadavky.
- **Autentifikácia:** používa authenticationManager, ktorý je štandardným mechanizmom Spring Security pre autentifikáciu, s volaním authenticate metódy. Príkaz new UsernamePasswordAuthenticationToken(loginDto.getUsername(), loginDto.getPassword()) vytvorí autentifikačný token s užívateľským menom a heslom získanými z loginDto. Výsledok autentifikácie sa priradí do premennej authentication.
- **Nastavenie bezpečnostného kontextu:** SecurityContextHolder.getContext() .setAuthentication(authentication) nastaví bezpečnostný kontext aplikácie na údaje o autentifikácií aktuálneho užívateľa. Tento krok je dôležitý, pretože určuje, že užívateľ je teraz prihlásený a jeho identity je známa v rámci aplikácie.

- **Generovanie JWT:** jwtGenerator .generateToken(authentication) volá metódu, ktorá generuje JWT token pre autentifikovaného užívateľa. JWT je štandard pre bezpečný prenos informácií medzi stranami a môže obsahovať rôzne tvrdenia (claims), ako sú užívateľské meno, role a platnosť tokenu.
- **Odoslanie odpovede:** return new ResponseEntity<>(new AuthResponseDTO (token), HttpStatus.OK) vráti odpoveď klientovi s HTTP statusom OK (200). Odpoveď obsahuje objekt AuthResponseDTO, ktorý obaluje vygenerovaný JWT token.

Výsledkom je, že keď užívateľ posielá správne užívateľské meno a heslo na koncový bod /login, systém ich overí a v prípade úspešnej autentifikácie vráti JWT token, ktorý potom môže byť použitý na prístup k zabezpečeným zdrojom v aplikácii.

3.8 Vytváranie novej akcie na strane servera

Jednou z klúčových funkcia našeho systému je možnosť vytvoriť inštanciu novej pátracej akcie, ktorá na backende uchováva informáciu o jej užívateľoch, sektorech a fotku hľadanej osoby. V nasledujúcim príklade kódu máme metódu, ktorá slúži na pridanie alebo aktualizáciu objektu Action v databáze a jeho pridružených Sector objektov. Tu je detailný popis činnosti tejto metódy:

- Anotácia **@PostMapping**: táto anotácia označuje, že metóda addAction bude spracovať HTTP POST požiadavky. V kontexte RESTful webových služieb, POST sa zvyčajne používa na vytváranie nových záznamov.
- Metóda **addAction(@RequestBody Action action)**: metóda prijíma objekt Action, ktorý je zaslaný ako časť tela (body) HTTP požiadavky. @RequestBody anotácia automaticky deserializuje telo požiadavky na objekt Action.
- **Uloženie objektu Action**: na začiatku metódy sa objekt Action ukladá do databáze pomocou actionRepository.save(action). Toto uloženie pravdepodobne vytvorí nový záznam alebo aktualizuje existujúci záznam v databáze.
- **Spracovanie sektorov**: metóda potom inicializuje prázdny zoznam sectorsToSet. Prechádza cez každý sektor priradený k akcii pomocou cyklu for-each. Pre každý sektor nastaví akciu na práve uloženú akciu (sector.setAction(actionToSave)) a potom sektor uloží pomocou sectorService.addNewSector(sector). Uložený sektor sa pridá do zoznamu sectorsToSet.

- **Nastavenie sektorov** akcie: po dokončení cyklu, metóda nastaví zoznam sectorsToSet ako sektory práve uloženej akcie (actionToSave.setSectors(sectorsToSet)).
- **Uloženie** a vrátenie upravenej akcie: nakoniec sa celá upravená akcia actionToSave znova uloží do databázy pomocou actionRepository.save(actionToSave) a vráti sa ako odpoveď volajúcemu.

Tento prístup zabezpečuje, že všetky zmeny na objekte Action a jeho pridružených Sector objektoch sú správne uložené a korektne reflektované vo vrátenej hodnote.

3.9 Získavanie akcií pre konkrétneho užívateľa

V našom systéme je potrebné, aby dokázal odfiltrovať zo všetkých pátracích akcií len konkrétné, ktoré obsahujú sektor pre prihláseného užívateľa. Na tento účel sme implementovali metódu, ktorá je určená na získanie všetkých akcií (actions) asociovaných s konkrétnym používateľom. Podrobne si popíšeme funkciu tejto metódy:

- **Anotácia @GetMapping("/username")**: táto anotácia určuje, že metóda bude reagovať na HTTP GET požiadavky na URL adresu, ktorá obsahuje používateľské meno ako súčasť cesty. Napríklad, ak je cesta nastavená na /john, metóda spracuje požiadavku pre používateľa s menom "john".
- **Anotácia @ResponseBody**: táto anotácia indikuje, že návratová hodnota metódy bude použitá ako telo (body) odpovede HTTP. Spring používa HTTP Message Converters na konverziu návratového objektu do formátu JSON alebo iného médiatypu, ktorý je nastavený v HTTP hlavičke Accept.
- **Metóda getAllActionsOfUser(@PathVariable(username) String username)**: metóda prijíma jeden parameter, username, ktorý je extrahovaný z URL. Anotácia @PathVariable uvádzia, že hodnota z časti URL sa má mapovať na tento parameter.
- **Získanie akcií užívateľa**: v tele metódy sa volá actionRepository .findAllActionsByUser(username), čo je metóda definovaná v rámci repozitára JPA, ktorý si nižšie uvedieme tiež. Táto metóda vyhľadá všetky objekty typu Action, ktoré sú asociované s daným používateľským menom.
- **Návrat zoznamu akcií**: Návratová hodnota findAllActionsByUser (List<Action>) sa priamo vracia ako odpoveď. Vďaka anotácii @ResponseBody, Spring konvertuje tento zoznam na JSON a odosiela ho ako odpoveď na HTTP požiadavku.

Listing 5: Autorizácia a autentifikácia užívateľa

```
1 @Query(value = "SELECT DISTINCT a.* FROM action a " +
2         "INNER JOIN sector s ON a.id = s.action_id " +
3         "WHERE s.user_id = ?1",
4         nativeQuery = true)
5 List<Action> findAllActionsByUser(String username);
```

V uvedenom kóde je definícia metódy rozhrania repozitára v rámci frameworku Spring Data JPA. Metóda slúži na vyhľadanie a získanie všetkých akcií, ktoré sú asociované s konkrétnym používateľom, identifikovaným cez jeho username.

