

UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE
FAKULTA PRÍRODNÝCH VIED A INFORMATIKY

**Vizualizácia dát získaných zo senzorov mobilného
zariadenia**
BAKALÁRSKA PRÁCA

UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE
FAKULTA PRÍRODNÝCH VIED A INFORMATIKY

VIZUALIZÁCIA DÁT ZÍSKANÝCH ZO SENZOROV
MOBILNÉHO ZARIADENIA
BAKALÁRSKA PRÁCA

Študijný odbor:	18. Informatika
Študijný program:	Aplikovaná informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	Mgr. Martin Vozár, PhD.



Univerzita Konštantína Filozofa v Nitre
Fakulta prírodných vied a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Martin Vrták
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., externá forma)
Študijný odbor: informatika
Typ záverečnej práce: Bakalárska práca
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Vizualizácia dát získaných zo senzorov mobilného zariadenia

Anotácia: Anotácia:
Zaznamenávanie prejdenej trasy z turistiky, cestovania, bicyklovania alebo lietania je v súčasnosti známe. Zaujímavý, efektívny a názorný pohľad na preletenú trasu môže byť vtedy, ak je na preletenej trase zobrazený aj objekt, ktorý mení polohu podľa zaznamenaných dát zo senzorov. Takto zobrazená trasa letu spolu so zobrazením polohy lietadla môže byť vhodnou názornou pomôckou pre budúcich pilotov.

Cieľ práce:
Analyzovať existujúce možnosti vizualizácie dát získaných zo senzorov a navrhnúť a naprogramovať vlastné riešenie pre vizualizáciu zaznamenaných GPS súradníc a iných dát zo senzorov ako je napríklad gyroskop, rýchlomer, výškomer apod. Na základe týchto dát aplikácia zobrazí na 3D mape priebeh zaznamenatej trasy s animáciou objektu (náklon, rýchlosť, výška,...), ktorý sa po dráhe pohyboval.

Charakter práce:
vývoj softvéru - pozostávajúci z popisu riešeného problému, analýzy existujúcich riešení, návrhu vlastného riešenia, metodiky vývoja/tvorby, implementácie, popisu vytvoreného riešenia, testovania aplikácie používateľmi.

Predmetové prerekvizity:
Vývoj mobilných aplikácií (3., bc)
Princípy softvérového inžinierstva (1., Mgr.)
Programovanie aplikácií so senzormi (3., bc)

Najdôležitejšie kompetentnosti získané spracovaním témy (cca 5):
formulácia požiadaviek na vývoj softvéru a kontrola požadovaných funkcií a funkcionality vyvíjaného softvéru
postupy programovania aplikácií pre webové stránky, mobilné telefóny a tablety
vyvinúť aplikácie na spracovanie údajov
navrhovanie vhodných softvérových technológií pre konkrétne zadanie, vrátane návrhu riešenia na optimalizáciu a zvyšovanie efektívnosti prostriedkov výpočtovej techniky
implementácia vhodných metód a techník na vývoj softvéru a testovanie vyvíjaných softvérových riešení



Univerzita Konštantína Filozofa v Nitre
Fakulta prírodných vied a informatiky

aplikácia metód a techník vývoja a integrácie softvérových komponentov
do rozsiahlejšieho celku (výber, vyhodnotenie a integráciu rôznych
softvérových komponentov a modulov do aplikačného celku)

Školiteľ: Mgr. Martin Vozár, PhD.
Oponent: PaedDr. Viera Michaličková, PhD.
Katedra: KI - Katedra informatiky
Dátum zadania: 04.11.2021

Dátum schválenia: 09.01.2023

RNDr. Ján Skalka, PhD., v. r.
vedúci/a katedry

POĎAKOVANIE

Predovšetkým by som chcel poďakovať môjmu školiteľovi Mgr. Martinovi Vozárovi, PhD. za dobrý prístup, komunikáciu, za konzultácie, zber údajov a rady.

Ďalej by som chcel poďakovať RNDr. Jánovi Skalkovi, 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. 56 s.

Tato záverečná práca popisuje vývoj mobilnej aplikácie od požiadaviek, k cieľom práce, po zobrazenie a následné testovanie prejdenej trasy. Výsledkom práce je funkčná mobilná aplikácia pre Android so zámerom ukázať používateľovi históriu prejdenej trasy údajovo aj vizuálne pre budúcu analýzu. Dáta z aplikácie môžu pomôcť na vylepšenie trasy, prípadne zdokonalenie vlastných schopností alebo možnosti predat' svoje skúsenosti, či na vzdelávanie študentov a poskytnutie ukážky správnej jazdy/trasy pri šoférovaní/riadení. Aplikácia navyše ponúka možnosť generovania súborov KML zo zozbieraných dát, možnosť slovenského alebo anglického jazyka podľa nastavení v telefóne, prípadný vyber bielej alebo tmavej témy podľa preferencií v telefóne. Vygenerované súbory KML umožňujú náhľad z hora, alebo použitie externých programov na vizualizáciu zaznamenananej trasy; kamera sa nakláňa v smere používateľovho pohľadu.

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

ABSTRACT

Vrták, Martin: Visualization of data obtained from mobile phone sensors. [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. 56 p.

This bachelor thesis describes the development of the mobile application from the requirements, to the work goals, to display and final testing of the route. The outcome is a practical mobile application for Android with the aim of showing the user the history of the taken route from the data and visual point of view for future analysis. The data from the application may help improve the route, possibly one's skills or the opportunity to sell one's experience, and help educate students by demonstrating the right driving route. In addition, the application offers the possibility of generating KML files from the collected data, the selection of the Slovak or English language, and white or dark theme according to the preferences on the phone. The generated KML files enable the view from above or the use of external programs to visualize the recorded route, the camera tilts in the direction of the user's view.

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

OBSAH

Úvod	10
1 Popis riešeného problému / Analýza súčasného stavu	11
1.1. Vývojové prostredie	11
Jazyk C	11
Swift	12
Kotlin	12
Java	12
1.2 Výber vývojového prostredia	14
1.3 Existujúce Aplikácie	15
2 Ciele záverečnej práce	18
3 Návrh a metodika	19
3.1 Návrh softvéru	19
Zbieranie údajov zo senzorov	19
Návrh Databázy	30
Návrh funkcií aplikácie	31
Návrh používateľského prostredia	32
3.2 Metodika vývoja softvéru	33
Úvodná obrazovka	33
Nastavenia	33
Zbieranie dát	34
Databáza	34
Vizualizácia aktuálnej polohy na mape v aplikácii	34
História	34
Zobrazenie informácií z databázy o ceste	34
4 Výsledky	35
4.1 Implementácia	35
Nastavenie	36
Nastavenie podľa preferencií v telefóne	37
Zbieranie dát	38
Zobrazenie údajov o ceste	41
Vizualizácia aktuálnej polohy na mape v aplikácii	42
Zobrazenie informácií z databázy o ceste	42
4.2 Popis vytvoreného softvéru	43
4.3 Testovanie	44

Záver	46
Zoznam bibliografických odkazov	47
Zoznam príloh	49

ÚVOD

V dnešnej dobe so stále 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 zaznamenatej trasy pomôže spočívať v návrhu lepšej trasy, nájdením nedostatkov v aktuálnej trase. V dnešnej dobe väčšina ľudí disponuje mobilným zariadením, a preto nie je potrebná kúpa špeciálneho zariadenia. Mobilné zariadenia majú v sebe zabudované rôzne senzory a prístupovať k nim môže každý programátor, ak si od používateľa vyžiada prístup k senzorom. Vďaka tomu, že dnešné mobilné operačné systémy majú vbudované aktualizácie vieme vďaka aplikácii rýchlo spraviť nový doplnok k trasovaniu alebo urobiť opravu aplikácie v prípade problémov. Ukážeme si, aké problémy je potrebné riešiť a na čo netreba zabudnúť. Prejdeme si vyžiadanie prístupu k senzorom a zberom údajov z mobilného zariadenia, ukladaním údajov do SQL Lite, vďaka čomu je možné čítanie z databázy, zobrazením máp v aplikácii a obnovovaním polohy, komunikáciou s Google mapami a možným exportom súborov do externých aplikácii ako napríklad Google Earth. Taktiež si ukážeme, ako spraviť multilingválne aplikácie podľa nastavenia telefónu.

1 POPIS RIEŠENÉHO PROBLÉMU / ANALÝZA SÚČASNÉHO STAVU

Aplikáciu je možné používať na záznam prejdenej trasy, ktorý sa dá neskôr zobrazit'. Možnosť prezrieť si trasu, kde všade sme boli, dokáže pomôcť pri budúcej analýze pre optimalizáciu trás. Nadobudnuté údaje môžu byť nápomocné pre nových pilotov, ktorí sa môžu poučiť zo svojich chýb. Učiteľ môže vizualizovať svoje skúsenosti svojim žiakom. Je však potrebné si dať pozor, pretože takéto informácie sa dajú ľahko zneužiť na tajné sledovanie používateľov.

1.1. VÝVOJOVÉ PROSTREDIE

Na svete sú najviac rozšírené operačné systémy iOS a Android. Existujú aj ďalšie operačné systémy, ale nie sú rozšírené. Operačný systém iOS patrí firme Apple. Android vyvíja firma Google. Najviac využívané vývojové prostredia pre operačné systémy iOS a Android sú:

- Xcode(Apple),
- Android Studio.

Xcode vývojové prostredie vyžaduje platformu od Apple (v dobe písania tejto práce) Mac, iPad. V tomto prípade nás Xcode obmedzuje iba na platformu od Apple. Pre Xcode je potrebné poznať jeden z týchto programovacích jazykov:

- Objective C,
- Swift code.

