UNIVERSITATEA DE VEST DIN TIMIŞOARA

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

PROGRAMUL DE STUDII DE LICENŢĂ: Informatică

LUCRARE DE LICENŢĂ

COORDONATOR:

Lector Dr. Liviu Octavian Mafteiu Scai

ABSOLVENT:

Iosifoni Valentin Tudor

TIMIŞOARA

2021

UNIVERSITATEA DE VEST DIN TIMIŞOARA

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

PROGRAMUL DE STUDII DE LICENŢĂ: Informatică

APLICAȚIE MOBILĂ PENTRU PARAPANTISM

COORDONATOR:

Lector Dr. Liviu Octavian Mafteiu Scai

ABSOLVENT:

Iosifoni Valentin Tudor

TIMIŞOARA

2021

Cuprins

Listă de figuri

Rezumat

Lucrarea de licență constă în realizarea unei aplicații mobile care va ușura zborul cu parapanta. În interiorul acesteia sunt disponibile mai multe facilități cum ar fi un asitent de zbor care va oferi o orientare mai bună în aer utilizatorului, vizualizarea locațiilor unde se poate practica parapantismul, estimarea condiților de zbor băzată pe procnoza meteo curentă și viitoare. Realizarea acestora se face utilizând limbajul de programare java împreună cu Google Maps, senzorii de poziție ai telefonului, Firebase ca și bază de date și informații meteo preluate de pe openweathermap.org . Aplicația este disponibilă pentru sistemul de operare Android și poartă numele de FlyMaster. Lucrarea este structurată în 4 capitole acestea fiind:

- Descrierea problemei și abordări existente unde se prezintă tehnologiile necesare realizării aplicației și implementările acestora în alte programe similare.
- Arhitectura Aplicației în care sunt prezente descrierile grafice a posibilelor interacțiuni
- Facilități Aplicație unde se prezintă fiecare ecran cu care utilizatorul va interacționa
- Detalii de implementare unde sunt descrise structurile de cod importante

Abstract

The bachelor's thesis consists of a mobile application that will make paragliding easier. Inside it are available several facilities such as: a flight attendant that will provide better orientation in the air to the user, the ability to view locations where paragliding can be practiced, the estimation of flight conditions based on current and future weather forecast. They are made using: java programming language ,Google Maps, the position sensors of the phone, Firebase as the database and weather information taken from openweathermap.org. The application is available for the Android operating system and is called FlyMaster. The structure of the thesis is made up of 4 chapters:

- Descrierea problemei și abordări existente Where the necessary technologies are presented along with their implementations in other programs
- Arhitectura Aplicației in which the graphical descriptions of possible interactions with the application are found
- Facilități Aplicație illustrates and describes each screen of the application
- Detalii de implementare where the important code structures in the application are found explained in detail

Introducere

Parapantismul este un sport extrem care constă în planarea ajutată doar de curenții de aer, deseori practicat de cei care sunt dependenți de senzații tari. Decolarea se face prin plasarea aripi într-un flux de aer, fie prin alergare, fie prin tragere, fie printr-un vânt existent. Aripa se deplasează peste pilot într-o poziție în care poate transporta pasagerul. Pilotul este apoi ridicat de la sol și, după o perioadă de siguranță, se poate așeza în ham. Echipamentul utilizat nu folosește un motor, cu toate acestea zborurile pot dura câteva ore și pot parcurge sute de kilometri. Faptul ca toată activitatea se desfășoara fara ajutorul unei elice face aceast sport să fie incredibil de mult influențat de condițile meteo. Datorită dezvoltari tehnologiei, parapantismul poate fi ușurat folosind telefonul mobil.

Figura 1: Parapanta în timpul zborului

Aplicația mobilă realizată are rolul de a oferii informații utile cum ar fi: vizualizarea pe google maps a locațiilor care sunt amenajate cu poligoane de decolare și aterizare, estimarea condiției de zbor în acele locații bazată pe direcția și viteza vântului, procnoza meteo pe urmatoarele 5 zile a locației respective, și câteva sfaturi care ar trebui să fie știute de practicanții acestui sport.

În interiorul acesteia este disponibil un “asistent de zbor” care odata pornit va emite semnale sonore distincte atunci când altitudinea crește sau scade pentru a putea determina mai ușor viteza de deplasare în aer. Acestă funcționalitate are rolul de a înlocui un aparat folosit de parapantiști numit altivariometru. Aplicația poartă numele de Fly Master și este disponibilă pentru telefoanele mobile care folosesc sistemul de operare Android.

Capitolul 1

Descrierea problemei și abordari existente

Afișarea zonelor de zbor rezprezintă un element important al acestei aplicații. Pentru a putea ușura modul de vizualizare și a oferi utilizatorului o reprezentare cât mai bună a locației respective sunt implementate hărțile oferite de Google Maps sub diferite aspecte în funcție de modul de preferință al celui care folosește aplicația. Google pune la dispoziție o vizualizare completă a globului pământesc cea ce permite o cartografiere foarte amplă a zoneler destinate activităților de parapantism. Hartile sunt complet editabile și permit schimbarea felului în care utilizatorul interacționează cu acestea. Sunt permise adăugarea următoarelor elemente grafice:

- Pictograme ancorate la poziții specifice de pe hartă (Markers).
- Seturi de segmente de linie
- Segmente închise
- Grafică bitmap ancorată pe poziții specifice de pe hartă
- Seturi de imagini care sunt afișate deasupra segmentelor hartii.

Doua aplicații foarte cunoscute care implementează Google Maps sunt Pokemon go și Snapchat. Snapchat este o platformă socială, unde printre alte funcționalități, utilizatorul poate vedea locația actuală a prietenilor săi iar Pokemon go este un joc cu ideea de treasure hunt, drept urmare ele trebuie să ofere informații diferite sub un format diferit.

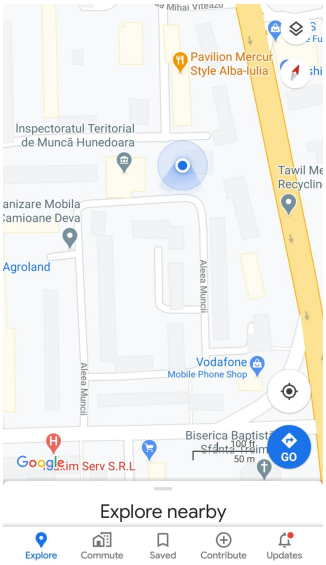
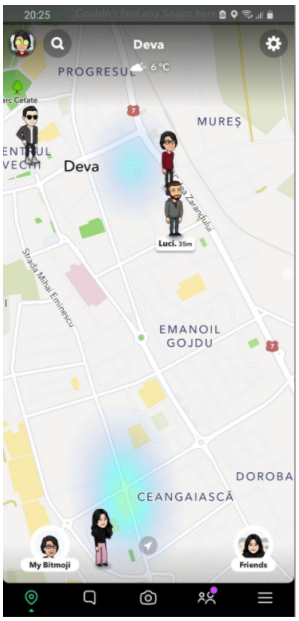


Figura 1.1: Harta vazută inițial în google maps

(a) PokemonGo



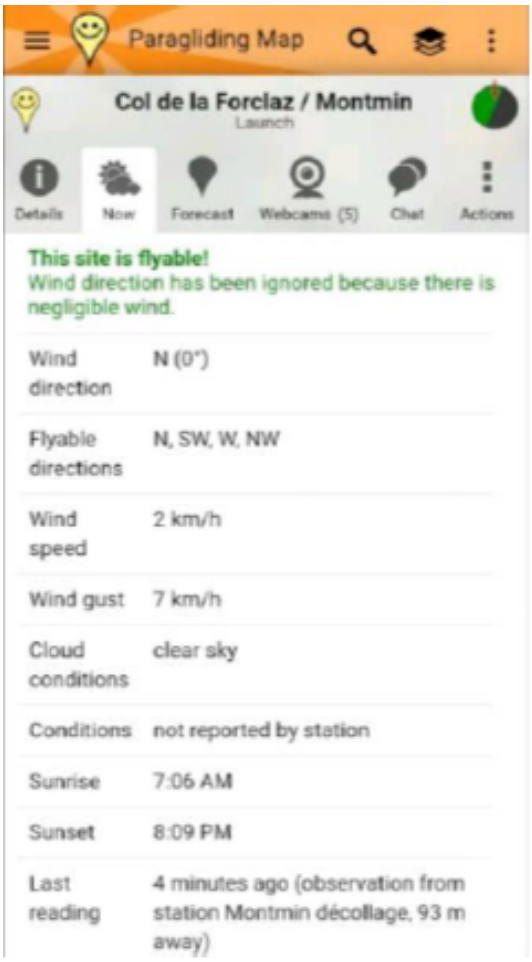
(b) Snapchat

Figura 1.2: Aplicații care implementează harți

În imaginile a și b este prezentată aceeași locație ca în figura 1.1

Este necesar ca utilizatorul să poata accesa procnoza meteo la o anumită locație. Pentru implementarea aceastei utilitați se extrag informații de pe openweathermap.org care pune la îndemână date colectate de la stații meteorologice de pe întregul glob, făcând posibilă vizualizarea procnozei în orice loc desemnat zborului cu parapanta. O aplicație care folosește această baza de date pentru determinarea condiților de zbor cu parapanta este Paragliding Map

