

Universitatea “Ștefan cel Mare”

Facultatea de Inginerie Electrică și Știința Calculatoarelor

PROIECT DE DIPLOMĂ

Coordonator științific,

conf. dr. ing. Schipor Ovidiu-Andrei

Absolvent,

Bolohan Alexandru-Cosmin

Suceava, 2018

Sistem de monitorizare al persoanelor vulnerabile

Cuprins

Motivația alegerii temei.....	3
Capitolul I. Tehnologii informaționale folosite	4
I.1 Platforma Software Android.....	4
I.1.1 Structura platformei	6
I.1.2 Java.....	7
I.1.3 Standardul Java în Android. Comparație între Android Dalvik și Android Runtime	10
I.2 Instrumentele Android.....	12
I.2.1 Pachetul de aplicații Android	12
I.2.2 Software Development Kit.....	13
I.3 Android Studion	13
I.3.1 Descrierea unui proiect în Android Studion	14
I.3.2 Descrierea aplicației Android	15
I.3.2.1 Activitatea	15
I.3.2.2 Layout-urile	17
I.3.2.3 Android Manifest	19
I.3.2.4 Procese și fire de execuție în Android.....	19
I.3.3 Descrierea senzorilor.....	21
I.3.3.1 Prezentare generală.....	21
I.3.3.2 Detectarea poziției	22
I.4 Firebase în Android Studio.....	23
I.4.1 Scurt istoric.....	23
I.4.2 Baze de date în timp real	24
Capitolul II Modul de implementare și descriere al aplicației.....	25
II.1 Înregistrarea	25
II.2 Logarea.....	27
II.3 Google Maps. Gestionarea datelor	28
II.4 Profilul utilizatorului și partea de notițe	30
Capitolul III Manual de utilizare	31
Concluzii.....	37
Bibliografie.....	38

MOTIVAȚIA ALEGERII TEMEI

Trăind într-o lume tot mai aglomerată care depinde din ce în ce mai mult de tehnologie, este evident că aproape toate lucrurile pot fi simplificate și controlate cu ajutorul acesteia. Ținând cont de asta, propun în această lucrare o modalitate prin care o persoană poate monitoriza în timp real o persoană vulnerabilă, prin vizualizarea locației în orice moment.

Prin „persoană vulnerabilă” fac referința la un copil, un bătrân, o persoană cu handicap sau dizabilități. O persoană care nu are capacitatea de a se descurca în anumite situații.

Am ales să realizez acest proiect din dorința de a face o lume cât mai sigură. Datorită faptului că unele persoane sunt foarte ocupate, și de cele mai multe ori nu știu unde s-ar putea afla copilul, sau o persoană cu dizabilități, m-am gândit să implementez o aplicație care are drept scop monitorizarea în timp real. Am încercat realizarea acestui lucru prin folosirea dispozitivelor android.

Drept dispozitiv de utilizare m-am decis să aleg cele care rulează pe android deoarece ele dețin cea mai mare cotă de pe piață. Sunt cele mai accesibile și cele mai utilizate.

Aplicația gândită de mine este menită să fie folosită de supraveghetor. Acesta va instala aplicația pe telefonul său după care pe al persoanei vulnerabile. După instalare, acesta va trebui să se înregistreze și să creeze un cont pentru persoana pe care o are în grijă. Deci, mai exact supraveghetorul este cel care administrează toate comenzile, iar persoana vulnerabilă este doar purtătorul aplicației.

Pentru supraveghetor aplicația va avea mai multe funcții. Voi prezenta în câteva rânduri funcționalitățile oferite supraveghetor. Acesta va avea posibilitatea să vadă în timp real unde se afla persoana pe care o are în grijă. Pe lângă asta aceasta poate seta locurile acceptate, locurile interzise caracterizate printr-un perimetru care nu ar trebui depășit. Dacă un lucru este încălcat acesta va fi notificat și va putea să-i trimită un mesaj de avertizare. Persoana vulnerabilă va putea folosi aplicația doar pentru: salvarea unor notițe și comunicarea cu supraveghetorul său.

Pentru a afla locația mă voi folosi de GPS și de API de la Google iar toate datele vor fi salvate pe firebase.

CAPITOLUL I. TEHNOLOGII INFORMAȚIONALE FOLOSITE

I.1 Platforma Software Android

În luna octombrie 2003, în orașul californian Palo Alto, a fost dezvoltat Android Inc de către Rich Miner, Andi Rubin, Nick Sears și Chris White cu scopul de a dezvolta dispozitivele mobile. Sistemul de operare creat de aceștia a avut un real succes, el fiind folosit de peste 40% din populația Pământului.

Telefoanele mobile au avut o creștere/dezvoltare uriașă în ultima perioadă, rezultat ce a condus la apariția dispozitivelor/telefoanelor interigente. Acestea au pătruns în viața noastră ajungând să fie folosite în mai multe domenii. Prin realizarea acestui dispozitiv modern, numit smartphone, dezvoltatorii au reușit să înglobeze nenumărate servicii și funcții precum: transfer de date, conectivitate, diferite aplicații, funcții inteligente, servicii de localizare, diferiți senzori prin care sunt monitorizate acțiunile unui utilizator, etc. Prin urmare, acest mic dispozitiv este mereu la îndemână și are capacitatea de a ne ușura munca prin funcțiile și serviciile înglobate.

În anul 2005 Google a cumpărat Android Inc, și a dat viață sistemului de operare. Spre sfârșitul anului 2007, a pus bazele alianței Open Handset Alliance(AHO) și împreună cu companiile asociate au pus bazele acestui nou sistem de operare. Prin urmare aceștia au realizat un sistem Open Source care a fost disponibil din octombrie 2008.

Primul dispozitiv mobil pe care a fost instalată prima versiune de Android 1.5 denumit Cupcake a fost HTC Dream. În septembrie 2009 apare următoarea versiune 1.6 Donut, și este pentru prima dată când un sistem de operare suportă diferite dimensiuni ale display-ului.

Trepat, prin fiecare versiune, sistemul a fost îmbunătățit.

- Versiunea 2.1 Eclair a fost prezentată de cei de la Google cu ajutorul primului dispozitiv, Nexus One, sub marcă proprie construit de cei de la HTC
- Versiunea 2.2 Froyo aduce îmbunătățiri legate de viteza de lucru a sistemului de operare, care crește considerabil de mult, securitatea este îmbunătățită și o funcție nouă de tethering: dispozitivul poate partaja internet prin Wi-fi sau cablu USB
- Versiunea 2.3 Gingerbread, lansată în decembrie 2010 duce la posibilitatea integrării unui cip NearField Communication (NFC) prin care dispozitivele mobile sunt

pregătite pentru plăți direct de pe mobil. Pe lângă asta au fost adăugate și servicii destinate telefoniei prin internet și videotelefoniei.

- Versiunea 3.0 Honeycomb aduce un salt important în februarie 2011 deoarece a fost optimizată și pentru tablete. Cei de la Google au pus accent pe experiența utilizatorului cu dispozitivul, acest fapt ducând la integrarea unor noi servicii precum: Calendar, Gmail și Search.
- Versiunea 4.0 IceCreamSandwich vine cu un design nou și nenumărate funcții: deblocarea prin recunoașterea feței, acces la aplicații din modul lock screen, îmbunătățirea camerei foto, apariția editorului de imagine, salvarea fotografiilor după locație, etc.
- Versiunea 4.1 Jelly Bean a fost lansată de cei de la Google în 2012, rulând pe o tabletă Nexus 7. Se aduc îmbunătățiri camerei, un control mai bun al notificărilor și noi îmbunătățiri pentru lock screen.
- Versiunea 4.4 KitKat lansată în septembrie 2013 aduce îmbunătățiri legate de managementul hardware și apariția senzorilor precum cel pentru mișcare.
- Versiunea 5.0 Lollipop apare în 2014 și aduce schimbări majore interfeței, un nou design, suport pentru procesoarele pe 64 bit, creșterea duratei de viață a bateriei, suport pentru diferite tipuri de document, etc
- Versiunea 6.0 Marshmallow a fost lansată în 2015. Această versiune va oferi back-up automat pentru fișierele stocate direct în contul Google Drive, un nou mod de economisire baterie, posibilitatea deblocării folosind amprenta, controlul memoriei și permisiuni pentru diferite aplicații
- Versiunea 7.0 Nougat lansată în 2016 vine cu o interfață prietenoasă, fiind ușor de folosit, mai fluidă, făcând ca operarea să fie mult mai ușoară. Se pot folosi simultan două aplicații pe ecranul telefonului și comuta foarte ușor aplicațiile între ele. Se introduce modul Blue Light Filter, durata de viață a bateriei este îmbunătățită prin micșorarea rezoluției și limitarea performanței, Samsung Pass, etc.
- Versiunea 8.0 Oreo a fost lansată în august 2017. Această nouă versiune vine cu următoarele îmbunătățiri: mai multe setări pentru Wi-Fi, o durată de viață mai mare pentru baterie, căutare îmbunătățită, dual messenger, Samsung DeX, etc.

I.1.1 Structura platformei

Android este un SO(Sistem de operare) care se bazează pe nucleul oferit de Linux fiind un sistem open sources. Acest sistem permite rularea/ procesarea în background, suportând grafică 2.D și 3.D. O aplicație realizată în Android este formată dintr-o componentă vizibilă și una care este non vizuală.

Platforma Android se împarte în patru nivele. Următoarea diagramă prezintă componentele principale ale platformei Android.

