UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE

FAKULTA PRÍRODNÝCH VIED A INFORMATIKY

Vizualizácia dát získaných zo senzorov mobilného zariadenia

BAKALÁRSKA/DIPLOMOVÁ práca

**2023 Martin Vrták**

UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE

FAKULTA PRÍRODNÝCH VIED A INFORMATIKY

Vizualizácia dát získaných zo senzorov mobilného zariadenia

BAKALÁRSKA/DIPLOMOVÁ práca

Študijný odbor: 18. Informatika

Študijný program: Aplikovaná informatika

Školiace pracovisko: Katedra informatiky

Školiteľ: Mgr. Martin Vozár, PhD.

Nitra 2023 Martin Vrták





# Poďakovanie

Predovšetkým by som chcel poďakovať môjmu školiteľovi Mgr.   Martin  Vozár,   PhD. za dobrý prístup, komunikáciu, za konzultácie, zberom údajov a s radami.

Ďalej by som chcel poďakovať RNDr. Ján Skalka, PhD. za konzultácie a návrhy riešení niektorých problémov pri programovaní.

Ďakujem Linde Veselej za zapožičanie Android telefónu na vývoj a testovanie.

# ABSTRAKT

Vrták, Martin: Vizualizácia dát získaných zo senzorov mobilného zariadenia. [Bakalárska]. Univerzita Konštantína Filozofa v Nitre. Fakulta prírodných vied a informatiky. Mgr. Martin Vozár, PhD.. Stupeň odbornej kvalifikácie: Bakalár odboru Aplikovaná informatika. Nitra: FPVaI, 2023. .54 s.

Tato záverečná práca popisuje vývoj mobilnej aplikácie od požiadaviek, k cieľom prace, po zobrazenie a následne testovanie prejdenej trasy. Výsledkom prace je funkčná mobilná aplikácia pre Android zo zámerom ukázať užívateľovi históriu prejdenej trasy údajovo aj vizuálne pre budúcu analýzu. Dáta z aplikácie vám môžu pomôžu pri zlepšení trasy, pripadne zdokonalenia svojich schopnosti alebo možnosť ako predať svoje skúsenosti a vzdelávanie ďalším študentom ako ukážku správnej jazdy/trasy pri šoférovaní/riadení. Aplikácia navyše mimo požiadaviek, cieľov ponuka možnosť generovania súborov KML zo zozbieraných dát, možnosť slovenského alebo anglického jazyka podľa nastavení v telefóne, prípadná biela alebo tmava téma podľa preferencii v telefóne. Vygenerovane súbory KML umožňujú náhľad z hora, alebo použitie externých programov na vizualizáciu zaznamenanej trasy, kamera sa nakláňa v smere vášho pohľadu.

Kľúčové slová: Android. Mobilná aplikácia. Vizualizácia v Google. Záznam prejdenej trasy. Informatika.

# ABSTRACT

Vrták, Martin: Visualization of data obtained from mobile phonesensors. [Bachelor]. Constantine the Philosopher University in Nitra. Faculty of Natural Sciences and Informatics. Supervisor: Mgr. Martin Vozár, PhD. Degree of Qualification: Bachelor of Applied Informatics. Nitra: FNSaI, 2023. 54 p.

This final work describes the development of mobile application from requirements, to work, to display and then test the route. The result is a functional mobile application for Android from the aim of showing the user's history of the route traveled in data and visually for future analysis. The data from the application can help you improve the route, eventually improve your skills or the opportunity to sell your experience and education to other students as a demonstration of the right driving/driving route. In addition, the application outside the requirements, the goal offer the possibility of generating KML files from the data collected, the possibility of Slovak or English by the settings on the phone, a possible white or a dark topic according to the preference on the phone. The generated KML files allow you to preview from the mountain or use external programs to visualize the recorded route, the camera tilts in the direction of your view.

Keywords: Android. Mobile application. Visualization in Google. Record of traveled route. Informatics.

# Obsah

[Úvod 9](#_Toc131622216)

[1 Popis riešeného problému / Analýza súčasného stavu 10](#_Toc131622217)

[1.1. Vývojove prostredie 10](#_Toc131622218)

[Jazyk C 10](#_Toc131622219)

[Swift 11](#_Toc131622220)

[Kotlin 11](#_Toc131622221)

[Java 12](#_Toc131622222)

[1.2 Vyber vývojového prostredia 13](#_Toc131622223)

[1.3 Existujuce Aplikacie 14](#_Toc131622224)

[2 Ciele záverečnej práce 18](#_Toc131622225)

[3 Návrh a metodika 19](#_Toc131622226)

[3.1 Návrh softvéru 19](#_Toc131622227)

[Zbieranie údajov zo senzorov 19](#_Toc131622228)

[Návrh Databázy 31](#_Toc131622229)

[Návrh funkcii 32](#_Toc131622230)

[Návrh užívateľského prostredia 32](#_Toc131622231)

[3.2 Metodika vývoja softvéru 33](#_Toc131622232)

[Úvodná obrazovka 34](#_Toc131622233)

[Nastavenia 34](#_Toc131622234)

[Zbieranie dát 34](#_Toc131622235)

[Databáza 34](#_Toc131622236)

[Vizualizácia aktuálnej polohy na mape v aplikácii 34](#_Toc131622237)

[História 34](#_Toc131622238)

[Zobrazenie informácii z databázy o ceste 35](#_Toc131622239)

[4 Výsledky 36](#_Toc131622240)

[4.1 Implementácia 36](#_Toc131622241)

[Nastavenie 37](#_Toc131622242)

[Zbieranie dát 38](#_Toc131622243)

[Vizualizácia aktuálnej polohy na mape v aplikácii 42](#_Toc131622244)

[Zobrazenie informácii z databázy o ceste 42](#_Toc131622245)

[4.2 Popis vytvoreného softvéru 43](#_Toc131622246)

[4.3 Testovanie 44](#_Toc131622247)

[Záver 46](#_Toc131622248)

[Bibliography 48](#_Toc131622249)

[Zoznam príloh 50](#_Toc131622250)

# 

# Úvod

V dnešnej dobe so stále sa rozrastajúcim sa turizmom je možné využiť na prepravu lietadlo, vlak, autobus či automobil, bicykel a pod. Väčšina ľudí, ktorí cestujú ľubovoľným dopravným prostriedkom, alebo sú na túre pešo, disponujú nejakým mobilným zariadením, pomocou ktorého sa vedia navigovať. Existujú viaceré aplikácie, ktoré dokážu zaznamenať prejdenú trasu. Význam zaznamenanej trasy pomôže v návrhu lepšej trasy, nájdením nedostatkov v aktuálnej trase. V dnešnej dobe väčšina disponuje s mobilným zariadením, a preto nie je potreba kupovať na to špeciálne zariadenie. Mobilne zariadenia majú v sebe zabudované rôzne senzory a pristupovať k nim môže každý programátor, ak si od užívateľa vyžiada prístup k senzorom. Vďaka tomu že dnešné mobilne operačne systémy majú vbudovane aktualizácie vieme vďake aplikácii rýchlo spraviť novy doplnok k trasovaniu alebo urobiť opravu aplikácie v prípade problémov. Ukážeme si aké problémy treba riešiť a načo netreba zabudnúť. Prejdeme si vyžiadaním prístupu k senzorom, zberom údajov z mobilného zariadenia. Ukladaním údajov do SQL Lite aj čítanie z databázy. Zobrazením máp v aplikácii a obnovovaním polohy. Komunikáciou s Google mapami a možným exportom súborov do Google Earth. Taktiež si ukážeme ako spraviť multilingválne aplikácie podľa nastavenia telefónu.

# 1 Popis riešeného problému / Analýza súčasného stavu

Aplikáciu je možne používať na záznam prejdenej trasy, a možnosť si to neskôr zobraziť. Pozrieť si trasu kde všade som bol môže pomôcť pri budúcej analýze aby som mohol optimalizovať cestu. Možne údaje pre nových pilotov ktorý sa môžu poučiť zo svojich chyb. Učiteľ môže použiť ako príklad nesprávneho a správneho letu. Treba si ale dát pozor lebo ľahko sa budú dát tieto informácie zneužiť a mohli by tajne aj sledovať užívateľov.

## Vývojove prostredie

Na svete sú najviac rozšírene operačne systémy iOS a Android. Existujú aj ďalšie operačne systémy ale nie sú rozšírene. Operačný systém iOS patri firme Apple. Android vyvíja firma Google. Vývojové prostredia pre operačné systémy iOS a Android sú najviac využívané:

* Xcode(Apple),
* Android Studio.

Xcode vývoje prostredie potrebujete platformu od Apple (v dobe písanie tejto prace) Mac, IPad. V tomto prípade nás Xcode obmedzuje iba na platformu od Apple. Pre Xcode je potrebne poznať jeden z týchto programovacích jazykov:

* Objective C,
* Swift code.

Pre vývojove prostredie Android Studio mame opäť ma vyber programovacie jazyky. Naproti tomu Android Studio môžeme nainštalovať na rôzne operačne systémy ako Windows, Mac, Linux. Android Studio nám ponuka programovacie jazyky:

* Java,
* Kotlin.

### Jazyk C

Jazyk C nie je založený na programovaní OOP pre vysvetlenie to znamená že nemôžeme vytvárať objekty(triedy). Jazyk C je to štruktúrovane programovanie to znamená že kód organizujeme do blokov a funkcii.

### Swift

Apple vyvinul jazyk Swift pre svoje prostredie. Swift nám ponuka možnosť Objektovo orientovane programovanie (OOP). OOP je programovacia paradigma, ktorá organizuje kód pomocou objektov a ich interakcii so zameraním na zapuzdrenie, dedičnosť, polymorfizmus a abstrakciu s cieľom vytvoriť modulárny, udržiavaný a opakovane používateľný softvér.

### Kotlin

Kotlin sa zameriava na bezpečnosť (náhodných chyb v kóde). Ponuka vylepšenú typovú kontrolu hodnoty null, čo pomáha minimalizovať nevhodné stavy v kóde. Kotlin ma jednoduchšiu syntax. Kotlin 1.0 bol vydaný začiatkom roka 2016 spoločnosťou JetBrains.

Kotlin 1.0 bol vydaný začiatkom roka 2016 spoločnosťou JetBrains. Kotlin je staticky typovaný programovací jazyk. Pri kompilácií je preložený na Java bytecode, ktorý beží na JVM (Java Virtual Machine). Kotlin nie je taký populárny ani rozšírený ako Java, ale existuje niekoľko známych spoločností, ktoré tento jazyk používajú: Pinterest, Trello, Kickstarter, GoodRequest...

Kotlin bol od začiatku navrhnutý tak, aby bol kompatibilný s Javou. Toto dokonale uľahčuje vývojárom nie len migrovať projekty na Kotlin, ale aj používať knižnice naprogramované v Jave. Jazyk je navrhnutý tak, aby sme dokázali spraviť viac za menej času.