(a) Detalii locație



(b) Estimarea condiților

Figura 1.3: Paragliding Map

Capitolul 2

Arhitectura Aplicației

2.1 Cazurlie de utilizare

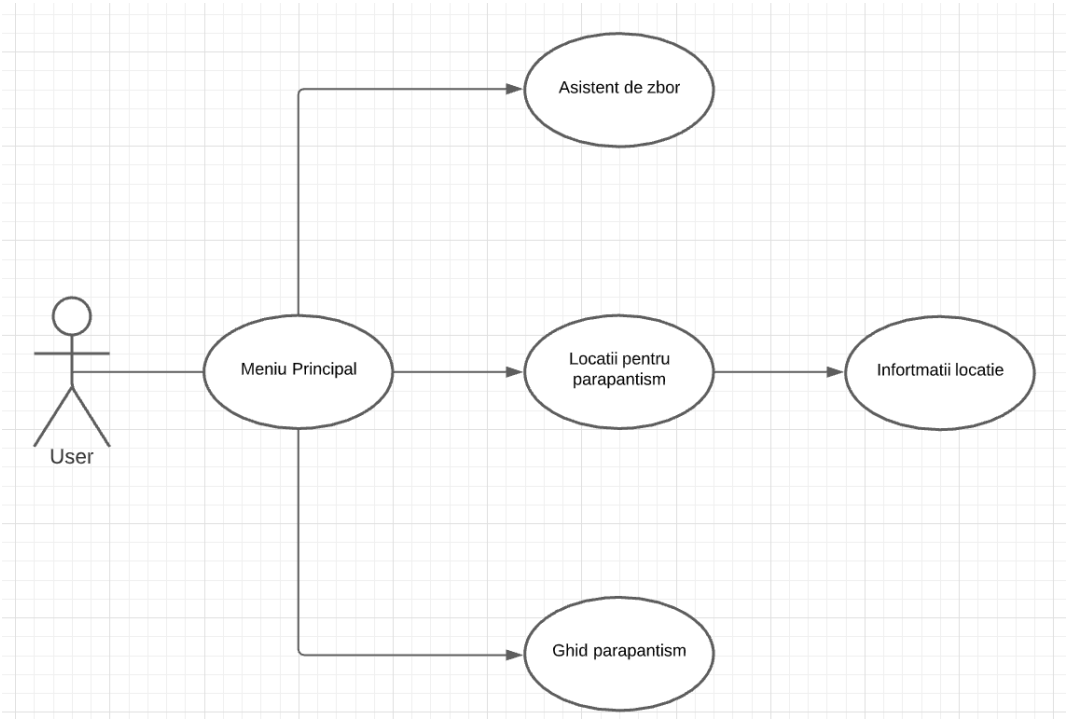


Figura 2.1: Diagrama use case

La deschiderea aplicației utilizatorului îi este prezentat un meniu cu urmatoarele optiuni:

- Asistent de zbor, unde este prezentă locația curentă și funcționalitatea de asistent
- Locații pentru parapantism, unde sunt prezentate locațiile care permit zborul.
- Ghid de parapantism, în interiorul căruia sunt prezentate informații despre acest sport

