**Monitorizarea poluării fonice într-un oraș utilizând sisteme de detectare oportuniste**

LUCRARE DE LICENŢĂ

|  |  |  |
| --- | --- | --- |
|  | Absolvent: | **Aurel-Ioan Vidrean** |
|  |  |  |
|  | Coordonator științific: | **Conf. Dr. Ing. Bogdan Iancu** |

**2023**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | |  | |
| DECAN, |  | | DIRECTOR DEPARTAMENT, | |
| **Prof. dr. ing. Liviu MICLEA** |  | **Prof. dr. ing. Rodica POTOLEA** | |

Absolvent: **Aurel-Ioan Vidrean**

**Monitorizarea poluării fonice într-un oraș utilizând sisteme de detectare oportuniste**

1. **Enunțul temei:** *Dezvoltarea unei aplicații mobile ce să permite detecția persoanelor și autovehiculelor, alături de poluarea fonică, cu ajutorul senzorilor încorporați ai dispozitivului mobil.*
2. **Conținutul lucrării:** *(enumerarea pârților componente) Exemplu: Pagina de prezentare, aprecierile coordonatorului de lucrare, titlul capitolului 1, titlul capitolului 2,… titlul capitolului n, bibliografie, anexe.*
3. **Locul documentării**: *Exemplu*: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare
4. **Consultanți**:
5. **Data emiterii temei:** 1 noiembrie 2022
6. **Data predării:** 8 iulie 2023

|  |  |
| --- | --- |
| Absolvent: | Vidrean Aurel-Ioan |
|  |  |
| Coordonator științific: | Conf. Dr. Ing. Bogdan Iancu |

**Declarație pe propria răspundere privind**

**autenticitatea lucrării de licență**

Subsemnatul(a) **Vidrean Aurel-Ioan**, legitimat(ă) cu CI seria SB nr. 849248  
CNP 5000616323525, autorul lucrării „Monitorizarea poluării fonice într-un oraș folosind sisteme de detectare oportuniste” elaborată în vederea susținerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea Tehnologia Informației din cadrul Universității Tehnice din Cluj-Napoca, sesiunea iulie a anului universitar 2023, declar pe propria răspundere că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării, și în bibliografie.

Declar că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

În cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

|  |  |  |
| --- | --- | --- |
| Data  04.07.2023 |  | Nume, Prenume  Vidrean Aurel-Ioan |
|  |  |  |
|  |  | Semnătura |

Cuprins

[Capitolul 1. Introducere – Contextul proiectului 1](#_Toc139022967)

[1.1. Dezvoltarea dispozitivelor mobile 1](#_Toc139022968)

[1.2. Detecția poluării fonice cu ajutorul dispozitivelor mobile 2](#_Toc139022969)

[Capitolul 2. Obiectivele proiectului 3](#_Toc139022970)

[2.1. Motivație 3](#_Toc139022971)

[2.2. Obiective 3](#_Toc139022972)

[2.2.1. Obiectivul principal 3](#_Toc139022973)

[2.2.2. Obiective Secundare 3](#_Toc139022974)

[2.2.3. Studiere soluțiilor existente 3](#_Toc139022975)

[2.2.4. Identificarea datelor necesare 3](#_Toc139022976)

[2.2.5. Testare ipoteze 4](#_Toc139022977)

[2.2.6. Implementarea soluției propuse 4](#_Toc139022978)

[2.2.7. Testare și evaluarea soluției propuse 4](#_Toc139022979)

[2.2.8. Interpretabilitate 4](#_Toc139022980)

[2.3. Provocări 4](#_Toc139022981)

[2.3.1. Senzori necalibrați 4](#_Toc139022982)

[2.4. Cerințe funcționale 5](#_Toc139022983)

[2.5. Cerințe non-funcționale 5](#_Toc139022984)

[2.5.1. Scalabilitatea 5](#_Toc139022985)

[2.5.2. Performanța 5](#_Toc139022986)

[Capitolul 3. Studiu bibliografic 6](#_Toc139022987)

[3.1. Metodologia măsurării zgomotelor 6](#_Toc139022988)

[3.1.1. Analiza caracteristicilor sunetului 6](#_Toc139022989)

[3.2. Analiza soluțiilor deja implementate 7](#_Toc139022990)

[3.2.1. EarPhone 7](#_Toc139022991)

[3.2.2. NoiseSpy 7](#_Toc139022992)

[3.3. Acuratețea și precizia măsurătorilor de zgomot 9](#_Toc139022993)

[3.4. Reglementari cu privire la măsurarea zgomotelor 11](#_Toc139022994)

[Capitolul 4. Analiză și fundamentare Teoretică 13](#_Toc139022995)

[4.1. Soluția propusa 13](#_Toc139022996)

[4.2. Prezentarea cazurilor de utilizare din cadrul sistemului 14](#_Toc139022997)

[4.3. TensorFlow în Android 15](#_Toc139022998)

[4.3.1. Preprocesarea datelor audio 15](#_Toc139022999)

[4.3.2. Augmentarea datelor și procesarea 15](#_Toc139023000)

[4.3.3. Integrarea TensforFlow în soluția propusa 15](#_Toc139023001)

[4.4. Localizarea în timp real în cadrul aplicației 16](#_Toc139023002)

[4.4.1. Cererea permisiunilor pentru locație 16](#_Toc139023003)

[4.4.2. Servicii de localizare și furnizori 16](#_Toc139023004)

[4.4.3. Integrarea localizării în timp real 17](#_Toc139023005)

[4.5. Calcularea numărului de decibeli cu ajutorul telefonului 18](#_Toc139023006)

[4.5.1. Preprocesarea datelor audio obținute 18](#_Toc139023007)

[4.5.2. Aplicarea filtrelor de pondere și conversia în dB(A) 18](#_Toc139023008)

[4.6. Integrarea Firebase 18](#_Toc139023009)

[4.6.1. Firebase Authentication 18](#_Toc139023010)

[4.6.2. Firebase RealTime Database 19](#_Toc139023011)

[4.6.3. Firebase Functions 20](#_Toc139023012)

[4.7. Arhitectura MVVM 20](#_Toc139023013)

[4.7.1. Model 20](#_Toc139023014)

[4.7.2. View 21](#_Toc139023015)

[4.7.3. ViewModel 21](#_Toc139023016)

[Capitolul 5. Proiectare de detaliu și implementare 22](#_Toc139023017)

[5.1. Descriere generala 22](#_Toc139023018)

[5.2. Crearea aplicației 22](#_Toc139023019)

[5.3. Arhitectura aplicației 25](#_Toc139023020)

[5.4. Navigarea între ecranele aplicației 26](#_Toc139023021)

[5.5. Implementarea metodelor pentru obținerea locației curente a utilizatorilor 29](#_Toc139023022)

[5.6. Implementarea metodelor pentru accesul la microfon și detecțiile efectuate 31](#_Toc139023023)

[5.6.1. Măsurarea numărului de decibeli 31](#_Toc139023024)

[5.7. Implementarea TensorFlow pentru detecția persoanelor și a mașinilor 34](#_Toc139023025)

[5.8. Implementarea fragmentelor din cadrul aplicației 36](#_Toc139023026)

[5.8.1. Implementarea MainFragment și a claselor auxiliare folosite pentru implementarea acestuia 36](#_Toc139023027)

[5.8.2. Implementarea DetailsFragment și a claselor auxiliare folosite pentru implementarea acestuia 38](#_Toc139023028)

[5.8.3. Implementarea ChartsFragment și a claselor auxiliare folosite pentru implementarea acestuia 40](#_Toc139023030)

[5.9. Scrierea și preluarea datelor din Firebase Realtime Database 42](#_Toc139023031)

[5.10. Integrarea Firebase Functions 45](#_Toc139023032)

[Capitolul 6. Testare și validare 47](#_Toc139023033)

[6.1. Testarea aplicației Android 47](#_Toc139023034)

[6.2. Testarea actualizării bazei de date în timp real 48](#_Toc139023035)

[6.3. Testarea detectiei TensorFlow 49](#_Toc139023036)

[Capitolul 7. Manual de instalare și utilizare 52](#_Toc139023037)

[7.1. Instalarea aplicației 52](#_Toc139023038)

[7.2. Mod de utilizare 52](#_Toc139023039)

[Capitolul 8. Concluzii 56](#_Toc139023040)

[8.1. Contribuții personale 56](#_Toc139023041)

[8.2. Analiza rezultatelor obținute 56](#_Toc139023042)

[8.3. Dezvoltări ulterioare 57](#_Toc139023043)

[Bibliografie 58](#_Toc139023044)

# Introducere – Contextul proiectului

## Dezvoltarea dispozitivelor mobile

Tehnologia dispozitivele mobile a revoluționat modul în care ne desfășuram activitățile și felul în care trăim, deoarece adoptarea dispozitivelor mobile a cunoscut o creștere semnificativa și rapida pe tot globul [1]. Telefoanele în sine au devenit instrumente ce facilitează comunicarea rapidă și o implicare colectivă. Diferite grupuri de persoane le folosesc în diverse moduri creative. Totodată, numărul senzorilor încorporați în telefoane și numărul aplicaților construite în jurul acestora a crescut vertiginos. În ultimii ani, telefoanele mobile au început să conțină senzori precum GPS, accelerometru, giroscop, microfon, camera, Bluetooth, etc. Astfel, diverse aplicații și servicii acoperă anumite necesitați precum accesul la informație, divertisment.

Omniprezenta dispozitivelor mobile și creșterea valorilor datelor generate de către senzori și aplicații oferta un nou domeniu de cercetare în științele comportamentale și sociale. Cercetătorii au început să examineze probleme în comportamentul utilizatorilor din perspectiva Big Data, folosind un volum mare de date pentru a caracteriza și înțelege diverse comportamente precum trăsături individuale, mobilitatea umana, comunicarea și modele de interacțiuni [2-4].

Oportunistic Sensing Systems (OSS) a devenit o arie de cercetare în creștere în știința calculatoarelor și inginerie. Sistemele utilizează dispozitive mobile și senzorii încorporați pentru a colecta, analiza și procesa datele în timp real, astfel oferind o nouă metodă de a monitoriza mediul înconjurător, evenimentele ce se petrec în jur și pentru a oferi diverse informații către utilizatori.

Creșterea preocupării pentru impactul poluării fonice asupra sănătății și bunăstării a dus la dezvoltarea a numeroase aplicații ce monitorizează mediul înconjurător, incluzând sisteme fixe sau mobile de monitorizare a măsurării zgomotului. Poluarea fonică este datorata mai multor factori precum transportul și activitățile industriale. Aceasta afectează atât mediul înconjurător (flora și fauna) cât și persoanele ce sunt expuse în mod repetat la acest fenomen, iar din păcate în orașe acest lucru se resimte din ce în ce mai mult. Expunerea îndelungată la poluarea fonică poate cauza pierderea auzului, hipertensiune și tulburări de somn. Totodată poate fi afectata și sănătatea mintală cauzând astfel stres, anxietate, chiar și depresie. Pe de alta parte acest fenomen poate afecta și calitatea vieții din anumite zone, deoarece fiind amplasate în locuri unde zgomotul este prezent constant vor exista pierderi din punct de vedere al prețurilor locuințelor în zonele respective și activitatea economica poate fi afectată. Pentru a combate efectele negative ale poluării fonice pot fi luate mai multe măsuri precum bariere de zgomot, izolarea sunetului, reducerea zgomotului produs de transport și de activitățile industriale. Acest lucru se poate realiza cu ajutorul unor sisteme ce măsoară concentrațiile de zgomot din diferite zone ale întinderilor urbane. Din păcate, aceste sisteme sunt foarte costisitoare, necesită o infrastructură foarte mare și totodată nu sunt ușor de accesat pentru public. Aici intervine o soluție și anume integrarea unor astfel de sisteme la îndemâna oamenilor chiar pe dispozitivele lor mobile. Dispozitivele mobile, precum smartphone-uri și tablete, oferă o alternativa accesibila din punct de vedere al costurilor pentru monitorizarea mediului înconjurător.

## Detecția poluării fonice cu ajutorul dispozitivelor mobile

Acest proiect se focusează pe implementarea unei aplicații Android care să funcționeze pe post de Oportunistic Sensing System. Aplicația o să poată detecta nivelul de poluare fonică și prezența persoanelor și a mașinilor din jur în timp real folosind microfonul încorporat al dispozitivului. Proiectul urmărește să atragă atenția asupra impactului pe care poluarea fonică o are asupra sănătății și astfel oferind utilizatorilor date în timp real despre mediul înconjurător. Pentru funcționare aplicația are nevoie de microfonul și GPS-ul încorporat al dispozitivului pentru a colecta date despre mediu și pentru a le procesa ulterior prin anumiți algoritmi de machine learning.

Acestea sunt echipate cu o varietate de senzori, incluzând microfon și GPS, ce pot fi folosiți pentru a monitoriza mediul înconjurător. Pentru a face asta trebuie prelucrate datele ce survin de la aceste dispozitive, centralizate, prelucrate, iar apoi oferindu-se raporturi amănunțite în urma analizei făcute în acest scop, raport public pentru orice utilizator al unei astfel de aplicații.

Contextul proiectului este de a implementa o aplicație de tipul Oportunistic Sensing Systems care să ruleze pe Android. Fiind un sistem open-source ce facilitează construirea de diverse aplicații și fiind la scara mondiala cel mai răspândit sistem de operare în rândul dispozitivelor mobile am ales să implementez o astfel de soluție pentru a combate problema descrisă mai sus.

Pentru realizarea obiectivului trebuie luate în considerare următoarele aspecte. Senzorii dispozitivelor mobile sunt calibrați într-un anume fel și trebuie să existe o calibrare a acestora pentru a interpreta cât mai corect datele colectate.

Astfel, pentru a ajunge la acest rezultat trebuie să fie implementați următorii pași. În primul rând arhitectura aplicației și interfața pentru utilizator trebuie să fie concepute și implementate. Alături de acestea, trebuie integrați anumiți algoritmi pentru a procesa datele, lucruri ce urmează a fi menționate în următoarele capitole.

# Obiectivele proiectului

În acest capitol se prezinta în detaliu motivația care a dus la alegerea acestei teme, obiectivul principal și o serie de obiective secundare, cerințele funcționale și non-funcționale care trebuie îndeplinite în urma implementării soluției propuse.

## Motivație

Modul accelerat în care urbanizarea se dezvolta peste tot a determinat ca poluarea fonică să devina o problema prevalenta și detrimentală. Pe măsură ce orașele cresc și dezvolta, poluarea fonică își face și ea simțită prezența. De aceea o aplicație care să monitorizeze și să înregistreze poluarea fonică este necesară pentru bunăstarea comunităților.

## Obiective

### Obiectivul principal

Obiectivul principal este dezvoltarea unei aplicații care permite **detecția** poluării fonice la nivel local, **înregistrarea** valorilor detectate și **indicarea** valorilor din toate zonele aflate sub investigare. În același timp, dorim ca prin intermediul acestei aplicații să putem detecta și concentrația autovehiculelor aflate în zona și a persoanelor de asemenea, astfel oferind o aplicație mobilă, portabilă și cu un grad de performanta cât mai ridicat.

### Obiective Secundare

Obiectivele secundare pe care aș dori să le ating pe parcursul dezvoltării aplicației sunt: studierea soluțiilor existente deja, definirea modului în care se va efectua analiza datelor, definirea consumului de energie necesar pentru prelucrarea datelor, analiza pe device-uri low-end și high-end.

### Studiere soluțiilor existente

Identificare și înțelegerea efectivității și limitărilor abordărilor curente duce la înțelegerea conceptelor și tehnologiilor folosite. Analiza oferă mai multe perspective ce pot fi ulterior transformate în strategii de succes. Prin urmare aceasta etapa necesită o atenție sporita, influențând rezultatul final.

### Identificarea datelor necesare

Pentru realizarea unei astfel de aplicații, este necesară o analiza a datelor provenite de la senzorii dispozitivului mobil pentru realizarea unui sistem utilizabil și fiabil. Odată făcută analiza va fi nevoie de un field de stocare de tipul [`noise\_detection`,`timestamp`], unde noise\_detection reprezinta dacă persoane sau autovehicule au fost detectate sau efectiv nivelul de poluare fonică este alarmant.

### Testare ipoteze

Pentru a ajunge la soluția finala, diverse implementări și soluții vor fi testate și analizate din punct de vedere al corectitudinii și integrității rezultatelor, prin anumite metrici de evaluare. Totodată, o să fie analizat și felul în care interpretarea datelor ajuta în rezolvarea problemei alese.

### Implementarea soluției propuse

Pentru a identifica o soluție care să satisfacă cerințele este nevoie de o tehnica de învățare automata care odată implementată și ulterior integrata în soluția finala, să ofere o analiza și o să putem măsură o acuratele a clasificării zgomotelor de tip ”persoana” și ”autovehicul”. Soluția propusă, pentru a fi accesibilă pentru testare și utilizare va fi implementată ca o aplicație Android ce o să ofere o interfața ce va fi actualizata în timp real cu datele primite ca input, dar și cu datele celorlalți utilizatori, actualizate din cloud.

