Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

PROGETTAZIONE E SVILUPPO DI BPMWALKER: UN'APP MOBILE ADATTIVA PER PILOTARE LA FALCATA TRAMITE L'ASCOLTO DI MUSICA

Relatore:
Dott.
FEDERICO MONTORI

Presentata da:
EMANUELE FAZZINI

Sessione II
Anno Accademico 2019/2020

Sommario

Questa tesi parla dello sviluppo di un'applicazione mobile per il sistema Android chiamata BPMWALKER. Questa applicazione è pensata principalmente per un pubblico il cui scopo è poter raggiungere, camminando, un determinato luogo in un determinato tempo, quest'ultimi scelti dall'utente. Nel momento in cui l'utente ha scelto la destinazione e il periodo di tempo da impiegare per raggiungerla, verrà fatto partire un metronomo che, se installata ed é stata data l'autorizzazione all'applicativo, riprodurrà delle canzoni da una playlist Spotify, scelta dall'utilizzatore, che hanno un bpm il più vicino possibile a quello calcolato, data distanza, tempo e lunghezza del passo dall'applicazione; altrimenti verrà riprodotto il suono di un classico metronomo ad ogni battito. Questo per fornire la cadenza dei passi da seguire al consumatore.
Il mio progetto trova applicazione nell'ambito del context-aware computing, in quanto, per dare la massima precisione dell'andamento del passo da far seguire all'utente, ho realizzato funzionalità in grado di approssimare quanto più possibile la lunghezza del passo dell'utente. Tutto questo utilizzando i sensori presenti in uno smartphone. Il mio contributo alla realizzazione di questo progetto è stata l'implementazione dell'applicazione Android attraverso il quale il soggetto fruitore del software può avere accesso, possedendo uno smartphone con sistema operativo Android, a tali funzionalità. L'obbiettivo principale del mio studio è stato cercare un approssimazione il più possibile vicina alla vera lunghezza del passo dell'utente in modo tale da riuscire ad avere una quanto più perfetta computazione della cadenza del passo dell'utente durante la corsa.

Indice

Introduzione

In questi ultimi 13 anni, dalla nascita del primo smartphone come lo conosciamo oggi, ovvero l'iPhone, il mondo di questi dispositivi é totalmente cambiato. All'inizio gli smartphone davano la possibilità, oltre alle comuni operazioni di un telefono cellulare, di leggere la mail o navigare nel web, ma ad oggi sono diventati totalmente necessari per la vita di tutti i giorni.
Con l'utilizzo di questo dispositivo possiamo fare di tutto, dallo studiare, al lavorare e addirittura giocare, con qualità di prestazioni a livelli altissimi, per tutti i campi di applicazione prima citati. Con l'avvento di nuove tecnologie sia software che hardware, posseggono quasi tutte le caratteristiche di un vero computer, ma disponibile nel palmo di una mano e dall'utilizzo molto intuitivo.

In più, grazie all'incorporazione di sensori e grazie alle funzionalità all' avanguardia, é stato possibile costruire applicazioni eseguite dagli smartphone in grado di analizzare il contesto che le circondano e comportarsi di conseguenza. Tale concetto viene espresso con il termine "context-aware", ovvero consapevolezza del contesto.

È qui che entra in gioco la mia applicazione. Nel primo capitolo, "Stato dell'arte", ho cercato di dare un'introduzione al mondo delle applicazioni mobili e al context-awareness; dopodiché sono andato ad analizzare delle applicazioni che hanno una rilevanza nel mio studio in quanto posseggono delle funzionalità simili all'app da me sviluppata.
Nel capitolo successivo descrivo la mia applicazione in ambito di progettazione architetturale.
Mi soffermerò poi sull'implementazione delle varie funzionalità, di seguito divise per capitoli:

- Implementazione Generale: setup iniziale, implementazione del database e del menu;

- Implementazione della Navigazione, integrando le Google API;

- Implementazione del Metronomo, per passare all'utente l'esatta cadenza da seguire con relativa integrazione di Spotify;
- Regolazione Adattiva della Falcata.

Infine, i risultati e le conclusioni tratte dallo studio effettuato sulla determinazione della lunghezza del passo.

Capitolo 1

Stato dell'Arte

In questo capitolo andremo a vedere un'introduzione al mondo in cui la mia applicazione è correlata, ovvero il mondo degli smartphone e del context-awareness.

1.1 Applicazioni Mobili

L'avvento degli smartphone ha ribaltato completamente il mercato dei dispositivi di computazione dei dati, come ad esempio Laptop, Tablet e computer Desktop, come spiegato in questo articolo [1]. Gli smartphone, infatti, sono considerati oltremodo come la nuova generazione multifunzionale di telefoni cellulari che facilitano l'elaborazione dei dati e migliorano la connettività [2]. La necessità di essere interconnessi con gli smartphone alla rete è arrivata a tal punto che la rete cellulare copre il 96.8% della popolazione mondiale, mentre si arriva al 100% nei paesi industrializzati [3]. Per evidenziare il fatto che gli smartphone stanno conquistando il mercato dei dispositivi personali metto a disposizione in Fig. 1 un grafico preso da Google Trends che mostra le ricerche degli smartphone rispetto ad altri dispositivi quali: Laptop, Tablet e Computer Desktop [4].

Per capire la rapida espansione che hanno avuto gli smartphones, e di conseguenza le app mobile, è necessario riassumere un po' di storia di questi dispositivi. Il primo smartphone in assoluto fu sviluppato da IBM nel 1992 e venduto da BellSouth a partire dal 1993 ed aveva il nome di "Simon". Tale dispositivo oltre alle normali funzioni di un telefono cellulare incorporava della applicazioni come: il calendario, l'orologio, la posta elettronica, fax e giochi. Si poteva scrivere direttamente sullo schermo attraverso l'uso di un pennino.

Dopo Simon c'è stato un susseguirsi di dispositivi, tutti simili tra loro ma con qualche differenza o nell'hardware o nel sistema operativo. I principali produttori

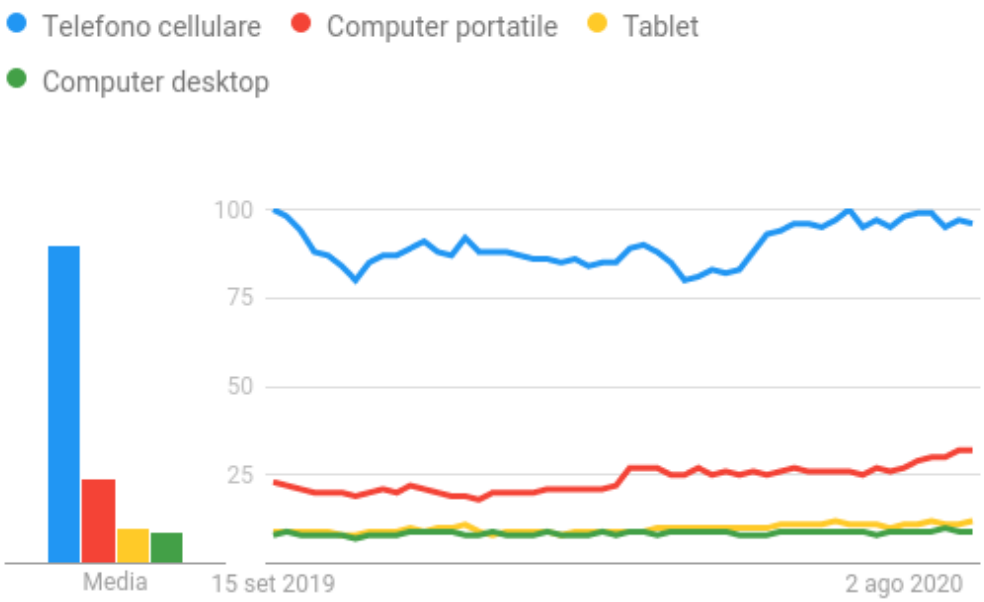


Figura 1.1: Sull' asse delle ascisse è rappresentato l'indice di gradimento del dispositivo in base alle ricerche fatte su di esso, sulle ordinate il periodo preso in considerazione, ovvero l'ultimo anno [4].