Pre vývojové prostredie Android Studio máme opäť na výber programovacie jazyky. Naproti tomu Android Studio môžeme nainštalovať na rôzne operačné systémy ako Windows, Mac, Linux. Android Studio nám ponúka 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 predstavuje štruktúrované programovanie, čo znamená, že kód organizujeme do blokov a funkcií.

Swift

Apple vyvinul jazyk Swift pre svoje prostredie. Swift nám ponúka možnosť Objektovo orientované 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 použiteľný softvér.

Kotlin

Kotlin sa zameriava na bezpečnosť (náhodných chýb v kóde). Ponúka vylepšenú typovú kontrolu hodnoty null, čo pomáha minimalizovať nevhodné stavy v kóde. Kotlin má jednoduchšiu syntax.

Kotlin 1.0 bol vydaný začiatkom roka 2016 spoločnosťou JetBrains. Kotlin je staticky typovaný programovací jazyk. Pri kompilácii 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 Kotlinе máme voľbu definovať premennú ako Nullable (v Kotlinе sa to nazýva Optional) val name: String? Tu s istotou vieme povedať, že premenná môže obsahovať null. Ak by sme ju definovali bez otáznika, vieme s istotou povedať, že premenná nikdy nemôže mať null.

Môžeme povedať, že Java a Kotlin majú rovnakú rýchlosť písania kódu. Kotlin má viac lakonických konštrukcií, ktoré umožňujú Android developerovi menej písať. Má to však jeden háčik - na hľadanie riešenia úlohy v Kotlinе potrebujeme 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).

Java

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 najpoužívanějšíe programovacie jazyky, čo vzhľadom na benefity Java programovania nie je žiadnym prekvapením.

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

Javu možno definovať ako univerzálny programovací jazyk. To znamená, že je možné ju použiť na programovanie takmer 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 Jave 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. Môžeme sa tak nerušene venovať programovaniu a nezaťažovať sa potenciálnymi problémami súvisiacimi so správou pamäte.

Okrem bohatej odbornej literatúry a online návodov nám pri programovaní v Java môžu pomôcť aj komunity pre developerov. Na svete existuje vyše 9 miliónov programátorov, ktorí sa s rovnakým, prípadne podobným problémom tomu nášmu už stretli, a teda si s ním vedia poradiť. A keďže Java predstavuje open-source programovací jazyk, tak nájsť pomoc je skutočne 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.

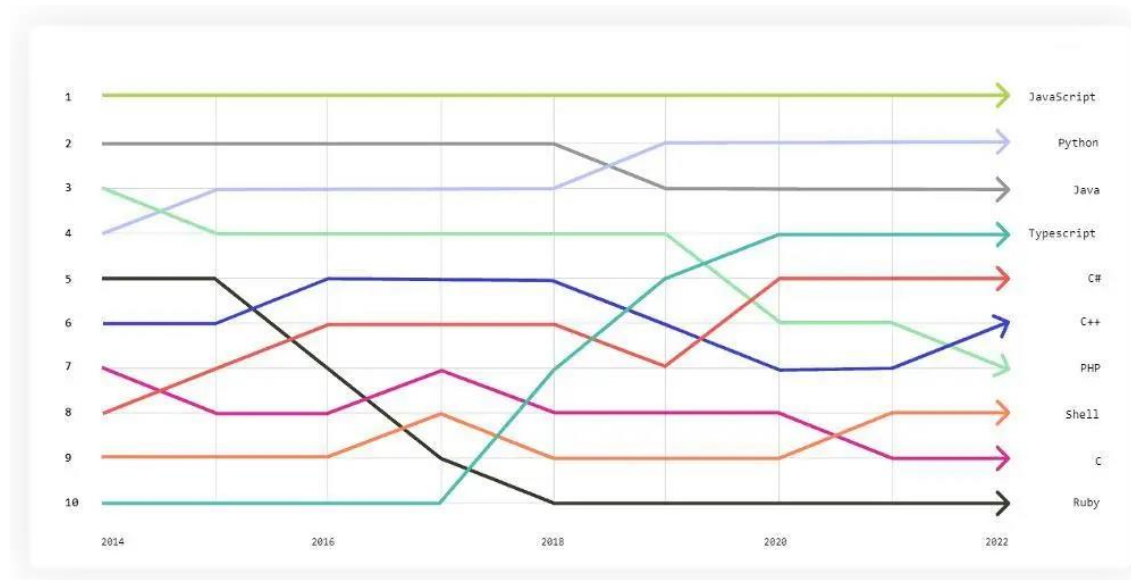
Na rýchlosť jazyka má negatívny dopad tiež automatická správa pamäte. Hoci programátorovi uľahčuje prácu, výrazne znižuje rýchlosť Javy v porovnaní s programovacími jazykmi, pri ktorých má správu pamäte na starosti programátor. Pre

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, 2023).

1.2 VÝBER VÝVOJOVÉHO PROSTREDIA

Na základe informácií, ktoré sme uviedli vyššie, sme sa rozhodli použiť ako vývojové prostredie Android Studio.



Zdroj Graf 1. Najrozšírenejšie programovacie jazyky a ich preferovanie od 2014 - 2022¹

Dôvody pre voľbu Android Studio:

- Android Studio môžeme nainštalovať na Windows / Linux / Mac,
- Android Studio sme sa naučili používať už na našej alma mater,
- Android Studio ma medzi primárnymi programovacími jazykmi Javu,
- dostupná literatúra pre Android Studio (Lacko, 2017).

Uvádzame ďalšie dôvody použitia jazyka Java:

- Java patrí medzi najpreferovanejšie jazyky (vid'. Graf 1) a tak nám ponúka bohatú základňu pri riešení problémov na fórach,

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/>

- kód v Jave sa dá použiť na akomkoľvek hardvéri,
- Javu vieme migrovať na Kotlin,
- programovanie v Jave sme si osvojili už v našej alma mater,
- z ponúkaných jazykov máme s Javou najviac skúsenosti,
- rozsiahle znalosti z dvoch odborných kníh o Jave, ktoré nám pomohli pri práci s týmto programovacím jazykom. (Schildt, 2012; Roubalová, 2015).

1.3 EXISTUJÚCE APLIKÁCIE

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 Consortium (OGC). (Mapserver, 2023).

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

Súbor KML môžeme vytvoriť pre importovanie aj do Google Earth, kde môže obsahovať rôzne informácie. Informácie, ktoré môžeme 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="WIVV"
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>
        </gx:FlyTo>
        <Camera>
          <longitude>18.09129627</longitude>
          <latitude>48.30826692</latitude>
          <altitude>41.96394348144534</altitude>
          <heading>241.63214</heading>
```

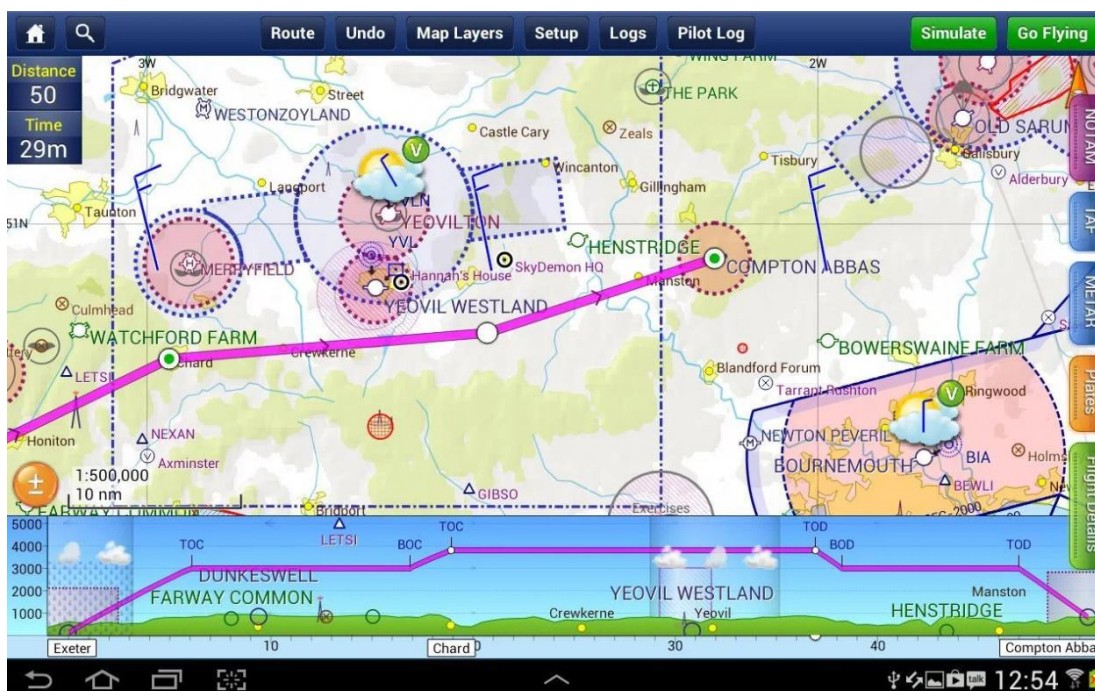
```