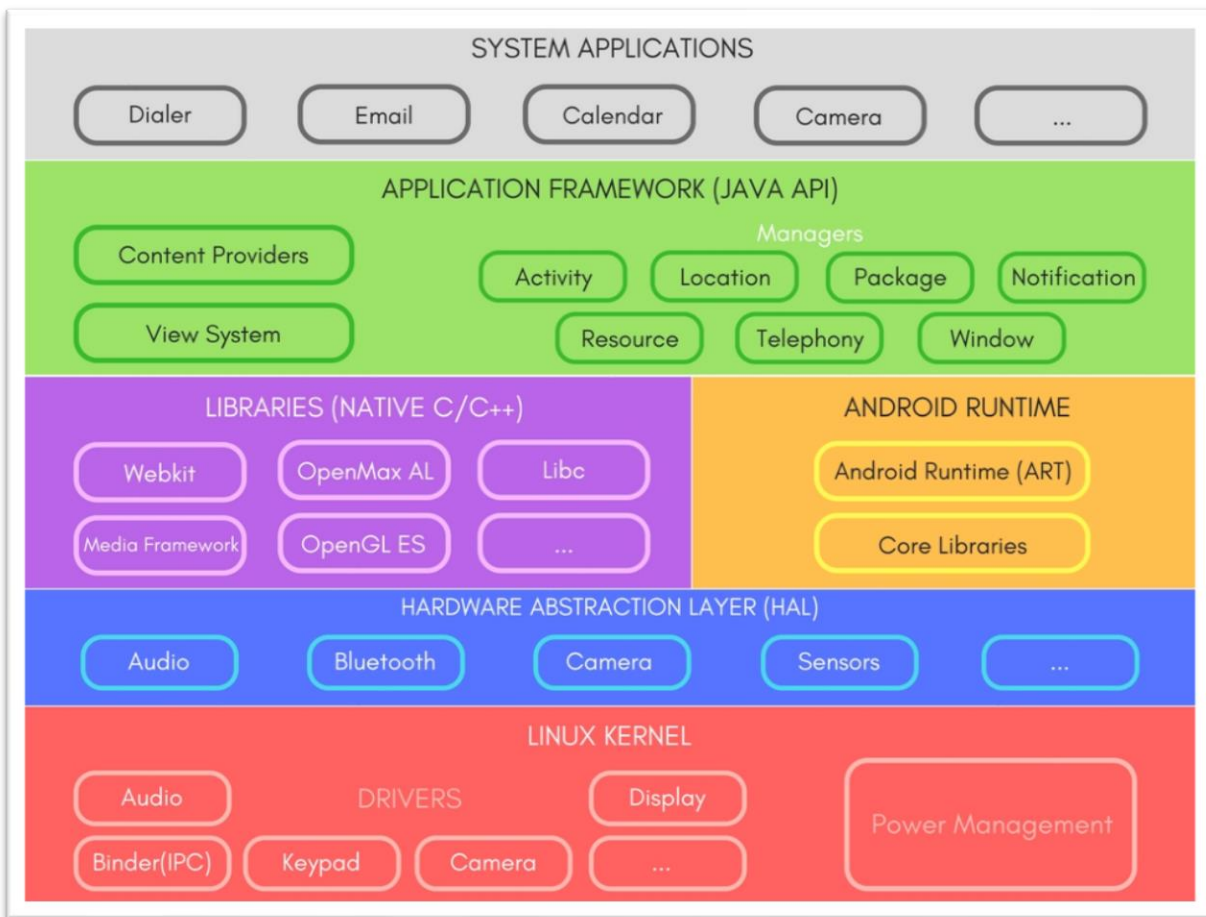


Fig. 1.1.1 Structura platformei

În partea de jos a straturilor este **Kernelul Linux**, Linux 3.6 cu aproximativ 115 patch-uri. Aceasta oferă un nivel de abstractizare între hardware-ul dispozitivului și conține toate driverele hardware esențiale, cum ar fi camera foto, tastatura, afișajul etc. De asemenea, kernel-ul se ocupă de toate lucrurile, cum ar fi crearea de rețele și gestionarea memoriei la un nivel scăzut.

Stratul de abstractizare hardware (HAL) furnizează interfețe standard care expun capabilitățile hardware la un nivel superior în cadrul framework-ului Java API.

Rutinele android sunt folosite pentru a executa mai multe virtual machine pe dispozitive cu memorie scăzută executând fișiere DEX, reprezentând un format bitecode special pentru sistemele android optimizat pentru un minim de memorie. Include câteva caracteristici din Java 8.

Bibliotecile natice C/C++. O multitudine de componente și servicii de bază ale sistemului Android, sunt construite din cod nativ care necesită biblioteci native scrise în C sau C++.

Application framework. Toate caracteristicile sistemului de operare sunt furnizate prin API scrise în limbajul JAVA.. API-urile sunt acele blocuri pe care un programator le folosește pentru a realiza o aplicație.

Aplicațiile de sistem reprezintă un set de aplicații de bază. Acestea pot fi: Camera, Muzică, Telefon, mesajele SMS, Calendar, un browser și multe altele.

Cei de la Google oferă pentru programatori o platformă numită Google Play, unde își pot pune aplicațiile pentru un utilizator. Aceștia oferă servicii și pentru programatorii, oferind diferite servicii, cum ar fi Google Maps, etc.

I.1.2 Java

Java este un limbaj de programare orientat-obiect, puternic tipizat, conceput inițial de James Gosling la compania Sun Microsystems. Acest limbaj a apărut la începutul anilor 90 și lansat abia în 1995. În mare parte aplicațiile distributive sunt scrise în Java și datorită evoluției tehnologice permite folosirea sa și pe dispozitive precum: agendă electronică, telefon, etc. Acest limbaj are sintaxa programului și este asemănătoare limbajurilor C și C++. Aduce îmbunătățiri acestuia printr-un model simplificat orientare pe obiecte. Față de limbajul C++, acesta suportă mai puține avantaje de nivel jos.

Limbajul Java poate fi descris prin următoarele cuvinte: simplitate, siguranță, orientat pe obiecte, lucrul cu clase, standardizare și concurență.

Chiar de la apariția limbajului, Java a stabilit câteva principii de bază. Fiind un limbaj orientat pe obiecte, acesta trebuie să fie simplu, ușor de învățat și familiar. Trebuie să fie interpretabil, dinamic, neutru din punct de vedere al arhitecturii și să folosească fire de execuție. Trebuie să fie robust și sigur. Dar nu în cele din urmă trebuie să fie portabil, să poată fi rulat de mai multe dispozitive și să ofere o performanță ridicată. Java oferă o platformă de dezvoltare curată și eficientă bazată pe obiecte.

Din punct de vedere al performanței, inițial, Java era considerat un limbaj de programare ce era mai lent și necesita mai multă memorie pentru aplicațiile sale comparativ cu limbajul C. Odată cu apariția versiunii Java 5.0, se oferă o interfață simplă și a fost descris un nou model al memoriei. Modelul specifică comportamentele pentru un program multithreaded. Prin asta se definește semantica programelor Java cu mai multe fire de execuție și implementările legate mașina virtuală Java și compilator.

Programatorii C și C++ sunt deja obișnuiți cu problemele legate de gestiunea memoriei: alocarea memoriei, eliberarea memoriei și urmărirea memoriei pentru a fi eliberată. Exploatarea managementului memoriei s-a dovedit a fi o sursă bogată de bug-uri, accidente, pierderi de memorie și performanță scăzută.

Pentru aceste deficiențe, Java elimină complet sarcina de gestionare a memoriei de la programator. Nu există tipuri explicite de date definite pentru pointeri, arhitecturi pointeri, alocare și eliberare de memorie. Automat colectarea gunoierului este o parte inteligentă a sistemului Java și a sistemului său de execuție. Odată ce ai creat un obiect, sistemul va urmări starea obiectului și va recupera automat memoria atunci când obiectul este scos din uz, eliberând memoria pentru utilizări ulterioare.

Modelul de gestionare al memoriei Java se bazează pe obiecte și referințe la obiect. Java nu are pointeri. În schimb are referințe la spațiul de memorie alocat, care în practică înseamnă toate referințele la un obiect. Managementul de memorie ține evidența referințelor la obiect. Atunci când un obiect nu are referințe, obiectul este un candidat pentru eliberare de memorie.

Prin urmare, modelul de gestionare al memoriei este extrem de simplu. Prin urmare gestionarea memoriei se face de către un colector automat de resurse ce nu mai pot fi folosit. Practic rolul programatorului s-a redus la a crea obiectele necesare iar colectorul recuperează memoria ce devine disponibilă odată cu încheierea ciclului de viață al obiectelor. Dacă

obiectul ce a fost preluat de colector este apelat de o metodă va apărea o eroare de tipul “null pointer exception”. Cel mai folosit tip de colector este CMS (Concurrent Mark Sweep collector).

Ca și alte limbaje, Java are o suită de clase speciale care sunt des utilizate. Dintre acestea cele mai importante sunt: applet, JavaServer Pages(JSF) si servlet.

Applet este un program java folosit în special în cadrul paginilor Web, făcând parte din pachetul javax.swing. JSF sunt componente ale Java EE de tip server-side. Acest lucru permite cod Java în paginile HTML prin folosirea delimitatorilor speciali <% JAVA CODE %>. Iar servlet oferă dezvoltatorilor Web un mecanism consistent de extindere a funcționalității unui server Web, el fiind o componentă Java EE.

Pe lângă nenumăratele avantaje, de-a lungul timpului ca și oricărui alt limbaj, și limbajului Java i-au fost gasite și defecte.

- cu toate că codul scris în Java este portabil și neutru din punct de vedere arhitectural, metoda nu este atât de eficientă. Codul java este compilat în bytecod iar un translator JVM execută programul. Asta duce la încetinirea procesorului, deci o viteză de execuție redusă.
- Memorie mai multă.
- gestionarea numerelor de tip unsigned
- vulnerabilități legate de sistemul de securitate

I.1.3 Standardul Java în Android. Comparație între Android Dalvik și Android Runtime

Limbajul Java este foarte cunoscut în rândul dezvoltatorilor/programatorilor, el fiind folosit foarte mult în industria dispozitivelor mobile, chiar și înainte să apară android. Cei de la Android s-au bazat pe acest limbaj deoarece era deja foarte cunoscut și cei care programau nu trebuiau să mai studieze/învețe un limbaj nou, așa cum s-a întâmplat la Apple.

Android-ul se deosebește de ceilalți prin faptul ca aplicațiile instalate rulează într-o virtual machine. Prima mașină virtuală a fost Dalvik.

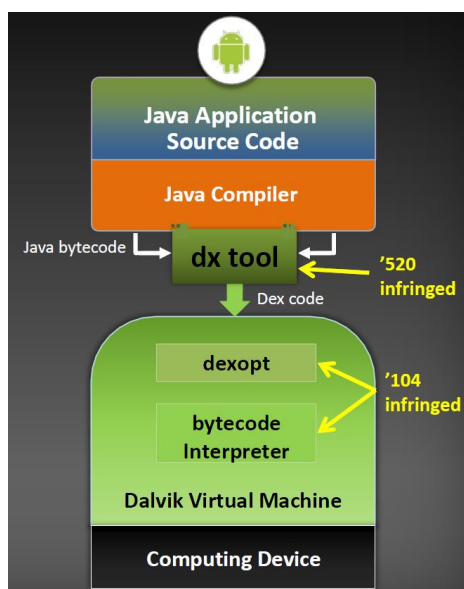


Fig. 1.1.3.1 Schema descriere masina virtuală Dalvik

Aplicațiile Android rulează pe mașina virtuală Dalvik, dezvoltată de cei de la Google. Programele sunt compilate în bytecode pentru mașina virtuală Java, care apoi sunt traduse în Dalvik bytecode și stocate în fișiere .dex și .odex.

Dalvik este un software open source care oferă câteva avantaje printre care: rularea rapidă pe unele procesoare mai slabe și nu avem nevoie de resurse multe sau un consum ridicat de energie. Sistemul se bazează pe nucleul Linux care lucrează cu fire de execuție și care dispune de un management bun al resurselor hardware.