di smartphones sono stati: Nokia, HP, Sony e IBM; ma ciò che non riusciva a far esplodere il mercato delle applicazioni era il fatto che la maggior parte di queste case produttrici permetteva di eseguire sui propri dispositivi solo applicazioni proprietarie, fino a quando non sono arrivati gli smartphone come li conosciamo oggi, ovvero, con schermi capacitivi e con sistemi operativi Android e iOS. Il primo smartphone android fu prodotto da HTC nel 2008 mentre il primo iPhone con iOS nel 2007. Da questo momento in poi il mondo delle applicazioni ha avuto un successo molto grande; e questo fenomeno è dovuto anche grazie crescita smisurata dell'utilizzo di smartphone, dal quale sono nate milioni di app, ognuna diversa dall'altra, che usiamo abitudinariamente.

Ad oggi sul Play Store, il negozio virtuale di Google presente sui dispositivi Android, sono presenti più di 2,8 milioni di applicazioni al 2020; mentre nell'Apple Store, il negozio virtuale di applicazioni per iOS, ne sono presenti 2,2 milioni [5]. L'incremento della produzione di applicazioni mobile è dovuto anche al fatto che le applicazioni di oggi non sono più semplici applicazioni dove controllare la mail, impostare una sveglia o salvarsi un evento sul calendario com'erano fatte le prime app, oggi le applicazioni arrivano a livelli di sofisticatezza molto elevati. Questo perché l'avvento di nuovi dispositivi hardware e software ha dato la possibilità

agli sviluppatori di creare prodotti molto più all'avanguardia e complicati, basti pensare a giochi con qualità grafiche elevatissime, oppure servizi di geolocalizzazione e navigazione come Google Maps, oppure software di order-tracking. Tutte queste possibilità di sviluppo hanno fatto si che il mercato dei programmi mobile arrivasse ad avere milioni di applicazioni nel giro di un decennio.

1.2 Context-Awareness nel Mondo Mobile

1.2.1 Introduzione al contesto

Prima di introdurre il concetto di context-aware diamo un chiarimento alla parola contesto. Dalla definizione dell'Enciclopedia Treccani vediamo che "Il contesto può essere definito in generale come l'insieme di circostanze in cui si verifica un atto comunicativo. Tali circostanze possono essere linguistiche o extra-linguistiche." [6], quindi il contesto in cui ci troviamo sono l'insieme di eventi che ci circondano in modo tale da farci capire cosa sta accadendo in quel preciso momento.
Quando parliamo di contesto in una interazione con un computer la cosa si complica perché il computer, di base, non riesce a captare il contesto come riesce una persona mentre parla, o si interfaccia, con un altra persona. È qui che entra in gioco il context-awarenwss, ovvero la "consapevolezza del contesto" da parte dei computer.

I primi a parlare di context-aware furono Schilit and Theimer [7],dove spiegano che secondo loro il significato di contesto poteva essere racchiuso nella locazione della persona, nelle persone e negli oggetti che la circondano e nelle modifiche che vengono apportate a questi oggetti.
Ma questa spiegazione di contesto è troppo semplice, mentre il contesto rappresenta un concetto con molte più variabili e più astratto. In seguito A.K. Day introdusse una visione alternativa di contesto che, ad oggi, è l'esplicazione più usata per spiegare il termine contesto, ed è:

> *Il contesto sono tutte quelle informazioni che possono essere utilizzate per caratterizzare la situazione di un' entità. Un' entità è una persona, un luogo o oggetto che viene considerato pertinente per l' interazione tra un utente ed un applicazione, compreso l' utente e applicazioni stesse.* [8]

Applicando la definizione sopra citata al mondo degli smartphone possiamo dedurre che il contesto è qualsiasi informazione che può essere usata per descrivere l'uso giornaliero dello smartphone da parte dell'utente dovuto alle situazioni intorno a lui che lo influenzano [1].

1.2.2 Cos'è il Context-Aware

Ai giorni d'oggi il possesso di uno smartphone è diventato un bisogno. Ormai con essi possiamo fare tutto ciò che prima facevamo con un computer di base, come ad esempio leggere la mail o navigare sul web e sui social network, giocare e molto altro. Per questo motivo gli smartphone sono diventati la fonte di apprendimento del contesto primaria. Ad oggi quasi tutte le applicazioni che usiamo ci chiedono dei permessi per accedere, per esempio, alla nostra posizione o alla nostra attività fisica, per apprendere meglio il contesto in cui ci troviamo e fornirci al meglio i servizi che mettono a disposizione.
Molte applicazioni vogliono poter accedere al nostro contesto anche per fornirci delle esperienze personalizzate, come ad esempio un esperienza di gioco, oppure, una semplice pubblicità localizzata nel posto in cui ci troviamo. [1]
Relativamente alla definizione di context-aware computing Day da una spiegazione che di nuovo viene considerata migliore delle altre, ovvero:

Un sistema viene definito context-aware se utilizza le informazioni del context per fornire informazioni e/o servizi rilevanti all' utente, dove la rilevanza dipende dalle preferenze dell'utente e compiti che esso deve svolgere. [8]

Anche dal punto di vista dell'hardware che gli smartphone mettono a disposizione rispetto agli altri dispositivi, come ad esempio accelerometro, GPS, giroscopio, tutti sensori che danno capacità al dispositivo di capire cosa l'utente sta facendo e quindi di apprendere meglio il contesto in cui si trova.
Le app basate sul context-aware computing sono composte da un gran numero di regole del tipo: IF-THEN-ELSE, perché, per esempio, se il contesto è questo allora il comportamento dell'utente sarà di un certo tipo, altrimenti di un altro [9].

Il tempo è l'informazione contestuale più importante che va ad impattare sull'uso del dispositivo da parte dell'utente. [10] Questo perché le persone variano i propri comportamenti nel tempo, sia nel breve che nel lungo periodo, ed essendo il comportamento dell'utente ciò che bisogna captare tramite il contesto, il tempo rappresenta una variabile molto importante; ma in accordo con [11] la modellazione del comportamento basata sul tempo è un problema ancora irrisolto. La potenza dei sistemi context-aware ad oggi è davvero eccezionale. Ci sono alcuni studi come questo [12], nel quale è stata creata una rete in grado di riconoscere le emozioni umane tramite un sistema context-aware e un sistema di riconoscimento facciale. Viene anche spiegato che non basta il riconoscimento facciale, ma anche essere consapevoli del contesto è di fondamentale importanza.

