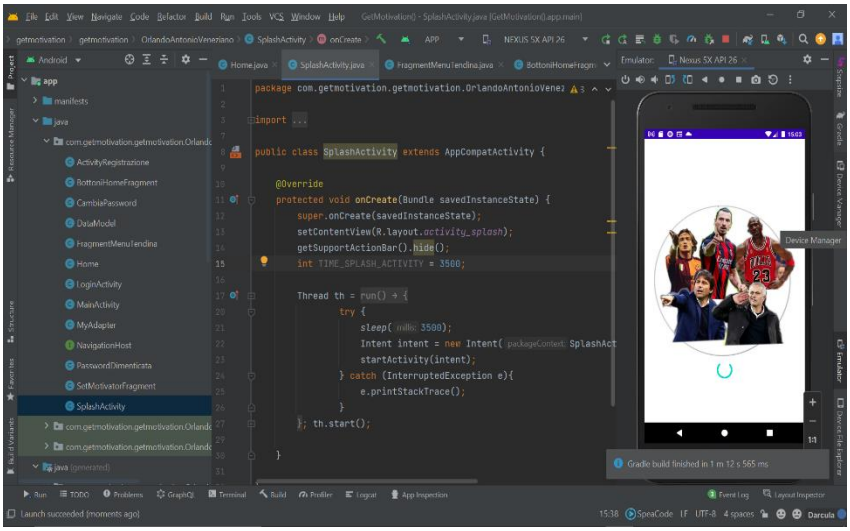
PROGETTO ANDROID STUDIO
ORLANDO ANTONIO VENEZIANO

GetMotivation()

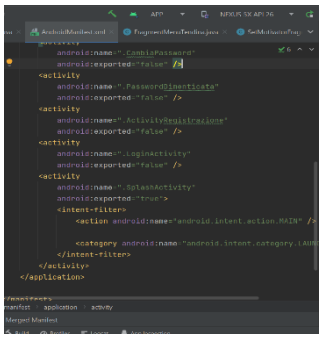
ILLUSTRAZIONE PROGETTO

SplashActivity ->

A questa prima activity ho legato un file xml, che contiene il logo dell'applicazione ed una progress bar. Attraverso il metodo getSupportActionBar(), della classe AppCompatActivity, ho recuperato l'istanza della classe ActionBar e attraverso questa è stato possibile usare il metodo hide(), per nascondere in presenza della SplashActivity la barra di navigazione. In seguito, ho usato, all'interno di un thread secondario, il metodo sleep all'interno di un blocco try-catch, per avviare la LoginActivity con un intent esplicito, passati 3.5 secondi, da quando è stato avviato il thread "th".

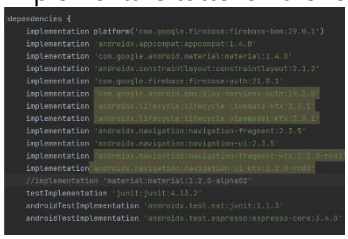


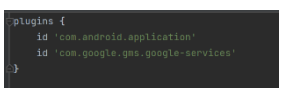
Ho trovato la SlashActivity didatticamente molto utile, perché avendola creata in un secondo momento ho dovuto modificare manualmente il Manifest dell'applicazione e così capire meglio il funzionamento della registrazione di tutte le activity. Qui a destra, ho riportato una piccola illustrazione del manifest.



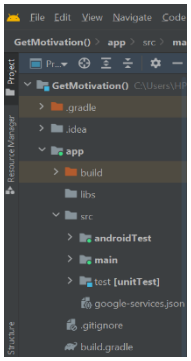
LoginActivity, ActivityRegistrazione, PasswordDimenticata, CambiaPassword(fragment), tasto di Logout->

Per implementare questi aspetti ho fatto uso di Firebase. In una prima fase, ho avuto difficoltà ad implementare tutte le librerie e plugin necessari al gradle, poiché l'implementazione automatica col tool di Android Studio non ha funzionato. Una volta importato tutto ciò che risultava necessario nel gradle, ho dovuto importare un file json, scaricato dalla console del sito di FireBase, all'interno dell'applicazione. A sinistra, sono riportate le diverse librerie che sono state implementate e più in basso, l'implementazione del plugin di google services.

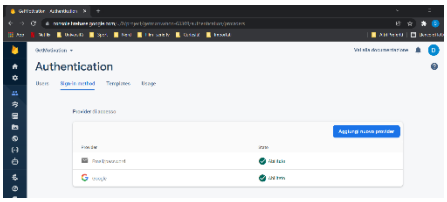


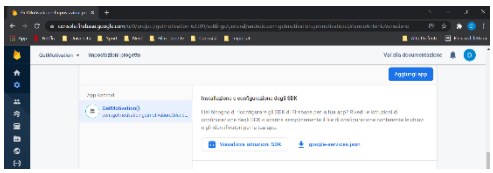


A destra, è riportato il percorso nel quale bisogna inserire il file json, all'interno dell'applicazione (bisogna selezionare la schermata del progetto ed inserire il file nella cartella src). Prima di poter scaricare il file json, bisogna entrare sulla pagina internet di Firebase e creare un nuovo progetto. Dopo si verrà guidati dal sito stesso, nell'esecuzione dei passaggi successivi. Bisognerà selezionare il tipo di database che si vuole che google gestisca per noi e, nel caso di un database per autenticazione degli utenti, bisognerà selezionare i provider che google dovrà gestire per noi.



Senza quest'ultimo passaggio, il database non risulterebbe abilitato. Qui a destra, è riportato l'url al quale accedere (all'interno del proprio progetto) all'abilitazione dei diversi provider.





A sinistra, ho riportato la schermata che viene raffigurata nel sito di Firebase per eseguire il download del file json. Presi tutti questi accorgimenti, si può passare all'implementazione del codice.

IMPLEMENTAZIONE CODICE:

Nella classe LoginActivity ed in tutte le classi in cui vengono gestiti i dati del provider "email", ho utilizzato un'istanza della classe FirebaseAuth, per poter accedere a tutti i metodi di questa classe. Particolare rilievo va dato ad alcuni metodi:

```
-FirebaseAuth.getInstance();
```

con questo semplice metodo è possibile ottenere un'istanza della classe di interesse. Fatto questo si può procedere con l'uso degli altri metodi.

```
-   if (authLogin.getCurrentUser() != null)
        startActivity(new Intent(LoginActivity.this, MainActivity.class));
    //qua ho utilizzato l'istanza della classe FirebaseAuth, per valutare se
l'utente era già stato loggato, se con un precedente uso dell'app, era già stata
fatta l'autenticazione, allora all'apertura l'utente viene subito mandato alla
MainActivity
}
```

Qui, come descritto nel commento, ho usato l'istanza ottenuta col metodo descritto in precedenza, per controllare se l'utente fosse già loggato ed in tal caso andare subito all'Activity successiva.

```
-authLogin.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
```

In queste righe, ho copiato solo la parte del codice più significativa.

Il metodo signInWithEmailAndPassword è il metodo più importante di questa classe, grazie al quale Firebase gestirà per noi le credenziali dell'utente. Nel momento in cui viene scritta questa istruzione, viene istanziata una classe anonima, che implementa un handler (OnCompleteListener<>()) che come si vede dal codice, prende in ingresso i risultati della verifica svolta da Firebase e, facendo override del metodo associato a questa interfaccia (con in ingresso i parametri di interesse), è possibile gestire il risultato che, se sarà corretto, porterà all'Activity successiva (per il modo in cui io ho implementato il metodo).

```
-private void signUpUser(String email, String password) {
    auth.createUserWithEmailAndPassword(email,password)
            .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if(!task.isSuccessful())
                    {
```

In modo del tutto analogo, con un metodo simile nell'activity di registrazione, ho creato un metodo, che facendo uso del metodo createUserWithEmailAndPassword e di una classe anonima che implementa la stessa interfaccia usata in precedenza, crea il profilo dell'utente su Firebase e restituisce dei Toast, in relazione all'esito dell'operazione.

In fase di inserimento delle password, ho implementato anche una classe trovata su internet, per gestire la conversione di ogni carattere inserito nell'editText della password al simbolo di un asterisco.

```
private void resetPassword(final String email) {
    auth.sendPasswordResetEmail(email)
            .addOnCompleteListener(this, new OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
```

```
                    if(task.isSuccessful())
                    {
```

Lo stesso ragionamento è stato ripetuto anche nell'Activity PasswordDimenticata, questa volta con un metodo che prende il nome di sendPasswordResetEmail, che permetterà, dalla mail inserita nell'editText, di resettare i dati sulla password per l'account associato a quella mail.

```
FirebaseUser user = auth.getCurrentUser();
user.updatePassword(newPassword).addOnCompleteListener(getActivity(), new
OnCompleteListener<Void>() {
```

Lo stesso meccanismo è stato ripetuto in un fragment, per cambiare la password.

```
auth.signOut();
if(auth.getCurrentUser() == null)
{
```

Molto più interessante è il fragment dal quale ho gestito il Logout. Con il metodo signOut() viene annullata l'istanza del CurrentUser ed in seguito a questo, controllando se l'operazione è andata a buon fine, si può eseguire un intent, per tornare alla schermata di Login. (Ho sfruttato questo momento per fare un'operazione di edit sulle sharedPreferences, istanziando un Editor e mettendo al valore "false" i valori associati alle chiavi di alcuni interruttori, dei quali parlerò, in modo più approfondito, in seguito)

```
AlertDialog.Builder builder = new AlertDialog.Builder(getContext());
    builder.setMessage("Sei sicuro di voler uscire?").setPositiveButton("Sì",
new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialogInterface, int i) {
            logoutUser();

        }
    }).setNegativeButton("No", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialogInterface, int i) {
            startActivity(new Intent(getActivity(),Home.class));
        }
    });
    AlertDialog alertDialog = builder.create();
    alertDialog.show();
}
```

Un'altra operazione nuova ed interessante, che ho voluto fare in fase di logout, è stata quella di costruire un AlertDialog attraverso un Builder, per poi settare un messaggio positivo ed uno negativo ed associare degli intent alle diverse operazioni.