<tilt>80</tilt>
<range>10</range>
<altitudeMode>absolute</altitudeMode>
</Camera>
</gx:FlyTo>
</gx:Playlist>
</gx:Tour>
</Document>
</kml>

```

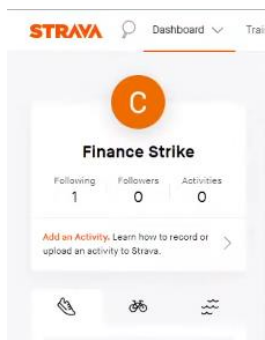
Ako vidieť na príklade, je podobný XML. XML umožňuje vytvoriť ľubovoľnú značku potrebnú na popis a štruktúrovanie údajov. (Microsoft, 2023)

SkyDemon je aplikácia na leteckú navigáciu a neponúka možnosť iných spôsobov transportu. Export súboru zo SkyDemon ponúka napríklad KML, CSV alebo GPX (GPS Exchange Format). Aplikácia tiež zobrazuje nepriaznivé počasie na mape. SkyDemon má vlastné mapy. Aplikácia je spoplatnená na báze mesačného / ročného predplatného. (Divelements Limited, 2009).



Obrázok 2 Aplikácia SkyDemon²

Strava je aplikácia, ktorá je zameraná na športovanie. Ponúka možnosť zaznamenávať trasu pri športoch ako behanie, bicyklovanie a plávanie. Chýba tu však možnosť zaznamenania lietania, výšky a náklonu. Strava neponúka možnosť importu KML súboru. (Strava, Inc., 2009).



Obrázok 3 Aplikácia Strava³

Hiking Project je zameraná hlavne na turizmus. Disponuje vlastnými mapami. Ponúka plánovanie výletov možnosť stiahnuť si chodníky v danej oblasti. Cez kameru ponúka zobrazenie výšky kopcov alebo zobrazenie hviezdnych súhvezdí. Chýba jej generovanie KML súboru a nedisponuje s možnosťou zaznamenávať preletenú trasu. (Hikingproject, 2014).



Obrázok 4 Zobrazenie výšky kopcov Hiking Project⁴

Obrázok 5 Mapa v Hiking Project⁵

³ Zdroj Obrázok 3 <https://play.google.com/store/apps/details?id=com.strava&hl=sk&gl=US>

⁴ Zdroj Obrázok 4 (Hikingproject, 2014)

⁵ Zdroj Obrázok 5 (Hikingproject, 2014)

2 CIELE ZÁVEREČNEJ PRÁCE

Cieľom práce bude zaznamenávanie prejdenej trasy na turistike, počas cestovania (zahŕňa bicykel, kolobežku, auto, autobus a iné dopravné prostriedky) a lietanie lietadlom. Zameriame sa 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 používateľského rozhrania v aplikácii,
- rozvrhnúť rozloženie v sekciách, aby bola jednoduchá na používanie,
- navrhnuť databázu pre záznamy zo senzorov,
- zaznamenávať údaje z mobilného zariadenia (gyroskop, GPS atď.),
- ukladať údaje do databázy,
- umožniť upravovanie niektorých záznamov,
- umožniť mazanie bodov z databázy, celých trás, celej databázy,
- vizualizovať aktuálnu polohu a smer pohybu,
- zobraziť údaje z trasy a body k danej trase,
- zobraziť ukážku prejdenej trasy na mape,
- naprogramovať vygenerovanie súboru pre vizualizáciu prejdenej trasy na rôznych systémoch,
- vyvinúť možnosť vytvorenia súboru na animáciu z dát na iných platformách.

3 NÁVRH A METODIKA

Návrhom riešenia bolo naprogramovať funkčnú aplikáciu. Rozdelili sme si problémy do viacerých kategórií a riešili sme ich postupne. Ako prvé sme si zistili informácie o senzoroch v mobilnom zariadení. Bez zistenia toho, ako fungujú a aké nám poskytujú informácie, sme nemohli spraviť dobrý a funkčný návrh. Typy senzorov a údaje, ktoré reprezentujú, sme si rozpísali. Vybrali sme tie, ktoré sú spojené s cieľom práce. Keď sme už vedeli, s akými senzormi budeme pracovať, navrhli sme databázu na ukladanie informácií zo senzorov. Po uložení dát sme už mali dostatok informácií na návrh používateľského prostredia.

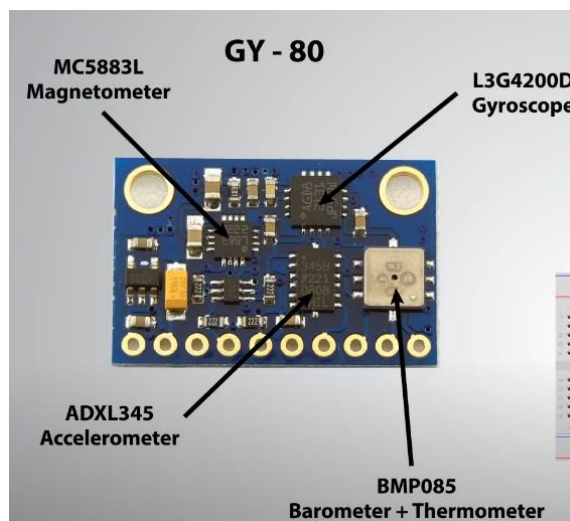
3.1 NÁVRH SOFTVÉRU

Navrhli sme si spôsob zozbierania hodnôt zo senzorov v slučke, ktorá sa bude opakovať. Z hodnôt zo senzorov sme navrhli databázu pre ukladanie informácií do zariadenia. Pokračovali sme návrhom používateľského 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, 2023).
- GPS.

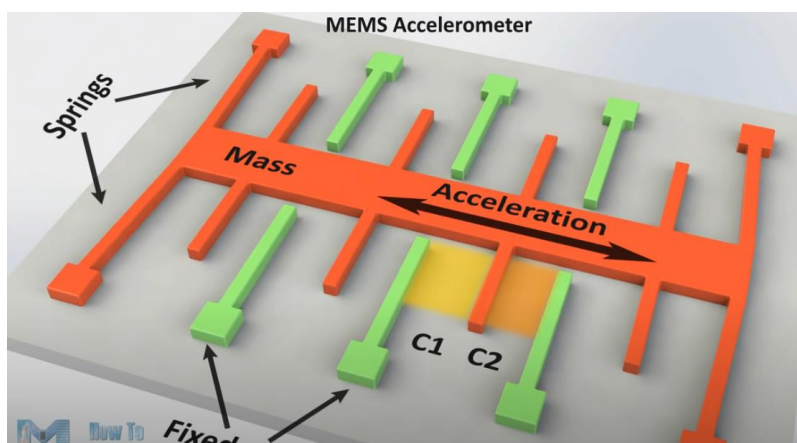


Obrázok 6 Hardvér v smartphonoch.⁶

Akcelerometer nám ukazuje zrýchľovanie. Čím viac zvyšujeme rýchlosť, tým viac sa číslo zväčšuje. Akcelerometer má tri súradnice. Každá súradnica ukazuje smer zrýchľovania. Pri ťahaní zariadenia hore nám na osi ukáže zrýchlenie. Hodnota údajov sa vyjadruje v m/s^2 .

Senzor môžeme použiť napríklad na:

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



Obrázok 7 Akcelerometer v mobile.⁷

6	Zdroj	video	6
	https://www.youtube.com/watch?v=eqZgxR6eRjo&t=60s&ab_channel=HowToMechatronics		
7	Zdroj	video	7
	https://www.youtube.com/watch?v=eqZgxR6eRjo&t=60s&ab_channel=HowToMechatronics		
			20

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

Možné využitie:

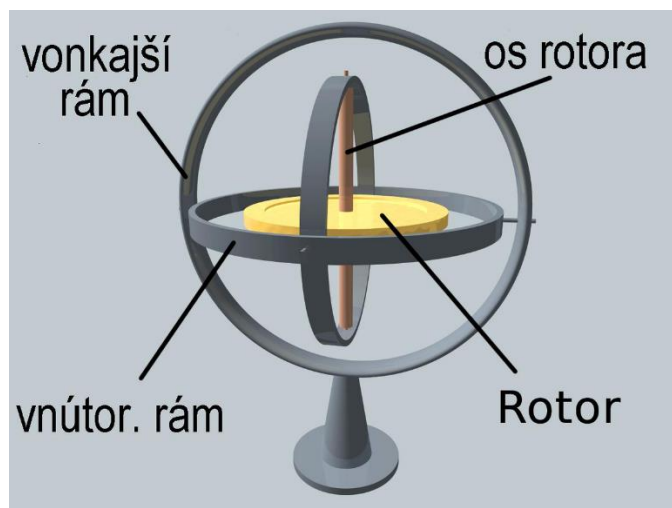
- pri hrách,
- na detekciu pohybu,
- pri Virtuálnej Realite (VR).

Gravity sa používa na získanie hodnôt gravitačného zrýchlenia zariadenia. Tento senzor je softvérový, čo znamená, že získava údaje z akcelerometra a gyroskopu. Gravity má tri hodnoty x, y, z v m/s^2 .

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.

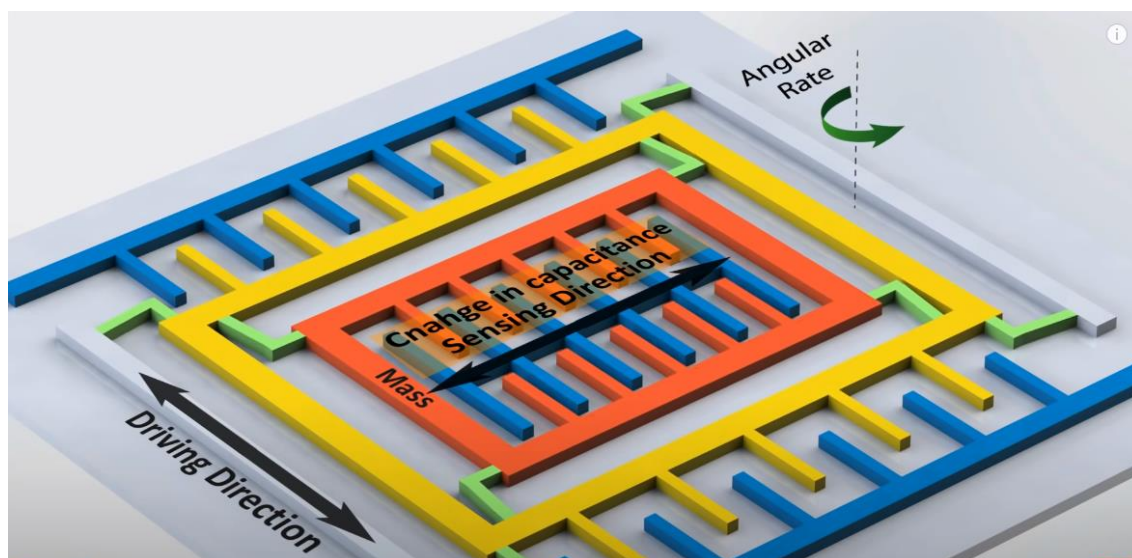
Gyroskop vykonáva v telefóne niekoľko dôležitých funkcií, ktoré používame každý deň bez toho, aby sme si to uvedomovali. Vďaka senzoru je okrem iného možné automaticky meniť orientáciu fotografií v galérii, ktorá sa odohráva v reakcii na pohyb smartphonu. Gyroskopy sú tiež používané vývojármi hier a aplikácií, pretože mnoho programov funguje 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 osiach. Takýto gyroskop označujeme tiež pojmom 3-AXIS. (Hecl, 2014).



Obrázok 8 Gyroskope. Takýto gyroskope sa nepoužíva v mobilných zariadeniach.⁸

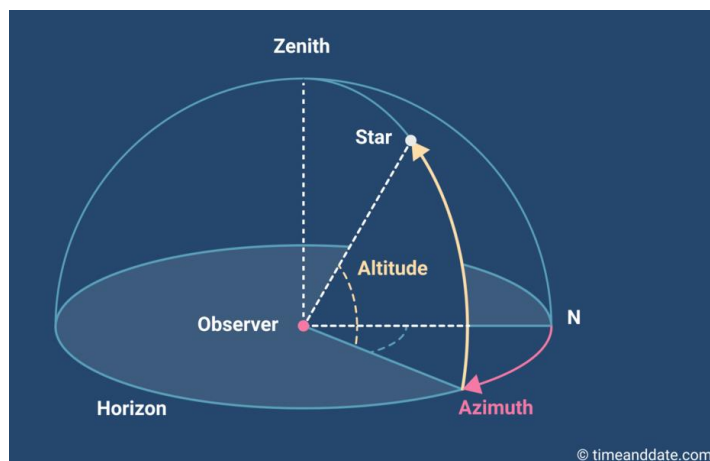
Na snímanie polohy rotora sa nepoužíva sledovanie polohy samotného gyroskopu, čo by bolo príliš zložité a technicky takmer nemožné, no namiesto toho sa sleduje zmena kapacity kondenzátorov, ktoré plnohodnotne zastupujú optické snímače, ako je to pri klasických veľkých gyroskopoch. Tieto kondenzátory 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 9 Gyroskop v smartphone.⁹

8	Zdroj	obrázkov	8
	https://upload.wikimedia.org/wikipedia/commons/thumb/f/f0/3D_Gyroscopesk.png/1200px-3D_Gyroscopesk.png		
9	Zdroj	video	9
	https://www.youtube.com/watch?v=eqZgxR6eRjo&t=60s&ab_channel=HowToMechatronics		
			22

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



Obrázok 10 Ukážka Smeru Azimut.¹⁰

Rotation Vector je konštanta popisujúca typ snímača vektora otáčania. Obsahuje 4 hodnoty x, y, z, w. W predstavuje hodnotu skalárnej zložky vektora rotácie ($\cos(\theta/2)$). Skalárna zložka je voliteľná hodnota. (Google LLC, 2023).

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 samotný GPS.

Všetky navigačné systémy sa v podstate skladajú z troch základných zložiek, ktorými sú kozmická, riadiaco/kontrolná a používateľskej časti. 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

¹⁰ Zdroj obrázok 10 <https://www.timeanddate.com/astronomy/horizontal-coordinate-system.html>

systémov prijat'. 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), Európsky 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 mnoho dní či mesiacov na nový štart satelitu zo Zeme.

Satelity GNSS, či už ide o GPS, GLONASS alebo Galileo, sú komplexným kusom technológie. Ich súčasťou sú stabilizátory, prijímacie a vysielacie antény na rôznych frekvenciách, slúžiace nielen na vysielanie samotného navigačného signálu, ale aj na komunikáciu s pozemnými stanovišťami a na 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 našimi hlavami 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.

GNSS	GPS	GLONASS	BeiDou	Galileo
				
Vlastník	USA	Rusko	Čína	EU
Pokrytie	Celosvetové	Celosvetové	Celosvetové v roku 2019	Celosvetové v roku 2019
Výška	20,1 km	19,1 km	21,1 km	23,2 km
Čas obehu	11 h, 58 min	11 h, 16 min	12 h, 38 min	14 h, 5 min