V Kotline máme voľbu, že či premennú definujeme ako Nullable (v Kotline sa to nazýva Optional) val name: String? Tu s istotou vieme povedať, že premenná môže obsahovať null. Ak by sme definovali bez otáznika, vieme s istotou povedať, že premenná nikdy nemôže mať null.

Môžeme povedať, že Java a Kotlin majú rovnakú rýchlosť písanie kódu. Kotlin má viac lakonických konštrukcií, ktoré umožňujú Android developerovi menej písať. Ale má to jeden háčik, hľadanie riešenia úlohy v Kotline potrebuje viac času ako na Jave. To znamená, že Kotlin má väčšiu kognitívnu záťaž ako Java. Ak ste fantastický abstraktný mysliteľ, Kotlin je vaša voľba. (Paronai, 2021).

Výhod programovacieho jazyka Java je hneď niekoľko, no tak ako pri každom inom programovacom jazyku, i tu sa nájdu určité negatíva. Java predstavuje viacúčelový, objektovo orientovaný programovací jazyk, ktorý založila v roku 1991 spoločnosť Oracle. Je rýchly, bezpečný a spoľahlivý a patrí dlhodobo medzi najpoužívanejšie programovacie jazyky, čo vzhľadom na benefity Java programovania nie je žiadnym prekvapením.

### Java

Java jazyk je bezplatný programovací jazyk, ktorý si môžeš stiahnuť a používať zadarmo, pričom návod na to, ako začať programovať v Jave nájdeš dnes takmer všade.

Javu možno definovať ako univerzálny programovací jazyk. To znamená, že ju je možné použiť na programovanie skoro v každej oblasti. Najčastejšie sa využíva na vývoj softvéru, Big Data, serverových back-endov, mobilných, desktopových alebo webových Java aplikácií, no skvelo poslúži napríklad aj v oblasti vývoja umelej inteligencie či strojového učenia.

Zároveň ide o interpretovaný jazyk, ktorý sa ľahko prenáša medzi jednotlivými platformami. Zdrojový kód napísaný v Jave tak možno spustiť všade bez ohľadu na danú platformu či operačný systém.

Programovanie v Java je nielen zaujímavé, ale aj pomerne jednoduché. Programovací jazyk Java je zrozumiteľný, ľahko čitateľný a jeho syntax vychádza z obľúbených jazykov C a C++. Pred spustením dokonca vykoná analýzu s cieľom overiť, či sa v zdrojovom kóde nenachádzajú chyby, problémy alebo nedostatky, ktoré by mohli znemožniť funkčnosť či ohroziť bezpečnosť aplikácie.

Skvelou správou je, že Java spravuje svoju pamäť automaticky bez nutnosti manuálneho spúšťania či cudzieho zásahu. Ty sa tak môžeš nerušene venovať programovaniu a nezaťažovať sa potenciálnymi problémami súvisiacimi so správou pamäte.

Okrem bohatej odbornej literatúry a online návodov ti pri programovaní v Java môžu pomôcť aj komunity pre developerov. Na svete existuje vyše 9 miliónov programátorov, ktorí sa s rovnakým, prípadne podobným problémom ako ty už stretli alebo si s ním nevedia poradiť. A keďže Java predstavuje open-source programovací jazyk, tak nájsť pomoc je skutočné jednoduché.

Programovací jazyk Java možno spustiť na rôznych platformách i operačných systémoch. Táto skutočnosť však negatívne vplýva na jeho rýchlosť. Java je v dôsledku toho pomalšia ako jazyky cielené na konkrétnu platformu.

Na rýchlosť jazyka má negatívny dopad tiež automatická správa pamäte. Hoci programátorovi uľahčuje prácu, výrazne znižuje rýchlosť Javy v porovnaní s programovacími jazykmi, pri ktorých má správu pamäte na starosti programátor. Pre programovanie hier alebo operácií náročných na výpočet preto programovací jazyk Java nie je úplne ideálnou voľbou.

Okrem toho je náročný aj na RAM pamäť a neodporúča sa ani v prípade tvorby komplikovaného používateľského rozhrania, aby sa predišlo zbytočným problémom a nezhodám, ku ktorým prichádza pri používaní Javy. Veľkú nevýhodu programovania v Jave predstavuje i absencia zálohovania. Programovací jazyk Java sa na zálohovanie neorientuje, v dôsledku čoho hrozí strata dôležitých dát. (Msg LIfe).

## 1.2 Vyber vývojového prostredia

Z informácii ktoré som uviedol vyššie, som sa rozhodol použiť ako vývojove prostredie Android Studio.



Zdroj Graf . Najrozšírenejšie programovacie jazyky a ich preferovanie od 2014 - 2022[[1]](#footnote-1)

Dôvody pre Android Studio:

* Android Studio môžem nainštalovať na Windows / Linux / Mac,
* Android Studio som sa naučil na mojej alma mater,
* Android Studio ma medzi primárnymi programovacími jazykmi Javu,
* literatúra pre Android Studio (Lacko, 2017).

Ako programovací jazyk bol zvolený Java:

* Java patrí medzi najpreferovanejšie jazyky (viď. Graf 1) a tak mi ponuka bohatú základnú pri riešení problémov na fórach,
* kód v jave sa da použiť na akomkoľvek hardvéri,
* Javu vieme migrovať na Kotlin,
* programovanie v jave som si osvojil v mojej alma mater,
* z ponúkaných jazykov mam s Javou najviac skúsenosti,
* rozsiahle znalosti z dvoch odborných kníh o Jave, ktoré mi pomohli pri práci s týmto programovacím jazykom. (Schildt, 2012; Roubalová, 2015).

## 1.3 Existujuce Aplikacie

V tejto časti sa zameriame na typ súboru KML. Keyhole Markup Language (KML) je jazyk založený na XML na správu zobrazenia 3D geopriestorových údajov. KML je štandard udržiavaný organizáciou Open Geospatial Consoritum (OGC). (Mapserver).

OGC je konzorcium odborníkov, ktorí sa zaviazali zlepšovať prístup ku geopriestorovým alebo lokalizačným informáciám. Spájame ľudí, komunity a technológie, aby sme riešili globálne výzvy a riešili každodenné potreby. (OGC, 1994).

Súbor KML môžete vytvoriť pre importovanie aj do Google Earth, kde môže obsahovať rôzne informácie. Informácie ktoré môžete vložiť do KML súboru sú zatriedene pomocou tagov. Príklad vygenerovaného súboru KML:

<?xml version="1.0" encoding="utf-8"?>

<kml xmlns:gx="http://www.google.com/kml/ext/2.2" creator="WIW" xmlns="http://www.opengis.net/kml/2.2">

<Document>

<name>2023-03-06 16:48:06</name>

<gx:Tour>

<name>Track Log Fly-Through</name>

<gx:Playlist>

<gx:FlyTo>

<gx:duration>4.2763481492</gx:duration>

<gx:flyToMode>smooth</gx:flyToMode>

<Camera>

<longitude>18.09129627</longitude>

<latitude>48.30826692</latitude>

<altitude>41.96394348144534</altitude>

<heading>241.63214</heading>

<tilt>80</tilt>

<range>10</range>

<altitudeMode>absolute</altitudeMode>

</Camera>

</gx:FlyTo>

</gx:Playlist>

</gx:Tour>

</Document>

</kml>

Ako vidieť na príklade tak je podobný XML. XML umožňuje vytvoriť ľubovoľnú značku potrebnú na popis a štruktúrovanie údajov. (Microsoft)

SkyDemon je aplikácia na leteckú navigáciu a neponúka možnosť iných spôsobov transportu. SkyDemon ponúka import z CSV súboru alebo GPX (GPS Exchange Format). KML súbor vieme konvertovať do týchto formátov. Aplikácia tiež zobrazuje nepriaznivé počasie na mape. SkyDemon ma vlastne mapy. SkyDemon je spoplatnená aplikácia na baze mesačného / ročného predplatného. (Divelements Limited, 2009).



Obrázok Aplikácia SkyDemon[[2]](#footnote-2)

Strava je aplikácia ktorá je zameraná na športovanie. Ponuka možnosť zaznamenávať športy behanie, bicyklovanie a plávanie chyba tu možnosť zaznamenanie lietania výšky náklonu. Strava neponúka možnosť importu KML súboru. (Strava, Inc., 2009).



Obrázok Aplikácia Strava[[3]](#footnote-3)

Hiking Project je zameraná hlavná na turizmus. Disponuje vlastnými mapami. Ponuka plánovanie výletu. Recenzie oblasti na turizmus. Ponuka možnosť stiahnuť si chodníky v oblasti. Cez kameru ponuka zobrazenie výšky kopcov alebo zobrazenie hviezdnych súhvezdí. Chyba mu generovanie KML súboru. Nedisponuje s možnosťou zaznamenávať preletenú trasu. (Hikingproject, 2014).



Obrázok Zobrazenie výšky kopcov Hiking Project[[4]](#footnote-4)

Obrázok Mapa v Hiking Project[[5]](#footnote-5)

# 2 Ciele záverečnej práce

Cieľom prace bude zaznamenávanie prejdených trás na turistike, cestovania(zahŕňa bicykel, kolobežku, auto, autobus a iné dopravne prostriedky) a lietanie lietadlom. Zameriame na implementáciu vizualizácie získaných GPS súradníc spolu s údajmi z ďalších senzorov, ako sú napríklad gyroskop, akcelerometer, barometer atď., čo umožní komplexné a intuitívne zobrazenie týchto multimodálnych dát.

Podciele:

* predstaviť návrh užívateľského rozhrania v aplikácii,
* rozvrhnúť rozloženie v sekciách aby bola jednoduchá na používanie,
* navrhnúť databázu pre záznamy zo senzorov,
* zaznamenávať údaje z mobilného zariadenia (gyroskop, GPS atď.),
* ukladanie údajov do databázy,
* umožniť upravovanie niektorých záznamov,
* mazanie bodov z databázy, celých tras, celu databázu,
* vizualizovať aktuálnu polohu a smer pohybu,
* zobraziť údaje z trasy a body k danej trase,
* ukážka prejdenej trasy na mape,
* naprogramovať vygenerovanie súbor pre vizualizáciu prejdenej trasy na rôznych systémoch,
* vyvinúť možnosť vytvorenia súboru na animáciu z dát na iných platformách.

# 3 Návrh a metodika

Návrhom riešenia je naprogramovať funkčnú aplikáciu. Rozdelíme si problémy do viacej kategórii a riešime ich postupne. Ako prvé si zistime informácie o senzoroch v mobilnom zariadení, bez toho ako fungujú a aké nám poskytujú informácie nemôžeme spraviť dobrý a funkčný návrh a tak sa predstavíme senzory, ktoré mi poskytujú aké údaje reprezentujú a vybral som tie ktoré mohli aj zahŕňajú ciel prace. Keď už vieme s akými senzormi budeme pracovať navrhneme databázu na ukladanie informácii zo senzorov. Po uložení dát už mame dostatok informácii na návrh užívateľského riešenia.