ON BACK PRESSED:

In molte activity, come suggerito a lezione, ho fatto override del metodo onBackPressed, per mandare dei semplici intent, oppure evitare che fosse eseguito il super del metodo. Questo è un esempio:

```
@Override
public void onBackPressed() {
    Intent a = new Intent(PasswordDimenticata.this, LoginActivity.class);
    //setAuthLoginNull();
    startActivity(a);
}
```

Particolare attenzione l'ho posta, in relazione alla gestione di questo metodo, in fase di Login e nell'activity Home e quando venivano visualizzati i risultati, per non tornare alla schermata precedente. Ho prestato attenzione soprattutto al Logout, descritto sopra, perché va evidenziato che il metodo onBackPressed è un metodo dell'activity, però quando il Context diventa l'AlertDialog, allora non funzionerà più l'onBackPressed, del quale è stato fatto override nell'activity. Premendo back in corrispondenza dell'AlertDialog, si resterà nel fragment di Logout. A destra, si può vedere un'immagine del tasto, che ho inserito, per rendere meno spiacevole il presentarsi di questa problematica. In questo caso, per affrontare il problema in maniera semplice ed evitare che l'utente sia davanti ad una schermata bianca senza opzioni, se non quella di premere per la seconda volta il tasto back, ho aggiunto un tasto col messaggio "torna alla home", in modo da facilitare l'interazione dell'utente col dispositivo, in presenza di questa problematica.

MAIN ACTIVITY E SCELTA "MOTIVATORI"

Dopo le schermate di Login, non ho subito inserito la schermata di Home dell'applicazione, ma ho inserito una Activity, alla quale ho attaccato il "SetMotivatorFragment" una recyclerView (gestita come fatto a lezione, in modo analogo al carrello), nella quale è possibile selezionare con degli switch i differenti "motivatori" che rimarranno salvati nelle SharedPreferences. Per modificare questi valori, bisognerà accedere allo stesso fragment presente anche nel menu, oppure bisognerà eseguire il logout, che pone a "false" tutti i valori booleani degli interruttori contenuti nelle SharedPreferences. Nota: ho gestito questa activity di passaggio, in modo che al secondo accesso, se sono già stati selezionati degli interruttori in precedenza e si è già loggati, allora si andrà direttamente alla Home, senza passare da questa activity. Ho fatto anche in modo che l'utente possa utilizzare questa applicazione come sola app per la corsa. Infatti, se non viene selezionato nessun interruttore, si può accedere all'interfaccia che gestisce l'attività di corsa, mentre questo non è possibile se si vuole accedere alla modalità in cui verrà solo riprodotto il suono. Quest'operazione verrà impedita infatti dalla presenza di un alertDialog, che genera un intent a ritroso, facendo tornare l'app alla MainActivity.
Di seguito ho riproposto il codice di gestione delle sharedPreferences:

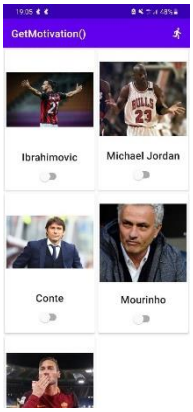
```
ArrayList<Boolean> valSwitch;
ArrayList<String> chiaviSpListMain = new ArrayList<String>();
valSwitch = new ArrayList<>();
//setto nomi chiavi, per recuperare booleani precedenti
chiaviSpListMain.add("value1");
chiaviSpListMain.add("value2");
chiaviSpListMain.add("value3");
chiaviSpListMain.add("value4");
chiaviSpListMain.add("value5");
SharedPreferences sharedPreferences =
getApplicationContext().getSharedPreferences("save", MODE_PRIVATE);
for (int i = 0; i < chiaviSpListMain.size(); i++) {
    valSwitch.add(sharedPreferences.getBoolean(chiaviSpListMain.get(i), false));

}
for(int i=0; i<valSwitch.size();i++){
    if(valSwitch.get(i)){
        startActivity(new Intent(getApplicationContext(), Home.class));
    }  //se tutti i controlli (login già eseguito e switch già selezionati in
passato) vanno bene così si va direttamente alla home activity
}

    getSupportFragmentManager()
            .beginTransaction()
            .add(R.id.contenitore, new SetMotivatorFragment())
            .commit();
```

In relazione ai valori contenuti nelle sharedPreferences, si passa alla Home, oppure si va al fragment di scelta.

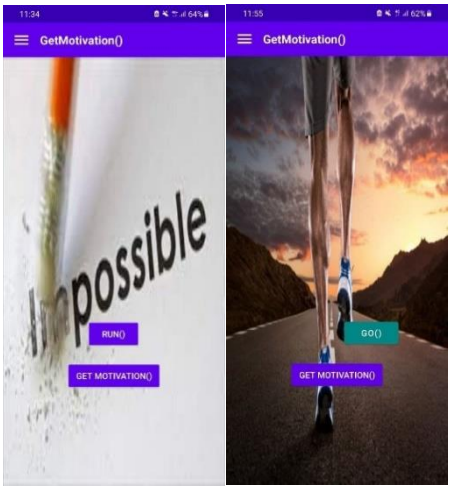
A destra, ho riprodotto un'immagine della recyclerView presente nell'applicazione con gli switch. Sopra la RecyclerView c'è un menu con una sola icona, che permette di passare alla schermata successiva (la Home).
Nota: una scelta progettuale "comoda", che non segue fedelmente quelli che sono gli ideali di buona programmazione, è stata quella di porre la lista contenenti i dati di ogni singolo viewHolder come statica, per poi usarla in lettura dopo la sua creazione ed avere una maggiore facilità nella scrittura dei metodi successivi, soprattutto per quanto riguarda la riproduzione del suono. Infatti, ho scelto di inserire nei dataModel, oltre agli indirizzi delle immagini, anche gli indirizzi dei file audio, per non dover in seguito creare liste o vettori (all'occorrenza) e poter accedere subito, dai data model, alle risorse audio. Sono in ogni caso consapevole del fatto che un approccio di buona programmazione sarebbe stato quello di creare nuove liste, dove necessario, e ripetere tutto il ragionamento senza istanziare una variabile statica.
(di questa nota ne parlerò in maniera più approfondita in fondo alla trattazione)

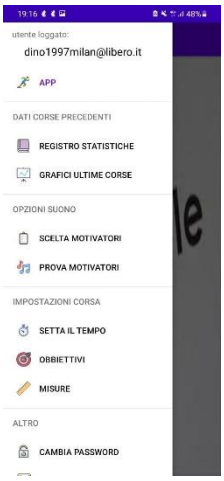


Home

In questa activity ho inserito un drawer layout, implementato seguendo fedelmente le istruzioni del link allegato nei riferimenti. Nel fragment principale ho inserito la schermata di scelta della modalità da svolgere, il cui background cambia in relazione al click dei tasti. Dopo aver fatto questo, ho aggiunto una serie di fragment, riguardanti impostazioni che descriverò durante la spiegazione della gestione del suono e dopo aver spiegato il meccanismo di funzionamento del service.

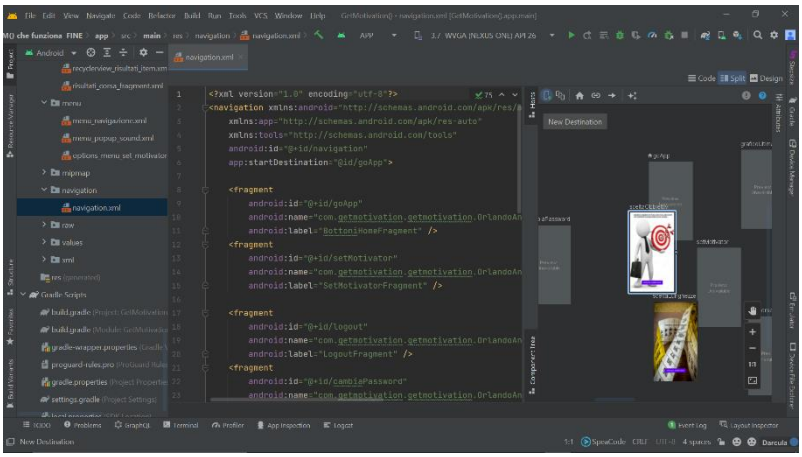
Qui a destra, ho posto le immagini che mostrano la commutazione del fragment principale, del DrawerLayout attaccato alla Home.





Questa a sinistra, invece, è un'immagine del menu a tendina, al quale si può accedere facendo uso del tasto, oppure trascinando il dito sullo schermo da sinistra verso destra. Tutte le opzioni di scelta sono relative a fragment che sono stati aggiunti ad un apposito file xml, chiamato "navigation". Poi ho creato un file xml nella cartella menu, contenente tutti gli item che si possono visualizzare nell'immagine a sinistra, le cui icone sono state scaricate da un sito che ho allegato nei riferimenti, il cui nome è icons8.