2.2 User Flow Diagram

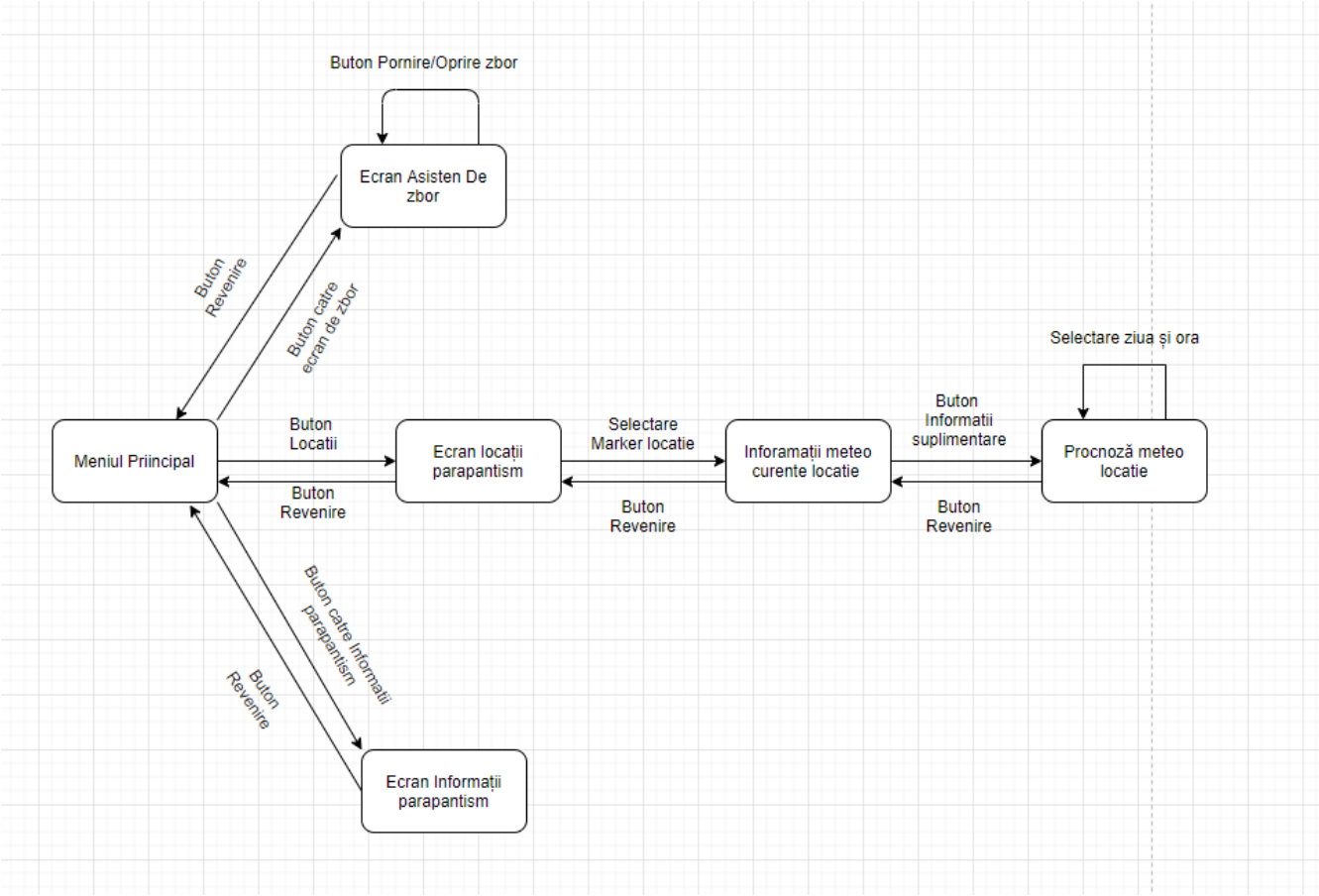


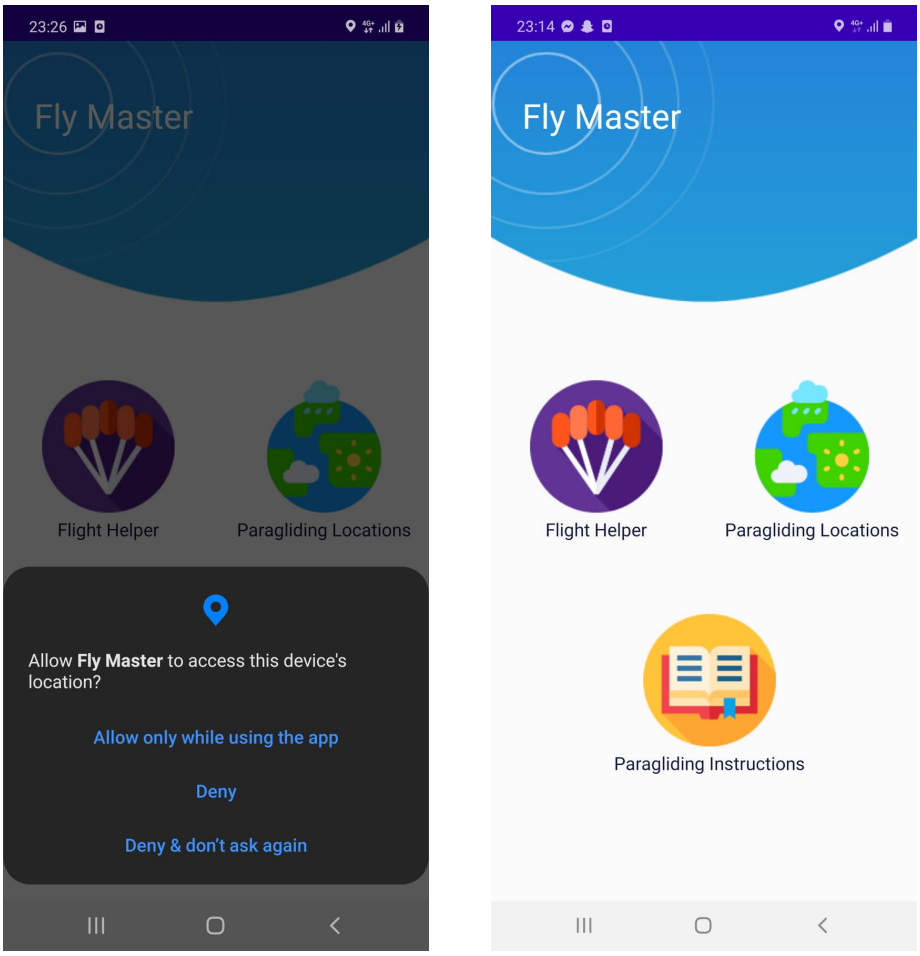
Figura 2.2: Diagrama User Flow

În Figura 2.2 sunt descrise în detaliu fiecare activitate pe care utilizatorul o poate efectua în interiorul aplicației.

Capitolul 3

Facilități Aplicație

La accesarea aplicației din telefonul mobil, aceasta se deschide cu o animație și prezintă un ecran utilizatorului cu 3 opțiuni: Flight Helper, Paragliding locations, Paragliding Instructions. Dacă aplicația este deschisă pentru prima dată sau dacă nu a fost acceptată până acum, se cere permisiunea de utilizare a locației.



(a) Cererea Permisiunii (b) Meniul Principal

Figura 3.1: Deschiderea aplicației

3.1 Flight Helper

Aplicația este concepută cu scopul de a ușura zborul cu parapanta și de a oferi informații utile celor care o utilizează.

Partea de asistent de zbor poate fi accesată din meniul principal apăsând pe butonul numit "Flight Helper". În interiorul acestui ecran sunt disponibile informații despre locația curentă oferite sub forma de latitudine, longitudine și altitudine. Utilizatorul le poate transforma în oraș și județ apăsând un buton numit convert. În partea de sus a ecranului se află o busolă marcată cu cele 4 puncte cardinale, rotindu-se în funcție de orientarea telefonului.

În partea de jos a ecranului se află un buton numit "Start Flight" care odata apăsat va emite semnale sonore distincte în funcție de scăderea sau creșterea altitudinii. Semnalele sonore împreună cu busola sunt menite să ușureze zborul cu parapanta deoarece fără acestea orientarea este mult mai dificilă. Practicanții acestui sport folosesc un aparat cu scop similar numit Alivariometru, dar acum telefonul îl va putea înlocui.

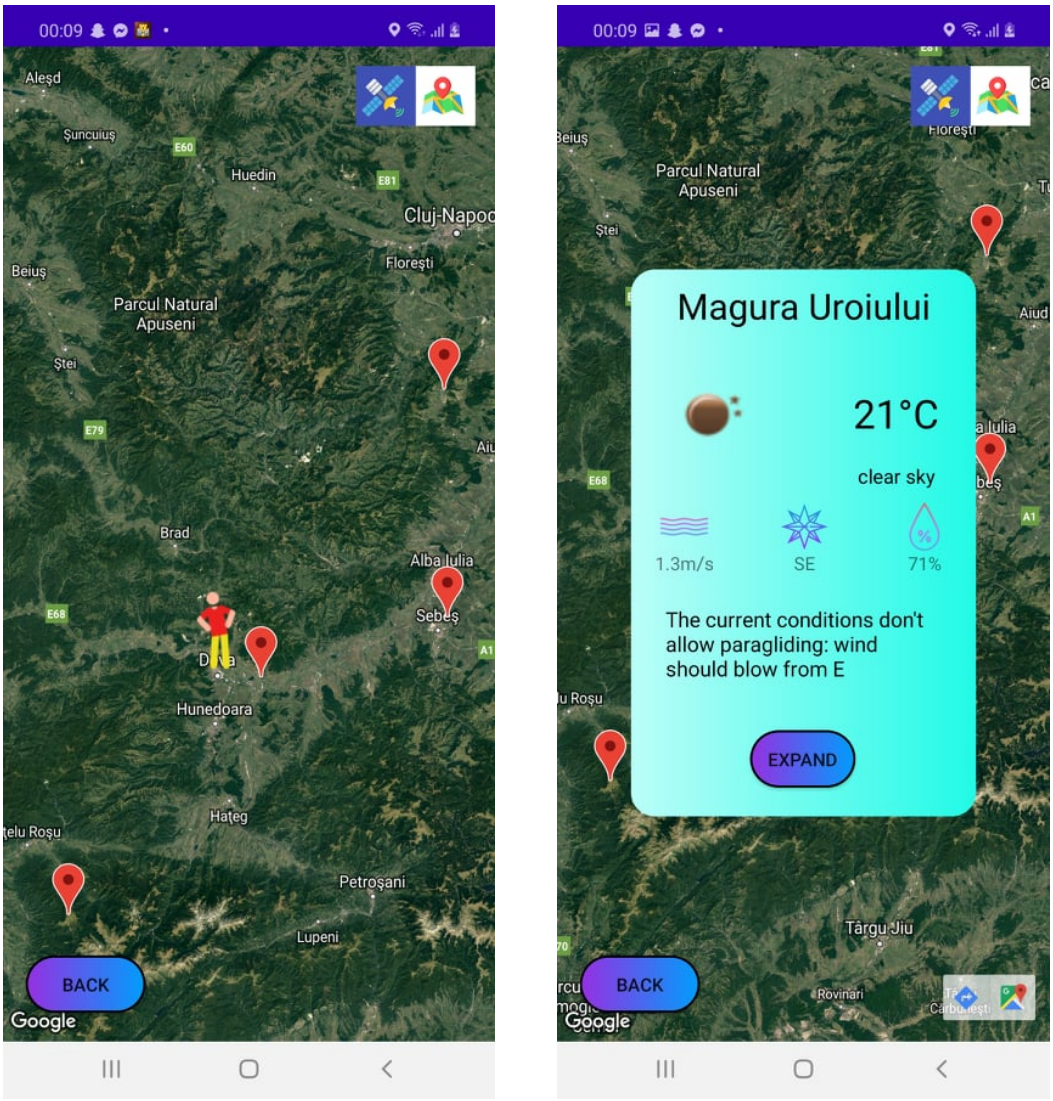
(a) Altivariometrul (b) Ecranul "Flight Helper"

Figura 3.2: Altivariometrul și echivalentul său în aplicație

3.2 Paragliding locations

La apăsarea opțiuni "Paragliding locations" utilizatorului îi este prezentată o hartă a globului pe care sunt ancorate "markere" în locațile care sunt prevăzute cu un poligon de aterizare și decolare. Harta este poziționată la deschidere pe locația actuală, aceasta fiind reprezentată de o pictogramă cu un om. În partea dreapta sus sunt doua butoane care pot altera modul de afișare al harții la vizualizare din satelit sau la vizualizare din punct de vedere al reliefului. Fiecare tip de hartă ajută într-un mod diferit utilizatorul cand vine vorba de înțelegerea topografiei locului destinat zborului. Harta cuprinde o selecție a mai multor locații aflate în Europa, predominând ca numar cele aflate în Romania, Austria, Spania.

La atingerea uneia dintre locațiile marcate se va deschide o fereastră peste hartă, care va oferi următoarele informații meteorologice actuale : viteza vântului și direcția acestuia, temperatura curentă, gradul de umiditate iar în funcție de cele enumerate va determina dacă zborul poate fi efectuat sau nu. Pe fereastră este disponibil un buton numit "Expand" care va deschide un nou ecran cu informații suplimentare legate de locul respectiv. În paretea de jos a ecranului din Figura 3.3 b se poate observa o săgeată în partea dreapta jos a ecranului. Daca aceasta este apasată se va deschide aplicația google maps cu o rută către acel loc plecand de la locația curentă.