Dar în timp, cei de la Google au dezvoltat altă mașină virtuală, denumită Android Runtime (ART), o alternativă pentru Dalvik. ART efectuează traducerea bytecode a aplicației în

instrucțiuni native care sunt ulterior executate de mediul de rulare al dispozitivului. Pe lângă asta, ART oferă detalii suficiente atunci când apare o excepție și oferă un raport detaliat în legătură cu evenimentul întâmplat. Odată cu versiunea 5.0 Dalvik este înlocuit complet.

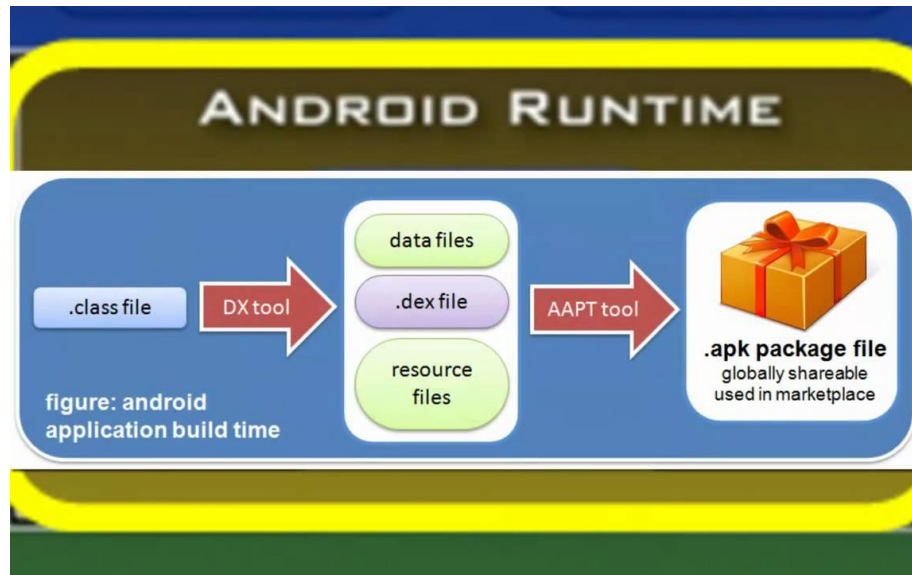


Fig. 1.1.3.2 Android Runtime

Un alt avantaj este acela că ART introduce compilarea A O T (ahead of time), adică compilează codul înainte să ruleze aplicația, acest lucru ducând la o îmbunătățire a performanței. Reduce consumul de resurse, oferă o eficiență ridicată ceea ce duce la o autonomie crescută pentru telefon.

I.2 Instrumentele Android

I.2.1 Pachetul de aplicații Android

APK-ul este formatul fișierelor folosite de sistemul de operare Android pentru distribuirea și instalarea aplicațiilor mobile și middleware. Acest tip de fișiere sunt similare cu pachetele software precum APPX sau MSI ce rulează pe Microsoft sau fișierele DEB bazate pe Debian destinate pentru Ubuntu.

Un fișier APK este o arhivă care conține de obicei următoarele fișiere și directoare:

- META-INF: fișierul Manifest, certificatul aplicației, lista de resurse și SHA-1
- Lib: directorul care conține codul compilat
- Res: directorul care conține resursele folosite
- Classes.dex: clasele compilate în format .dex înțelese de mașina virtuală Dalvik
- Codul programului

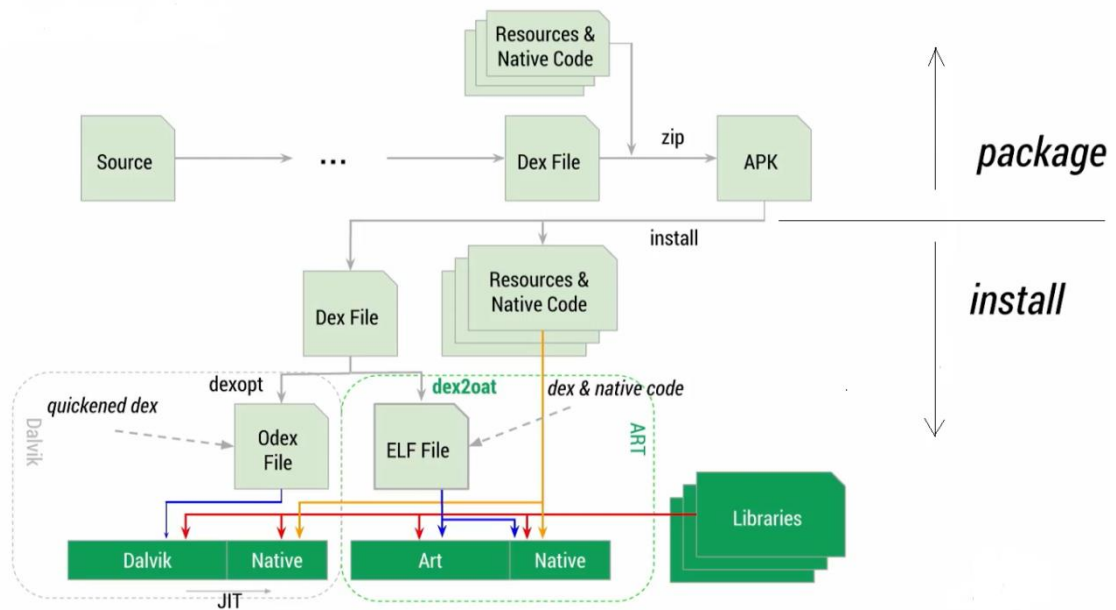


Fig. 1.2.1.1 Descriere pachet aplicații android

Fișierele apk sunt fișiere de tip arhivă, în special pachete cu formatul JAR, cu extensia de nume *.apk. Fișierele APK pot fi instalate pe dispozitivele Android, la fel ca instalarea unui soft pe PC. Un fișier .apk este format din: biblioteci native, resurse folosite în aplicație și un executabil Dalvik.

I.2.2 Software Development Kit

SDK include o colecție cuprinzătoare de instrumente de dezvoltare ce permite realizarea aplicațiilor. Android SDK Java include: un program de depanare, biblioteci, librării, un emulator de telefon bazat pe QEMU, debugger și documentație.

Până la sfârșitul anului 2014 Eclipse a fost mediul oficial de dezvoltare (IDE) folosind Android DevelopmentTools, după care începând cu 2015 Android Studio i-a luat locul, devenind oficial mediul de dezvoltare Android.

I.3 ANDROID STUDIO

Android Studio este oficial mediul de dezvoltare pentru sistemele de operare Android. A fost anunțat la o conferință Google și lansat oficial în 2014. Acest sistem a fost construit special pentru dezvoltarea Android, el fiind disponibil pe Windows, MacOS și Linux. Aceasta poate fi considerat un înlocuitor pentru Eclipse Android Development Tools.

Funcționalități oferite:

- Suport pentru platforma Google Cloud
- Suport integrat pentru Firebase Cloud
- Suport pentru crearea aplicațiilor Android Wear
- Editor de coduri inteligente
- Emulator bogat și rapid
- Șabloane de cod și exemple de aplicații
- Instrumente și cadre de testare

I.3.1 Descrierea unui proiect în Android Studio

Android Studio structurează un proiect în mai multe module care vor fi concepute în final într-un fișier .apk. Proiectul este alcătuit din toate resursele și informațiile ce definesc o aplicație, incluzând și codul sursă, ele fiind structurate sub formă de module.

După cum se poate observa și în Fig. 1.6 principalele module sunt:

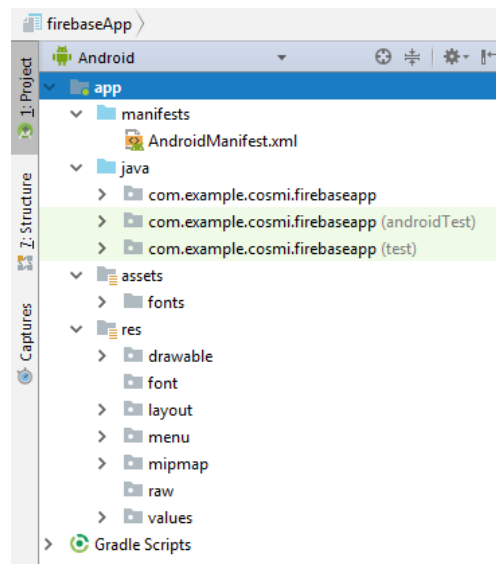


Fig 1.3.1.1 Structura unui program android

- **AndroidManifest.xml** – un document care conține descrierile pentru fiecare componentă din aplicație. Spre exemplu: librării externe, permisiuni, etc.
- **Java** – locul în care se găsește codul sursă
- **Res** – un director important deoarece aici se găsesc alte subdirectoare în care se găsesc resursele disponibile în proiect. Subdirectoarele sunt:
 - a. *drawable* - se găsesc elementele ce țin de grafică
 - b. *layout* – aici sunt definite elementele care definesc interfața aplicației și aranjarea elementelor grafice
 - c. *values* – valorile care sunt folosite în program
- **Assets** – aici se găsesc fișiere de tip audio, img, video, fonturi, baze de date, etc.

Pe lângă modulele principale, programatorul are posibilitatea de a crea noi foldere/directoare, pentru a structura cât mai bine proiectul. Acesta este un avantaj, deoarece proiectul va fi structurat după propriul mod de programare.

Acest aspect este foarte important, fiind necesar atunci când proiectul este foarte mare și trebuie împărțit în module.

I.3.2 Descrierea aplicației Android

O aplicație este compusă din mai multe părți: activități, servicii, broadcast receiver și manager de conținut.

I.3.2.1 Activitatea

Activitatea este o componentă esențială a unei aplicații Android, precum și modul în care este lansată. Practic, este o interfață cu utilizatorul, sau poate fi un formular ori o fereastră. O aplicație android mai complexă este alcătuită din mai multe activități puse într-o anumită ordine. La deschiderea aplicației doar o activitate va fi afișată pe întreg ecranul, ea fiind activitatea principală. Orice activitate este caracterizată prin: date proprii, un ciclu de viață și o stare proprie.

Starea unei activități este dată de stiva de activități. Cu ajutorul ei știu starea activității.