Molto importante è anche rendere context-aware ciò che non lo è, perché come dice Dey [7]:

La consapevolezza del contesto da parte dei software aumenta notevolmente l'esperienza ce l'utente avrà nell'interagirci.

A tal proposito, in questo documento vi è riportato uno studio [13] in cui gli autori sono riusciti a rendere context-aware alcune applicazioni, che all'inizio non lo erano.

1.2.3 Caratteristiche di un'applicazione Context-Aware

Dal punto di vista delle caratteristiche che una applicazione context-aware deve avere, vari ricercatori hanno di nuovo provato ad elencarle, ma ancora una volta non sono riusciti nell'intento perché troppo legate e specifiche al compito dell'applicazione stessa. [14, 15]
In seguito Dey riesce a fornire dei punti chiave che un software context-aware deve rispettare per essere definito tale.
Ciò che propone Dey cerca di andare a combinare le idee che vi sono state precedentemente cercando di generalizzarle in modo tale che queste caratteristiche abbiano tutti i requisiti per tutte le applicazioni context-aware esistenti [8].
Le principali categorie di caratterizzazione a cui deve fare riferimento una applicazione context-aware sono:

- Presentazione di informazioni e servizi ad un utente.
- Esecuzione automatica di un servizio per un utente.
- Etichettatura del contesto alle informazioni per supportarne il successivo recupero.

Queste categorizzazioni, come si può osservare più avanti nel capitolo, vanno a conformarsi anche per le tecnologie nel mondo dell'IoT.

Qui di seguito ho riportato alcuni esempi di lavori correlati con il context-aware nel mondo mobile.
Questo studio [16] ha voluto provare ad usare un un modello Context-Aware Mobile Learning System (CAMLS) per introdurre la tecnologia nell'ambito Insegnare-Imparare, un evoluzione al modello mobile learning systems (MLS), in modo tale da migliorare l'user experience trmite l'introduzione del context-aware.

In questo report [17], invece, viene proposto uno studio sullo sviluppo di un'applicazione, chiamata POST-VIA 360, la quale dedicata a supportare l'intero ciclo di vita della fidelizzazione turistica dopo la prima visita. Una volta visitato per la prima volta un luogo, POST-VIA 360 é in grado di offrire raccomandazioni pertinenti basate sul posizionamento e sistemi di raccomandazione bio-ispirati. Questo grazie alla raccolta di dati, in seguito analizzati, da sistemi context-aware.

1.2.4 IoT e Context-Aware

A stretto contatto con il context-awareness computing troviamo il concetto di Internet of Things (IoT). Questo concetto fu coniato da Kevin Ashton [18]. L'IoT è un concetto che sta a rappresentare lo sviluppo di quelle tecnologie che tramite la comunicazione in internet con altri dispositivi riescono a scambiare ed elaborare tra loro le informazioni.
Grazie all'unione dell'IoT e del context-awareness computing, infatti, riusciamo ad ottenere delle informazioni sui nostri smartphone, come ad esempio la localizzazione GPS oppure le previsioni meteo, che ci permettono di usufruire di servizi che hanno una grande utilità per gli utenti. Naturalmente l'IoT non lo troviamo solo in dispositivi come smartphones o tablet, ma ormai viene applicato a tutti quegli apparecchi che ci circondano; Basti pensare ad un tostapane che può essere azionato con una applicazione, oppure programmato per attivarsi ad una certa ora, lo stesso vale anche per molti altri elettrodomestici e non solo.

1.3 Applicazioni Rilevanti

In questa sezione andrò ad elencare alcune app che permettono di compiere azioni simili alle funzionalità presentate nella mia applicazione.

1. **Calcolatore per la corsa** [19]. È un app disponibile sul Play Store che permette di misurare la lunghezza del passo inserendo il tempo e la cadenza dei passi tenuta per compiere un chilometro. Oltre a questa ha anche altre funzionalità di calcolo tutte relative alla corsa.

2. **StepsApp** [20]. Applicazione mobile disponibile sia sul Play Store che su Apple Store. Software contapassi che, tra le sua funzionalità, ha anche la possibilità di misurare la lunghezza di un passo tramite la moltiplicazione dell'altezza per un numero che varia dal sesso dell'utente.

3. **RunBeat for Spotify** [21]. Applicazione mobile disponibile sul Play Store che permette di scegliere un determinato bpm da seguire per la corsa e far partire, o dalla playlist Running di Spotify, o da una delle playlist che si seguono, le canzoni che più si avvicinano a quel bpm in modo tale da correre seguendo il ritmo delle canzoni.

4. **PaceDJ** [22]. È un applicazione disponibile su Apple Store e Play Store che permette di scegliere un bpm e far eseguire determinate canzoni con quel bpm presenti nella libreria iTunes personale oppure in una playlist personale. Questo per far si che avendo un allenamento intensivo la musica si adatti all'allenamento e viceversa.

Oltre alle applicazioni presentate ve ne sono altre, ma con minore corrispondenza nei confronti della mia o con minore rilevanza rispetto a quelle elencate sopra.

Confrontando le app ivi riportate posso affermare che la mia permette di interagire con delle funzionalità che non sono mai state implementate. Ad esempio: la prima applicazione permette di calcolare la lunghezza del passo inserendo i minuti e i secondi impiegati per percorrere un chilometro e la cadenza dei passi, mentre la seconda sfrutta i sensori posti nello smartphone senza tenere conto della distanza percorsa. La terza e quarta applicazione riguardano un altro aspetto, ovvero scegliere dei brani da riprodurre dato un certo BPM che la canzone deve avere; in questo caso, però, tali app richiedono l'immissione da parte dell'utente di un certo BPM per poi andare a scegliere i brani da far partire.

Le motivazioni che mi hanno spinto a voler sviluppare questo progetto sono state:

- La possibilità di poter sviluppare un progetto completamente da solo e partendo da zero.
- Trovare utili le funzionalità che mi sono state proposte

- Poter compiere uno studio su delle funzionalità che, singolarmente sono già state proposte come visto sopra, ma non sono mai state implementate in modo complementare.

Riguardo al primo punto sono convinto che aver creato un progetto ed averlo portato a termine da solo mi abbia portato a lavorare meglio e a guardare tutti gli aspetti che il progetto dovesse avere, dalle funzionalità tecniche ai dettagli estetici e sicuramente mi ha aiutato a crescere nell'ambiente dello sviluppo di applicazioni mobile per Android.
Trovo che le questa applicazione possa essere utile per quei soggetti che, ad esempio, hanno la necessità di arrivare in un determinato luogo in un tempo limite e hanno bisogno di conoscere la cadenza dei passi da seguire per raggiungere la destinazione entro il periodo temporale prescelto.
Ed infine questo progetto mi ha dato la possibilità di compiere uno studio su delle funzionalità, che non sono mai state sviluppate prima.