- **Anotácia @Query:** táto anotácia je použitá na definovanie SQL dopytu, ktorý sa má vykonať. Je to významná časť, keďže umožňuje priamo zadávať SQL, čo je užitočné, keď štandardné metódy poskytované Spring Data JPA nie sú dostatočné alebo efektívne pre špecifické požiadavky.
- **Parameter value:** tento parameter obsahuje SQL príkaz. V prípade tohto kódu: SELECT DISTINCT a.*: Vyberie všetky stĺpce z tabuľky action, ktorá je označená ako a. Použitie DISTINCT zabezpečuje, že každá akcia bude v zozname iba raz, aj keď by mohli existovať viacnásobné prepojenia cez tabuľku sector.
- **INNER JOIN sector s ON a.id = s.action id:** ide o spojenie (JOIN) s tabuľkou sector, ktorá je označená ako s. Spojenie je na základe stĺpca id tabuľky action a action id tabuľky sector. Toto zabezpečuje, že vrátené akcie sú tie, ktoré majú odpovedajúce záznamy v tabuľke sector.
- **WHERE s.user id = ?1:** táto časť obmedzuje dopyt na záznamy, kde stĺpec user id v tabuľke sector zodpovedá hodnote poskytnutej parametrom. ?1 znamená, že hodnota pre tento parameter sa získa z prvého parametra metódy, ktorý sa volá.
- **nativeQuery = true:** tento parameter anotácie @Query informuje Spring Data JPA, že dopyt je natívny SQL dopyt a nie JPQL (Java Persistence Query Language). To umožňuje použitie priamo SQL syntaxe špecifické pre databázu, ktorá je používaná.
- **Parameter metódy String username:** metóda prijíma parameter username, ktorý sa používa v SQL dopyte na určenie, ktorého používateľa akcie majú byť získané. Tento parameter sa mapuje na ?1 v SQL dopyte.

3.9.1 Hlavný komponent frontendovej časti

Hlavným komponentom našej frontendovej časti je súbor app.js, v ktorom sa nachádza funkcia vracajúca celý obsah.

Listing 6: Hlavný komponent frontendu

```
1   return (
2     token !== null ? (
3       <SafeAreaProvider>
4         <NavigationContainer>
5           <BottomNavigation
6             navigationState={{ index, routes }}
7             onIndexChange={setIndex}
8             renderScene={renderScene}
9           />
10          </NavigationContainer>
11        </SafeAreaProvider>
12      ) : (
13        <LoginScreen onSignIn={(token) => {
14          SecureStore.setItemAsync("accessToken", token);
15          setToken(token);
16        }} userInfo={(userInfo) => setUserInfo(userInfo)} />
17      )
18    );
19  }
```

Táto implementácia komponentu používa navigáciu a autentifikáciu. Tu je podrobnejší rozbor toho, čo robí každá časť kódu:

- **Podmienené zobrazenie:** kód používa ternárny operátor (`token !== null ? X : Y`), ktorý je základným prvkom jazyka JavaScript. Tento operátor kontroluje, či hodnota `token` nie je `null`. Ak `token` existuje (nie je `null`), zobrazí sa navigačný kontajner s navigačnými zložkami. V opačnom prípade sa zobrazí prihlásovacia obrazovka.
- **SafeAreaProvider a NavigationContainer:** ak je užívateľ prihlásený (`token` nie je `null`), kód zabalí navigačné komponenty do `<SafeAreaProvider>` a `<NavigationContainer>`. `SafeAreaProvider` je komponent, ktorý zabezpečuje, že UI (užívateľské rozhranie) sa nezobrazuje v oblastiach, ktoré by mohli byť zakryté, napríklad na výrezoch displeja alebo na senzorových ostrovoch. `NavigationContainer` je kontajner, ktorý obaluje navigačné komponenty a poskytuje navigačnú logiku.

- **BottomNavigation:** ide o komponent, ktorý poskytuje navigáciu na spodnej časti obrazovky. Vlastnosti navigationState a onIndexChange sú použité na kontrole stavu navigácie. navigationState obsahuje aktuálny index (ktorá záložka je aktívna) a cesty (routes), zatiaľ čo onIndexChange je funkcia, ktorá aktualizuje tento index, keď užívateľ vyberie inú záložku.
- **renderScene:** funkcia renderScene sa volá na vykreslenie scény (komponentu alebo obrazovky) na základe aktuálneho stavu navigácie. Táto funkcia zrejme definuje, aký obsah sa má zobraziť pre každú záložku v BottomNavigation.
- **LoginScreen:** ak užívateľ nie je prihlásený (token je null), zobrazí sa komponent LoginScreen. Tento komponent obsahuje prop onSignIn, ktorý je funkciou. Keď užívateľ vykoná prihlasovanie, táto funkcia sa volá s token ako argumentom. Vo vnútri onSignIn, SecureStore.setItemAsync(accessToken, token) ukladá token do bezpečného úložiska a setToken(token) aktualizuje stav tokenu v aplikácii.

3.10 Spresnenie lokalizácie pomocou akcelerometra

Počas implementácie našej práce sme narazili na problém nepresnosti získavania polohy, pomocou mobilného zariadenia. Tento problém sme vyriešili zapojením ďalšieho senzora, ktorým každý moderný mobilný iOS alebo android telefón disponuje.

Listing 7: Zapojenie akcelerometra

```

1   Accelerometer.setUpdateInterval(1000);
2       const subscription = Accelerometer.addListener(
3           accelerometerData => {
4               const { x, y } = accelerometerData;
5               const angle = Math.atan2(y, x) * (180 / Math.
6                   PI);
7               if (isSubscribed) {
8                   setCurrentDirectionAngle(angle);
9               }
10      });

```

Vďaka akcelerometru môže tento kód merať zrýchlenie zariadenia v rôznych smeroch. Tu je podrobnejší rozbor toho, ako kód funguje:

- **Nastavenie intervalu aktualizácie akcelerometra:** prvý riadok kódu, Accelerometer.setUpdateInterval(1000);, nastavuje frekvenciu, s ktorou akcelerometer po-

skytuje údaje. Hodnota 1000 znamená, že údaje sa aktualizujú každú sekundu (1000 milisekúnd).