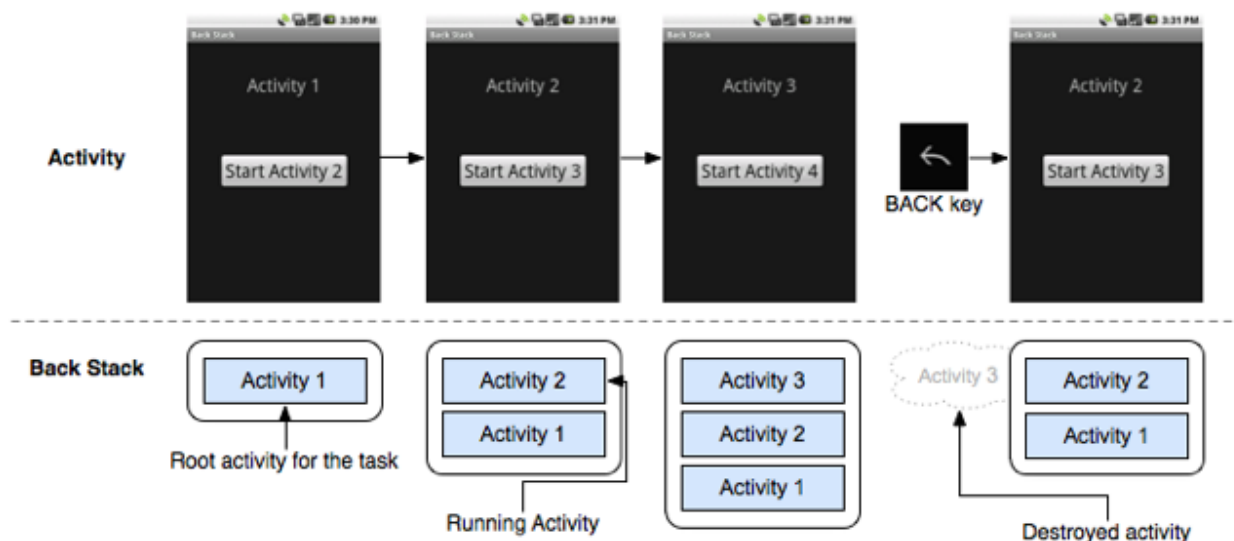


Fig. 1.3.2.1.1 Funcționalitatea activităților

În Fig. 1.3.2.1.1 se poate vedea principiul de funcționare al stivei, FILO(First In Last Out). După cum se poate observa activitățile noi deschise vor fi puse deasupra stivei, iar pentru a reveni la activitatea precedentă ea este eliminată din stivă. Cu ajutorul butonului înapoi se poate reveni la activitatea anterioară.

În Fig. 1.3.2.1.2 este prezentat ciclul de viață al unei activități Android:

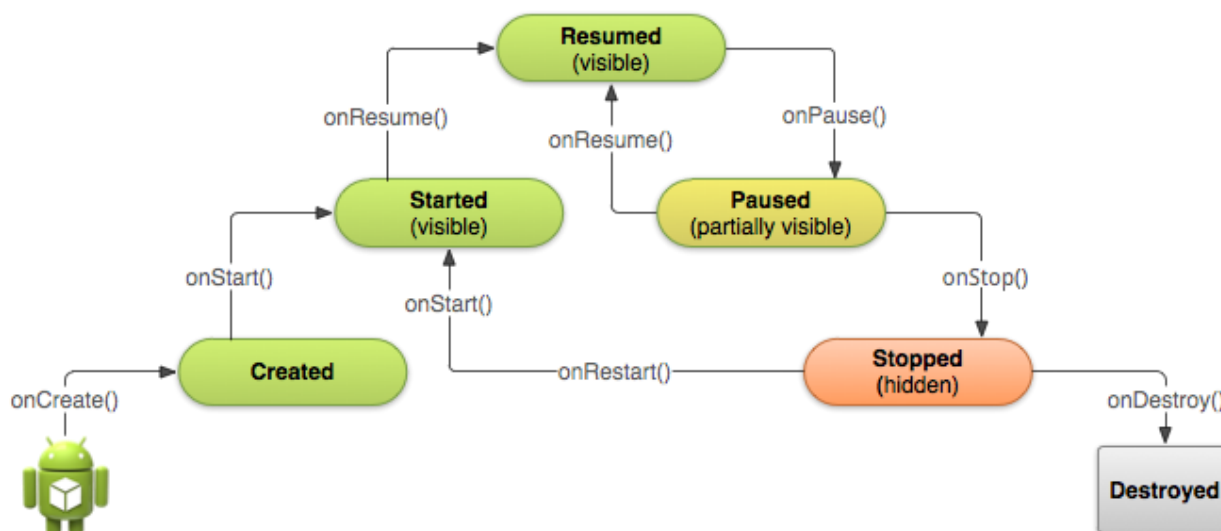


Fig. 1.3.2.1.2 Ciclul de viață al uni activități

O activitate poate reprezenta într-o aplicație o fereastră sau un ecran unde sunt afișate diferite date. Un astfel de activitate poate fi fereastra care afișează contactele sau mesajele.

Fiecare activitate al unei aplicații Android are un ciclu de viață. Atunci când deschizi aplicația un fel de ecran vă va întâmpina. În spatele scenei, acest ecran trebuie creat înainte de a putea oferi informații sau conținut.

Pe măsură ce un utilizator navighează prin aplicație, instanțele activității trec prin diferite stări din ciclul lor de viață. Clasa „Activity” oferă un număr de apeluri care permit activităților să știe cand o stare s-a schimbat: că sistemul a creează, oprește și reia o activitate sau distruge procesul în care se află o activitate.

Fiecare activitate oferă un set de bază de șase apeluri sau funcții:

- onCreate();
- onStart();
- onResume();
- onPause();
- onStop();
- onDestroy();
- onRestart();

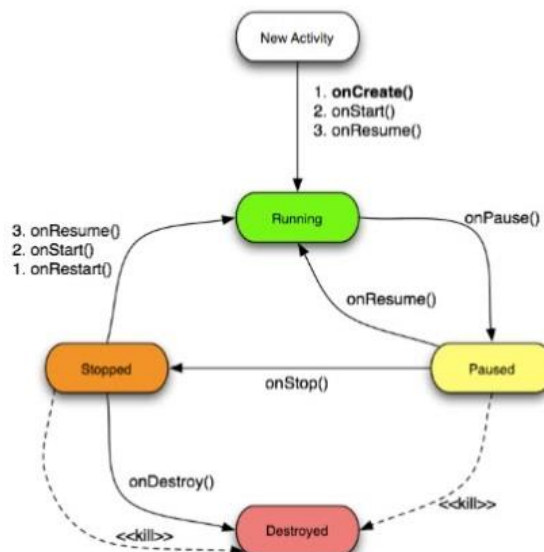


Fig. 1.3.2.1.3
Ciclul de viață

Începerea unei activități se face prin apelarea metodei onCreate(), fiind momentul în care se apelează și funcția setContentView(), care se ocupă cu popularea interfeței și inițializarea datelor.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_home);
}
```

Fig. 1.3.2.1.3 Metoda OnCreate

Odată ce activitatea intră în starea inițială, sistemul invocă apelul către funcția onStart(). Prin acest apel activitatea devine vizibilă și interactivă pentru utilizator. Metoda se încheie foarte repede, și după terminarea apelului, activitatea intră în starea Resume, iar sistemul invocă metoda onResume().

Metoda onStop() este opusă ca efect. Această metodă are ca efect faptul că activitatea nu mai este vizibilă deoarece o altă activitate a fost deschisă sau altă activitate revine pe ecran, sau are loc distrugerea activității care rulează prin apelul metodei onDestroy, care are un efect ireversibil asupra aplicației.

I.3.2.2 Layout-urile

Un layout definește structura unei interfețe de utilizator în aplicație. Toate elementele din layout sunt construite folosind o ierarhie **View** și **ViewGroup**. Obiectul View este creat din clasa View și ocupă o zonă dreptunghiulară pe ecran și este responsabilă pentru manipularea evenimentelor și pentru desen. ViewGroup este o subclasă din clasa View, fiind un container invizibil care definește alte View-uri sau alte ViewGroup-uri, așa cum este prezentat în Fig. 1.3.2.2.1.

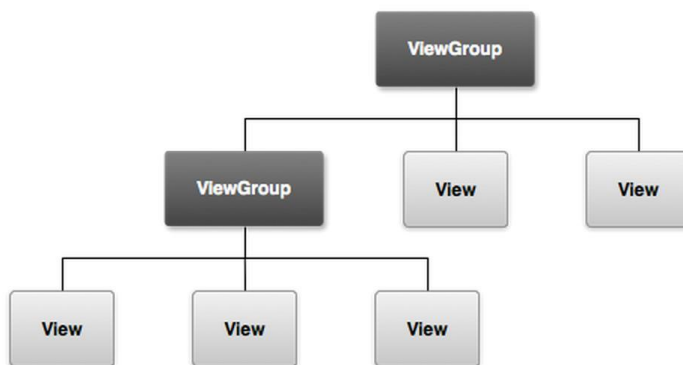


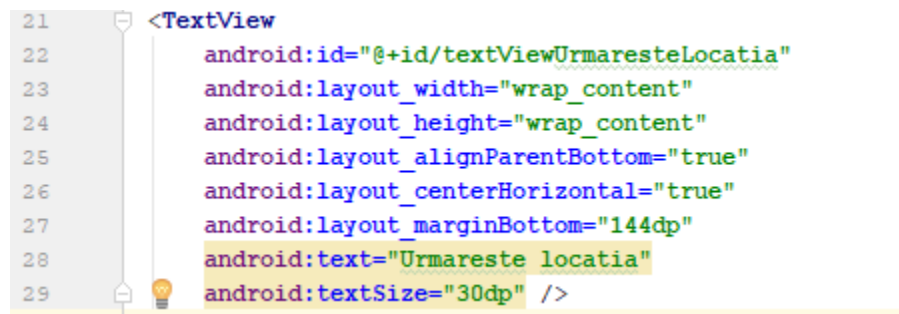
Fig. 1.3.2.2.1 Structura ViewGroup

Fișierele în care se definesc layout-urile sunt în format XML scrise în Java. Declararea interfeței în XML va permite separarea codului care controlează componentele de pe layout și prezentarea aplicației.

Layout-urile pot fi clasificate în:

- Relative Layout
- Linear Layout
- Absolute Layout
- Table Layout
- List View
- Grid View

În cadrul layout-ului se pot defini diferite obiecte (View), descrise în limbaj Java, cu un ID unic și un set de caracteristici precum în Fig. 1.3.2.2.2.