Sulla destra ho lasciato uno screenshoot del file xml navigation. Per aggiungere fragment (o activity) a questo file basta premere sull'icona sulla quale avevo lasciato il puntatore del mouse nell'immagine a destra (è presente un cerchietto blu trasparente). Per associare i diversi fragment alle icone del menu, è necessario sostituire gli id dei fragment con quelli delle icone del menu. Quello che viene mostrato nella foto è il risultato finale.



GESTIONE SUONO

Questo come descritto sotto è uno degli aspetti che globalmente mi ha creato un po' di difficoltà durante la realizzazione del service. Mi sono approcciato a questa tematica in due casi, durante l'uso nel service e in un'activity che permette di provare i diversi suoni, alla quale si può accedere da una recyclerView legata allo stesso file xml della recyclerView di scelta dei "motivatori", con un adapter differente, che gestisce per ogni ViewHolder la pressione di un tasto.
Per ogni suono ho dovuto prestare particolare attenzione alle operazioni di rilascio e di gestione dell'audioFocus, in particolare:

```
public class ProvaSuoniActivity extends AppCompatActivity implements
AudioManager.OnAudioFocusChangeListener{
```

```
mAudioManager = (AudioManager)this.getSystemService(Context.AUDIO_SERVICE);
```

in tutti i casi di gestione del suono ho creato un'istanza dell'audioManager, dopo aver implementato il Listener legato all'audioFocus, in un certo senso "in comunicazione col sistema", per poi gestire il focusAudio nei metodi soundStart e soundStop. In particolare ho fatto override del metodo onAudioFocusChange, nel quale ho gestito l'interazione dell'audio della mia applicazione con possibili eventi esterni:

```
@Override
public void onAudioFocusChange(int audioFocusChanged) {
    switch(audioFocusChanged){
        case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT:
        case AudioManager.AUDIOFOCUS_LOSS:
            mp.setVolume(0.7f,0.7f);  //siccome l'esecuzione dell'asyncTask
continua per fare in modo che l'utente sia a conoscenza del fatto che l'app è
ancora in esecuzione abbasso solo il volume anzicchè stoppare il suono
(ovviamente con le chiamate e col navigatore non ci sono problemi)
            //soundStop();
            break;
        case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK:
            mp.setVolume(0.2f,0.2f);  //così se l'utente usa googleMaps mentre
corre non ci sono problemi
            break;
        case AudioManager.AUDIOFOCUS_GAIN:
            mp.setVolume(1f,1f);
            break;
    }
}
```

L'audioFocus è qualcosa che viene anche gestita dal sistema che decreta a quale applicazione dare il focus, in relazione ad alcuni livelli di priorità. Ad esempio, se vi è una telefonata in ingresso, il focus viene completamente tolto alle altre app, con priorità normale o bassa. Questa digressione è per introdurre le scelte fatte all'interno del metodo onFocusChange. Nel service ho gestito una playlist di suoni, nella quale terminato un suono dopo un certo tempo (scelto dall'utente) parte il suono successivo, che richiederà l'audio focus. Se avessi annullato il suono della mia app per lasciarlo ad un'app di musica, dopo un certo tempo la mia app avrebbe richiesto nuovamente il focus, in modo spiacevole. Per render meno appariscente questo fenomeno quindi ho solo abbassato il volume, per poi farlo ritornare alla riproduzione dei file audio successivi.
Nota: in presenza di eventi con priorità maggiore ovviamente il sistema toglie l'audioFocus alla mia app, lo fornisce all'app con priorità maggiore, e lo restituisce alla mia applicazione solo quando è terminato il funzionamento delle altre app. Come nel caso della chiamata.
Allo stesso modo, seguendo gli standard indicati da GoogleDeveloper ho abbassato l'audio in modo consistente in presenza di eventi come la riproduzione audio da parte di GoogleMaps.

A seguire ripropongo il codice dei metodi soundStart e soundStop che è sono rappresentati in modo analogo nell'activity e nel service:

```
public void soundStart(){
    registerReceiver(mAudioRumoroso,rumoreIntentFilter);
        int requestAudioFocusResult =
mAudioManager.requestAudioFocus(this,AudioManager.STREAM_MUSIC,AudioManager.AUDI
OFOCUS_GAIN);
        if(requestAudioFocusResult== AudioManager.AUDIOFOCUS_REQUEST_GRANTED){
        mp.start();
    }
}

public void soundStop(){
    if(mp!=null) {
        mp.stop();
        mp.release();
        mp= null;
        mAudioManager.abandonAudioFocus(this);
        unregisterReceiver(mAudioRumoroso);
    }
}
```

Un'altra operazione fondamentale alla quale prestare attenzione è l'operazione di release. Il caso indicato sopra è quello della "ProvaSuoniActivity" (nel service ho gestito questo aspetto in modo analogo). L'aspetto fondamentale a cui prestare attenzione è che il suono deve esser rilasciato sempre prima della riproduzione dell'audio successivo ed è fondamentale il rilascio anche in fase di distruzione dell'activity o stop del service. Sempre in relazione al rilascio del suono, ho dovuto fare particolare attenzione a non svolgere l'operazione di rilascio su una risorsa già rilasciata, poiché questa operazione avrebbe compromesso il funzionamento dell'activity.
Ho aggiunto anche un broadcast di sistema per rilasciare il suono in presenza di eventi che possono far diventare il suono rumoroso. L'esempio principale associato a questa operazione, con il quale ho testato anche il funzionamento del codice, è quello di scollegare gli auricolari durante la riproduzione del suono. Durante questa operazione l'audio può diventare fastidioso, per cui ho aggiunto un broadcast che interagendo col sistema gestisce ed evita questo evento:

```
rumoreIntentFilter = new IntentFilter(AudioManager.ACTION_AUDIO_BECOMING_NOISY);
private class AudioRumoroso extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        soundStop();
    }
}
```

```
public void soundStart(){
    registerReceiver(mAudioRumoroso,rumoreIntentFilter);
```

istanziando l'intent filter che riconosce questo fenomeno ed il broadcast mostrato nelle righe sopra (che stoppa il suono), è possibile poi registrare il broadcast in base di riproduzione della risorsa audio per gestire lo stop del suono in presenza di audio rumoroso.
Nel service in aggiunta ho creato una playlist di suoni, riprodotti all'interno di un AsyncTask, che ho chiamato GetMotivation(), necessaria alla riproduzione degli audio e dalla quale prende il nome l'applicazione. Col metodo che si chiama anch'esso getMotivation() (con la prima lettera minuscola) creo la playlist di suoni da riprodurre. Una cosa che può risultare interessante è che ho pensato anche ad un metodo per disordinare in modo random la playlist più volte prima della riproduzione e disordinarla nuovamente una volta riprodotti tutti i suoni contenenti nella playlist. Al termine della playlist la riproduzione continua, facendo ripartire la playlist precedente dopo che questa è stata ulteriormente disordinata.
Ho pensato di gestire il suono in questo modo per rendere meno ripetitiva e più coinvolgente l'esperienza di uso dell'applicazione.

```
public ArrayList<Integer> getMotivation() {
    ArrayList<Integer> listaDaCaricare;
    listaDaCaricare = new ArrayList<>();
    ArrayList<Boolean> valSwitch = new ArrayList<>();
    valSwitch = valSwitch();
    for (int l = 0; l < valSwitch.size(); l++) {
        if (valSwitch.get(l)) {
            listaDaCaricare.add(mDataModelList.get(l).getmMotivatorSound1());
            listaDaCaricare.add(mDataModelList.get(l).getmMotivatorSound2());
            listaDaCaricare.add(mDataModelList.get(l).getmMotivatorSound3());
            listaDaCaricare.add(mDataModelList.get(l).getmMotivatorSound4());
            listaDaCaricare.add(mDataModelList.get(l).getmMotivatorSound5());
        }
    }

    return listaDaCaricare;
}
```

```
public void playNext() {
        //final int ATTESA = 1000 *
(sharedPreferences.getInt("secondiGetMotivation", 0));
//        for(int j=0;j<playlist.size();j++){
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                soundStop();
                mp.release();
                unregisterReceiver(mAudioRumoroso);
                mp = MediaPlayer.create(getApplicationContext(),
playlist.get(++i));
                soundStart(); //per esser più precisi questa riga di codice
andrebbe eseguita sse questa applicazione aveva già l'audioFocus
                if (playlist.size() > i + 1) {
                    playNext2();
                    Log.d("PlayNext", String.valueOf(++conta));
                } else {
                    i = -1;
                    playlist = disordinaPlaylist(playlist);
                    playNext2();
                    Log.d("PlayNext", String.valueOf(++conta));
                }
            }
        }, mp.getDuration() + 100 + (time * 1000));
```

```
        //}
    }
```

Oltre a questo metodo, ho creato un metodo uguale e questi due si alternano tra loro per fare in modo che la riproduzione sonora non venga mai interrotta fino alla pressione del tasto stop.

```
public void loopMusic() {
        playlist = new ArrayList<>();
//              playlist.add(R.raw.sound1conte);
//              playlist.add(R.raw.sound4conte);
        playlist = getMotivation();    //col metodo getMotivation() mi costruisco
la playlist (lista degli indirizzi associati ad i diversi suoni)
        playlist = disordinaPlaylist(playlist);   //per rendere l'esperienza di
ascolto non ridondante ho aggiunto un metodo che rende casuale l'ordinamento dei
suoni all'interno della playlist
//    timer = new Timer();
        mp = MediaPlayer.create(getApplicationContext(), playlist.get(0));
        soundStart();
        if (playlist.size() > 1) {
            playNext();
        }
    }
```

```
public class GetMotivation extends AsyncTask<String, String, String> {


    @Override
    protected String doInBackground(String... strings) {

        loopMusic();
```

(in onPostExecute non ho fatto nulla)
Come si può intuire dai commenti del codice, prima dell'esecuzione del metodo loopMusic viene istanziato un Timer, che poi sarà opportunamente cancellato in fase di distruzione del service.