## 3.1 Návrh softvéru

Zozbieranie údajov zo senzorov. Podľa dát sme navrhli databázu pre informácie zo senzorov. Pokračovali sme návrhom užívateľského rozhrania. Posledným krokom ostalo navrhnutie vizualizácie na základe získaných dát.

### Zbieranie údajov zo senzorov

Druhy senzorov v Android Studio ktoré sme použili:

* Sensor.TYPE\_ACCELEROMETER,
* Sensor.TYPE\_LINEAR\_ACCELERATION,
* Sensor.TYPE\_GRAVITY,
* Sensor.TYPE\_GYROSCOPE,
* Sensor.TYPE\_ORIENTATION,
* Sensor.TYPE\_ROTATION\_VECTOR, (Google LLC).
* GPS.



Obrázok Hardvér v smartphonoch.[[6]](#footnote-6)

Akcelerometer nám ukazuje zrýchľovanie čim viac zvyšujeme rýchlosť tým sa číslo zväčšuje. Akcelerometer ma tri súradnice. Každá súradnica ukazuje smer zrýchľovania, pri ťahaní zariadenia hore nám na z osi ukáže zrýchlenie. Hodnota údajov sa vyjadruje v m/s².

Senzor môžeme použiť napríklad na:

* ovládanie hier,
* vyhodnocovanie krokov,
* detekciu pádu,
* monitorovanie pohybu.



Obrázok Akcelerometer v mobile.[[7]](#footnote-7)

Linear acceleration nám ukazuje zrýchlenie zariadenia, ktorý je bez gravitačnej sily. Zrýchlenie, ktoré meria senzor, sa nezohľadňuje gravitačná príťažlivosť Zeme a hodnoty sa získavajú iba z pohybu zariadenia. Údaje zo senzora sú opäť v troch osiach.

Možne využitie:

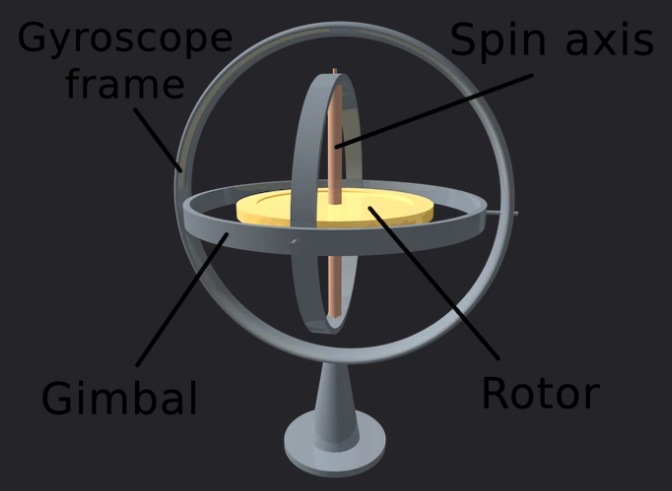
* pri hrach,
* na detekciu pohybu,
* alebo pri Virtuálnej Realite (VR)

Gravity sa používa na získanie hodnôt gravitačného zrýchlenia zariadenia. Tento senzor je softvérový to znamená že získava údaje z akcelerometra a gyroskopu. Gravity ma tri hodnoty.

Gyroskop je zariadenie, ktoré meria polohu smartphonu v priestore pomocou špeciálnych senzorov. Prvok sa skladá z mikroskopických dosiek, ktoré vibrujú pri zmene tlaku. Zmeny polohy zariadenia aktivujú senzory, ktoré prenášajú informácie do aplikácie a spúšťajú zodpovedajúci efekt.

Gyroskop vykonáva v telefóne niekoľko dôležitých funkcií, ktoré používame každý deň, bez toho aby sme si to uvedomovali. Vďaka senzoru je mimo iného možné automaticky meniť orientáciu fotografií v galérii, ktorá sa odohráva v reakcii na pohyb smartphonu. Gyroskopy sú tiež používané vývojári hier a aplikácií, pretože mnoho programov pracuje na základe orientácie zariadenia. Vďaka popularizácii technológie VR našiel gyroskop tiež uplatnenie vo virtuálnej realite. (Jarko, 2019).

Gyroskop je akýsi typ detskej hračky, ktorá funguje podobne ako zotrvačník. Napriek tomu že má len tenkú nohu, na ktorej stojí, je prakticky nemožné ho prevrátiť. Gyroskop disponuje na rozdiel od klasického zotrvačníka ešte aj dvoma prstencami (jeden je vertikálny a druhý horizontálny). Zotrvačník sa v prípade gyroskopu však nazýva rotor. Gyroskop využíva jav známy aj ako gyroskopická akcelerácia, ktorá zaručuje jeho stabilitu v akejkoľvek polohe. Práve tomuto javu vďačíme tiež aj za to, že môžeme pomocou zmeny polohy zotrvačníka vzhľadom ku gyroskopu kalkulovať zmenu polohy celého zariadenia v 3 osách. Takýto gyroskop označujeme tiež pojmom 3-AXIS. (Hecl, 2014).

[[8]](#footnote-8)

Obrázok Gyroskope. Takýto gyroskope sa nepoužíva v mobilných zariadeniach.

Na snímanie polohy rotora sa nepoužíva sledovanie polohy samotného gyroskopu, čo by bolo príliš zložité a technicky takmer nemožné, no namiesto toho sa sleduje zmena kapacity kondenzátorov, ktoré plnohodnotne zastupujú optické snímače ako je to pri klasických veľkých gyroskopoch. Tieto kondenzátory sa nachádzajú na obale gyroskopu a je ich tam niekoľko stoviek. Metóda snímania pohybu rotora sa môže líšiť v závislosti od výrobcu systému, ktorý sa generálne nazýva MEMS. (Hecl, 2014).



Obrázok Gyroskop v smartphone.[[9]](#footnote-9)

Orientation poskytuje údaje o azimute, náklone a valcovaní zariadenia v stupňoch. Udaj x nám poskytuje azimute čiže kompas od 0 po 360 . Druha hodnota nám ukazuje náklon dopredu dozadu od -180 po 180. Posledná informácia nám ukazuje náklon do ľava a do prava od -90 po 90.



Obrázok Ukážka Smeru Azimut.[[10]](#footnote-10)

Rotation Vector je konštanta popisujúca typ snímača vektora otáčania. Obsahuje 4 hodnoty x, y, z, w. W predstavuje hodnotu skalárna zložka vektora rotácie ((cos(θ/2)). Skalárna zložka je voliteľná hodnota. (Google LLC)

GPS (Globálny Pozičný Systém). Problémom je, že skratka označuje iba americký globálny pozičný systém, ktorý vstúpil do prevádzky ako prvý a použil takýto všeobecný názov. Keďže v súčasnosti existuje už niekoľko ďalších systémov, ako napríklad GLONASS a Galileo, pričom majú odlišné a dokonca aj lepšie vlastnosti, je vhodné mať nejaké spoločné označenie, ktoré by pokrylo všetky systémy.

Správnejšia a v odborných kruhoch a technických špecifikáciách dnes už zaužívaná skratka je GNSS, teda globálne navigačné satelitné systémy (alebo kratšie GNS, teda globálne navigačné systémy). Tejto skratke je dnes vhodné dávať prednosť, pretože moderné smartfóny obvykle používajú viacero navigačných systémov naraz a už len zriedkavo iba samotnú GPS.

Všetky navigačné systémy sa v základe skladajú z troch základných zložiek, ktorými sú kozmická, riadiaco/kontrolná a používateľská časť. Tú používateľskú všetci dobre poznáme, pretože ju tvoria naše smartfóny, tablety, či akékoľvek iné zariadenia, ktoré vďaka interným anténam a zabudovaným modulom dokážu signál takýchto systémov prijať. Kozmickú časť tvoria satelity na obežnej dráhe Zeme, ktoré signálovú sieť vytvárajú.

Keďže každý navigačný systém má vlastné satelity a používa rozličné frekvencie a takisto hardvér, má oproti ostatným aj rozdielne vlastnosti. Tretia a posledná časť je tvorená riadiaco/kontrolnými strediskami, teda pozemnými stanicami rozmiestnenými po celom svete, ktoré konkrétnu sústavu GNSS kontrolujú a udržujú, pričom ide napríklad o korekcie trajektórií družíc a overovanie ich správnej funkčnosti.

V súčasnosti existujú globálne a regionálne navigačné systémy. Americký GPS ruský GLONASS (Globalnaya Navigatsionnaya Sputnikovaya Sistema), Europsky Galileo, Čínsky BeiDou. Niektoré krajiny prevádzkujú aj regionálne navigačné systémy, ktoré poskytujú navigáciu len nad konkrétnou krajinou a jej okolím. Takýmito systémami sú indický NAVIC a japonský QZSS.

Zvyčajne je na úplné pokrytie potrebných 24 satelitov v rovnomerných rozstupoch, avšak GNSS operujúca v plnej prevádzke ich má obvykle o niečo viac (dohromady cca 30), pričom zostávajúce slúžia ako náhradné. Na ktoromkoľvek satelite sa totiž môže kedykoľvek vyskytnúť porucha, čo ho vyradí z prevádzky, pričom náhradný kus môže v krátkej dobe zaujať jeho miesto bez toho, aby bolo nutné čakať mnohé dni či mesiace na nový štart satelitu zo Zeme.

Satelity GNSS, či už ide o GPS, GLONASS alebo Galileo, sú komplexným kusom technológie. V základe v ich útrobách nájdeme stabilizátory, prijímacie a vysielacie antény na rôznych frekvenciách, slúžiace nielen na vysielanie samotného navigačného signálu, ale aj na komunikáciu s pozemnými stanovišťami a pre kontakt so sesterskými družicami.

Družice sú totiž navrhnuté tak, aby dokázali niekoľko týždňov fungovať aj plne autonómne a spolupracovať s ostatnými družicami aj bez riadenia a kontaktu s pozemným riadiacim strediskom (pre prípad jeho zničenia pri vojenskom konflikte). Z hľadiska celkového princípu fungovania je najkľúčovejšou výbavou samotných družíc trojica alebo štvorica (z dôvodu zálohy) mimoriadne presných atómových hodín.

Satelity obvykle obiehajú Zem vo výške zhruba 20 000 kilometrov (detaily v tabuľke) a ich poloha sa neprestajne mení. Nad vašou hlavou sú tak v priebehu dňa vždy iné satelity, pričom ich rotačná doba je obvykle 12 až 14 hodín. Zem neobiehajú v náhodných smeroch. Obvykle sú sústredené do šiestich kružníc, ktoré sú od seba odchýlené o 60° (teda obvykle 5 satelitov na kružnicu). Nad každým miestom na Zemi je tak v jednom okamihu v dohľade 6 až 12 družíc jedného systému GNSS.

Životnosť satelitov je obmedzená, a preto je nutné vypúšťať na obežnú dráhu stále nové kusy. To umožňuje aj postupnú modernizáciu systému. Tieto satelity boli konštruované na životnosť 5 až 8 rokov, ale mnohé ju prekročili.



