ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra měření

Bakalárska práca

Uživatelská aplikace pro detektor akustické události

Peter Suchý
Otevřená informatika

Květen 2021
Vedúci práce: prof. Ing. Jan Holub, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Suchý** Jméno: **Peter** Osobní číslo: **465989**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Otevřená informatika**

Studijní obor: **Internet věcí**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Uživatelská aplikace pro detektor akustické události

Název bakalářské práce anglicky:

Design and implementation of an application for Android detecting an acoustic event by cooperation with a central server. Recommended platform: Android smartphone.

Pokyny pro vypracování:

Navrhněte, realizujte a otestujte aplikaci pro detekci akustických událostí pro Android. Aplikace musí být schopna komunikovat se sítí čidel pomocí serveru. Aplikace musí obsahovat tyto funkce: notifikace a lokalizace nové události s minimálním zpožděním při jejím zpracování, prohlížení historie událostí, správa uživatelů, správa databáze senzorů. Pro serverovou část je možné využít existující aplikaci (výsledek týmového projektu).

Seznam doporučené literatury:

[1] Sallai, J., Hedgecock, W., Völgyesi, P. et al: "Weapon classification and shooter localization using distributed multichannel acoustic sensors." Journal of Systems Architecture - Embedded Systems Design. Vol. 57, 2011
[2] Svatos, J., Holub, J.: "Smart Acoustic Sensor. 5th International Forum on Research and Technologies for Society and Industry: Innovation to Shape the Future, pp. 161-165, 2019

Jméno a pracoviště vedoucí(ho) bakalářské práce:

prof. Ing. Jan Holub, Ph.D., katedra měření FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **21.07.2020** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce:
do konce letního semestru 2021/2022

prof. Ing. Jan Holub, Ph.D.	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce		podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Poďakovanie / Prehlásenie

Hlavné poďakovanie patrí vedúcemu práce prof. Ing. Janovi Holubovi, Ph.D. za podporu a pomoc pri spracovávaní tejto práce. Ďalej ďakujem mojej rodine a priateľom za ich podporu počas celého štúdia.

Prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky použité informačné zdroje v súlade s Metodickým pokynom o dodržiavaní etických princípov pri príprave vysokoškolských záverečných prác.

V Prahe dňa 21.5.2021

.....

Abstrakt / Abstract

Cieľom tejto práce je návrh a implementácia mobilnej aplikácie na detekciu akustických udalostí. K aplikácii bolo implementované API, ktorého hlavná funkcia je spracovanie týchto udalostí a následna notifikácia používateľov aplikácie.

V návrhu je bližšie predstavená vybraná technológia pre vývoj mobilnej aplikácia a API.

V teoretickej časti je súhrn základných informácii o použitých technológiach a ich špecifikách. Ďalej je predstavený algoritmus multilaterácie a vysvetlenie jeho použitia.

Záverečná časť rieši konkrétnu implementáciu a funkcionalitu ako mobilnej aplikácie tak aj API.

Kľúčové slová: Mobilná aplikácia, Android, Flutter, Dart, API, Spring Boot, Multilaterácia, Firebase

The aim of this bachelor thesis is the design and implementation of a mobile application for the detection of acoustic events. Notifications about new events are handled by API, which was implemented as a part of the final solution.

In the section named design is explained the selection of programming framework and programming language for mobile application and API.

The theoretical part is a summary of basic information about the chosen technologies and their specifics. Following with the explanation of the multilateration algorithm and its use.

The final part addresses the specific implementation and functionality of both projects - the mobile application and the API.

Keywords: Mobile application, Android, Flutter, Dart, API, Spring Boot, Multilateration, Firebase

Title translation: Design and implementation of an application for Android detecting an acoustic event by cooperation with a central server

Obsah /

/ Obrázky

Kapitola 1
Úvod

Mobilné aplikácie sú v dnešnej dobe neoddelitelnou súčasťou takmer každého jednotlivca. Široké spektrum dostupných aplikácii zrýchľuje a uľahčuje užívateľom riešenie jednoduchých aj zložitých problémov. Väčšina aplikácii je po uvedení na trh naďalej vyvíjaná a zlepšovaná podľa potrieb používateľov.

Cieľom práce je návrh a implementácia mobilnej aplikácie, ktorá bude slúžiť na reprezentáciu zaznamenaných akustických udalostí, ako je napríklad výstrel zo zbrane. Aplikácia je určená pre mobilné telefóny s operačným systémom Android, hlavne z dôvodu nižšej obstarávacej ceny v porovnaní s telefónmi s operačným systémom iOS.

Zaznamenanie akustického pulzu je realizované pomocou detektoru s mikrofónom, ktorý po zaznamenaní udalosti odošle vopred dohodnutý časový údaj na výpočtový server. Na úspešnú lokalizáciu zaznamenanej udalosti je potrebný časový údaj z minimálne troch čidiel, pričom hustota umiestnenia týchto čidiel zvyšuje presnosť lokalizácie. Po úspešnom vypočítaní zdroja konkrétnej udalosti sú všetci používatelia upozornení pomocou notifikácie. Udalosť bude v aplikácii reprezentovaná zobrazením všetkých dôležitých prvkov na mape - samotný zdroj, všetky čidlá, ktoré zaznamenali udalosť a poloha používateľa. Na správne fungovanie aplikácie je potrebné internetové pripojenie.

Systémy s podobnou funkciou boli v minulosti vyvíjané výhradne pre vojenské účely. Vzhľadom na kritický stav v spoločnosti je držanie a použitie zbraní častejšie ako kedysi. To znamená, že tieto systémy by mali uplatnenie aj na verejných priestranstvách ako sú nemocnice, univerzitné kampusy a miesta s vysokou fluktuáciou obyvateľstva. Aplikácia, spolu s výpočtovým serverom a čidlami, by mohla byť v budúcnosti veľkým prínosom najmä pre bezpečnostné zložky. Jednou z hlavných výhod je okamžité nahlásenie vzniknutej udalosti, ktorá by bola pravdepodobne nahlásená telefónatom od svedka. Okrem bezpečnostných zložiek štátu by mohla byť vhodná pre súkromné firmy, ktorých zameranie je stráženie objektov.

Kapitola 2
Návrh

Spolu s mobilnou aplikáciou bolo navrhnuté aj API (viď Kapitola 6). Je využívané na odosielanie notifikácii a schopnosť lokalizácie udalosti. Výpočty spojené s lokalizáciou je optimálnejšie realizovať na strane servera, ako na strane klienta. Základné požiadavky pri návrhu tohto riešenia sú:

- spustiteľnosť na operačnom systéme Android,
- komunikácia so sieťou čidiel pomocou centrálneho serveru alebo API,
- lokalizácia udalosti na mape s aktuálnou polohou používateľa,
- prijatie a spracovanie notifikácie pri novej udalosti,
- možnosť zobrazenia predchádzajúcich udalostí,
- správa databázy senzorov a uživateľov.

2.1 Mobilná aplikácia

Na vývoj mobilnej aplikácie bola zvolená technológia Flutter (viď Kapitola 5), vďaka ktorej je veľmi jednoduché v budúcnosti danú aplikáciu spustiť na operačnóm systéme iOS. Ďalšou výhodou je množstvo pluginov a balíkov, ktoré sú ľahko dostupné a využiteľné v rámci frameworku.

Jeden z nich je aj balík na správu jazyka aplikácie. To znamená, že aplikácia podporuje jazyky čeština a angličtina. Jazyk aplikácie je odvodený od predvoleného jazyka daného mobilného zariadenia.

Väčšina funkcii je možná vďaka jednoduchej integrácií určitých služieb platformy Firebase (viď Kapitola 7) ako autentifikácia používateľov, používanie real-time databázy a takisto spracovanie notifikácií.

2.2 API

Implementácia API bola potrebná kvôli nasledujúcim základným funkciám:

- lokalizácia zdroja akustickej udalosti pomocou algoritmu TDoA (viď Kapitola 8.1),
- odoslanie notifikácie všetkým používateľom v prípade novej akustickej udalosti, bez rozielu spôsobu jej spracovania - samotné API alebo centrálny server,
- uloženie výsledkov do real-time databázy po spracovaní akustickej udalosti.

Výber technológie na implementáciu API bola podmienka integrácie so službami platformy Firebase (viď Kapitola 7). Firebase poskytuje integráciu *Admin SDK* pre programovacie jazyky (v prípade Javascriptu sa jedná o frameworky), ktoré su často používane na tvorbu serverových aplikácii: Node.js, Java, Python, Go, C#. Takisto bolo najlepším riešením použíť framework, ktorý je overený časom a tým, že je používaný v produkčných aplikáciach.

Počas analýzy a porovnávania rôznych možností bol najvhodnejší balík, ktorý obsahoval základnú implementáciu multilateračného algoritmu TDoA (viď Kapitola 8.1)

v programovacom jazyku Java. Túto implementáciu bolo jednoduché upraviť konkrétne pre tento prípad[1], takže výber frameworku na implementáciu celého API bol zúžený na technológie využívajúce jazyk Java.

Ideálnou možnosťou bol framework **Spring boot**, ktorý spĺňoval všetky podmienky - podpora jazyku Java a využitie vo veľkom množstve produkčných aplikácií.

Nakoniec bol zvolený programovací jazyk **Kotlin**, ktorý je jeden z JVM jazykov, takže jeho integrácia s Javou je jednoduchá a zároveň eliminuje nevýhody jazyka Java, ako napríklad potreba písania veľkého množstva kódu, ktorý Kotlin rieši interne.[2]

Ukážka vytvorenia *data class* v jazyku Kotlin:

```
//Person.kt
data class Person(var name: String)
```