Capitolo 2

Progettazione Architetturale

In questo capitolo inizierò a trattare la struttura architetturale della mia applicazione. La struttura che sta alle fondamenta é formata da un'Activity chiamata MainActivity su cui è stato costruito un Navigation Drawer per navigare tra i vari Fragment che si andranno ad intercambiare nella schermata iniziale sotto la Toolbar in cui è presente il tasto menu per aprire il Navigation Drawer, oppure basta fare uno scorrimento da sinistra verso destra per aprirlo.

2.1 Visione Generale

La mia applicazione è stata pensata per dare la possibilità all'utente, ad esempio in ritardo per un determinato appuntamento da dover raggiungere a piedi, di poter scegliere un luogo di destinazione sulla mappa e raggiungerlo nel tempo che ha a disposizione per arrivarci, inserendolo, in modo tale che l'app poi sia in grado di stabilire l'andatura necessaria da seguire per darla in input all'utente sotto forma di canzone o di metronomo. Ho, in seguito, implementato un sistema di regolazione autonoma della falcata in modo tale da rendere i calcoli il più precisi possibile al momento della computazione dei passi per minuto al quale attenersi.

Uno schema riassuntivo della mia architettura é riportato qui di seguito. In questo schema troviamo due Activity principali, la MainActivity, che viene lanciata all'avvio dell'applicazione, l'InitialActivity che viene aperta solo al primo avvio dalla MainActivity. Dopodiché possiamo osservare i vari componenti dal quale sono formate. L'InitialActivity dai vari Fragment in cui l'utente inserirà i suoi dati iniziali. La MainActivity dal Navigation Drawer Menu che va a prendere i dati dell'utente per inserirli nell'Header e le varie voci di menu, 5 delle quali sono i vari Fragment associati, ad esempio, alla mappa, alla scelta della playlist Spotify dal quale riprodurre i brani, alla possibilità di cambiare i propri dati personali e alla

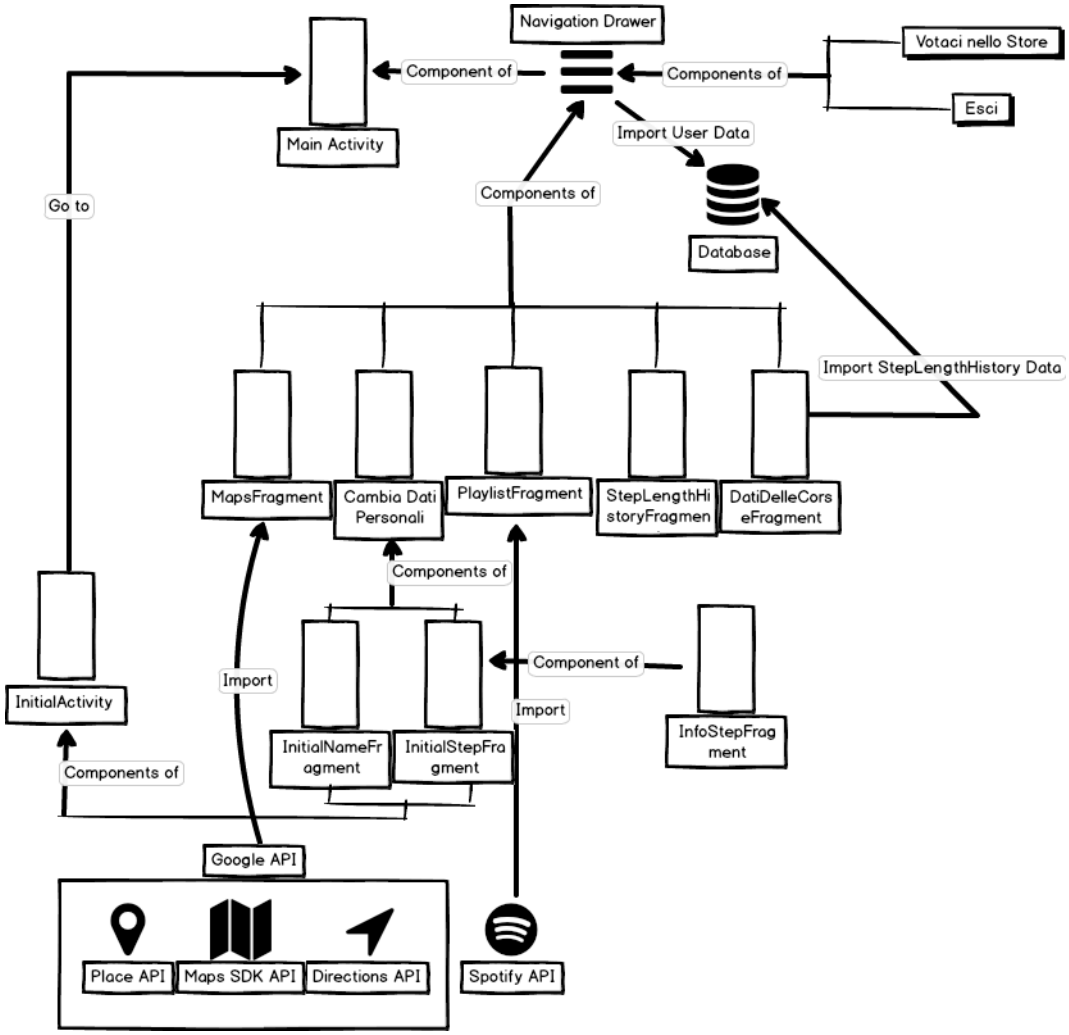


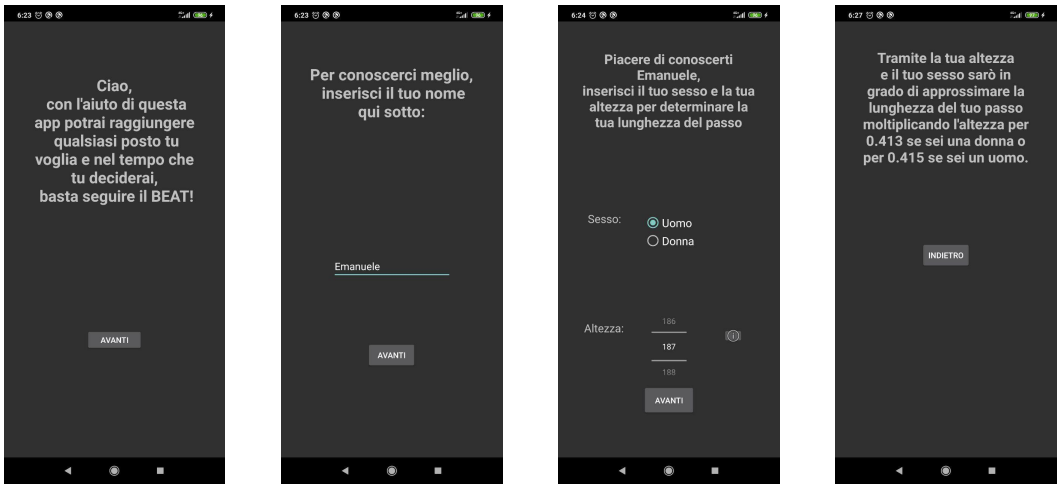
Figura 2.1: Schema architteturale dell'applicativo

visione dei dati riportati nel database. Infine è riportato il fatto che MapsFragment, ovvero la schermata della mappa, e PlaylistFragment, ovvero la voce di menu in cui scegliere la playlist, importando rispettivamente le Google API e le Spotify API per svolgere le loro funzioni.