- **Pridanie počúvača (listener) na akcelerometer:** const subscription = Accelerometer.addListener(accelerometerData => ...) pridáva funkciu počúvača na akcelerometer. Táto funkcia sa spustí vždy, keď akcelerometer získa nové údaje. accelerometerData je objekt obsahujúci údaje z akcelerometra, konkrétnie zrýchlenia na osiach x a y. **Spracovanie údajov z akcelerometra:** dekonštrukcia objektu: const x, y = accelerometerData; extrahuje hodnoty x a y z údajov akcelerometra. Tieto hodnoty predstavujú zrýchlenie zariadenia na osiach x a y.
- **Výpočet uhla:** const angle = Math.atan2(y, x) * (180 / Math.PI); používa funkciu Math.atan2, ktorá vypočíta arktangent dvoch čísel, čo je užitočné na zistenie uhla rotácie zariadenia v stupňoch. Math.PI je konštanta pi a prevádzza uhly z radiánov na stupne.
- **Aktualizácia stavu:** podmienka: if (isSubscribed) ... kontroluje, či je zariadenie stále prihlásené na odber údajov z akcelerometra.
- **Nastavenie smeru:** setCurrentDirectionAngle(angle); aktualizuje stav aplikácie s novým uhlom, ktorý zodpovedá aktuálnej orientácii zariadenia.
- **Uloženie odoberania na odber údajov:** premenná subscription drží referenciu na odber údajov. Táto referencia sa môže neskôr použiť na zrušenie odberu, ak je to potrebné.

3.11 Parsovanie JWT tokenu

Ako sme už vyššie spomínali, užívatelia sa autorizujú vďaka JWT tokenu, ktorý je uložený na serveri. Ak je token validny, užívateľ môže vykonávať vďaka endpointom jednotlivé akcie.

Listing 8: Parsovanie JWT tokenu

```
1  async function getUsernameFromJwt(token) {  
2      const base64Url = token.split('.')[1];  
3      const base64 = base64Url.replace(/-/g, '+').replace(/_/g,  
4          '/');  
5      const jsonPayload = decodeURIComponent(atob(base64).split  
6          ('')).map(function (c) {
```

```

5           return '%' + ('00' + c.charCodeAt(0).toString(16)).
6           slice(-2);
7       }).join(''));
8   const payload = await JSON.parse(jsonPayload);
9   return payload.sub;
}

```

V uvedenom kóde vyššie môžeme vidieť asynchrónnu funkciu getUsernameFromJwt, ktorá dekóduje JWT (JSON Web Token) a získava z neho užívateľské meno (alebo subjekt, teda sub). Táto funkcia sa delí na jednotlivé kroky:

- **Rozdelenie tokenu:** token je najskôr rozdelený na časti pomocou bodky (.), čo je štandardný formát JWT, ktorý obsahuje tri časti: hlavičku, obsah (payload) a podpis. Funkcia vyberie druhú časť, ktorá obsahuje potrebné údaje (payload).
- **Úprava base64 kódovania:** JWT je zakódovaný v base64url, ktorý je mierne odlišný od štandardného base64 kódovania. Nahradenie znakov - za + a podtržítka za / konvertuje base64url na štandardné base64 kódovanie, ktoré možno ľahko dekódovať.
- **Dekódovanie base64 retazca:** dekódované base64 na retazec je vykonané pomocou funkcie atob(), ktorá transformuje zakódovaný base64 retazec na pôvodný retazec znakov.
- **Percentové kódovanie na URI komponenty:** po dekódovaní base64, každý znak retazca je prevedený na jeho percentové kódovanie. Toto sa vykonáva pre každý znak retazca, kde sa jeho charakterová hodnota (získaná funkciami charCodeAt(0)) prevedie na hexadecimálny formát. Tento krok pomáha správne naformátovať retazec pre ďalšie spracovanie.
- **Parsovanie JSONu:** výsledný retazec sa potom parsovaný na JSON objekt s použitím JSON.parse(). Tento JSON objekt obsahuje všetky dátá uložené v JWT, vrátane užívateľského mena (subjektu).
- **Vrátenie užívateľského mena:** Nakoniec funkcia vráti hodnotu sub (subjektu) z dekódovaného payloadu, ktorá je zvyčajne používaná na identifikáciu užívateľa.

3.12 Controllery na strane užívateľa

V projekte frontendu máme pre každý dostupný objekt na backende vlastný controller, ktorým ovládame jednotlivé endpointy na backende. Nakolko máme takýchto controllers, tak máme aj ich controllery na strane užívateľa.

lerov veda, uvedieme si len dva príklady:

Listing 9: Zapojenie akcelerometra

```
1   export const getAllActionsByUserName = async (userName, token
2     ) => {
3     try {
4       const response = await axios.get(`${
5         BASE_URL}/action/${
6           userName}`,
7           {
8             headers: {
9               Authorization: `Bearer ${token}`
10            }
11          });
12        if (response.status === 200) {
13          return response.data;
14        } else {
15          console.log(response.status)
16        }
17      } catch (error) {
18        console.log(error)
19        throw error;
20      }
21    };
22  
```

Tento kód predstavuje funkciu v JavaScripte, ktorú sme navrhli na asynchronné získavanie údajov z externého zdroja pomocou HTTP požiadavky. Funkcia getAllActionsByUserName je exportovaná, aby sa dala použiť v iných súboroch alebo moduloch. Funkciu by sme detailnejšie mohli rozobrať takto:

- **Definícia funkcie:** funkcia getAllActionsByUserName je asynchronná, čo znamená, že používa `async` klauzulu. Toto umožňuje vo vnútri funkcie používať `await` pre asynchronné operácie. Funkcia prijíma dva parametre, `userName` a `token`.
- **Try-catch blok:** kód je obalený v try-catch bloku. Tento prístup sa používa na zachytenie a spracovanie chýb, ktoré môžu vzniknúť pri asynchronných operáciách, ako je HTTP požiadavka.
- **HTTP požiadavka:** vo vnútri try bloku sa vykonáva HTTP GET požiadavka pomocou knižnice `axios`. URL adresa požiadavky je konštruovaná pomocou `BASE_URL`

(konšanta, ktorá obsahuje základnú URL API) a userName. Požiadavka zahŕňa aj hlavičku s autorizačným tokenom, ktorý je formátovaný ako Bearer token.