(a) Harta plasată pe locația actuală (b) Fereastra de informații meteo

Figura 3.3: Ecranul "Paragliding locations"

3.3 Procnoza meteo

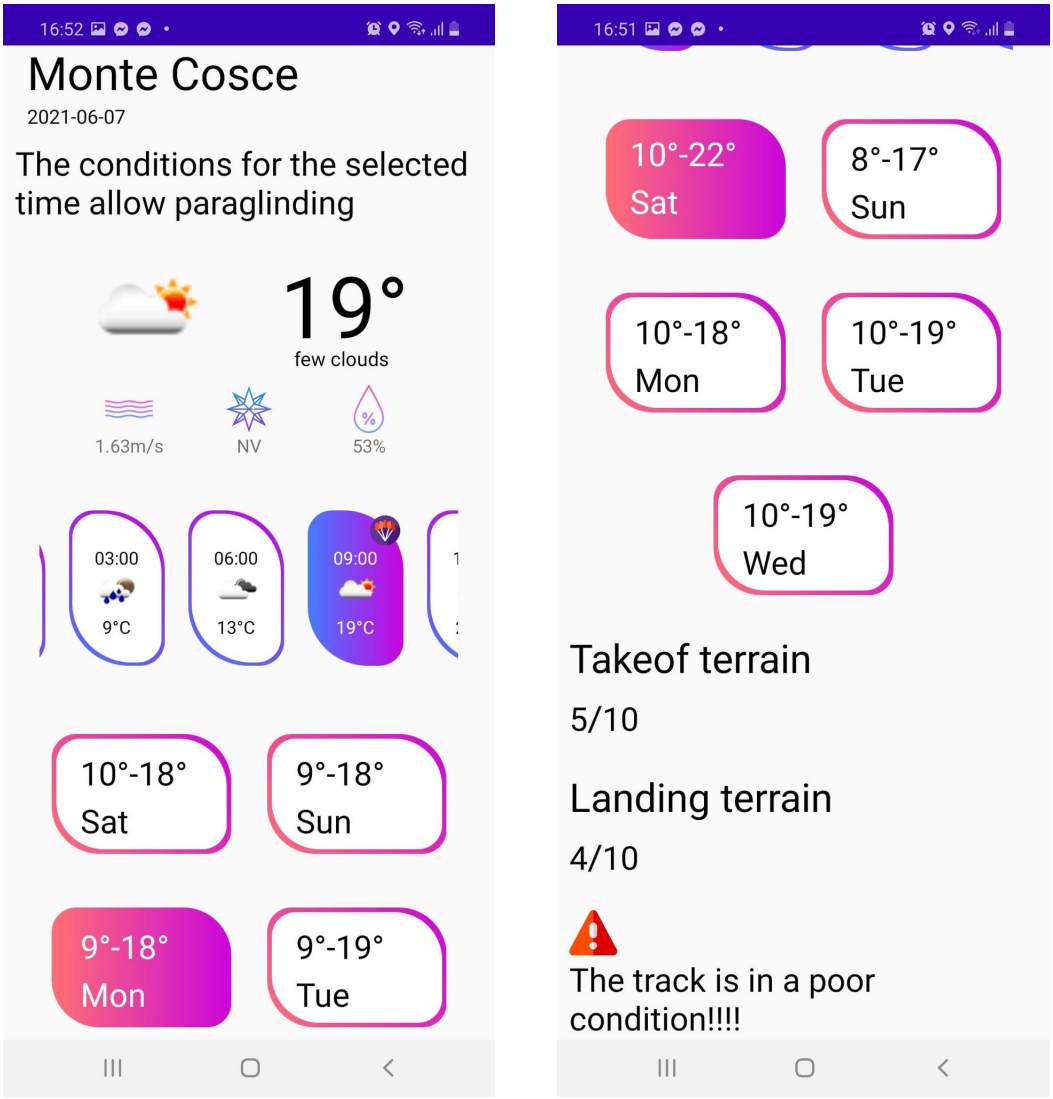
Dacă utilizatorul decide să acceseze secțiunea de informații suplimentare a unei locații se va deschide un ecran unde sunt disponibile informații meteorologice pentru urmatoarele 5 zile. Acesta poate alege un interval orar pentru a vedea condițile viitoare de zbor. În partea de jos a ecranului sunt afișate note pentru starea actuală a poligonului de decolare și aterizare și un avertisment dacă sunt prea scăzute. Notele au foste estimate în funcție de informațile găsite pe mai multe forumuri destinate parapantismului.

Cartonașele violet din partea de sus reprezintă un interval orar. În interiorul acestora este afișată ora, o pictogramă care reprezintă condiția meteo, și temperatura respectivă. Dacă un cartonaș este selectat va deveni complet colorat, fiind distinct diferit de cele neselectate care au doar conturul colorat. Ora la care se estimează că

se poate zbura este semnalată cu o iconiță cu o parapantă în partea dreapta sus a cartonașului având rolul de a ușura găsirea unui moment prielnic zborului de către utilizator.

Puțin mai jos sunt disponibile 5 cartonașe cu formă și culoare diferită care reprezintă cele 5 zile care pot fi vizualizate. Acestea au scrise pe ele o aproximare a intervalului temperaturii din aceea zi și o prescurtare a zilei pe care o reprezintă. Se aplică aceleași condiții de selectare a cartonașelor de mai sus.

Informațile despre direcția și viteza vântului, dar și estimarea dacă se poate practica zborul din partea de sus a ecranului se schimbă în funcție de combinația de zi și oră care este selectată.



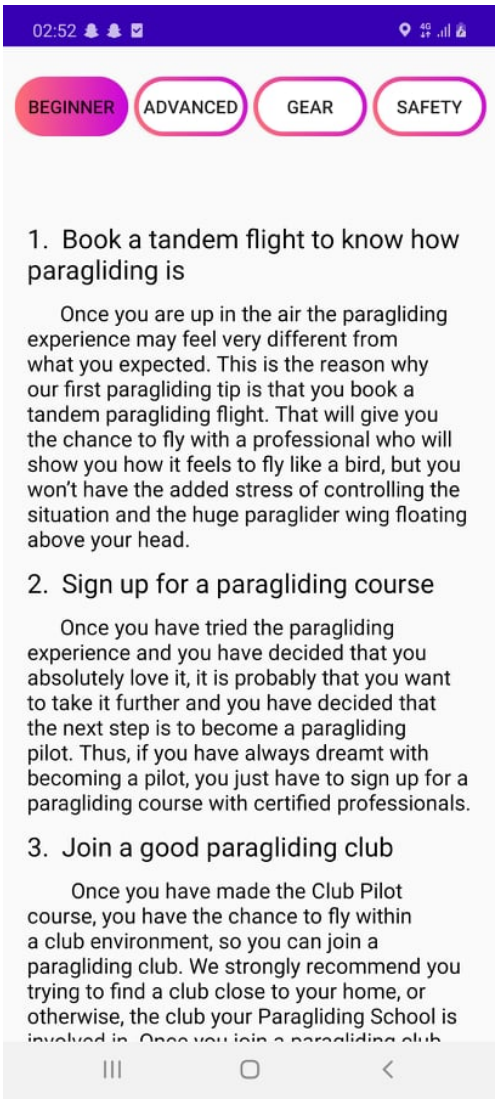
(a) Condițile vitoare de zbor (b) Condiția terenului

Figura 3.4: Ecranul cu detalile locației

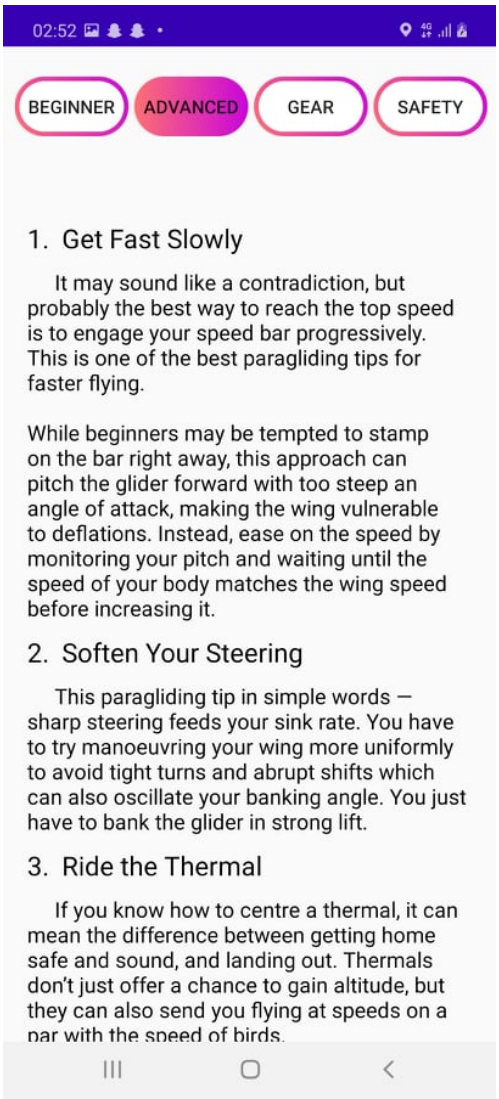
3.4 Paragliding Instructions

La accesarea secțiunii de "Paragliding Instructions" din ecranul principal utilizatorului îi sunt prezentate informații care au rolul de a explica anumite aspecte ale parapantismului.

Informațiile sunt împărțite în 4 categorii: Beginner, Advanced, Gear, Safety. Categoria afișată este reprezentată de butonul complet colorat aflat în partea de sus a ecranului, iar apăsărea oricăruia va schimba conținutul ecranului.