Qui infine ho copiato il metodo per disordinare la playlist:

```
// "algoritmo per una permutazione casuale di un insieme finito"
// con questo metodo scritto qui di seguito ho riprodotto un semplice algoritmo
trovato su internet, inizialmente ideato da Ronald Fisher e Frank Yates, che si
pone l'obbiettivo di disordinare l'ordine degli elementi all'interno di un
vettore
// per far si che l'ordinamento sia il più casuale possibile ho introdotto un
ulteriore ciclo for, che permette di ripetere l'algoritmo un numero di volte
pari al numero
public ArrayList<Integer> disordinaPlaylist(ArrayList<Integer> playlist) {
    for (int j = 0; j < playlist.size(); j++) {
        for (int i = 0; i < playlist.size(); i++) {
            int randomIndex = generator.nextInt(playlist.size());
            tmp = playlist.get(i);
            playlist.set(i, playlist.get(randomIndex));
            playlist.set(randomIndex, tmp);
        }
    }
    return playlist;
}
```

Il meccanismo è semplicemente quello di sostituire più volte gli elementi tra loro usando una variabile di appoggio. Ho visto su internet che questo approccio è stato riconosciuto come l'algoritmo di due famosi matematici.

Infine, all'interno del service per tenere la cpu sveglia anche quando lo schermo è nero ho implementato la wakeLock, che ho opportunamente rilasciato in fase di distruzione del service:

```
PowerManager powerManager = (PowerManager) getSystemService(POWER_SERVICE);
wakeLock = powerManager.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
        "MyApp::MyWakelockTag");
wakeLock.acquire();
```

facendo override di onCreate ho implementato queste righe di codice.

```
@Override
    public void onDestroy() {
        soundStop();
        mp.release();
        timer.cancel();
        unregisterReceiver(mAudioRumoroso);
//        wakeLock.release();
        if (modalità == 0) {
            sensorManager.unregisterListener(this);
        }
        wakeLock.release();
        //so che esiste un istante esatto in cui l'utente premendo il tasto stop
svolgerà due unregister e farà crashare l'app
        //però: questa cosa può avvenire solo durante l'esecuzione del metodo
playNext quando vengono svolte le istruzioni unregister, create e poi
soundStart;
        //in media un'istruzione impiega due cicli di clock per esser svolta e
la frequenza dei processori odierni è elevatissima
        //mi è sembrato molto più produttivo deregistrare il Broadcast quando il
service viene distrutto, senza lasciare il compito Garbage Collector, altrimenti
riaprendo e chiudendo il service verrà allocata sempre più memoria
        super.onDestroy();
        Log.d(TAG, "onDestroy: MyFrgService has been killed!");
    }
```

In onDestroy del service ho rilasciato tutti gli elementi precedentemente usati.

USO DEGLI EXTRA DEGLI INTENT

Approcciandomi alla gestione del suono ho dovuto riprodurre interfacce ricche di operazioni, che a volte finivano per esser sempre le stesse. Quindi ho fatto un ampio uso degli extra degli intent sia nel service, per creare un solo service funzionante allo stesso modo per entrambe le modalità, sia nel FragmentProvaSuoni. Grazie agli extra degli intent, ho potuto gestire la visualizzazione di immagini ed oggetti differenti ed ho potuto in un certo senso "nascondere" differenti funzionalità del service, in modo da non realizzare più volte interfacce con operazioni ridondanti, ma gestire in entrambi i casi tutto con un'interfaccia sola.

Service:

```
modalità = intent.getIntExtra("TIME", 1);
        if (modalità == 1) {
            SharedPreferences sharedPreferences =
this.getSharedPreferences("save", MODE_PRIVATE);
            time = sharedPreferences.getInt("secondiGetMotivation", 0);

            createNotificationChannel();
//            Intent resultIntent = new Intent(this,GetMotivation.class);
//            PendingIntent resultPendingIntent =
PendingIntent.getActivity(this,0,resultIntent,PendingIntent.FLAG_UPDATE_CURRENT)
;
            Notification mNotification = new NotificationCompat.Builder(this,
                    "MY_CHANNEL_ID")
                    .setContentTitle("Get focused")
                    .setContentText("Fallisci solo quando smetti di provare")
                    .setSmallIcon(R.drawable.icons8motivation256)
//                    .setContentIntent(PendingIntent
//                    .getActivity(this, 0, new Intent(this,
GetMotivation.class),
```

```
//                              PendingIntent.FLAG_UPDATE_CURRENT))
//                      .setContentIntent(resultPendingIntent)
                    .build();
// SE IL SERVICE è AVVIATO DALL?ACTIVITY GETMOTIVATION IL PENDINGINTENT NON
FUNZIONA ANCHE SE HO USATO LE STESSE RIGHE DI CODICE, HO SUPPOSTO CHE IL VIDEO
APPESANTISCA L'ACTIVITY E PER QUESTO NON FUNZIONI
// HO PREFERITO RIMUOVERE LE RIGHE DI CODICE DEL PENDINGINTENT PERCHè INSISTENTO
CON SVARIATI TENTATIVI HO NOTATO UN PAIO DI VOLTE CHE QUESTO INTENT FINISCE PER
FAR CRASHARE L'APP
            startForeground(1, mNotification);
        } else {
```

qui ho mostrato come in relazione alla chiave "TIME" nel service avvio la barra delle notifiche legata alla modalità GetMotivation(), oppure avvio la barra legata alla modalità Run() e tutte le funzioni legate alla corsa col sensore (il codice è implementato dopo l'else).

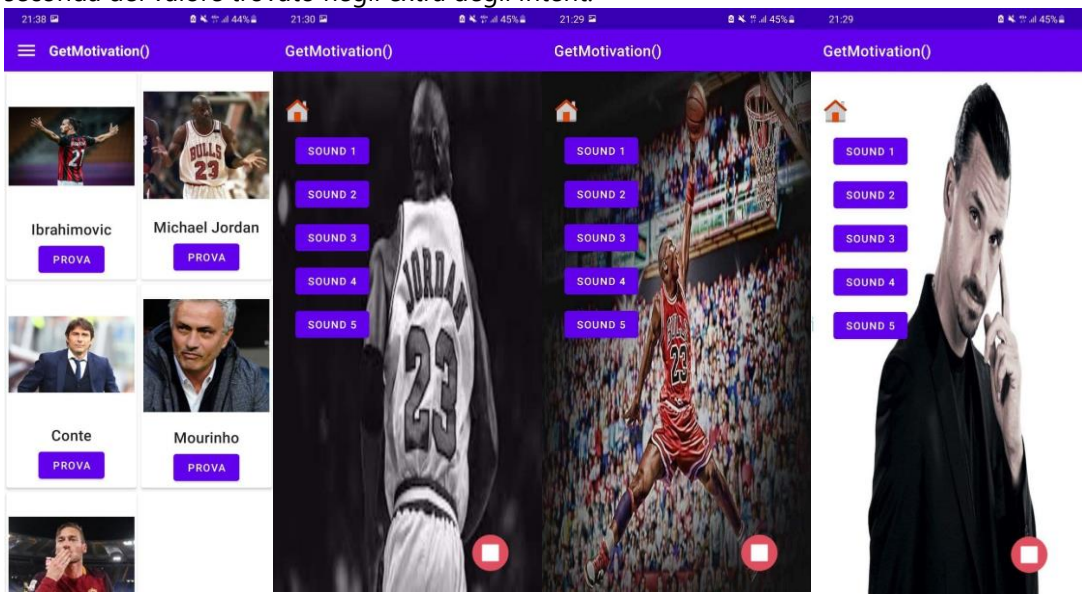
ProvaSuoniActivity:

In questa activity l'interfaccia cambia a seconda di qual è l'intent con il quale arriviamo all'activity:

```
public void bind(DataModel dataModel, Context mContext) {
    mImageView.setImageDrawable(ContextCompat.getDrawable(mContext,
dataModel.mMotivatorImageId));
    mMotivatorName.setText(dataModel.getmMotivatorName());

    mButton.setOnClickListener(new View.OnClickListener() {
                                   @Override
                                   public void onClick(View v) {
                                       Intent intent = new Intent(mContext,
ProvaSuoniActivity.class);
                                       intent.putExtra("VedoChièIlGiocatore",
dataModel.posizioneNellaRecyclerView);
                                       mContext.startActivity(intent);
                                   }
                               });
```

Nel metodo bind del MyViewHolderProvaSuoni (nel corrispondente adapter) ho implementato un click che registra negli extra dell'intent un valore differente a seconda del viewHolder selezionato, prima di questo vengono resi invisibili gli switch perché il file xml è lo stesso usato in precedenza per la RecyclerView iniziale. Dalla recyclerView si avrà accesso ad un'activity che è sempre la stessa ed il cui funzionamento cambia a seconda del valore trovato negli extra degli intent.



Qua ci sono alcune immagini del risultato.

In aggiunta con queste immagini volevo far vedere come ho implementato il click anche sullo sfondo di questa activity, poiché cliccando sull'immagine di Michael Jordan appaiono nuove immagini, così come succede anche con gli altri "motivatori".