### Testare și evaluarea soluției propuse

Pentru a interpreta nivelul de performanta al soluției alese, o să se efectueze o analiza și o comparație în același timp cu alte soluții deja existente și a celei implementate și descrisa în acest document. Acest lucru o să se realizeze cu ajutorul unor grafice ce vor indica performanța obținută.

### Interpretabilitate

Soluția va justifica alegerea făcută prin prezentarea nivelului de acuratețe al detecției cu ajutorul unor progress bar-uri, astfel se poate oferi o vizualizare a comportamentului modelului asupra datelor.

## Provocări

### Senzori necalibrați

Senzorii necalibrați reprezintă o problemă semnificativa atunci când vine vorba de măsurarea poluării fonice. Calibrarea este un proces de ajustare și aliniere a senzorilor pentru a asigura măsurători făcute cu acuratețe și de încredere. Inacuratețea în măsurători poate duce la rezultate eronate și a oferi valori îndepărtate de adevăr. Acest lucru este firesc să se întâmple deoarece dispozitivele mobile au senzorii calibrați astfel încât să funcționeze în conformitate cu sistemul de operare al telefonului și astfel pot apărea mici discrepanțe între valoarea calculată și valoarea reală.

**2.4.1. Contextualizarea detecției**

De multe ori, pentru a ne da seama ca o mașină este în apropiere, nu ne putem baza doar pe sunetul produs de accelerația acesteia. De aceea trebuie luate în considerare și frânările, claxonatul, sunetul motorului. Astfel detecția va avea mai multa acuratețe.

## Cerințe funcționale

Principala funcționalitate pe care aplicația trebuie să o îndeplinească este detecția zgomotelor și calcularea numărului de decibeli pentru a indica poluarea fonică, detecția persoanelor și a mașinilor, stocarea și preluarea datelor în timp real din cloud pentru a oferi o privire de ansamblu asupra unei zone întregi.

## Cerințe non-funcționale

### Scalabilitatea

Aplicația trebuie să fie scalabilă, astfel ar trebui să funcționeze chiar dacă pe parcurs o să se adauge noi funcționalități, seturile de date o să se schimbe sau arhitectura proiectului va suferi schimbări.

### Performanța

Sistemul trebuie să stocheze datele în timp real și în același timp atunci când se produce o schimbare să actualizeze interfața pentru a oferi informații relevante. Totodată, dorim performanta din punct de vedere al rezultatelor să fie ridicată și ca acestea să poată fi interpretate ulterior.

# Studiu bibliografic

În acest capitol sunt prezentate o parte din soluțiile existente în domeniul detecției poluării fonice și soluții existente pentru detecția zgomotelor și reglementări cu privire la măsurarea poluării fonice.

## Metodologia măsurării zgomotelor

Indicatorii de zgomot sunt folosiți să determine nivelul de zgomot dintr-un mediu. În general, indicatorii de sunet sunt descriptori ce indică valoarea zgomotului în decibeli. Valoarea unui indicator de zgomot de mediu este determinată prin implementarea unui sistem de măsurări și de estimare, după anumite standarde.

Înainte de a începe procesul de măsurare este necesar să definim obiectivul măsurătorii, a alinia obiectivele și implementarea, definirea surselor de zgomot și intervalelor de timp măsurătorilor.

Calcularea anumitor indici de zgomot dintru-un anumit context, precum traficul, pot fi dificili de implementat deoarece într-un mediu deschis se află mai mulți factori ce influențează măsurătoarea și diverse surse de zgomot. De altfel, atât din punct de vedere științific și profesional, se folosesc indicatori de zgomot de mediu [5]. Cel mai popular indicator este Lden, care este format din diferențele de sunet din timpul zilei Lday, seara Levening și noapte Lnight. Acesta permite compararea mai multor infrastructuri și poate fi reprezentat prin harți de zgomot [5]. Măsurarea sunetului se face 24h pe zi calendaristica, ce este împărțite în trei intervale de referință:

* Ziua durează 12h (06:00 – 18:00) Lday
* Seara durează 4h (18:00 – 22:00) Levening
* Noaptea durează 8h (22:00 – 06:00) Lnight

Nivelul zgomotului Lden pentru zi-seară-noapte este calculat în decibeli după următoarea ecuație [6]:

Lday = 10log(12⋅100.1⋅Lday + 4⋅100.1⋅ (Levening + 5) + 8⋅100.1⋅(Lnight+10))

Unde Lday, Levening și Lnight sunt media ponderata A pe termen lung a nivelelor de sunet, determinate pentru toate zilele, serile și nopțile de-a lungul unui an. Media ponderata A este filtrul standard de pondere în frecvență pentru nivelurile de sunet măsurate cu ajutorul unor instrumente, utilizată de obicei în multe standarde naționale și internaționale [7]. Aceasta acoperă intervale audio de la 20 la 20kHz și corelează cu intensitatea receptată de către urechea umana. Unitatea de măsură pentru măsurători este dB(A).

### Analiza caracteristicilor sunetului

Pe lângă calculul indicatorilor de sunet, caracteristicile acestuia pot fi studiate analizându-i spectrele de frecvență. Transformata Fourier Rapidă (FFT) s-a dovedit a fi un instrument foarte puternic în calcularea caracteristicilor sunetului [8]. Unul din beneficiile transformării semnalul în domeniul frecvențelor este că volumul de date este redus și de asemenea datele pot fi transferate printru-un mediu de comunicare mai repede; totuși, datorită necesităților computaționale ridicate, analiza spectrală necesită mai mult timp de procesare și consum mai mare de energie [9].

## Analiza soluțiilor deja implementate

În ziua de azi, zgomotul produs de trafic este de obicei măsurat cu dispozitive specializate. Mai mult decât atât, aceste dispozitive pot să ofere date adiționale, iar din aceasta cauza este nevoie de un spațiu de stocare foarte mare pentru volumul de date obținut. Recentele avansări în inginerie, în particular tehnologiile de tipul big data, au eficientizat felul în care se stochează volume mari de date și oferă analize avansate în timp real. Odată cu creșterea folosirii dispozitivelor mobile, modalități de colectare de date mai facile au apărut. Tehnologia crowdsensing oferă colectarea informațiilor referitoare la mediu cu ajutorul senzorilor implementați sau alte dispozitive și să transmită datele recepționate către centrele de colectare [10-13].

Metodele tradiționale de măsurat zgomotul sunt scumpe și de obicei se aplică în măsurători oficiale făcute de către agenții specializate [14]. În contrast cu aceste tehnici sunt studii ce caută să exploreze potențialul pe care îl oferă dispozitivele mobile pentru măsurări și analiza traficului [14-17]. Majoritatea acestor studii se bazează pe aplicații mobile precum EarPhone, NoiseSpy, NoiseTube, NoiseBatle și NoiseMap. Undele din aceste aplicații sunt distribuite la nivel comercial disponibile în magazinul de aplicații predestinat dispozitivelor de rulează Android și iOS, în timp ce altele au fost dezvoltate pentru interes de cercetare.

### EarPhone

În acest studiu [14] autorii descriu ca dezvoltarea aplicației EarPhone ce cartografiază zgomotul este un sistem bazat pe crowdsensing. Sistemul EarPhone consta într-o aplicație mobila și un server central. Sistemul include un software pentru procesarea de semnale și măsurarea sunetului de către telefonul mobil, precum și un software pentru reconstruirea semnalului pe serverul central. Testarea sistemului a constat în măsurarea nivelul de zgomot la reconstruirea unui drum principal din Brisbane. Măsurătoarea a fost efectuată timp de o săptămâna, în intervalele orare de 8-9h și 14-15h. Opt măsurători au fost efectuate în fiecare oră, iar în fiecare oră participanții la test mergeau timp de cinci minute și colectau date. Autorii acestui studiu se așteaptă ca sistemul lor să reducă semnificativ costurile de operare în comparație cu utilizarea dispozitivelor de înregistrat și cartografiat zgomote tradiționale.

### NoiseSpy

Sistemul de măsurători NoiseSpy oferă o vizualizare a colecțiilor înregistrate în timp real în timp ce se explorează diferite parți ale orașului [15]. Sistemul de măsurare al zgomotului folosește microfonul incorporat al telefonului și GPS-ul pentru a determina locația exacta a măsurătorii. Testarea sistemului a constat în amplasarea aplicației pe bicicletele curierilor. Pe parcursul rutelor acestora, sistemul măsura nivelul de zgomot și îl atribuia locației oferita de GPS [15]. Un concept similar pentru măsurare zgomotului a fost prezentat în studiul [16] testând aplicația crowdsensing NoiseTube. Spre deosebire de alte aplicații, NoiseTube oferă abilitatea de a atașa diverse etichete la anumite înregistrări, distribuind datele direct din aplicație, aceasta fiind disponibila pe mai multe platforme precum Android și iOS [18]. Pe lângă aplicația mobilă, NoiseTube oferă de asemenea o platforma web ce permite utilizatorilor să cerceteze, vizualizeze și să căuta datele. NoiseTube este aplicația ce recalibrează locația pentru a reduce erorile de poziționare [19].

În figura 3.2.1 este prezentata arhitectura sistemului NoiseSpy în conformitate cu lucrarea [15].

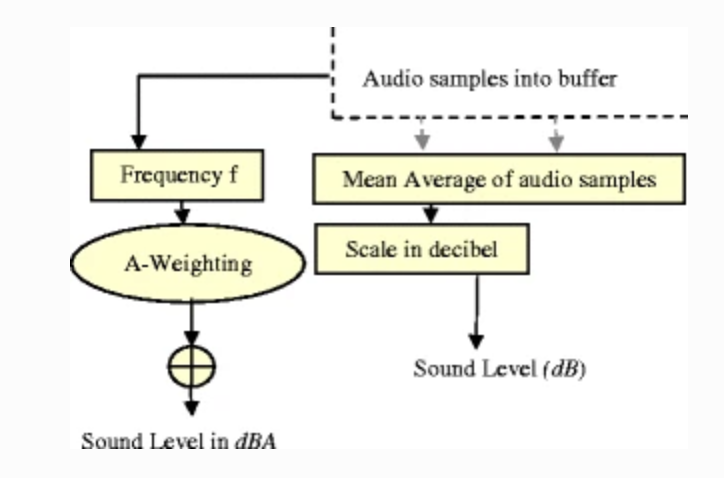


Figura 3.2.1: Arhitectura logica a sistemului NoiseSpy [15]

Autorii studiului [16] au descoperit o metoda inovativa de a anima cetățenii în timp ce colectau și distribuiau date cu privire la zgomot. Folosind proiectul open-source

NoiseDroid pentru a măsura zgomotul, ce necesita utilizatorii să se miște în tot orașul pentru a efectua măsurători ale zgomotului. În figura 3.2 este prezentata o astfel de abordare. Orașul este împărțit în mai multe parți, iar utilizatorul care „câștigă” trebuie să facă mai multe măsurători decât ceilalți participanți. Pe lângă numărul de măsurători, algoritmul de asemenea ia în calcul și calitatea sunetului măsurat.

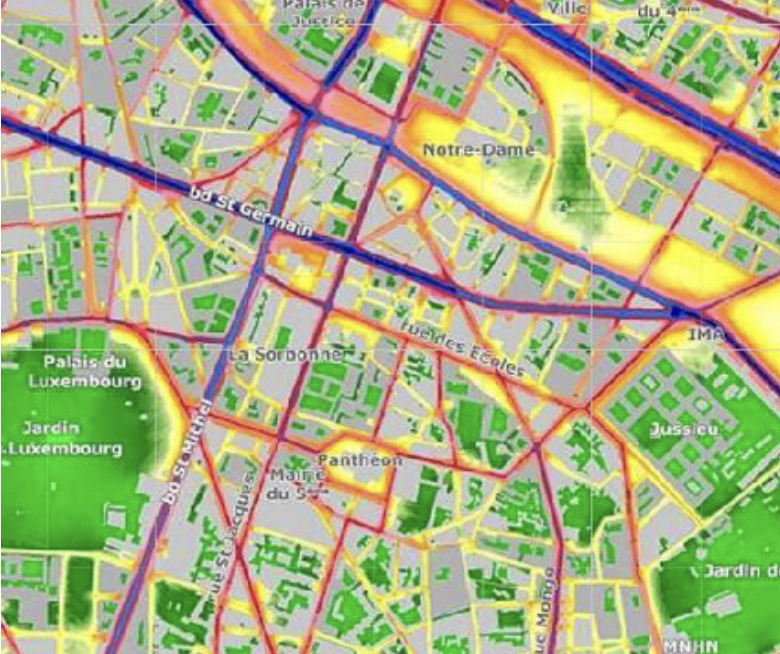


Figura 3.2.2: harta de zgomot a orașului Paris generata folosind propagarea unui model și anumite măsurători efectuate într-un număr limitat de locații. Zonele colorate în verde sunt zone cu zgomot redus, în timp ce zonele zgomotoase sunt colorate în albastru/violet. Zonele gri reprezintă zone fără informatii disponibile (ex. în clădiri) [18].

Cercetătorii de la Universitatea Tehnica din Darmstadt au implementat aplicație NoiseMap, ce colectează date și trimite în platforma web da\_sense [17]. Platforma permite utilizatorilor să vizualizeze datele colectate, să genereze grafice și harți de zgomot. Într-un studiu ce a avut loc în 2012, cercetătorii au creat o harta de zgomot pentru orașul Frankfurt, luând în calcul doar șoselele pe unde trec mai mult de 6 milioane de mașini anual. Spre deosebire de alte proiecte, datele sunt disponibile pe o platforma web publica sau printr-un JavaScript API.

Bazat pe analiza făcută anterior, putem concluziona ca cele mai multe sisteme de măsurare a zgomotului utilizate în orașele smart folosesc: 1) aplicații mobile ce folosesc microfonul pentru a colecta date legate zgomot și un sistem GPS pentru locația în care măsurătoarea a fost efectuată; 2) platforme web pentru vizualizarea datelor legate de zgomot și locație. Totodată, nu se menționează un plan pentru stocarea pe termen lung și a managementului calupului mare de date colectat sau capabilitatea de a face analize complexe în timp real. Avantajele implementării sistemului prezentat în lucrarea aceasta spre deosebire de cele menționate anterior sunt: 1) dezvoltarea unui sistem ce permite stocarea datelor într-o baza de date ce se afla în cloud și este accesibilă oricând; 2) datele colectate pentru analiza sunt interpretate pentru a distinge mașinile și persoanele de alte surse de zgomot din exterior, ceea ce permite o analiza mai sofisticata a zgomotului; 3) analiza este bazata pe datele măsurate disponibila în timp real.

## Acuratețea și precizia măsurătorilor de zgomot

Acuratețea dispozitivelor pentru măsurat zgomote, în termen de apropiere a măsurătorilor cu valorile fizice, reale, a fost considerata în studii din mai multe puncte de vedere. În contextul de crowdsensing, acuratețea folosirii telefoanelor mobile pentru măsurarea zgomotului a stârnit interes. Vorbind despre comparația directa a capabilității telefoanelor mobile de a fi calibrate pentru a exprima metrici de măsurare a sunetului, aceasta a generat un nivel de acuratețe acceptabil, conducând astfel la concluzia ca crowdsensing-ul este o metodă acceptabilă de măsurare a zgomotului generat de trafic [21]. În general, acuratețea măsurării sunetului cu ajutorul telefoanelor mobile depinde de caracteristicile sistemului de operare, producătorului, modelului și condiția din punct de vedere al funcționarii a dispozitivului. [22, 23]. De asemenea, acuratețea depinde și de microfonul încorporat în telefonul mobil. Rezultatele arata faptul ca intervalul în care se încadrează înregistrările de pe un telefon mobil sunt între 50 și 90 dB(A), ceea ce corespunde cu nivele de zgomot obișnuite înregistrate din așezările urbane [23].

Acuratețea măsurătorilor poate fi îmbunătățita cu ajutorul unei calibrări telefonului mobil. Calibrarea se efectuează de obicei cu sonometre (SLMs). Datorită faptului menționat anterior și anume ca diferențele individuale dintre dispozitivele mobile [22], nu este cu adevărat posibil să existe calibrare la telefoanele mobile ale participanților în crowdsensing; în acest caz ar trebui aplicate alte abordări [22, 24, 25]. Din analiza studiilor nu a rezultat nicio soluție ce este uniforma și general aplicabila pentru calibrare, nici măcar pentru modele identice de telefoane mobile [26]. O posibila soluție pentru aceasta problema ar presupune ca fabricanții de dispozitive să publice instrucțiunile tehnice cu privire la calibrarea propriilor dispozitive în conformitate cu standardele [27].

Pe lângă acuratețe, calitatea măsurătorilor depinde și de precizie, de exemplu cât de apropiate sunt efectuate măsurătorile, Precizia depinde de configurația locației, poziția dispozitivelor mobile în raport cu obstacolele fizice, orientarea dispozitivelor mobile, expunerea microfonului în raport cu sursele de zgomot, etc. [23]. Mai mult decât atât, precizia depinde și de mișcarea participanților [28]. În momentul în care mai mulți participanți se afla în experimentul de crowdsensing, precizia poate fi îmbunătățita urmărind protocoale de măsurare stricte [27].