Satelity	31 v prevádzke 24 globálne minimum	25 v prevádzke 24 globálne minimum	22 v prevádzke 35 do roku 2020	19 v prevádzke 30 do roku 2020
Frekvencie	1,57542 GHz (L1) 1,2276 GHz (L2)	1,602 GHz (SP) 1,246 GHz (SP)	1,561098 GHz (B1) 1,589742 GHz (B1-2) 1,20714 GHz (B2) / 1,26852 GHz (B3)	1,164 – 1,215 GHz (E5a/b) 1,260 – 1,300 GHz (E6) 1,559 – 1,592 GHz (E2- L1-E11)
Presnosť	15 m (bez asistencie)	4,5 m – 7,4 m	10 m (Verejná) 10 cm (Šifrovaná)	1 m (Verejná) 1 cm (Spoplatnená)
Status	v prevádzke	v prevádzke	v čiastočnej prevádzke	v čiastočnej prevádzke

Obrázok 11 Tabuľka GNS.¹¹

Ak chceme navigačné služby používať, potrebujeme k tomu príslušný hardvér. V podstate 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 samostatný čip, ktorý sa nachádza na základnej doske. Môže byť však integrovaný 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.

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

¹¹ Zdroj Obrázok <https://touchit.sk/ako-spozna-smartfon-kde-ste/206763>

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.

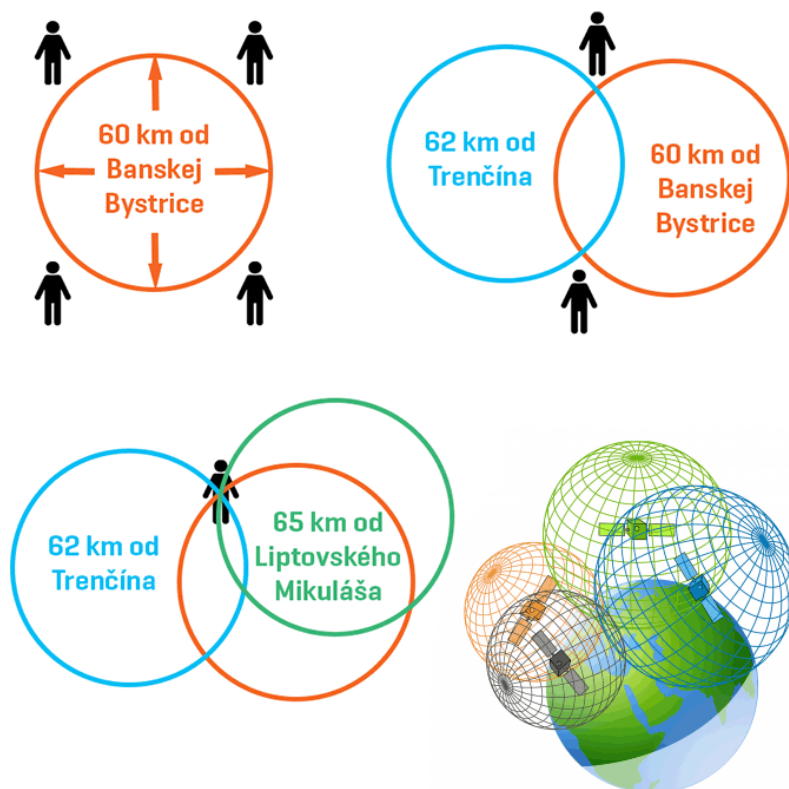
Satelity GNSS o prijímačoch, ktoré ich používajú, „nevedia“ a ani ich polohu nijako nepočítajú. Pri pripojení na internet alebo pri telefonovaní totiž s telefónom komunikuje nielen veža mobilného signálu, alebo Wi-Fi router, ale aj smartfón s nimi komunikuje spätne. Satelity nevedia, aké a koľko zariadení 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 jednom momente. 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. Tento princíp je možné si jednoducho predstaviť v dvojrozmernej podobe. Predstavme si, že stojíme niekde na Slovensku a netušíme kde. Vytiahneme pritom z vrečka prístroj, ktorý by dokázal odmerať vzdialenosť od blízkych miest. Prístroj nám ako prvé zahlási, že sme 60 km od Banskej Bystrice. To je dôležitá informácia, avšak problémom je, že môžeme byť od tohto mesta ktorýmkoľvek smerom, teda napríklad na východ, západ, juh či sever, či niečo medzi tým.

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

Vďaka tomu, že signál vysielajú družice, ktoré sú nad zemou, však trilaterácia nefunguje v 2D, ale 3D priestore. Namiesto kruhovej vzdialenosti od vysieláča tak nestojíme 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 12 Určovanie polohy v 3D priestore.¹²

Ak teda smartfón pozná svoju vzdialenosť od 4 či viacerých satelitov, ktorých poloha je známa (vysielajú ju), vie, kde na Zemi sa nachádza. Ako ale túto 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

¹² Zdroj Obrázok 12 <https://touchit.sk/ako-spozna-smartfon-kde-ste/206763>

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 oneskoreniu 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 v 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 otočenie 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 potrebujeme merať čas na nanosekundy, vďaka čomu budú rozdiely o vzdialenosti len niekoľko metrov prípadne centimetrov.

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.

Niekedy sa môžeme zarazit' nad tým, že nejaká prenosná alebo integrovaná auto navigácia či tablet zisťuje pri prvotnom zapnutí našu polohu značne pomalšie, ako náš smartfón. Ak stojíme na mieste, rozdiel môže byť len pár sekúnd, no 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 ju už aktualizuje pri vašom pohybe rýchlo.

Je to dôsledok mechanizmu, ktorý sa označuje ako A-GPS, teda Asistovaný GPS. Jeho princípom je, že pri zisťovaní polohy asistuje smartfónu vysielateľ mobilnej siete, ku ktorému je v rámci svojej SIM karty pripojený. V podstate ide o to, že server mobilného operátora do nášho telefónu pošle už vypočítané navigačné dáta v súvislosti s polohou pozemnej vysielacej veže a zároveň aj informácie o satelitoch, ktoré sú práve nad nami. Aj keď sa telefón z nich nedozvie, kde presne sa nachádza, 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 čakanie na kompletné informačné dáta vysielané satelitmi rýchlosťou 50 bit/s. To chvíľu trvá a navyše, ak pritom zjídeme za prekážku a signál sa stratí, zariadenie musí začať znova od nuly. Čím viac signálov z rozdielnych satelitov zariadenie prijíma, tým lepšie, avšak ak sme 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 sme sa otázkami snažili prísť na to, na aké číslo od 1 do 1000 niekto myslí, pričom nám pomôže tým, že je medzi 720 a 730. O to rýchlejšie sa nám tak podarí dané číslo uhádnuť.

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 to od operátora. Tak či onak, na našom účte a dátovom limite sa to nijako neprejaví, pretože ide o textové dáta, ktoré sú skutočne miniatúrne.

Ak používame 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 nášho smartfónu vyberieme, alebo sa nachádzame mimo pokrytia mobilnej siete. Smartfón je plne schopný vypočítať polohu sám zo signálu satelitov. Je to však na úkor času a taktiež z dôvodu náročnosti výkonu aj na úkor výdrže na batérie.

GNSS nám pri navigovaní v aute ukazuje trochu inú rýchlosť, ako náš tachometer. Čo je presnejšie? Možno je to prekvapujúce, ale aj napriek tomu, že napríklad GPS má v praxi presnosť len okolo 3 až 5 m, z hľadiska rýchlosti je počítanie mimoriadne presné. Zmena našej 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), sa skladá 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.

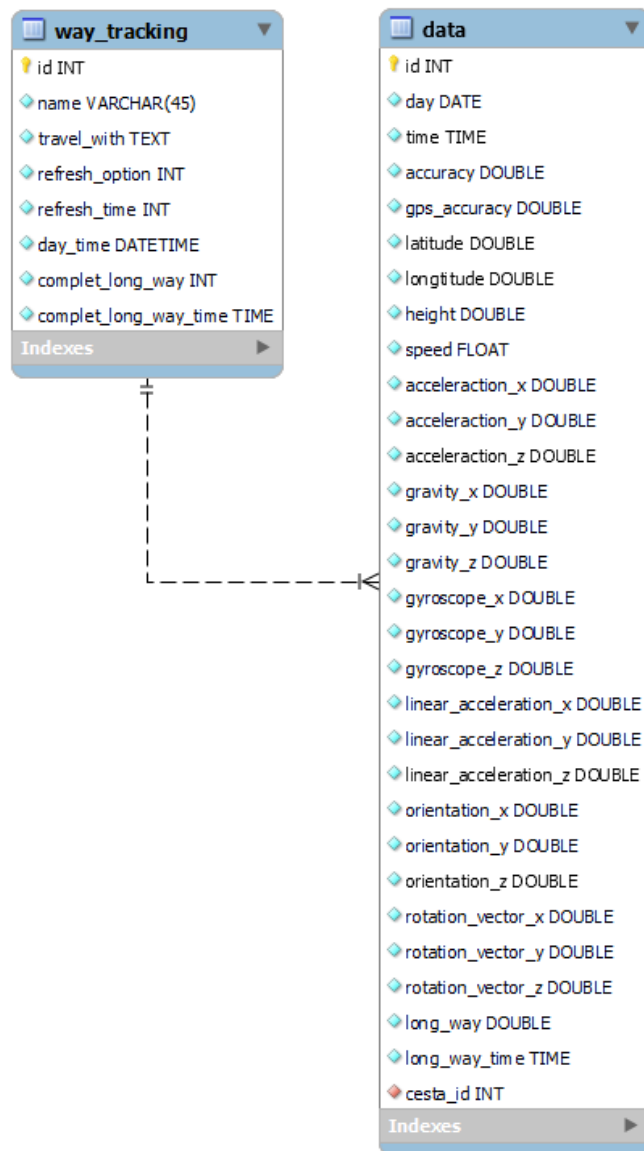
Obrovský pokrok v rámci presnosti prináša európsky Galileo, ktorého rubídiové atómové hodiny sú schopné presnosti na 1,8 nanosekundy za 12 hodín, čo umožňuje dosiahnuť presnosť na 1 m. Vysielanie na základe týchto hodín je dostupné všetkým, čo znamená, že používaním systému Galileo vo svojom smartfóne si z hľadiska presnosti výrazne polepšíme (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 vlastnime relatívne nový telefón, môžeme 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 podstate 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.

V konečnom dôsledku je teda obrovským pozitívom fakt, 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 sme sa rozhodli pre dve tabuľky. Prvá tabuľka obsahuje záznamy a sumár údajov. Druhá tabuľka zahŕňa informácie z každého uloženého bodu záznamu. Tabuľky sú prepojené 1 ku N. Databáza je zhotovená podľa tretej normálovej formy.



Obrázok 13 Návrh Databázy SQL LITE

Databáza je použitá na ukladanie údajov zo senzorov zariadenia, doménový model v PRÍLOHA B1.

Návrh funkcií aplikácie

K ceste je potrebné vytvoriť id cesty, aby sme mali k čomu priradovať body z cesty. Aplikácia aby niečo robila, budú potrebné niektoré funkcie:

- vytvorenie nastavenia, pre frekvenciu ukladania údajov,
- vytvorenie id pre cestu, pre frekvenciu ukladania úsekov,
- zbieranie údajov zo senzorov,
- vkladanie dát zo senzorov do databázy,
- upravovanie niektorých informácií v databáze (napríklad názov cesty),

- mazanie cesty, mazanie bodov z cesty v databáze,
- vymazanie celej databázy,
- čítanie informácií z databázy o ceste a bodoch cesty,
- zobrazenie cesty a záchytných bodoch z cesty,
- vytvorenie súboru podľa údajov z databázy na zobrazenie prejdenej trasy,
- zobrazenie polohy na mape,
- zobrazenie prejdenej trasy,
- výpis všetkých ciest.

Diagram prípad použitia v PRÍLOHA B2.

Návrh používateľského prostredia

Používateľské prostredie bude mať 7 obrazoviek: úvodná, zber dát, zobrazenie aktuálnej pozície, nastavenia, história, zobrazenie údajov o ceste, zobrazenie zaznamenanej trasy.

Úvodná obrazovka bude obsahovať:

- názov aplikácie, krátky popis,
- tlačidlo História - na tomto mieste budú zobrazené trasy,
- nastavenie aplikácie,
- možnosť výberu cestovného prostriedku a to potvrdenie tlačidlom Štart.

Zber dát bude obsahovať možnosť zrušenia cesty, dokončenia cesty, zobrazenia aktuálnej polohy a vrátenia sa späť na domovskú obrazovku. Táto obrazovka bude zobrazovať údaje:

- dátum,
- čas,
- GPS(x, y, z, rýchlosť),
- dĺžka cesty,
- čas cestovania,
- senzory (akcelerometer, gravity, gyroskop, lineár akcelerometer, orientantion, rotation vector). Výpis z týchto senzorov je formátovaný na 3 desatinné miesta, každý senzor má minimálne tri výstupné hodnoty,
- senzor presnosť polohy a presnosť GPS, ktorú môže dosiahnuť za normálnych okolností.

Nastavenia disponujú možnosťami výberu predpripravených a vlastných hodnôt. Používateľ si tu môže zvoliť rýchlosť zaznamenávania bodov počas cesty. Ak je úložisko

zariadenia malé, odporúča sa zvoliť nižšiu frekvenciu ukladania. Každý bod zaberá miesto s informáciami v zariadení. Z nastavení bude možnosť vrátiť sa späť na úvodnú obrazovku.

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

Zobrazenie údajov o ceste vypíše údaje o ceste, o každom bode záznamu z cesty. Pri podržaní prsta 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 celú cestu,
- vygenerovať súbor KML prejdenej trasy,
- vygenerovať súbor KML pre vizualizáciu trasy používateľa,
- prejsť na mapu a ukázať trasu po ktorej sme sa hýbali.

3.2 METODIKA VÝVOJA SOFTVÉRU

Po návrhu sme si museli premyslieť, kde presne začať, pretože pri nesprávnom rozložení krokov by nebolo možné softvér testovať po jednotlivých fázach a bolo by nutné sa vracieť a prerábať osobitné kroky.

Úvodná obrazovka

Z dôvodu dobrého rozloženia práce sme začali s úvodnou obrazovkou. Tu sme si spravili ako prvú vizuálnu časť. Popridávali sme komponenty, texty a zarovnali komponenty do prehľadného stavu.

Nastavenia

Po rozložení šablóny na úvodnej obrazovke sme sa zamysleli nad ďalším postupom. História 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 používateľ podľa toho, čo bude potrebovať, a preto bolo potrebné vytvoriť nastavenia pre používateľa ako druhý krok. Opäť sme spravili rozloženie nastavenia a prepojili to s hlavnou obrazovkou.

Zbieranie dát

Po príprave šablóny a prepojení s úvodnou obrazovkou sme chceli získať informácie z SharedPreferences. Naprogramovali sme senzory dynamicky aby sme ich vedeli pridávať a odoberať. Na čítanie dát zo senzorov použijeme cyklus a ukladanie údajov do tabuľky podľa nastavení používateľa. Celá aplikácia je závislá na prístupe k GPS a tak musíme pred zberom dát vyžiadať od používateľa prístup a kontrolovať ho po celý čas, či nie je vypnutý.

Databáza

V databáze sme sa inšpirovali z učiva, ktoré sme preberali na vysokej škole. Funkcie sú základné: vkladanie, čítanie, upravovanie, mazanie. Naprogramovali sme hlavne základ (vložiť, upraviť, zmazať, jednoduché čítanie z databázy). Čítanie z databázy sa rozšíri podľa potreby zobrazovania v nasledujúcich obrazovkách aplikácie.

Vizualizácia aktuálnej polohy na mape v aplikácii

Vizualizácia bola navrhnutá do máp od Google priamo v aplikácii. Priamo v aplikácii si môžete pozrieť aktuálnu polohu. Pri pohybe sa vám bude znázorňovať, aj ktorým smerom sa pohybujete. Znázornenie polohy dokáže aplikácia vďaka tomu, že to porovnáva s predošlou polohou, a tak vám vektor ukáže smer, v ktorom sa pohybujete.

História

V tejto časti sa zobrazia už zaznamenané cesty. Ako názov sme pre tieto cesty vymysleli formát „rok-mesiac-deň hh:mm:ss“ s možnosťou premenovať tieto názvy. V tejto časti sa nachádza aj voľba Vyčistiť celú databázu.

Zobrazenie informácií z databázy o ceste

Tu je možné vyčítať údaje o ceste a bodoch, všetky tieto informácie sa dajú zobrazit'. Taktiež je k dispozícii stiahnutie záznamu trasy exportom do KML súboru.

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

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

4 VÝSLEDKY

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

4.1 IMPLEMENTÁCIA

Po návrhu aplikácie, databázy, výzoru a funkčnosti môžeme začať s tvorbou aplikácie. Pre testovacie účely sme použili Android telefón na testovanie. Postup implementácie obrazoviek a tried v Android Studiu:

- úvodná obrazovka,
- nastavenia,
- zber údajov zo senzorov a GPS, vytvorenie databázy,
- zobrazenie aktuálnej pozície,
- história,
- zobrazenie údajov z databázy,
- vizualizácia zaznamenananej trasy,
- export súboru KML so zaznamenanou trasou,
- export súboru KML s animáciou trasy.

Postupne sme si ladili výzor a funkčnosť aplikácie. Každý krok, ktorý sme naprogramovali, sme aj hneď otestovali. Upravovali sme výzor a rozloženie, a snažili sme sa ich čo najviac sprehľadniť.

Pri programovaní aplikácie sme natrafili aj na náročné úlohy, ktoré boli nahradené iným spôsobom riešenia, ako napríklad Google Earth bol nahradený exportom súborov KML, ktorý je vysvetlený v našej práci. Nahradenie bolo nutné z dôvodu, že Google Earth sa môže programovať len po API 25 a čase písania tejto práce sa používajú API 33. Súbor KML môžeme importovať do Google Earth, a takto sa nám podarilo zobrazenie presunúť z aplikácie na web a vizualizovať zaznamenanú trasu.

Výsledná aplikácia a prechody medzi obrazovkami sú znázornené v prílohe

Používateľské rozhranie PRÍLOHA B3.

Po naprogramovaní prišlo reálne testovanie a zaznamenanie prvej trasy autom. Zobrazenie v Google mapách sa ukázalo ako nie presné. Na mape aplikácia ukazovala,

že sme leteli vzduchom cca 240m nad zemou. Tento problém bol spôsobený tým, že sme boli v danej nadmorskej výške reálne. Keď sme túto nadmorskú výšku vložili do Google mapy, zistili sme, že tam nie je presná vizualizácia nadmorskej výšky, a museli sme to vypočítať tak, aby to sedelo s Google mapami. Problém sa však dal vyriešiť, lebo Google mapy majú v sebe zabudovanú funkciu, kde môžeme zadať rôzne parametre, na základe ktorých nám Google zistí danú informáciu. Napríklad ak zadáme zemepisnú šírku a dĺžku, môžeme sa opýtať, aká je nadmorská výška na tejto pozícii. Po odrátaní rozdielu našej nameranej výšky a výšky z Google máp sme sa dostali na výšku 40 metrov z 200m. Na upresnenie tejto odchýlky (40m) bude potrebných viac zariadení na testovanie, aby sme mohli daný problém vyladiť. Bohužiaľ sa nám nepodarilo zohnať viac zariadení na testovanie.

Ďalšie prekážky nastali, keď sme chceli použiť rotáciu smerom hore a dole z leteckého záznamu. Tento problém sme vyriešili tak, že pri postupe spúšťania aplikácie sme ako prvé položili telefón do pozície, v akej potom bol po celý čas, a následne sme stlačili Štart. Aplikácia si prvú hodnotu pri štarte vždy uloží a od tejto hodnoty bude počítat ostatné hodnoty.

Nastavenie

V nastaveniach po zostavení vizualizácie sme začali programovať nastavenia. Na ukladanie informácií z nastavení sme sa rozhodli pre využitie vstavanej funkcie v Android Studio s názvom `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 naše 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 (je skratka pre "enumeration" a predstavuje dátový typ v niektorých programovacích jazykoch, ktorý slúži na definovanie množiny predefinovaných konštánt s priradenými hodnotami), 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 keďže bolo viac stringov v enume, museli sme tento problém vyriešiť, na čo sme použili stránku stackoverflow. Našli sme toto riešenie problému:

```
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).
```

Uvedený kód sme pretransformovali na vyriešenie nášho 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];
    }

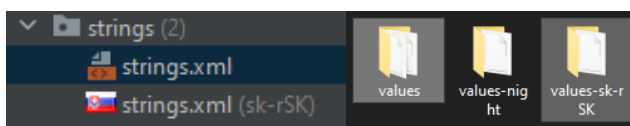
    public int length() {
        return gNameKey.length;
    }
}
```

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

Nastavenie podľa preferencií v telefóne

Na využívanie viacjazyčnosti sme v súbore strings.xml použili editor zabudovaný v Android Studiu a pridali sme jazyk slovenčina zo svetových jazykov. Pribudol nám nový stĺpec v jazykoch a vytvoril sa nám nový súbor strings.xml, ktorý je začlenený

v inom priečinku, a preto môže mať rovnaký názov a vizualizuje sa nám s rovnakým menom.



Obrázok 14 Ukážka súborov prekladu textu v aplikácii Android Studio.

Obrázok 15 Ukážka súborov zatriedených v priečinkoch.

Ako vidieť na obrázkoch, v aplikácii sa zobrazujú v jednom priečinku, ale na disku sa nachádzajú v rozdielnych priečinkoch.

Na dopísanie prekladu sa nám ponúkajú dve možnosti. Je to buď priamo do súboru cez textový editor alebo priamo v Android Studiu, kde sa nám farebne odlíši text a značkovací jazyk. Nevýhodou tejto možnosti je písanie textu v značkovacom jazyku. Druhou možnosťou je použitie editora jazykov v Android Studiu, kde nám prvý stĺpec predstavuje kľúč, podľa ktorého budeme volať preklad, a každý jeden jazyk predstavuje jeden stĺpec bez značkovacieho jazyka. Anglický jazyk je nastavený ako predvolený pre prípad, že aplikácia neobsahuje preferovaný jazyk.

Android Studio má už pripravenú funkciu na volanie týchto prekladov. Stačí na to použiť napríklad príkaz:

```
R.string.deleted_all
```

Aplikácia už doplní za náš príkaz text zo súboru strings.xml.

Zbieranie dát

Pri testovaní sme narazili na problém, že ak používateľ od inštalácie nikdy nič nenastavil v nastaveniach aplikácie, aplikácia nevedela, čo má nastaviť a nefungovala, ako sme chceli. Preto sme pridali premennú `nFirstSet`, ktorá reprezentuje to, či bola niekedy nastavená hodnota v nastaveniach. Ak nebola, priradili sme jej predvolenú hodnotu, ako vidieť v kóde na GITE.

Ako sme sa už zmienili v metodike, pri programovaní senzorov sme sa čo najviac snažili držať dynamického programovania. Dynamické programovanie nám zjednodušuje 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 vyvolaná, keď sa údaj v senzore zmení. Pomocou nej sa len opýtame, 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 vyžiadanie si povolenia k GPS a následná aktualizácia 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 ktorej sme si vyžiadali prístup k GPS (kód nižšie). 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 sme použili túto funkciu a vložili sme ju do časovača, aby sme vždy vyžiadali nové informácie podľa času. Toto však nefungovalo, a tak sme museli vyriešiť tento problém aktualizáciou GPS, a následne boli informácie uložené a zobrazené.

```

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).

```

Táto funkcia nám vyriešila problém s aktualizáciou GPS. Na získanie rýchlosti ktorú sme si vysvetlili vyššie, sme použili výpočet:

```

//speed
double speed = location.getSpeed() * 3.6f;
temp = String.format("%.2f km/h", speed);
goTempSpeed.setText(temp);
(Amjad, 2016).

```

V neposlednom rade je potrebné predstaviť aj časovač k zberu údajov. Na zber údajov sme vytvorili časovač, ktorý sa nám opakuje v intervale 1000 milisekúnd. To znamená, že každú sekundu sa spustí 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, 2023).

Ako vidíme v kóde, porovnáme čas z nastavenia a keď sa rovnajú alebo je gnRefreshTime väčší, dáta sú vložené do databázy cez addSensorDataToDB. Tu narazíme na to, že databázu ešte nemáme naprogramovanú, a preto sme túto funkciu zatiaľ nechali prázdnu. Vrátime sa k nej, keď naprogramujeme databázu. V ďalšej fáze dopíšeme funkciu, keď budeme vkladať dáta do databázy.