Vytvorenie triedy s rovnakými vlastnosťami v jazyku Java:

```
//Person.java
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Kapitola 3
Operačný systém Android

Android je operačný systém, ktorého základom je upravená verzia linuxového jadra. Pôvodne bol vytvorený pre mobilné telefóny a tablety, no dnes podporuje veľké množstvo platforiem ako napríklad inteligentné hodinky alebo televíziu.

Jeho vznik a vývoj je zásluhou skupiny vývojárov, známych pod menom Open Handset Alliance, ktorého súčasťou sú globálne spoločnosti ako Google, Samsung, Intel Corporation a iné.[3] Aktuálna verzia systemu je Android 11, oficiálne vydaná dňa 8. 9. 2020.[4]

Od roku 2012 je Android najrozšírenejším operačným systémom v inteligentných telefónoch v celosvetovom merítku a svoje prvenstvo si zatiaľ každý nasledujúci rok udržal.[5] V jeho predchodcaČeskej republike je posledné tri roky nad hodnotou 75% trhového podielu mobilných operačných systémov.[6] Jedným z dôvodov je nižšia obstarávacia cena v porovnaní s konkurenčným operačným systémom iOS, ktorého tvorcom je spoločnosť Apple. Jeho podieľ na trhu v Českej republike je okolo 20%.

3.1 Architektúra

Štruktúra operačného systému Android je z pohľadu architektúry tvorená z piatich vrstiev, ktoré sú rozdelené do šiestich hlavných zložiek. (viď Obrázok 3.1).[7]

Obrázok 3.1. Hlavné časti architektúry.

3.1.1 Linuxové jadro

Linuxové jadro je jednou z najdôležitejších súčastí softvéru na každom zariadení so systémom Android. Toto open-source systémové jadro je súčasťou operačných systémov osobných počítačov, serverov a niektorých typov vstavaných zariadení ako smerovače, tablety, inteligentné televízie a hodinky. Súčasťou jadra je aj kód pre rôzne

hardvérové ovládače a architektúry, ktoré podporuje. Každý systém používa vždy iba zlomok kódu, definovaný cieľovou architektúrou.

Zložka Android runtime(viď Kapitola 3.1.3) sa v rámci základných funkcií spolieha práve na jadro linuxu - spravovanie nízkoúrovňovej pamäte a operácie s vláknami.

3.1.2 Hardvérová abstrakčná vrstva

Hardvérová abstrakčná vrstva je zložená z viacerých knižníc, ktoré tvoria moduly. Každý modul obsahuje implementáciu rozhrania pre konkrétny typ hardvérovej časti, ako je napríklad bluetooth alebo fotoaparát. Hlavnou úlohou je poskytnúť všetky hardvérové možnosti daného zariadenia pre výššiu vrstvu Android framework (viď Kapitola 3.1.4).

3.1.3 Natívne C/C++ knižnice a Android runtime

Android runtime je využívaný systemovými službami, niektorými aplikáciami a bol vytvorený výhradne pre platformu Android. Spúšťa tzv. DEX súbory, ktoré majú formát *byte* kódu. Tento byte kód je optimalizovaný na minimálnu pamäťovú záťaž. Do verzie 5.0 bol používaný jeho predchodca Dalvik runtime. Medzi jeho nevýhody patrí napríklad horšia podpora ladenia programu, akým je nedostatok podrobných výnimiek a hlásení o zlyhaniach. Aplikácia fungujúca na Android runtime je spustiteľná aj na Dalvik runtime.[8] Opačný prípad nie je garantovaný.

Určité komponenty ako Hardvérová abstrakčná vrstva (viď Kapitola 3.1.2) alebo Android runtime sú napísané v natívnom kóde, ktorý využíva knižnice v jazyku C/C++.

Ak je aplikácia naprogramovaná pomocou jazyka C/C++, programátor má vďaka Android-NDK prístup k týmto natívnym knižniciam.

3.1.4 Android framework

Android framework je rozhranie pre programovanie aplikácií (viď Kapitola 6), ďalej iba API, napísané v jazyku Java. Hlavná úloha je poskytovať základne sady funkcií operačného systému Andorid, ktoré sú základom každej aplikácie. Napríklad na používanie fotoaparátu, je vytvorené API priamo na jeho činnosť. Prostredníctvom API prístupuje k hardvéru zariadenia čo znamená, že Android načíta modul knižnice pre túto hardvérovú súčasť (viď. Kapitola 3.1.2).

Vývojári majú plný prístup k rovnakým API[9], ktoré používajú systémové aplikácie Android ako:

- správca upozornení, ktorý umožňuje aplikáciam zobrazovať vlastné upozornenia,
- správca zdrojov, ktorý poskytuje prístup k nekódovým zdrojom, ako sú napríklad lokalizované texty, grafika,
- rôzne grafické komponenty ako zoznamy, mriežky, textové polia, tlačidlá a iné, vďaka ktorým je môžné vytvorenie používateľského rozhrania aplikácie.

3.1.5 Aplikácie

Súčasťou operačného systému Android je aj množstvo aplikácii, vhodných k bežnému používaniu ako napríklad aplikácia na správu elektronickej pošty, správ, kontaktov alebo webový prehliadač. Aplikácie tretích strán môžu byť predvolenými pre určité použitie. Základné systémové aplikácie väčšinou poskytujú kľúčovú funkciu, ku ktorým majú prístup všetci vývojári a vďaka tomu si nemusia všetkú funkcionalitu implementovať sami. Vhodným príkladom je odosielanie SMS správ. Pre odoslanie správy stačí zavolať funkciu na to určenej aplikácie.

3.2 Možnosti pri vývoji Android aplikácie

Pred samotným začatím vývoja ľubovolnej aplikácie je dôležité urobiť analýzu rôznych prístupov a vybrať ten, ktorý prináša najväčšie množstvo výhod. Najbežnejšie je rozhodovanie medzi natívnou, hybridnou a multiplatformnou aplikáciou.[10] Správny výber môže mať veľký vplyv na celkový úspech aplikácie medzi používateľmi.

3.2.1 Natívna aplikácia

Na vývoj natívnej aplikácie je používaný programovací jazyk Java a od roku 2017 aj jazyk Kotlin. Aplikácie získavajú všetky možné výhody zariadenia a funkcií operačného systému. To znamená, že môžu využívať priamy prístup k hardvérovým možnostiam daného zariadenia, ako je GPS, fotoaparát, mikrofón a ďalšie. Práve vďaka tomu sú tieto aplikácie charakteristické vysokým výkonom a prijemným používateľským dojmom.

3.2.2 Hybridná aplikácia

Tieto aplikácie sú špecifické svojím zložením. Pozostávajú z webových technológii ako HTML, CSS a Javascript, ktoré sú pomocou technológii ako Apache Cordova alebo Ionic transformované do nativných komponentov (viď. Kapitola 3.1.4). Vďaka pluginom sú tieto technológie schopné pristupovať k natívnym funkciám platformy. Všetky hybridné aplikácie majú v sebe zabudované prehliadače pomocou WebKitu. Webkit je jadro webového prehliadača, ktoré má na starosti vykreslovanie webových stránok v samostatnom okne. V Android sa používa **WebView**, ktoré je jeho implementáciou.

3.2.3 Multiplatformná aplikácia

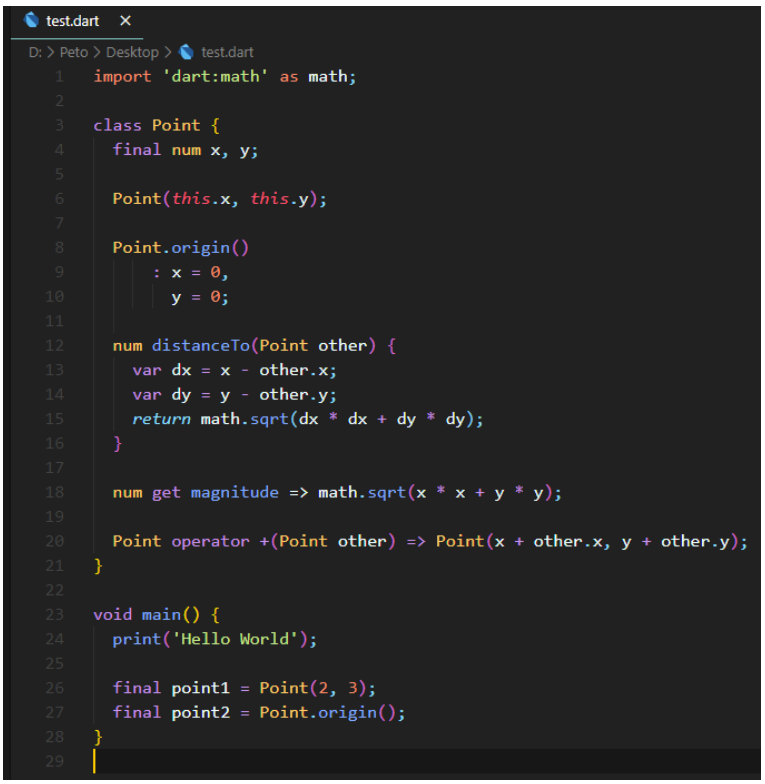
Jediná spoločná vlastnosť multiplatformných aplikácií s hybridnými je možnosť zdielať kód medzi viacerými platformami. Tento prístup je preferovaný v prípadoch, kde je potrebný vysoký výkon aplikácií, avšak finančné prostriedky nie sú dostačujúce na vývoj natívnych aplikácií pre obe platformy (Android, iOS). Existuje veľké množstvo technológií, ktoré takýto vývoj umožňujú. Medzi najznámejšie a najpoužívanejšie patrí React Native, Xamarin a Flutter.

Kapitola 4
Programovací jazyk Dart

Dart je objektovo orientovaný jazyk na univerzálne použitie. Je optimalizovaný pre rýchly vývoj klientských aplikácií, ktoré je možné spustiť na akejkoľvek platforme. Napriek tomu, že je Dart navrhnutý na klientské aplikácie, vďaka veľkému množstvu knižníc je použiteľný aj na strane servera.[11]

Prvá verzia Dartu bolá navrhnutá dvoma programátormi z Dánska - Larsom Bakom a Kasperom Lundom. Za ďalší vývojom jazyka Dart stojí globálna spoločnosť *Google*. Cieľom bolo nahradiť jazyk Javascript vo webových aplikáciach. Javascript umožňuje programátorom veľkú voľnosť, čo znamenalo množstvo zbytočných chýb. V roku 2015 bol tento plán zrušený a novým cieľom bola možnosť skompilovať Dart do Javascriptu, pretože aj napriek svojim nedostatkom patrí medzi najpoužívanejšie programovacie jazyky.[12]

Dart ma veľmi podobnú syntax ako Java (viď Obrázok 4.1) a je ovplyvnený mnohými modernými programovacími jazykmi ako Swift, C#, Kotlin alebo TypeScript.



Obrázok 4.1. Ukážka programovacieho jazyka Dart.

Dnes je Dart možné skompilovať do nativného kódu alebo do Javascriptu. Túto možnosť využívajú frameworky **Flutter** (viď Kapitola 5) a **AngularDart**, ktoré môžu byť použité na tvorbu:

- mobilných aplikácii, spustiteľných na operačnom systéme Andorid a iOS - Flutter
- aplikácii pre stolné počítače, spustiteľných na najpoužívanejších operačných systémoch ako Windows, Linux a Mac OS - Flutter
- webových aplikácácii, ktoré je lepšie rozdeliť na:
  - jednoduché aplikácie alebo tzv. single-page aplikácie - Flutter, AngularDart,
  - rozsiahle aplikácie alebo tzv. enterprise aplikácie, na ktoré sú vyššie funkcionálne a používateľské nároky - AngularDart.

4.1 Základné vlastnosti

V jazyku Dart neexistujú finálne metódy, takže existuje možnosť prepísať takmer všetky metódy. Určité zabudované operátory tvoria výnimky a nie je možné zmeniť ich pôvodnú implementáciu. To ale neznamená, že Dart nepodporuje finálne premenné. Naopak, v zdrojových kódoch sú veľmi často používané, rovnako ako aj finálne atribúty tried.[11]

Na rozdiel od mnohých iných programovacích jazykov, Dart nepoužíva na definovanie identifikátora (trieda, premenná, metóda, a ďalšie...) kľúčové slová: **Public**, **Private** a **Protected**. Na označenie private identifikátora je využívaný znak podčiarknutia „ _ “. Pre využitie tejto možnosti, musí byť prvým znakom v názve identifikátora, čo obmedzí jeho viditeľnosť na rámec prisluchajúcej knižnice.

```
Class Person{
    final String _name;