2.2 Initial Setup

Al primo avvio dell'applicazione si aprirà un'altra Activity, nominata InitialActivity, che darà un introduzione all'utente sulla applicazione e gli chiederà di inserire dei dati iniziali, nome, sesso e altezza. L'altezza servirà per andare a

calcolare poi, approssimativamente la lunghezza del passo; di fianco al NumberPicker per inserire l'altezza vi è posto un InfoButton che, se premuto, aprirà un terzo Fragment nel quale viene spiegato come si ottiene per approssimazione la lunghezza di un passo dal sesso e dall'altezza, ovvero si è un uomo l'altezza verrà moltiplicata per 0,415, mentre se si è una donna verrà moltiplicata per 0,413 [23].



InitialSetupFragment InitialNameFragment InitialStepFragment InfoStepFragment

Figura 2.2: Da sinistra verso destra troviamo i vari step da seguire che compongono il setup iniziale.

2.3 Home Fragment

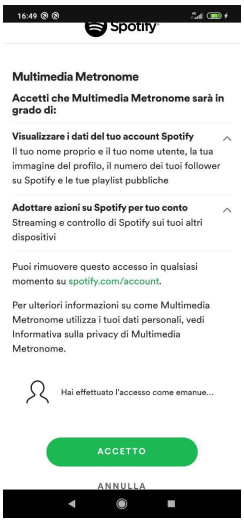


Figura 2.3: Spotify installato

Una volta premuto sul pulsante avanti del terzo Fragment, si verrà reindirizzati alla Home dell'applicazione, ma prima che venga visualizzato il Fragment con la mappa corrispondente alla Home, verrà richiesto l'accesso a Spotify. Tale accesso potrà essere eseguito in 2 modi distinti, se si ha l'applicazione installata e con l'accesso eseguito verrà aperta una finestra come quella mostrata in Fig. 2.3 dove basterà premere "ACCETTO" ed il proprio account verrà sincronizzato, altrimenti si aprirà una WebPage dove eseguire l'accesso, come viene mostrato in in Fig. 2.4.

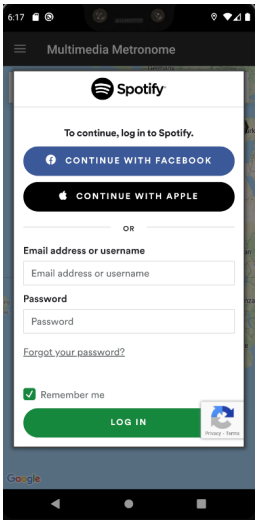


Figura 2.4: Spotify non installato

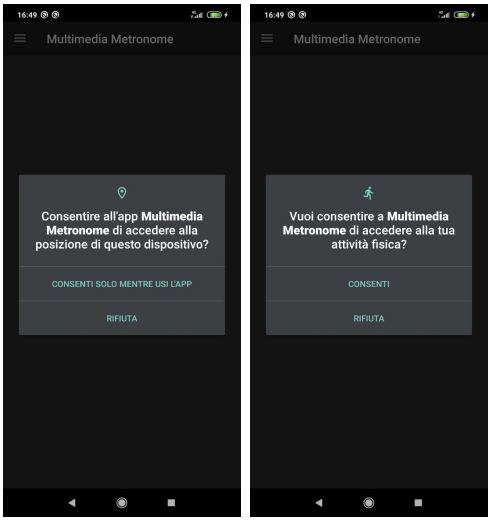


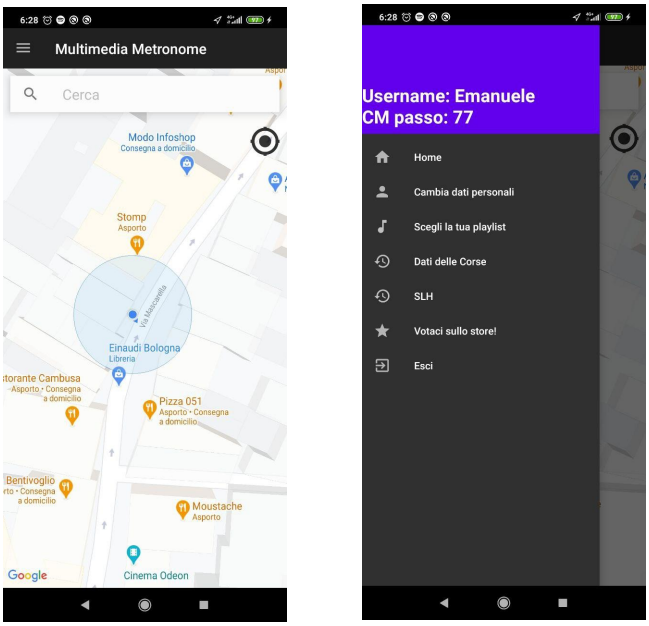
Figura 2.5: Richiesta dei permessi per accedere alla posizione e all'attività fisica

Successivamente verranno richiesti i permessi per poter accedere alla propria posizione e alla propria attività fisica; questi ultimi devono essere richiesti dalla versione 10 di Android e serviranno per dare la possibilità all'app di usare il sensore contapassi del telefono, se esso ne è dotato. Dopodiché, il Fragment associato alla Home è il MapsFragment, ovvero il Fragment in cui è presente la mappa di GoogleMaps, implementata tramite le GoogleMaps API e spiegata nel capitolo successivo. Da qui si potrà scegliere una destinazione tenendo premuto un punto sulla mappa in cui apparirà un marker, oppure, scrivendo la destinazione nella barra di ricerca con autocompletamento.

Nel momento in cui l'utente sceglierà una destinazione sulla mappa, insieme al marker apparirà in basso a sinistra l'immagine stilizzata di un uomo che cammina con scritto sopra "START". Appena si preme su questa immagine verrà creata la Polyline da seguire per arrivare alla destinazione e un Fragment, chiamato MinuteFragment, sovrapposto alla mappa che chiede all'utente di indicare il tempo che desidera impiegare per percorrere la distanza. Il tempo da impiegare si può inserire in due modi:

- Inserendo manualmente i minuti da impiegarci.
- Scegliere un orario di arrivo a destinazione.