(a) Secțiunea Beginner



(b) Secțiunea Advanced

Figura 3.5: Ecranul "Paragliding Instructions"

Capitolul 4

Detalii de implementare

4.1 Android Studio

Android Studio este un editor dezvoltat de Google având ca fundație IntelliJ IDEA. În interiorul editorului sunt disponibile de la bun început multe dintre uneltele necesare dezvoltări aplicaților mobile, deoarece a fost creat strict cu acest scop. Android Studio oferă urmatoarele functionalități:

- Suport pentru construcții bazat pe Gradle
- Posibilitatea de a monitoriza performanța
- Sistem de protecție ProGuard
- Posibilitatea de a folosi șabloane predefinite
- Un editor pentru interfață cu aspect bogat, care permite utilizatorilor să tragă și să fixeze componente UI, dar și opțiunea de a previzualiza aspectul acesteia pe configurații diferite de ecran.
- Suport pentru integrarea Firebase Database
- Emulator pentru fiecare versiune de Android.

Fiecare proiect Android trebuie să aibă un fișier AndroidManifest.xml (cu exact acest nume) la baza proiectului. Fișierul manifest descrie informații esențiale despre aplicație pentru instrumentele de compilare Android, sistemul de operare Android și Google Play.

În manifest trebuie declarate întotdeauna următoarele:

- Numele pachetului aplicației. Instrumentele de compilare Android folosesc acest lucru pentru a determina locația entităților de cod atunci când proiectul este construit. La instalarea aplicației, instrumentele de compilare înlocuiesc valoarea

respectivă cu ID-ul aplicației din fișierele de construcție Gradle, care este utilizat ca identificator unic al programului în sistem și pe Google Play.

- Componentele aplicației, care includ toate activitățile, serviciile, receptoarele de difuzare și furnizorii de conținut. Fiecare componentă trebuie să definească proprietăți de bază, cum ar fi numele clasei sale Java.
- Caracteristicile hardware și software necesare, specificând exact dizpozitivele ce vor putea descărca aplicația din Google Play.
- Permisiunile de care are nevoie aplicația pentru a accesa părți protejate ale sistemului cum ar fi permisiunea de a accesa locația dispozitivului, sau de a folosi rețeaua de internet la care este conectat acesta. De asemenea, declară orice permisiuni pe care trebuie să le aibă alte programe dacă doresc să acceseze conținut din interiorul aplicație.

Gradle este un sistem de compilare care este utilizat pentru automatizarea construcției, testării, implementării etc.

Fiecare proiect Android are nevoie de un gradle pentru a genera un apk (Android application package) din fișierele .java și .xml din proiect. Un gradle preia toate fișierele sursă și aplică instrumentele adecvate, de exemplu, convertește fișierele java în fișiere dex și le comprimă într-un singur fișier cunoscut sub numele de apk care este de fapt folosit. Există doua tipuri de scripturi build.gradle, acestea fiind Top-level build.gradle si Module-level build.gradle. În această aplicație este folosit Module-level build.gradle.

Module-level build.gradle este situat în directorul modul al proiectului. Scriptul Gradle este locul în care sunt definite toate dependințele și sunt declarate versiunile sdk. Dependințele sunt un aspect foarte important al proiectului deoarece permit includerea librăriilor externe în program. Acestea pot fi de tipul implementation, api, compileOnly, runtimeOnly, annotationProcessor, lintChecks, lintPublish. În interiorul aplicației se folosesc urmatoarele dependințe de tipul implementation:

- 'com.google.android.gms:play-services-location:18.0.0', acesta ne permite sa folosim locația utilizatorului
- 'com.google.android.gms:play-services-maps:17.0.0', are scopul de a implementa google maps cu toate funcțiile sale ce permit modificarea hartii inițiale
- 'com.google.code.gson:gson:2.8.6' ne permite să folosim operații ce includ fișiere de tip Json
- 'com.squareup.picasso:picasso:2.71828' este folosit la crearea unei interfețe mai plăcute
- 'com.squareup.retrofit2:adapter-rxjava2:2.9.0' ne permite să manipulăm linkuri și să extragem date din surse de tip HTTP
- 'com.squareup.retrofit2:converter-gson:2.3.0' are scop similar ca cel de mai sus doar că este folosit la fișierele Json

- 'com.google.firebase:firebase-database:20.0.0' ne permite să folosim o bază de date externă realizată în Firebase

Android Studio acceptă limbaje de programare similare cu IntelliJ cum ar fi Java, C ++ și multe altele cu extensii, cum ar fi Go. Android Studio 3.0 sau o versiune ulterioară acceptă Kotlin și toate caracteristicile de limbaj Java 7 și un subset de caracteristici de limbaj Java 8 care variază în funcție de versiune.

Java este un limbaj de programare orientat pe obiecte. Programele sau aplicațiile dezvoltate în acesta se vor executa folosind o mașină virtuală Java prin care putem rula același program pe mai multe platforme și dispozitive. Java este utilizat în cea mai mare parte pentru aplicații independente sau dezvoltare back-end.

Limbajul împrumută o mare parte din sintaxă de la C și C++, dar are un model al obiectelor mai simplu. Toate fișierele în java sunt fie de tip clasă sau de tip interfață, depinzând de funcțile sau întrebuințarile care sunt definite în interiorul acestora. Clasele și interfețele sunt găsite în interiorul unui package iar un proiect poate conține mai multe elemente de tip package.

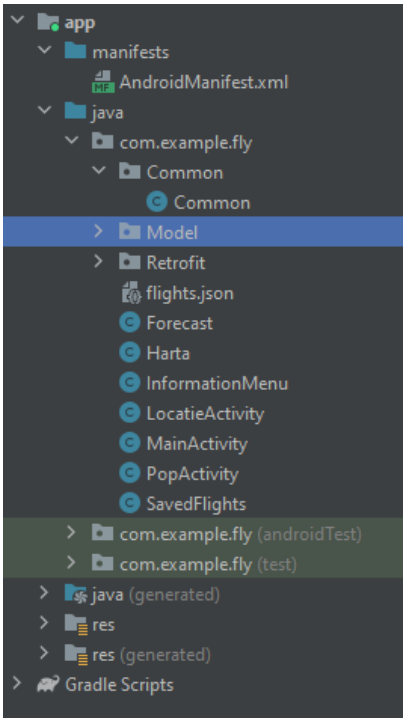


Figura 4.1: Structura aplicației

În pachetul com.example.fly avem clasele Harta, LocatieActivity, MainActivity, PopActivity, Forecast, InformationMenu care reprezintă ecranele cu care utilizatorul va interacționa și pachetele common, unde este salvată cheia necesară accesări datelor meteorologice , model, unde se află toate clasele care reprezintă obiecte folosite la datele meteorologice și datele despre o anumită locație, și Retrofit care conține o clasă și o interfață, amblele necesare la formarea linkului pentru extragerea acestora.

```
package com.example.fly.Model;

public class locatie implements java.io.Serializable{

    String nume;
    double longitudine;
    double latitudine;
    int aterizare;
    int decolare;
    String vant;
    int id;

    public locatie() {
    }

    public locatie(locatie original)
    {...}

    public String getNume() { return nume; }

    public void setNume(String nume) { this.nume = nume; }

    public double getLongitudine() { return longitudine; }
```

Figura 4.2: Exemplu de clasă din interiorul fișierului model

În figura 4.2 este definiă clasa locație care este folosită la stocarea informaților din firebase și are atribute specifice, funcții de creare, get și set (folosite la returnarea unei valori respectiv schibarea acesteia).

4.2 Meniul principal

În progamarea pe dispozitive mobile majoritatea ecranelor cu care un utilizator interacționează sunt scrise ca o Activitate sau ca un Fragment.

O activitatate reprezintă un ecran unic prezent în aplicație. Aproape toate activitățile interacționează cu utilizatorul, astfel încât clasa Activitate se ocupă de crearea unei ferestre unde interfața poate fi plasată cu setContentView (View) care îi va oferi aspectul definit în XML. În timp ce activitățile sunt adesea prezentate utilizatorului ca ferestre cu ecran complet, ele pot fi folosite și în alte moduri: ca ferestre floating (ca cea prezentă în figura 3.3 b), sau încorporate în alte ferestre. Există două metode pe care le vor implementa aproape toate subclasele din activitate:

onCreate (Bundle) este locul unde se inițializează activitatea. Aici este apelat de obicei setContentView (int) cu o resursă de aspect care va defini interfața de utilizare și findViewById (int) pentru a prelua widgeturile (butoane, spațiul pentru text, spațiul pentru imagine, etc.) din aceea interfață.

onPause () este partea care intervine când utilizatorul întrerupe interacțiunea cu activitatea. În această stare, activitatea este încă vizibilă pe ecran.