- **Spracovanie odpovede:** ak je stavový kód odpovede HTTP 200 (čo znamená úspešnú odpoveď), funkcia vráti dátu získané v odpovedi (response.data). Ak je stavový kód iný ako 200, funkcia zaloguje tento stavový kód pomocou console.log.
- **Zachytenie a spracovanie chýb:** ak dôjde k chybe počas vykonávania požiadavky (napríklad problém so sietou, neplatný token, alebo server neodpovedá), chyba je zachytená v catch bloku. Zachytená chyba je zalogovaná a následne znova vyvolaná (throw error), aby mohla byť spracovaná vyššie v kóde alebo aby informovala volajúcu funkciu o probléme.

Druhý príklad, ktorý si ukážeme, je pomerne komplikovanejší nakoľko v sebe volá ďalšie API volanie.

```
1 [caption={Kontroler pre vytvorenie novej akcie}, label=lst:code11
  ]
2 export const addNewAction = async (data, token, searchPersonImage
  ) => {
3   console.log("Adding new action")
4   try {
5     const response = await axios.post(`${BASE_URL}/action`,
6       data, {
7         headers: {
8           Authorization: `Bearer ${token}`,
9           'Content-Type': 'application/json'
10        }
11      });
12
13     const actionId = response.data.id;
14     console.log("Action created successfully with ID:",
15       actionId);
16     const formData = new FormData();
17     formData.append("personImage", {
18       uri: searchPersonImage.assets.at(0).uri,
19       type: 'image/jpeg',
20       name: 'searchPersonImage.jpg',
21     });
22
23   } catch (error) {
24     console.error(error);
25   }
26 }
```

```

20
21     const imageResponse = await axios.put(`${BASE_URL}/action
22         `/`${actionId}`, formData, {
23             headers: {
24                 Authorization: `Bearer ${token}`,
25                 'Content-Type': 'multipart/form-data'
26             }
27         });
28
29     if (imageResponse.status === 200) {
30         console.log("Image uploaded successfully");
31         return imageResponse.data;
32     } catch (error) {
33         console.error("Error in action creation or image upload:"
34             , error);
35         throw error;
36     }
37 };

```

- **Definícia a začiatok funkcie:** funkcia addNewAction je asynchronná a prijíma tri parametre: data (údaje o novej akcii), token (autorizačný token pre autentifikáciu), a searchPersonImage (informácie o obrázku hľadanej osoby). Na začiatku funkcie sa zobrazí logovacia správa "Adding new action", ktorá indikuje, že proces pridávania novej akcie začal.
- **Odoslanie údajov akcie:** vykoná sa POST požiadavka na API endpoint /action s údajmi data. Autorizačný token a typ obsahu sú zahrnuté v hlavičkách požiadavky. Po úspešnom odoslaní požiadavky sa z odpovede (response.data) extrahuje ID novej akcie (actionId), a zaloguje sa správa o úspešnom vytvorení akcie s týmto ID.
- **Príprava obrázka na nahratie:** inicializuje sa objekt FormData, ktorý sa používa na formátovanie údajov potrebných na odoslanie súborov. Do formData sa pridá obrázok osoby. Detaily obrázka (URI, typ, názov) sa získavajú z prvého prvku zoznamu aktív, ktorý je súčasťou searchPersonImage.
- **Nahratie obrázka:** vykoná sa PUT požiadavka na API endpoint /action/actionId, kde actionId je ID akcie získané z predchádzajúcej odpovede. Táto požiadavka obsa-

huje formData s obrázkom a v hlavičkách požiadavky je uvedený autorizačný token a typ obsahu multipart/form-data. Ak odpoveď na nahrávanie obrázka (imageResponse) má stavový kód 200, zaloguje sa správa o úspešnom nahratí obrázka a funkcia vráti dátá získané v tejto odpovedi.

- **Spracovanie chýb:** ak v ktoromkoľvek kroku dôjde k chybe (napríklad zlyhanie siete, chyba servera, neplatný token), chyba sa zachytí v bloku catch. Zachytená chyba sa zaloguje s príslušnou správou a následne sa znova vyvolá (throw error), čím sa signalizuje, že funkcia zlyhala a umožňuje sa ďalšie spracovanie chyby vyššie v kóde.

Tento kód je typickým príkladom implementácie API volaní na pridanie a spracovanie údajov v moderných webových alebo mobilných aplikáciách, kde sa často používajú asynchronné funkcie na zabezpečenie plynulosti užívateľského rozhrania počas sietových operácií.

3.13 Získavanie polohy na strane užívateľa

V tejto časti si popíšeme kód, ktorý sa nachádza na frontendovej časti a získavame pomocou neho polohu zariadenia, v ktorom je spustený náš systém.

Listing 10: Získavanie polohy na frontendovej časti

```
1   Location.requestForegroundPermissionsAsync()
2       .then(({ status }) => {
3           if (status !== 'granted') {
4               console.error("Permission to access location
5                           was denied");
6               setLoading(false);
7               return;
8           }
9
10          const updateLocation = async () => {
11              const locationData = await Location.
12                  getCurrentPositionAsync({});
13              if (isSubscribed) {
14                  setLocation(locationData);
15                  webSocket.send(JSON.stringify(
16                      locationData));
17
18      }
19
20      setLoading(true);
21
22      const interval = setInterval(() => {
23          updateLocation();
24      }, 1000);
25
26      return () => {
27          clearInterval(interval);
28      };
29  }
30
31  updateLocation();
32
33  return () => {
34      updateLocation();
35  };
36}
```

```

14         setRoute((currentRoute) => [...
15             currentRoute, {
16                 latitude: locationData.coords.
17                     latitude,
18                     longitude: locationData.coords.
19                         longitude
20                     }] );
21     }
22     updateLocation();
23     const locationInterval = setInterval(
24         updateLocation, 5000);
25     setLoading(false);