În contextul colectării datelor prin crowdsensing, s-ar putea observa o mai mică acuratețe și precizie în momentul în care sunt puține dispozitive în zona, lucru ce ar putea fi compensat printr-un număr ridicat de măsurători efectuat de un număr mai mare de dispozitive [29]. O fiabilitate mai mare a sumarizării măsurătorile este obținută combinând datele măsurate de mai multe dispozitive și aplicând diverse metode statistice și tehnici pentru filtrarea datelor, explorare și analiza [30].

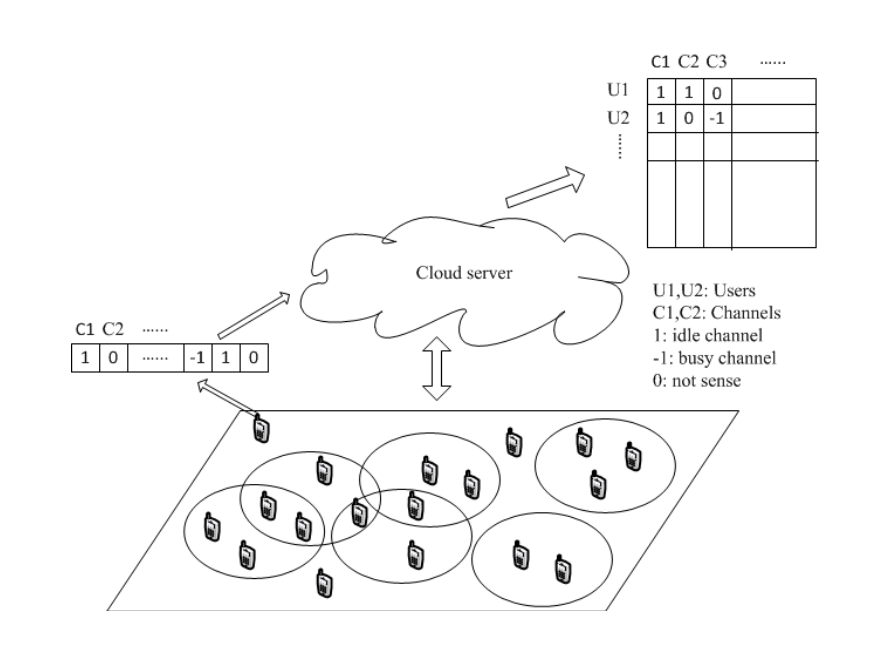


Figura 3.3.1: Descoperirea spectrului bazată pe fuziunea datelor provenite de la dispozitive mobile [30].

## Reglementari cu privire la măsurarea zgomotelor

Uniunea Europeana este considerata ca fiind lider în domeniul măsurării zgomotelor [31]. Statele membre efectuează cartografieri ale zgomotelor de mulți ani, datorită recomandărilor din cadrul directivei UE 2002-49-EC [32]. Directiva a încurajat avansul în metodele de măsurare a zgomotului și în crearea de harți de zgomot pentru mai multe orașe importante [33]. Cu privire la poluarea fonică, UE are diferite legi prevăzute pentru traficul la sol, traficul aerian, cât și pentru sursele de zgomot industrial. În zonele urbane, doar șoselele principale, liniile principale de cale ferata și aeroporturile principale sunt luate în considerare. Spre deosebire de Europa, unde cartografierea zgomotului este obligatorie, Statele Unite ale Americii și Japonia au legi ce datează din anii 1970, dar cartografierea zgomotului este opțională. În dezvoltarea tarilor, de obicei este o lipsă la legislaturii pentru măsurători, evaluări și control al nivelelor de zgomot [34].

Conform lucrării [35] se observa răspunsul populației mai multor orașe importante în care se exprima nivelul de satisfacere cu nivelul de zgomot local.

A picture containing text, map, atlas

Description automatically generated

Figura 3.4.1: Proporția de persoane satisfăcută cu nivelul de zgomot în orașele unde locuiesc, 2015 [35]

# Analiză și fundamentare Teoretică

În acest capitol urmează să fie prezentata soluția aleasa, după care o să fie descrise în detaliu conceptele teoretice și algoritmii utilizați pentru implementare.

## Soluția propusa

Soluția propusa presupune utilizarea unui model pre-antrenat, integrat într-o aplicație Android, alături de senzorii incorporați ai telefonului mobil, iar acestea însumate rezultând într-un sistem capabil să recunoască persoanele și mașinile din apropiere, bazat pe datele de intrare de la microfonul telefonului, să calculeze poluarea fonică în timp real să amplaseze pe harta zonele afectate de poluare fonică indiferent dacă utilizatorul în mișcare sau nu. Datele recepționate vor fi transmise în cloud, iar ceilalți utilizatori pot beneficia de ele în timp real. Pentru realizarea sistemului s-au efectuat următoarele procese prezentate în figura 4.1., în care datele provenite de la microfonul telefonului sunt calibrate, iar ulterior sunt trimise către modelul propriu-zis, procesate si vizibile tuturor utilizatorilor.

A picture containing text, screenshot, diagram, font

Description automatically generated

Figura 4.1. Diagrama generala a sistemului

Modelul a fost antrenat pe un set de date ce este bazat pe sunetul din diverse clip-uri video, din care au fost extrase segmente de 10 secunde, etichetate de oameni și împărțite în ontologii. O ontologie este o colecție de sunete organizate într-o ierarhie, iar aceasta acoperă o arie largă din sunetele pe care le receptam în fiecare zi precum, sunete produse de oameni și animale, sunete din natura și din mediul înconjurător, muzică, trafic, etc. Pentru antrenarea modelului în ceea ce privește detecția vehiculelor au fost folosită următoarea ponderea de material video.

A screenshot of a data

Description automatically generated with low confidence

Figura 4.1.2: Antrenarea modelului pentru detecția vehiculelor [39].

## Prezentarea cazurilor de utilizare din cadrul sistemului

Bazat pe descrierea generală a sistemului, un utilizator trebuie să poată efectua o serie de acțiuni specifice pentru buna funcționare. Precum este prezentat in Figura 4.2 acesta trebuie să ofere în primul rând diverse permisiuni ce vor fi explicate ulterior în cadrul lucrării pentru a putea beneficia de funcționalitate a soluției propuse.

A picture containing text, diagram, circle, screenshot

Description automatically generated

Figura 4.2 Diagrama cazurilor de utilizare

## TensorFlow în Android

TensorFlow este un framework open-source de machine-learning, ce oferă diverse librării pentru clasificare audio. Acesta poate fi modelat în funcție de cerințele funcționale ale proiectului. Pentru clasificarea audio următorii pași de preprocesare și procesare de date sunt necesari.

### Preprocesarea datelor audio

TensorFlow oferă diverse funcții și librarii pentru a decodifica diverse formate audio precum mp3 sau WAV, iar mai apoi se eșantionează pentru a avea același eșantion cu modelul. Formele de unda se normalizează în intervalul -1 și 1, pentru a se asigura nivele constante de amplitudine. De obicei acest procedeu se efectuează împărțind datele audio la valoarea lor maxim absoluta. Acestea sunt împărțite apoi în secvențe mai mici de câteva milisecunde și se calculează caracteristicile relevante din fiecare secvența audio. Cele mai comune caracteristici includ: Mel-frequency cepstral coefficients(MFCC) [36] și spectrograme ce reprezintă magnitudinea sau puterea spectrală a semnalului.

### Augmentarea datelor și procesarea

Tehnici pentru augmentarea datelor pot fi aplicate pentru a crește artificial dimensiunea și diversitatea data setului de învățare. TensorFlow include funcționalități pentru implementarea tehnici ca și parte a preprocesării. Arhitectura clasificării audio este efectuate de API-uri de nivel înalt precum Keras [37] sau de nivel scăzut precum tf.nn [38]. De obicei arhitectura acestor modele este de tipul CNNs (rețele neuronale convoluționale) sau RNNS (rețele neuronale recurente). Antrenarea acestora se efectuează cu datele preprocesate și codificate sub forma de date de intrare. Parametrii modelelor se pot optimiza folosind algoritmi specifici pentru acest lucru.

### Integrarea TensforFlow în soluția propusa

Pentru integrarea TensorFlow în cadrul aplicației Android creata a fost nevoie de folosirea librăriei TensorFlow LiteTask Library [39]. Următorul pas consta în filtrarea output-ului venit pe baza detecției, afișarea acestuia și actualizarea datelor pe baza a ceea ce se dorește. Pentru a face lucrul acesta a fost nevoie să implementez un Adapter [40]. În programarea în Android, un adapter este de fapt un canal de comunicare între interfața grafică și baza de date. Un view de tip adapter se asigura ca fiecare element de pe interfața grafică corespunde unui element din baza de date, formând o legătura între ele, astfel încât orice acțiune atât de pe frontend cât și de pe backend să se propage în ambele zone. Astfel în momentul în care un element este recunoscut pe baza sunetului emis (ex. un autovehicul), pe interfața grafică va apărea informația respectiva, va și scrisa automat și în baza de date, iar eticheta respectiva va dispărea în momentul în care nu este nimic de detectat.

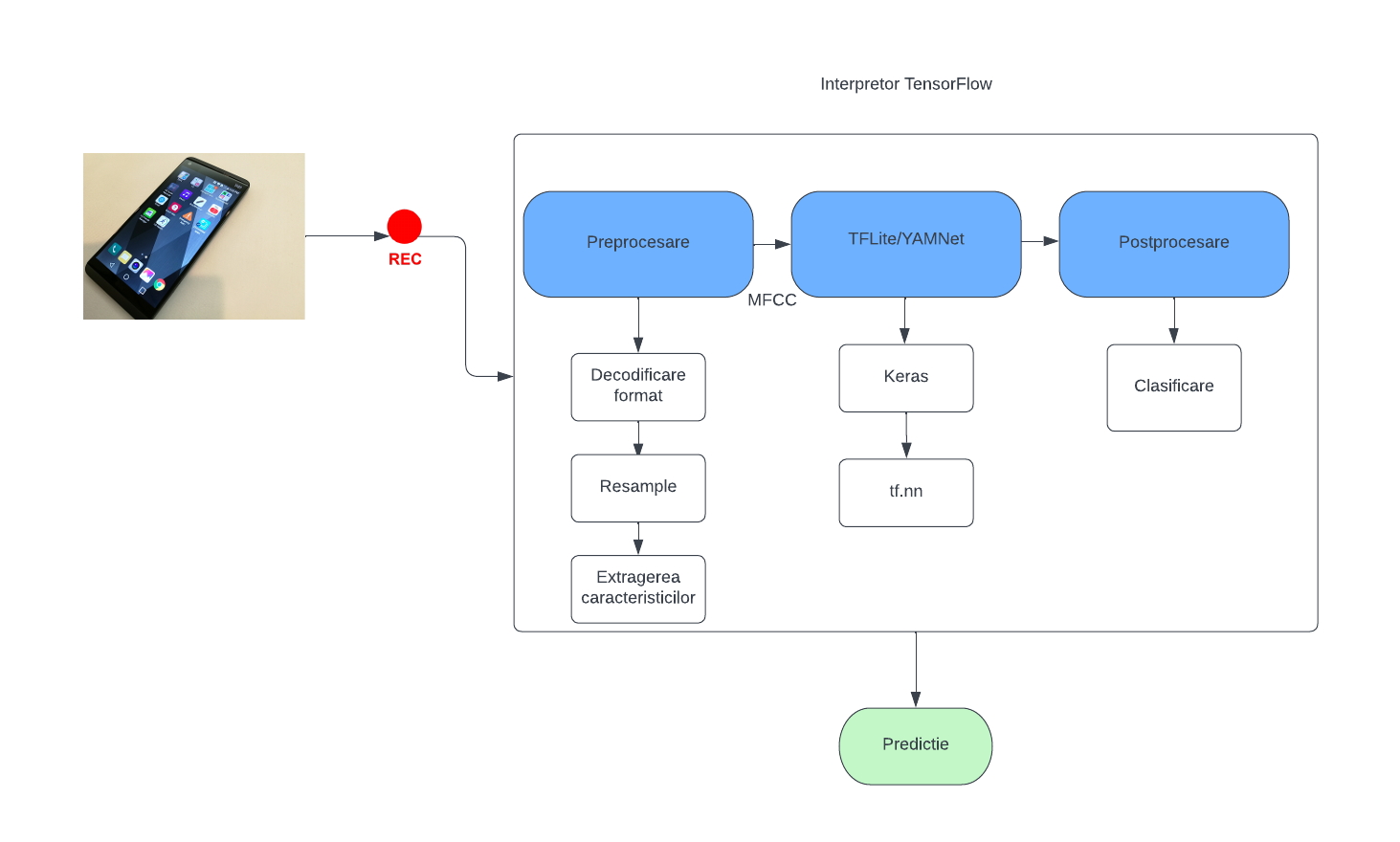


Figura 4.2.1: Conexiunea TensorFlow cu Android

## Localizarea în timp real în cadrul aplicației

Locația utilizatorilor, cu acordul acestora, este o caracteristica a soluției propuse deoarece aceasta presupune amplasarea cu exactitate a zonelor de interes și a informațiilor necesare ce corespund fiecărei zone.

### Cererea permisiunilor pentru locație

Pentru a accesa locația utilizatorilor, aplicația Android trebuie să ceară permisiunile necesare. Permisiunea ”ACCESS\_FINE\_LOCATION” este necesară, deoarece aceasta permite aplicației să acceseze datele GPS exacte. Pentru a obține aceasta permisiune următorii pași trebuie urmați: 1) în cadrul fișierului AndroidManifest, unde se găsesc detalii cu privire la versiunile folosite în cadrul aplicației, se mai adaugă o secțiune (<uses-permission>) în care se menționează că aplicația folosește permisiunile cerute; 2) cererea permisiunilor în care aplicația rulează, astfel în cazul în care acestea sunt acordate, amplasarea pe harta se va face conform cerinței; 3) verificarea faptului că sistemul de localizare din cadrul sistemului este activ, iar dacă acesta nu este, cererea permisiunii pentru activare din cadrul aplicației automat.

### Servicii de localizare și furnizori

În mediul Android sunt numeroase servicii de localizare și furnizori pentru a obține informații cu privire la locație. Cei mai întâlniți sunt cei de tipul GPS, locația bazata pe rețea și furnizori de locație fuzionați. Furnizorii de locație fuzionați combină datele din mai multe surse pentru a asigura acuratețe și eficiență în actualizarea locației. Astfel, pentru a integra servicii de localizare am realizat conexiunea folosind serviciile Google Play API [41], astfel formând-se legătura cu furnizorul de locație fuzionat. Ulterior a setat un LocationListener [42] pentru a obține actualizări de la furnizor. Iar pentru a recepționa schimbările am implementat apelul onLocationChanged()[42] pentru a primi actualizări ale locației, mai exact latitudinea și longitudinea.

### Integrarea localizării în timp real

Pentru a asigura localizarea în timp real în aplicație, actualizările cu privire la locația utilizatorului trebuie integrate și utilizate. Pentru a obține acest lucru trebuie integrata o hartă; acest lucru este posibil prin integrarea cu Google Maps API [43] pentru a putea afișa locațiile utilizatorilor și a actualiza în timp real acțiunile acestora. Pentru rularea în fundal a acestui lucru este necesară cererea de permisiuni pentru obținerea locației chiar dacă utilizatorul nu se afla în cadrul aplicației. Toate aceste date menționate anterior trebuie stocate într-un loc centralizat, iar pentru asta am ales Firebase, ce va fi prezentat în capitolele următoare.

A diagram of a flowchart

Description automatically generated with low confidence

Figura 4.3.1: Diagrama ce descrie declararea și cererea permisiunilor la rulare Android.

## Calcularea numărului de decibeli cu ajutorul telefonului

Pentru calcularea numărului de decibeli în soluția propusa, trebuie obținută permisiunea pentru acces la microfonul telefonului apoi detecția sunetului în SPL (nivelul de presiune al sunetului) și convertit în dB(A).

### Preprocesarea datelor audio obținute

Pentru a obține datele audio prin microfonul incorporat al dispozitivului o să ne folosim de o caracteristica incorporata în Android și anume AudioManager. Acesta oferă acces la datele audio ca și flux, lucru ce va fi folositor pentru procesarea ulterioara. Odată ce datele sunt obținute, trebuie făcută o conversie deoarece inițial, datele audio brute sunt reprezentate ca și forme de undă, dar pentru a calcula SPL trebuie să fie convertite într-o reprezentare numerica. După ce componentele de frecventa au fost extrase, valoarea SPL poate fi calculata astfel:

Presiunea se referă la amplitudinea presiunii sonore, iar referința pentru nivelul presiunii sunetului este de obicei 20 μPa.