Prima activitate carea se execută în interiorul aplicației poartă numele de MainActivity și reprezintă meniul principal.

Interfața este realizată în XML. Un Layout definește structura unei interfețe cu utilizatorul din interiorul aplicației. Toate elementele din Layout sunt construite folosind o ierarhie a obiectelor View și ViewGroup. Un view reprezintă de obicei ceva pe care utilizatorul îl poate vedea și cu care poate interacționa în timp ce un ViewGroup este un container invizibil care definește structura de aspect pentru un View și alte obiecte de tip ViewGroup.

Obiectele de tip View sunt denumite de obicei „widget-uri" și pot fi una dintre numeroasele subclase, cum ar fi Button sau TextView. Obiectele ViewGroup sunt denumite de obicei „Layout" și pot fi unul dintre multele tipuri care oferă o structură de aspect diferită, cum ar fi LinearLayout sau ConstraintLayout.

În meniul pricipal folosim LinearLayout, ImageView, TextView.

```
<LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginBottom="70dp"
            android:layout_gravity="center"
            android:orientation="horizontal">
```

În acestă stuctură de cod este prezentat un exemplu de LinearLayout care conține urmatorele variabile: width și height care reprezintă mărimea spațiului, fiind setat la wrap content acesta își va schimba mărimea în funcție de dimenisunea obictelor

pe care le conține, marginBottom care reprezintă marimea marginii aflată în partea de jos, gravity care are rolul de a poziționa obiectele din interior, fiind setat la center acestea vor fi puse în mijloc, și orientation care reprezintă direcția unde se vor amplasa obiectele, fiind setat la horizontal însemna ca se vor pune pe orizontală, în ordinea în care sunt definite.

```
<ImageView
            android:id="@+id/instruciuni"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginBottom="4dp"
            android:src="@drawable/openbook3" />
```

La declararea acestei imagini avem doua variabile diferite acestea fiind id, care ne va permite să lucrăm cu widget-ul în partea de java, referindu-ne la el cu numele trecut, și src care reprezintă o adresă din interiorul aplicației unde se găsește imaginea care trebuie afișată.

```
<TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Paragliding Instructions"
            android:textColor="#0E1838"
            android:textSize="16sp" />
```

Pentru afișarea textului pe ecran este necesar un textview care conține textcolor, unde este trecut un cod ce reprezintă o anumită culoare, în acest caz negru, textsize ce reprezintă mărimea textului și text unde este trecut textul care trebuie să fie afișat pe ecran.

Tot aspectul este realizat utilizând aceste 3 elemente, folosind variabile diferite în funcție de necesitate.

Partea de animație este realizată astfel:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_interpolator"
    >
    <translate
        android:fromYDelta="0%"
        android:toYDelta="30%"
        android:duration="800"
        />
</set>
```

from y reprezintă poziția inițiala a obiectului în spațiu (y fiind direcția verticală și x cea orizontală) iar to y este direcția în care se va mișca obictul care va avea atribuită animația. duration reprezintă viteza cu care va avea loc aceasta.

Declararea variabilelor din MainActivity

```
ImageView bgapp;
Animation frombottom;
LinearLayout appname, menu;
ImageView locatie, zbor, instruciuni;
```

ImageView reprezintă variabilele care o să primească id-ul unei imagini, animation reprezintă animația definită în xml și LinearLayout sunt containerele ce conțin grupări de TextView și ImageView.

```
zbor = (ImageView) findViewById(R.id.zbor);
```

Acesta linie de cod este folosită pentru a atribui o varibilă din xml la o varibilă din java, folosind un id definit anterior în xml.

```
zbor.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, Harta.class);

                startActivity(intent);
            }
        });
```

Folosind această structură se adaugă funcționalitate de click unui element definit în xml. În acest caz la apăsarea pictogramei pentru zbor, va fi inițializat un obiect de tip intent, care are rolul de a deschide un nou ecran utilizatorului.

4.3 Afișarea locaților de parapantism

Fragmentele permit împarțirea interfeței în bucăți. Activitățile sunt un loc ideal pentru a pune elemente globale în jurul interfeței de utilizare a aplicației. Fragmentele sunt mai potrivite pentru a defini și gestiona interfața de utilizare a unui singur ecran sau a unei porțiuni de ecran.

Un fragment reprezintă o porțiune reutilizabilă din interfața de utilizare a aplicației. Un fragment își definește și gestionează propriul aspect și are propriul ciclu

de viață. Fragmentele nu pot exista singure - trebuie găzduite de o activitate sau de un alt fragment. Ierarhia de vizualizare a fragmentului devine parte din sau se atașează la ierarhia de vizualizare a gazdei.

Ecranul pentru google maps este definit ca un fragment iar acesta este realizat în mai mulți pași, primul fiind adaugarea google play services.

Google play services furnizează un set larg de SDK-uri pe Android pentru a ajuta la crearea unei aplicații. Un kit de dezvoltare software (SDK) este o colecție de instrumente de dezvoltare software într-un singur pachet instalabil care facilitează crearea de aplicații, având un compilator, un depanator și poate un cadru software. În mod normal, acestea sunt specifice unei combinații de platforme hardware și sisteme de operare, și se folosesc la crearea aplicațiilor cu funcționalități avansate. Un SDK poate lua forma unor application programming interfaces(API-uri) sub formă de biblioteci reutilizabile utilizate pentru interfața cu un anumit limbaj de programare sau pot lua forma unor instrumente specifice hardware-ului care pot comunica cu un anumit sistem încorporat. Instrumentele comune includ facilități de depanare și alte utilități, adesea prezentate într-un mediu de dezvoltare integrat (IDE). SDK-urile pot include bucați de software și note tehnice împreună cu documentație și tutoriale pentru a ajuta la clarificarea punctelor făcute de materialul de referință principal. Din google play services este utilizat google maps SDK.

Pentru a folosi google maps SDK este nevoie de un API-key care va fi obținut de la Google cloud platform. O cheie API este un cod care este transmis de aplicațiile computerizate. Programul sau aplicația apelează apoi API-ul pentru a identifica utilizatorul, dezvoltatorul sau programul. Cheia este stocată într-un fișier xml numit mapapi.xml iar mai departe este preluată în manifest și transmisă servicilor google, la final permițând utilizarea tuturor funcționalităților din google maps SDK.

Stocharea cheii într-un fișier xml

```
<resources>

    <string name="map_key" translatable="false">
        AIzaSyApSYZOTgJcxYoTBHhu9MEehfeFIGYaQ_U
    </string>

</resources>
```

Preluarea acesteia în manifest

```
    <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/map_key" />
```

O caracteristică foarte importantă prezentă în google maps sunt Markers.

Markers identifică locațiile de pe hartă. Markers implicit folosește o pictogramă standard, comună aspectului Google Maps. Este posibilă schimbarea culorii pictogramei, imaginea sau punctul de ancorare prin API. Informațile care vor fi atribuite unui marker sunt stocate într-o bază de date externă realizată în firebase.

Firebase este o platformă folosită la construirea aplicaților Web, Android și IOS care oferă următoarele facilități fară partea complicată de backend:

- Real-time Database - Firebase acceptă date JSON și toți utilizatorii conectați la aceasta primesc actualizări live după fiecare modificare
- Autentificare - Se pot folosi autentificări anonime
- Hosting - Aplicațiile pot fi implementate prin conexiune securizată la servere Firebase

Baza de date folosită în această aplicație conține mai multe obiecte de tip locație, care după preluare sunt atribuite unui marker.

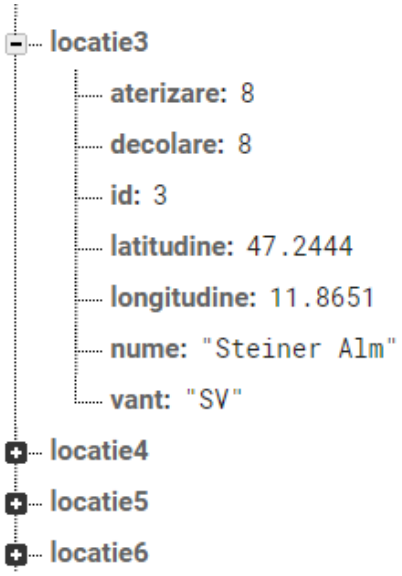


Figura 4.3: Structura bazei de date

Un obiect de tip locație din bază de date conține latitudine și longitudine, care au rolul de a plasa markerul pe hartă la coordonatele respective, aterizare și decolare care reprezintă condiția zoneler desemnate acestora, vânt unde este trecută direcția necesară a vantului pentru a putea fi practicat zborul, id și nume care sunt folosite la identificarea mai ușoară a locației.

Conexiunea cu baza de date se face astfel: Se inițializează o variabilă de tip FirebaseDatabase care conține un link către baza de date și una de tip DatabaseReference care va fi setată la coloana "locație" a bazei de date. "Markers" sunt adăugate în interiorul funcției onMapReady(GoogleMap googleMap) unde folosim DataSnapshot pentru a extrage fiecare element al coloanei, iar pentru fiecare element plasăm un marker la locația corespunzătoare folosind map.addMarker().