```

Tento kód využíva funkcie dostupné v rámci Expo Location API v mobilnej aplikácii React Native na získavanie a pravidelnú aktualizáciu polohy zariadenia. Tu je podrobný rozbor toho, ako kód funguje:

- **Žiadosť o prístupové práva k polohe:** metóda Location .requestForegroundPermissionsAsync() sa volá na získanie práv na prístup k polohe zariadenia v popredí aplikácie. Toto je nevyhnutné pre aplikácie, ktoré vyžadujú informácie o polohe používateľa. Výsledok tejto požiadavky sa spracuje pomocou metódy .then, ktorá prijíma funkciu s deštrukturovaným parametrom status obsahujúcim stav schválenia.
- **Overenie schválenia prístupu:** v prípade, že stav prístupu (status) nie je 'granted' (t.j. schválený), kód loguje chybu a deaktivuje indikátor načítania (setLoading(false)), a funkcia sa ukončí bez ďalšieho spracovania.
- **Definícia funkcie na aktualizáciu polohy:** vo vnútri podmienky, kde boli prístupové práva udelené, je definovaná asynchronná funkcia updateLocation. Táto funkcia získa aktuálnu polohu zariadenia volaním Location.getCurrentPositionAsync(). Ak je hodnota isSubscribed nastavená na TRUE (čo znamená, že komponent alebo entita stále existuje a čaká na dátu), aktualizuje sa stav polohy pomocou setLocation(locationData), a dátá o polohe sa odosielajú prostredníctvom WebSocketu vo formáte JSON na ďalšie spracovanie.
- **Pravidelné aktualizácie polohy:** Po definícii funkcie sa updateLocation ihned volá, aby sa okamžite získala poloha. Následne sa nastaví interval volania update-

Location každých 5000 milisekúnd (5 sekúnd) pomocou setInterval(updateLocation, 5000). To zabezpečí pravidelné aktualizácie polohy.

- **Vypnutie indikátora načítania:** Na konci procesu sa vypne premenná fungujúca ako indikátor načítania (setLoading(false)), čo signalizuje, že počiatočná požiadavka na prístup k polohe a prvotné získanie údajov bolo dokončené.

3.14 Pomocné služby na frontendovej časti

Rovnako ako na backendovej, tak aj na frontendovej časti máme v projekte rôzne služby, pod ktorými si môžeme predstaviť implementáciu logiky, ktorá je potrebná na strane frontendu. V tejto podkapitole si rozoberieme jednotlivé najpoužívanejšie služby, ktoré v projekte využívame.

Prvú, ktorú by sme radi spomenuli je imagePicker, ktorý využívame na získavanie obrázkov z mobilného zariadenia. Táto funkcia sa využíva napríklad pri volbe profilovej fotky užívateľa, alebo pri nahrávaní fotky nezvestnej osoby.

Listing 11: Získavanie obrázkov z knižnice

```
1  const pickImage = async () => {
2      const permissionResult = await ImagePicker.
3          requestMediaLibraryPermissionsAsync();
4      if (permissionResult.granted === false) {
5          Alert.alert("Permission required", "Permission to access
6              gallery is required!");
7          return;
8      }
9      const pickerResult = await ImagePicker.
10         launchImageLibraryAsync();
11     console.log(pickerResult)
12     if (!pickerResult.cancelled) {
13         setSearchPersonImage(pickerResult);
14     }
15 };
16 
```

Ide o službu, používa knižnicu Expo's ImagePicker na výber obrázkov z galérie zariadenia. Podrobnejší popis funkcie:

- **Žiadosť o prístupové práva:** funkcia začína získaním prístupových práv k médiami uloženým v zariadení pomocou metódy ImagePicker .requestMediaLibrary-

PermissionsAsync(). Toto je asynchrónna operácia, takže používa await na čakanie na odpoved.

- **Kontrola prístupových práv:** po získaní výsledku z prístupových práv (permissionResult), kód kontroluje, či bolo prístupové právo udelene (permissionResult.granted). Ak nie je právo udelene (granted === false), funkcia zobrazí výstrahu pomocou Alert.alert(), informujúc užívateľa, že je potrebné prístupové právo na prístup k galérii. Ak užívateľ nedá prístup, funkcia sa ukončí a ďalšie kroky sa nevykonajú.
- **Spustenie selektora obrázkov:** ak sú prístupové práva udelene, funkcia spustí selektor obrázkov pomocou ImagePicker.launchImageLibraryAsync(). Toto je ďalšia asynchrónna funkcia, ktorá otvára galériu zariadenia, umožňujúc užívateľovi vybrať obrázok.
- **Spracovanie výberu obrázka:** po výbere obrázka sa získa výsledok (pickerResult). Funkcia zaloguje tento výsledok pomocou console.log(pickerResult) na zobrazenie informácií o obrázku, ako sú jeho umiestnenie, typ a veľkosť. Ak výber obrázka neboli zrušený (!pickerResult.cancelled), znamená to, že užívateľ vybral obrázok. Výsledok výberu sa potom uloží pomocou funkcie setSearchPersonImage, ktorá aktualizuje stav aplikácie s novým obrázkom.

Ďalšími dôležitými službami, ktoré v našom projekte využívame na kľúčovú funkciu je obsluha udalostí, ktorú môžeme nazvať dotyk obrazovky na mape. Túto udalosť odchytávame pri vytváraní nových sektorov na mape, kedy chceme zakresliť jednotlivé sektory pre kynológov.

Listing 12: Obsluha dotyku na mape

```
1 const handleMapPress = (e) => {
2     if (addingSector) {
3         setCurrentSector([...currentSector, e.nativeEvent.
4                         coordinate]);
5     }
5;
```

- **Parameter:** funkcia prijíma objekt udalosti e ako argument.
- **Podmienená kontrola:** kontrola, či je premenná addingSector nastavená ako TRUE. Táto premenná označuje, či je aplikácia aktuálne v režime, kedy môže pridávať

nové sektory (alebo značky) na mapu. **Akcia:** ak je addingSector pravdivá, aktualizuje stav currentSector (pole súradníc) pridaním novej súradnice z objektu udalosti (e.nativeEvent.coordinate). To sa vykonáva pomocou operátora šírenia (...) na zahrnutie všetkých existujúcich súradníc v currentSector a potom pridaním novej súradnice na koniec.