### Aplicarea filtrelor de pondere și conversia în dB(A)

Termenul dB(A) este o unitate de măsura pentru sunet ce ia în considerare variația sensibilității sistemului de auz al oamenilor. Aplicarea filtrelor de pondere A, reprezintă frecvente de pondere aplicate pe valorile nivelului presiunii sunetului (SPL). Pentru a se face conversia din SPL în dB(A), valoare SPL trebuie înmulțită cu coeficienții filtrelor de ponderea A, astfel valoarea obținută este aceeași cu nivelul perceput de urechea umana. Aceasta valoare odată calculata va fi stocată în baza de date din cloud, va fi actualizata mereu și va fi disponibila pe interfața grafică pentru utilizatori.

## Integrarea Firebase

Firebase este o platforma dedicata dezvoltării aplicațiilor mobile și web ce oferă o gama larga de servicii și resurse. A fost preluata de cei de la Google în 2014 și de atunci a devenit una din cele mai popularea variante pentru construirea de aplicații scalabile. La baza, Firebase oferă o infrastructura de backend bazata pe cloud ce simplifică mai multe sarcini ce trebuie îndeplinite. Acesta oferă o suită de beneficii ce pot fi folosite la implementare precum stocare de date în timp real, autentificare, servicii de administrare și multe altele.

### Firebase Authentication

În orice sistem, la fel ca și în cel propus, trebuie să existe o parte de autentificare. Pentru a asigura acest lucru în soluția propusa, am folosit Firebase Authentication, care este un sistem securizat de permite autentificarea utilizatorilor și autorizarea acestora. Firebase Authentication suporta autentificarea prin mai multe moduri precum: autentificarea cu numărul de telefon, autentificarea prin diverse rețele sociale precum Google, GitHub, Facebook și Twitter, dar și prin email și parola.

Pentru a integra acest serviciu am folosit un SDK [44] oferit de cei de la Firebase [45] ce permite încapsularea într-o aplicație Android. Prin comunicarea efectuata în cadrul aplicației utilizatorii pot efectua operații de înregistrare și autentificare securizate.



Figura 4.5.1: Modalitățile de autentificare folosind Firebase Authentication [45].

### Firebase RealTime Database

Firebase RealTime Database este o bază de date în cloud de tipul NoSQL [46] oferita de Firebase. Aceasta oferă sincronizare în timp real, posibilitate de stocare de date, oferind astfel opțiunea de a construi aplicații ce pot să colecteze date și să se actualizeze în timp real.

Acest sistem, spre deosebire de cele clasice în care structura era sub forma tabelara, primește și înmagazinează datele sub forma de JSON, adică o forma arborescenta. Pentru integrarea acestei caracteristici cu soluția propusa a fost nevoie de încadrarea anumitor dependințe în fișierul build.gradle [47]. Astfel, datele cu privire la poluarea fonică provenite de la utilizatori pot fi recepționate, procesate și transmise către alți utilizatori în timp real.

A picture containing text, diagram, plan, rectangle

Description automatically generated

Figura 4.5.2: Stocarea și actualizarea datelor în timp real folosind Firebase RealTime Database.

### Firebase Functions

Firebase Functions reprezintă o platforma oferita de Firebase ce permite rularea de cod pentru backend precum schimbări în baza de date, autentificarea utilizatorilor sau operații pentru întreținerea infrastructurii.

În soluția propusa a fost necesară integrarea acestui modul pentru a putea prelua datele anonime ale fiecărui utilizator și anume locația și datele cu privire la poluarea fonică. Utilizatorii sunt stocați după id-ul autogenerat de sistemul de autentificare, ceea ce nu periclitează informațiile cu caracter personal ale acestora.

Funcțiile au fost scrise în JavaScript, după care s-a făcut deploy pe serverul dedicat oferit de Firebase. După integrarea dependințelor necesare în proiect, funcțiile scrise au putu fi apelate.

## Arhitectura MVVM

Arhitectura folosită în dezvoltarea proiectului este Model-View-ViewModel (MVVM). Aceasta este răspândită în dezvoltarea aplicațiilor Android deoarece oferă o structură organizată în construirea aplicațiilor robuste și scalabile. MVVM separa preocupările cu privire la integrarea datelor pe interfața grafică și procesarea propriu-zisă a lor, rezultând astfel o separare a responsabilităților în implementare.

### Model

Modelul reprezintă datele și partea de bussiness logic [48] ale aplicației. Acesta încapsulează sursele de date precum bazele de date locale, API-uri sau baze de date aflate în cloud. Modelul este responsabil pentru aducerea, manipularea și persistența datelor. Nu este dependent de View și ViewModel. În soluția propusa, modelul se preocupa de colectarea datelor de la senzorii telefonului, procesarea acestora și adăugarea în baza de date.

### View

View-ul reprezintă componentele interfeței grafice, în Android definite ca și Activități [49] și Fragmente [50]. Acesta este responsabil de afișarea datelor către utilizatori și să se ocupe de interacțiunile cu aceștia. View-ul nu trebuie să conțină elemente de bussiness logic sau să mențină starea în care se afla aplicația.

### ViewModel

ViewModel-ul este o componenta ce face parte din Componentele de Arhitectura Android, conceputa pentru a stoca și gestiona datele legate de interfața de utilizare într-un mod conștient de ciclul de viață. Acționează ca un hub de comunicare între interfața grafică (Activitate sau Fragment) și datele aflate într-o baza de date sau un server aflat la distanta.

Un ViewModel este responsabil pentru păstrarea datelor solicitate de interfața grafică și pregătirea lor pentru afișare. Acesta supraviețuiește schimbărilor de configurare (navigarea între diferite ecrane ale aplicației) și se asigura ca datele rămân intacte și ca rămân disponibile pe tot parcursul ciclului de viață al componentei căreia îi este asociat.

Prin folosirea de ViewModel se separa preocupările cu privire la interfața grafică și gestionarea datelor, codul fiind mai modular, ușor de întreținut și de testat.

# Proiectare de detaliu și implementare

## Descriere generala

În cadrul acestui capitol o să prezint modul de funcționare al soluției propuse și o să prezint implementarea acesteia alături de diagramele UML și scenariile de utilizare pana la structura pachetelor și a claselor, a șabloanelor arhitecturale folosite și cele mai importante secvențe de cod pentru a putea prezenta detaliat conceptele și algoritmii enumerați anterior. O să fie prezentate și provocările pe care le-am întâmpinat în procesul de implementare; soluția propusa folosește diverse framework-uri și anumite librarii. Aplicația este scrisa în limbajul Kotlin, deoarece este limbajul responsabil pentru programarea aplicațiilor Android de la interfețele grafice, comunicarea cu baza de date, navigarea între fragmente și inclusiv comunicarea cu senzorii incorporați ai telefonului mobil. Un al doilea limbaj folosit este JavaScript pentru scrierea funcțiilor în Firebase, pentru a putea accesa informațiile referitoare fiecărui utilizator precum locație, detecțiile din jurul său și coeficienții de poluare fonică pe parcursul zilei, bazat pe id-ul autogenerat al acestuia. Aceste doua limbaje au fost integrate prin Firebase, ca într-un final să rezulte soluția ce va fi prezentata în amănunt în cele ce urmează.

## Crearea aplicației

Pentru a crea o noua aplicație Android este nevoie de Android Studio, iar odată instalat acesta trebuie să fie selectate opțiunile File > New > New Project, ce va deschide un dialog precum cel din Figura 5.1. Aici poate fi selectat tipul aplicației (ce tip de navigare dorim să avem deja implementate), iar pentru soluția propusa o să alegem Empty Views Activity. După ce am selectat tipul suntem nevoiți să alegem limbajul trebuie să alegem limbajul în care aplicația o să fie dezvoltata, iar aici o să alegem Kotlin, fiind limbajul de programare recomandat de Google pentru dezvoltarea aplicațiilor Android. Tot aici o să selectam și versiunea minimă a sistemului de operare pe care să ruleze aplicația, Figura 5.2. Am ales Android 8.0 (Oreo) ca versiune minima, în prezent fiind lansată versiunea de Android 13, iar versiunea 14 fiind în varianta Beta disponibilă din partea celor de la Google.

A screenshot of a computer

Description automatically generated

Figura 5.1 Alegerea tipului aplicației create.

A screenshot of a computer

Description automatically generated with medium confidence

Figura 5.2 Alegerea limbajului și a sistemului de operare minim pe care aplicația să ruleze.

Pasul următor consta în adăugarea dependințelor pentru funcționarea aplicației. Inițial, atunci când aplicația este creata o multitudine de dependințe sunt adăugate automat pentru buna funcționare a componentelor din interfața grafică cât și pentru a face posibila navigarea între fragmente și activități. Pe lângă librăriile preinstalate am adăugat și diverse librarii auxiliare.

O prima librărie utilizata pentru implementare a fost cea pentru integrarea Firebase, Figura 5.4. Alături de aceasta au fost necesare integrarea și dependințelor pentru folosirea Firebase Authentication și Firebase Functions. Pentru a sincroniza proiectul creat cu Firebase este necesară conectarea cu un cont Google pe platforma celor de la Firebase. Odată creat proiectul în consola, un fișier numit google-services.json va fi generat și trebuie integrat în directorul în care se afla proiectul, Figura 5.3.

A screenshot of a computer

Description automatically generated with medium confidence

Figura 5.3 Adăugarea fișierului google-services.json

A screenshot of a computer

Description automatically generated with medium confidence

Figura 5.4 Dependințele necesare pentru integrarea cu Firebase [51]

O a doua librărie externa utilizata a fost cea pentru integrarea Google Maps [52]. Aceasta librarie a fost necesară pentru localizarea pe harta. Alături de dependința adăugată în fișierul build.gradle a fost nevoie și de adăugarea unei chei API, generata de cei de la Google și folosită în fișierul AndroidManifest.xml, pentru a putea utiliza platforma alături de serviciile lor.

A screenshot of a phone

Description automatically generated with low confidence

Figura 5.5 Apelurile făcute prin Google Maps API [52]

A treia librărie utilizata în cadrul soluției este MPAndroidChart [53] ce este folosită pentru generarea de grafice interactive. Pentru crearea și folosirea acestor grafice este necesară integrarea următoarelor dependințe, vezi Figura 5.6. Odată adăugate acestea, obiectele din cadrul librăriei sunt disponibile și pot fi folosite atât în fișierele .xml unde este creata interfața grafică, dar și în fragmente, viewmodels sau clase ajutătoare pentru a manipula datele ce urmează a fi afișate.

A screen shot of a computer code

Description automatically generated with low confidence

Figura 5.6 Dependințele MPAndroidChart

Cea de a patra librărie folosită în cadrul aplicației este TensorFlow, ce permite utilizarea elementelor pentru efectuarea detecțiilor, comutarea între CPU și GPU pentru efectuarea detecțiilor și afișarea acestora în cadrul soluției propuse.

implementation 'org.tensorflow:tensorflow-lite-task-vision:0.4.0'  
implementation 'org.tensorflow:tensorflow-lite-gpu-delegate-plugin:0.4.0'  
implementation 'org.tensorflow:tensorflow-lite-gpu:2.9.0'

Listing 5.1 Dependințele pentru integrarea librăriei TensorFlow

## Arhitectura aplicației

Aplicația Android create a fost conceputa pe modelul Model-View-ViewModel și adaptata pe o arhitectura Layered compusa din presentation și data. Aceasta a fost necesară pentru a separa interfața grafică de funcțiile ce reprezintă partea de business logic și clasele de tip helper care ajuta comunicarea cu baza de date aflată în cloud, Figura 5.7.

A picture containing diagram, plan, technical drawing, line

Description automatically generated

Figura 5.7 Diagrama de pachete

Aplicația dispune de mai multe cazuri de utilizare; acestea au fost prezentate în capitolul anterior. Inițial utilizatorul va fi redirecționat către pagina de register, unde are opțiunea de a își crea un cont, iar în cazul în care are unul are opțiunea de a naviga spre pagina de login. Daca utilizatorul a efectuat deja logarea, acesta va fi redirecționat direct de fiecare data când deschide aplicația către ecranul principal care este compus la rândul sau din trei componente. Prima componenta reprezintă amplasarea pe harta a acestuia și a celorlalți utilizatori din cadrul aplicației. Sub harta, vor apărea detecțiile făcute, dacă sunt persoane sau autovehicule în zona prin intermediul detecției făcute cu datele provenite de la microfonul telefonului. Un alt element ce se regăsește este un buton ce oferă posibilitatea de a arata pe harta în timp real aglomerațiile de mașini de pe străzile orașului. În partea de jos este un view ce se extinde pentru a putea modifica felul în care detecțiile sunt făcute și anume pe CPU sau GPU și threshold-ul detecției. Pe a doua componenta se poate găsi o listă ce conține ultimele date înregistrate în baza de date bazate pe fiecare utilizator, unde sunt prezentate locația acestuia, data ultimei înregistrări a persoanelor sau autovehiculelor, precum și zgomotul măsurat în decibeli. Pe cea de a treia componenta se găsesc niște grafice interactive ce reprezintă indicii Lden inregistrati de către fiecare utilizator, aceștia se modifica în fiecare minut pe baza măsurătorilor efectuate și în intervalele orare prestabilite. Ecranele în care se găsesc componentele descriese mai sus sunt:

* RegisterActivity – activitatea responsabilă pentru crearea unui cont nou și redirecționarea fie către Login sau ecranul principal
* LoginActivity - responsabilă pentru autentificarea utilizatorilor și redirecționarea către ecranul principal
* MainActivity – compus din trei fragmente și anume: MainFragment unde se afla harta și afișarea detecțiilor cu preciziile cu care s-au efectuat, DetailsFragment unde se poate observa lista cu datele colectate de la fiecare utilizator și ChartsFragment, unde se pot vizualiza coeficienții Lden calculate și zona translatată din coordonatele în adresa fizica a fiecărui utilizator.

## Navigarea între ecranele aplicației

La prima instalare a aplicației, utilizatorul va fi redirecționat către pagina de Register unde are posibilitatea de a crea un cont nou. Odată completate datele de autentificare, utilizatorul va observa un Toast [54] care ii va transmite ca a fost înregistrat cu succes, respectiv ca datele introduce nu sunt valide. Când butonul de register este apăsat, se face un request la Firebase Authentication și utilizatorul va fi redirecționat către ecranul principal, vezi Listing 5.1. Din RegisterActivity, utilizatorul are posibilitatea să fie redirecționat către LoginActivity în cazul în care are deja un cont creat. Acest lucru este realizat cu un Intent [55], vezi Listing 5.2.

firebaseAuth.createUserWithEmailAndPassword(emailField, passwordField).addOnCompleteListener **{** task **->** if (task.isSuccessful) {  
 Toast.makeText(this@RegisterActivity, "Account created", Toast.LENGTH\_SHORT).show()  
 startActivity(Intent(applicationContext, MainActivity::class.java))  
 } else {  
 Toast.makeText(this@RegisterActivity, "Account not created ${task.exception?.message}", Toast.LENGTH\_SHORT).show()  
 }  
**}**

Listing 5.2 Navigarea către MainActivity în cazul în care datele introduse sunt valide, altfel afișarea unui mesaj de avertizare

loginButton.setOnClickListener **{** startActivity(Intent(this@RegisterActivity, LoginActivity::class.java))  
 finish()  
**}**

Listing 5.3 Navigarea din RegisterActivity către LoginActivity

În MainActivity sunt prezente trei fragmente: MainFragment, DetailsFragment și ChartsFragment. Pentru integrarea acestora am folosit BottomNavigationView [56] din Android. Pentru integrarea acestuia a fost nevoie să creez o resursă Menu [57] în xml care să reprezinte id-ul fiecărui fragment, vezi Listing 5.3. După care din MainActivity se face navigarea către fiecare Fragment pe baza id-urilor menționate anterior în funcție de selecția făcută de către utilizator. Acest lucru este posibil prin folosirea unui obiect de tipul SupportFragmentManager [58]. Acesta se ocupa de încărcarea fragmentelor și afișarea celor corespunzătoare fiecărei selecții făcute, vezi Listing 5.4. În Figura 5.9 se poate observa navigarea completă între ecranele aplicației.

<menu xmlns:android="http://schemas.android.com/apk/res/android">  
 <item  
 android:title="Map"  
 android:id="@+id/map"  
 android:icon="@drawable/ic\_map"/>  
 <item  
 android:title="Details"  
 android:id="@+id/details"  
 android:icon="@drawable/vector"/>  
 <item  
 android:title="Charts"  
 android:id="@+id/charts"  
 android:icon="@drawable/ic\_chart"/>  
</menu>

Listing 5.4 Resursa folosită pentru BottomNavigationView

override fun onNavigationItemSelected(item: MenuItem): Boolean {  
 when (item.itemId) {  
 R.id.map -> supportFragmentManager.beginTransaction().hide(detailsFragment).hide(chartsFragment).show(mapFragment).commit()  
 R.id.details -> supportFragmentManager.beginTransaction().hide(mapFragment).hide(chartsFragment).show(detailsFragment).commit()  
 R.id.charts -> supportFragmentManager.beginTransaction().hide(mapFragment).hide(detailsFragment).show(chartsFragment).commit()  
 }  
 return true  
}

Listing 5.5 Navigarea folosind SupportFragmentManager

A picture containing text, post-it note, diagram, screenshot

Description automatically generated

Figura 5.8 Navigarea în aplicație printre activități și fragmente

În diagrama următoare este prezentat întreg procesul pentru localizarea pe harta a utilizatorilor, detecția de persoane și autovehicule, măsurarea nivelului de presiune al sunetului și stocarea datelor în baza de date. Astfel se oferă o privire de ansamblu a modului de funcționare al proiectului.