```
FirebaseDatabase database =
FirebaseDatabase.getInstance
        ("https://flymaster-default-rtdb.europe-west1.firebasedatabase.app/");
DatabaseReference myRef = database.getReference("locatie");

public void onMapReady(GoogleMap googleMap) {
    map = googleMap;
    map.setMapType(GoogleMap.MAP_TYPE_HYBRID);

    myRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NotNull DataSnapshot dataSnapshot) {

            Iterable<DataSnapshot> children = dataSnapshot.getChildren();
            for (DataSnapshot child : children) {

                Loc = child.getValue(com.example.fly.Model.locatie.class);

                LatLng latLng = new LatLng(Loc.getLatitudine(),
                Loc.getLongitudine());
                map.addMarker(new MarkerOptions()
                .position(latLng)
                .title(Loc.getNume()))
                .setTag(Loc);

                locatieList.add(Loc);
            }
        }
```

Pentru a obține comportamentul dorit de la un Marker, este adaugat un eveniment ce are loc atunci când utilizatorul va apăsa pe unul dintre ele. În această aplicație markerele la atingere vor deschide o fereastră de tip "PopUp" care va conține informațile meteo prezente la locația respectivă. Variabila de tip Intent este inițializată cu un ecran prezent în aplicatie, iar startActivity(Intent) va deschide ecranul atribuit. putExtra va trimite variabilele respective catre noul ecran deschis sub numele trecut între "".

```
map.setOnMarkerClickListener(new GoogleMap.OnMarkerClickListener() {
        @Override
        public boolean onMarkerClick(Marker marker) {

            if(!marker.getTitle().equals("ME")) {
                Intent i = new Intent(Harta.this, PopActivity.class);
                LatLng position = marker.getPosition();
                i.putExtra("lat", position.latitude);
                i.putExtra("lon", position.longitude);
                i.putExtra("nume", marker.getTitle());
```

```
                locatie data = (locatie) marker.getTag();
                i.putExtra("Data", data);


                startActivity(i);
            }
            return false;
        }
    });
```

4.4 Datele Meteorologice

Informațiile meteorologice sunt importate din surse externe. Pentru a face rost de acestea este nevoie de un API-key catre https://openweathermap.org unde sunt stocate informațiile meteo globale. După obținerea accesului catre baza de date trebuie construit linkul catre informațile solicitate de utilizator. Obținerea acestora este efectută folosind Retrofit

Retrofit este un client REST pentru Java și Android care face relativ ușoară extragerea datelor din fișiere de format JSON (sau alte date structurate) printr-un serviciu web bazat pe REST. Retrofit folosește biblioteca OkHttp pentru solicitări HTTP. Acesta necesită o clasa numita RetrofitClient și o interfață, numită în acest caz IOpenWetherMap, pentru a funcționa. În interiorul interferței este definit modelul și parametri care pot fi inlocuiți din linkul către datele meteo.

```
    @GET("weather")
Observable<WeatherResult>
getWeatherByLatLng(@Query("lat") String lat,
                   @Query("lon") String lng,
                   @Query("appid") String appid,
                   @Query("units") String unit);
```

În clasa PopActivity accesarea datelor meteo se face folosind longitudinea (lat) și latitudinea (lat) transmisă de marker la apăsare.

```
compositeDisposable.add((Disposable) mService
.getWeatherByLatLng(String.valueOf(lat),
                String.valueOf(lon),
                Common.APP_ID,
                "metric")
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(new Consumer<WeatherResult>()
```

Exemplu de link format în urma solicitarii:

```
http://api.openweathermap.org/data/2.5/weather?lat=35&lon=139
&appid=71e0938c71df7db893da4b08058b64c9&units=metric
```

Dupa construirea linkului acesta va fi accesat si va returna un fisier de tip JSON

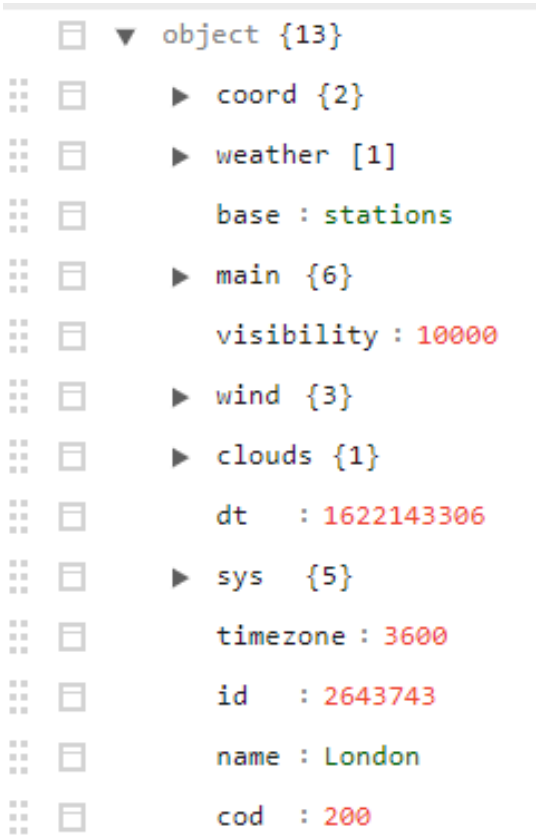


Figura 4.4: Structura fișierului JSON

Ca aplicația să poată folosi datele extrase sunt create clase corespunzătoare datelor din fișier: wind, care va avea doua atribute, main, care va conține atributele legate de temperatură, presiune, umiditate, clouds etc. După extragere datele necesare sunt afișate folosind .setText() la elemntele de tip TextView corespunzătoare.

```
public void accept(WeatherResult weatherResult) throws Exception {

    Picasso.get()
    .load(new StringBuilder("https://openweathermap.org/img/w/")
    .append(weatherResult.getWeather()
    .get(0).getIcon())
    .append(".png").toString())
    .into(imagine);
```

```
        temperatura.setText(new StringBuilder(String.valueOf(round(weatherResult
        .getMain()
        .getTemp())))
        .append("°C")
        .toString() );

        descriere.setText(weatherResult.getWeather()
        .get(0)
        .getDescription());

        vitezavant.setText(new StringBuilder(String.valueOf(weatherResult
        .getWind()
        .getSpeed()))
        .append("m/s"));

        umiditate.setText(new StringBuilder(String.valueOf(weatherResult
        .getMain()
        .getHumidity()))
        .append("%"));

        nume.setText(NumeLocatie);
        vant=weatherResult.getWind().getDeg();
        vantSpeed=weatherResult.getWind().getSpeed();
        setWeather();


    }
```

Realizarea aspectului ferestrei cu informații actuale se face astfel:

```
 DisplayMetrics dm = new DisplayMetrics();
        getWindowManager().getDefaultDisplay().getRealMetrics(dm);
        int width =dm.widthPixels;
        int height= dm.heightPixels;

        getWindow().setLayout((int)(width*.7),(int)(height*.5));

        WindowManager.LayoutParams params = getWindow().getAttributes();
        params.gravity = Gravity.CENTER;
        params.x = 0;
        params.y = -20;
        getWindow().setAttributes(params);
```

Este creată o activitate normală asupra căruia au loc modificări de mărime și poziție. Este creată o variabilă de tip DisplayMetrics cu informațile de poziție ale ecranului, de unde extragem lațimea, care va fi redusă la 70% din valoarea inițială,

și înalțimea, care va fi redusă la 50% din valoarea inițială. Folosind variabila param care este de tip WindowManager.LayoutParams se centrează ferestra folsind Gravity.CENTER și se fac mici ajustări folosind axa de abscisă și ordonată.

Conexiunea către partea de forcast se face într-un mod similar ca cel al procnozei curente doar că de acestă dată sunt returnate 40 de obiecte de tip WeatherForecast care se vor reține folosind o listă definită pentru elementele de acest tip. Se extrag după strict informațile necesare folosind o clasă numită dataW, iar folosind funcția setWeather(); sunt setate elementele de tip TextView corespunzător

```
private void getForecastWeather()
    {
    compositeDisposable.add((Disposable) mService
    .getForecastWeatherByLatLng(String.valueOf(lat),
    String.valueOf(lon),
    Common.APP_ID,
    "metric")
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Consumer<WeatherForecastResult>() {

   @Override
   public void accept(WeatherForecastResult weatherForecastResult)
   throws Exception {

       for(int i=0;i<40;i++)

       {

           dataW[i].setClouds(weatherForecastResult
           .getList().get(i).getClouds().getAll());

           dataW[i].setDate(weatherForecastResult
           .getList().get(i).getDt_txt());

           dataW[i].setDescription(weatherForecastResult
           .getList().get(i).getWeather().get(0).getDescription());

           dataW[i].setHumidity(weatherForecastResult
           .getList().get(i).getMain().getHumidity());

           dataW[i].setIcon(weatherForecastResult
           .getList().get(i).getWeather().get(0).getIcon());

           dataW[i].setTemp(weatherForecastResult
           .getList().get(i).getMain().getTemp());
```

```
            dataW[i].setWindDeg(weatherForecastResult
            .getList().get(i).getWind().getDeg());

            dataW[i].setWindSpeed(weatherForecastResult
            .getList().get(i).getWind().getSpeed());

        }
        setWeather();

    }
```

Definirea cartonașelor pentru ora selectată se realizează prin modificarea unei forme de tip rectangele. Acesteia îi sunt rotunjite colțurile și îi este oferită o culoare de tip gradient.