Listing 13: Pridávanie sektorov

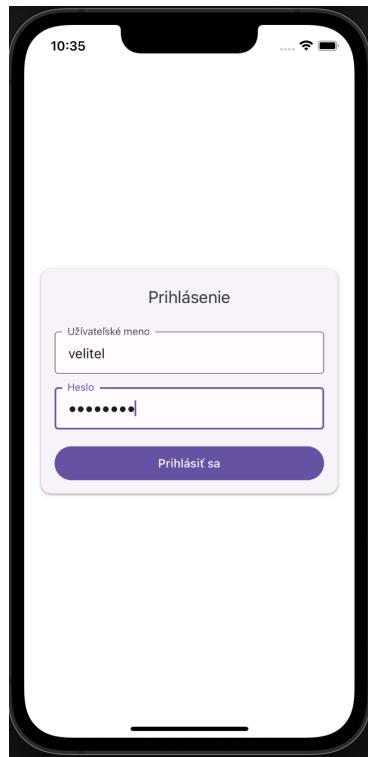
```
1 const toggleAddingSector = () => {
2     setAddingSector(!addingSector);
3     if (addingSector && currentSector.length > 0) {
4         setShowUserModal(true);
5     }
6 };
```

Táto funkcia prepína režim, ktorý umožňuje užívateľom pridávať sektory na mapu.

- Prepnutie stavu: prepne stav addingSector na jeho opačnú hodnotu (!addingSector). Toto prepínanie určuje, či môže užívateľ pridávať nové sektory na mapu.
- **Podmienená kontrola:** po prepínaní skontroluje, či bola addingSector pravdivá pred prepnutím a či currentSector obsahuje nejaké súradnice (currentSector.length > 0).
- **Akcia:** ak sú splnené obe podmienky, nastaví stav setShowUserModal na TRUE, čo spustí modálne okno na vytváranie sektorov. Tento modál môže byť použitý na to, aby veliteľ akcie mohol priradiť jednotlivé sektory užívateľom.

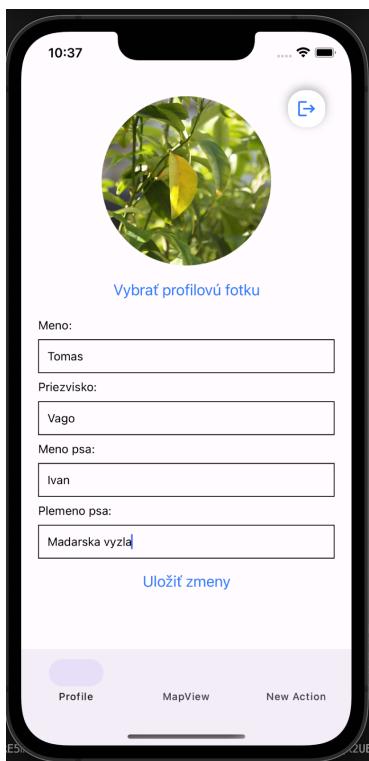
Funkcia toggleAddingSector slúži ako kontrolný mechanizmus na povolenie alebo zakázanie funkcie pridávania sektorov a vyzve užívateľa k ďalšej akcii prostredníctvom modálneho okna, ak vypne tento režim po pridaní niektorých sektorov.