    Person(this._name)
}
Person p = Person("Karel")
```

Dart, ako mnohé iné jazyky, podporuje funkciu najvyššej úrovne - funkcia main(), ktorá je štartovým bodom každej aplikácie. Takisto je možné využitie vnorených funkcií. Vnorenou funkciou sa rozumie funkcia definovaná v rámci inej funkcie.

4.2 Typy

Všetky typy funkcií a premenných v Darte sú založené na rozhraniach, nie na triedach ako je to bežné. To znamená, že každá trieda implicitne vytvára rozhranie, ktoré je možné implementovať na iné triedy. Dart je považovaný za typovo bezpečný jazyk. Dosahuje to vďaka kontrole statického typu, ktorý zabezpečí, aby bola vždy hodnota v súlade s typom premennej.

Využitie typov je povinné, no ich explicitné určenie nie, vďaka možnosti odvodenia. Toto neurčenie je využívané pri komplexných operáciach, ktorých výsledky môžu byť odlišné na základe veľkého množstva parametrov.[12] V takomto prípade je preferovaným spôsobom používanie dynamického typu, ktorý je kontrolovaný počas behu programu. Používanie explitných typov je doporučované najmä z dôvodov, akými sú presnejšie chybové hlásenia, zlepšenie výkonu, jednoduchšia práca s vývojovým prostredím.[11]

4.3 Hodnoty premenných

Ďalšou výhodou oproti iným programovacím jazykom je **null-safety**. To znamená, že hodnota premenných nemôže byť *null*, pokiaľ nie je explicitne určené, že *null* je prípustná hodnota. Vďaka tejto vlastnosti a pomocou statickej analýzy kódu, Dart predchádza známym výnimkam počas spustenia programu, ktoré sú spôsobované tým, že hodnota premennej je *null*.[11]

4.4 Synchrónny a asynchrónny kód

Napriek tomu, že je Dart jednovláknový jazyk, existuje možnosť písať kód, ktorý môže bežať asynchrónne. Základ tejto funkcie je v knižnici *dart:async*. Vytvorenie metódy, ktorá je asynchrónna k zvyšnej časti kódu je dosiahnuteľne vďaka kľučovému slovu **async**. Metóda označená ako asynchrónna sa začne vykonávať až po skončení synchrónnej časti. Toto poradie je možné zmeniť pomocou kľúčového slova **await** pred zavolaním asynchrónnej metódy.[12]

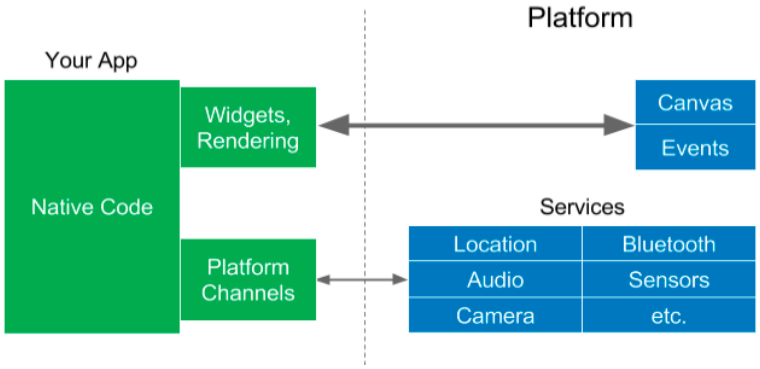
4.5 Ďalšie knižnice

Väčšina moderných jazykov obsahuje knižnice pre zjednodušenie kódu a odstránenie nutnosti implementovať všetko od nuly. Dart nie je výnimkou a jeho súčasťou je množstvo knižníc, ktoré za programátora riešia veľa problémov. Napríklad knižnica na implementáciu HTTP komunikácie, kódovanie a dekódovanie do/z JSON, zložitejšie datové štruktúry ako binárne stromy alebo spojové zoznamy a mnoho ďalších.[11]

Kapitola 5
Framework Flutter

Flutter je open-source sada nástrojov na vytváranie používateľského rozhrania, vytvorená v programovacom jazyku Dart (viď Kapitola 4) spoločnosťou *Google*. Cieľom je poskytnúť vývojárom môžnosť vytvárať aplikácie, ktorým nezáleží na akej platforme sú spustené, bez ovplyvnenia výkonu. Vďaka tomu má používateľ pri práci s aplikáciou pocit, že bola vyvíjana na danú platformu.[13] Kvôli špecifickým potrebám každej platformy existujú rozdiely, ktoré *Flutter* zohľadňuje, aby aplikácia pôsobila na používateľa prirodzene a zároveň, aby bolo možné zdieľať čo najväčšiu časť kódu. Všetky výhody prináša programovací jazyk Dart. Flutter využíva reaktívny prístup, čo znamená, že aktualizuje to, čo používateľ vidí na základe aktuálneho stavu aplikácie.[14]

Výnimočnosť Flutteru, narozdieľ od iných mobilných frameworkov, je v prístupe dosahovania multiplatformnosti. Najväčšiou výhodou je elimnínácia jazyka Javascript, ktorý na komunikáciu s platformou využíva spojenie spôsobujúce zníženie výkonu. Ďalšou výhodou je vykreslovanie elementov na obrazovku, čo je riešené priamo vo frameworku (viď Obrázok 5.1) a vďaka kompilácii do natívneho kódu nenastáva znižovanie výkonu.



Obrázok 5.1. Schéma architektúry frameworku Flutter.[15]

Hoci je Flutter považovaný za mladý projekt, existuje niekoľko produkčných aplikácii, u ktorých našiel využitie. Sú to napríklad aplikácie Google Ads, Reflectly, Music Tutor alebo Cryptomaniac Pro.[16]

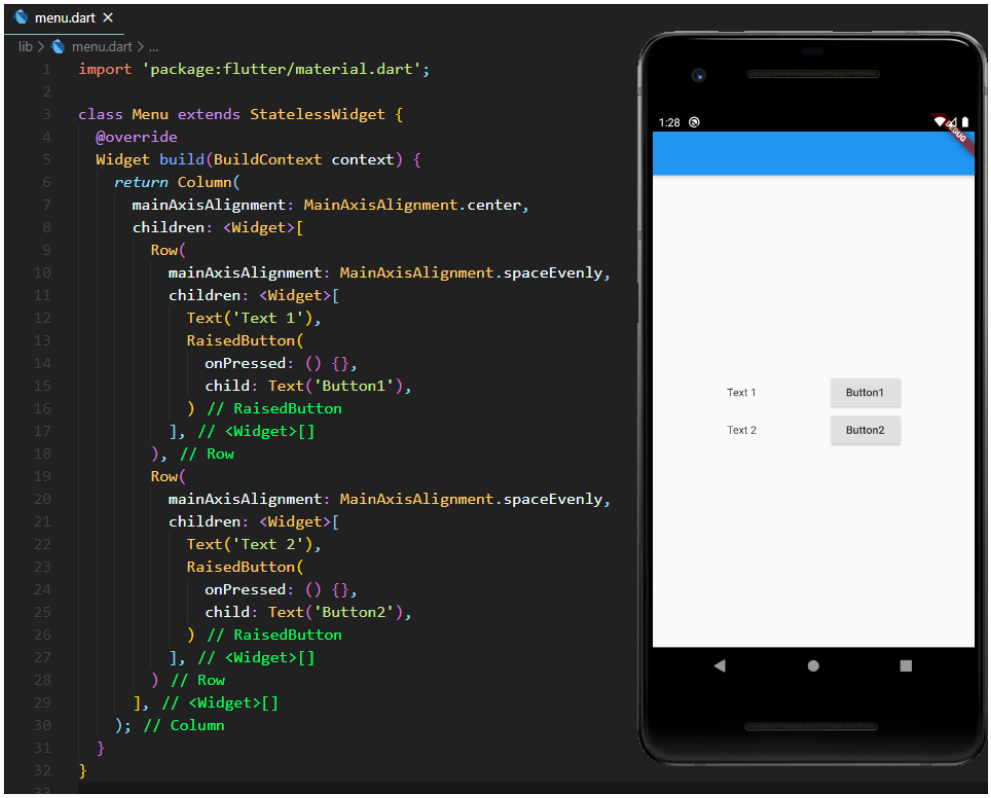
Flutter, rovnako ako programovací jazyk Dart, je tvorený veľkým množstvom knižníc (viď Kapitola 4.5), ktoré zjednodušujú vývojárom prácu a tým pádom zrýchľujú celý proces vývoja aplikácie. Množstvo knižníc je výtvorom tímu, ktorý stojí za celým frameworkom. Za ešte väčším počtom knižníc stojí práve komunita nadšencov Flutteru. Toto je ďalší dôvod prečo má Flutter budúcnosť vo svete tvorby nových aplikácii.[13]

5.1 Tvorba používateľského rozhrania

Základný stavebný prvok používateľského rozhrania je **widget**.[17] Existujú dva typy widgetov - **statefull** a **stateless**.[18] Widgety, ktoré nepotrebujú udržiavať nejaký vnútorný stav, patria do skupiny *stateless*. Do tejto skupiny patrí napríklad widget na zobrazenie textu, obrázku alebo prázdny widget na vytvorenie medzery.

Zložitejšie widgety, ktoré si uchovávajú svoj vnútorný stav na zobrazenie určitého obsahu, patria do skupiny *statefull*. To, že majú widgety vnutorný stav, znamená možnosť dynamického menenia výzoru widgetu počas behu aplikácie. Napríklad zoznamy obrázkov, ktoré podľa vnútorného stavu meneného interakciou používateľa, pohybujú svojím obsahom a zobrazujú iba určitý obrázok. Ďalším príkladom môže byť sťahovanie obrázku z internetu. Používateľ na začiatku vidí informáciu o tom, koľko percent je stiahnutých, ktoré sa môžu pravidelne aktualizovať, a po úspešnom dokončení sťahovania vidí iba správu, že bol obrázok úspešne stiahnutý.[13]

Programátor ma možnosť využiť základné widgety, ktoré sú súčasťou Flutteru a zároveň dodržiavajú *Material* dizajn, známe z platformy Android, alebo *Cupertino* dizajn, ktoré svojím výzorom kopírujú natívne komponenty z platformy iOS. Ďalšou možnosťou je vytvoriť vlastný komplexnejší widget, pomocou využitia už základných widgetov (viď Obrázok 5.2).



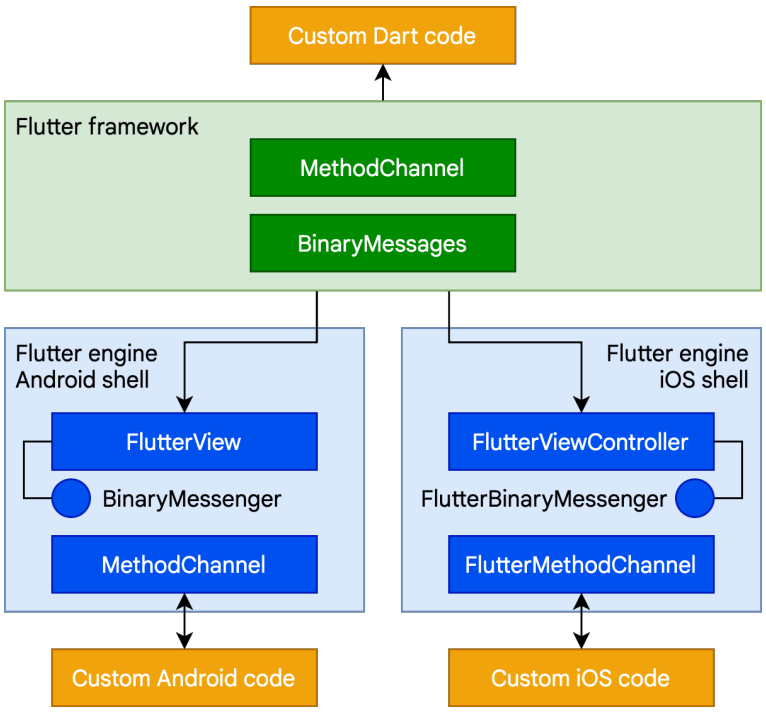
Obrázok 5.2. Komplexný vlastný widget.

5.2 Integrácia so špecifickým kódom platformy

Flutter poskytuje rôzne možnosti integrácie kódu alebo rozhrania, napísaného v jazyku špecifického pre danú platformu - jazyk Java alebo Kotlin pre Android, jazyk Swift alebo Objective-C pre iOS alebo pri volaní natívnych rozhraní v jazyku C. Takisto podporuje vloženie Flutter modulu do natívnej aplikácie. Tieto funkcie sú možné vďaka **platformným kanálom**.[19]

5.2.1 Platformné kanály

Využívajú sa v aplikáciach určených pre stolové počítače alebo mobilné zariadenia. Vytvorením spoločného kanálu, je Flutter schopný odosielať a prijímať správy medzi Dartom a komponentom, ktorá je napísaný pre špecifickú platformu, napríklad v jazyku Kotlin (viď Obrázok 5.3). Pre vytvorenie spoločného kanálu je potrebné použiť rovnaké meno a *Kodek* na oboch koncoch. Kodek má na starosti kódovanie dát z rôznych typov premenných do validných typov danej platformy a späť. Napríklad typ Map v Darte je reprezentovaný typom HashMap v jazyku Kotlin.[19]



Obrázok 5.3. Integrácia s platformou Android a iOS.[19]

Kapitola 6
Application programming interface

V oblasti softverového inžinierstva je pojem *Application programming interface* (ďalej iba API) veľmi často používaný. Predstavuje rozhranie pre aplikácie a ich následnú tvorbu. Podrobnejšie pomenovanie je sada knižníc, funkcii, tried, protokolov a procedúr, ktorá uľahčuje prácu programátorom.[20] Podstata je v abstrahovaní základnej implementácie a poskytnutie iba určitých objektov alebo akcií, ktoré vývojár môže potrebovať. Najčastejšie využitie je pri vývoji webových a mobilných aplikáciách. (viď Obrázok 6.1).

Primárnou funkciou je jednosmerná (v určitých prípadoch môže byť obojsmerná) komunikácia a výmena dát medzi dvoma platformami. Napríklad na zobrazenie polohy firmy, predajne alebo určitých miest sa používa práve API poskytnuté v rámci Google maps. Pri použití sa zavolá potrebná časť kódu, ktorá zobrazí požadované údaje. Mapy predstavujú iba jedno z mnohých API, ďalším príkladom je prihlásenie do rôznych aplikácii pomocou externého účtu na sociálnych sieťach (Facebook, LinkedIn). Najznámejšie API, ktoré umožnuje vývojárom integráciu videí je Youtube API. Toto rozhranie je často využívané aj na zobrazenie streamov na iných weboch alebo pre vloženie playlistu.[21]

Využitie nachádza aj pri automatizácii určitých procesov vo firmách - jedno z typických použití môže byť napojenie na softvér pre účtovné procesy. Vďaka prepojeniu je možné ihneď po zaplatení objednávky vygenerovať faktúru, odoslať ju zákazníkovi, a prípadne ju rovno uložiť do databázy.

6.1 Typy rozhrania

Existujú tri základné typy rozhraní, ktoré sa stále používajú. Najstarším je *Simple Object Access Protocol* (ďalej iba SOAP), ktorý vo väčšine prípadov nahradil *Representational state transfer* (ďalej iba REST). Najmladším z nich je *GraphQL*.

6.1.1 SOAP

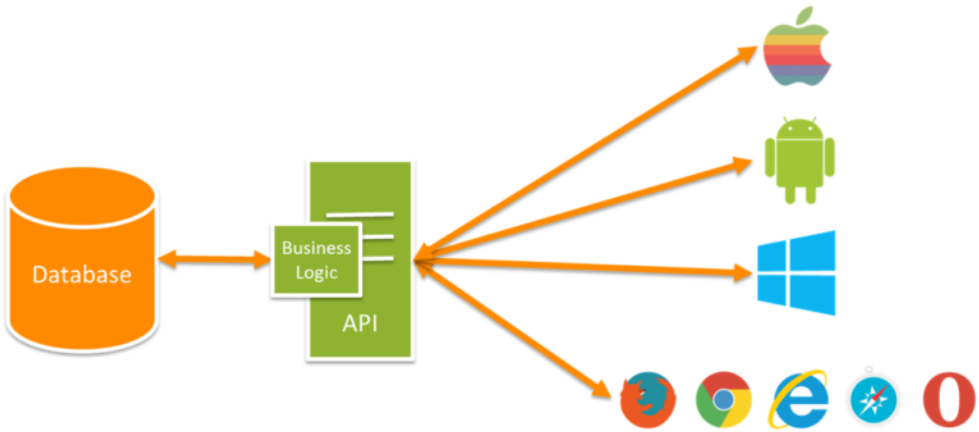
SOAP je protokol na výmenu správ, ktorý vytvára základnú vrstvu komunikácie medzi jednotlivými webovými službami. Jeho výhodou je, že poskytuje relatívne jednoduché prostredie na prípadnú tvorbu zložitejšej komunikácie. Správy využívajú značkovací jazyk XML, ktorý je ľahko čítateľný pre človeka a zároveň je ľahko spracovateľný strojom.

6.1.2 REST

REST je architektúra rozhrania klient-server, ktorá sa vyznačuje jednoduchým a jednotným prístupom k štyrom základným akciam, ktoré slúžia na vytvorenie, úpravu, odstánenie alebo čítanie dát zo servera. Každa z týchto akcií musí mať svoje jedinečné URL. Využíva protokol HTTP a považuje sa za relatívne bezpečný spôsob, pretože všetky citlivé údaje sú uložené na strane klienta.

6.1.3 GraphQL

GraphQL je dátový dotazovací jazyk pre API, ktorý je spustený na strane servera. Umožňuje presnú definíciu požadovaných dát, čo môže značne zredukovať množstvo nepodstatných informácii. Sprostredkováva získavanie dát z viacerých zdrojov v rámci jedného požiadavku. Podporuje čítanie, zapisovanie a dokáže sledovať aktualizácie údajov v reálnom čase (najčastejšie implementované pomocou websocketov).