Inserendo manualmente i minuti, l'orario di arrivo verrà ignorato.
Una volta scelto il tempo premendo sul pulsante "PARTI" verrà fatto partire un metronomo che, se si ha Spotify installato e si è eseguito l'accesso a Spotify tramite l'app, riprodurrà delle canzoni dalla playlist scelta nel Fragment PlaylistsFragment che più si avvicinano al bpm calcolato in base alla distanza da percorrere, alla lunghezza del passo e al tempo scelto. Nel caso in cui non si sia fatto l'accesso e/o non si ha installato Spotify sul proprio dispositivo verrà riprodotto il suono di un classico metronomo. Premendo invece sul pulsante "ESCI" MinuteFragment si chiuderà e per riaprirlo basterà premere di nuovo sull'immagine "START". All'inizio della schermata di MinuteFragment vi sarà anche scritto quant'è lo spazio



Home Navigation Drawer

Figura 2.6: Sopra riportati vi sono le immagini relative alla Home ed al Menu presenti nell'app.

da percorrere.
Nel momento in cui il metronomo verrà avviato l'immagine dell'uomo stilizzato che cammina diventerà l'immagine di un uomo stilizzato che sta fermo con su scritto sopra il capo "STOP". Premendoci sopra, verrà fermato il metronomo. Può essere anche fermato premendo "STOP" nella notifica che apparirà in Foreground

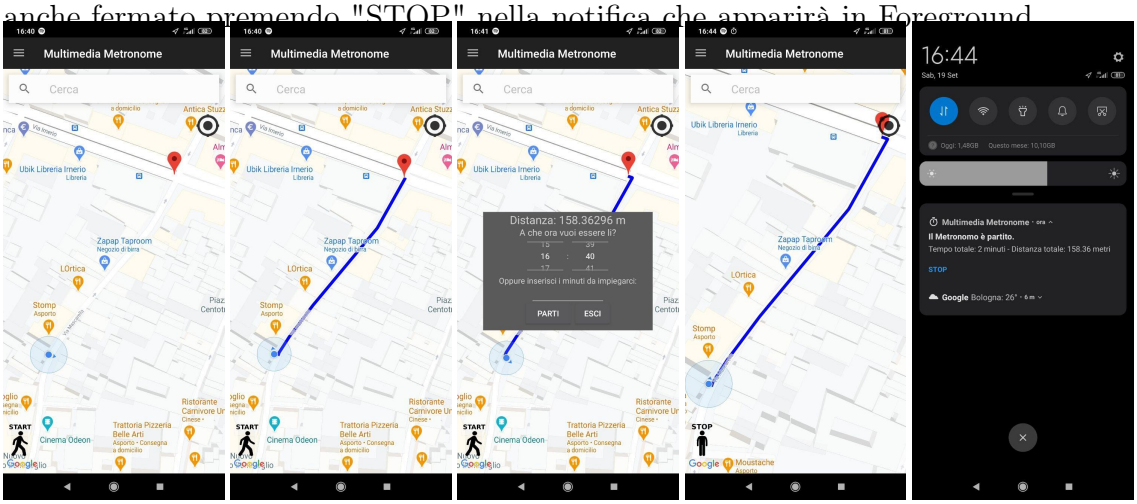


Figura 2.7: Da sinistra verso destra é rappresentato l'iter per poter iniziare una camminata.

2.4 Navigation Drawer

Le voci di menu presenti oltre alla Home sono:

- Cambia dati personali
- Scegli la tua playlist
- Dati delle corse
- SLH
- Votaci sullo store
- Esci

Di seguito riporterò il comportamento di ogni singola voce di menu.

2.4.1 Cambia Dati Personali

Premendo sulla voce di menu "Cambia dati personali" verrà aperto l'InitialNameFragment, dal quale poter andare a cambiare le informazioni inizialmente salvate riguardanti i dati dell'utente. Verranno salvati solo nel momento in cui cambieremo tutti i dati, quindi non si può cambiare solo il nome o solo l'altezza, per esempio. Tale voce di menu può servire, ad esempio, nel caso in cui un utente diverso da quello iniziale volesse provare ad utilizzare l'applicazione. Può farlo semplicemente recandosi in questa sezione e andando, con due semplici passaggi, a cambiare i dati necessari per far funzionare l'app.
Come possiamo notare, la differenza dagli screenshot fatti sopra è la Toolbar in alto, proprio perché qui ci troviamo sulla MainActivity mentre nel setup iniziale ci troviamo nella InitialActivity.

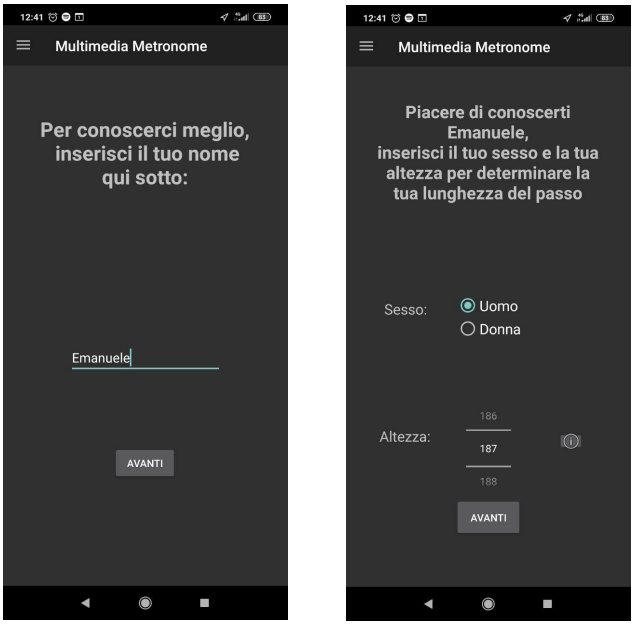


Figura 2.8: Voce del menu: "Cambia dati personali".

2.4.2 Scegli la tua Playlist

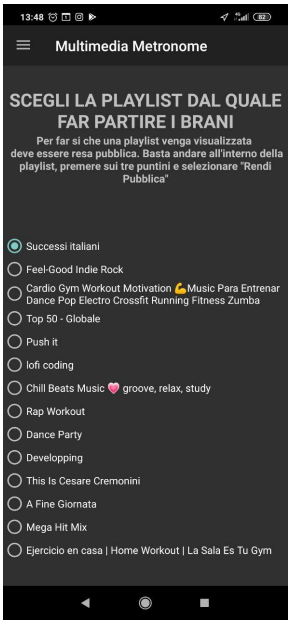


Figura 2.9: Voce del menu: "Segli la tua playlist".

In questa voce di menu l'utente, se ha effettuato l'accesso a Spotify nel momento in cui gli è stato chiesto, visualizzerà tutte le playlist di Spotify da lui create e da lui seguite sull'app. Ad ogni playlist sarà associato un RadioButton che rappresenterà la playlist selezionata per il filtraggio dei brani da riprodurre in base al bpm calcolato nel momento in cui si fa partire il metronomo. Nel caso in cui l'utente non avesse effettuato l'accesso, non verrà stampata una lista, bensì una frase con scritto "No results found". La lista dei RadioButtons è stata inserita in una ScrollView in modo tale che se le playlists fossero più di quelle visibili dalla schermata dell'applicazione, basterà scrollare la lista per visualizzarle tutte.