```
21 <TextView
22     android:id="@+id/textViewUrmaresteLocatia"
23     android:layout_width="wrap_content"
24     android:layout_height="wrap_content"
25     android:layout_alignParentBottom="true"
26     android:layout_centerHorizontal="true"
27     android:layout_marginBottom="144dp"
28     android:text="Urmareste locatia"
29     android:textSize="30dp" />
```

Fig. 1.3.2.2.2 Proprietățile unui obiect

Fiecare layout are un set de atribute care definesc proprietățile vizuale ale acelui layout. Acestea sunt:

- **Id:** identifică în mod unic layout-ul
- **Layout_width:** definește lățimea
- **Layout_height:** definește înălțimea
- **Layout_margitTop:** spațiu suplimentar din partea de superiară
- **Layout_marginBotton:** spațiul suplimentar din partea de jos
- **Layout_marginLeft:** spațiu suplimentar din partea stângă
- **layout_marginRight:** spațiu suplimentar din partea dreaptă
- **layout_gravity:** specifică modul în care e poziționat copilul
- **layout_weight:** cât spațiu trebuie alocat vizualizării
- **layout_x:** coordonatele X ale layout-ului
- **layout_y:** coordonatele Y ale layout-ului
- etc

I.3.2.3 Android Manifest

Fiecare proiect Android trebuie să aibă în mod obligatoriu un fișier AndroidManifest.xml. Fișierul manifest descrie informațiile esențiale de care are nevoie aplicația pentru a rula. Printre alte lucruri, în fișier este obligatoriu de declarat: numele pachetului aplicației, componentele aplicației care include toate activitățile și serviciile, lista permisiunilor și caracteristicile hardware și software ale aplicației. Tot aici găsim și versiunea de SDK, de la cea minimă necesară pentru a rula aplicația, până la cel setat de noi pentru care este construită aplicația. În Android Studio, fișierul este generat automat și conține majoritatea elementelor esențiale.

Mai exact fișierul AndroidManifest.xml poate fi considerat ca o legătură între aplicație, programator și utilizator.

Permiisiunile joacă un rol important în cadrul acestui fișier. Aplicațiile Android trebuie să solicite permisiuni pentru a accesa anumite date sau anumite caracteristici ale sistemului. Este o restricție care-ți restricționează accesul asupra funcționalității. Permiisiunile au fost introduse ca o măsură de protecție, pentru datele critice sau pentru aplicațiile care folosesc datele personale în scopuri greșite. În Fig. 1.3.2.3.1 se pot observa câteva permisiuni folosite de mine.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Fig. 1.3.2.3.1 Permiisiunile din AndroidManifest

I.3.2.4 Procese și fire de execuție în Android

Atunci când o aplicație pornește și ea nu are alte aplicații care rulează, sistemul Android pornește un nou proces Linux cu un singur fir de execuție. În mod prestabilit toate componentele aceleiași aplicații rulează în același proces și thread. Dacă o componentă a aplicației este pornită și există deja un proces, ea va folosi acel proces și va utiliza același fir de execuție. Cu toate acestea, putem seta ca componente diferite din aplicație să ruleze în procese separate, și putem crea fire suplimentare pentru fiecare proces.

Un proces poate avea unul sau mai multe fire de execuție.

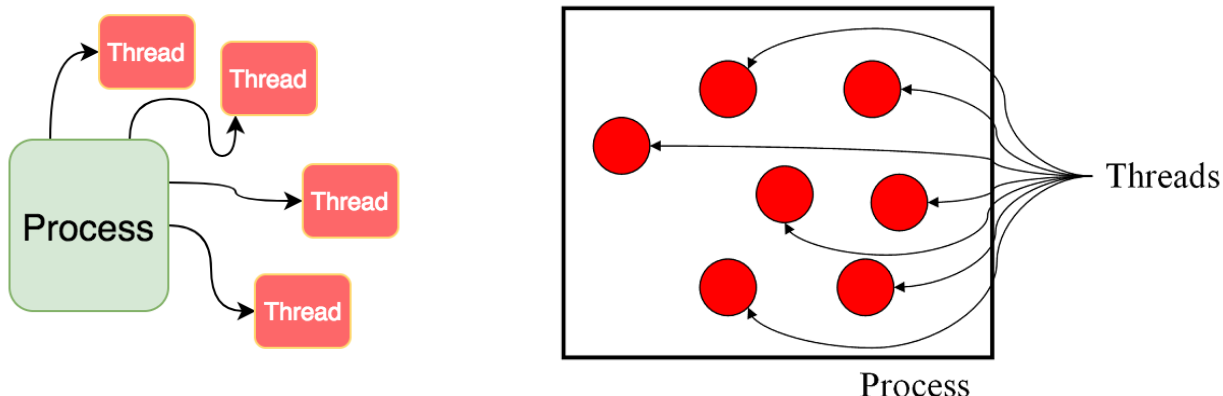


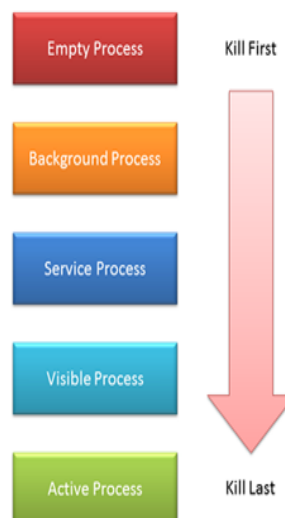
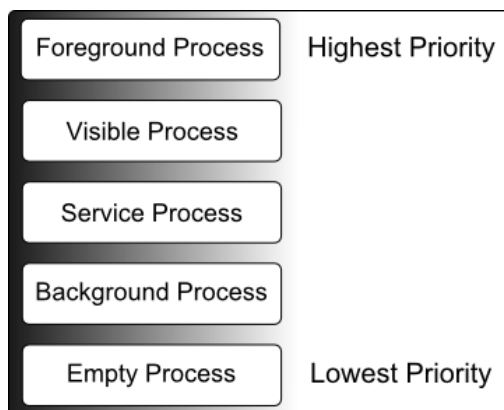
Fig. 1.3.2.4.1 Procese si Threaduri

Prin folosirea proceselor și a firelor de execuție se evidențiază caracteristica de multitasking. Această caracteristică face posibilă execuția în același timp a mai multor aplicații pe același dispozitiv în același timp. Procesele și firele sunt importante și necesare din cauză că aplicațiile ce rulează au nevoie de memorie.

Sistemul de operare Android gestionează aplicațiile care rulează și determină ordinea în care procesele sunt create, oprite sau distruse. Odată creat un proces, el va rămâne în memoria dispozitivului până când sistemul va decide dacă procesul va fi oprit sau distrus și dacă va alocă memoria altui proces.

Se cunosc cinci nivele de importanță ale unui proces:

- Foreground Process
- Visible Process
- Service Process
- Background Process
- Empty Process.



I.3.3 Descrierea senzorilor

I.3.3.1 Prezentare generală

Cele mai multe dispozitive care rulează pe Android au senzori diferiți senzori încorporați care măsoară mișcarea, orientarea și diferite condiții de mediu. Acești senzori sunt capabili să furnizeze date cu o precizie ridicată. Datele furnizate fiind utile pentru a monitoriza mișcarea sau poziția dispozitivului. De exemplu, un joc poate folosi anumiți senzori pentru mișcarea, rotația, înclinatul, tremuratul caracterului, sau camera care folosește gps pentru a salva locația fotografiei. Sau o aplicație meteo ar putea utiliza senzori de temperatură și umiditate. Prin urmare senzorii se împart în trei categorii:

- Senzori de poziție: măsoară poziția fizică a unui obiect. Această categorie include senzori de orientare și magnetometre
- Senzori de mediu: măsoară diferiți parametri din mediu, cum ar fi temperatura și presiunea aerului ambiant, iluminarea și umiditatea. În această categorie intră barometre, fotometre și termometre
- Senzori de mișcare: măsoară forțele de accelerație și forțele de rotație de-a lungul a trei axe. Această categorie include accelerometre, senzori de gravitație, giroscopuri și senzori vectori rotativi.

O altă clasificare este făcută după tipul său: hardware sau software. Senzori hardware pot fi definiți ca și componente fizice încorporate într-un dispozitiv, ei obținând datele prin măsurare directă. Iar cei bazați pe software se comportă ca și cei hardware, derivând datele de la cei bazați pe hardware.

Senzori hardware sunt:

- Type_accelerometer
- Type_gyroscope
- Type_light
- Type_ambient_temperature
- Type_magnetic_field
- Type_proximity
- Type_pressure
- Type_rotate_vector
- Type_temperature
- Type_relative_humidity

Senzori software sunt:

- Type_orientation
- Type_rotation_vector
- Type_linear_acceleration
- Type_gravity

Pentru a avea acces la senzori și la datele furnizate vom folosi framework-ul Android Sensor, din pachetul android.hardware. Acest pachet conține următoarele clase și interfețe: `EventListener`, `Sensor`, `SensorEvent`, `SensorManager`.

I.3.3.2 Detectarea poziției

GPS este un sistem de radionavigație care permite utilizatorilor terestri, marini și aerieni să-și verifice locația exactă, viteza și timpul de 24 de ore pe zi, în orice condiții meteorologice, oriunde în lume. Când dezvoltăm o aplicație bazată pe locație pentru Android, GPS-ul este cel mai precis în mod special în aer liber.

Trebuie să integram modulele de localizare în aplicația dvs. pentru a utiliza GPS. Pentru a obține locația utilizatorului în aplicațiile Android prin apel invers. Avem posibilitatea de a afla locația în două moduri.

LocationManager

Această clasă oferă acces la serviciile de localizare a sistemului. Puteți primi actualizări de locație apelând `requestLocationUpdates()` și să-i transmiteți un `LocationListener`.

LocationListener

Folosit pentru primirea notificărilor de la `DeviceManager`, când locația se schimbă

I.4 Firebase în Android Studio

I.4.1 Scurt istoric

Firebase este o platformă de dezvoltare a aplicațiilor mobile și web. Ea oferă dezvoltatorilor o mulțime de instrumente și servicii pentru a le ușura munca și pentru a dezvolta aplicații de înaltă performanță.

Totul a început în 2011, cu un startup numit Envolv, care oferea dezvoltatorilor un API care a permis integrarea funcționalității de chat online pe site-ul lor. Interesant este faptul că oamenii foloseau Envolv pentru a transmite date care erau mai mult decât mesaje. Dezvoltatorii utilizau Envolv și pentru a sincroniza datele pentru anumite aplicații, cum ar fi un joc, în timp real pentru utilizatorii.

Acest lucru a condus la separarea arhitecturii în timp real și a sistemului de chat. În 2012 Firebase a fost creat ca o companie separată care furniza servicii de Backend cu funcționalitate în timp real.

După ce a fost achiziționată de Google în 2014, Firebase a evoluat rapid oferind o multitudine de servicii, împărțite în două grupuri precum în Fig. 1.4.1.1.

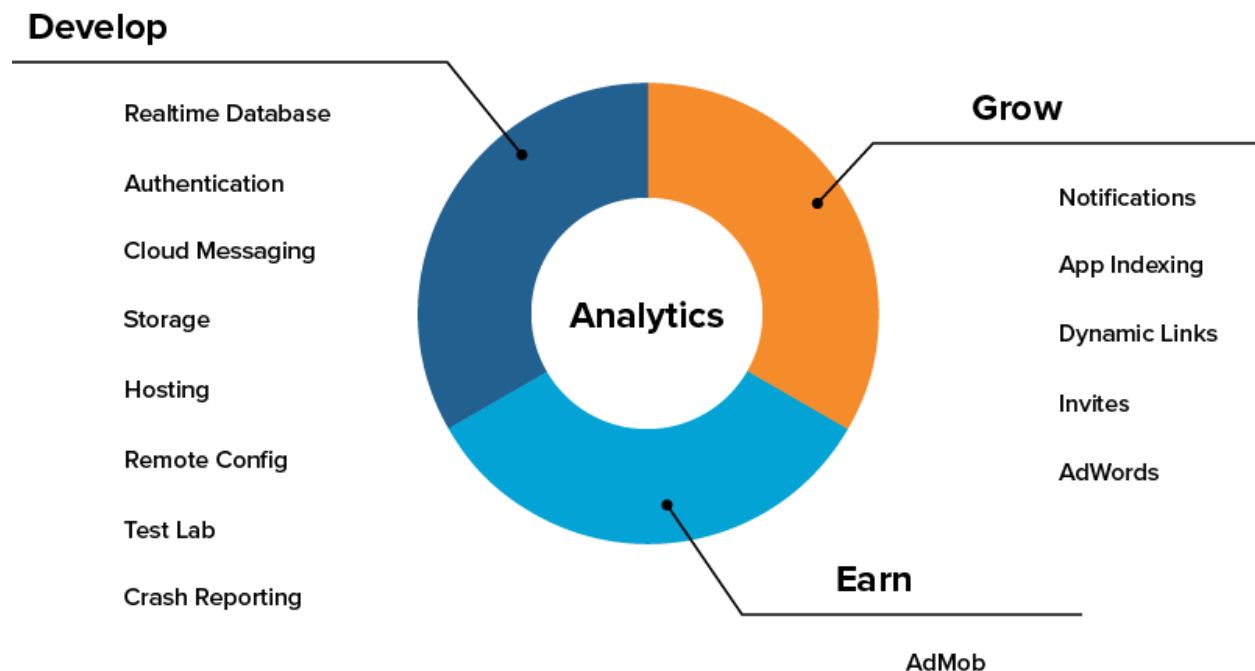


Fig.1.4.1.1 Structura Firebase

I.4.2 Baze de date în timp real

Baza de date oferită de cei de la Firebase Realtime este o bază de date NoSQL găzduită în cloud care permite stocarea și sincronizarea în timp real a datelor. Ea fiind un obiect în format JSON pe care dezvoltatorii îl pot gestiona în timp real. Acest lucru se poate vedea în Fig. 2.9



Fig. 1.4.2.1 Structura datelor salvate în Firebase

Cu un singur API, baza de date Firebase furnizează valorile curente ale datelor, cât și orice actualizare a acestor date folosite în aplicație. Datorită faptului că totul este în timp real accesul utilizatorilor este ușurat și accesul la date se poate face pe orice dispozitiv sau pe un site web.

Un alt beneficiu al bazei de date este că livrează un SDK pentru mobile și web, permițând ca aplicația să fie construită fără a avea nevoie de un server. Atunci când un utilizator se deconectează de la aplicație, SDK-ul folosește cache-ul local pentru a schimba setările. Iar când dispozitivul se conectează, datele locale sunt sincronizate automat.

Pe lângă SDK, avem nevoie să adăugăm și dependențe.

CAPITOLUL II MODUL DE IMPLEMENTARE ȘI DESCRIERE AL APLICAȚIEI

În acest capitol voi prezenta modul de implementare al aplicației. Voi începe cu partea de înregistrare, după care logarea, recuperarea parolei, partea de profil, partea de notite, și partea cea mai importată monitorizarea copilului și salvarea locațiilor preferate.

II.1 Înregistrarea

Pentru a putea folosi aplicația utilizatorul are nevoie de un cont. Acest lucru se face foarte ușor. Atunci când deschid aplicația pe activitatea de logare se află un buton pentru funcția de înregistrare. La apăsarea butonului, activitatea în care se face înregistrarea devine vizibilă.

Formularul este simplu și ușor de folosit. Pentru a ne înregistra avem nevoie de un email și o parolă. Datele folosite trebuie să fie valide, de aceea înainte de a le salva voi face niste verificări.

Pentru e-mail, stringul introdus trebuie să aibă un format specific. Pentru asta folosesc un pattern cu un format standard.