A picture containing text, diagram, parallel, plan

Description automatically generated

Figura 5.9 Diagrama de secvența pentru MainFragment pentru localizarea pe harta, efectuarea detecției și obținerea coeficienților de poluare fonică.

## Implementarea metodelor pentru obținerea locației curente a utilizatorilor

Pentru a obține locația utilizatorilor, în primul rând trebuie ca aplicația să includă în AndroidManifest.xml următoarele permisiuni:

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />  
<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

Listing 5.6 Permisiunile pentru accesarea locației

Odată ce utilizatorul va porni aplicația acestuia i se va afișa un dialog în care este întrebat dacă este de acord să ofere permisiunile de acces pentru locație sistemului. Inițial se face o verificare dacă permisiunea este deja oferita aplicației conform următoarei secvențe de cod:

if (ActivityCompat.checkSelfPermission(activity, android.Manifest.permission.ACCESS\_FINE\_LOCATION) ==  
 PackageManager.PERMISSION\_GRANTED)

Listing 5.7 Verificare dacă permisiunile sunt atribuite aplicației

Pentru a crea o experiența cât mai plăcută pentru utilizatori, am implementat o metoda care verifica dacă GPS-ul este pornit din sistemul telefonului. Pentru a implementa aceasta functionalitate, am folosit LocationManager și se returnează un Boolean pentru confirmare.

private fun isGPSEnabled(activity: Activity): Boolean {  
 val locationManager: LocationManager? = activity.getSystemService(Context.LOCATION\_SERVICE) as LocationManager?  
 var isEnabled = false  
  
 if (locationManager != null) {  
 isEnabled = locationManager.isProviderEnabled(LocationManager.GPS\_PROVIDER)  
 }  
 return isEnabled  
}

Listing 5.8 Verificare dacă GPS-ul telefonului este activ

În cazul în care GPS-ul este oprit, am implementat o metoda prin care acesta să poată fi deschis automat de către aplicație cu acordul utilizatorului. Astfel, când un utilizator intra în aplicație, după ce oferă permisiunile necesare acesta va fi localizat în timp real pe hartă, alături de ceilalți utilizatori ale căror locații sunt actualizate și ele constant în funcție de mișcarea acestora. Diagrama prin care se efectuează ciclul prezentat anterior se poate observa în Figura 5.11.

A diagram of a gps system

Description automatically generated with low confidence

Figura 5.10 Diagrama ce descrie cererea permisiunilor pentru accesarea locației

## Implementarea metodelor pentru accesul la microfon și detecțiile efectuate

În AndroidManifest.xml este nevoie de integrarea următoarelor permisiuni pentru accesul la microfon:

<uses-permission android:name="android.permission.RECORD\_AUDIO" />

Listing 5.9 Permisiunile necesare pentru accesul la microfonul telefonului

### Măsurarea numărului de decibeli

Pentru a accesa datele provenite de la microfonul telefonului a fost nevoie de verificarea dacă utilizatorul a acordat permisiunile pentru acest lucru. În cazul în care avem acces la datele provenite de la microfon o să folosim mai departe MediaRecorder [59] unde am configurat ca și sursa audio să fie microfonul incorporat al telefonului. Ulterior s-a creat un thread care să ruleze în permanenta ca să nu se suprasolicite firul de execuție principal al aplicației. În dezvoltarea aplicațiilor Android, thread-ul principal este responsabil pentru încărcarea elementelor interfeței grafice și nu este tocmai adecvată suprasolicitarea acestuia. Pentru calcularea nivelului presiunii sunetului este nevoie de calcularea RMS [60] (root mean square) care se calculează în metoda prezentata în Listing 5.9. În metoda se calculează suma indicilor din buffer bazata pe inputul produs de microfonul telefonului și se returnează radicalul acesteia.

private fun calculateRms(buffer: ShortArray): Double {  
 var sum = 0.0  
 for (i în buffer.indices) {  
 sum += buffer[i] \* buffer[i]  
 }  
 val mean = sum / buffer.size  
 return sqrt(mean)  
}

Listing 5.10 Calcularea root mean square

Pentru calcularea nivelului presiunii sunetului se va lua presiunea de referință ca și valoare standard 20\*10-5, iar formula pentru calculare este prezenta în metoda din Listing 5.10

private fun calculateSPL(rms: Double): Double {  
 val refPressure = 20 \* 10.0.pow(-5) // reference sound pressure în Pascals  
 return 20 \* log10(rms / refPressure)  
}

Listing 5.11 Calcularea nivelului de presiune a sunetului

Pentru a avea rezultatele calculate continuu a fost nevoie de crearea unei interfețe Listener și a unei funcții care să fie apelata în momentul în care fragmentele sunt afișate de către activitate.

În aplicația am implementat calcularea indicilor Lday, Levening și Lnight. Intervalele prestabilite pentru acești indici au fost aleși în felul următor: pentru Lday am ales intervalul orar de la ora 06:00 pana la 18:00, Levening în intervalul 18:00 pana la 22:00, iar Lnight se calculează între ora 22:00 și 06:00. Pentru calcularea fiecărui indice am avut nevoie de crearea unui buffer circular, astfel după ce numărul măsurătorilor necesare pentru calcularea unei valori sunt efectuate, noile măsurători le înlocuiesc pe cele vechi. Clasa pentru CircularBuffer se afla în pachetul models și este implementată în Listing 5.11.

class CircularBuffer<T>(private val capacity: Int) {  
 private val buffer: Array<Any?> = arrayOfNulls<Any?>(capacity)  
 private var head: Int = 0  
 private var tail: Int = 0  
 private var size: Int = 0

Listing 5.12 Implementarea CircularBuffer

Pentru a afla valorile corespunzătoare pentru fiecare din indicii descriși anterior a fost nevoia să se facă conversia nivelului de presiune al sunetului măsurat în dB în dB(A), cu ajutorul unor factori de corecție cu pondere A. Odată declarați acești factori, am căutat cea mai apropiata valoare de valoarea nivelului de presiune al sunetului masurata și am returnat valoarea respectiva. În cazul în care nu se găsește o valoare care să fie în șirul de factori declarați se va returna valoarea 0.0.

private fun getCorrectionFactor(spl: Double): Double {  
 // Define the frequency correction factors for A-weighting at different SPL ranges  
 val correctionFactors = mapOf(  
 40.0 to -27.0,  
 50.0 to -16.0,  
 60.0 to -8.0,  
 70.0 to -3.0,  
 80.0 to 0.0,  
 90.0 to 1.2,  
 100.0 to 1.0,  
 110.0 to 0.8,  
 120.0 to 0.7,  
 130.0 to 0.7,  
 140.0 to 0.7  
 )  
  
 // Find the closest lower SPL range în the correction factors map  
 val closestLowerRange = correctionFactors.keys.lastOrNull **{ it** <= spl **}** ?: correctionFactors.keys.first()  
  
 // Retrieve the correction factor for the closest lower range  
 return correctionFactors[closestLowerRange] ?: 0.0  
}

Listing 5.13 Factorii de corecție pentru nivelul de presiune a sunetului pentru calcularea indicilor Lday, Levening și Lnight

Astfel pentru calcularea indicelui dorit am însumat valorile corectate din fiecare minut și am aplicat următoarea formula:

Pentru a calcula valorile în fiecare minut am folosit un obiect de tipul TimerTask [61] care permite rularea anumitor secvențe în mod repetat în funcție de timpul precizat. Folosind Calendar.HOUR\_OF\_DAY am verificat în ce interval orar ne aflam, astfel indicele dorit este calculat și adăugat în baza de date, la fiecare minut schimbând-se valorile în funcție de zgomotul din exterior.

private fun startTimerTask() {  
 if (timer == null) {  
 timer = Timer()  
 val timerTask = object : TimerTask() {  
 override fun run() {  
 val currentHour = Calendar.getInstance().get(Calendar.HOUR\_OF\_DAY)  
 val measurements = measurementsBuffer.toList() // Get the measurements from the buffer  
 if (currentHour în L\_DAY\_START\_HOUR until L\_DAY\_END\_HOUR) {  
 val lday = calculateLday(measurements) // Calculate Lday  
 listener.onLdayCalculated(lday)  
 measurementsBuffer.clear()  
 }

}

timer?.scheduleAtFixedRate(timerTask, 0, 1000 \* 60) // Run every minute

Listing 5.14 Calcularea Lday în intervalul de timp prestabilit în fiecare minut

## Implementarea TensorFlow pentru detecția persoanelor și a mașinilor

În cele ce urmează o să descriu în detaliu felul în care am integrat TensorFlow Audio Classification în cadrul soluției propuse. Primul lucru necesar a fost integrarea dependințelor în fișierul build.gradle după care a fost nevoie să aplic un “Sync Project with Gradle Files”, astfel librăriile cu metodele specific lor au fost descărcate. Următorul lucru a fost adăugarea în directorul assets al fișierelor speech.tflite, respectiv yamnet.tflite, acestea reprezentând modelele pentru detectie.

Prima clasa implementată a fost AudioClassificationHelper ce are în constructor următoarele elemente:

* Context
* AudioClassificationListener
* Denumirea modelului utilizat
* Threshold-ul după care să se facă clasificările
* Numărul de thread-uri
* Numărul de rezultate afișate

Intra-o aplicație Android, Context-ul reprezintă o interfața globala ce oferă informații despre mediul aplicației. La baza este o clasa abstracta a cărei implementare este oferita de sistemul Android. Aceasta permite accesul la resursele specifice aplicației, totodată și operații în cadrul aplicației precum navigarea între activități și fragmente. AudioClassificaitonListener este o interfața creata pentru implementare în care am declarat doua metode și anume: onError în cazul în care apare o eroare să pot primi informații cu privire la ce s-a întâmplat și onResult ce oferă lista de rezultate la durata de timp stabilita. Threshould-ul după care se fac clasificările reprezintă de fapt scorul detecției după care să se afișeze un rezultat. Acesta inițial este setat la 30%. În partea de început se generează opțiunile folosind parametri constructorului clasei.

val options = AudioClassifier.AudioClassifierOptions.builder()  
 .setScoreThreshold(classificationThreshold)  
 .setMaxResults(numOfResults)  
 .setBaseOptions(baseOptionsBuilder.build())  
 .build()  
  
try {  
 // Create the classifier and required supporting objects  
 classifier = AudioClassifier.createFromFileAndOptions(context, currentModel, options)  
 tensorAudio = classifier.createInputTensorAudio()  
 recorder = classifier.createAudioRecord()  
 startAudioClassification()  
} catch (e: IllegalStateException) {  
 listener.onError("Audio Classifier failed to initialize. See error logs for details")  
  
 Log.e("AudioClassification", "TFLite failed to load with error: " + e.message)  
}

Listing 5.15 Crearea opțiunilor de clasificare și a clasificatorului

Fiecare model are nevoie de secvențe audio cu lungimi prestabilite pentru a calcula detecția. Modelul folosit, YAMNET, așteaptă secvențe audio de 0.975 secunde. Pentru a putea oferi secvențe care să satisfacă aceste cerințe am folosit următoarea formula ce returnează valoare în milisecunde pentru a nu se pierde zecimale în urma conversiilor.

val lengthInMilliSeconds = ((classifier.requiredInputBufferSize \* 1.0f) /  
 classifier.requiredTensorAudioFormat.sampleRate) \* 1000

Listing 5.16 Formula pentru calcularea lungimii secvențelor folosind dimensiunea bufferului și sample rate-ul oferit de tensorflow

Pentru afișarea detecțiilor în fragmentul desemnat am creat un adapter și anume ProbabilitiesAdapter. În cadrul adapter-ului am implementat cele trei metode principale necesare:

* override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder
* override fun onBindViewHolder(holder: ViewHolder, position: Int)
* override fun getItemCount(): Int

Pentru a funcționa, un adapter are nevoie de un ViewHolder și de aceea am creat

și o clasa specifică. Un ViewHolder este un view ce are ca scop afișarea elementelor într-o anumita ordine și este bazat pe evenimente. Adapter-ul extinde un RecyclerView care permite adăugarea și eliminarea mai multor ViewHolder-e în timp real în funcție de datele primite. Astfel de fiecare dată când o detecție este făcută ViewHolder-ul se ocupa de afișarea acestuia, în timp ce adapter-ul îl pune în locul atribuit în fragment. Anumiți parametri din cei descriși anterior din cadrul AudioClassificationHelper pot fi modificați, iar pentru acest lucru am creat un view în cadrul controls\_bottom\_sheet.xml. Acesta este un view ce se extinde și se află în partea de jos a fragmentului, vezi Figura 5.12 Bottom Sheet Controls.

A screenshot of a calculator

Description automatically generated

Figura 5.11 Bottom Sheet Controls

## Implementarea fragmentelor din cadrul aplicației

Cele trei fragmente principale implementate în cadrul soluției propuse sunt MainFragment, DetailsFragment și ChartsFragment. Acestea sunt atașate de MainActivity și navigarea între ele cu ajutorul BottomNavigationView, fiecare fragment fiind ascuns în momentul în care un altul urmează să fie afișat.

### Implementarea MainFragment și a claselor auxiliare folosite pentru implementarea acestuia

În MainFragment este creat un obiect de tipul AuidoClassificationListener și implementate metodele onResult() și onError(). Pe onResult se face verificarea ca fragmentul să fie afișat deoarece detecția rulează pe thread-ul principal al aplicației. Astfel în momentul în care ne aflăm în MainFragment pe adapter se adaugă detecția făcută alături de o bară de progres care indică scorul cu care detecția a fost făcută, vezi Listing 5.16. În cazul în care se intră pe funcția de onError, se va afișa un mesaj Toast care să indice eroarea întâmpinată. În cadrul unui fragment în Android, exista trei metode ce pot fi suprascrise în momentul creării view-ului și anume onCreate, onCreateView și onViewCreated. Aceste metode se apelează în momentul în care un fragment este afișat în cadrul aplicației. În fragmentul principal din aplicația prezentată pe onCreate se instanțiază un obiect de tipul SoundLevelMeter pentru a porni măsurarea zgomotului din jur în momentul în care fragmentul este creat veziListing 5.17. Pe metoda onCreateView se face inflate la resursa xml asociată fragmentului și anume fragment\_main.xml. Pe lângă acest lucru aici se instanțiază localizarea pe harta alături de permisiunile pentru obținerea locației, se creează obiectul de tip MapView pentru a putea afișa harta și se face corelarea celorlalte elemente din fragment. Pe metoda onViewCreated se atașează adapterul responsabil pentru afișarea detecțiilor, se inițiază clasificatorul cu modelul YAMNET și se dă start. Tot în această metodă este efectuată logica pentru modificarea rezultatelor afișate prin intermediul BottomSheetLayout astfel atunci când va fi apăsat butonul pentru overlap se face o înmulțire cu 0.25 \* poziția selectată. Atunci când se va selecta plus sau minus la numărul de rezultate afișate se va modifica dimensiunea adapterului (numărul minim de rezultate afișate este 1, cel maxim fiind 5). Pentru selecția threshold-ului pentru acuratețea detecției se va incrementa, respectiv decrementa în funcție de opțiunea selectata, threshold-ul minim fiind 0.2, cel maxim fiind 0.8. Pentru selecția numărului de thread-uri se vor putea crea maxim 4 pentru că procesarea se face direct pe telefon, veziListing 5.18.

private val audioClassificationListener = object : AudioClassificationListener {  
 override fun onResult(results: List<Category>, inferenceTime: Long) {  
 if (isAdded) {  
 requireActivity().runOnUiThread **{** adapter.categoryList = results  
 adapter.notifyDataSetChanged()  
 fragmentMainBinding.bottomSheetLayout.inferenceTimeVal.text =  
 String.format("%d ms", inferenceTime)  
 **}** }  
 }

Listing 5.17 Crearea interfeței responsabilă pentru afișarea detecției

override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 soundLevelMeter = SoundLevelMeter(this)  
}

Listing 5.18 Crearea SoundLevelMeter pe onCreate

fragmentMainBinding.bottomSheetLayout.threadsPlus.setOnClickListener **{** if (audioHelper.numThreads < 4) {  
 audioHelper.stopAudioClassification()  
 audioHelper.numThreads++  
 fragmentMainBinding.bottomSheetLayout.threadsValue.text = audioHelper.numThreads.toString()  
 audioHelper.initClassifier()  
 }  
**}**

Listing 5.19 Incrementarea numărului de thread-uri din BottomSheetLayout

Pentru a nu primi crash-uri în momentul în care se face navigarea între fragmente am suprascris următoarele metode din ciclul de viață al unui fragment în Android: onResume() care se apelează în momentul în care un fragment își continua ciclul de viață, onPause() care este responsabilă de oprirea temporara a acțiunilor din cadrul unui fragment în cazul în care acesta este ascuns, onStart() care este responsabil de lansarea fragmentului și onDestroyView() care se ocupa de ruperea legăturilor dintre fragment și view-urile atribuite acestuia.