3.15 Ukážka hlavných častí mobilného systému



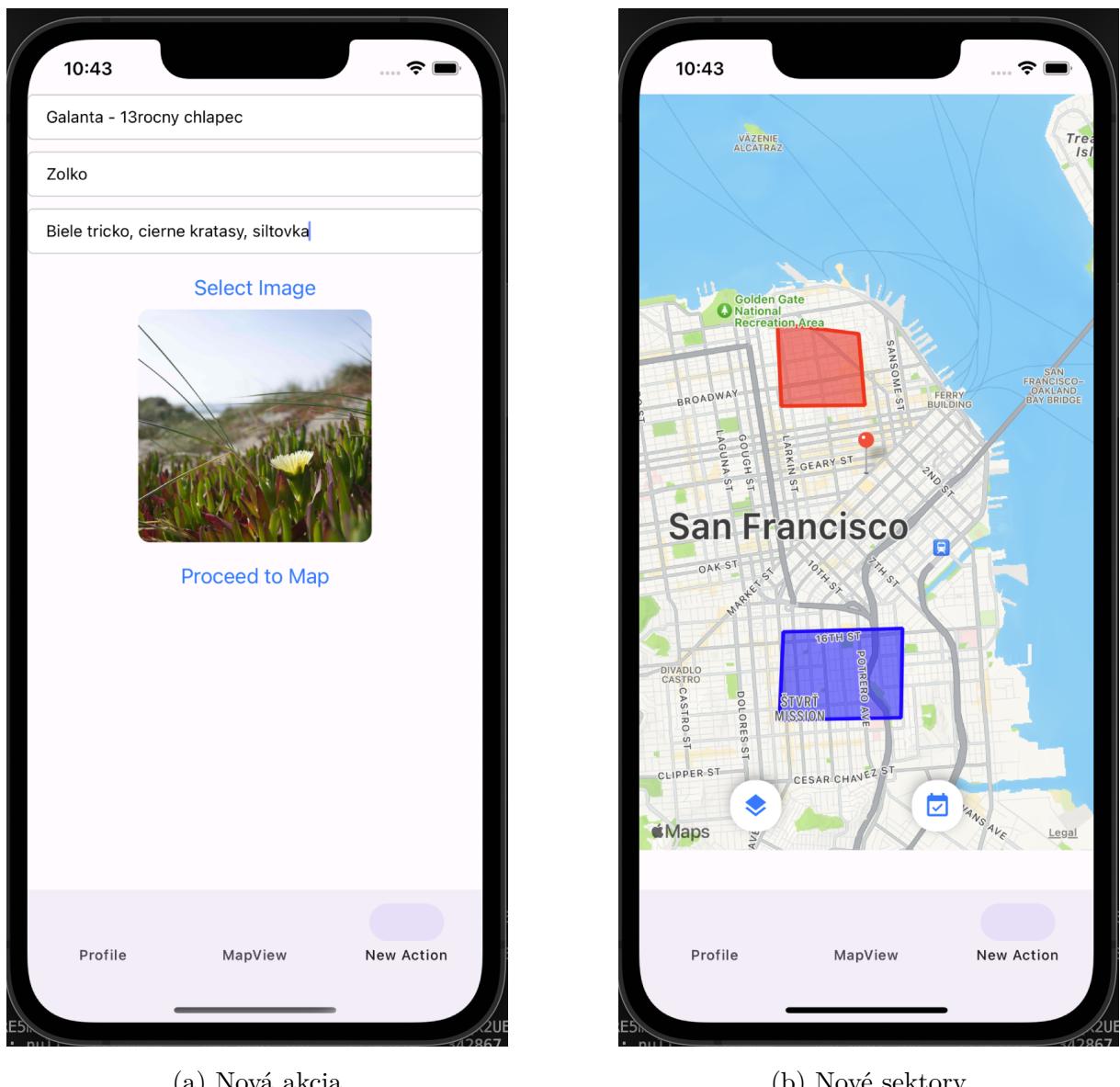
Obr. 9: Screen pre prihlásenie

Prvou úlohou, ktorú musí náš systém splniť, je možnosť prihlásenia pomocou užívateľského mena a hesla. V tejto časti vidí užívateľ na frontendovej časti dva vstupné objekty, do ktorých zadá meno a heslo a stlačí tlačidlo login pre prihlásenie. Po prihlásení je užívateľ presmerovaný automaticky na hlavnú obrazovku.



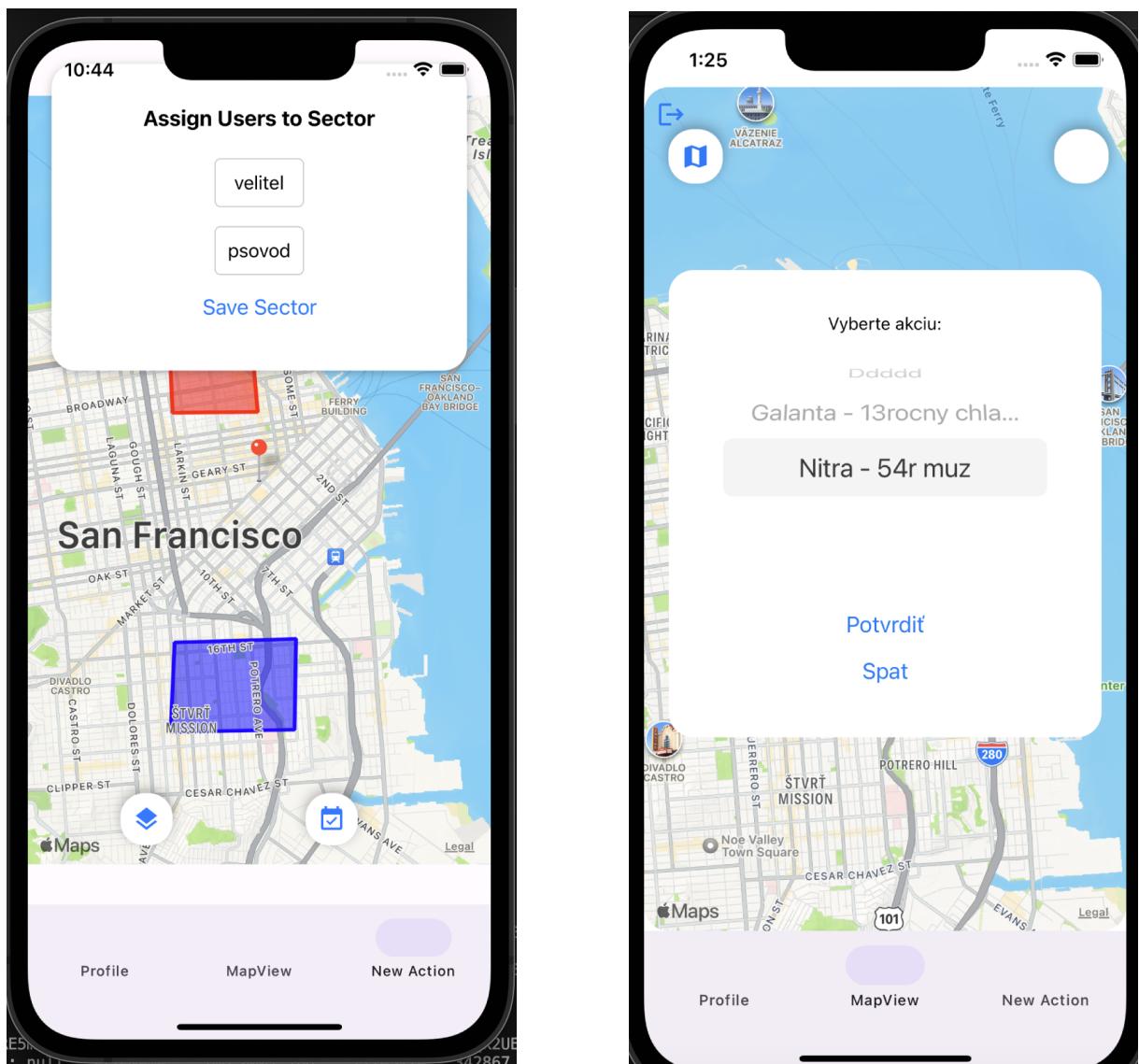
Obr. 10: Screen pre zmenu profilových údajov

Na tejto obrazovke si psovod alebo veliteľ zásahu vyplnia profil a profilovú fotku, ktorá sa zobrazí na mape ako ukazovateľ polohy.



Obr. 11: Postup vytvárania nových akcii a sektorov

V tomto bode sa vykonáva najdôležitejšia funkciu systému a tou je vytváranie novej pátracej akcie, ktorú priradia jednotlivým psovodom. Vytváranie je veľmi intuitívne a jednoduché, čo je dôležitou vlastnosťou nášho systému, pretože psovodi, pracujú na päťracích operáciach pod stresom. Užívateľ iba jednoducho vyplní názov pátracej akcie, meno nezvestnej osoby a nejaký základný popis o nezvestnom. Rovnako nahrá fotku nezvestného a prejde do ďalšieho kroku pomocou tlačidla : Proceed to map. Následne v ďalšom kroku užívateľ vytvára jednotlivé sektory, ktoré môže priradiť svojim kolegom psovodom, ktorí jednotlivé sektory budú prepátravať.