```
String email=eTextEmail.getText().toString();
String emailPattern = "[a-zA-Z0-9._-]+@[a-z]+\\.+[a-z]+";
if(TextUtils.isEmpty(email)){
    eTextEmail.setError("Introdu un email!");
    return;
}
if (!email.matches(emailPattern)) {
    eTextEmail.setError("Format incorect! Ex: test@yahoo.com ");
    return;
}
```

Fig. 2.1.1 Validarea emailului

Pentru parolă verificările sunt mai simple. Se verifică dacă câmpul este gol, sau parola este mai mică de cinci caractere. Pe lângă aceste verificări am pus ca parola să fie introdusă de două ori pentru a fi o sigur că parola a fost introdusă corect.

```

if(TextUtils.isEmpty(pass)){
    eTextpassword.setError("Introdu o parola!");
    return; }
if(pass.length()<5){
    eTextpassword.setError("Parola trebuie sa fie mai mare de 5 caractere!");
    return; }
if(TextUtils.isEmpty(passConfirm)){
    eTextPassConf.setError("Introdu din nou parola!");
    return; }
if(passConfirm.length()<5){
    eTextPassConf.setError("Parola trebuie sa fie mai mare de 5 caractere!");
    return; }
if(!pass.equals(passConfirm)){
    eTextPassConf.setError("Parolele nu sunt la fel");
    return; }

```

Fig. 2.1.2 Validarea parolei

Dacă datele au fost introduse corect, se va efectua salvarea acestora. Pentru asta ma folosesc de funcția de autentificare oferită de Firebase, autentificarea prin email și parolă. Pentru salvarea noului utilizator voi folosi apelul metodei `createUserWithEmailAndPassword` oferit de `FirebaseAuth`.

II.2 Logarea

Partea de logare este asemănătoare atât pentru copil cât și pentru părinte. Copilul nu poate face nimic cu aplicația, ea fiind activată de către parinte. Logarea se face cu ajutorul emailului și al parolei.

Cei de la Firebase pun la dispoziție foarte multe lucruri. După ce utilizatorul s-a logat, aplicația trebuie să știe cine este utilizatorul current. Pentru asta mă folosesc de proprietatea *currentUser*.

```
firebaseAuth.signInWithEmailAndPassword(email, pass).addOnCompleteListener( activity: this, (task) → {  
    progressDialog.dismiss();  
    if (task.isSuccessful()) {  
        finish();  
        startActivity(new Intent(getApplicationContext(), ProfileActivity.class));  
    } else {  
        Toast.makeText( context: LoginActivity.this, text: "Emailul si parola nu sunt valide!", LENGTH_LONG).show();  
    }  
});
```

Fig. 2.2.1 Metoda implementată pentru logare

Și pentru logare datele introduse sunt validate. Dacă datele introduse sunt greșite, aplicația va afișa mesaje de atenționare. Dacă datele au fost introduce corect utilizatorul se poate loga. Aplicația dezvoltată de mine poate rula atât în modul background cât și în modul on-screen.

După ce s-a logat aplicația trebuie să știe cine este utilizatorul curent. Pentru asta voi folosi o metodă oferită de cei de la firebase. De fiecare dată când intru voi face o interogare cu userul curent.

```
if (firebaseAuthNote.getCurrentUser() == null) {  
    finish();  
    startActivity(new Intent( packageContext: this, LoginActivity.class));  
}
```

Fig 2.2.2 Verificare user current

II.3 Google Maps. Gestionarea datelor

În această parte voi prezenta funcționalitatea codului care se ocupă de:

- Hartile Google Maps
- Modul în care datele sunt preluate
- Prelucrarea, salvarea și reutilizarea datelor

În principal aplicația dezvoltată de mine se folosește de hărțile oferite de cei de la Google. După configurarea API-ului în aplicație, hărțile și funcționalitățile pot fi folosite foarte ușor.

Pentru monitorizarea poziției vom avea nevoie de coordonatele GPS ale utilizatorului, care vor fi citite din 5 în 5 minute, prelucrate și transmise mai departe către baza de date unde vor fi stocate. De la GPS vom avea nevoie de latitudine și longitudine, date cu care vom lucra mai departe în hărțile oferite de cei de la Google Maps. Pe lângă latitudine și longitudine, funcția oferă și date exacte despre adresa utilizatorului: oraș, strada, etc.

```
void getLocationUsingNetworkProvider() {  
    try {  
        locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);  
        locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, minTime: 3000, minDistance: 5, listener: this);  
    }  
    catch (SecurityException e) {  
        e.printStackTrace();  
    }  
}  
  
public void onLocationChanged(Location location) {  
    latitudine = location.getLatitude();  
    longitudine = location.getLongitude();  
    try {  
        Geocoder geocoder = new Geocoder(context, Locale.getDefault());  
        List<Address> addresses = geocoder.getFromLocation(location.getLatitude(), location.getLongitude(), maxResults: 1);  
        addressLocation = (addresses.get(0).getAddressLine(index: 0) + ", " +  
            addresses.get(0).getAddressLine(index: 1) + ", " + addresses.get(0).getAddressLine(index: 2));  
    }  
    catch (Exception e) {  
    }  
}
```


Fig. 2.3.1 Funcția pentru aflarea locației

Datele furnizate de GPS sunt salvate în baza de date, pentru a putea fi folosite mai departe la gestionarea locațiilor.

```
final DatabaseReference databaseReferenceNote = databaseReference.push();  
final Map noteMap = new HashMap();  
noteMap.put("copil", nameCopil);  
noteMap.put("longitudine", longitudine);  
noteMap.put("latitudine", latitudine);  
noteMap.put("adresa", addressLocation);  
noteMap.put("timestamp", ServerValue.TIMESTAMP);  
Thread mailThread = new Thread((Runnable) () -> {  
    databaseReferenceNote.setValue(noteMap).addOnCompleteListener((task) -> {  
        if (task.isSuccessful()) {  
            Toast.makeText(getApplicationContext(), text: "Adresa a fost salvata", Toast.LENGTH_LONG).show();  
        } else {  
            Toast.makeText(getApplicationContext(), text: "Eroare la salvarea datelor", Toast.LENGTH_LONG).show();  
        }  
    }  
});
```

Fig. 2.3.2 Salvarea datelor

Mai departe vom folosi hărțile celor de la Google Maps pentru a seta pe hartă locurile interzise și locurile acceptate. Acestea vor fi setate pe hartă cu ajutorul unor marcare de diferite culori. Pentru locurile interzise se folosește culoarea roșie și pentru locurile bune culoarea verde. De exemplu pentru setarea unui locurilor interzise. Vom selecta pe hartă o locație după care cu ajutorul unui progress bar vom crea o arie de acoperire care ne va furniza adresa caracterizată de latitudine și longitudine, aria de acoperire si tipul locației. Aceastea vor fi salvate în baza de date si folosite mai târziu pentru monitorizare. La fel se va face si pentru locatiile acceptate. Daca un utilizator a intrat într-o locație interzisă vom fi notificați.