Zobrazenie údajov o ceste

Pri vizualizácii všetkých bodov sme museli pridať text k údajom, aby používateľ vedel, o aký údaj sa jedná. To bolo náročnejšie, ako sa spočiatku zdalo, pretože sa body

zobrazujú cez listview priamo z databázy. Pri riešení tohto problému sme museli ku každému údaju vložiť správny popis. Pomohli sme si na fóre stackoverflow (Ganguly, 2015). Kde sme našli návrh, ako takýto listview dynamicky upravovať.

Vizualizácia aktuálnej polohy na mape v aplikácii

Google Earth sa dá použiť len v API po verziu 25. Náš softvér ma základ API 33, a preto sme sa museli uchýliť k inej platforme. Ako náhradnú platformu sme si vybrali Google mapy. Ak chceme využívať niektoré funkcie v Google mapách, musíme si vyžiadať unikátny kód, ktorý je potrebné vložiť do programu. Z dôvodu, že tento kód nemôže byť zverejnený, je prepísaný aj v kóde na GITE a nie je zverejnený funkčný kľúč. Na získanie návodu, ako si tento kľúč vyžiadať, sme sa inšpirovali videami 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 pre možnosť náklonu. Po naprogramovaní náklonu sa ukázalo, že hodnoty sú opačné. Zariadenie pri náklone vľavo zobrazovalo náklon vpravo, a preto sme hodnoty otočili tak, že sme dali pred premennú znamienko mínus. Tým sa nám problém vyriešil.

Zobrazenie informácií z databázy o ceste

Tu len načítame údaje z databázy a všetky ich zobrazíme v obrazovke s názvom ShowData. V tomto bode procesu nám padla aplikácia. Pri alfa testovaní sme zistili, že komplikácia nastala, keď sme chceli vyčítať cesty a body, no daná cesta žiadny bod cesty neobsahovala. 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 a 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 zaznamenananej trasy.

Údaje do súboru sme si vytiahli z databázy, vložili sme si ich do poľa a pole sme poslali do triedy na vytvorenie súboru. Pred odoslaním poľa sme si ešte dáta spracovali na výpočet výšky. Ak je zaznamenaná trasa pre lietadlo, spracovávame výšku. V prípade, že ide o iný dopravný prostriedok, posielame automaticky výšku nula a ani si nepýtame nadmorskú výšku z Google máp. Táto funkcia na získanie údajov o nadmorskej výške najviac spomaľuje generovanie súboru KML. Zozbierali sme si údaje o zemepisnej šírke a dĺžke a vypýtali sme si z Google máp nadmorskú výšku na danej polohe. Existuje aj funkcia, v ktorej si môžeme zadať všetky body a Android Studio nám odpovie hodnotami v poli. Táto funkcia nám zvýšila rýchlosť aplikácie - keby mala aplikácia čakať po každom bode na odpoveď, bola by extrémne 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 dostať 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 napájať všetky systémy do triedy na vytvorenie súboru. Vyžadovalo by to aj prístup k polohe, kde by sme museli robiť ďalšie zabezpečenia. Z triedy, kde sme pole vytvorili, bolo už všetko zabezpečené. Nasledovalo vytvorenie súboru pre animácie a len sme rozšírili už funkčnú triedu, kde sme mali už dobrý základ na rozšírenie.

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

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

4.2 POPIS VYTVORENÉHO SOFTVÉRU

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

V aplikácii máme 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 zo svojich chýb, a preto história trasy môže poskytnúť dobrý náhľad do prejdenej trasy s cieľom vlastného zdokonaľovania.

4.3 TESTOVANIE

Programátor aplikáciu naprogramuje a tester sa snaží aplikáciu pokaziť. Takmer vždy sa nájde niečo, na čo sa zabudlo, alebo možnosť, s ktorou sa nepočítalo, či skutočnosť, že jedna funkcia môže ovplyvniť druhú funkciu. Pri testovaní je potrebné skúšať všetky možnosti, ako reagujú iné funkcie, zadávať nulu, záporné, kladné, prehnané čísla, neplatné znaky.

Alfa testovanie prebiehalo počas vývoja. Pri testovaní bola testovaná vizáž a funkčnosť. Boli zistené chyby, ako napríklad text, ktorý pretekal a rozhádzal celú obrazovku. Zlyhanie aplikácie nastalo pri chybách v programovaní. Ako príklad uvedieme nasledovné: aplikácia si vypýtala údaje z databázy, no v databáze nič nebolo, a tak poslala výsledok null, avšak program s takýmto výsledkom nepočítal a ukončil sa. Pri alfa testovaní bol objavený ďalší problém. Ak obrazovka zhasne, neukladá žiadne údaje zo senzorov. Tento problém sme vyriešili tak, že po štarte sa obrazovka nevypne, až pokiaľ nebude ukončená aplikácia alebo zrušená trasa.

Po funkčnej aplikácii a zozbieraní niekoľkých dát prišlo na beta testovanie. Tu sme zvolili metódu testovania „pokaz, čo sa dá“.

Ako prvé sme zvolili testovanie zadávania znakov, čísel, písmen. Po neúspechu zhodiť aplikáciu sme sa pokúsili vložiť záporné číslo alebo nulu, a opäť sme sa stretli s neúspechom. Skúsili sme prekročiť povolené hodnoty, no nič aplikáciu nepokazilo. Vyskúšali sme všetky možnosti v nastaveniach s funkčnosťou aplikácie. Aplikácia ukázala jednosekundový rozdiel vo frekvencii ukladania údajov, a tým sme prvýkrát uspeli v hľadaní chýb aplikácie. Chybu sme opravili z pôvodného kódu:

```
finalSaveDataTimer < ++gnRefreshTime
```

na opravený kód:

```
finalSaveDataTimer <= ++gnRefreshTime.
```

Ďalšou fázou bolo testovanie senzorov a spôsobov cestovania, aby sme mali k dispozícii dáta na testovanie pri ďalších funkciách. Vybrali sme si prvú možnosť cestovania „chôdza“ a začali sme robiť v aplikácii veci, ktoré sa robiť nemajú - zakázali sme jej napríklad zaznamenávať polohu počas ukladania údajov. Aplikácia si opätovne vyžiadala prístup k polohe. Skúsili sme najskôr vypnúť v nastaveniach povolenie polohy a zvoliť si druhú možnosť cestovania v aplikácii. Stlačili sme Štart a nechali ju chvíľku stáť bez údajov a potom sme zamietli povolenie polohy. Aplikácia si opätovne vyžiadala

prístup a všetko fungovalo. Pokúsili sme sa rýchlo spustiť štart a hneď trasu ukončiť, aby sa nevytvoril žiadny bod zo senzorov. Aplikácii to neuškodilo. Vyskúšali sme si režim lietadla a s telefónom sme pohybovali do všetkých strán a skúšali, či nenastane chyba kvôli nežiadanej hodnote. Aplikácia to však zvládla.

Pokračovali sme ďalším testovaním. Vymazali sme databázu a vyskúšali sme, či nám aplikácia nepadne, ak dostane z databázy odpoveď null. Túto chybu sme odstránili už pri alfa testovaní, ale pri beta testovaní sme si ju zopakovali, aby sme zistili, či to neovplyvní aj niečo iné.

Ďalší krok bolo otestovanie zobrazenia údajov, a tak sme sa snažili vytvoriť cestu bez jediného bodu zo senzorov. Aplikácia to zvládla (táto chyba bola tiež odstránená počas alfa testovania). Pokračovali sme generovaním súborov bez údajov. Aplikácia to prežila, no nastal tu problém s povolením ukladať súbor, ak neboli udelené práva, a my sme ich potvrdili. Aplikácia súbor ani opätovne neuložila, a tým pádom sme uspeli druhýkrát pri testovaní. Problém sme vyriešili tým, že si overujeme povolenie ukladať súbory hneď na úvodnej obrazovke a taktiež práva overujeme pri vstupe na obrazovku, kde je funkcia na ukladanie súborov. V prípade, že pri overení nebol udelený súhlas, si ho vyžiadame od používateľa. Aplikácia taktiež zvládla premenovanie cesty alebo zmazanie. Nasledovalo testovanie vizualizácie mapy bez údajov - všetko aplikácia vydržala. To znamená, že sme neobjavili ďalšie problémy s aplikáciou.

Počas testovania aplikácie sa našli chyby, ktoré boli odstránene alebo čiastočne vyriešené. Prípadné ďalšie skryté chyby, ktoré sa nám nepodarilo testovaním odhaliť, môžu byť odhalené šikovnými používateľmi.

ZÁVER

Na vývoj softvéru sme sa snažili používať už existujúce pripravené technológie. Softvér sa podarilo úspešne dokončiť. Aplikácia nepadá a je stabilná. Cieľ záverečnej práce bol vizualizovať zaznamenanú trasu. Cieľ bol dosiahnutý – ako v aplikácii, tak aj možnosťou exportu súboru.

Aplikácia je vhodná pre používateľov, ktorí by radi názorne videli, ako a kadiaľ počas svojej cesty šli, prípadne pre tých, ktorí majú záujem zlepšovať svoje trasy či letecké manévry.

Aplikácia navyše ponúka možnosť prispôbiť sa používateľovým preferovaným nastaveniam v telefóne, ako je biele alebo čierne pozadie. Ďalej ponúka preklad do angličtiny, ktorá je základným jazykom, no v prípade, že máte v telefóne nastavenú slovenčinu, jazyk aplikácie sa automaticky zmení na slovenský.

Aplikáciu sa podarilo prepojiť Google Earth. Tento problém sme vyriešili exportom súboru KML, ktorý je možné vložiť na stránke do Google Earth.