RIPRODUZIONE VIDEO

Nelle due activity principali ho deciso di riprodurre due video. Nell'activity legata alla corsa ho attaccato il fragment con il bottone che permette di cambiare la visualizzazione delle features acquisite. Questa operazione l'avevo fatta perché, anche avendo impostato dal file xml il video su tutto lo schermo, succedeva che la pressione del tasto presente nel layout innescava il ridimensionamento del video, oltre a modificare le view presenti come desiderato; comunque utilizzando un fragment per separare il video dalle altre view questa problematica non era ancora stata risolta. Per risolvere questo problema ho introdotto il seguente codice:

```
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

settando i parametri del layout col flag fullscreen, non ho visualizzato più problemi durante la riproduzione del video.

La riproduzione dei video continua in loop senza sosta finché le activity non vengono interrotte (distruzione, tasto back o schermata di visualizzazione di tutte le app usate). Questo è possibile grazie ad una classe anonima che implementa un Listener ed alla classica operazione di override di quello che è il metodo principale:

```
videoRun.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
    @Override
    public void onCompletion(MediaPlayer mediaPlayer) {
        mediaPlayer.start();
    }
});
```

in questo modo è possibile creare un loop nell'esecuzione della risorsa MediaPlayer.

(Ovviamente i video sono privi di audio, che altrimenti avrebbe finito col sovrapporsi con l'audio dei "motivatori")

Mentre il video dell'activity Run è una gif trovata su internet, il secondo video l'ho creato io con windows movie maker, utilizzando alcune immagini trovate su internet.

Activity Run

Le istruzioni innescate dalla presenza di questa activity rappresentano il cuore dell'applicazione. Infatti nell'onCreate di questa activity viene gestito un intent che avvia il service "in modalità run": ovvero passerà al service, attraverso gli extra, un valore che permetterà di svolgere contemporaneamente le operazioni relative all'acquisizione dei dati da parte dell'accelerometro e della riproduzione della musica, che ho discusso nei paragrafi precedenti.

Ho costruito un metodo all'interno del service che ho chiamato "getAccelerazione" e permette di prelevare i valori dal sensore accelerometro e poi utilizzarli per estrapolare analiticamente i dati di interesse, inviati all'activity attraverso un CUSTOM_BROADCAST, per aggiornare continuamente quello che è lo stato dell'activity. (Il nome del custom broadcast come suggerito a lezione contiene al suo interno il nome del package dell'applicazione)

```
public static final String CUSTOM_BROADCAST =
"com.getmotivation.getmotivation.miosensore_CUSTOM_BROADCAST";
```

```
private void getAccelerazione(SensorEvent event) {
    float[] values = event.values;

    float x = values[0];
    float y = values[1];
    float z = values[2];

    double stimaRadiceSommaInQuadratura = Math.sqrt((x * x + y * y + z * z));
    a = stimaRadiceSommaInQuadratura;
```

```
    delta_a = a - a0;
    a0 = a;
    long actualTime = event.timestamp;
    //t= (actualTime-lastUpdate)*Math.pow(10,-3);
    if (delta_a > 6) {
        if (actualTime - lastUpdate < 200) {
            return;
        } //se non è trascorso almeno 1/5 di secondo non si aggiorna
        lastUpdate = actualTime;
        passi++;
        s = valLstep * passi* Math.pow(10,-3);   //un passo equivale a 60cm , ho
moltiplicato per 10^-3 per rappresentare la distanza in chilometri
        s = Math.ceil(s * 1000) / 1000;
        //s=((s0+(t*t*a))/1000); //ho considerato il percorso svolto rettilineo
ed ho usato una semplice formula di fisica classica
        //s0=s*1000;  //questo perchè all'interno della formula s0 mi serve in
metri
        //v= delta_a*t*3.6;
        kmPercorsi = String.valueOf(s);
        kcal = passi * 0.5 * Math.pow(10, -3) * valPeso;  //ho trovato questa
formula su internet, 65 è il valore di default del peso
        kcal = Math.ceil(kcal * 1000) / 1000;
        intent = new Intent(CUSTOM_BROADCAST);
        intent.putExtra("passi", passi.toString());
        intent.putExtra("distanza", kmPercorsi);
        intent.putExtra("kcal", String.valueOf(kcal));
        sendBroadcast(intent);
    }
}
```

Il fragment legato all'activity conterrà il receiver del broadcast inviato dal service:

```
private BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {

        String message = intent.getAction();
        passi =intent.getStringExtra("passi");
        kcal =intent.getStringExtra("kcal");
        distanza =intent.getStringExtra("distanza");
        //Toast.makeText(getContext(),passi +"  "+ kcal + "  "+distanza ,
Toast.LENGTH_LONG).show();

        if (nomeDati.getText().toString() == " PASSI: ") {
            dati.setText(passi);
        } else {
            if (nomeDati.getText().toString() == " KM PERCORSI: ") {
                dati.setText(distanza);
            } else {
                dati.setText(kcal);
            }
        }

    }
};
```

Come si può vedere dal codice, ho implementato il broadcast ed il metodo onReceive con una classe anonima, al momento stesso dell'istanza del broadcast. Il broadcast ovviamente andrà opportunamente registrato e deregistrato:

```
getActivity().registerReceiver(mReceiver, new IntentFilter(CUSTOM_BROADCAST));
```

```
getActivity().unregisterReceiver(mReceiver);
```

All'interno sempre del fragment, attraverso il metodo getActivity() ho potuto svolgere registrazione e deregistrazione del broadcast rispettivamente in onViewCreated ed in onDestroyView.
Il broadcast aggiornerà il testo sul bottone dell'activity in relazione al parametro che l'utente vuole visualizzare. Ovviamente tutti i valori inviati dal broadcast verranno registrati nell'activity; nel momento in cui verrà premuto il tasto stop, tutti i parametri verranno salvati nelle sharedPreferences, per poi esser visualizzati nel fragment relativo ai risultati ed in più verrà svolta una query per inserire i dati nel databaseHelper (tenendo conto anche della data e della mail dell'utente che si è loggato) :

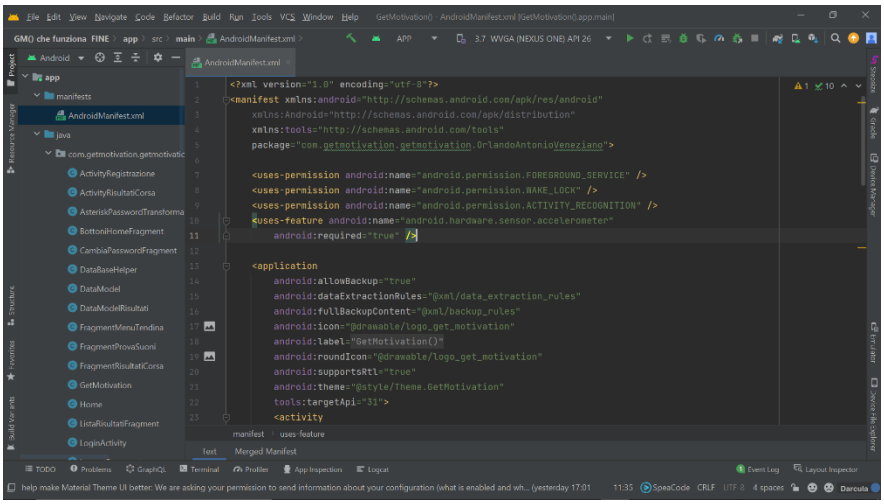
```
buttonStop.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(getContext(),
ActivityRisultatiCorsa.class);
//                intent.putExtra("risultato passi",passi);
//                intent.putExtra("risultato km",distanza);
//                intent.putExtra("risultato kcal",kcal);
                Date c = Calendar.getInstance().getTime();
                Log.d("ListaRisultatiFragment","Current time => " + c);

                SimpleDateFormat dataFormat = new SimpleDateFormat("dd-MMM-
yyyy", Locale.getDefault());
                String formattedDate = dataFormat.format(c);
                String email =
FirebaseAuth.getInstance().getCurrentUser().getEmail();


mDatabaseHelper.doInsert(formattedDate,passi,distanza,kcal,email,user);

                SharedPreferences.Editor editor =
getContext().getSharedPreferences("save",MODE_PRIVATE).edit();
                editor.putString("risultato passi",passi);
                editor.putString("risultato km",distanza);
                editor.putString("risultato kcal",kcal);
                editor.apply();
                stopFrg();
                startActivity(intent);
            }
        });
```

Permessi:
Nel manifest dell'applicazione ho implementato diversi permessi, per fare uso di wakelock, service e dell'accelerometro.
(credo manchino i permessi di runtime e purtroppo non sono a conoscenza della tecnica di implementazione)



DataBaseHelper

Attraverso l'uso di questa classe, che implementando l'SQLiteOpenHelper semplifica fortemente la creazione dei database, ho implementato la tabella contenenti tutti i dati delle corse, di tutti gli utenti , per questa applicazione. Ovviamente poi in

relazione all'utente loggato ho distinto i dati che sarebbero stati visualizzati nella recyclerView che contiene le statistiche delle corse del singolo utente:

```
private static final String COL0 = "ID";
private static final String COL1 = "data";
private static final String COL2 = "passi";
private static final String COL3 = "km";
private static final String COL4 = "kcal";
private static final String COL5 = "email";
public static final String USER = "corse";
String user;
```

Questi sono i parametri che ho preso in considerazione nella tabella (gli attributi della tabella).
Ho creato un metodo di creazione della tabella, che poi ho ripetuto nell'onCreate della classe DatabaseHelper:

```
public void onCreateModified(SQLiteDatabase db,String user){
    String createTable = "CREATE TABLE " + user + " (ID INTEGER PRIMARY KEY
AUTOINCREMENT, " +
            COL1 +" TEXT,"+ COL2 + " TEXT,"+
            COL3+" TEXT,"+COL4+" TEXT,"+COL5+" TEXT)";
    db.execSQL(createTable);
    Log.d(TAG,"Tabella creata correttamente");
}
```

Una volta creata la tabella ho scritto alcune query e prima di svolgere troppe operazioni con i cursori ho riflettuto su quale fosse il modo per usare meno cursori possibili ed ho scritto un ciclo, che facendo uso di un solo cursore (poi opportunamente chiuso), acquisisce tutti i dati che serviranno per popolare la recyclerView.
Query definite:

```
public boolean doInsert(String data,String passi,String km,String kcal,String
email,String user) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put(COL1,data);
    contentValues.put(COL2,passi);
    contentValues.put(COL3,km);
    contentValues.put(COL4,kcal);
    contentValues.put(COL5,email);

    Log.d(TAG, " Dati aggiunti " + data + " "+ passi +" "+km +" "+kcal+ " "+"
aggiunti alla tabella " + user);

    long result = db.insert(user, null, contentValues);

    if (result == -1) {
        return false;
    } else {
        return true;
    }
}
```

Questo è il codice dell'insert.