```
    <?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
android:shape="rectangle">

    <gradient android:startColor="#FF7074"
    android:endColor="#CB05DD"
    android:angle="360"/>

    <corners android:topLeftRadius="20dp"
    android:topRightRadius="40dp"
    android:bottomRightRadius="20dp"
    android:bottomLeftRadius="40dp"/>
</shape>
```

Definirea formei cartonașelor pentru ora neselectată se face în stil similar doar ca de această dată sunt suprapuse doua forme de tip rectangle pentru a putea oferi o culoare diferită conturului.

```
    <item>
        <shape android:shape="rectangle">
            <gradient
                android:angle="45"
                android:endColor="#CB05DD"
                android:startColor="#FF7074" />
            <corners android:topLeftRadius="20dp"
            android:topRightRadius="40dp"
            android:bottomRightRadius="20dp"
            android:bottomLeftRadius="40dp"/>
```

```
        <padding
            android:bottom="4dp"
            android:left="4dp"
            android:right="4dp"
            android:top="4dp" />

    </shape>
</item>
<item>
    <shape android:shape="rectangle">
        <solid android:color="#FFFF" />
        <corners android:topLeftRadius="20dp"
        android:topRightRadius="40dp"
        android:bottomRightRadius="20dp"
        android:bottomLeftRadius="40dp"/>

    </shape>
</item>
```

4.5 Ajutor de zbor

O funcționalitate importantă a aplicației este determinarea locației curente a utilizatorului în coordonate geografice. Pentru reazlizarea aceasteia se folosește location SDK prezent în google play services. Ca aplicația să poată utiliza gps-ul telefonului este nevoie ca utilizatorul să ofere permisiunea de utilizare a locației curente. Această solicitare este declarată în manifest și va avea un id unic în telefon.

```
private void getLocation() {
    fusedLocationProviderClient.getLastLocation()
    .addOnCompleteListener(new OnCompleteListener<Location>() {
        @Override
        public void onComplete(@NonNull Task<Location> task) {
            location = task.getResult();
            /**se verifica daca locatia exista **/
            if (location != null) {
        try {

            geocoder = new Geocoder(LocatieActivity.this,
            Locale.getDefault());

            addresses = geocoder.getFromLocation(location.getLatitude(),
            location.getLongitude(), 1);

                    latv = addresses.get(0).getLatitude();
                    stringdouble = String.format("%.7f",latv);
```

```
                    lati.setText(stringdouble);

                    longiv = addresses.get(0).getLongitude();
                    stringdouble = String.format("%.7f",longiv);
                    longi.setText(stringdouble);

                    altv = ceil(location.getAltitude());
                    stringdouble = Double.toString(altv);
                    alt.setText(stringdouble);

                    String cityName = addresses.get(0)
                    .getLocality();

                    String stateName = addresses.get(0)
                    .getAdminArea();

                    /** se apeleaza functia de update la locatie **/
                    requestNewLocationData();

                }
```

După obținerea permisiuni aplicația va determina locația curentă folosind funcția getLocation(). În interiorul acesteia este folosită o variabilă de tip geocoder care va conține latitutinea, longitudinea și altitudinea. Aceste informații sunt transmise utlizatorului folosind elementele de tip TextView corespunzătoare.

După aflarea locației inițiale, aplicația va continua să determine noile coordonate o data la 2 secunde folosind requestNewLocationData()

```
    private void requestNewLocationData() {

        LocationRequest mLocationRequest = new LocationRequest();
        mLocationRequest.setPriority(LocationRequest
        .PRIORITY_HIGH_ACCURACY);
        mLocationRequest.setInterval(2000);
        mLocationRequest.setFastestInterval(2000);

        fusedLocationProviderClient = LocationServices
        .getFusedLocationProviderClient(this);
        fusedLocationProviderClient.requestLocationUpdates(
                mLocationRequest, mLocationCallback,
                Looper.myLooper()
        );

    }
```

Compasul rezprezintă alt instrument foarte important când vine vorba de zbor. Acesta ia în considerare senzorul de accelerație și cel al câmpului magnetic iar

la fiecare mișcare va roti imaginea atribuită compasului în una din cele 360 de poziții posibile.

```
public void onSensorChanged(SensorEvent event) {

   final float alpha=0.97f;
   synchronized (this){

       if(event.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
       {
           mGravity[0] = alpha*mGravity[0]+(1-alpha)*event.values[0];
           mGravity[1] = alpha*mGravity[1]+(1-alpha)*event.values[1];
           mGravity[2] = alpha*mGravity[2]+(1-alpha)*event.values[2];
       }
       if(event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD)
       {
           mGeomantic[0] = alpha*mGeomantic[0]+(1-alpha)*event.values[0];
           mGeomantic[1] = alpha*mGeomantic[1]+(1-alpha)*event.values[1];
           mGeomantic[2] = alpha*mGeomantic[2]+(1-alpha)*event.values[2];
       }

       float R[] = new float[9];
       float I[] = new float[9];
       boolean succes = SensorManager.getRotationMatrix(R,I
       ,mGravity,mGeomantic);
       if(succes)
       {
           float orientation[] = new float[3];
           SensorManager.getOrientation(R,orientation);
           azimuth = (float)Math.toDegrees(orientation[0]);
           azimuth = (azimuth+360)%360;
           Animation anim = new RotateAnimation(-currectAzimuth,-azimuth
           ,Animation.RELATIVE_TO_SELF,0.5f
           ,Animation.RELATIVE_TO_SELF,0.5f);
           currectAzimuth=azimuth;
           anim.setDuration(100);
           anim.setRepeatCount(0);
           anim.setFillAfter(true);

           compass.startAnimation(anim);

       }
```

Emitera de semnale audio în funcție de schimbarea altitudinii se realizeză prin verificarea altitudinii precedente cu cea actuală și este emis un sunet corespunzător.

```
if(altv > mLastLocation.getAltitude() && ok==1
&& altv-mLastLocation.getAltitude()>1)
       {
           descending.start();
           maxalt=altv;
       }
       else
       {
           if(altv < mLastLocation.getAltitude()
           && ok==1 && mLastLocation.getAltitude()-altv>1)
           {
               ascending.start();
               maxalt=mLastLocation.getAltitude();
           }
       }
```

Capitolul 5

Concluzii și direcții vitoare

Analiza realizată în capitolele anterioare demonstrează că aplicația reușește să ofere o experiență mai bună parapantiștilor.

Cu toate acestea aplicația prezintă o anumită problemă. Condițile zonelor de decolare și aterizare se pot schimba în timp iar notele oferite trebuie modificate de cel care deține baza de date, facând impractic acest lucru. O soluție la această problemă ar fi adăugarea unui sistem de conturi unde fiecare utilizator va putea lăsa o recenzie unei zone împreună cu o notă, asigurând ca în acest fel este afișată o informație mai aproape de adevăr.

Un alt neajuns îl prezintă marimea relativ mică a numărului de zone de zbor afișate. Este destul de dificilă adăugarea acestora deoarece trebuie determinată direcția în care trebuie să bată vantul pentru zbor, aceasta fiind de multe ori obținută din forumuri sau alte surse. Faptul ca este determinată condiția de zbor în acest fel poate duce la afișarea unor informații incorecte.

Cea mai dificilă parte a reprezentat-o importarea datelor meteo deoarece a fost necesar un ansamblu de clase destul de mare, ducând de multe ori la erori neprevăzute, iar documentația pusă la dispoziție de către openweathermap.org nu oferă întotdeana explicații clare.

O funcționalitate care ar fi fost utilă pentru pentru această aplicație o reprezintă salvarea traseului parcurs în zbor împreună cu date cum ar fi viteza medie, altitudinea maximă, zona unde s-a folosit un curent termic etc. Toate acestea ar fi putu oferi o idee mai bună legată de performanța cu care zboară cel care o utilizează.

Adaugarea unui mod de cautare a unei locații ar putea ușura experiența utilizatorului.

Pasul final îl reprezintă încarcarea aplicației pe Google Play.

Capitolul 6

Bibliografie

https://developers.google.com/maps/documentation/android-sdk/start
https://openweathermap.org/current
https://www.tutorialspoint.com/android/android_studio.htm
https://www.paraglidingforum.com
https://developers.google.com/android/guides/setup