Obrázok Tabuľka GNS.[[11]](#footnote-11)

Ak chceme navigačné služby používať, potrebujeme k tomu príslušný hardvér. V základe sú potrebné dve veci. Anténa, ktorá dokáže signál satelitu prijať a GNSS modul, ktorý dokáže signál rozkódovať a spracovať.

V smartfónoch, tabletoch a takisto napríklad v automobilových navigáciách sa pod slovom modul myslí skrátka samostatný čip, ktorý sa nachádza na základnej doske. Jeho útroby však môžu byť integrované aj do iného čipu, kde zdieľa priestor spolu s modulmi na dekódovanie mobilného signálu 3G, LTE, 5G a podobne. Rôzne moduly majú rôznu kvalitu i cenu. Drahšie smartfóny môžu mať tak osadený lepší čip ako lacnejšie modely, podporujúci viacero GNSS a ich frekvencií naraz. Tieto rozdiely sa ale čoraz viac stierajú, pretože ceny týchto modulov sú na úrovni desiatok centov či jednotiek eur.

Čip má tvar štvorčeka či obdĺžnika s rozmerom 3 až 5 mm. Obvyklými výrobcami sú americké firmy Broadcom alebo Qualcomm. Modul sa, pravdaže, nedá používateľsky vymeniť. Ak smartfón rozoberiete, čip síce na doske nájdete, ale podobne ako ostatné je pripojený pomocou BGA (Ball grid array), čo znamená, že jeho kontakty majú tvar malých roztopených guľôčok, pomocou čoho je spájkovaný s kontaktmi na doske.

Z hľadiska antény je situácia pomerne jednoduchá. Signál jednotlivých GNSS, teda GPS, GLONASS, Galileo a BeiDou sa mierne odlišuje, ale vo všetkých prípadoch sa nachádza v mikrovlnnom frekvenčnom rozsahu. V ňom sa nachádza aj signál Wi-Fi a Bluetooth, takže smartfóny môžu signál satelitov prijímať totožnou anténou, bez potreby dodatočnej.

Satelity GNSS o prijímačoch, ktoré ich používajú, „nevedia“ a ani ich polohu nijako nepočítajú. Pri pripojení do internetu alebo pri telefonovaní totiž s telefónom komunikuje nielen veža mobilného signálu, alebo Wi-Fi router, ale aj smartfón s nimi komunikuje späť. Satelity nevedia, aké a koľko zariadení ich signál momentálne prijíma, podobne ako rozhlasová stanica nevie, koľko ľudí si ju v aute naladí na svojom rádiu.

Satelity fungujú ako hodiny, z ktorých čas čítame nie pozeraním, ale prijímaním mikrovlnného signálu. Keďže satelitov je na oblohe viac, smartfón dostáva časový údaj viacnásobne, pričom sa mierne líši. A táto odlišnosť je mimoriadne dôležitá.

Úlohou smartfónu alebo akéhokoľvek iného prijímača na Zemi je prijať časový signál od štyroch alebo viacerých družíc GNSS v jeden moment. Zo zistených rozdielov v čase sám odvodí svoju vzdialenosť od jednotlivých satelitov a na základe toho sám vypočíta svoju pozíciu na povrchu Zeme. K tomu sa používa geometrický princíp známy ako trilaterácia. Princíp sa dá jednoducho predstaviť v dvojrozmernej podobe. Predstavte si, že stojíte niekde na Slovensku a netušíte kde. Vytiahnete pritom z vrecka prístroj, ktorý by dokázal odmerať vzdialenosť od blízkych miest. Prístroj vám ako prvé zahlási – ste 60 km od Banskej Bystrice. To je dôležitá informácia, avšak problémom je, že môžete byť od tohto mesta ktorýmkoľvek smerom, teda napríklad na východ, západ, juh, či sever či niečo medzi tým.

Dorazí však druhá informácia – ste zároveň 62 km od Trenčína. Vaša poloha sa razom značne spresní. Dva okruhy sa totiž pretnú (pozri obrázok). Problémom je, že sa pretnú v dvoch miestach, čo znamená, že ste buď niekde na severe pri Žiline alebo na juhu pri Zlatých Moravciach. Tretí údaj všetko rozhodne – ste aj 65 km od Liptovského Mikuláša. Vašou pozíciou je teda Žilina. Rovnaký princíp je použitý aj pri GNSS.

Vďaka tomu, že signál vysielajú družice, ktoré sú nad zemou, však trilaterácia nepracuje v 2D, ale 3D priestore. Namiesto kruhovej vzdialenosti od vysielača tak nestojíte na hrane kruhu, ale gule. Vzhľadom na to, že máme o jeden rozmer navyše, potrebujeme na lokalizáciu aj štvrtý zdroj. Dve gule, respektíve sféry, sa totiž pretnú v jednom kruhu, tri v dvoch bodoch a až štyri sféry v jedinom. V núdzovom prípade môže GNSS prijímač počítať za štvrtú sféru samotnú Zem. S použitím štvrtého satelitu sú však dáta presnejšie.



Obrázok Určovanie polohy v 3D priestore.[[12]](#footnote-12)

Ak teda smartfón pozná svoju vzdialenosť od 4 či viacerých satelitov, ktorých poloha je známa (vysielajú ju), vie teda, kde na Zemi sa nachádza. Ako ale tú vzdialenosť smartfón odmeria len podľa toho, aký čas ukazujú hodiny satelitu? Je to vďaka tomu, že chvíľu trvá, kým k nemu dorazí elektromagnetická vlna vysielaného mikrovlnného signálu. Je to podobné, ako oneskorenie hromu za bleskom.

Ak pri búrke zbadáte záblesk, môžete začať počítať. Ak napočítate napríklad do 9 a hrom sa ozve, blesk bol od vás 3 km ďaleko. Zvuk sa totiž v našej atmosfére šíri rýchlosťou zhruba 333 metrov za sekundu (1200 km/h), zatiaľ čo svetlo zhruba o miliardu krát rýchlejšie (300 000 km/sekundu). Keďže viete, že záblesk a zvuk vznikli v rovnakú dobu, vďaka ich rozdielnej rýchlosti a oneskorenia zistíte ako ďaleko od zdroja stojíte.

Podobne je to aj so satelitmi GNSS, ktoré majú navzájom zosynchronizovaný čas svojich hodín a neprestajne ho všetky vysielajú. Váš smartfón vie, že je práve čas 14 hodín, 25 minút a 15 sekúnd. Zároveň ale prijíma signál od satelitov, pričom Satelit 1 hovorí, že je 14:25:15, ale práve prijatý signál zo Satelitu 2 dáva na známosť, že je 14:25:14. Keďže vieme, že oba satelity vysielajú rovnaký čas v rovnaký moment, oneskorenie signálu druhého je z dôvodu, že je od nás ďalej. A rozdiel v čase nám povie koľko, podobne ako pri prípade hromu a blesku.

Problémom je, že signál, ktorým satelity vysielajú, je elektromagnetickým žiarením, čo znamená, že sa šíri rýchlosťou svetla. A jeho rýchlosť je masívna. Rozdiel v jednej sekunde znamená vzdialenosť 300 miliónov metrov, čo by stačilo na sedemnásobné obtočenie okolo Zeme.

Keďže my chceme merať našu vzdialenosť na jednotky metrov presne, potrebujeme presnejšie merať aj čas. Ak by sme ho merali tisíckrát presnejšie na milisekundy, stačilo by to na presnosť 300 000 metrov. Ak by sme ho merali miliónkrát presnejšie na mikrosekundy, stačilo by nám to na rozdiely väčšie ako 300 metrov. Očividne teda potrebuje merať čas na nanosekundy, čo nám poskytne vzdialenosti na metre a prípadne aj centimetre.

Na moderných hodinách a elektronike je čas odvodzovaný na základe elektrických pulzov vyvolaných kmitaním kryštálu kremeňa pod elektrickým napätím. Pravidelnosť pulzov umožňuje, že hodiny dokážu udržať veľkú presnosť, pričom u kvalitnejších kusov môže ísť napríklad len o rozdiel 15 až 30 sekúnd za mesiac, teda nechcené oneskorenie či zrýchlenie 0,5 až 1 sekundu za deň.

Potrebujeme presnosť na nanosekundy. Jediné oscilátory, ktoré takéto niečo umožňujú, sú tie založené na atómových rezonanciách. A práve takéto hodiny majú na svojej palube aj satelity GNSS. Obvykle sú založené na osciláciách v atómoch cézia alebo rubídia a oneskorujú sa za deň len o rozsah okolo niekoľkých nanosekúnd, či inak povedané, oneskoria sa o 1 sekundu až za viac ako miliónov rokov.

Možno ste sa niekedy zarazili nad tým, že nejaká prenosná alebo integrovaná auto navigácia, alebo napríklad aj tablet, zisťuje pri prvotnom zapnutí vašu polohu značne pomalšie, ako váš smartfón. Ak stojíte na mieste, rozdiel môže byť len pár sekúnd, ale pri pohybe rýchlosťou 50 či 100 km/h môže ísť aj o niekoľko desiatok sekúnd, či pokojne aj viac ako minútu. Tento rozdiel sa pri tom prejavuje len pri úvodnom zapnutí zisťovania polohy. Akonáhle zariadenie už polohu zistí, následne už ju aktualizuje pri vašom pohybe rýchlo.

Je to dôsledok mechanizmu, ktorý sa označuje ako A-GPS, teda Asistovaná GPS. Jej princípom je, že pri zisťovaní polohy asistuje smartfónu vysielač mobilnej siete, ku ktorému je v rámci svojej SIM karty pripojený. V základe ide o to, že server mobilného operátora do vášho telefónu pošle už vypočítané navigačné dáta, v súvislosti s polohou pozemnej vysielacej veže a zároveň aj informácie o satelitoch, ktoré sú práve nad vami. Aj keď sa telefón z nich nedozvie, kde presne je, približná poloha je mimoriadne cenná. Prvotné zistenie polohy je totiž značne náročné na výpočet aj na čas.

Zariadenie nevie, kde na Zemi je a keďže môže byť všade, musí vypočítať svoju polohu „od nuly“, čo znamená počkanie na kompletné informačné dáta vysielané satelitmi rýchlosťou 50 bit/s. To chvíľu trvá, pričom ak pritom zájdete za prekážku a signál sa stratí, zariadenie musí začať znova od nuly. Čím viac signálov rozdielnych satelitov prijíma, tým lepšie, avšak ak ste napríklad v meste a signál sa odráža od budov, alebo v lese od stromov, čas ktorý je potrebný na vyriešenie všetkých chýb a konfliktov narastá.

Výpočet tak môže trvať niekoľko desiatok sekúnd a prinajhoršom aj zopár minút. Aby sme zariadeniam činnosť uľahčili, veža mobilnej siete telefónu odošle približnú polohu, od ktorej sa môže odraziť, spoločne so stiahnutými dátami aktuálnych satelitov nad ňou. Je to ako keby ste sa otázkami snažili prísť na to, na aké číslo od 1 do 1000 niekto myslí, pričom kamarát vám pomôže tým, že je medzi 720 a 730. Svoje hádanie následne už masívne urýchlite.