```
public Cursor getDay(String email,String user,String day){
    SQLiteDatabase db = this.getWritableDatabase();
    String query = "SELECT * FROM " +user +
            " WHERE " + COL5 + " = '" + email + "' AND "+
            COL1 + " = '"+ day + "'";
    Cursor data = db.rawQuery(query, null);
    return data;
}
```

Qusta è la query che mi permette di prelevare i dati dell'utente con email "email" il giorno "day". L'ho usata alla fine per la creazione dei 3 differenti grafici.

```
public Cursor getDatiUtente(String email,String user){
    SQLiteDatabase db = this.getWritableDatabase();
    String query = "SELECT * FROM " +user +
            " WHERE " + COL5 + " = '" + email + "'";
    Cursor data = db.rawQuery(query, null);
    return data;
}
```

Questa è la query usata per il popolamento della recyclerView, contenente tutti i dati di un solo utente con email "email".

```
String email = FirebaseAuth.getInstance().getCurrentUser().getEmail();
Cursor cursor = mDatabaseHelper.getDatiUtente(email,user);
mDataModelList = new ArrayList<DataModelRisultati>();
while(cursor.moveToNext()){
    ++conta;
    mDataModelList.add(new DataModelRisultati(String.valueOf(conta),
            cursor.getString(1),cursor.getString(2),
            cursor.getString(3),cursor.getString(4),cursor.getString(5)));
//così posso ottenere a mio piacimento tutte le informazioni di ogni colonna

}
cursor.close();
```

Questo è il codice scritto con un ciclo while, nel fragment nel quale vengono creati i DataModelRisultati per generare la RecyclerView con i dati presenti nel database. Questi dati poi li ho prima invertiti, in modo da far visualizzare all'utente in cima alla lista le ultime corse, poi li ho caricati nella recyclerView.
A destra ho riportato il risultato finale:

FILE DI LOG

L'uso dei file di log è stato particolarmente importante, per approcciarmi sia ai service che al database, proprio per comprendere la presenza di errori ed eccezioni e vedere il risultato delle operazioni lunghe prima che queste fossero portate a termine.

GRAFICO CORSE SVOLTE NELL'ULTIMA SETTIMANA

Dopo aver implementato nel gradle il seguente codice, ho fatto uso della libreria MPAndroidChart ed in particolare delle barchart per fare 3 differenti grafici:

```
implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'
```

questo va implementato all'interno del build gradle.

```
maven { url 'https://jitpack.io' }
```

questa linea di codice invece deve esser aggiunta alle repositories.

Le coordinate dei differenti barChart che ho scelto erano giorni ed i valori di passi, km e kcal. Per cui ho dovuto ricavare questi valori che poi ho inserito come barEntries (coordinate) nei differenti barchart. Per i giorni ho usato la classe Calendar, mentre per gli altri dati ho fatto sempre uso di una query dentro un ciclo for, per ottenere i dati di interesse facendo uso di un solo cursore.

In questo fragment ho fatto tantissimo uso delle istanze della classe Calendar, che mi permettevano assieme al metodo add di ottenere le date del giorno corrente e dei 6 giorni precedenti.

```
calendar.add(Calendar.DATE,-7);

        for(int i=0; i<7;i++) {
//            if(i==0){
//                calendar.add(Calendar.DATE,0);
//            }
//            else{
            calendar.add(Calendar.DATE,+1);//}
            giorni[i] = calendar.getTime();
            stringheGiorni[i]= dataFormat.format(giorni[i]);
            Log.d("Date girni registrate",stringheGiorni[i]);
        }
```

Ho utilizzato molti vettori di Stringhe per poter creare cicli di questo tipo, senza dover ripetere manualmente le stesse operazioni.

Per ottenere i dati da inserire nei grafici dal database ho scritto il seguente codice:

```
for(int i=0; i<7;i++){
    Cursor cursor = dataBaseHelper.getDay(email, user, stringheGiorni[i]);
    while (cursor.moveToNext()) {
        floatDiAppoggio = 0;
        floatDiAppoggioKm=0;
        floatDiAppoggioKcal=0;
        floatDiAppoggio = parseFloat(cursor.getString(2));
        floatDiAppoggioKm=parseFloat(cursor.getString(3));
        floatDiAppoggioKcal=parseFloat(cursor.getString(4));
        passiXday[i] = floatDiAppoggio + passiXday[i];
        kmXday[i] = floatDiAppoggioKm + kmXday[i];
        kcalXday[i] = floatDiAppoggioKcal + kcalXday[i];
        //Log.d("FragmentRisultatiCorsa",cursor.getString(2));
    }
    cursor.close();
}
```

Inizialmente ho caricato le coordinate in questo modo:

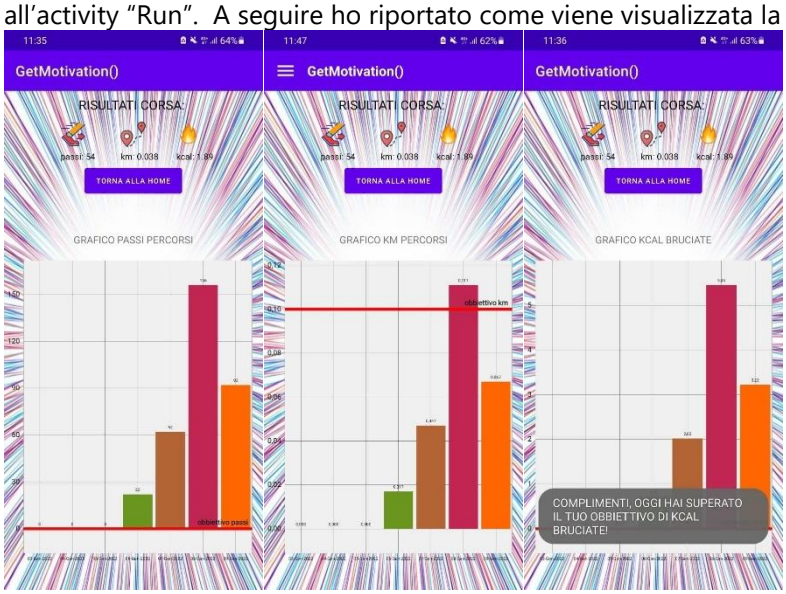
```
for(int j=0;j<7;j++){
barEntries.add(new BarEntry(j+1, passiXday[j]));}
```

come si nota, qui i parametri nelle ascisse non sono i giorni. Attraverso l'uso di una classe che estende ValueFormatter, ho potuto inserire i giorni come ascisse dei grafici col seguente codice:

```
public class DayAxisValueFormatter extends ValueFormatter {
    private final BarLineChartBase<?> chart;
    public DayAxisValueFormatter(BarLineChartBase<?> chart) {
        this.chart = chart;
    }
    @Override
    public String getFormattedValue(float value) {
        return stringheGiorni[((int)value-1)];  //qua sono indicati i valori
restituiti sull'asseX
    }
}
```

Poi, in generale, per i diversi grafici ho fissato tutte le caratteristiche degli assi e del grafico, come ad esempio la granularità tra i diversi elementi delle ascisse o la dimensione del testo dei giorni (senza un corretto dimensionamento, i valori delle ascisse sarebbero stati sovrapposti, quindi ho scelto una dimensione abbastanza piccola per permettere la corretta visualizzazione anche sui telefoni da 2.7 pollici). Ho aggiunto anche degli asintoti associati agli obbiettivi dell'utente, visibili solo quando ci si avvicina a questi obbiettivi,

oppure quando gli stessi vengono superati: in tal caso, viene stampato un messaggio con un Toast, che si complimenta con l'utente per i risultati della giornata corrente. Ed infine ho settato l'onClickListener sui 3 barchart (due dei quali inzialmente con visibilità GONE) per passare in seguito all'esecuzione del click dalla visualizzazione di un grafico a quello successivo, in analogia a ciò che è stato fatto nel fragment legato all'activity "Run". A seguire ho riportato come viene visualizzata la schermata in cui sono presenti i grafici:



FRAGMENT PER SETTAGGIO IMPOSTAZIONI

Nel menu ho aggiunto dei fragment che permettono all'utente di gestire a proprio piacimento il funzionamento dell'applicazione e di impostare caratteristiche personali, come altezza, peso, dimensione di un passo e gli obbiettivi. Infine, al tasto del fragment di inserimento delle misure ho legato, in seguito all'operazione di click, l'esecuzione di un intent che manda in un'activity nella quale viene mostrato il proprio indice di massa corporea, con 2 immagini che rappresentano lo stato del soggetto, in relazione al proprio indice di massa corporea. Oltre alle misure ed agli obbiettivi, ho dato la possibilità all'utente di scegliere il tempo che passa tra la fine di un suono e la riproduzione del suono successivo (distintamente per le due diverse modalità). Nella parte di codice associata ai fragment lato java ho inserito per lo più operazioni sulle sharedPreferences e molti controlli svolti prima di fare uso dei dati presenti negli EditText.
A seguire ripropongo due metodi di controllo degli editText utilizzati:

```
public boolean valutaDominio(){
    if(parseFloat(altezza.getText().toString())<2.5 &&
parseFloat(peso.getText().toString())<235 &&
parseFloat(lPasso.getText().toString())< 2.6 &&
(parseFloat(altezza.getText().toString())>0.2) &&
parseFloat(peso.getText().toString())>10 &&
parseFloat(lPasso.getText().toString())> 0.1){
        //con questi valori in un certo senso esagerati ho tenuto anche conto
della presenza di persone con malattie, bambini e per quanto riguarda i passi di
persone che corrono; gli estremi inferiori servono solo ad evitare lo zero che
annullerebbe tutti i conti
        return true;
    }
    else{return false;}
    }

public boolean contieneSpazzatura(String string){
    if(string.contains(" ")||string.contains(",") ||string.contains("-")){
        return true;
    }
    else{
```

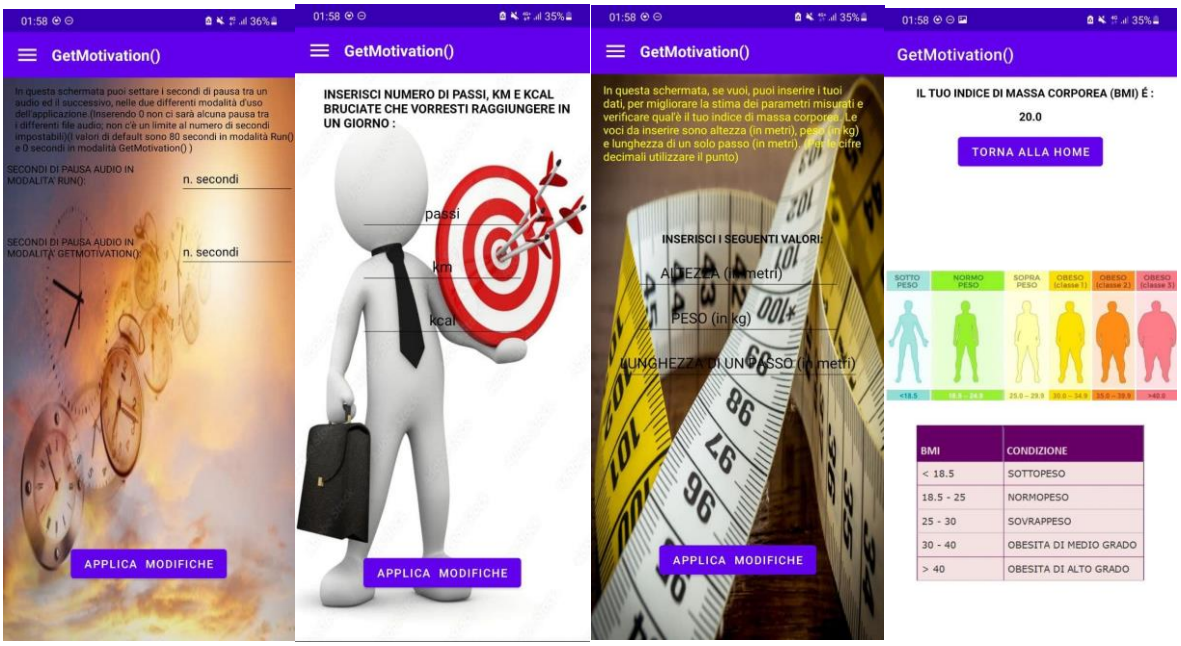
```
        return false;
    }
```

Ovviamente, se tutti i controlli andavano a buon fine, venivano salvati i valori nelle sharedPreferences e nel caso del fragment delle misure si passava all'activity relativa all'indice BMI.
Implementazione del metodo onClick per il fragment relativo all'inserimento delle misure:

```
@Override
            public void onClick(View view) {
        if(contieneSpazzatura((altezza.getText().toString())) ||
contieneSpazzatura(peso.getText().toString()) ||
contieneSpazzatura(lPasso.getText().toString())){
            Toast.makeText(getContext(), "Inserire valori numerici positivi!
(per le cifre dopo la virgola usare il punto)", Toast.LENGTH_SHORT).show();
        }
        else if (altezza.getText().toString().length() == 0 ||
peso.getText().toString().length()==0 || peso.getText().toString().length()==0)
{
            Toast.makeText(getContext(), "Inserire campi mancanti!",
Toast.LENGTH_SHORT).show();}

        else{
            if(!valutaDominio()){
                Toast.makeText(getContext(), "Attenzione! Forse stai inserendo
valori troppo alti o troppo bassi", Toast.LENGTH_SHORT).show();
            }
            else{
            SharedPreferences.Editor editor =
getContext().getSharedPreferences("save",MODE_PRIVATE).edit();
            editor.putFloat("altezza",
parseFloat(altezza.getText().toString()));
            editor.putFloat("peso", parseFloat(peso.getText().toString()));
            editor.putFloat("lunghezzaPasso",
parseFloat(lPasso.getText().toString()));
            editor.apply();
            Toast.makeText(getContext(), "Modifiche eseguite correttamente!",
Toast.LENGTH_SHORT).show();
            Intent a = new Intent(getContext(),TabellaBMIActivity.class);
            float bmi1
=parseFloat(peso.getText().toString())/(2*parseFloat(altezza.getText().toString(
)));
            double bmi = (double)bmi1;
            bmi = Math.ceil(bmi*1000)/1000;
            a.putExtra("valore BMI",bmi);
            startActivity(a);}
        }
    }
});
```

Infine, qui propongo alcune immagini di come compaiono le interfacce descritte:



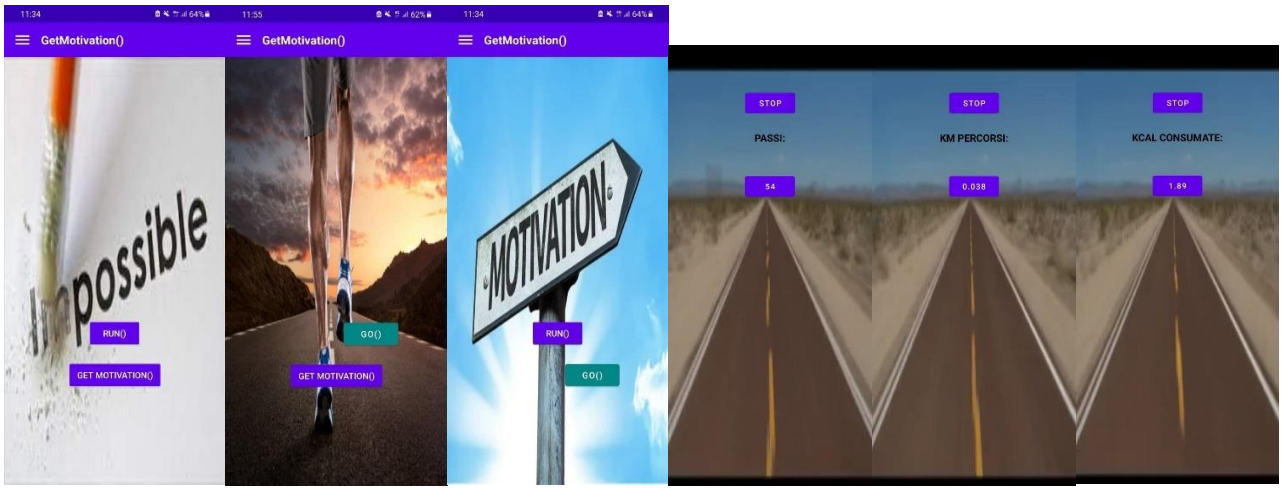
ORIENTAZIONE:

Siccome questa applicazione dovrebbe esser per lo più utilizzata all'interno del service, che rappresenta il cuore dell'applicazione, ho ritenuto non eccessivamente utile gestire i layout della modalità landscape ed ho fissato l'orientazione di tutte le Activity, in modalità portrait con una semplice istruzione:

```
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
```

HUMAN COMPUTER INTERACTION:

L'interazione dell'utente con il dispositivo, che permette la riproduzione dell'applicativo sviluppato, è stato l'aspetto al quale ho cercato di prestare più attenzione. Per rendere l'applicazione interessante, ho cercato di curare il più possibile il design, inserendo quasi sempre delle immagini di sfondo e rendendo cliccabile tutto ciò che si potesse cliccare. Ho introdotto il menu a tendina (DrawerLayout), proprio perché favorisce un'interazione più veloce tra i fragment. Nell'activity "ProvaSuoni", ho inserito l'opzione di click sullo sfondo, per visualizzare immagini differenti aggiunte in seguito. Un'operazione analoga è stata svolta nel fragment principale attaccato alla Home, necessario ad accedere alle due modalità presenti nell'app: in questo caso, ho sfruttato la visibilità dei tasti per migliorare quello che era il design dell'interfaccia. Infine, ho cercato di riprodurre le stesse operazioni sia nell'activity Run, che nel fragment finale, in cui vengono visualizzati i grafici; in queste interfacce cliccando su tasti o grafici è possibile modificare l'interfaccia, visualizzando features differenti, passi, km o kcal.
Qui di seguito ci sono alcuni esempi:



DESCRIZIONE APPROCCIO A DIVERSE PROBLEMATICHE INCONTRATE:

PROBLEMATICHE INCONTRATE:

-USO IMPROPRIO DI VARIABILI STATICHE:

Questo è stato l'aspetto che mi ha creato più difficoltà durante la programmazione. Un uso improprio di queste variabili, come detto a lezione, può risultare nocivo al programma (ed al programmatore). All'interno del programma, fatta eccezione per qualche stringa costante, inerente a database e file di Log, ho lasciato solo un oggetto statico, da utilizzare in lettura in diverse activity, una volta costruito. Questo oggetto è la lista di DataModel, nella quale per comodità ho inserito tutti gli indirizzi relativi alle risorse. Questa è stata una scelta di programmazione, per non dover istanziare più volte liste contenenti gli indirizzi delle risorse, però sono a conoscenza del fatto che, se si volesse permettere all'utente di aggiungere altre risorse (cosa a cui accennerò nell'ultimo paragrafo), questo approccio sarebbe pessimo perché non permette di modificare la lista durante l'uso del programma.

-SOLUZIONE:

Uso degli extra degli intent, uso di broadcast (per service) e quando realmente necessario uso di sharedPreferences.

-GESTIONE RILASCIO DEL SUONO ED AUDIOFOCUS PER UNA PLAYLIST DI SUONI

Questi aspetti trattati singolarmente, come nell'activity "ProvaSuoni", non sono particolarmente complessi. Però gestire una playlist, che in relazione ad un timer permette il passaggio dell'istanza dell'oggetto MediaPlayer da una risorsa audio a quella successiva, mi ha creato un po' di difficoltà.

-SOLUZIONE:

Rilasciare la risorsa MediaPlayer e l'audioFocus ogni volta che viene terminata la riproduzione di un suono ed imporre delle condizioni per cui non finiscano per sovrapporsi più istruzioni di rilascio del suono, prima che l'oggetto MediaPlayer sia associato ad una nuova risorsa audio.

-GESTIONE DATI UTENTI DIFFERENTI NEL DATABASE:

Un'operazione che avrei voluto fare sarebbe quella di creare tabelle differenti all'interno del db per ogni utente (credo esista un modo per farlo con i placeholder). Come si può vedere dal codice, ho cercato di passare il nome della tabella come parametro nei metodi del "DatabaseHelper", però alla fine ho optato per un approccio alternativo.

-SOLUZIONE:

Ho aggiunto alla tabella creata un attributo (la colonna "email") ed ho creato una query che mi permettesse di distinguere i vari utenti.

-GESTIONE PESO DELLE VIEW NEI DIFFERENTI FILE XML:

Questa operazione rispetto alle precedenti non rappresenta una vera problematica, però mi è sembrato un aspetto a cui dare rilevanza. Ritengo sia importante gestire questo aspetto, per permettere a qualsiasi dispositivo di utilizzare senza alcuna differenza l'applicazione.

-SOLUZIONE:

All'interno di tutti i Constraint layout (fatta escusione per la SlashActivity, nella quale ho mantenuto le dimensioni originali) ho aggiunto diversi LinearLayout ed attraverso l'attributo weightSum dei LinearLayout, ho adattato tutte le View a qualsiasi display. Come si può notare dal codice, in realtà ho annidato più LinearLayout, per permettere di mantenere inalterate le dimensioni dei bottoni e degli EditText, che altrimenti risulterebbero distorte.

(Ho testato i risultati in dispositivi con tutte le dimensioni possibili. Nel dispositivo di 2.7 pollici ho fatto in modo da far scomparire i tasti back di alcune activity, non da java, ma solo attraverso la scelta dei pesi. Questa scelta crea sui dispositivi di 3.2 pollici un effetto visivo leggermente sgradevole, perché nelle 3 activity di interesse non si legge il testo del tasto back. Sono a conoscenza del fatto che è possibile dal codice java risalire alle dimensioni dello schermo ed aggiungere una condizione per cui sotto i 3.7 pollici imposto la visibilità dei tasti in questione come "gone", però considerando che è un evento raro trovare telefoni odierni con display più piccoli di 3.7 pollici, ho sorvolato su questo aspetto, che comunque non influenza in alcun modo il funzionamento dell'applicazione)

IDEE PER MIGLIRARE L'APPLICAZIONE IN FUTURO:

Nello sviluppo dell'applicazione ho omesso alcune idee, che avrei voluto realizzare e che in caso si potrebbero aggiungere in futuro.

Innanzitutto, l'idea di base era quella di aggiungere anche altri tre "motivatori" nell'applicazione, questi erano: Steve Jobs, Gattuso e Steve Harvey. Ho omesso la presenza di queste persone, e mi sono limitato ai personaggi presenti sul logo, senza modificare nuovamente il codice, per non avere problemi nel rispettare le tempistiche relative alla data di consegna del progetto.

Avevo pensato pure di aggiungere un fragment di impostazioni, che permettesse di far scegliere all'utente di non visualizzare più il fragment di scelta iniziale (se ad esempio l'utente fosse solo interessato all'app come app di corsa). In questo modo, impostando un valore booleano contenuto nelle sharedPreferences a "true", avrei potuto con una semplice condizione saltare la MainActivity. Per fare questa operazione però bisogna prima di tutto mettere al valore "false" questo booleano in fase di logout e poi bisogna gestire l'alertDialog che blocca la modalità "GetMotivation()" in modo appropriato.

Un'altra operazione che avrei voluto svolgere (non in sede di esame), sarebbe stata quella di aggiungere un tasto con un'icona di aggiornamento nel fragment contenente la RecyclerView di scelta dei "motivatori", per far aggiungere all'utente qualsiasi "motivatore". Per far questo sicuramente andava cancellato l'approccio con cui dichiaro statica la lista di DataModel; rivisto questo aspetto e creati più vettori di accesso alle risorse, ho pensato genericamente con un approccio statico di aggiungere una ventina di viewHolder tutti uguali con stessa immagine e stessi suoni, non visibili, per poi modificare questi viewHolder in seguito al caricamento di dati da parte dell'utente. Per l'accesso agli indirizzi delle risorse avrei usato stavolta gli Uri, che ho testato durante l'uso delle immagini dell' "ActivityProvaSuoni" (infine ho lasciato quelle righe di codice commentate, perché in quel caso mi interessava modificare lo sfondo e non un'ImageView o un MediaPlayer). Per concludere, l'idea finale era quella di aggiungere anche un metodo di eliminazione dei dati di un singolo viewHolder aggiunto dall'utente; dunque avere un tasto con un'icona di aggiornamento che permetteva di accedere ad un popup menu con le due opzioni descritte sopra. Ovviamente sono consapevole del fatto che per aggiungere queste opzioni è necessario rivalutare ed eliminare gran parte del codice, poiché la scelta "comoda" che ho fatto di dichiarare statica la lista con i dati ha condizionato negativamente il paradigma di riuso del codice.

RIFERIMENTI

-https://developer.android.com/

-https://www.it-swarm.it/it/android-studio/

-https://stackoverflow.com/

-https://console.firebase.google.com/

-https://icons8.it/icons

-https://www.youtube.com/

-https://www.youtube.com/watch?v=9ARoMRd1kXo&t=167s (video FireBase)

-https://www.youtube.com/watch?v=iSsa9OlQJms (video FireBase)

-https://www.youtube.com/watch?v=6SrKOBV_hx8 (video Menu a tendina(DrawerLayout))

-https://www.youtube.com/watch?v=ZcWN-d3tTT4 (video uso Handler per Switch)

-https://www.youtube.com/watch?v=athzb2q-sWw (video AlertDialog)

-https://www.youtube.com/watch?v=C_Ka7cKwXW0 (video MediaPlayer)

-https://www.youtube.com/watch?v=bWBkBo4U3kg (video Audio focus e Broadcast audio rumoroso)

Modus operandi: L'iter che ho seguito nel fare uso dei riferimenti, soprattutto per approcciarmi a problematiche nuove, è stato quello di fare una ricerca generica sul web del nome di un metodo e di possibili esempi per svolgere un'operazione. A questo punto, se sussistevano ulteriori perplessità, dopo aver compreso il corretto uso dei parametri, visualizzavo da Android Studio la classe da cui proveniva il metodo ed infine ricercavo questa classe su Google Developer, per fare una ricerca più approfondita dell'argomento. In alternativa, per approcciarmi ad aspetti da approfondire maggiormente, come ad esempio il menu a tendina, la gestione del suono e l'implementazione di login, ho usato tutorial trovati su YouTube. Alcuni argomenti, come il menu a tendina, sono stati implementati seguendo fedelmente le istruzioni trovate (l'unica modifica che ho fatto è stata rendere quella che nel video è l'activity principale, un fragment attaccato alla home activity vuota, solo per personalizzare un minimo il codice e per comprendere se questo approccio fosse utilizzabile anche con fragment). Per quanto riguarda la gestione del login e tutto ciò che risulta inerente a Firebase, ho personalizzato un po' di più il codice ed ho fatto maggiormente uso di Google Developer, per la comprensione dei metodi della classe FirebaseAuth e dei diversi metodi che poi ho usato all'interno dell'applicazione.

(questo paragrafo è stato scritto quando ancora gran parte dell'applicazione doveva esser sviluppata, grossomodo esprime il modo con il quale mi sono approcciato alle problematiche)

ORLANDO ANTONIO VENEZIANO