```

case R.id.add_bad_location:
    pushPin.setBackground(context.getResources().getDrawable(R.drawable.push_pin_bad));
    isBadLocationSelected = true;
    isGoodLocationSelected = false;
    pushPin.setVisibility(View.VISIBLE);
    break;

public void drawArea(LatLng latLng, int range) {
    currentRange = range;
    if (circle != null) circle.remove();
    int strokeColor = context.getResources().getColor(R.color.color_bad_Location_stroke);
    int fillColor = context.getResources().getColor(R.color.color_bad_Location_fill);
    CircleOptions circleOptions = new CircleOptions()
        .center(latLng)
        .radius(range)
        .strokeColor(strokeColor)
        .fillColor(fillColor);
    circle = mMap.addCircle(circleOptions);

case R.id.map_location_save:
    Position position = new Position();
    position.setGoodLocation(isGoodLocationSelected);
    CameraPosition cameraPosition1 = mMap.getCameraPosition();
    LatLng latLng1 = cameraPosition1.target;
    position.setLat(latLng1.latitude);
    position.setLng(latLng1.longitude);
    position.setLocationTitle(locationName.getText().toString());
    position.setRadius(currentRange);
    String type = (position.isGoodLocation()) ? "good" : "bad";
    FirebaseDatabase.getInstance().getReference()
        .child("Users")
        .child(firebaseAuth.getCurrentUser().getUid())
        .child("positions")
        .child(type)
        .child(position.getLocationTitle())
        .setValue(position);

```

Fig. 2.3.1 Marcarea și salvarea punctelor rele

II.4 Profilul utilizatorului și partea de notițe

Aplicația oferă adminului posibilitatea creării unui profil și capacitatea de a-și salva anumite notițe personale.

După înregistrare utilizatorul este redirecționat către pagina cu generarea profilului. Aici, se pot seta datele personale dar și posibilitatea de a seta o poză de profil. Poza de profil va fi selectată din telefon și stocată în baza de date. Înainte de salvare vom face o copie la URL general de salvarea în baza de date. Url găsit va fi salvat împreună cu numele, prenumele, emailul, telefonul și adresa utilizatorului. Afisare se face relativ simplu, cu ajutorul url-ului imaginea va fi preluată din storage-ul oferit de cei de la Firebase.