A práve vďaka tomu je na smartfónoch úvodné zistenie polohy také bleskové, behom jednej či dvoch sekúnd. Tento mechanizmus môže, ale aj nemusí byť vedený ako dátový (internetový) prenos. Závisí od operátora. Avšak tak či onak, na vašom účte a dátovom limite sa to nijako neprejaví, pretože ide o textové dáta, ktoré sú skutočne miniatúrne.

Ak používate na navigovanie tablet bez SIM karty, alebo navigáciu v aute, tie tento asistenčný luxus nemajú a úvodné zistenie polohy im preto trvá o niečo dlhšie. To isté by nastalo aj v prípade, že SIM kartu z vášho smartfónu vyberiete, alebo sa nachádzate mimo pokrytia mobilnej siete. Smartfón je plne schopný vypočítať polohu sám zo signálu satelitov. Len je to na úkor času a takisto z dôvodu náročnosti výkonu aj na úkor výdrže na batériu.

GNSS vám pri navigovaní v aute ukazuje trochu inú rýchlosť, ako váš tachometer. Čo je presnejšie? Možno vás to prekvapí, ale aj napriek tomu, že napríklad GPS má v praxi presnosť len okolo 3 až 5 m, tak z hľadiska rýchlosti je počítanie mimoriadne presné. Zmena vašej polohy vzhľadom na čas k jednotlivým družiciam je v prípade GPS presná až na 0,006 m/s v rámci trojsekundového intervalu s 95 % pravdepodobnosťou.

WAAS (Wide Area Augmentation System), skladajúci sa z mnohých pozemných staníc a takisto podporných satelitov (obvykle ide o satelity určené prioritne na iný účel, ktoré ale asistujú GPS ako doplnok k svojej štandardnej činnosti). Tie vysielajú korelačné navigačné údaje, ktoré v kombinácii s časovým signálom satelitov navigovanie pozemných zariadení spresňujú.

Obrovský pokrok v rámci presnosti prináša európsky Galileo, ktorého rubídiové atómové hodiny sú schopné presnosti na 1,8 nanosekundy za 12 hodín, čo umožňuje dosiahnuť presnosť na 1 m. Vysielanie na základe týchto hodín je dostupné všetkým, čo znamená, že používaním systému Galileo vo svojom smartfóne si z hľadiska presnosti výrazne polepšíte (hlavne od budúceho roku, keď bude v plnej prevádzke). Moduly s podporou tohto systému má už viac ako 30 modelov smartfónov na súčasnom trhu a ak máte relatívne nový telefón, môžete sa o jeho podpore presvedčiť napríklad pomocou aplikácie GPSTest (barbeau), ktorá je zdarma a bez reklám.

Zaujímavosťou je, že Galileo obsahuje ešte aj presnejší mechanizmus, založený na väčších a komplexnejších atómových hodinách s vodíkovým maserom. Maser je v základe podobné zariadenie, ako laser, avšak s tým rozdielom, že namiesto lúča viditeľného svetla zostruje svetlo na inej, konkrétne mikrovlnnej frekvencii. Ide o komplexný stroj, ktorého cena sa šplhá až na 200 000 eur. V rámci takýchto atómových hodín sa používa frekvencia prechodu vodíkového atómu, ktorá tiká na extrémne stabilnej frekvencii 1,4 GHz.

Vo výsledku je obrovsky pozitívnym faktom, že moderné smartfóny dnes môžu používať až štyri nezávislé GNSS, vďaka čomu majú v jednom momente prístup k viac ako 40 satelitom z obrovského počtu smerov (Urban, 2018).

### Návrh Databázy

Pre návrh databázy som sa rozhodol pre dve tabuľky. Prvá tabuľka obsahuje záznamy a sumár údajov. Druha tabuľka zahrnuje informácie z každého uloženého bodu záznamu. Tabuľky sú prepojene 1 ku N. Databáza je zhotovená podľa tretej normálovej formy.



Obrázok Návrh Databázy SQL LITE

### Návrh funkcii

Návrh databázy z dvomi tabuľkami kde prvá obsahuje informácie o jednej ceste a neznámy počet bodov. K ceste potrebujeme ako prvé vytvoriť id cesty aby sme mali k čomu priraďovať body z cesty. Aplikácia bude potrebovať:

* vytvoriť nastavenia ako často ukladať údaje,
* vytvorenie id pre cestu do ktorej budeme ukladať body,
* zbieranie údajov zo senzorov,
* do databázy vkladať dáta zo senzorov,
* do databázy upravovať niektoré informácie(napríklad názov cesty),
* do databázy mazať cestu, mazať body z cesty,
* vymazať celu databázu,
* z databázy čítať informácie o ceste a body cesty,
* zobrazenie cesty a záchytne body z cesty,
* vytvoriť súbor podľa údajov z databázy na zobrazenie prejdenej trasy,
* zobrazenie polohy na mape,
* zobrazenie prejdenej trasy,
* výpis všetkých ciest.

### Návrh užívateľského prostredia

Užívateľské prostredie bude mat 7 obrazoviek: úvodná, zber dát, zobrazenie aktuálnej pozícii, nastavenia, história, zobrazenie údajov o ceste, zobrazenie zaznamenanej trasy.

Úvodná obrazovka bude obsahovať:

* názov aplikácie, krátky popis,
* tlačítko história - na tomto mieste budú zobrazene trasy,
* nastavenie aplikácie,
* možnosť výberu cestovného prostriedku a potvrdiť to tlačítko štart.

Zber dát bude obsahovať možnosť zrušiť cestu, dokončiť cestu, zobraziť aktuálnu polohu a vrátiť sa späť na domovskú obrazovku. Tato obrazovka bude zobrazovať údaje:

* dátum
* čas
* GPS( x, y, z, rýchlosť)
* dĺžka cesty
* čas cestovania
* senzory(akcelerometer, gravity, gyroskop, linear akcelerometer, orientantion, rotation vector) výpis z týchto senzorov je formátovaný na 3 desatinné miesta, každý senzor ma minimálne tri výstupne hodnoty
* senzor presnosť a presnosť GPS.

Nastavenia disponujú s možnosťami výberu predpripravenými a vlastnými hodnotami. Užívateľ si to môže zvoliť rýchlosť ukladania bodov v ceste. Ak je úložisko zariadenia malé je odporúčane zvoliť frekvenciu ukladania na pomalšiu. Každý bod zaberá miesto s informáciami v zariadení. Z nastavenia bude možnosť sa vrátiť späť na úvodnú obrazovku.

História bude zobrazovať názvy z ciest. Po kliknutí na cestu sa mu cesta otvorí v nasledujúcej šablóne „zobrazenie údajov o ceste“. Taktiež bude obsahovať možnosť vyčistiť celu databázu a voľbu vrátiť sa späť na hlavnú obrazovku.

Zobrazenie údajov o ceste vypíše údaje o ceste o každom bode k záznamu z cesty. Pri podržaní prstu na bode cesty vyskočí okno s otázkou o voľbe zmazať označený bod. Budú tu voľby ako:

* vrátiť sa na šablónu história,
* premenovať názov cesty,
* vymazať celu cestu,
* vygenerovať súbor KML prejdenej trasy,
* vygenerovať súbor KML pre vizualizáciu ako užívateľ prechádzal po trase,

prejsť na mapu a ukázať trasu po ktorej sme sa hýbali.

## 3.2 Metodika vývoja softvéru

Po návrhu si ako prvé usporiadajme kde začneme, pretože ak si to zle rozložíme prácu nebudeme to môcť testovať po krokoch a ak by sme to prerobili aby sme si to mohli otestovať by sme plytvali časom aby sme to prerábali a na konci to vracali k pôvodnému návrhu.

### Úvodná obrazovka

Z dôvodu dobre rozloženia prace začneme úvodnou obrazovkou. Tu si spravíme ako prvé vizuálnu časť. Popridávame komponenty texty a zarovnáme to do prehľadného stavu.

### Nastavenia

Po rozložení šablóny na úvodnej obrazovke sme sa zamysleli čo vieme použiť ďalej, história už zobrazuje údaje a tak najskôr potrebujeme údaje k zobrazeniu zozbierať. Získanie dát zo senzorov sme navrhli v intervaloch. Ako často sa bude slučka opakovať rozhodne užívateľ podľa toho čo bude potrebovať a preto je za potrebne spraviť nastavenia pre užívateľa ako krok druhý. Opäť spravíme rozloženie nastavenia a prepojíme to s hlavnou obrazovkou.

### Zbieranie dát

Po príprave šablóny a prepojenie s úvodnou obrazovkou sme chceli získať informácie z SharedPreferences. Naprogramujeme senzory dynamicky aby sme ich vedeli pridávať a odoberať. Na čítanie dát zo senzorov použijeme cyklus a ukladanie údajov do tabuľky podľa nastavení od užívateľa. Cela aplikácia je závislá na prístupu k GPS a tak musíme pred zberom dať vyžiadať od užívateľa prístup a kontrolovať ho po cely čas ci nebol vypnutý.

### Databáza

V databáze sme sa inspirovali z učiva ktoré sme preberali na svojej vysokej škole. Funkcie budú základne vkladanie, čítanie, upravovanie, mazanie. Naprogramuje sa hlavne základ, čítanie z databázy sa rozšíri podľa potreby zobrazovania v nasledujúcich obrazovkách.

### Vizualizácia aktuálnej polohy na mape v aplikácii

Vizualizácia bola navrhnutá do 3D máp Google Earth priamo v aplikácii.

### História

V tejto časti zobrazíme už zaznamenané cesty. Ako názov sme pre tieto cesty vymysleli formát „rok-mesiac-deň hh:mm:ss“ a dáme možnosť premenovať tieto názvy. V tejto časti zahrnieme aj voľbu vyčistiť celu databázu.

### Zobrazenie informácii z databázy o ceste

Tu len vyčítame údaje o ceste a bodoch, všetky tieto informácie zobrazíme. Ponúkneme tu stiahnutie záznamu trasy exportom do KML súboru.

Vďaka exportu súboru KML je možne použiť tento súbor aj v iných aplikáciách a platformách na vizualizáciu prejdenej trasy alebo pre ukážku animáciu zo záznamu ako sme cestovali.

Pred uložením sme si museli vyžiadať prístup od užívateľa k úložisku na uloženie súboru. Súbor sa ukladá v smartphone v priečinku ktorý patri danej aplikácii.

# 4 Výsledky

Aplikácia zbiera údaje, ukladá ich do databázy, zobrazuje záznamy z ciest a následne ponuka vizualizáciu v aplikácii alebo možnosť exportovať súbor KML do mobilného zariadenia a následne použiť na inej platforme. Aplikácia je funkčná a použiteľná.