Napokon sa nám podarilo aj vizualizovať náklon do strán. Túto možnosť náklonu sme použili pre všetky druhy zaznamenávania, vďaka čomu je možné sledovať náklon aj pri chôdzi alebo terén, po ktorom sa prechádzalo.

Aplikácia taktiež obsahuje dáta, ktoré sa v budúcnosti budú môcť doprogramovať a využiť, čo používateľa môže milo prekvapiť, napríklad využitie noviniek a aktualizácií zobrazenia trás, ktoré už dávnejšie zaznamenal. Taktiež ponúkame používateľovi súkromie, keďže dáta neodosielame na server, kde k nim môže mať prístup niekoľko spoločností či iní ľudia.

Vytváranie súboru KML zo záznamu lietania trvá dlhšie. Je to spôsobené tým, že aplikácia si vyžaduje výšku polohy z Google máp a pre každý zaznamenaný bod čaká na odpoveď zo servera. To môže používateľovi pripadať tak, akoby aplikácia zamrzla. V skutočnosti je tam však tak veľa bodov, že aplikácia musí čakať na odpoveď z Google máp a používateľ musí jednoducho aplikáciu nechať pracovať a prepočítavať výšku, ktorú napokon zobrazí.

Aplikáciu na záver hodnotíme pozitívne a užitočne. Pevne dúfame, že bude používateľom dobre slúžiť. Je používateľsky jednoduchá - používateľ by sa mal v nej dokázať zorientovať do niekoľkých minút po nainštalovaní.

ZOZNAM BIBLIOGRAFICKÝCH ODKAZOV

- 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. 2023. android.com. [Online] 2023. [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. 2023. javatpoint.com. [Online] 2023. [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. 2023. mapserver.org. [Online] 2023. [Dátum: 31. 3 2023.] <https://mapserver.org/el/input/vector/kml.html>.
- Microsoft. 2023. microsoft.com. [Online] 2023. [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. 2023. msgprogramator.sk. [Online] 2023. [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 – Aplikácia + Zdrojový kód + odkaz na GIT

Príloha B – Fotodokumentácia + odkaz na GIT

PRÍLOHA A

Príloha A

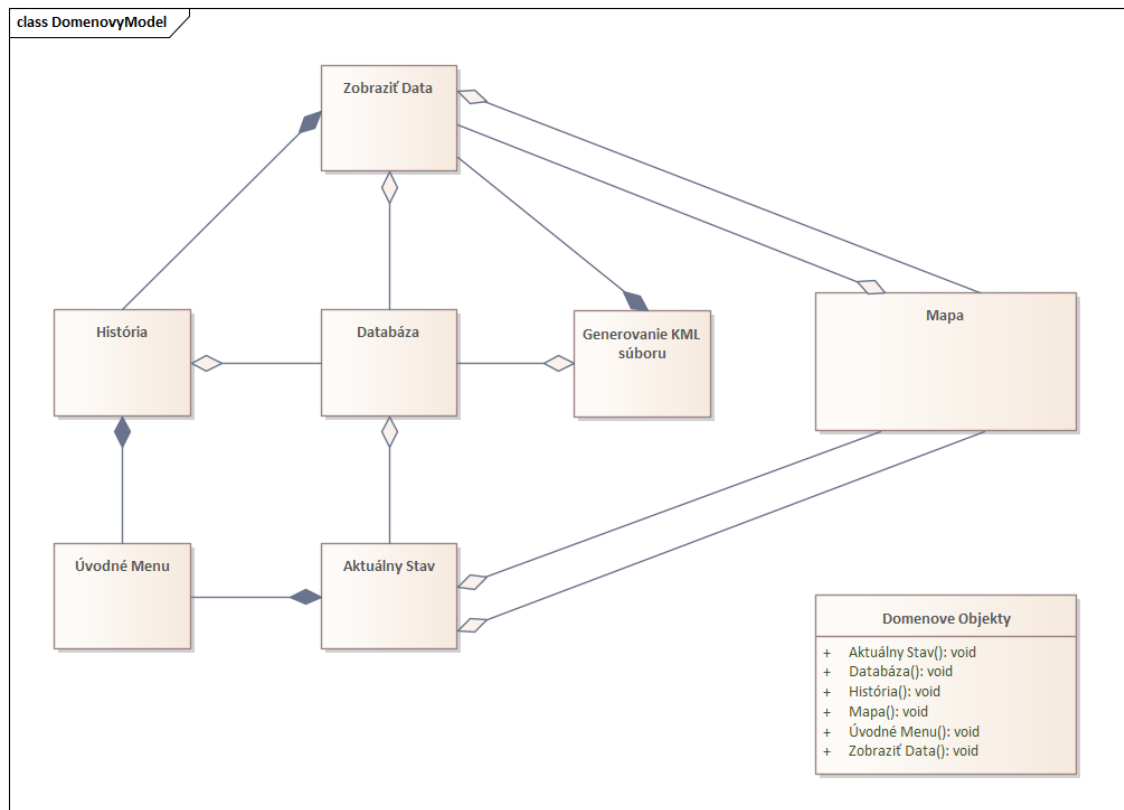
- A1 Odkaz na GIT: https://github.com/gameblaer/zaverecna_praca_bc/
- A2 Zdrojový kód odkaz na GIT:
https://github.com/gameblaer/zaverecna_praca_bc/blob/main/Aplikacia/WIW.zip
- A3 Aplikáciu do Androidu na GIT:
https://github.com/gameblaer/zaverecna_praca_bc/blob/main/Aplikacia/WIW_Google_Earth/app/build/intermediates/apk/debug/app-debug.apk

PRÍLOHA B

Príloha B

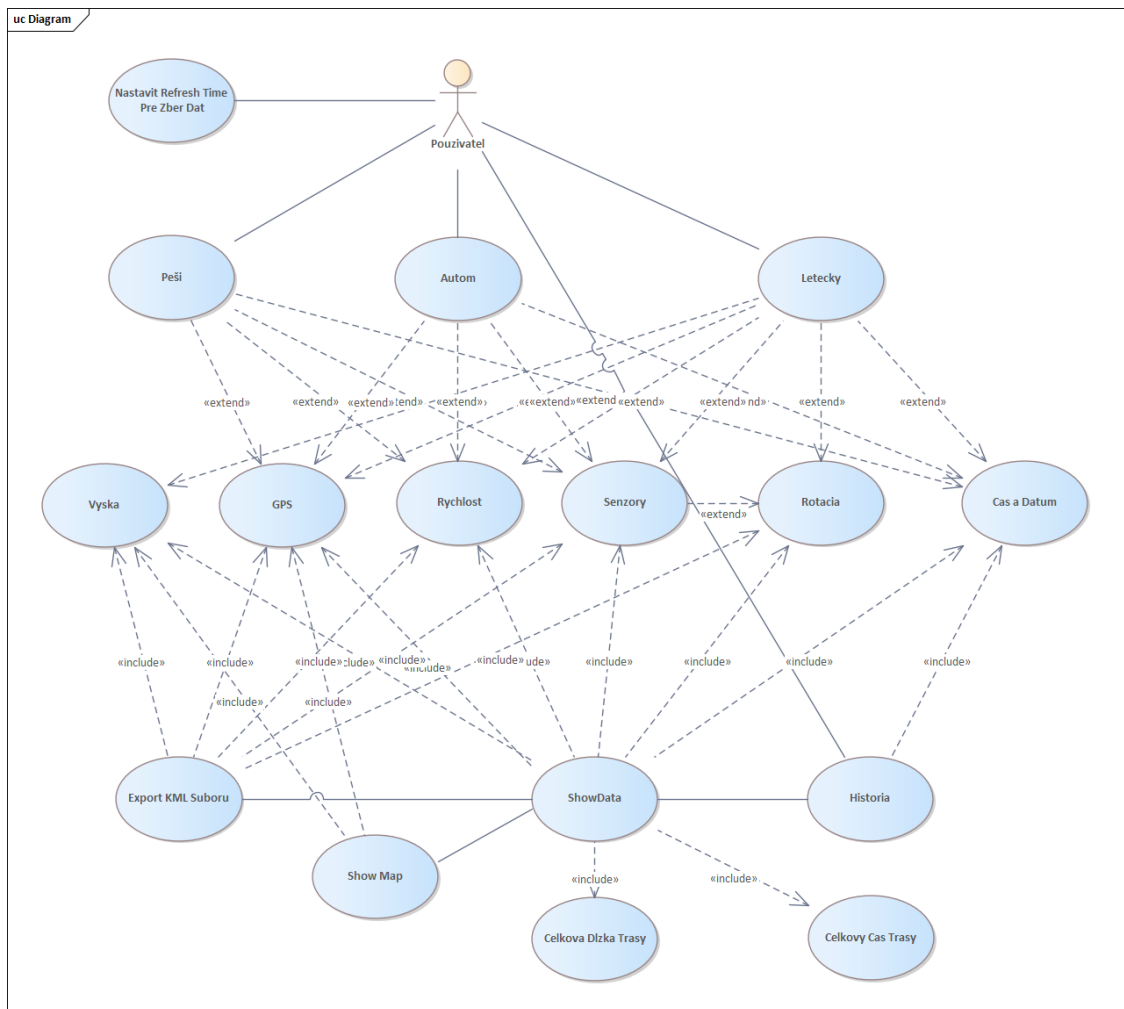
- **B1** Odkaz na obrázok v GIT:

https://raw.githubusercontent.com/gameblaer/zaverecna_praca_bc/main/Images/DomenovyModel.png



- **B2** Odkaz na obrázok v GIT:

https://raw.githubusercontent.com/gameblaer/zaverecna_praca_bc/main/Images/Diagram.png



- **B3** Odkaz na obrázok v GIT:

https://raw.githubusercontent.com/gameblaer/zaverecna_praca_bc/main/Images/B3.png