Figura 2.10: Voce di menu: "Dati delle corse"

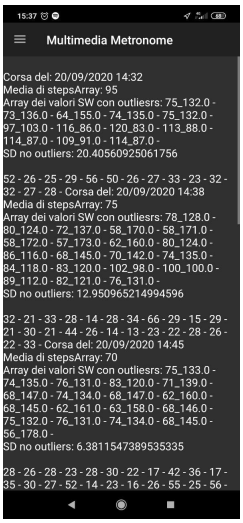


Figura 2.11: Voce di menu: "SLH"

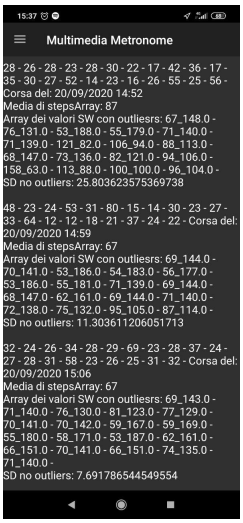


Figura 2.12: Voce di menu: "SLH"

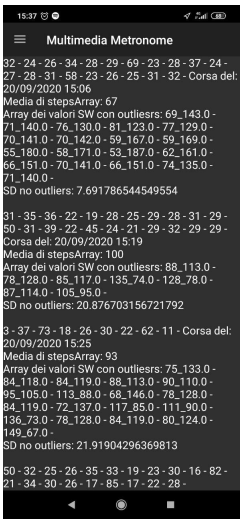


Figura 2.13: Voce di menu: "SLH"

2.4.3 Dati delle Corse

In questo Fragment sono riportati, sotto forma di stringhe, i dati contenuti nei record del database corrispondenti alla tabella StepLengthHistory. Esempio riportato in Fig. 2.10. Fragment utilizzato solo per visualizzare i LOG restituiti dall'applicazione. Nel software che sarà distribuito all'utente non sarà presente o, se lo sarà, avrà funzionalità diverse.

2.4.4 SLH

SLH sta per Step Length History. In questa sezione si trova una ScrollView con all'interno una TextView che riporta i dati scritti su un file, più precisamente il nome del file è datiCorse.txt, dove sono riportati i dati relativi ad ogni corsa così da poter essere studiati e migliorati.
Per ogni corsa sono riportati:

- Data e ora della corsa.
- La media della lunghezza del passo che é stata tenuta per quella corsa.
- Un array di valori dove in ogni cella è rappresentato: la lunghezza del passo in quei 100 metri_i passi compiuti in quei 100 metri.

- Un array in cui in ogni posizione corrisponde ai passi compiuti ogni 20 metri della camminata totale. Quindi alla posizione i troveremo il numero di passi fatti dalla distanza i-1*20 alla distanza i*20.
- La deviazione standard calcolata sulle lunghezze dei passi di ogni cella dell'array precedente togliendo gli outliers.

Fragment utilizzato solo per visualizzare i LOG restituiti dall'applicazione. Nel software che sarà distribuito all'utente non sarà presente o, se lo sarà, avrà funzionalità diverse. Esempi riportati in Fig. 2.11, Fig. 2.12 e Fig. 2.13.

2.4.5 Votaci sullo store

Questa voce di menu è semplicemente un Intent al playstore che visualizzarà, nel momento in cui sarà caricata, la pagina corrispondente all'applicazione all'interno dello Store.

2.4.6 Esci

Infine, come ultima voce di menu, troviamo "Esci" che, se premuta, riporta alla Home del proprio dispositivo, mettendo l'applicazione in background.

Capitolo 3

Implementazione Generale

3.1 Implementazione Setup Iniziale

Come applicazione predefinita nel manifest, all'avvio verrà aperta la MainActivity, che, però, eseguirà subito un controllo su una variabile booleana inserita all'interno delle SharedPreferences. Se questa variabile risultasse true, verrebbe richiamato un Intent verso L'initialActivity. Come visto sopra è composta da tre Fragment, nel quale, all'interno del terzo Fragment nel momento in cui si clicca sul pulsante Avanti del Fragment InitialStepFragment verrà assegnata a false la variabile "init" in modo tale che l'Intent alla InitialActivity non sarà più richiamato. Inoltre, quando il pulsante "AVANTI" di InitialStepFragment viene attivato sarà calcolata la lunghezza del passo e in seguito verranno salvati sul database il nome e tale lunghezza tramite un AsyncTask. Dopodiché verrà aperta la MainActivity, tramite un Intent, impostata per default sul'HomeFragment, ovvero la mappa.

3.2 Navigation Drawer

Per l'implementazione di un menu all'interno dell'app ho scelto un Navigation Drawer, ovvero un menu a scomparsa, sulla sinistra o sulla destra, che é possibile richiamare facendo uno scorrimento oppure premendo sull'apposito pulsante nella Toolbar in alto.
L'implementazione di tale componente è divisa in 3 passaggi:

- Creazione del file menu.xml in cui sono andato ad inserire tutte le relative voci che andranno a comporre il menu.

- Creazione della componente Toolbar per avere una barra in alto in grado di avere un bottone per richiamare il menu.

- Creazione dell'header del mio Navigation Drawer, dove sono andato ad inserire una TextView che mostrerà i dati salvati dall'utente nel database.

- Inclusione della Toolbar nel layout xml della MainActivity e creazione di una NavigatioView in cui sono andato ad inserire il layout dell'header precedentemente creato e ad includere il file menu.xml-

- Far implementare alla MainActivity l'interfaccia NavigationView.OnNavigationItemSelectedListener.

- Infine nella MainActivity.java vado ad inizializzare e a stabilire i comportamenti della Navigation Drawer, ovvero: vado ad assegnare i Fragment relativi alle voci di menu nel metodo onNavigationDrawerItemSelected(); creo il toggle per l'apertura e la chiusura del menu e lo vado ad assegnare alla Toolbar ed infine stabilisco la Home come voce di menu predefinita in modo tale che venga aperta lei come prima schermata.

Il risultato finale lo possiamo osservare nello screenshot pubblicato nella sezione Home Fragment.

3.3 Room Database

Per salvarmi i dati, come ad esempio il nome e la lunghezza del passo dell'utente, oppure, come vedremo più avanti, i vari dati relativi ad ogni camminata per il calcolo della falcata, ho scelto di implementare un database attraverso il framework Room che fornisce un'astrazione di SQLite in modo tale da rendere più fluido e meno complesso l'utilizzo di SQLite [24].
Tale database viene implementato nella cartella database del progetto, nel quale troviamo 5 file:

- User.java : è la classe java corrispondente ai dati dell'utente, ovvero nome e lunghezza del passo.

- StepsLengthHistory.java : corrisponde al formato della tabella nel database che userò per salvarmi i dati delle corse compiute, ovvero: l'id della corsa che è ad incremento automatico,la media della lunghezza del passo tenuta per quella corsa e la deviazione standard di quella corsa entrambi relativi ai metri percorsi e ai passi compiuti.

- MyDBAccessObjectSH.java : interfaccia DAO (Database Access Object) relativa alla tabella StepsLengthHistory. Qui sono dichiarate le varie query da poter chiamare per interagire con questa tabella.

```
    @Insert
    public void addStep(StepsLengthHistory slh);

    @Query("select * from StepsLengthHistory")
    public List<StepsLengthHistory> getSLH();
```

- MyDBAccessObjectUser.java : interfaccia, rappresenta il DAO (Database Access Object) relativo alla tabella User. Qui sono dichiarate le varie query da poter chiamare per interagire con questa tabella.

```
    @Insert
    public void addUser(User u);

    @Query("select * from User")
    public List<User> getUser();

    @Query("delete from User")
    public void removeOldUser();
```

- MyDatabase.java : classe astratta in cui sono elencati gli oggetti DAO sopra elencati in modo tale da accedervi in seguito tramite una variabile MyDatabase inizializzata nella MainActivity.

```
public abstract class MyDatabase extends RoomDatabase {

    public abstract MyDBAccessObjectsUser mydao();
    public abstract MyDBAccessObjectsSH mydaoSH();

}
```