În cadrul metodei onResume am reapelat metoda start() pentru obiectul de tipul SoundLevelMeter ce este responsabil pentru detecția nivelului de zgomot și totodată am reinițializat modelul de clasificare audio.

În metoda onPause am apelat metoda stop() pe obiectul soundLevelMeter deoarece fragmentul urma să fie detașat și automat thread-ul pe care lucra acesta să fie compromis. Același lucru este valabil și pentru clasificarea audio, dar și pentru MapView.

În metoda onStart() a fost nevoie să apelez mapView.onStart() pentru a accesa resursele harții ulterior.

Pe onDestroyView() a fost necesar să distrug legăturile între fragment și resursele disponibile acestuia pentru a se reinițializa la fiecare lansare a activității.

Ultimele funcții ce trebuie menționate din MainFragment sunt onMapReady și onSPLMeasured. Funcția onMapReady a trebuit implementată deoarece fragmentul implementează OnMapReadyCallback care se ocupa de atașarea harții de tip GoogleMap în obiectul de tip MapView, astfel obiectul din clasa este inițializat și tot aici se preia locația curenta a fiecărui utilizator, urmând ca focusul să se facă în locul în care utilizatorul curent se afla, vezi Listing 5.19. Functia onSPLMeasured este implementată deoarece fragmentul implementează SoundLevelMeter.Listener și în cazul acesta se verifica dacă fragmentul este afișat și TextView-ul responsabil pentru afișarea nivelului de presiune a sunetului se modifica în funcție de valorile detectate de către microfonul telefonului și procesate ulterior, vezi Listing 5.20. Nu în ultimul rând aici mai sunt trecute și funcțiile pentru calcularea indicilor Lday, Levening și Lnight.

override fun onMapReady(googleMap: GoogleMap) {  
 mMap = googleMap  
 SensorHelper().getCurrentLocation(requireActivity(), locationRequest, mMap)  
}

Listing 5.20 Implementarea metodei onMapReady

override fun onSPLMeasured(spl: Double) {  
 if (isAdded) {  
 requireActivity().runOnUiThread **{** decibelTextView.text = String.format("%.1f Sound Pressure Level", spl)  
 if (spl > x) {  
 x = spl  
 }  
 soundLevelMeter.checkArea(mMap, requireContext())  
 **}** }

Listing 5.21 Implementarea metodei onSPLMeasured

### Implementarea DetailsFragment și a claselor auxiliare folosite pentru implementarea acestuia

### Pentru a implementa DetailsFragment a fost nevoie să creez un ViewModel și un Adapter. ViewModel-ul asociat fragmentului se numește DetailsViewModel. Acesta extinde ViewModel() din Android Achitecture Components și conține o listă de UserDetails, vezi Listing 5.21. Acesta din urmă este un model creat ce preia din baza de date emailul utilizatorului, locația acestuia, nivelul de zgomot din jurul său și momentul în care a fost detectata o mașina sau o persoana în jur. DetailsViewModel este responsabil pentru preluarea datelor din Firebase Realtime Database și adăugarea lor în lista de MutableLiveData.

MutableLivedata este o subclasa a LiveData, care deține modelul observatorului. Am folosit MutableLiveData pentru ca aceasta, spre deosebire de LiveData, oferă metode suplimentare pentru a-și modifica valoarea. Este proiectata să mențină și să emită o singură valoare la un moment dat și anume un MutableSet. MutableLiveData permite comunicare fără întreruperi între ViewModel și componentele interfeței grafice prin expunerea datelor într-un flux de observare. Avantajul principal este ca prin aceasta observatorii sunt notificați când valoarea stocată se modifică ori de cate ori este emisa o nouă valoare, astfel obiectele aflate în adapter sunt actualizate cu ultimele valori. În cadrul actualizării datelor am folosit metoda postValue(), vezi Listing 5.22. Aceasta comuta automat execuția la thread-ul principal pentru a furniza valoarea actualizata.

val userDetailsList = MutableLiveData<MutableSet<UserDetails>>(null)

Listing 5.22 Lista declarata pentru a prelua datele utilizatorilor în cadrul DetailsViewModel

userDetailsList.postValue(userList)

Listing 5.23 Execuția postValue cu datele emise pentru fiecare utilizator în parte

Pentru afișarea datelor am folosit un adapter pe care l-am denumit ListItemAdapter ce adaugă elemente de tipul ListItemViewHolder. În ListItemViewHolder am creat mai multe TextView-uri responsabile pentru menținerea datelor din UserDetails. Layout-ul creat pentru ViewHolder se regăsește în fișierul list\_view\_item.xml, vezi Figura 5.13.

În cadrul DetailsFragment am inițializat ViewModel-ul fragmentului, vezi Listing 5.23. Acesta trebuie declarat explicit deoarece un ViewModel trebuie asignat unui singur fragment sau unei singure activități pentru a nu se periclita datele. Fragmentul implementează la rândul sau, precum MainFragment, SoundLevelMeter.Listener, lucru ce ne obligă să implementăm metodele onSPLMeasured(), onLdayCalculated(), onLeveningCalculated() și onLnightCalculated() pentru a obține nivelul de zgomot și indicii pentru poluare fonică și în acest fragment.

În metoda onViewCreated, adapter-ul este creat și atribuit către recyclerView, vezi Listing 5.24,ce se afla în fragment\_details.xml. După aceea am apelat observe pe MutableLiveData din ViewModel pentru a prelua datele din baza de date și actualizează lista din adapter, vezi Figura 5.14.

A screenshot of a phone

Description automatically generated with medium confidence

Figura 5.12 Layout-ul pentru ListItemViewHolder

private val viewModel by lazy **{** ViewModelProvider(this)[DetailsViewModel::class.java]  
**}**

Listing 5.24 Inițializarea DetailsViewModel în DetailsFragment

val detailsAdapter = ListItemAdapter()  
  
recyclerView.adapter = detailsAdapter

Listing 5.25 Instanțierea adapter-ului și atribuirea catre recyclerView

A picture containing text, screenshot, diagram, font

Description automatically generated

Figura 5.13 Preluarea detaliilor utilizatorilor și actualizarea lor pe interfața grafică

### Implementarea ChartsFragment și a claselor auxiliare folosite pentru implementarea acestuia

În cadrul acestui fragment am ales să creez cate un grafic pentru fiecare utilizator în care să fie afișați indicii Lday, Levening și Lnight ai fiecărui utilizator. Pentru a face acest lucru și pentru a actualiza graficele în fiecare minut odată cu calculul fiecărui indice am folosit un ViewModel denumit InfoChartViewModel. La fel cum am menționat anterior, am folosit și aici, de asemenea, MutableLiveData pe un data class [62], InfoChartDetails. Datele din InfoChartDetails sunt email-ul utilizatorului, regiunea în care se afla și lista de intrări pe graficele alese.

Am ales să implementez grafice de forma PieChart. Implementarea lor se face în clasa ListInfoChartViewHolder, veziFigura 5.15. Pentru configurarea graficelor am creat metoda configPieChart ce primește ca parametru o lista de PieEntry. În aceasta metoda se creează un nou set PieDataSet cu lista de intrări primita ca parametru. Apoi declar lista de culori pentru fiecare element din chart, culoarea textului și dimensiunea acestuia. După care atribui obiectului de tipul PieChart datele din PieDataSet, setez graficul ca fiind animat, se setează o descrierea al acestuia iar apoi se apelează metode de invalidate() și postInvalidate() pentru a se face o reîmprospătare a graficului pe interfața grafică în momentul în care datele sunt actualizate fără a fi nevoie să se refacă navigarea spre ChartsFragment.

A screenshot of a phone

Description automatically generated with low confidence

Figura 5.14 Graficele PieChart folosite

private fun configPieChart(pieChartEntries: List<PieEntry>) {  
  
 val pieDataSet = PieDataSet(pieChartEntries, "Categories")  
 val colorList: MutableList<Int> = mutableListOf()  
 for (joyfulColor în ColorTemplate.COLORFUL\_COLORS) {  
 colorList.add(joyfulColor)  
 }  
 pieDataSet.colors = colorList  
 pieDataSet.valueTextColor = Color.BLACK  
 pieDataSet.valueTextSize = 16f  
  
 pieChart.data = PieData(pieDataSet)  
 pieChart.animate()  
 pieChart.description = Description().apply **{** text = "\n\nChart based on average noise for each moment of the day"  
 **}** pieChart.data.notifyDataChanged()  
 pieChart.notifyDataSetChanged()  
 pieChart.invalidate()  
 pieChart.postInvalidate()  
}

Listing 5.26 Metoda configPieChart prin care se populează graficele

În InfoChartViewModel locația utilizatorilor se preia din baza de date sub forma de latitudine și longitudine, prezentând doua valori de tipul double. Pentru afișarea adresei cu nume de strada, număr, oraș și cod poștal am folosit Geocoder [63]. Astfel, în cazul în care din coordonatele utilizatorului se poate translata într-o locație cum am menționat anterior o să se configureze, iar în cazul în care nu o să se pota face conversia locația va fi trecuta ca null, vezi Listing 5.26. În Kotlin, simbolul „?” după o instrucțiune marchează faptul ca rezultatul poate fi null, astfel tipurile de date pot fi adnotate ca și null în momentul în care sunt declarate.

val geocoder = Geocoder(context, Locale.getDefault())  
val address = geocoder.getFromLocation(latitude, longitude, 1)?.get(0)?.getAddressLine(0)

Listing 5.27 Conversia locației din latitudine și longitudine în adresa

În ChartsFragment pe onViewCreated se instanțiază adapter-ul și este atribuit la recyclerView, urmând să se facă observe() pe datele din InfoChartViewModel și să se actualizeze adapterul.

## Scrierea și preluarea datelor din Firebase Realtime Database

Pentru scrierea în baza de date a fost nevoie de preluarea adresei URL a bazei de date, aceasta fiind făcută pe serverele de Europa ale Firebase. Pentru a avea acces la aceasta am integrat următoarea dependință în fișierul build.gradle, vezi Listing 5.1. După care, în fiecare fișier în care am avut nevoie de URL-ul specificat mai sus l-am declarant în companion object-ul fiecărei clase sub denumirea de DB\_URL.

implementation 'com.google.firebase:firebase-database-ktx:20.1.0'

Listing 5.28 Integrarea dependinței pentru folosirea Firebase Realtime Database

Spre deosebire de o baza de date clasica SQL, Firebase Realtime Database are structura unui fișier JSON, iar felul în care am ales să implementez structura acesteia este vizibil în Figura 5.16. Am ales ca User să fie nodul principal al sistemului, acesta având ca și copii noduri de reprezintă id-urile fiecărui utilizator. Sub fiecare id se găsește un nod ce reprezintă momentul detecției unui autovehicul, a unei persoane, email-ul utilizatorului, nivelul de presiune a zgomotului și indicii de măsurare e poluării fonice Lden.

A screenshot of a computer

Description automatically generated with medium confidence

Figura 5.15 Configurația bazei de date

Accesul la baza de date se face printr-un obiect de tipul FirebaseDatabase [64]. Acesta permite obținerea unei instanțe a bazei de date ce poate fi folosită pentru scrierea de noi valori și actualizarea acestora.

Pentru adăugarea locației utilizatorului în baza de date, în clasa SensorHelper, în metoda în care accesez locația curentă a acestuia, fac verificarea să fie logat, după care sub nodul User din baza de date o să fie adăugat un nod cu id-ul unic autogenerat de Firebase Authentication, iar sub acesta se vor crea nodurile „LatLng”, ce are copiii „Latitude” și „Longitue”, și „Email”, vezi Listing 5.28. În lista coordinates sunt memorate ultimele valori cu privire la latitudinea și longitudinea la care se afla utilizatorul. Accesul la baza de date se face prin apelarea getInstance(DB\_URL). Implicit se poate apela doar cu getInstance() dacă locația bazei de date este pe varianta de locație standard (Statele Unite ale Americii), dar fiind făcută pe locația de Europa a fost nevoie apelarea cu parametru. Prin apelarea setValue() sub nodul specificat se adăuga valorile dorite. Aceasta metoda actualizează mereu datele de sub nodul curent.

if (FirebaseAuth.getInstance().currentUser != null) {  
 val userConnected = FirebaseAuth.getInstance().currentUser?.uid  
 val database = userConnected?.let **{** FirebaseDatabase.getInstance(DB\_URL).getReference("User").child(**it**) **}** database?.child("LatLng")?.child("Latitude")?.setValue(coordinates[0])  
 database?.child("LatLng")?.child("Longitude")?.setValue(coordinates[1])  
 database?.child("Email")?.setValue(FirebaseAuth.getInstance().currentUser?.email)  
}

Listing 5.29 Adaugarea datelor despre locația și email-ul utilizatorului în baza de date.

Datele cu privire la detecția de persoane sau autovehicule făcută de către un utilizator se face sub forma de HashMap<String, String> unde cheia reprezintă categoria detectată, iar valoarea reprezintă data cu formatul yyyy-mm-dd hh:mm:ss la care a fost făcută detecția respectivă. Pentru a înregistra detecțiile fiecărui utilizator am creat nodurile „CarDetection” și „PersonDetection” sub care am adăugat și actualizat continuu cu valoarea din HashMap-ul menționat anterior, vezi Listing 5.30. Pentru adăugarea datei exacte la care a fost făcută detecția am folosit un obiect de tipul Date care apoi a fost formatat cu SimpleDateFormat().

val carDetectionEvent: HashMap<String, String> = HashMap()  
val currentDate = Date()  
val dateFormat = SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.getDefault())  
val formattedDate = dateFormat.format(currentDate)  
carDetectionEvent[CAR] = formattedDate  
val userConnected = FirebaseAuth.getInstance().currentUser?.uid  
val database = userConnected?.let **{** FirebaseDatabase.getInstance(SensorHelper.DB\_URL).getReference("User").child(**it**) **}**database?.child("CarDetection")?.setValue(carDetectionEvent)

Listing 5.30 Adăugarea unui eveniment de tipul car detection în baza de date în funcție de detecția făcută de utilizator

Pentru adăugarea nivelului de presiune al sunetului care este măsurat continuu am ales să adaug valoarea maxima înregistrata de un utilizator deoarece acesta variază în funcție de specificațiile fiecărui dispozitiv mobil în parte. Acesta a fost adăugat în implementarea metodelor onSPLMeasured sub nodul asociat fiecărui utilizator cu denumirea de soundLevel și valoarea acestuia.

val userConnected = FirebaseAuth.getInstance().currentUser?.uid  
val database = userConnected?.let **{** FirebaseDatabase.getInstance(SensorHelper.DB\_URL).getReference("User").child(**it**) **}**database?.child("soundLevel")?.setValue(x)

Listing 5.31 Adăugarea valorii nivelului de presiune a sunetului înregistrat de fiecare utilizator

Adăugarea indicilor de măsurare a poluării fonice pe parcursul zilei se efectuează în clasa SoundLevelMeter. Se face o verificare să se verifice ca valoarea calculata este diferita de isNaN() (Not-a-Number) și totodată diferita de plus și minus infinit, deoarece în urma calculelor efectuate se pot obține valori ce nu au un factor de corecție A prestabilit. Acest lucru poate fi cauzat de diferența microfoanelor din diferitele dispozitive mobile și nivelul lor de calibrare. După ce sunt efectuate aceste verificări, valoarea este adăugată sub nodul corespondent momentului zilei în care măsurătorile au fost făcute.

if (!lEvening.isNaN() && lEvening != Double.NEGATIVE\_INFINITY && lEvening != Double.POSITIVE\_INFINITY) {  
 val userConnected = FirebaseAuth.getInstance().currentUser?.uid  
 val database = userConnected?.let **{** FirebaseDatabase.getInstance(SensorHelper.DB\_URL).getReference("User").child(**it**) **}** database?.child("L\_Evening")?.setValue(lEvening)  
}

Listing 5.32 Adăugarea indicelui Levening în baza de date

Preluarea valorilor se face în clasele ViewModel. Acest lucru permite manipularea lor și modificare interfeței grafice în mod continuu fără a altera valorile în vreun fel. Pentru a face acest lucru am adăugat pe referința la baza de date un obiect de tipul ValueEventListener. Acesta impune implementarea a doua metode și anume onDataChange și onCancelled. Pe onDataChange se verifica dacă DataSnapshot-ul exista iar apoi se merge pe fiecare nod copil al referinței. Atunci când se face getValue() pe un nod datele returnate sunt considerate native și acestea pot fi: String, Double, Long, Boolean, List<Object>, Map<String, Object>. Pentru preluarea latitudinii și longitudinii am accesat nodul „LatLng”, iar valorile le-am extras în double, restul valorilor fiind preluate ca și string. Aceasta abordare permite ca în momentul în care orice valoare se modifică, ValueEventListener-ul să notifice acest lucru, să se facă un apel către baza de date și să se preia noile valori actualizate.