Obrázok 6.1. Schéma použitia API.[22]

Kapitola 7
Firebase

Firebase je platforma od spoločnosti Google, často označovaná ako BaaS (backend as a service) alebo PaaS (platform as a service), ktorá je určená na jednoduchší vývoj mobilných a webových aplikácií. Použitie Firebase prináša vývojárovi API, servery, databázu a mnoho ďalšieho v jednom celku (viď Obrázok 7.1). Firebase poskytuje rôzne plány využitia a základný plán je úplne bez poplatkov a obsahuje to najdôležitejšie. Vďaka tomu sú vývojári sústredení na celkový dojem používateľského rozhrania a na vytvorenie príjemného zážitku používaním aplikácie. Firebase je nastavené čo najviac všeobecne, aby bolo konfigurovateľné a vyhovujúce širokej škále nárokov. Samozrejme, pre pokročilejšie aplikácie je určite potrebné využiť dalšie produkty alebo služby. Takisto existuje mnoho aplikácií, ktorým Firebase nedokáže poskytnúť všetko potrebné, no pre väčšinu aplikácií je Firebase veľkým prínosom a spôsobom ako ušetriť čas potrebný na vývoj aplikácie.[23]

7.1 Výhody a nevýhody

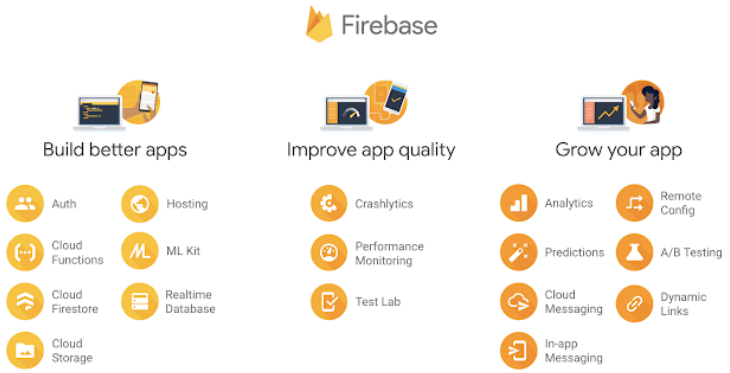
Toto rozdelenie je relatívne a vždy závisí od danej situácie.

Mezdi **výhody** patria:

- realtime databáza, ktorá je implementovaná pomocou websocketov, takže rýchlosť aktualizácií dát je obmedzovaná iba rýchlostou pripojenia používateľa,
- autentifikácia používateľov pomocou emailu a hesla alebo pomocou účtov na iných sociálnych sieťach ako Facebook, Twitter a Google,
- Firebase Storage poskytuje jednoduchý spôsob ukladania binárnych súborov - najčastejšie obrázkov,
- možnosť odosielať notifikácie a personalizované správy používateľom.

Medzi **nevýhody** patria:

- tradičné relačné dátové modely nie sú použiteľné pre NoSQL databázu,
- obmedzené možnosti dotazovania dát, kvôli architektúre datového modelu,
- pri aplikáciach, ktoré sú využívané veľkým množstvom používateľov môže byť bezplatný plán nedostatočný

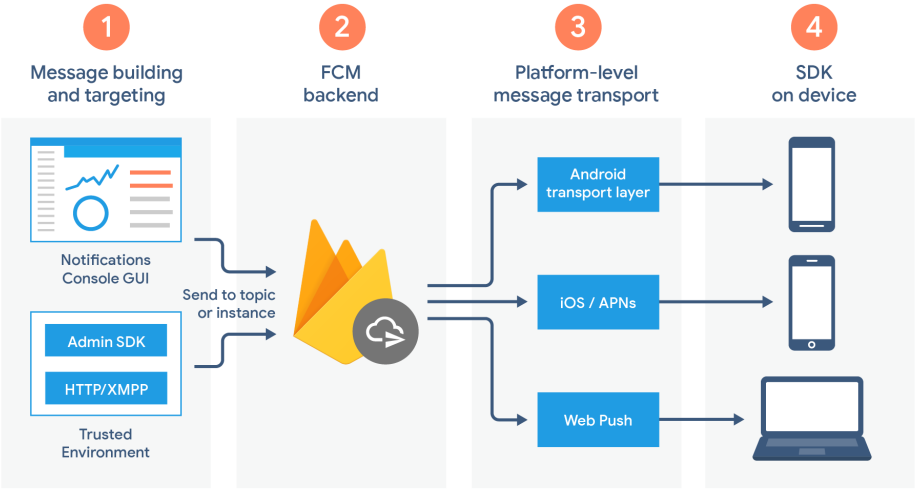


Obrázok 7.1. Služby zahrnuté v platforme Firebase.[24]

7.2 Odosielanie a spracovanie notifikácii

Na odosielanie a spracovanie notifikácii je využitá jedna zo služieb platformy Firebase. **Firebase Cloud Messaging** je riešenie na odosielanie správ medzi platformami.[25]

Celý princíp od odoslania až po spracovanie notifikácií je zložený zo štyroch krokov (viď Obrázok 7.2). Prvý krok je samotné vytvorenie notifikácie alebo správy. Firebase ponúka možnosť jej vytvorenia pomocou Firebase konzoly (táto možnosť je využívaná na rýchle otestovanie). Na automatické vygenerovanie notifikácie je potrebné vytvoriť žiadosť o správu pomocou Firebase Cloud Messaging serverových protokolov alebo pomocou *Firebase Admin SDK*. Následne je táto žiadosť odoslaná na servery, kde je spracovaná a odoslaná na koncové zariadenie. Tretí krok je prijatie správy koncovým zariadením. Toto je zabezpečené tranportnou vrstvou danej platformy. Spracuje danú správu a pridá k nej potrebné špecifické údaje podľa cieľovej platformy. Štvrtý, zároveň posledný krok je spracovanie správy na strane klienta pomocou SDK na danej platforme.[26] Takže vývojar implementuje iba krok jedna a štyri, pomocou SDK na to pripravenými.



Obrázok 7.2. Schéma architektúry odosielania notifikácii.[26]

Kapitola 8
Multilaterácia

Multilaterácia je technika využívaná na zisťovanie polohy. Princíp je založený na meraní a zaznamenávaní presného času, kedy bola prijatá energetická vlna (seizmická, rádiová, akustická, atď). Rýchlosť šírenia vĺn je vopred známa a takisto dôležitá pre ďalšie výpočty. Nevyhnutná podmienka na funkčnosť tejto metódy je použitie synchronyzovaných hodín. Napríklad za účelom sledovania nejakého predmetu je potrebné, aby vysielal určitý signál na viaceré sledovacie stanice, ktorých poloha je vopred známa a všetky tieto stanice majú zosynchronizované hodiny.

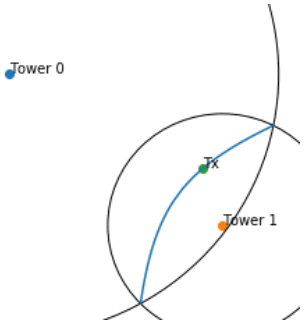
Multilateračný systém je v podstate meranie času príchodu signálu alebo zvuku, pomocou synchronizovaných hodín s neznámym časovým posunom od jeho skutočného času vyslania. Z nameraných časov príchodu je pomocou určitého algoritmu nájdených R súradníc cieľa. Na nájdenie súradníc v rozmere R potrebujeme najmenej $R+1$ meraní.

8.1 Algoritmus TDoA

Algoritmus *Time difference of arrival* (ďalej iba TDoA) využíva rozdiely medzi časmi, kedy bol zaznamenaný určitý signál alebo zvuk. Tieto rozdiely sú využívané na ďalšie výpočty pomocou ktorých nájde práve R súradníc zdroja. Pri tomto algoritme nie je potrebný čas prenosu, pretože po vynásobení časových rozdielov príchodov s rýchlosťou šírenia, získava skutočné vzdialenosti medzi zdrojom a minimálne dvoma stanicami. Kvôli tomu, že sa jedná o algoritmus využívaný pri multilaterácii, je nevyhnutá prítomnosť synchronizovaných hodín s vysokou presnosťou na každej stanici.

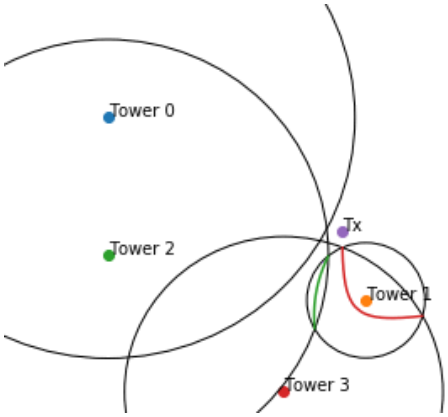
Po prijatí signálu aspoň dvoma stanicami je možné určiť časový rozdiel príchodu $tdoa$ medzi prvou a druhou stanicou. Čas z prvej stanice označíme ako t_1 a čas z druhej stancice ako t_2 a potom $tdoa = t_1 - t_2$. Vypočítaný rozdiel $tdoa$ je využívaný na určenie vzdialenosti, v ktorej je umiestnený zdroj signálu. Pri predpoklade, že $t_1 < t_2$ a zdroj leží na kružnici k_1 s polomerom r metrov od prvej stanice, potom musí ležať tiež na kružnici k_2 s polomerom $r + v * tdoa$ metrov od druhej stanice. Zdroj signálu, ktorého polohu chceme zistiť, musí ležať na priesečníku kružníc k_1 a k_2. Vo väčšine prípadov existujú dva priesečníky a tým získavame dve možné pozície zdroja. Existujú prípady, kedy dostaneme iba jeden priesečník, to znamená, že sa kružnice k_1 a k_2 iba dotýkajú. Ak sa kružnice k_1 a k_2 ani nepretínajú, ani nedotýkajú, nedostavame žiadny priesečník. Pomocou iterácie, počas ktorej je zväčšovaný polomer kružníc k_1 a k_2 a teda nájdením priesečníkov v každej iterácií, získavame hyperbolickú krivku, na ktorej leží zdroj signálu (viď Obrázok 8.1).

Pri počte staníc n sa tento proces opakuje medzi stanicou, ktorej prináleží najmenší čas t_n a každou ďalšou stanicou, čo vytvorí $n - 1$ hyperbol(viď Obrázok 8.2, 8.3, 8.4).

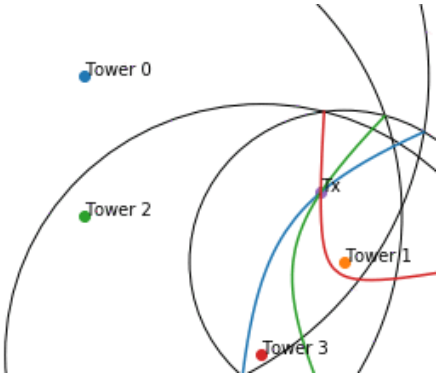


Obrázok 8.1. Vytvorenie hyperbolickej krivky.[27]

Obrázok 8.2. Začiatok TDoA algoritmu so štyrmi stanicami.[27]



Obrázok 8.3. Priebeh algoritmu - vytváranie hyperbol.[27]



Obrázok 8.4. Výsledok algoritmu.[27]

8.1.1 Použitie v R^2

Zdroj umiestnime na pozíciu $source = [source_x, source_y]$, ktorý odošle signál v čase t_{source}, sa šíri rýchlosťou v a prijíma ho n staníc. Stanica, ktorá ako prvá zachytí signál v čase t_{ref}, je na pozícii $pos_{ref} = [pos_{ref,x}, pos_{ref,y}]$. Zostávajúcich $n-1$ staníc je na pozícii $pos_i = [pos_{i,x}, pos_{i,y}]$ a prijali signál v čase t_i. Pre referenčnú stanicu na pozícii pos_{ref} môžeme usúdiť, že jej vzdialenosti od zdroja signálu na pozícii $source$ je rovná súčinu rýchlosti šírenia a času šírenia, ktorý získame rozdielom času, kedy ho prijal prvý prijímač a času odoslania:

$$\|(source - pos_{ref})\| = v * (t_{ref} - t_{source})) \tag{1}$$

Podobne vyjadríme ďalšie stanice:

$$\|source - pos_i\| = v * (t_i - t_{source})$$

$$\|source - pos_i\| = v * (t_i - t_{ref} + t_{ref} - t_{source})$$

$$\|source - pos_i\| = v * (t_i - t_{ref}) + v * (t_{ref} - t_{source}) \tag{2}$$

Kombináciou rovníc (1) a (2) dostaneme:

$$\|source - pos_{ref}\| = \|source - pos_i\| - v * (t_i - t_{ref}) \tag{3}$$

Úpravou rovnice (3) dostaneme $n-1$ rovníc hyperból pre n staníc:

$$\|source - pos_{ref}\| - \|source - pos_i\| - v * (t_i - t_{ref}) = 0 \tag{4}$$

kde $i = 1, ..., n-1$.

Tieto rovníce potom zapíšeme ako sústavu rovníc a pomocou metódou nelineárnych najmenších štvorcov hladáme optimálnu hodnotu $source$.[27]

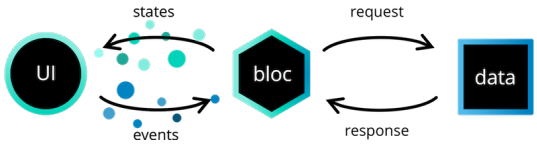
Kapitola 9
Implementácia mobilnej aplikácie

Aplikácia, pomenovaná **Acoustic Event Detector**, bola vytvorená v editore *Visual Studio Code* s rozšíreniami na prácu s programovacím jazykom Dart (viď Kapitola 4) a frameworkom Flutter (viď Kapitola 5). Ďalej boli využité rozšírenia na zjednodušenie práce s architektúrou BLoC (viď Kapitola 9.1) a na jednoduchšie spravovanie textov v českom a anglickom jazyku.

Po spustení aplikácie sa zobrazí obrazovka na prihlasovanie a až po úspešnom prihlásení je používateľ presmerovaný na hlavnú obrazovku. **Fungovanie aplikácie záleží od práv**, ktoré musia byť nastavené pri vytváraní nového používateľského účtu. Hlavná obrazovka vždy zobrazuje jednu zo štyroch obrazoviek a spodný panel so štyrmi ikonami. Kliknutím na ľubovolnú z nich nastáva zmena obrazovky. V prípade kliknutia na aktuálne zvolenú obrazovku nie je vykonaná žiadna akcia.

9.1 Business Logic Components (BLoC)

Business Logic Components (BLoC) je nový vzor architektúry. Jeho princíp je založený na prúde udalostí, ktorý zabezpečuje komunikáciu prostredníctvom komponént na to určenými (viď Obrázok 9.1).[28]



Obrázok 9.1. Architektúra BLoC.[28]

9.1.1 Dátova vrstva

Táto vrstva je zložená z dvoch zložiek. Prvá zložka má na starosti jednoduché API, pomocou ktorého vykonáva *CRUD* operácie (create, read, update a delete). Implementácia tejto časti býva čo najviac všeobecná a všestranná. Spravidla poskytuje nespracované údaje z databázy alebo sieťových volaní.

Druhá zložka prijíma nespracované údaje, ktoré môžu byť z viacerých zdrojov. Hlavná funkcia je zabezpečiť ich transformáciu na údaje, ktoré je možné odovzdať do vrstvy BLoC.[28]

9.1.2 Vrstva BLoC

Funkcia vrstvy BLoC je vytvoriť nový stav aplikácie z údajov z dátovej vrstvy na základe udalosti z prezentačnej vrstvy. Takisto môže byť závislá na viacerých komponentoch, ak z nich potrebuje údaje na vytvorenie nového stavu. Každá táto vrstva má svoj vlastný prúd udalostí. Viaceré prúdy môžu byť spolu prepojené, čo umožnuje spoločné reagovanie na udalosti a prípadne chyby.[28]