## 4.1 Implementácia

Po návrhu aplikácie, databázy, výzoru a funkčnosti sa môžeme pustiť do tvorby aplikácie. Pre testovacie účely sme si zohnali Android telefón pre testovanie. Postup implementácie obrazoviek a tried v Android Studiu:

* úvodná obrazovka,
* nastavenia,
* zber údajov zo senzorov a GPS, vytvorenie databázy,
* zobrazenie aktuálnej pozícii,
* históriu,
* zobrazenie údajov z databázy,
* vizualizácia zaznamenanej trasy,
* export súboru KML zo zaznamenajú trasy,
* export súboru KML animácie z trasy.

Postupne sme si ladili výzor a funkčnosť aplikácie. Každý krok ktorý sme naprogramovali sme otestovali. Upravovali sme výzor a rozloženie a snažili sme sa ho čo najviac spriehľadniť o informáciách.

Pri programovaní aplikácie nastali aj náročne úlohy ktoré boli nahradené iným spôsobom riešenia ako napríklad Google Earth alebo náklon lietadla do strán.

Výsledná aplikácia a prechody medzi obrazovkami sú znázornene v prílohe:

Užívateľské rozhranie PRÍLOHA B1.

Po naprogramovaní prišlo reálne testovanie a zaznamenanie prvej trasy autom. Pri zobrazenie na Google máp sa ukázalo ako nie presne. Na mape ukazovalo že som letel vzduchom cca 240m nad zemou. Tento problém bol spôsobený tým že sme boli v danej nadmorskej výške reálne. Keď sme tuto nadmorskú výšku vložili do Google mapy tak na mapách nemajú presnú vizualizáciu nadmorskej výšky a musel som to vypočítať aby to sedelo s Google Mapami. Problém sa dal vyriešiť lebo Google mapy majú v sebe zabudovanú funkciu kde môžete zadať rôzne informácie a on vám podľa toho odpovie napríklad ak mu zadanému zemepisnú šírku a dĺžku sa môžeme spýtať aká je nadmorská výška na tejto pozícii. Po rozdielu našej nameranej výšky a výšky z Google máp sme sa dostali na výšku 40 metrov z 200m. Na upresnenie tejto odchýlky(40m) bude treba viac zariadení na testovanie aby sme mohli daný problém odstrániť.

Ďalšie prekážky nastali keď sme chceli použiť rotáciu smerom hore a dole. Vyriešili sme tento problém tak že postup spúšťania aplikácie je ako prvé poloziť telefón do pozícii v akej bude po cely čas a následne dať štart. Aplikácia si prvú hodnotu pri štarte uloží a od tejto hodnoty bude počítať ostatné hodnoty.

### Nastavenie

V nastaveniach po zostavení vizualizácie začíname programovať. Ukladanie informácii z nastavení sme sa rozhodli pre využitie vstavanej funkcii v Android Studio s názvom funkcie SharedPreferences(v preklade zdieľané predvoľby). Zdieľané predvoľby predstavujú spôsob, akým je možné ukladať a získavať malé množstvá primitívnych údajov ako páry kľúč/hodnota do súboru v úložisku zariadenia, ako je reťazec, int, float, boolean, ktoré tvoria vaše preferencie v súbore XML v aplikácii.

Zdieľané predvoľby sú vhodné pre rôzne situácie. Napríklad, keď je potrebné uložiť nastavenia používateľa alebo uložiť údaje, ktoré možno použiť pri rôznych činnostiach v rámci aplikácie. Údaje uložené pomocou zdieľaných preferencií sú v rámci aplikácie uchovávané ako súkromné. (Prateek\_Aggarwal, 2023).

Pri ukladaní nastavení sme vytvorili enum aby sme sa nemohli pomýliť v písaní kľúču na čítanie alebo zápis hodnoty. Zamedzili sme písaniu iných hodnôt ako čísla a obmedzili sme to na čísla od 1 po 999. Enum nahradil stringovu formu a mali sme tam viac stringov v enume tak sme museli tento problém vyriešiť a na to sme použili stránku stackoverflow. Našli sme riešenie problém

public enum AccessType {

Full(0),

ReadOnly(1),

Delete(2),

Add(3);

private final int code;

AccessType(int code) {

this.code = code

}

public int getCode() {

return this.code;

}

}

(GGizmos, 2015).

Z tohto kódu som si to upravil na vyriešenie svojho problému:

public enum NameKeys {  
 *UserSettings*(0),  
 *SettingsCheckedRadioButtonIndex*(1),  
 *RereshTime*(2),  
 *TextViewValue*(3),  
 *MainCheckedRadioButtonIndex*(4),  
 *FirstSet*(5);  
  
 NameKeys(int value) {  
 this.value = value;  
 }  
  
 private final int value;  
  
 private final String[] gNameKey = { "UserSettings", "SettingsCheckedRadioButtonIndex", "RereshTime", "TextViewValue", "MainCheckedRadioButtonIndex", "FirstSet" };  
  
 public String KeyName() {  
 return gNameKey[value];  
 }  
  
 public int length() {  
 return gNameKey.length;  
 }  
}

Číslo v enume sme použili ako index k poľu gNameKey. V tejto časti sme mali pripravene hodnoty z natavení.

### Zbieranie dát

Pri testovaní sme narazili na problém, ak užívateľ od inštalácie nikdy nič nenastavil v nastaveniach aplikácie. Aplikácia nevedela čo ma nastaviť a nefungovala ako sme chceli. Preto sme pridali premennú nFirstSet ktoré reprezentuje ci bola niekedy nastavená hodnota v nastaveniach. Ak nebola priradili sme jej predvolenú hodnotu ako vidieť v kóde na GITe.

Pri programovaní senzorov ako sme písali v metodike sme sa snažili čo najviac spraviť dynamicky pridávanie a odoberanie senzorov.

for (int i = 0 ; i < goSensorsName.length; i++) {  
 goSensorManager[i] = (SensorManager) getSystemService(*SENSOR\_SERVICE*);  
 if (goSensorManager[i] != null) {  
  
 goSensor[i] = goSensorManager[i].getDefaultSensor(goSensorsName[i]);  
  
 if (goSensor[i] != null) {  
 goSensorManager[i].registerListener(this, goSensor[i], SensorManager.*SENSOR\_DELAY\_NORMAL*);  
 gbHasSensor[i] = true;  
 } else {  
 gbHasSensor[i] = false;  
 }  
 } else {  
 gbHasSensor[i] = false;  
 }  
}

Na senzory je vytvorená funkcia priamo v Android Studiu ktorá je zavolaná keď sa udaj v senzore zmení vďaka nej sa len spýtame ktorý senzor nám posiela zmenu. Do tejto funkcie si napíšeme všetky senzory na ktoré sa odkazujeme

@Override

public void onSensorChanged(SensorEvent event) {  
 int row;  
   
 if (event.sensor.getType() == Sensor.*TYPE\_ACCELEROMETER*) {  
 row = 0;  
 if ( !gbHasSensor[row] ) return;  
  
 goTempAcceleration.setText(Data3D(event, row));  
 }

...

}

(Hamad, 2013).

Nasledujúci problém ktorý sme museli vyriešiť bolo vyžiadať si povolenie k GPS a následne aktualizácie údajov z GPS. Najskôr sme museli v súbore AndroidManifest.xml dopísať riadky:

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />  
<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />  
<uses-permission android:name="android.permission.ACCESS\_BACKGROUND\_LOCATION" /> (Lacko, 2017).

Vytvorili sme si funkciu v ktoré sme si zavolali na vyžiadanie prístupu k GPS už vo svojej triede v zber údajov a na zobrazenie aktuálnych údajov nám poslúžila funkcia updateGPS

private void getCurrentPosition() {  
 LocationManager locationManager = (LocationManager) getSystemService(Context.*LOCATION\_SERVICE*);  
  
 Criteria criteria = new Criteria();  
 String provider = locationManager.getBestProvider(criteria, false);  
 if ( ActivityCompat.*checkSelfPermission*(this, Manifest.permission.*ACCESS\_FINE\_LOCATION*) != PackageManager.*PERMISSION\_GRANTED* &&  
 ActivityCompat.*checkSelfPermission*(this, Manifest.permission.*ACCESS\_COARSE\_LOCATION*) != PackageManager.*PERMISSION\_GRANTED*) {  
 return;  
 }  
  
 updateGPS(locationManager, provider);  
  
 if( gbFirstUse ){  
 gbFirstUse = false;  
 locationManager.requestLocationUpdates(provider, 0, 0, locationListener);  
 }  
 }

Ako prvé som použil tuto funkciu a vložil som ju do časovača aby mi vždy vyžiadal nove informácie podľa času, ale toto nefungovalo a tak som musel vyriešiť tento problém z aktualizáciou GPS a následne informácie boli uložene a zobrazene

private final LocationListener locationListener = new LocationListener() {  
  
 @Override  
 public void onLocationChanged(Location location) {  
 String temp;  
 //gps

temp = String.*valueOf*(location.getLatitude());  
 goTempGPS.setText(temp);  
 temp = String.*valueOf*(location.getLongitude());  
 goTempGpsLong.setText(temp);  
 temp = String.*valueOf*( location.getAltitude() );  
 goTempHeight.setText(temp);  
  
 // Získať hodnotu presnosti polohy  
 float accuracy = location.getAccuracy();  
 temp = String.*valueOf*( accuracy );  
 goTempAccuracy.setText(temp);  
  
 // Získať hodnotu presnosti GPS signálu  
 gLocationProvider = gLocationManager.getProvider(LocationManager.*GPS\_PROVIDER*);  
 float gpsAccuracy = gLocationProvider.getAccuracy();  
 temp = String.*valueOf*( gpsAccuracy );  
 goTempAccuracyGPS.setText(temp);  
 }  
 @Override  
 public void onStatusChanged(String s, int i, Bundle bundle) { }  
  
 @Override  
 public void onProviderEnabled(String s) { }  
  
 @Override  
 public void onProviderDisabled(String s) { }  
};

(Long, 2017).

Tato funkcia nám vyriešila problém s aktualizáciou GPS. Na získanie rýchlosti ktorú sme si vysvetli vyššie sme použili výpočet

//speed  
double speed = location.getSpeed() \* 3.6f;  
temp = String.*format*("%.2f km/h", speed);  
goTempSpeed.setText(temp);

(Amjad, 2016).