dbRef.addValueEventListener(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 if (snapshot.exists()) {  
 userList.clear()  
 snapshot.children.forEach **{** val latitude = **it**.child("LatLng").child("Latitude").getValue(Double::class.java)  
 val longitude = **it**.child("LatLng").child("Longitude").getValue(Double::class.java)  
 if (latitude != null && longitude != null) {  
 val userEmail = **it**.child("Email").value.toString()  
 val location = LatLng(latitude, longitude).toString()  
 val carDetection = **it**.child("CarDetection").child("Car").value.toString()  
 val personDetection = **it**.child("PersonDetection").child("Speech").value.toString()  
 val soundLevel = **it**.child("soundLevel").value.toString()  
  
 val userDetails = UserDetails(userEmail, location, soundLevel, personDetection, carDetection)  
 userList.add(userDetails)  
 userList.distinctBy **{** userEmail **}** }  
 **}** userDetailsList.postValue(userList)  
 }  
 }

Listing 5.33 Preluarea datelor din DetailsViewModel

## Integrarea Firebase Functions

Pentru obținerea informațiilor cu privire la id-urile utilizatorilor și accesarea nodurilor din baza de date a fost nevoie să integrez Firebase Functions, deoarece integrarea de Firebase Admin SDK [65] presupunea pierderea opțiunilor de înregistrare și logare a utilizatorilor. Astfel a fost nevoie să îmi declar propria funcție în JavaScript și îi fac deploy din linia de comanda, vezi Listing 5.33. Astfel am reușit să folosesc aceeași funcționalitate oferită de Admin SDK și să funcționeze în continuare și serviciile de înregistrare și logare. După ce am făcut deploy funcției, am putut efectua cereri de tipul HTTP direct de pe serverele celor de la Google.

const functions = require("firebase-functions");

const admin = require("firebase-admin");

admin.initializeApp();

exports.getAllUsers = functions.https.onCall(async (data, context) => {

try {

const listUsersResult = await admin.auth().listUsers();

const users = listUsersResult.users.map((user) => {

return {

uid: user.uid,

email: user.email,

};

});

return users;

} catch (error) {

console.error(error);

throw new functions.https.HttpsError("internal", "Error retrieving users");

}

});

Listing 5.34 Functia getAllUsers

Pentru a face apelul către serverele celor de la Google și a accesa funcția creată anterior e nevoie de instanțierea unui obiect de tipul FirebaseFunctions, mai apoi apelând-se pe el getHttpsCallable(”getAllUsers”). Astfel metoda returnează un o lista de perechi de tipul user.id și user.email. Implementarea codului pentru obținerea detaliilor cu privire la fiecare utilizator pentru a se putea face localizarea lor pe harta se găsește în clasa SensorHelper, vezi Listing 5.34. După ce apelul este efectuat se verifica dacă acesta a fost completat folosind addOnCompleteListener, urmând apoi să se verifice și dacă a fost făcut cu succes. În caz afirmativ, datele provenite în urma task-ului sunt convertite din perechea de user.id și user.email intr-o lista de HashMaps ce au atât cheile cât și valorile de tipul string. În cadrul funcției ce face apel pentru a obține datele menționate anterior se verifică dacă utilizatorul curent se află în lista respectivă, urmând apoi ca localizarea pe harta să se facă cu focus pe acesta, astfel restul utilizatorilor doar sunt amplasați în pozițiile în care se afla.

val functions = FirebaseFunctions.getInstance()  
val getAllUsers = functions.getHttpsCallable("getAllUsers")  
  
getAllUsers.call()  
 .addOnCompleteListener **{** task **->** if (task.isSuccessful) {  
 val result = task.result?.data as List<HashMap<String, String>>?  
 if (result != null) {  
 map.clear()  
 for (user în result) {  
 val uid = user["uid"] as String?  
 val email = user["email"] as String?  
 Log.d("TAG", "User ID: $uid")  
 Log.d("TAG", "Email: $email")  
 if (user["uid"]?.equals(FirebaseAuth.getInstance().currentUser?.uid) == true) {  
 locateUser(user["uid"], map, true)  
 } else {  
 locateUser(user["uid"], map, false)  
 }  
 }  
 }  
 }

Listing 5.35 Apelarea funcției getAllUsers și localizarea utilizatorilor pe harta

# Testare și validare

Testarea este procesul prin care cerințele unui proiect, indiferent de obiectivele acestuia, sunt validate sau nu pentru a verifica dacă rezultatele obținute se apropie de cele stabilite inițial.

În capitolul ce urmează o să prezint funcționalitățile aplicației Android în detaliu pentru a verifica ca am ajuns la rezultatele dorite. Se va testa fiecare caz de utilizare și se va pune accent pe detecția de persoane și de mașini dar și de calcularea indicilor de poluare fonică la momentele zilei stabilite. Pentru a evalua clasificatorul o să folosesc următoarele metrici: specificitatea, precizia, acuratețea, sensibilitatea și F-measure.

## Testarea aplicației Android

Aplicația a fost testată pe mai multe dispozitive, dar în special pe un Samsung Galaxy S10e cu versiune de Android 12 și pe un Samsung Galaxy S9 cu Android 10. În Figura 6.1 se poate observa ca atunci cand aplicația este instalată pentru prima oară, o să fim redirecționați spre pagina de register, unde odată contul creat cu succes, va fi afișat un toast care să indice acest lucru. Mai departe sunt cerute permisiunile de înregistrare audio și accesare a locației de către aplicație. În cazul în care opțiunea „Don’t allow” este selectata, detecțiile și localizarea pe harta nu vor avea loc. În cazul în care opțiunea „Only this time” este selectata, în instanța curentă a aplicației se vor putea accesa funcționalitățile acesteia, însă la următoarea lansare va fi nevoie din nou de oferirea lor. Opțiunea „While using the app” permite accesarea funcționalităților fără a mai fi nevoie de reconfirmarea lor la o redeschidere a aplicației.

A screenshot of a phone

Description automatically generated with low confidence

Figura 6.1 Lansarea aplicației pentru a ajunge în activitatea principala

După ce permisiunile aplicației au fost oferite, utilizatorul poate observa pe harta locația să curenta, a altor utilizatori și detecțiile făcute în jurul sau. Pe harta se poate naviga și se poate observa amplasamentul fiecărui utilizator. Mai exista un buton predestinat traficului din oraș, care odată apasat modifică harta, colorând străzile în funcție de nivelul de trafic de pe acestea. În partea de jos a acestui fragment este prezent „panoul de comanda” pentru detecția făcută de TensorFlow. Cel de al doilea fragment, menține în format de lista detaliile fiecărui utilizator la momentul în care s-a făcut o detecție și nivelul maxim de zgomot din zona acestuia. Ultimul fragment reprezintă reprezentarea indicilor de poluare fonică pe grafice atribuite fiecărui utilizator alături de locația să sub forma de adresa.

A screenshot of a phone

Description automatically generated with low confidence

Figura 6.2 Fragmentele aplicației și cazurile de utilizare ale acesteia

## Testarea actualizării bazei de date în timp real

Elementele actualizate în cadrul Firebase Realtime Database pot fi observate vizual atunci când se modifica fiind încadrate de un chenar pentru a atenționa acest lucru. În cazul în care nodurile nu sunt expandate, chenarul va fi acționat asupra nodului părinte. Pentru testarea calculării indicilor de poluare fonică, în aplicația implementată, în intervalele menționate în capitolele de mai sus și am verificat să existe modificări în fiecare minut. Valorile fiind de tipul double, chiar dacă de la un minut la altul nivelul de zgomot părea asemănător, valorile își schimbau zecimalele după cum se aștepta să se întâmple.

A screenshot of a computer

Description automatically generated with medium confidence

Figura 6.3 Modificarea valorilor în baza de date

## Testarea detectiei TensorFlow

Pentru a testa clasificatorul de sunete cu ajutorul TensorFlow Audio Classification am folosit următoarele metrici comune și anume: specificitatea, sensibilitatea, precizia, acuratețea și F-Measure. Pentru testare am folosit un set audio de 60 de înregistrări ale sunetelor de mașini și persoane, un set audio având o lungime de 0.975 secunde, aceasta fiind lungimea secvenței cu care acționează modelul YAMNET. Pentru calcularea metricilor menționate anterior am avut nevoie de un număr de predicții adevărate și pozitive (TruePositive), adevărate și negative (TrueNegative), false și pozitive (FalsePositive), false și negative (FalseNegative).

Pentru obținerea valorilor pentru TP, TN, FP și FN din 60 de secvențe audio am utilizat Android Log pentru a marca momentul în care se efectuează o detecție. În Figura 6.4 Detecția persoanelor cu 80% threshold a fost setat threshold-ul la valoarea de 0.8 astfel afișându-se valori doar în momentul în care acuratețea era peste 80%. În se poate observa numărul de detecții pentru acuratețe de peste 30% din 60 de secvențe. Valorile metricilor sunt prezente în Tabelul 6.1, iar metricile menționate anterior în Tabelul 6.2.

A screenshot of a computer

Description automatically generated with medium confidence

Figura 6.4 Detecția persoanelor cu 80% threshold

A screenshot of a computer

Description automatically generated with medium confidence

Figura 6.5 Detecția persoanelor cu 30% threshold

A picture containing text, screenshot

Description automatically generatedFigura 6.6 Detecția când nu s-a aflat nicio persoana în cele 60 de secvențe

|  |  |  |
| --- | --- | --- |
|  | Positive | Negative |
| True | 95 | 53 |
| False | 7 | 25 |

Tabelul 6.1 Matricea de confuzie

|  |  |
| --- | --- |
| Metrica de evaluare | Valoare |
| Specificitate | 88,33% |
| Sensibilitate | 79,16% |
| Precizie | 93,13% |
| Acuratețe | 82,22% |
| F-measure | 85,57% |

Tabelul 6.2 Metricile de evaluare

Pentru F-measure, valoarea de 85,57% reprezintă o performanță satisfăcătoare a clasificatorului pentru identificarea exemplelor atât pozitive cât și negative într-un mod cât mai corect. Exista diverse limitări ale modelului și totodată ale hardware-ului pe care se lucrează, diferența fiind între modelele de telefoane disponibile și tipul microfoanelor integrate ale acestora. Cu toate acestea, clasificatorul oferă rezultate prin care se pot identifica aglomerări urbane și se pot efectua măsurători în zonele de interes ale orașelor.

# Manual de instalare și utilizare

În acest capitol o să prezint procesul de instalare pentru soluția propusa din perspectiva unui utilizator. Totodată, o să prezint modul de funcționare al acesteia alături de câteva capturi de ecran sugestive de pe dispozitivul Android de pe care am conceput aplicația.

## Instalarea aplicației

Pentru a se putea folosi aplicația este nevoie de un dispozitiv cu sistem de operare Android 8 Oreo sau mai actual. Pentru instalarea direct din Android Studio a aplicației trebuie activata opțiunea de „Developer Options”din setările telefonului urmata de activarea opțiunii „USB Debugging”. Astfel, conectând dispozitivul la un laptop sau calculator o să avem opțiunea din Android Studio de a selecta dispozitivul pe care dorim să efectuam instalarea.

A screenshot of a computer

Description automatically generated with medium confidence

Figura 7.1 Selectarea dispozitivului pe care să ruleze aplicația

Pentru instalarea aplicației prin intermediul fișierului cu extensia apk trebuie efectuați următorii pași:

1. Se accesează linkul [66] și se descarcă fișierul.
2. Oferirea de permisiuni de instalare din surse străine, opțiune de setare ce apare în momentul instalării
3. Acceptare instalare și oferire permisiuni de rulare aplicației

## Mod de utilizare

La deschiderea aplicației, implicit se va deschide ecranul de Register, considerându-se ca un nou utilizator ce nu are cont creat deja. În cazul în care acesta are un cont, are opțiunea de a selecta „Already have an account? Login” ce îl va redirecționa spre ecranul de Login. Odată introduse datele acesta este autentificat și redirecționat către ecranele principale. Daca autentificarea s-a efectuat deja pe dispozitiv, la următoarele lansări ale aplicației utilizatorul o să fie redirecționat direct către ecranele principale.

În fragmentul Map, în jumătatea de sus a ecranului se afla harta unde utilizatorul este localizat și de asemenea pot observa și alte markere în locul în care sunt și alți utilizatori. Imediat sub harta se poate observa butonul „Show Traffic” care odată apăsat permite vizualizarea traficului din zona. Lângă acesta este și butonul de „Hide Traffic” care oprește vizualizarea traficului din zonă. Deasupra butoanelor se afla o etichetă responsabil cu afișarea în permanenta a nivelului de presiune a sunetului din jurul utilizatorului, urmând ca sub cele doua butoane să apară detecțiile în momentul în care sunt făcute. Acestea se pot observa sub eticheta de „Speech” și „Car” alături de o bară de progres care indica nivelul de precizie cu care s-a făcut predicția. În partea de jos a ecranului se poate observa o iconiță cu o săgeata ce semnalează faptul ca este prezent un meniu de setări sub aceasta. Odată deschis meniul se pot observa și modifica diferite metrici cu privire la detecțiile făcute. Se poate oferi timpul de inferență, pe ce este delegat modelul să ruleze, anume CPU și GPU, nivelul de suprapunere de la 0% la 75%. După care se pot observa intervalele pentru numărul maxim de rezultate care variază de l la 5, pragul de acuratețe cu care se face detecție ce variază de la 20% la 80%, respectiv numărul maxim de fire de lucru pe care să se facă detecțiile ce pot varia între 1 și 4. În fragmentul Details se pot observa sub forma de lista locul în care se află utilizatorii aplicației și nivelul maxim de zgomot înregistrat, detecțiile făcute și locația sub forma de latitudine și longitudine. În ultimul fragment și anume Charts, se pot observa grafice cu date provenite de la fiecare utilizator alături de adresa la care a fost făcută detecția.

A screenshot of a login screen

Description automatically generated with medium confidence

Figura 7.2 Ecranul de autentificare cu opțiunea de a naviga pe ecranul de înregistrare în cazul în care utilizatorul nu are un cont creat

A screenshot of a phone

Description automatically generated with medium confidence

Figura 7.3 Ecranul map unde se poate observa traficul în zona, nivelul de presiune al sunetului și detecțiile din jur

A screenshot of a phone

Description automatically generated with medium confidence

Figura 7.4 Panoul de control pentru modificarea parametrilor detecției

A screenshot of a phone

Description automatically generated with medium confidence

Figura 7.5 Ecranul details unde se pot observa detaliile fiecărui utilizator

A picture containing text, screenshot, software, website

Description automatically generated

Figura 7.6 Ecranul charts unde se pot vizualiza indicii de poluare fonică din jurul fiecărui utilizator în timp real

# Concluzii

În cadrul acestui capitol o să prezint contribuțiile personale cu privire la realizarea proiectului, concluziile și observațiile deduse cu privire la detecția făcută de clasificator, măsurătorile făcute cu anumite dispozitive mobile. Nu în ultimul rând, o să fie prezentate și eventuale îmbunătățiri ce pot fi aduse soluției propuse.

## Contribuții personale

Proiectul consta în implementarea unei aplicații destinate dispozitivelor mobile cu sistem de operare Android ce poate detecta activitatea persoanelor și a autovehiculelor din jur, măsurarea nivelului de presiune a sunetului și a anumitor indici pentru detecția poluării fonice în anumite momente ale zilei.

Din punct de vedere al contribuțiilor personale, în primul rând se poate menționa implementarea folosind arhitectura Model-View-ViewModel, a bazei de date de tipul NoSQL în Firebase Realtime Database, integrarea sistemului de autentificare și logare cu ajutorul Firebase Authentication și apelarea metodelor la care s-au făcut deploy pe serverele Google prin intermediul Firebase Functions.

O a doua contribuție ce merita menționata consta în integrarea Google Maps API pentru localizarea pe harta atât a utilizatorului curent cât și al celorlalți utilizatori din cadrul sistemului și vizualizarea în timp real a locației tuturor cât și integrarea modulului de vizualizare a densității traficului în cadrul orașului.

A treia contribuție majora consta în integrarea framework-ului open source Tensorflow Audio Classification prin care am realizat detecția persoanelor și a autovehiculelor cu ajutorul microfonului incorporat al dispozitivului mobil. Tot aici trebuie menționat și faptul ca permanent se face o monitorizare a nivelului de zgomot din exterior după o calibrare efectuata microfonului și astfel se obțin indici prin care se poate detecta poluarea fonică calculata din minut în minut în fiecare moment al zilei. Efortul depus în efectuarea clasificării și măsurarea nivelului de decibeli din exterior a fost unul semnificativ deoarece procesarea se face pe dispozitiv, astfel toate procesele au fost efectuate pe fire de lucru separate pentru a nu cauza probleme la rularea aplicației.