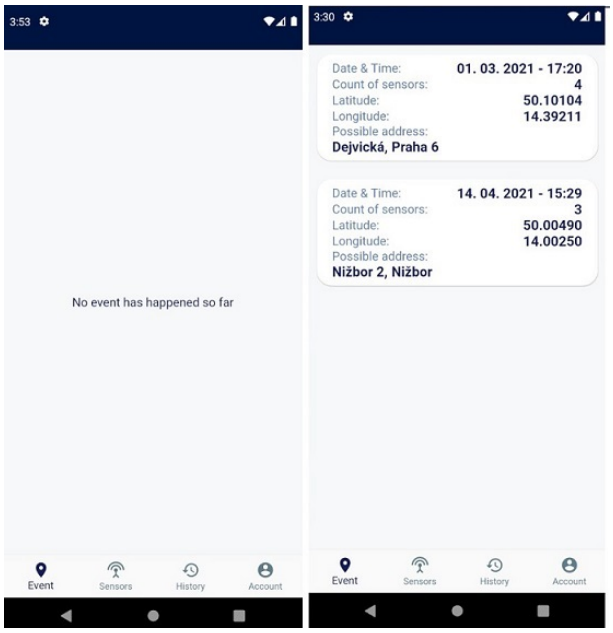
9.1.3 Prezentačná vrstva

Primárna funkcia prezentačnej vrstvy je rozpoznať, ako má vyzerať to, čo vidí používateľ. Činnosti ako prekreslovanie obrazovky, prípadne navigácia na inú obrazovku, sú závislé a odvodené od stavu aplikácie, ktorý dostane od BLoC vrstvy. Súčasne spracováva vstupy od používateľa, ktoré posiela ako udalosti do vrstvy BLoC.[28]

9.2 Obrazovka *Event*

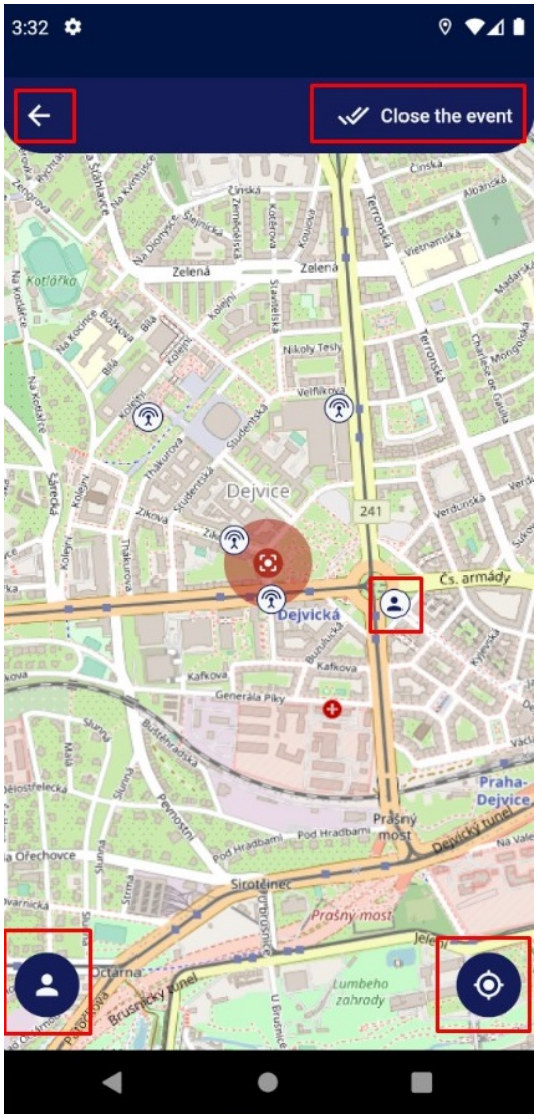
Táto obrazovka je vlastne základ celej aplikácie. Na obrazovke je zoznam nových udalostí, ktoré boli zaznamenané a zároveň ešte neboli uzavreté. Ľubovoľná udalosť je reprezentovaná kartou (viď Obrázok 9.2), ktorá obsahuje najdôležitejšie údaje:

- dátum a čas zaznamenania udalosti,
- počet senzorov, ktoré zaregistrovali udalosť,
- zemepisná šírka a dĺžka, podľa ktorých je určená približná adresa.



Obrázok 9.2. Obrazovka zobrazujúca zoznam nových/neuzatvorených udalostí.

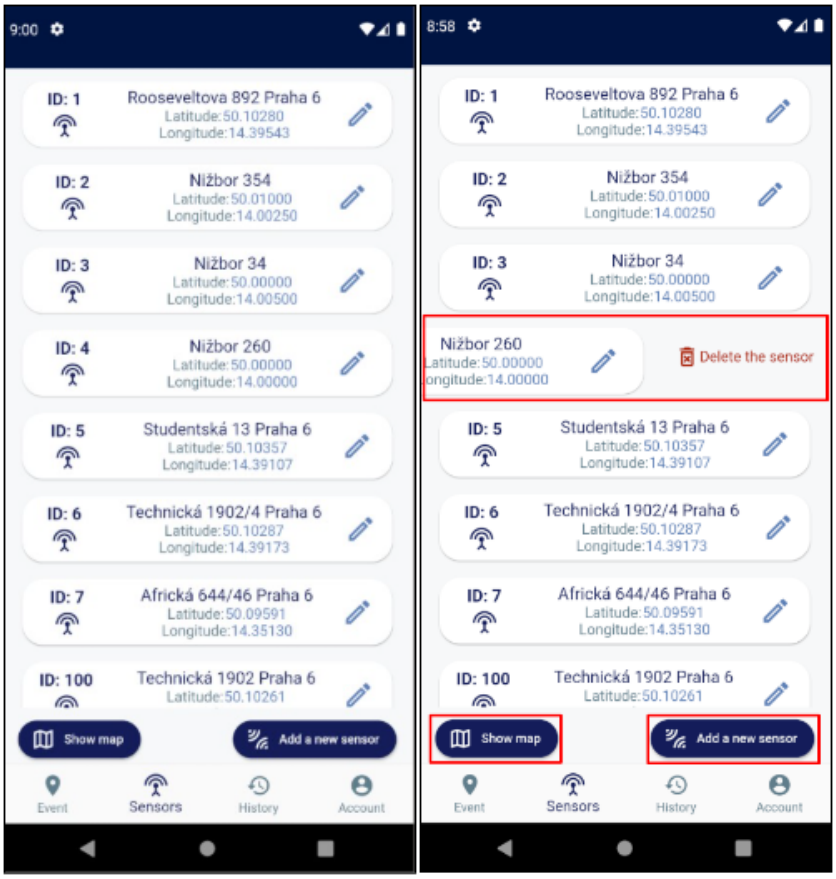
V prípade, že momentálne neprebieha žiadna udalosť alebo boli všetky uzavreté, je na stránke text, ktorý informuje používateľa o tejto situácii (viď Obrázok 9.2). Kliknutím na kartu je otvorená obrazovka, na ktorej je vybraná udalosť reprezentovaná na mape (viď Obrázok 9.3). Na zobrazenie tejto obrazovky je potrebné povoliť získavanie polohy. Na mape je potom zobrazená poloha senzorov, ktoré registrovali udalosť, poloha zdroja udalosti a zároveň poloha používateľa reprezentujúca skutočnú polohu. Táto poloha sa pravidelne aktualizuje bez nutnosti znova načítať obrazovku. Šípka v ľavom hornom rohu je štandardné tlačidlo na návrat na predchadzajúcu obrazovku. Tlačidlo s textom *Close the event* slúži na uzatvorenie udalosti, čo znamená, že bude odstránená zo zoznamu na obrazovke *Event* a pridaná do zoznamu na obrazovke *History* (viď Kapitola 9.4). Toto tlačidlo sa nezobrazuje používateľom s **obmedzenými právami**. Funkcia tlačidiel v spodnej časti obrazovky je vycentrovanie mapy na určitú polohu - ľavé na používateľa, pravé na zdroj udalosti.



Obrázok 9.3. Obrazovka zobrazujúca udalosť detailne.

9.3 Obrazovka *Sensors*

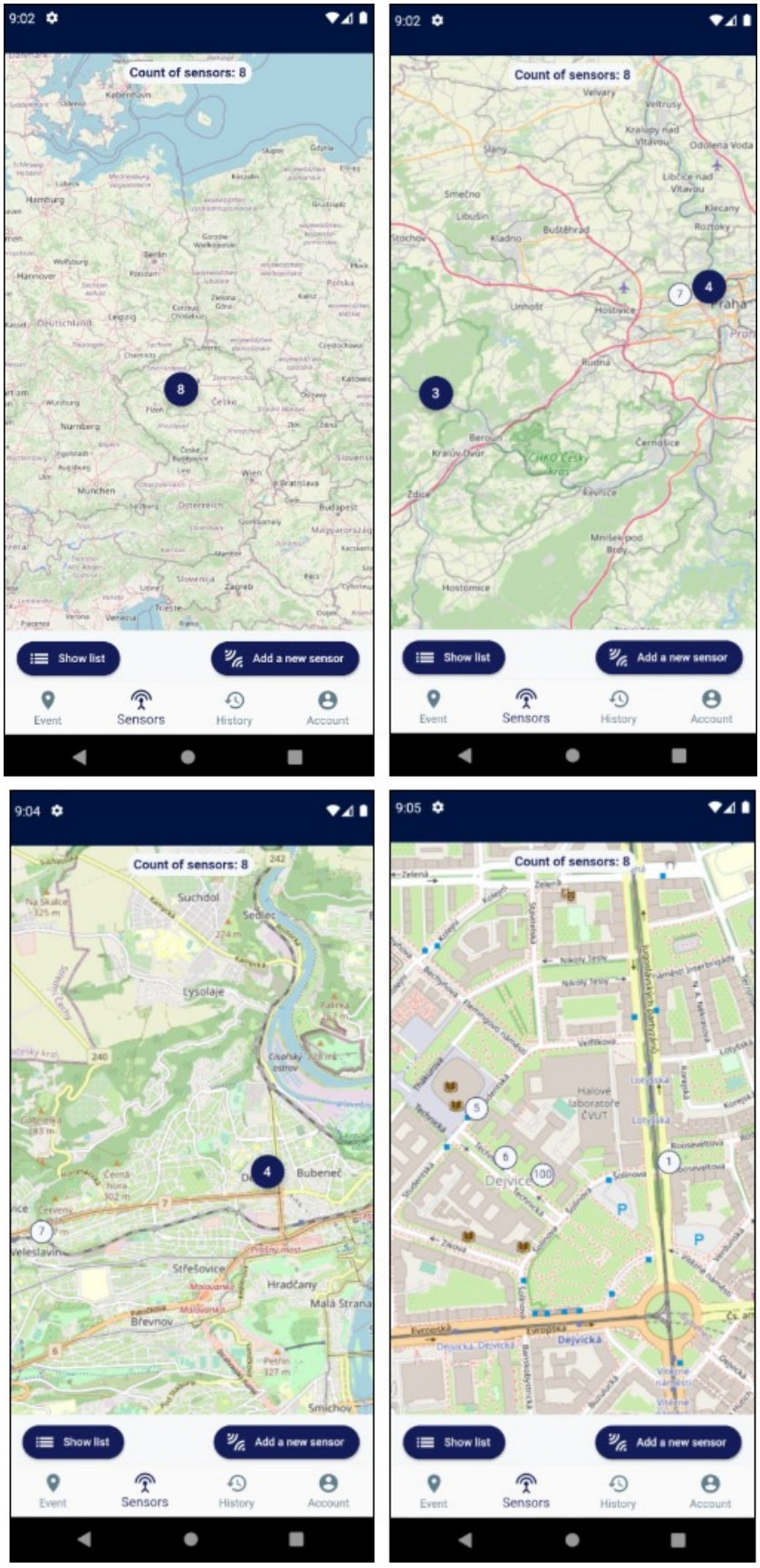
Úlohou tejto obrazovky je správa senzorov. Obrazovka je zložená zo zoznamu senzorov uložených v databáze (viď Obrázok 9.4). Senzor je reprezentovaný kartou, ktorá má ľavej časti zobrazené *ID* senzora. Stred zobrazuje údaje o polohe, na ktorej sa nachádza. Prvý riadok je približná adresa určená zo zemepisnej dĺžky a šírky, ktoré sa nachádzajú hneď pod približnou adresou. Výzor a funkčnosť spodnej časti obrazovky je závislá na právach používateľa. Ak má užívateľ plné práva, na spodnej časti obrazovky sú zobrazené dve tlačidlá. Tlačidlo na ľavej strane slúži na zobrazenie senzorov ma mape. Funkcia pravého tlačidla umožňuje používateľovi pridať nový senzor do databázy a po jeho stlačení je používateľovi zobrazená obrazovka slúžiaca na pridanie nového senzoru. Ak používateľ nemá všetky práva, tlačidlo na pravej strane sa nezobrazí. To čo platí pre pridávanie senzorov, platí aj pre ich odstraňovanie. Používateľ s plnými právami je schopný senzor zmazať potiahnutím karty reprezentujúcej daný senzor do ľavej strany (viď Obrázok 9.4). Následne je používateľovi zobrazené dialogové okno



Obrázok 9.4. Obrazovka zobrazujúca zoznam senzorov.

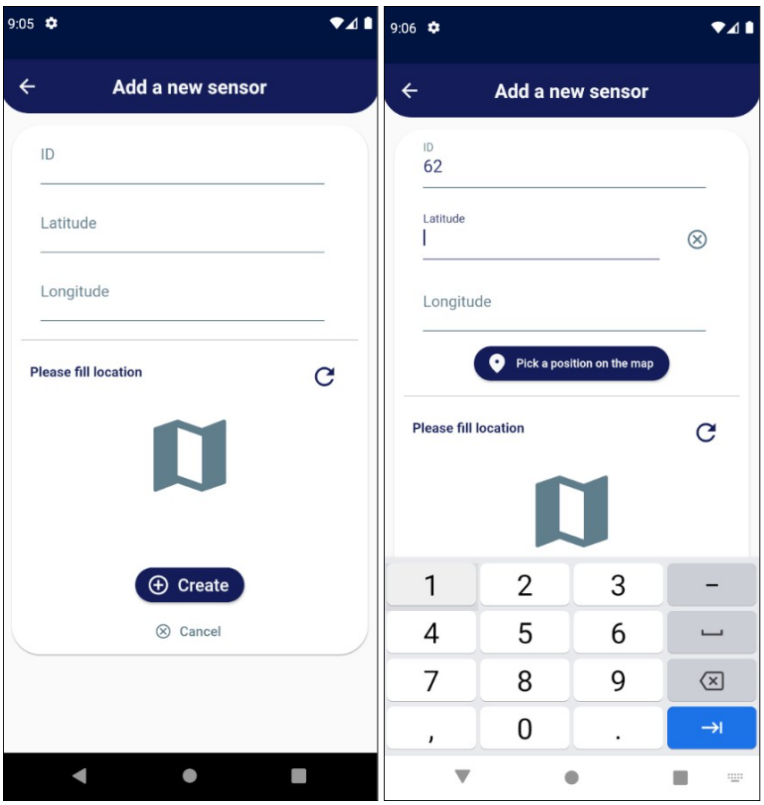
s otázkou, či si naozaj praje odstrániť senzor a tým aj všetky údaje o ňom. Toto zamedzí nechcenému odstráneniu senzora.

Tlačidlo na ľavej strane slúži na zobrazenie mapy so senzormi. Mapu je možné ľubovoľne približovať, oddiaľovat a podľa toho sú zobrazené informácie o senzoroch (viď Obrázok 9.5). Ak je mapa dostatočne priblížená, je možné vidieť jednotlivé senzory, ktoré sú reprezentované bielym kruhom s modrým ohraničením. V prípade, že je v danej lokalite viac senzorov, spoločne vytvárajú modrý kruh s počtom senzorov. Modrou farbou je zobrazené aj *ID* senzora v bielom kruhu (1, 5, 6, 100). V hornej časti mapy je zobrazený celkový počet senzorov v databáze.



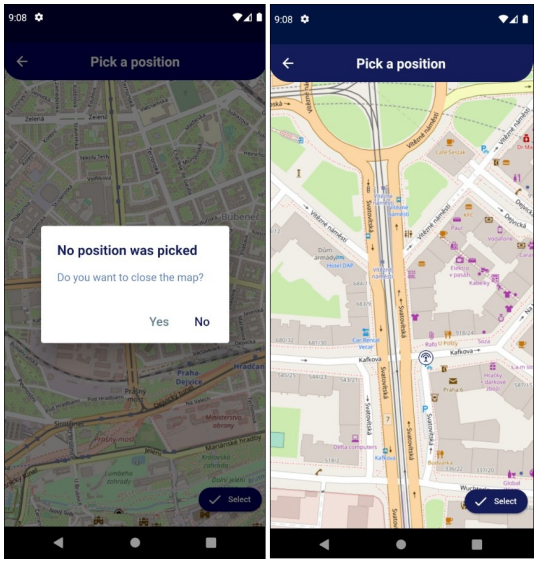
Obrázok 9.5. Obrazovka zobrazujúca senzory na mape s rôznym priblížením.

Tlačidlo na pravej strane je zobrazené iba používateľom so všetkými právami. Kliknutím je otvorená obrazovka na pridávanie nového senzoru (viď Obrázok 9.6). Obrazovka je tvorená z troch riadkov, slúžiacich na zadanie potrebných údajov o novom senzore. V druhej polovici obrazovky je po zadaní zemepisnej dĺžky a šírky zobrazená daná poloha na mape. Pokiaľ je názov ulice známy, je zobrazený v hornom ľavom rohu mapy. Ak nie je vyplnená zemepisná šírka alebo dĺžka, nie je zobrazený názov ulice. Pod mapou sú dve tlačidlá pod sebou - *Create* a *Cancel*. Tlačidlo *Cancel* zatvorí obrazovku a používateľ je presmerovaný späť na zoznam senzorov. Po stlačení tlačídla *Create* nastane kontrola zadaných údajov. Pokiaľ niektorý z údajov nie je správny, používateľ je upozornený červeným textom pod riadkom, obsahujúcim chybný údaj. Ak kontrola nezistí žiadny problém, používateľ je presmerovaný na zoznam senzorov. Tam je zobrazené informatívne okno, kde sú zhrnuté informácie o novom pridanom senzore.



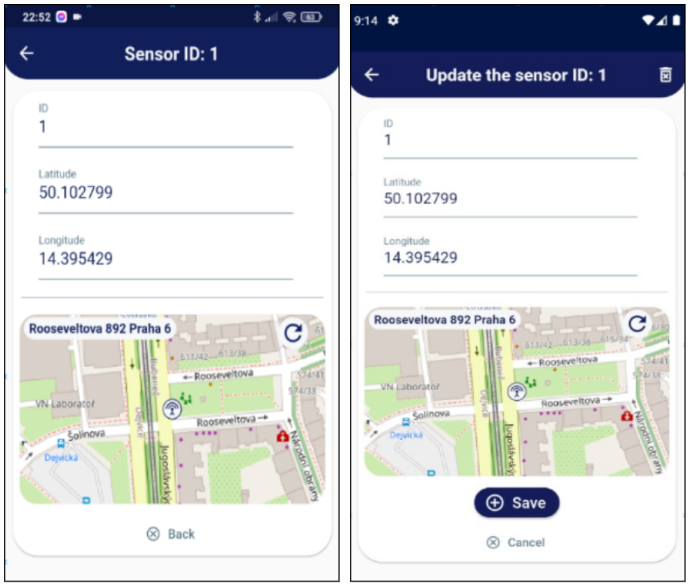
Obrázok 9.6. Obrazovka zobrazujúca pridávanie nového senzoru.

Pri kliknutí na riadok, slúžiací na zadanie zemepisnej dĺžky alebo šírky, je zobrazené dočasné tlačidlo umožňujúce zadanie polohy výberom miesta priamo na mape. Po kliknutí je zobrazená mapa s aktuálnou polohou používateľa. Výber polohy na mape používateľ potvrdí pomocou tlačidla *Select* v pravom dolnom rohu. Pokiaľ používateľ nevyberie žiadnu polohu a iba stlačí tlačidlo *Select*, je zobrazené dialogové okno, ktoré ho informuje o tom, že nebola vybraná žiadna poloha (viď Obrázok 9.7). V prípade, že je poloha zvolená, po stlačení je používateľ presmerovaný späť na obrazovku, slúžiacu na pridávanie senzorov. Riadky určené na zadávanie zemepisnej šírky a dĺžky sú vyplnené podľa zvolenej polohy, ktorá je takisto zobrazená na mape.



Obrázok 9.7. Obrazovka zobrazujúca výber polohy nového senzoru na mape.

Kliknutím na senzor na mape (viď Obrázok 9.5) alebo na senzor v zozname (viď Obrázok 9.4) je otvorená obrazovka, reprezentujúca detail vybraného senzoru. Vzhľad obrazovky opäť závisí na právach používateľa (viď Obrázok 9.8). Ak má obmedzené práva, obrazovka má čisto informatívny charakter a používateľ nemôže zmeniť žiadny údaj - *ID*, zemepisná dĺžka a šírka. Používateľ so všetkými právami vidí obrazovku s úpravou vybraného senzora. Používateľ má tým pádom možnosť upraviť všetky údaje. Tieto údaje sú po kliknutí tlačidla *Save* uložené a do pár sekúnd aj prepísané v databáze. Táto zmena sa takmer hneď zobrazí iným používateľom. Po zmene zemepisnej dĺžky alebo šírky je automaticky zobrazená nová poloha. V pravom hornom rohu mapy sa nachádza tlačídlo, ktorého úlohou je aktualizácia mapy. Senzor môže byť odstránený kliknutím na tlačidlo s ikonou koša v pravom hornom rohu - len v prípade, že má používateľ všetky práva.



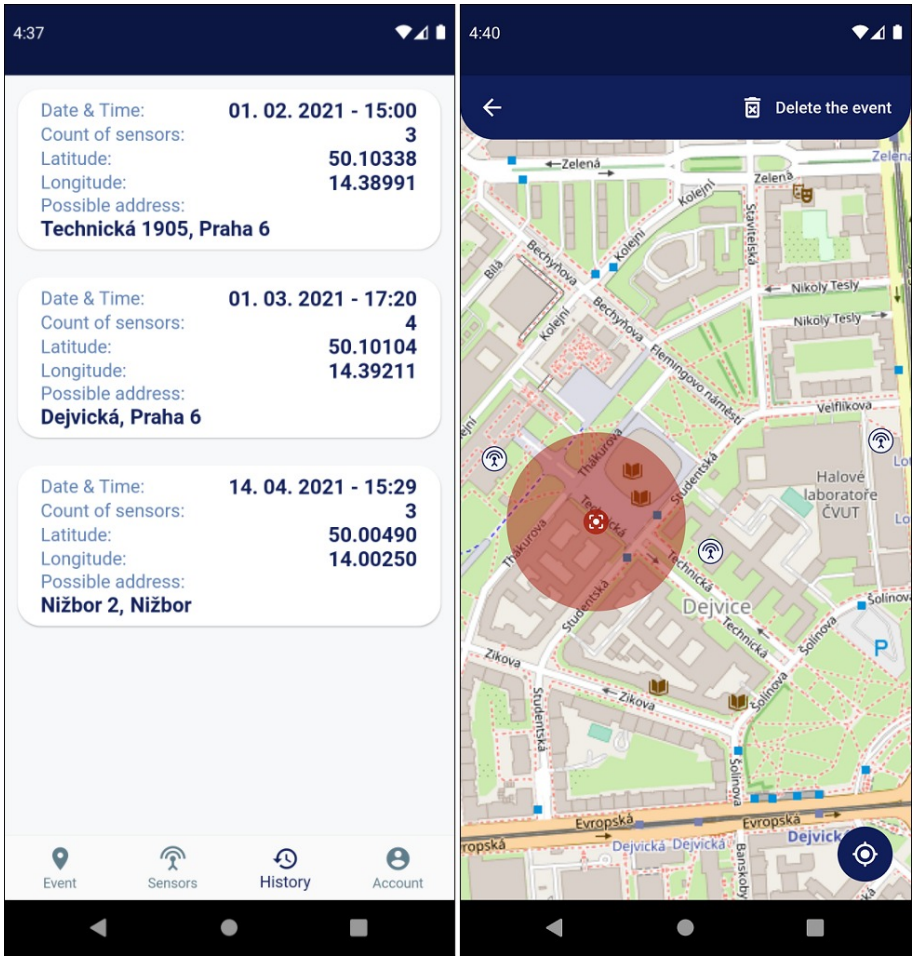
Obrázok 9.8. Obrazovka zobrazujúca detail senzoru.

9.4 Obrazovka *History*

Obrazovka *History* funguje ako archív predchadzajúcich udalostí. Na obrazovke je možné vidieť zoznam udalostí, ktoré sú do neho pridávané uzavretím udalosti na obrazovke *Event* (viď Obrázok 9.3). Kokrétna udalosť v zozname je reprezentovaná rovnako, ako na obrazovke *Event* (viď Kapitola 9.2), kartou obsahujúcou najdôležitejšie informácie:

- dátum a čas zaznamenania udalosti,
- počet senzorov, ktoré zaregistrovali udalosť,
- zemepisná šírka a dĺžka, podľa ktorých je určená približná adresa.

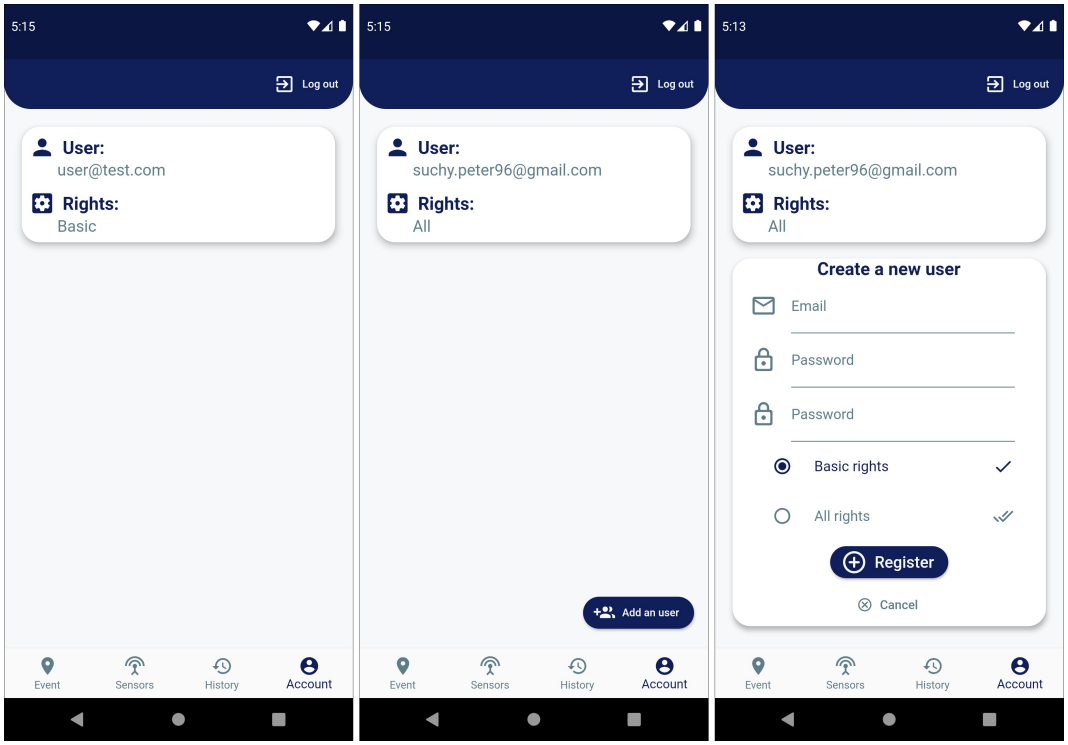
Kliknutím na ľubovolnú udalosť zo zoznamu je používateľ presmerovaný na obrazovku s detailom udalosti. Je zložená z mapy, vycentrovanej na údajny zdroj udalosti, označený červenou ikonou. Na mape sú takisto zobrazené akustické senzory, ktoré zaznamenali túto udalosť (viď Obrázok 9.9). Používateľ môže ľubovoľne pohybovať s mapou. Tlačidlom v pravom dolnom rohu vycentruje mapu tak, aby bol zdroj udalosti v strede mapy.



Obrázok 9.9. Obrazovka *History* a detail jednej z udalostí.

9.5 Obrazovka *Account*

Funkcia tejto obrazovky je opäť závislá od práv. Používateľ, ktorý nemá všetky práva, túto obrazovku zrejme nijako nevyužije. Naopak, používateľ so všetkými právami je schopný na tejto obrazovke vytvárať nových používateľov. Obrazovka v hornej časti zobrazuje *email* používateľa, pod ktorým je prihlásený a tiež informuje o stave jeho práv - základné alebo úplne. Pokiaľ nemá používateľ obmedzené práva, v dolnej časti obrazovky je zobrazené tlačidlo na vytvorenie nového používateľského účtu. Po kliknutí na tlačidlo sa zobrazí karta s registračným formulárom (viď Obrázok 9.10).



Obrázok 9.10. Obrazovka *Account* a vytváranie nového používateľského účtu.

Kapitola 10
Implementácia API

Aplikácia, pomenovaná **Acoustic Event Detector API**, bola vytvorená v jednom z najznámejších IDE na vývoj JVM aplikácií - *IntelliJ IDEA Community Edition*. API je naprogramované pomocou jazyka *Kotlin*, ktorý bol tiež použitý na konfiguráciu *build* nástroja *Gradle*. Táto aplikácia nie je štandardné REST API (viď Kapitola 6.1.2), pretože nie sú implementované všetky štyri základné funkcie. Hlavnou úlohou je spracovanie novej udalosti a odoslanie notifikácie o nej. Toto API je pomerne jednoduchá Spring Boot aplikácia (viď Kapitola 10.1).

Z aplikácie je vytvorený **.jar** súbor, ktorý je následne spustiteľný pomocou príkazového riadku. Aplikácia je spustiteľná s minimálne jedným argumentom, ktorý reprezenuje cestu k súboru na identifikovanie daného Firebase projektu, ako vidieť na nasledujúcom príkaze:

```
java -jar acoustic_event_detector-1.0.0.jar aed-firebase-admin-sdk.json
```

Vďaka tomuto súboru je aplikácia schopná inicializovať Firebase Admin SDK, pomocou ktorého posiela notifikácie používateľom. Predvolený port, na ktorom je spustená aplikácia, je 8080. To samozrejme nemusí vyhovovať každému. Preto je možné pomocou druhého argumentu zmeniť tento port. Po spustení API a úspešnom inicializovaní Firebase projektu nasleduje vždy pokus o vytvorenie administrátorského účtu - pokiaľ už existuje, je vypísaný text, ktorý o tomto stave informuje. Toto zaručí, že vždy existuje aspoň jeden používateľský účet so všetkými právami. Potrebné je to z dôvodu, že aplikácia nie je určená pre širokú verejnosť, takže počet používateľov je obmedzený.

10.1 Spring Boot

Platforma *Spring* zrýchľuje a zjednodušuje vývoj Java aplikácií. Obsahuje veľké množstvo modulov, ktoré slúžia na rešenie určitých záležitostí ako bezpečnosť, komunikácia s databázou a iné webové služby.[29] Tento projekt patrí medzi kľúčové v rámci framworkov pre vývoj webových aplikácii. [30]

Spring Boot zastrešuje najpoužívanejšie súčasti prostredia Spring, umožňujúc tak celkové zjednodušenie a urýchlenie vývoja ľubovolnej Spring aplikácie tým, že jej konfigurácia základných prvkov je vytvorená automaticky. Zároveň tým vývojár získava spustiteľnu aplikáciu, ktorá spĺňa logiku a funkčnosť podľa najpoužívanejších princípov pri tvorbe komplexnej Spring aplikácie. [20]

10.2 Spracovanie a notifikácia udalosti

Na spracovanie udalosti bola implementovaná trieda s menom *EventController*, ktorá reaguje iba na dva *endpoint*-y. Prvý z nich je k dispozícii v prípade, že existuje už spracovaná udalosť - to znamená, že poloha zdoja už je známa a od API je potrebná iba notifikácia. URL adresa endpointu v tomto prípade je **IP-ADDRESS : PORT/newFinalEvent**.

Typ requestu - žiadosti musí byť **POST**, v hlavičke musí byť informácia o tom, že sa bude jednať o JSON. Dáta v žiadosti reprezentujú udalosť. Formát dát je JSON, kde je špecifikovaná zemepisná šírka a dĺžka zdroja udalosti a následne zoznam polôh senzorov, ktoré zaznamenali danú udalosť. Nepovinný údaj je dátum a čas, v ktorom bola táto udalosť zaznamenaná. Formát údaju je **yyyy-MM-dd hh:mm:ss**. Ak dáta neobsahujú tuto informáciu, použije sa aktuálny čas a dátum. Odoslaná odpoveď po prijatí správneho requestu je nasledujúci text:

```
{"Message" : "Ok"}
```

Príklad validného POST requestu na prvý endpoint.

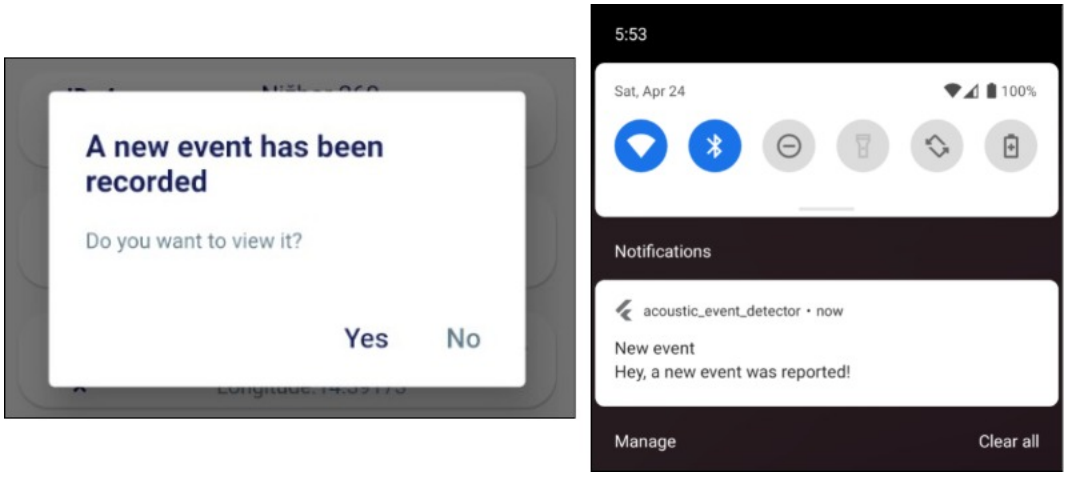
```
curl -X POST --location "http://localhost:8080/newFinalEvent" \
    -H "Content-Type: application/json" \
    -d "{
          \"event\": {
            \"latitude\": 50.10338,
            \"longitude\": 14.38991,
            \"timestamp\": \"2021-02-01 15:00:35\"
          },
          \"sensors\": [
            {
              \"lat\": 50.103082,
              \"lon\": 14.391701
            },
            {
              \"lat\": 50.104194,
              \"lon\": 14.394366
            },
            {
              \"lat\": 50.104012,
              \"lon\": 14.388319
            }
          ]
        }"
```

Druhý endpoint je používaný v prípade neznámej polohy zdroja danej udalosti. URL adresa requestu je **IP-ADDRESS : PORT/newEvent**. Tento request musí byť opäť typu POST a obsahovať rovnakú hlavičku, ako prvý request. Po spracovaní dát je vypočítaný zdroj udalosti. Správne dáta majú formát JSON a sú reprezentované zoznamom senzorov. Senzor je zložený z dvoch údajov - ID a čas, v ktorom daná udalosť bola zaznamenaná. Na základe ID sú potom polohy príslušných senzorov použité vo výpočte zdroja pomocou triedy na to určenej (viď Kapitola 10.3). Odpoveď po requeste so správnymi dátami je rovnaká, ako v prvom prípade. Príklad validného POST requestu na druhý endpoint:

```
curl -X POST --location "http://localhost:8080/newEvent" \
```

```
    -H "Content-Type: application/json" \
    -d "{
          \"report\": [
            {
              \"id\": 4,
              \"time\": 6698
            },
            {
              \"id\": 2,
              \"time\": 6698
            },
            {
              \"id\": 3,
              \"time\": 6689
            }
          ]
        }"
```

Používatelom mobilnej aplikácie je po spracovaní udalosti odoslaná notifikácia. Kým má používateľ otvorenú aplikáciu, je po prijatí notifikácii zobrazené dialógové okno, ktoré ho informuje o novej udalosti. Po kliknutí tlačidla *Yes* je používateľ presmerovaný na obrazovku *Event*, kde uvidí novú udalosť. Pokiaľ je aplikácia spustená na pozadí alebo nie je vôbec zapnutá, dostane používateľ notifikáciu ako pri inej aplikácii-v hornej časti obrazovky (viď Obrázok 10.1). Po kliknutí na notifikáciu je používateľovi zobrazená obrazovka *Event*.



Obrázok 10.1. Notifikácia o novej udalosti.

Ak by neboli dostupné servery na FCM, môže sa stať, že je udalosť spracovaná, no notifikácia odoslaná nebola. V tomto prípade odpoveď na request na neodoslanie notifikácie upozorní:

```
{"Message" : "Notification was not send"}
```

Na použitie druhého endpointu je potrebné v requeste uviesť aspoň tri senzory, inak výpočet neprebehne a odpoveď od API obsahuje správu, v ktorej o tom informuje:

```
{"Message" : "Not enough sensors"}
```

10.3 Výpočet polohy zdroja

Celý výpočet je rozšírením voľne dostupného balíku menom *Trilateration* [1]. Tento balík obsahuje implementáciu výpočtu nelineárnych najmenších štvorcov pomocou Levenberg-Marquardtovej metódy. Celý výpočet je implementovaný v triede *Calculator*, ktorá má jednu metódu *getCenter()*. Táto metóda predstavuje výpočet polohy zdroja. Na výpočet je použitá trieda *NonLinearLeastSquaresSolver*, ktorá dostane ako argumenty funkciu a optimizáčnú metódu, ktoré su reprezentované príslušnými triedami.

```
val func = MultilaterationFunction(positions, distances, referencePoint)
val optimizer = LevenbergMarquardtOptimizer()
val solver = NonLinearLeastSquaresSolver(function, optimizer)

val optimum = solver.solve()
```

Premenná *optitmum* obsahuje vypočítané súradnice zdroja a štandardnú odchýlku. Pokiaľ by súradnice neboli v potrebnom intervale, čo je pre zemepisnú šírku interval -180.0 až +180.0 a pre zemepisnú dĺžku interval -90.0 až +90.0, sú pomocou vypočítanej odchýlky upravené.

Trieda *MultilaterationFunction* dedí zo základnej funkcie, ktorá bola implementovaná v balíku *Trilateration* a reprezentuje vlastnú implementáciu pre výpočet Jakobiho matice a hodnotu funkcie. Použitá funkcia je definovaná v kapitole 8.1.1. Jakobiho matica je matica parciálných derivacií tejto funkcie.

Kapitola 11
Záver

Zámerom tejto práce bol návrh a implementácia mobilnej aplikácie pre operačný systém Android. Aplikácia bola vytvorená pomocou frameworku Flutter, kvôli jednoduchej spustiteľnosti aplikácie aj na operačnom systéme iOS v prípade, že bude v budúcnosti potrebná.

Aplikácia slúži na reprezentáciu zaznamenaných akustických udalostí, ktoré boli zachytené detektormi - čidlami s mikrofónom. Umožňuje vytváranie nových používateľských účtov s rôznymi pravidlami. Momentálne sú implementované dve možnosti - všetky práva alebo iba základné. Práva ovplyvňujú rozsah funkčnosti aplikácie. Autentifikácia a real-time databáza je riešená pomocou služieb platformy Firebase. Databáza má široké využitie - od ukladania práv pre daného uživateľa až po archiváciu zaznamenaných akustických udalostí, ktoré slúžia na spätné prezeranie. Takisto je využitá na uloženie dôležitých informácii o senzoroch ako je ID jednotlivého senzora a jeho poloha - zemepisná šírka a dĺžka. Táto poloha je dôležitá pri lokalizácii zdroja akustickej udalosti.

Spolu s aplikáciou bolo implementované API, ktoré slúží na spracovanie akustickej udalosti a následnú notifikáciu používateľov o tejto skutočnosti. Api je schopné spracovať udalosť dvoma spôsobmi:

- udalosť už obsahuje polohu zdroja a polohu všetkých senzorov, ktoré udalosť zaznamenali,
- poloha zdroja nie je známa a je potrebné ju vypočítať.

Presná reprezentácia dát, ktoré sú spracovateľné je v kapitole 10.2. Po úspešnom spracovaní sú používatelia upozornení notifikáciou, ktorá je riešená pomocou služby Firebase Cloud Messaging. Testovanie aplikácie v reálnej prevádzke nebolo možné z dôvodu globálnej pandémie ochorenia COVID-19. Štátom nariadené opatrenia neumožňovali voľný pohyb po verejnom priestranstve, takže testy nebolo možné vykonať.

Aplikácia spolu s API vytvára základ pre komplexnejšiu aplikáciu, ktorá nie je závislá na konkrétnych senzoroch. Možné zlepšenie by bolo pridanie možnosti meniť práva používateľov nie len pri vytváraní daného účtu. Ďalšou dôležitou informáciou by bolo zobrazovať aktuálny stav batérie v jednotlivých senzoroch.

Literatúra

[1] *Trilateration* [online]. [cit. 2021-04-13].
https://github.com/lemmingapex/trilateration#trilateration

[2] *Kotlin Docs* [online]. [cit. 2021-04-13].
https://kotlinlang.org/docs/home.html

[3] *Open Handset Alliance:Alliance Members* [online]. [cit. 2021-03-13].
https://www.openhandsetalliance.com/oha_members.html

[4] *Google Blog: Turning it up to Android 11* [online]. [cit. 2021-03-13].
https://blog.google/products/android/android-11/

[5] *Statista: Mobile operating system's market share worldwide from January 2012 to January 2021* [online]. [cit. 2021-03-18].
https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

[6] *Statista: Market share of mobile operating systems in Czechia from 2010 to 2020* [online]. [cit. 2021-03-18].
https://www.statista.com/statistics/669587/market-share-mobile-operating-systems-czech-republic/

[7] *Android Developers: Platform Architecture* [online]. [cit. 2021-03-22].
https://developer.android.com/guide/platform

[8] *Android Source: Android Runtime (ART) and Dalvik* [online]. [cit. 2021-03-22].
https://source.android.com/devices/tech/dalvik

[9] *Android Developers: Package Index* [online]. [cit. 2021-03-25].
https://developer.android.com/reference/packages

[10] *Native vs Hybrid vs Cross-Platform — What To Choose?* [online]. [cit. 2021-03-26].
https://medium.com/flutterdevs/native-vs-hybrid-vs-cross-platform-what-to-choose-3221130f7cc5

[11] *Dart overview* [online]. [cit. 2021-03-30].
https://dart.dev/overview

[12] Martin Sikora. *Dart Essentials*. Packt, 2015. ISBN 978-1-78398-960-7

[13] Frank Zammetti. *Practical Flutter: Improve your Mobile Development with Google's Latest Open-Source SDK*. Apress, 2019. ISBN 978-1-4842-4971-0

[14] *Flutter architectural overview* [online]. [cit. 2021-03-30].
https://flutter.dev/docs/resources/architectural-overview

[15] *Hackernoon: What's Revolutionary about Flutter* [online]. [cit. 2021-04-02].
https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514

[16] *Apps take flight with Flutter* [online]. [cit. 2021-03-30].
https://flutter.dev/showcase

[17] *Basic widgets* [online]. [cit. 2021-04-05].
https://flutter.dev/docs/development/ui/widgets-intro

[18] *Widget State* [online]. [cit. 2021-04-05].
https://flutter.dev/docs/resources/architectural-overview#state-management

[19] *Platform channels* [online]. [cit. 2021-04-02].
https://flutter.dev/docs/resources/architectural-overview

[20] Ludovic Dewailly *Building a RESTful Web Service with Spring*. Packt, 2015. ISBN 978-1-78528-571-4

[21] *Application Programming Interface (API)* [online]. [cit. 2021-04-06].
https://www.ibm.com/cloud/learn/api

[22] *PPC Investigative Report: What Are APIs?* [online]. [cit. 2021-04-06].
https://www.ppchero.com/what-an-api-is-and-how-it-can-enhance-ppc/

[23] *Firebase: Learn the fundamentals* [online]. [cit. 2021-04-8].
https://firebase.google.com/docs

[24] *Exploring Firebase (Setup Auth)* [online]. [cit. 2021-04-8].
https://dev.to/subashkarthik/exploring-firebase-setup-auth-4aio

[25] *Firebase Cloud Messaging* [online]. [cit. 2021-04-18].
https://firebase.google.com/docs/cloud-messaging

[26] *FCM Architectural Overview* [online]. [cit. 2021-04-18].
https://firebase.google.com/docs/cloud-messaging/fcm-architecture

[27] *Multilateration in 2D* [online]. [cit. 2021-04-10].
https://github.com/jurasofish/multilateration

[28] *Bloc: Architecture* [online]. [cit. 2021-04-10].
https://bloclibrary.dev/#/architecture

[29] *Spring Boot* [online]. [cit. 2021-04-13].
https://spring.io/projects/spring-boot

[30] *Java* [online]. [cit. 2021-04-18].
https://www.jetbrains.com/lp/devecosystem-2020/java/

Príloha A
Skratky

OS	Operačný systém - softvér, ktorý spravuje počítač a poskytuje programom rozhranie na prístup k rôznym perifériam.
API	Application programming interface - rozhranie, vďaka ktorému môžu medzi sebou komunikovať dve aplikácie.
JSON	JavaScript Object Notation - formát pre uloženie a prenos dát, používa sa napr. pri posielaní dát zo servera na web. Podstata je, aby dáta mali jednoduchý formát, no zároveň vhodný formát
HTTP	Hypertext Transfer Protocol - protokol na komunikáciu so webovými servermi.
TDoA	Time difference of arrival - algoritmus multilaterácie, ktorého princíp je založený na rozdieloch času, kedy zaznemenali nejakú udalosť
SDK	Software development kit - nástroje, ktoré uľahčujú prácu vývojarom.
JVM	Java virtual machine - Virtuálny stroj, ktorý umožňuje počítaču spúšťať programy v jazykoch, ktoré sú kompilované do Java bytecode.
Android-NDK	Android native development kit - nástroje, ktoré umožňujú tvoriť Android aplikáciu v jazykoch C/C++.
GPS	Global positioning system - navigačný systém používaný na zistenie presnej polohy na Zemi.
HTML	HyperText Markup Language - značkovací jazyk určený na vytváranie webových stránok.
CSS	Cascading Style Sheets - technológia na úpravu výzoru elementov vytvorených pomocou HTML.
IDE	Integrated development environment - vývojove prostredie, ktoré uľahčuje prácu vývojarovi
URL	Uniform Resource Locator - univerzálny formát mien používaný na označenie zdroja na internete. Definuje doménovú adresu servera, umiestnenie zdroja na serveri a protokol, ktorým je možné k zdroju pristupovať

Príloha B
Obsah priloženého CD

aplikacia.zip	Zdrojový kód k mobilnej aplikácii
api.zip	Zdrojový kód k API
BP_SuchyPeter.pdf	Text bakalárskej práce