Ešte nám ostal si predstaviť časovač k zberu údajov. Na zber údajov sme vytvorili časovač ktorý sa nám opakuje v intervale 1000 milisekúnd. To znamená že každú sekundu sa spusti kód v bloku

private void startTimer(){  
 Double SaveDataTimer = *DEFAULT\_COMPARE\_TIME\_REFRESH*;  
 try {  
 SaveDataTimer = Double.*parseDouble*( String.*valueOf*(gnRefreshTimeMinute) );  
 Log.*d*("SaveDataTimer", "SaveDataTimer is " + SaveDataTimer);  
 } catch(Exception e){  
 Log.*e*("ErrorStartTimer", "error " + e);  
 }  
  
 Double finalSaveDataTimer = SaveDataTimer;  
 timerTask = new TimerTask() {  
 @Override  
 public void run() {  
 runOnUiThread(() -> {  
 String temp;  
  
 //timer  
 time++;  
 goTempTravelTime.setText( getTimerText(time) );  
  
 //datum  
 temp = new SimpleDateFormat("dd.MM.yyyy", Locale.*getDefault*()).format(new Date());  
 goTempDate.setText(temp);  
  
 //time  
 temp = new SimpleDateFormat("HH:mm:ss", Locale.*getDefault*()).format(new Date());  
 goTempTime.setText(temp);  
  
 //gps  
 if ( checkGPSLocation() ) getCurrentPosition();  
  
 //save data  
 if( ( finalSaveDataTimer <= ++gnRefreshTime ) || ( gbDistanceGoOut ) ){  
 gnRefreshTime = 0.0;  
 gbDistanceGoOut = false;  
  
 addSensorDataToDB();  
 updateWay();  
 }  
 });  
 }  
 };  
 timer.scheduleAtFixedRate(timerTask, 0, *INTERVAL*);  
}

(Javatpoint).

Ako vidíme v kóde porovnáme čas z nastavenia a keď sa rovnajú alebo je gnRefreshTime väčší tak vloží data do databázy cez addSensorDataToDB. Tu narazíme na to že databázu ešte nemáme naprogramovanú a preto sme tuto funkciu zatiaľ nechali prázdnu. Vrátime sa k nej keď naprogramujeme databázu. Neskôr dopíšeme funkciu kde budeme vkladať data do databázy podľa funkcii v databáze.

### Vizualizácia aktuálnej polohy na mape v aplikácii

Google Earth sa da použiť len v API po verziu 25 nás softvér ma základ API 33 a preto sme museli uchýliť k inej platforme. Ako náhradnú platformu sme si vybrali Google mapy. Ak chceme využívať niektoré funkcie na Google máp musíme si vyžiadať unikátny kód ktorý musíme vložiť do programu. Z dôvodu že tento kód nemôže byt zverejnený je prepísaný aj v kóde na GITe a nie je zverejnený funkčný kľuč. Návod ako si tento kľuč vyžiadať sme sa inspirovali na YouTube. (Tech\_WsCube, 2022; freeCodeCamp.org, 2021).

Pri vizualizácii sme narazili na úskalie obnovovania pozície. To sme znova museli vyriešiť cez podobnú funkciu, ako v zbere údajov.

Pri náklone do strán sme našli v dokumentácii pre KML súbory možnosť náklonu. Po naprogramovaní náklonu sa ukázalo že je to náklon stúpania a klesania a nebolo možne ho použiť do strán.

### Zobrazenie informácii z databázy o ceste

Tu len vyčítame údaje a všetky ich zobrazíme. Pasovali sme sa tu s komplikáciami že nám padla aplikácia. Pri alfa testovaní sme zistili že ťažkosť nastala keď sme chceli vyčítať cesty a body ale daná cesta žiadny bod cesty nemala. Ďalšie prekážky sme museli riešiť so súbormi. V systéme Android 11 už aplikácie nemôžu pristupovať k súborom vo vyhradenom adresári inej aplikácie v rámci externého úložiska. (Delgado, 2021). Museli sme si vyžiadať prístup k súborom v súbore AndroidManifest.xml dopísať riadky.

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />  
<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />  
<uses-permission android:name="android.permission.MANAGE\_EXTERNAL\_STORAGE"  
 android:minSdkVersion="25"  
 tools:ignore="ScopedStorage" />

(PerracoLabs, 2020).

Keď sme mali prístup k ukladaniu súborov mohli sme si naprogramovať triedu na vytvorenie súboru KML. Ako prvý sme naprogramovali súbor na zobrazenie zaznamenanej trasy.

Údaje do súboru sme si vytiahli z databázy a vložili sme si ich do poľa a pole sme poslali do triedy na vytvorenie súboru. Pred odoslaním poľa si ešte data spracujeme na výpočet výšky. Ak je zaznamenaná trasa pre lietadlo spracovávame výšku, v prípade že je iný dopravný prostriedok tak posielame výšku nula. V prípade že ide o iný dopravný prostriedok ako lietadlo tak ani si nepýtame nadmorskú výšku z Google máp. Tato funkcia na získanie údajov o nadmorskej výške najviac spomaľuje generovanie súboru KML. Zoberieme si údaje zemepisnej šírky a dĺžky a vypýtame si z Google máp nadmorskú výšku na danej polohe. Existuje aj funkcia ktorej si môžeme zadať všetky body a Android Studio nám odpovie hodnotami v poli. Tato funkcia nám urýchli aplikáciu keby mala aplikácia čakať po každom bode na odpoveď by aplikácia bola veľmi pomalá. Po získaní všetkých údajov o výške na daných pozíciách si môžeme odpočítať našu výšku od nadmorskej výšky na mape. Po prepočítaní by sme mali dostáť výsledok 0 ak sme boli na zemi. V prípade že sme vo výške dostaneme hodnoty o koľko sme boli vyššie voči nadmorskej výške.

Pole dát posielame do triedy na vytvorenie súboru. Spravili sme to preto tak aby sme nemuseli pridávať databázu do triedy na vytvorenie súboru. Vyžadovalo by to aj prístup k polohe kde by sme museli robiť ďalšie kontroly. Z triedy kde sme pole vytvorili už všetko bolo zabezpečene. Nasledovalo prídavok vytvorenie súboru pre animácie a len sme rozšírili už funkčnú triedu kde už sme mali dobrý základ na rozšírenie.

Vďaka exportu súboru KML je možne použiť tento súbor aj v iných aplikáciách a platformách na vizualizáciu prejdenej trasy alebo pre ukážku animáciu zo záznamu ako sme cestovali.

Pred uložením sme si museli vyžiadať prístup od užívateľka k úložisku na uloženie súboru. Súbor sa ukladá v smartphone v priečinku ktorý patri danej aplikácii.

Pri vizualizácii všetkých bodov sme museli pridať text k údajom aby používateľ vedel o aký udaj sa jedna. Čo sa ukázalo ako náročnejší zádrhel než to znie. Pretože sa body zobrazujú cez listview do ktorého išli priamo údaje z databázy. Sme pri riešení tohto problému museli ku každému udajú vložiť správny popis. Pomohli sme si na fóre stackoverflow. (Ganguly, 2015).

## 4.2 Popis vytvoreného softvéru

Vytvorení softvér ponúka možnosť zaznamenania prejdenej trasy. Softvér ma možnosť exportu súborov do KML a môžete ich tak medzinárodným formátom prepojiť s inými aplikáciami. Aplikácia nepoužíva server a tak vaše údaje ostávajú len v majiteľových rukách a nikto s nimi nebude okrem majiteľa disponovať.

V aplikácii mate možnosť si vybrať spôsob cestovania alebo aj frekvenciu ukladania údajov v zariadení. Program ukladá pozíciu len po stlačení tlačidla štart až do ukončenia alebo zrušenia.

Človek sa tiež učí aj zo svojich chyb a preto história trasy môže poskytnúť dobrý náhľad na svoje zdokonaľovanie.

## 4.3 Testovanie

Ak chcete byt dobrým testerom vašou úlohou je pokaziť aplikáciu. Ak vás výsledok bude že aplikácia bola bez chybná boli ste zlý tester. Takmer vždy sa nájde niečo na čo sa zabudlo alebo možnosť s ktorou sa nepočítalo ako jedna funkcia môže ovplyvniť druhu funkciu. Skúšať všetky možnosti a ako reagujú iné funkcie. Zadávať tam nulu, záporne, kladne, prestrelene čísla, neplatne znaky.

Alfa testovanie prebiehalo počas vývoja. Pri testovaní bola testovaná vizáž funkčnosť. Boli zistene chyby ako text ktorý pretekal a rozhádzal celu obrazovku. Zlyhanie aplikácie pri chybách v programovaní ako príklad uvediem aplikácia si vypýtala údaje z databázy ale v databáze nič nebolo poslala výsledok null a program s takýmto výsledkom nepočítal a program sa ukončil. Pri alfa testovaní bol objavený problém ak obrazovka zhasne neukladá žiadne údaje zo senzorov. Tento problém som vyriešil že po štarte sa obrazovka nevypne až kým nebude ukončená aplikácia alebo zrušená trasa.

Po funkčnej aplikácii a zozbieraní niekoľkých dát prišlo na beta testovanie. Tu som zvolil metódu testovania „pokaz čo sa da“.

Ako prvé som zvolil testovanie nastavenia skúsiť zadať znaky čísla písmena. Po neúspechu som sa pokúsil vložiť záporne číslo alebo nulu opäť som sa stretol s neúspechom. Skúsil som prekročiť povolene hodnoty nič aplikáciu nepokazilo. Vyskúšal som všetky možnosti v nastaveniach s funkčnosťou aplikácie. Aplikácia ukázala 1 sekundový rozdiel a tým padol som prvý krát uspel.

Prišlo na radu otestovať senzory a spôsoby cestovania aby sme mali dáta na testovanie pri ďalších funkciách. Vybral som si prvú možnosť cestovania a začal som robiť zle aplikácii zakázal som jej zaznamenávať polohu počas ukladania údajov aplikácia si opätovne vyžiadala prístup k polohe. Skúsil som najskôr vypnúť v nastaveniach povolenie polohy a zvoliť si druhu možnosť cestovania v aplikácii a dať štart v aplikácii a nechať ju chvíľku stáť bez údajov a potom zamietnuť povolenie polohy. Aplikácia si opätovne vyžiadala prístup a všetko fungovalo. Snažil som sa rýchlo dať štart a hneď trasu ukončiť aby nevytvoril žiadny bod zo senzorov. Aplikácii to neuškodilo. Vyskúšal som si režim lietadla a s telefónom som hýbal do všetkých strán a skúšal ci to nezhodí nežiadaná hodnota aplikácia to zvládla.

Pokračoval som ďalším testovaním vymazal som databázu a vyskúšal som ci mi aplikácia nepadne ak dostane z databázy odpoveď null. Tuto chybu som odstránil pri alfa testovaní ale pri beta som si ju zopakoval ci to neovplyvni aj niečo iné.

Ďalší krok bolo otestovanie zobrazenia údajov a tak som sa snažil vytvoriť cestu bez jediného bodu zo senzorov aplikácia to zvládla (tiež bola tato chyba odstránená počas alfa testovania). Pokračoval som generovaním súborov bez údajov aplikácia to prežila, ale nastal tu problém s povolením ukladať súbor ak neboli udelene pravá a mi sme ich potvrdili aplikácia súbor ani opätovne neuložila a tým padol som uspel druhy krát pri testovaní. Problém som vyriešil že si vyžiadam potvrdenie ukladať súbory hneď na úvodnej obrazovke a taktiež ho overujem pri vstupe na obrazovku kde je funkcia na ukladanie súborov. Taktiež zvládla premenovanie cesty alebo zmazanie. Nasledovalo otestovať vizualizáciu mapy bez údajov všetko aplikácia vydržala.