## Analiza rezultatelor obținute

Implementarea aplicației a fost făcută după arhitectura Model-View-ViewModel, interfața grafică fiind implementată în fișiere layout cu extensia XML și se comporta conform așteptărilor la apelurile făcute. Fiecare utilizator poate să modifice setările din cadrul detecției făcute pentru a obține rezultate cu o acuratețe cât mai ridicata. Orice date, pornind de la locație, detecții efectuate, nivelul de presiune al sunetului sau indicii de poluare fonică de pe parcursul unei zile, sunt modificate, acestea sunt automat actualizate și în baza de date și în același timp se face actualizarea și în cadrul fiecărei instanțe de aplicație ce rulează în momentul respectiv. Clasificatorul de sunete poate beneficia de ulterioare îmbunătățiri, cu toate ca momentan echilibrul dintre precizie și sensibilitate este unul ridicat și anume 85,57%.

## Dezvoltări ulterioare

Printre posibilele dezvoltări ce le poate avea aplicația se numără lansarea acesteia în mod oficial, dezvoltarea acesteia și pe alte platforme, adăugarea de noi funcționalități și migrarea proceselor grele pe un server remote.

Pentru lansarea publica a aplicației se poate folosi Google Play Store, ce reprezintă platforma oficiala de unde se pot instala și actualiza aplicațiile Android.

Dezvoltarea și pe alte platforme presupune scrierea aplicației ca aceasta să ruleze și pentru dispozitive cu sistem de operare iOS. Integrarea modelului și a graficelor din cazul aplicației prezentate pot fi migrate către iOS alături de modelul arhitectural.

Legat de adăugarea de noi funcționalități aici pot fi menționate diverse aspecte. Unul dintre acestea ar fi adăugarea de noi elemente în de interes pentru a fi detectate cu ajutorul clasificatorului. Un alt aspect ce ar putea fi menționat ar fi adăugarea procesării comune în cazul în care mai mulți utilizatori sunt în aceeași zona pentru a spori gradul de acuratețe. Astfel în adunările mai mari de utilizatori atât predicțiile făcute cât și detecția poluării fonice o să fie de o precizie mult mai ridicată.

Nu în ultimul rând, trebuie menționat faptul ca adăugarea proceselor ce necesita mai multe putere operațională ar putea fi migrate pe un server la distanță pentru a nu consuma resursele dispozitivului mobil și pentru a nu îl suprasolicita.

# Bibliografie

|  |  |
| --- | --- |
| [1] | [P. Ross, Top 11 technologies of the decade, Spectrum, IEEE 48 (2011) 27–63](https://ieeexplore.ieee.org/abstract/document/5676379) |
| [2] | [N. Eagle, A. Pentland, D. Lazer, Inferring friendship network structure by using mobile phone data, Proceedings of the National Academy of Sciences 106 (2009) 15274–15278.](https://www.pnas.org/doi/full/10.1073/pnas.0900282106) |
| [3] | [M.C. Gonzalez, C.A. Hidalgo, A.-L. Barabasi, Understanding individual human mobility patterns, Nature 453 (2008) 779–782](https://www.nature.com/articles/nature06958) |
| [4] | [G. Chittaranjan, J. Blom, D. Gatica-Perez, Mining large-scale smartphone data for personality studies, Personal and Ubiquitous Computing 17 (2013) 433–450.](https://link.springer.com/article/10.1007/s00779-011-0490-1) |
| [5] | C. Pronello, C. Camusso, A review of transport noise indicators, Transp. Rev. 32 (2012) 599–628, <https://doi.org/10.1080/01441647.2012.706332> . |
| [6] | [Mark Brink, Beat Schäffer, Reto Pieren, Jean Marc Wunderli Conversion between noise exposure indicators Leq24h, LDay, LEvening, LNight, Ldn and Lden: Principles and practical guidance](https://www.sciencedirect.com/science/article/abs/pii/S1438463917304819?casa_token=iBVDSvkkkwYAAAAA:kdfA_X9jgL7ASrfCrIKo8fSq3tD1gIXB8esFsxIb7CzwOfaKnlvc85db4FivfMfAP9rmW4ic) |
| [7] | A.-M.O. Mohamed, E.K. Paleologos, F.M. Howari, Noise pollution and its impact on human health and the environment, Pollut. Assess. Sustain. Pract. Appl. Sci. Eng. (2021) 975–1026, <https://doi.org/10.1016/b978-0-12-809582-9.00019-0> . Elsevier. |
| [8] | J. Púčik, P. Kubinec and O. Ondřáček, "FFT with modified frequency scale for audio signal analysis," 2014 24th International Conference Radioelektronika, Bratislava, Slovakia, 2014, pp. 1-4, doi: 10.1109/Radioelek.2014.6828482. |
| [9] | [K.R. Rao, D.N. Kim, J.J. Hwang, Fast Fourier Transform - Algorithms and Applications, Springer, 2010.](https://link.springer.com/book/10.1007/978-1-4020-6629-0) |
| [10] | L. Sanchez, L. Munoz, ˜ J.A. Galache, P. Sotres, J.R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis, D. Pfisterer, SmartSantander: IoT experimentation over a smart city testbed, Comput. Netw. 61 (2014) 217–238, <https://doi.org/10.1016/j.bjp.2013.12.020>. |
| [11] | M. Zappatore, A. Longo, M.A. Bochicchio, Using mobile crowd sensing for noise monitoring în smart cities, 2016 Int. Multidiscip. Conf. Comput. Energy Sci. Split. 2016 (2016), <https://doi.org/10.1109/SpliTech.2016.7555950>. |
| [12] | G. Singh, D. Bansal, S. Sofat, A smartphone based technique to monitor driving behavior using DTW and crowdsensing, Pervasive Mob. Comput. 40 (2017) 56–70, <https://doi.org/10.1016/j.pmcj.2017.06.003>. |
| [13] | G. Cardone, L. Foschini, P. Bellavista, A. Corradi, C. Borcea, M. Talasila, R. Curtmola, Fostering participaction în smart cities: a geo-social crowdsensing platform, IEEE Commun. Mag. 51 (2013) 112–119, [https://doi.org/10.1109/ MCOM.2013.6525603](https://doi.org/10.1109/%20MCOM.2013.6525603). |
| [14] | R.K. Rana, C.T. Chou, S.S. Kanhere, N. Bulusu, W. Hu, Ear-phone an end-to-end participatory urban noise mapping system, Proc. 9th ACM/IEEE Int. Conf. Inf. Process. Sens. Networks - IPSN’ 10 (2010) 105–116, [https://doi.org/10.1145/ 1791212.1791226](https://doi.org/10.1145/%201791212.1791226) . ACM Press, New York, New York, USA |
| [15] | E. Kanjo, NoiseSPY: a real-time mobile phone platform for urban noise monitoring and mapping, Mob. Netw. Appl. 15 (2010) 562–574, [https://doi.org/10.1007/ s11036-009-0217-y](https://doi.org/10.1007/%20s11036-009-0217-y). |
| [16] | [STAPLES, E. EnviSensor: Environmental Data Collection via Participatory Sensor Networks Utilizing Mobile Devices. *Master’s thesis*, 2011.](https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=da64526acae44b036287bc62e51aac8b73fe4a10) |
| [17] | [I. Schweizer, R. Artl, A. Schulz, F. Probst, M. Uhl Auser, NoiseMap -Real-time participatory noise maps, Second Int. Work. Sens. Appl. Mob. Phones (2011) 1–5.](https://hmr.biomedcentral.com/articles/10.1007/s10152-010-0194-3) |
| [18] | N. Maisonneuve, M. Stevens, M.E. Niessen, L. Steels, NoiseTube: Measuring and Mapping Noise Pollution with Mobile Phones, Springer, Berlin, Heidelberg, 2009, pp. 215–228, <https://doi.org/10.1007/978-3-540-88351-7_16>. |
| [19] | [M. Stevens, E.D. ’hondt, Crowdsourcing of pollution data using smartphones, Int. Conf. Entertain. Comput. (2012) 562–571.](https://www.researchgate.net/profile/Matthias-Stevens/publication/228911632_Crowdsourcing_of_Pollution_Data_using_Smartphones/links/00b4953a20b590bc8a000000/Crowdsourcing-of-Pollution-Data-using-Smartphones.pdf) |
| [20] | I.G. Martí, L.E. Rodríguez, M. Benedito, S. Trilles, A. Beltr´ an, L. Díaz, J. Huerta, Mobile Application for Noise Pollution Monitoring through Gamification Techniques, Springer, Berlin, Heidelberg, 2012, pp. 562–571, [https://doi.org/ 10.1007/978-3-642-33542-6\_74](https://doi.org/%2010.1007/978-3-642-33542-6_74). |
| [21] | S. Grubeˇsa, A. Petoˇsi´c, M. Suhanek, I. Đurek, Mobile crowdsensing accuracy for noise mapping în smart cities, Automatika 59 (2018) 286–293, [https://doi.org/ 10.1080/00051144.2018.1534927](https://doi.org/%2010.1080/00051144.2018.1534927). |
| [22] | E. Murphy, E.A. King, Testing the accuracy of smartphones and sound level meter applications for measuring environmental noise, Appl. Acoust. 106 (2016) 16–22, <https://doi.org/10.1016/j.apacoust.2015.12.012>. |
| [23] | P. Aumond, C. Lavandier, C. Ribeiro, E.G. Boix, K. Kambona, E. D’Hondt, P. Delaitre, A study of the accuracy of mobile technology for measuring urban noise pollution în large scale participatory sensing campaigns, Appl. Acoust. 117 (2017) 219–226, <https://doi.org/10.1016/j.apacoust.2016.07.011> |
| [24] | J. Zuo, H. Xia, S. Liu, Y. Qiao, Mapping urban environmental noise using smartphones, Sensors 16 (2016) 1692, <https://doi.org/10.3390/s16101692>. |
| [25] | E. D’Hondt, M. Stevens, A. Jacobs, Participatory noise mapping works! An evaluation of participatory sensing as an alternative to standard techniques for environmental monitoring, Pervasive Mob. Comput. 9 (2013) 681–694, <https://doi.org/10.1016/j.pmcj.2012.09.002>. |
| [26] | [C. Beyers, Calibration methodologies and the accuracy of acoustic data, în: C. Beyer (Ed.), INTER-NOISE NOISE-CON Congr. Conf. Proc., 2014, pp. 3104–3111. Institute of Noise Control Engineering, Melbourne](https://www.acoustics.asn.au/conference_proceedings/INTERNOISE2014/papers/p449.pdf). |
| [27] | J.M. Barrigon ´ Morillas, D. Montes Gonzalez, ´ G. Rey Gozalo, A review of the measurement procedure of the ISO 1996 standard. Relationship with the European Noise Directive, Sci. Total Environ. 565 (2016) 595–606, [https://doi.org/10.1016/ j.scitotenv.2016.04.207](https://doi.org/10.1016/%20j.scitotenv.2016.04.207). |
| [28] | S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, M. Srivastava, Using mobile phones to determine transportation modes, ACM Trans. Sens. Netw. 6 (2010) 1–27, <https://doi.org/10.1145/1689239.1689243>. |
| [29] | [X. Hao, L. Xu, N. Lane, X. Liu, T. Moscibroda, Density-aware compressive CrowdSensing, IPSN2017 (2017) 29–40](https://dl.acm.org/doi/10.1145/3055031.3055081). |
| [30] | S. Lin, J. Zhang, L. Ying, Crowdsensing for Spectrum discovery: a waze-inspired design via smartphone sensing, IEEE/ACM Trans. Netw. 28 (2020) 750–763, <https://doi.org/10.1109/TNET.2020.2976927>. |
| [31] | [Noise Nuisance and the Law, 2015](https://scholar.google.com/scholar?q=Noise%20Nuisance%20and%20the%20Law). |
| [32] | [DIRECTIVE 2002/49/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 25 June 2002](https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2002:189:0012:0025:en:PDF). |
| [33] | E. Murphy, E.A. King, Strategic environmental noise mapping: methodological issues concerning the implementation of the EU Environmental Noise Directive and their policy implications, Environ. Int. 36 (2010) 290–298, [https://doi.org/ 10.1016/J.ENVINT.2009.11.006](https://doi.org/%2010.1016/J.ENVINT.2009.11.006). |
| [34] | [Community and regional noise mapping în the United States](https://www.researchgate.net/publication/279555007_Community_and_regional_noise_mapping_in_the_United_States). |
| [35] | [Eurostat, 2016, Urban Europe — Statistics on cities, towns and suburbs — Eurostat, Cap. 6, Map 6.5](https://ec.europa.eu/eurostat/documents/3217494/7596823/KS-01-16-691-EN-N.pdf/0abf140c-ccc7-4a7f-b236-682effcde10f) |
| [36] | MFCC Coefficients <https://ismir2000.ismir.net/papers/logan_abs.pdf> |
| [37] | Keras model <https://keras.io/api/> |
| [38] | tf.nn model <https://www.tensorflow.org/api_docs/python/tf/nn> |
| [39] | TensorFlow Library [https://www.tensorflow.org/lite/inference\_with\_](https://www.tensorflow.org/lite/inference_with_metadata/task_library/audio_classifier" \o "https://www.tensorflow.org/lite/inference_with_metadata/task_library/audio_classifier)  [metadata/task\_library/audio\_classifier](https://www.tensorflow.org/lite/inference_with_metadata/task_library/audio_classifier" \o "https://www.tensorflow.org/lite/inference_with_metadata/task_library/audio_classifier) |
| [40] | Android Adapter [https://developer.android.com/reference/android/widget/](https://developer.android.com/reference/android/widget/%20%20Adapter" \o "https://developer.android.com/reference/android/widget/  Adapter  )  [Adapter](https://developer.android.com/reference/android/widget/%20%20Adapter" \o "https://developer.android.com/reference/android/widget/  Adapter  ) |
| [41] | [GooglePlay API](https://developers.google.com/android/reference/com/google/android/gms/location/LocationServices) |
| [42] | LocationListener <https://developer.android.com/reference/android/location/LocationListener> |
| [43] | GoogleMaps API <https://developers.google.com/maps> |
| [44] | Android SDK <https://en.wikipedia.org/wiki/Android_SDK> |
| [45] | Firebase SDK <https://firebase.google.com/docs/auth> |
| [46] | NoSQL https://www.mongodb.com/nosql-explained |
| [47] | fisierul build.gradle <https://developer.android.com/build> |
| [48] | BussinessLogic <https://en.wikipedia.org/wiki/Business_logic> |
| [49] | Activități <https://developer.android.com/reference/android/app/Activity> |
| [50] | Fragmente <https://developer.android.com/guide/fragments> |
| [51] | Integrarea Firebase SDK <https://firebase.google.com/docs/android/setup#add-sdks> |
| [52] | GoogleMaps SDK <https://developers.google.com/maps> |
| [53] | MPChartsAndroid <https://github.com/PhilJay/MPAndroidChart> |
| [54] | Android Toast <https://developer.android.com/guide/topics/ui/notifiers/toasts> |
| [55] | Android Intent <https://developer.android.com/reference/android/content/Intent> |
| [56] | BottomNavigationView Android [https://developer.android.com/reference/](https://developer.android.com/reference/com/google/android/material/bottomnavigation/BottomNavigationView" \o "https://developer.android.com/reference/com/google/android/material/bottomnavigation/BottomNavigationView)  [com/google/android/material/bottomnavigation/BottomNavigationView](https://developer.android.com/reference/com/google/android/material/bottomnavigation/BottomNavigationView" \o "https://developer.android.com/reference/com/google/android/material/bottomnavigation/BottomNavigationView) |
| [57] | Andriod Menu [https://developer.android.com/develop/ui/views/components/](https://developer.android.com/develop/ui/views/components/menus" \o "https://developer.android.com/develop/ui/views/components/menus)  [menus](https://developer.android.com/develop/ui/views/components/menus" \o "https://developer.android.com/develop/ui/views/components/menus) |
| [58] | Fragment manager [https://developer.android.com/guide/fragments/](https://developer.android.com/guide/fragments/fragmentmanager" \o "https://developer.android.com/guide/fragments/fragmentmanager )  [fragmentmanager](https://developer.android.com/guide/fragments/fragmentmanager" \o "https://developer.android.com/guide/fragments/fragmentmanager ) |
| [59] | MediaRecorder <https://developer.android.com/reference/android/media/MediaRecorder> |
| [60] | Root Mean Square <https://en.wikipedia.org/wiki/Root_mean_square> |
| [61] | TimerTask Android [https://developer.android.com/reference/java](https://developer.android.com/reference/java/util/TimerTask" \o "https://developer.android.com/reference/java/util/TimerTask )  [/util/TimerTask](https://developer.android.com/reference/java/util/TimerTask" \o "https://developer.android.com/reference/java/util/TimerTask ) |
| [62] | Kotlin Data Classes <https://kotlinlang.org/docs/data-classes.html> |
| [63] | Location Geocoder [https://developer.android.com/reference/android/location/ Geocoder](https://developer.android.com/reference/android/location/Geocoder) |
| [64] | Firebase Realtime Database [https://firebase.google.com/docs/database/ android/read-and-write](https://firebase.google.com/docs/database/android/read-and-write) |
| [65] | Firebase Admin SDK <https://firebase.google.com/docs/admin/setup> |
| [66] | Fișierul .apk cu aplicația <https://drive.google.com/file/d/1ePBEXQf-tguFUTvrZ30dBiya0IfMOf7S/view?usp=sharing> |