(a) Priradenie zo zoznamu užívateľov

(b) Výber akcie

Obr. 12: Priradovanie sektorov a zobrazovanie sektorov na akcii

Užívateľ si môže zvoliť sektor, ktorý mu bol priradený a začať prepátrvanie. Pričom trasa celého prehľadávania je celý čas zaznamenávaná.

Záver

Táto diplomová práca úspešne vyvinula GPS systém, ktorý zvyšuje schopnosti kynologických záchranných jednotiek poskytovaním presného, reálneho sledovania psovodov a ich psov. Zistenia naznačujú, že takáto technologická integrácia do záchranných operácií výrazne zlepšuje koordináciu, skracuje časy reakcie a zvyšuje celkovú efektivitu vyhľadávacích misií. Aj keď systém preukazuje podstatné výhody, boli zaznamenané určité obmedzenia, najmä v kontexte extrémne náročného terénu a v husto zarastenom lese. Budúca práca by sa mala zamerať na zvýšenie spoľahlivosti signálu v takýchto podmienkach a skúmanie robustnejších hardvérových riešení. Pokračujúci vývoj tohto systému má potenciál revolucionizovať efektivitu záchranných operácií, čo by malo významný dopad na záchrannu životov v núdzových situáciách.

Zoznam použitej literatúry

1. ARDUINO. *Arduino: datasheets*. 2024. Dostupné tiež z: <https://docs.arduino.cc/resources/datasheets/A000005-datasheet.pdf>.
2. *What is a Raspberry Pi?* 2022. Dostupné tiež z: <https://opensource.com/resources/raspberry-pi>.
3. VALERIY PAKULIN. *Photon and electron structure*. 2021. Dostupné tiež z: https://www.researchgate.net/publication/354136663_Photon_and_electron_structure.
4. DARKO HERCOG. *Design and Implementation of ESP32-Based IoT Devices*. 2023. Dostupné tiež z: <https://www.mdpi.com/1424-8220/23/15/6739>.
5. DARLLAINE LINCOPINIS. *A Review on Java Programming Language*. 2023. Dostupné tiež z: https://www.researchgate.net/publication/371166744_A_Review_on_Java_Programming_Language.
6. CADENHEAD, Rogers. *Java in 21 days, sams teach yourself (covering java 8)*. 7. vyd. Indianapolis, IN: Sams Publishing, 2015.
7. MARTIN HELLER. *Choosing your Java IDE*. 2022. Dostupné tiež z: <https://www.infoworld.com/article/3114167/choosing-your-java-ide.html>.
8. *Introduction to Node.js*. 2024. Dostupné tiež z: <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>.
9. JOSSEF HARUSH KADOURI. *When ‘Everything’ Goes Wrong: NPM Dependency-Hell Campaign — 2024 Edition*. 2024. Dostupné tiež z: <https://medium.com/checkmarx-security/when-everything-goes-wrong-npm-dependency-hell-campaign-2024-edition-1652384d5633>.
10. ELENA KOSOUROVA. *What is Python Used For? 7 Real-Life Python Uses*. 2022. Dostupné tiež z: <https://www.datacamp.com/blog/what-is-python-used-for>.
11. ASTARI S. *What Is HTML? Hypertext Markup Language Basics Explained*. 2023. Dostupné tiež z: <https://www.hostinger.com/tutorials/what-is-html>.
12. AMIT SHEEN. *How CSS Perspective Works*. 2020. Dostupné tiež z: <https://css-tricks.com/how-css-perspective-works>.
13. *An Introduction to JavaScript*. 2022. Dostupné tiež z: <https://css-tricks.com/how-css-perspective-works>.

14. VICTORIA PUZHEVICH. *Pros and Cons of Using Spring Boot*. 2020. Dostupné tiež z: <https://scand.com/company/blog/pros-and-cons-of-using-spring-boot/>.
15. ANASTASIA STAVER. *Pros and Cons of Spring Boot*. 2021. Dostupné tiež z: <https://bambooagile.eu/insights/pros-and-cons-of-using-spring-boot>.
16. SIVATEJA KANDULA. *Main Advantage And Disadvantages Of Hibernates*. 2011. Dostupné tiež z: <https://www.java4s.com/hibernate/main-advantage-and-disadvantages-of-hibernates/>.
17. HITESH AGARWAL. *Advantages and Disadvantages of Using React Native*. 2023. Dostupné tiež z: <https://techexactly.com/blogs/advantages-and-disadvantages-of-using-react-native>.
18. *React Native: Advantages and disadvantages of this framework*. 2021. Dostupné tiež z: <https://rootstack.com/en/blog/react-native-advantages-and-disadvantages-framework>.
19. XAVIER SEIGNARD. *Our journey from React Native to Expo for mobile app development*. 2023. Dostupné tiež z: <https://medium.com/alan/our-journey-from-react-native-to-expo-for-mobile-app-development-at-alan-%EF%B8%8F-3b1569e8ab7c>.
20. ÖMÜR BILGILI. *Exploring Expo in React Native: A Comprehensive Guide to Cross-Platform App Development*. 2023. Dostupné tiež z: <https://omurbilgili.medium.com/exploring-expo-in-react-native-a-comprehensive-guide-to-cross-platform-app-development-45e6a3bfa111>.
21. EVI MARIA. *Measure distance locating nearest public facilities using Haversine and Euclidean Methods*. 2020. Dostupné tiež z: https://www.researchgate.net/publication/339906456_Measure_distance_locating_nearest_public_facilities_using_Haversine_and_Euclidean_Methods.
22. DAVE PETERSON. *Distances on Earth 2: The Haversine Formula*. 2021. Dostupné tiež z: <https://www.themathdoctors.org/distances-on-earth-2-the-haversine-formula/>.

Prílohy

Príloha A: Zdrojový kód celého systému

Príloha B: Používateľská príručka

Zdrojový kód celého systému sa nachádza na url: <https://github.com/tvago/DP>

Používateľská príručka sa nachádza na url: <https://drive.google.com/drive/folders/1WasMyP1ko-GNXnw7Nr0Rp2uRFK48mMiM?usp=sharing>