Vo výsledku testovania sa našli chyby ktoré boli odstránene alebo čiastočne vyriešene.

# Záver

Na vývoj softvéru som sa snažil používať už existujúce pripravene technológie, Softvér sa podarilo úspešne dokončiť. Aplikácia nepadá a je stabilná. Ciel záverečnej prace bol vizualizovať zaznamenanú trasu. Ciel bol dosiahnutý ako v aplikácii tak aj možnosťou exportu súboru.

Aplikácia je vhodná pre užívateľov ktorý by radi ukázali ako išli a kde boli. Ďalšia skupina užívateľov je pre ľudí ktorý by mali záujem zlepšovanie trasy alebo zlepšovať svoje letecké manévre.

Aplikácia ponuka navyše podľa vašich preferovaných nastavení v telefóne ako biele alebo čierna téma. Ďalej aplikácia navyše ponuka preklad v angličtine to je základný jazyk v prípade že mate v telefóne nastavenú slovenčinu tak aplikácia automaticky zmení jazyk na slovensky.

Do aplikácii sa nepodarilo dostáť Google Earth. Tento problém sme vyriešili exportom súboru KML ktorý môžete vložiť na stránke do Google Earth. Taktiež sa nepodarilo vizualizovať náklon do strán. Tieto dáta sú dostupne v zariadení ale aktuálne KML kódovanie nám tuto možnosť neponúka. V prípade že by ju v budúcnosti pridali by sme ju veľmi ľahko pridali do zobrazenia. Pokúšal som sa dostáť aj obrázok na export súboru aby prezentoval lietadlo ale nepodarilo sa mi daný obrázok otáčať a nepripadalo mi to vizuálne pekne keď lietadlo cúvalo a tak z konečného dôsledku som od toho upustil.

Aplikácia taktiež obsahuje dáta ktoré sa v budúcnosti budú môcť doprogramovať a využiť ak ich bude potreba, čo užívateľa môže milo prekvapiť novinkami a aktualizáciou zobrazenia tras ktoré už dávnejšie zaznamenal. Taktiež ponúkame užívateľovi súkromie vo forme že dáta neodosielame na server kde môže mat k týmto dátam prístup niekoľko spoločnosti a ľudí.

Pri výtvarní súboru KML zo záznamu lietania trvá dlhšie je to spôsobene tým že aplikácia si vyžaduje výšku polohy z Google máp a pre každý zaznamenaný bod čaká na odpoveď zo servera čo užívateľa nepoteší z dôvodu že to bude vyzerať že aplikácia zamrzla. No ak je tam veľa bodov musí čakať na odpoveď z Google máp a nechať aplikáciu pracovať a prepočítavať výšku ktorú zobrazí.

Aplikáciu vo výsledku hodnotím pozitívne a užitočne. Pevne dúfam že bude dobre slúžiť aj ďalšie dekády. Je užívateľsky jednoduchá na používanie. Užívateľ by sa mal v aplikácii dokázať zorientovať do niekoľkých minút po nainštalovaní.

# Bibliography

**Amjad, Haroon. 2016.** stackoverflow. *com.* [Online] 31. 10 2016. [Dátum: 31. 3 2023.] https://stackoverflow.com/questions/40339628/onlocationchanged-called-twice-on-every-interval.

**Delgado, Carlos. 2021.** ourcodeworld. *com.* [Online] 28. 7 2021. [Dátum: 31. 3 2023.] https://ourcodeworld.com/articles/read/1559/how-does-manage-external-storage-permission-work-in-android.

**Divelements Limited. 2009.** skydemon. *aero.* [Online] 2009. [Dátum: 31. 3 2023.] https://www.skydemon.aero/.

**freeCodeCamp.org. 2021.** youtube. *com.* [Online] 26. 1 2021. [Dátum: 31. 3 2023.] https://www.youtube.com/watch?v=\_xUcYfbtfsI&ab\_channel=freeCodeCamp.org.

**Ganguly, Kanishka. 2015.** stackoverflow. *com.* [Online] 18. 1 2015. [Dátum: 31. 3 2023.] https://stackoverflow.com/questions/28002090/append-text-to-data-returned-from-sqlite-query-before-displaying-in-listview.

**GGizmos. 2015.** stackoverflow. *com.* [Online] 5. 1 2015. [Dátum: 2023. 3 31.] https://stackoverflow.com/questions/27771936/java-enum-with-numeric-values.

**Google LLC.** android. *com.* [Online] [Dátum: 31. 3 2023.] https://developer.android.com/guide/topics/sensors/sensors\_motion.

**Hamad. 2013.** stackoverflow. *com.* [Online] 28. 12 2013. [Dátum: 31. 3 2023.] https://stackoverflow.com/questions/20813386/onsensorchangedsensorevent-event-always-show-the-same-result.

**Hecl, David. 2014.** letemsvetemapplem. *eu.* [Online] 23. 1 2014. [Dátum: 27. 3 2023.] https://www.letemsvetemapplem.eu/2014/01/23/jak-funguje-digitalni-gyroskop-princip-toho-proc-iphone-dokaze-urcit-v-jake-je-poloze/.

**Hikingproject. 2014.** hikingproject. *com.* [Online] 2014. [Dátum: 31. 3 2023.] https://www.hikingproject.com/.

**Jarko, Milan. 2019.** etuo. *sk.* [Online] 3. 4 2019. [Dátum: 27. 3 2023.] https://www.etuo.sk/blog/gyroskop-v-telefone-co-to-je-a-na-co-je/1012.

**Javatpoint.** javatpoint. *com.* [Online] [Dátum: 31. 3 2023.] https://www.javatpoint.com/post/java-timer-scheduleatfixedrate-method.

**Lacko, Ľuboslav. 2017.** *Mistrovství - Android.* Brno : Computer Press, 2017. 978-80-251-4878-5.

**Long, Jian Wei. 2017.** stackoverflow. *com.* [Online] 14. 2 2017. [Dátum: 31. 3 2023.] https://stackoverflow.com/questions/42218419/how-do-i-implement-the-locationlistener.

**Mapserver.** mapserver. *org.* [Online] [Dátum: 31. 3 2023.] https://mapserver.org/el/input/vector/kml.html.

**Microsoft.** microsoft. *com.* [Online] [Dátum: 31. 3 2023.] https://support.microsoft.com/sk-sk/office/z%C3%A1kladn%C3%A9-inform%C3%A1cie-o-form%C3%A1te-xml-a87d234d-4c2e-4409-9cbc-45e4eb857d44.

**Msg LIfe.** msgprogramator. *sk.* [Online] [Dátum: 31. 3 2023.] https://msgprogramator.sk/java-vyhody/.

**OGC. 1994.** ogc. *org.* [Online] 1994. [Dátum: 31. 3 2023.] https://www.ogc.org/.

**Paronai, Tomáš. 2021.** goodrequest. *com.* [Online] 18. 6 2021. [Dátum: 18. 6 2021.] https://www.goodrequest.com/sk/blog/java-vs-kotlin.

**PerracoLabs. 2020.** stackoverflow. *com.* [Online] 28. 12 2020. [Dátum: 31. 3 2023.] https://stackoverflow.com/questions/64221188/write-external-storage-when-targeting-android-10.

**Prateek\_Aggarwal. 2023.** geeksforgeeks. *org.* [Online] 2. 3 2023. [Dátum: 2023. 3 31.] https://www.geeksforgeeks.org/shared-preferences-in-android-with-examples/.

**Roubalová, Eliška. 2015.** *Java bez předchozích znalostí.* Brno : Computer Press, 2015. 978-80-251-4572-2.

**Schildt, Herbert. 2012.** *Java 7.* Brno : Computer Press, 2012. 978-80-251-3748-2.

**Strava, Inc. 2009.** strava. *com.* [Online] 2009. [Dátum: 31. 3 2023.] https://www.strava.com/.

**Tech\_WsCube. 2022.** youtube. *com.* [Online] 29. 1 2022. [Dátum: 31. 3 2023.] https://www.youtube.com/watch?v=BO1utHYhsms&ab\_channel=WsCubeTech.

**Urban, František. 2018.** touchit. *sk.* [Online] 8. 12 2018. [Dátum: 29. 3 2023.] https://touchit.sk/ako-spozna-smartfon-kde-ste/206763.

# Zoznam príloh

Príloha A – Zdrojový kód + odkaz na GIT

Príloha B – Fotodokumentácia + odkaz na GIT

PRÍLOHA A

**Príloha A**

* A1 Odkaz na GIT: https://github.com/gameblaer/zaverecna\_praca\_bc/
* A2 Aplikácia odkaz na GIT: https://github.com/gameblaer/zaverecna\_praca\_bc/blob/main/Aplikacia/WIW\_Google\_Earth.zip

**Príloha B**

* **B1** Odkaz na GIT: https://raw.githubusercontent.com/gameblaer/zaverecna\_praca\_bc/main/Images/B1.png

1. Zdroj Graf 1 https://crast.net/245070/if-you-want-to-work-as-a-programmer-in-2023-these-are-the-languages-you-should-learn/ [↑](#footnote-ref-1)
2. Zdroj Obrázok 2 <https://play.google.com/store/apps/details?id=aero.skydemon.skydemonandroid&hl=en_US&pli=1> [↑](#footnote-ref-2)
3. Zdroj Obrázok 3 <https://play.google.com/store/apps/details?id=com.strava&hl=sk&gl=US> [↑](#footnote-ref-3)
4. Zdroj Obrázok 4 (Hikingproject, 2014) [↑](#footnote-ref-4)
5. Zdroj Obrázok 5 (Hikingproject, 2014) [↑](#footnote-ref-5)
6. Zdroj video 6 https://www.youtube.com/watch?v=eqZgxR6eRjo&t=60s&ab\_channel=HowToMechatronics [↑](#footnote-ref-6)
7. Zdroj video 7 https://www.youtube.com/watch?v=eqZgxR6eRjo&t=60s&ab\_channel=HowToMechatronics [↑](#footnote-ref-7)
8. Zdroj obrázkov 8 <https://www.letemsvetemapplem.eu/2014/01/23/jak-funguje-digitalni-gyroskop-princip-toho-proc-iphone-dokaze-urcit-v-jake-je-poloze/> [↑](#footnote-ref-8)
9. Zdroj video 9 https://www.youtube.com/watch?v=eqZgxR6eRjo&t=60s&ab\_channel=HowToMechatronics [↑](#footnote-ref-9)
10. Zdroj obrázok 10 https://www.timeanddate.com/astronomy/horizontal-coordinate-system.html [↑](#footnote-ref-10)
11. Zdroj Obrázok https://touchit.sk/ako-spozna-smartfon-kde-ste/206763 [↑](#footnote-ref-11)
12. Zdroj Obrázok 12 https://touchit.sk/ako-spozna-smartfon-kde-ste/206763 [↑](#footnote-ref-12)