```
private void uploadImageToFirebaseStorage() {
    final StorageReference profileImageRef=
        FirebaseStorage.getInstance().getReference( s: "profilepics/"+System.currentTimeMillis()+".jpg");
    if(uriPhoto!=null){
        progressBar.setVisibility(View.VISIBLE);
        uploadTask = profileImageRef.putFile(uriPhoto);
        Task<Uri> urlTask = uploadTask.continueWithTask((task) -> {
            if (!task.isSuccessful()) {
                throw task.getException();
            }
            return profileImageRef.getDownloadUrl();
        }).addOnCompleteListener((task) -> {
            progressBar.setVisibility(View.GONE);
            if (task.isSuccessful()) {
                downloadURL = task.getResult();
                // editTextSaveNameDB.setText(downloadURL.toString());
                Toast.makeText( context: EditUser.this, text: "Imaginea a fost salvata!", LENGTH_LONG).show();
            } else {
                Toast.makeText( context: EditUser.this, text: "Eroare la salvarea datelor!", LENGTH_LONG).show();
            }
        });
    }
}
```

Fig.2.4.1 Salvarea unei imagini în baza de date

Pentru salvarea notitelor am creat o clasa NoteModel în care am salvat numele notiței, descrierea și timpul în care a fost realizată. Ele vor fi salvate în baza de date separat pentru fiecare utilizator. Datorită funcționalităților oferite de cei de la Firebase avem posibilitatea să salvăm, să ștergem și să actualizăm în timp real.

Capitolul III Manual de utilizare

După instalarea aplicației aceasta poate fi utilizată foarte ușor. Vom căuta aplicația în meniul telefonului și o vom deschide.

Când deschidem pentru prima dată aplicația va apărea un splash screen cu numele și logoul aplicației.

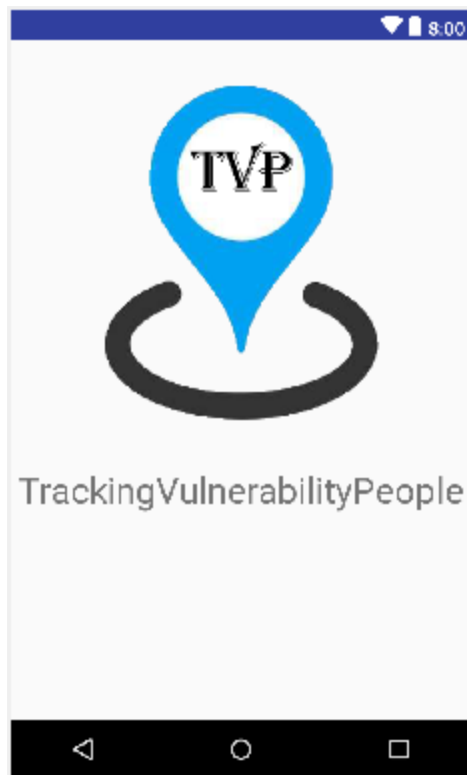


Fig. 3.1 Pagina de start

Aplicația oferă utilizatorului siguranță, el având posibilitatea să-și păstreze datele în siguranță cu ajutorul creării unui cont. Dar mai întâi de toate acesta trebuie să se înregistreze. După ce ecranul de start a dispărut, va apărea activitatea în care utilizatorul are capacitatea de a se loga, înregistra și recupera parola, dacă aceasta a fost uitată.

Mai întâi vom crea un cont, prin simpla apăsare a butonului **"Not have an account? Sing up here!"**. După apăsarea butonului se va deschide activitatea destinată înregistrării. În imaginile care urmează voi prezenta pas cu pas crearea unui cont nou.

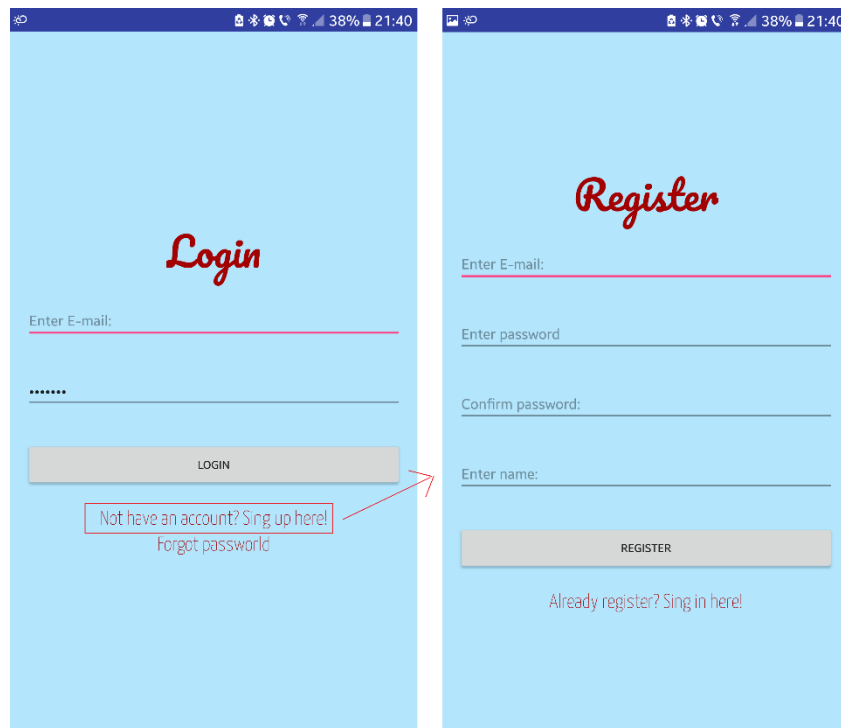


Fig.3.2 Logarea și înregistrarea

Pentru a putea folosi aplicația, utilizatorul trebuie să se înregistre cu ajutorul emailul, parola și numele acestuia. Adresa trebuie să fie una validă având formatul [exemplu1995@yahoo.com](#), [exemplu2108@gmail.com](#).

Dacă datele au fost introduce corect sunt trimise la baza de date, utilizatorul putând folosi aplicația. Înainte de asta voi prezenta și partea de recuperare parole. Daca parola a fost uitat se va trimite un email prin care utilizatorul o poate reseta

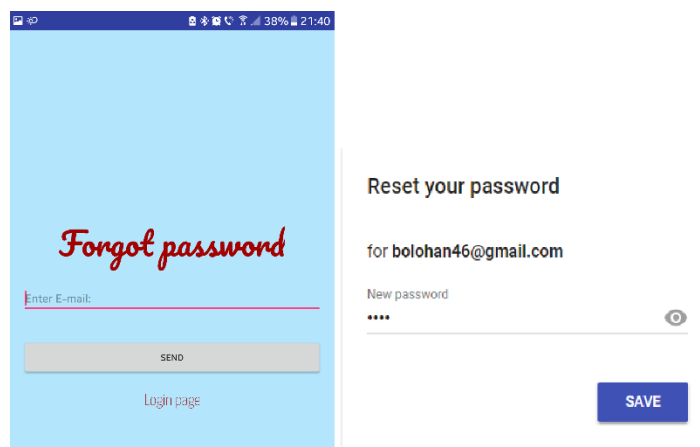


Fig. 3.3 Recuperarea parolei

După ce utilizatorul a reușit să se logeze acesta poate intra și folosi aplicația. Mai întâi acesta va trebui să-și configureze profilul.

The image displays two side-by-side screenshots of a mobile application's user profile screen. Both screens have a light green background and a blue header bar. The header bar contains a 'PROFILE' button on the left and a 'LOGOUT' button on the right. A circular profile picture of a man in a suit is centered at the top. Below the picture, there are five input fields. In the left screenshot, the fields are empty and labeled 'Enter name', 'Enter address', 'Enter email:', 'Enter phone:', and 'Enter age:'. In the right screenshot, the fields are filled with the following data: 'Cosmin Alex', 'Salcea Suceava', 'bolohan46@gmail.com', '0741376109', and '23'.

Field	Value
Name	Cosmin Alex
Address	Salcea Suceava
Email	bolohan46@gmail.com
Phone	0741376109
Age	23

Fig. 3.4 Profilul utilizatorului

Dacă datele vor fi introduse corect, acestea vor fi salvate și afisate ulterior pentru editare și vizualizare.

Dacă acest pas a fost finalizat cu succes utilizatorul poate folosi aplicația. După ce a fost instalată și pe telefonul copilului în meniul aplicației va apărea o listă cu persoanele pe care le putem supraveghea. Vom alege una din ea, după care vom deschide harta și vom începe urmărirea acesteia. Pe hartă vom putea vizualiza poziția curentă, locurile interzise și locurile bune. Pentru început vom marca pe hartă câteva zone de siguranță și un loc interzis. Pentru asta avem nevoie să selectăm persoana care va fi monitorizată și să intrăm în activitatea de monitorizare.

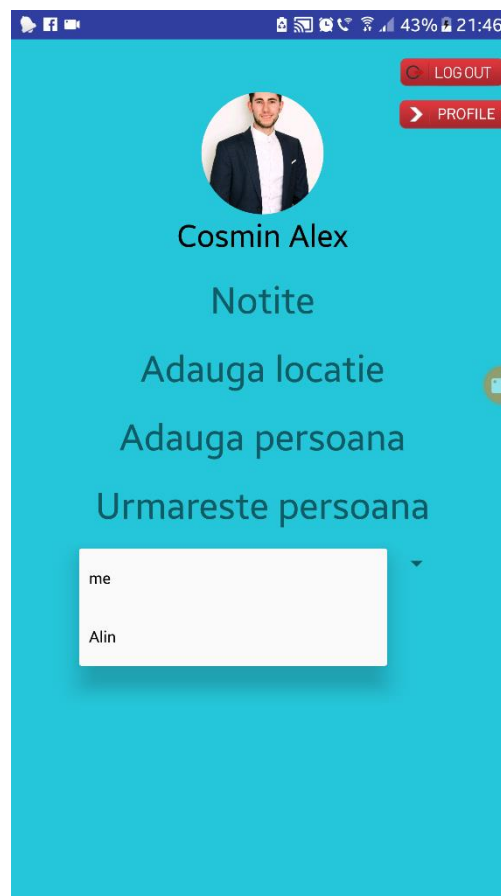


Fig.3.5 Meniu utilizator

În imaginile care urmează voi prezenta modul prin care supraveghetorul marchează pe hartă locurile de interes, dar și felul în care vor apărea persoanele urmărite. În doar 5 pași acesta poate seta pe hartă locațiile permise cât și cele interzise.

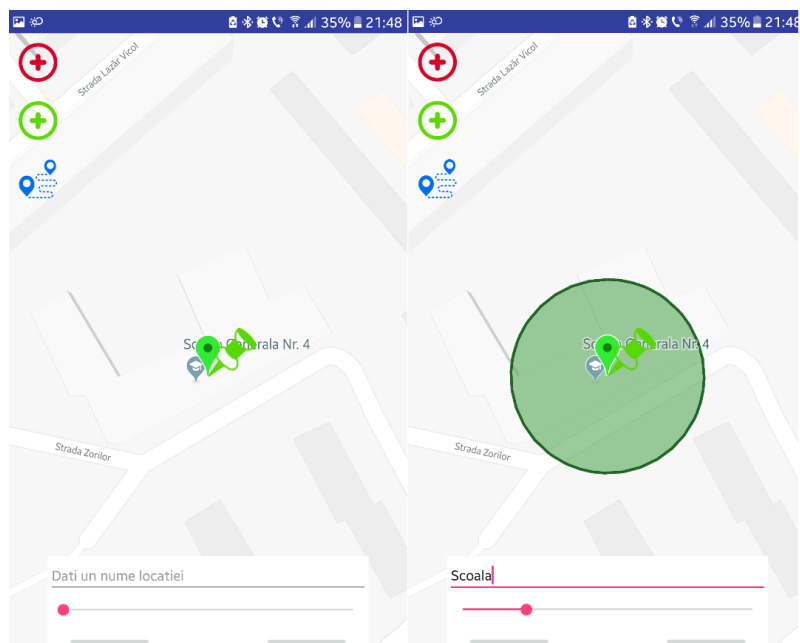


Fig. 3.6 Setarea unui loc permis

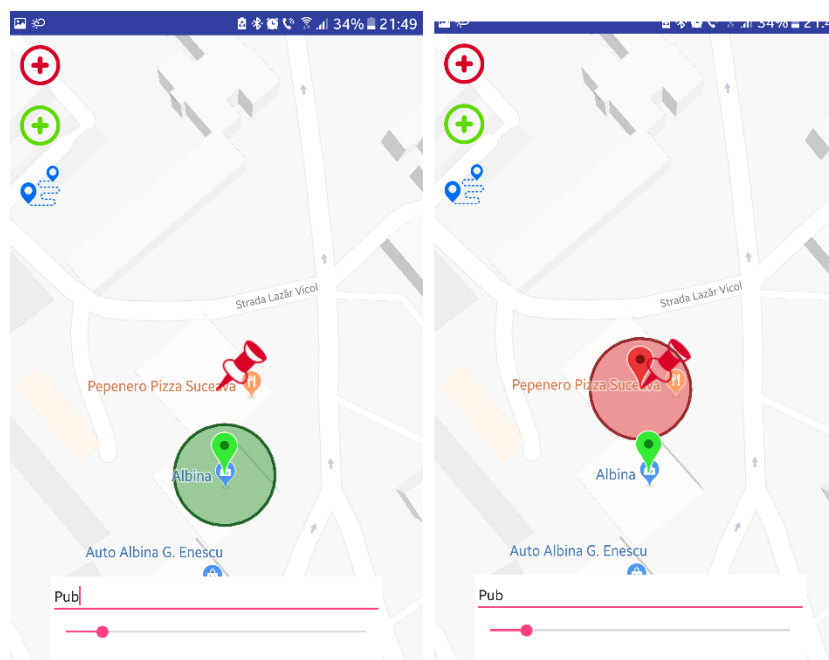


Fig. 3.7 Marcarea unui loc interzis

După ce am selectat locurile permise și locurile interzise putem urmări pe hartă poziția persoanelor de interes. Putem vedea poziția în timp real, iar dacă persoana urmărită a pătruns într-o locație interzisă vom primi o notificare iar persoana va fi avertizată că a pătruns într-o locație rea.

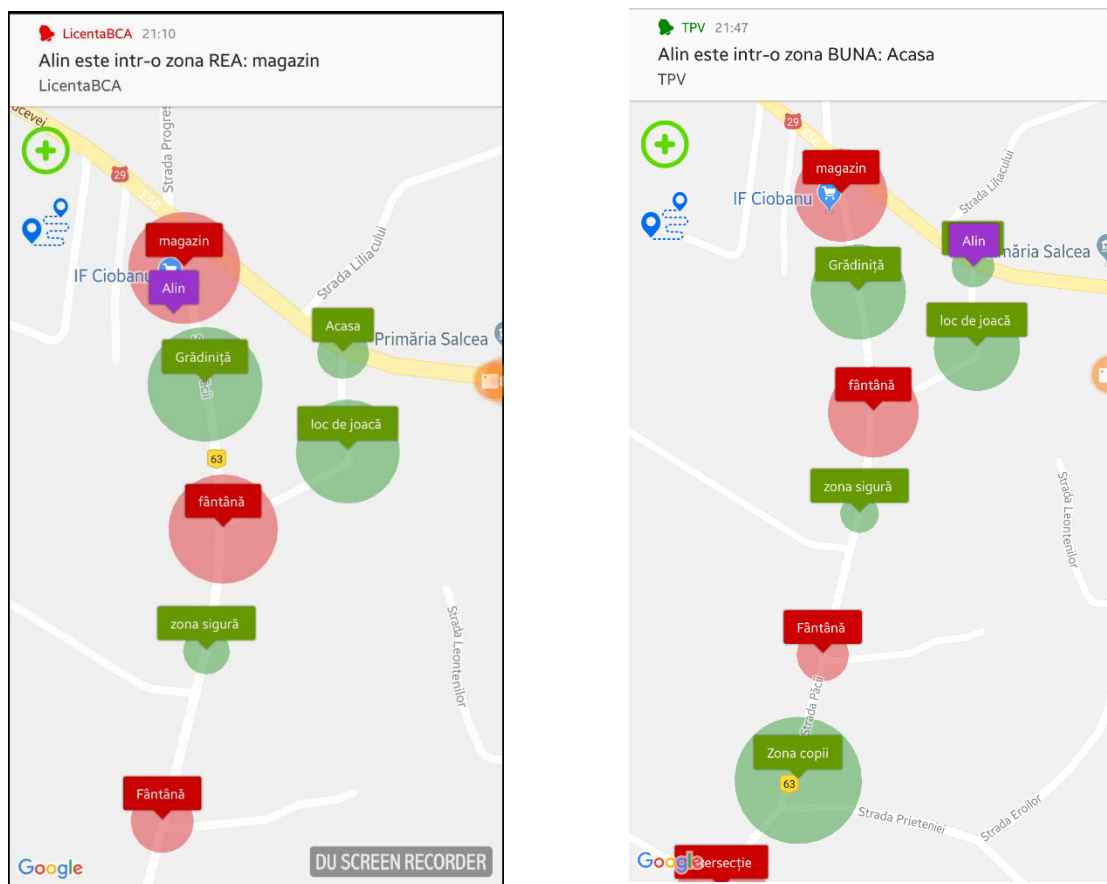


Fig.3.8 Monitorizarea pe hartă

După cum se poate vedea în Fig. 3.8, părintele este avertizat atunci când copilul părăsește o locație bună și intră în una rea. Se poate vedea și poziția în timp real al persoanei.

Concluzii

Aplicația implementată de mine are drept scop exploatarea tehnologiilor oferite de Android, Google Maps și Firebase în contextual realizării unei aplicații benefice, cu scopul de a face lumea un loc cât mai sigur, prin monitorizarea persoanelor cu probleme sau a copiilor.

În decursul realizării acestui proiect m-am confruntat cu diferite probleme legate de modul și structura în care vor fi salvate datele, preluarea locațiilor oferite de GPS și modul în care copii pot fi văzuți pe hartă.

Aplicației îi pot fi aduse multe îmbunătățiri care ar face monitorizarea și mai sigură. O primă modificare ar fi prin realizarea unui orar în care copilul trebuie să stea în interiorul zonelor sigure, spre exemplu, când este la școală se va putea seta un anumit interval orar. O altă îmbunătățire ar fi prin verificarea vitezei cu care se deplasează copilul, sau cât de repede își modifică poziția. Acest lucru ar putea fi folosit în momentul în care el are setat ca în intervalul orar 10-11 să fie în parc. În cazul cel mai rău, când copilul ar urca într-o mașină am primi o avertizare de tip sonor că locația a fost părăsită și copilul se află într-un autovehicol.

În concluzie, consider că am reușit să realizez ceea ce mi-am propus, urmând ca pe viitor să îmbunătățesc aplicația și să o aduc la un alt nivel. Am început acest proiect cu un bagaj de cunoștințe relative scăzute și consider că am reușit să aprofundez și să învăț bazele programării în Android Studio.

Bibliografie

[https://en.wikipedia.org/wiki/Android_\(operating_system\)](https://en.wikipedia.org/wiki/Android_(operating_system))

<https://developer.android.com/guide/components/processes-and-threads>

www.tutorialspoint.com

www.developer.android.com

www.stackoverflow.com

<https://developer.android.com/reference/android/location/Location>