Una volta create queste classi posso andare ad istanziare un oggetto mydb di tipo MyDatabase nella MainActivity che userò per andare a invocare le chiamate al database tramite i DAO.

```
mydb = Room.databaseBuilder(getApplicationContext(),
            MyDatabase.class,"User").build();
```

Capitolo 4

Implementazione della Navigazione

4.1 Google Maps SDK per Android

Per la creazione della mappa ho optato per Google Maps.
Tramite le Maps SDK API è possibile importare la mappa di Google Maps sulla propria applicazione. Per fare ciò basta andare sul sito di Google Cloud Platform [25], accedere e abilitare il proprio account come developer, creare un nuovo progetto e scegliere le API di cui abbiamo bisogno, nel mio caso le Maps SDK API per Android, le Places API e le Direction API. Riceveremo una chiave univoca da parte del sito che poi andremo ad utilizzare per integrare le API attivate sul sito nella nostra applicazione.
Una volta ricevuta la chiave ci sposteremo su Android Studio dove creeremo un nuovo Fragment o una nuova Activity di tipo Google Maps. Verrà automaticamente creato un file google_maps_api.xml nella cartella res > values del mio progetto dove andremo ad inserire la API key ricevuta poco fa.

```
<resources>
    <string name="google_maps_key"
        templateMergeStrategy="preserve" translatable="false">
            AIzaSyDGICldFs263W4ZmsiPAZc2rFwM7QJofo8
    </string>
</resources>
```

Il Fragment in cui ho inizializzato la mappa nel mio progetto si chiama MapsFragment. In esso troviamo tutti i metodi necessari per inizializzare e interagire con la mappa.
Per prima cosa nel file xml del layout, ovvero frament_maps.xml, sono andato ad inserire il Fragment statico che sarà il componente della schermata in cui verrà caricata e visualizzata la mappa.

```
<fragment
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MapsFragment"
    android:gravity="center"/>
```

In seguito sono andato a scrivere il codice java nel MapsFragment.java necessario alla creazione e visualizzazione della mappa.
Nell'onStart() controllo che siano stati dati i permessi di localizzazione e, in caso affermativo richiamo il metodo initMap(), un metodo che crea un SupportMapFragment(forse da citare) e lo va ad assegnare al Fragment statico posizionato prima nel fragment_maps.xml.
Dopodice l'oggetto SupportMapFragment chiamato mapFragment andrà a fare una chiamata asincrona al metodo onMapReady() che, nel mio caso andrà a compiere queste operazioni:

- Assegnare ad una variabile, myMap, l'oggetto che rappresenta la mappa.
- myMap.setMyLocationEnabled() serve per far comparire la mia posizione sulla mappa con il classico "pallino blu".
- Richiamo il metodo getDeviceLocation() per avere la posizione attuale, che vedremo nella sezione più avanti.
- Richiamo il metodo initializeAutoCompleteFragment(), funzione per andare ad inizializzare la barra di ricerca con autocompletamento dei luoghi, che vedremo più avanti.
- Richiamo il metodo gpsImageOnClick(), metodo che va a creare una nuova immagine che, se premuta, muove la camera della mappa sulla posizione attuale.
- ed infine con l'ultimo metodo myMap.getUiSettings().setMyLocationButtonEnabled(false) vado ad eliminare il pulsante di default di Google Maps che ha la stessa funzionalità di quello del pulsante creato da gpsImageOnClick().

Il metodo gpsImageOnClick(), initMap() e onMapReady() sono riportati di seguito.
Nel primo metodo quello che faccio è andare a controllare tramite il metodo isLocationEnabled() se il GPS del dispositivo è accesso e, in caso affermativo, controllo che la posizione corrente sia diversa da null e se così eseguo il moveCamera della visuale della mappa sulla posizione corrente.

```
@Override
public void onStart() {
    super.onStart();
    if (ActivityCompat.checkSelfPermission(getActivity(),
            Manifest.permission.ACCESS_FINE_LOCATION) ==
                    PackageManager.PERMISSION_GRANTED
        && ActivityCompat.checkSelfPermission(getActivity(),
                Manifest.permission.ACCESS_COARSE_LOCATION) ==
                    PackageManager.PERMISSION_GRANTED)
        initMap();
}
```

```
private void initMap() {
    SupportMapFragment mapFragment = (SupportMapFragment)
        getChildFragmentManager().findFragmentById(R.id.map);
    if (mapFragment != null) {
        mapFragment.getMapAsync(this);
    }
}
```

```
@Override
public void onMapReady(GoogleMap googleMap) {
    if (ActivityCompat.checkSelfPermission(getActivity(),
        Manifest.permission.ACCESS_FINE_LOCATION) ==
            PackageManager.PERMISSION_GRANTED
        && ActivityCompat.checkSelfPermission(getActivity(),
            Manifest.permission.ACCESS_COARSE_LOCATION) ==
                PackageManager.PERMISSION_GRANTED) {
        myMap = googleMap;
        mapForActivity = googleMap;
        myMap.setMyLocationEnabled(true);
        getDeviceLocation();
        initializeAutocompleteFragment();
        clickForMarker();
        gpsImageOnClick();
        myMap.getUiSettings().setMyLocationButtonEnabled(false);
    }
}
```

```
private void gpsImageOnClick(){
    imageViewGPS.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            isLocationEnabled();
            if(currentLocation != null)
                moveCamera(
```

```
                        new LatLng(currentLocation.getLatitude(),
                                   currentLocation.getLongitude()),
                        DEFAULT_ZOOM, "My Location");
            }
        });
}
```

```
private void gpsImageOnClick(){
    imageViewGPS.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            isLocationEnabled();
            if(currentLocation != null)
                moveCamera(
                    new LatLng(
                        currentLocation.getLatitude(),
                        currentLocation.getLongitude()),
                    DEFAULT_ZOOM, "My Location");
        }
    });
}
```

4.2 Geolocalizzazione

Per la geolocalizzazione ho scelto di usare il FusedLocationProviderClient (forse da citare). Per prima cosa vado ad inizializzare, nella onCreate(), un oggetto di tipo LocationRequest(forse da citare), che servirà per sancire ogni quanto tempo l'applicazione dovrà chiedere l'aggiornamento della posizione; tramite il metodo getLocationCallback, invece, vado a creare la callback per dire all'applicazione cosa fare quando riceve gli aggiornamenti della posizione ed infine, prima controllo che i permessi di localizzazione siano stati accettati, poi vado ad istanziare l'oggetto fusedLocationProviderClient che provvederà a restituirmi l'ultima posizione aggiornata del dispositivo, nel mio caso, ogni 2 secondi.

```
locationRequest = LocationRequest.create();
locationRequest
        .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)
        .setInterval(2000)
        .setFastestInterval(1000)
        .setMaxWaitTime(1000);
getLocationCallback();
fusedLocationProviderClient = LocationServices
    .getFusedLocationProviderClient(getContext());
```

Una volta creato l'oggetto che mi da la possibilità di conoscere la posizione ogni 2 secondi, con il metodo getDeviceLocatio() creo una Task(forse da citare) chiamata location a cui assegno la funzione fusedLocationProviderClient.getLastLocation(). Creo la callbak alla task e al completamento assegnerò il risultato della task, ovvero la posizione attuale, ad una variabile locale di nome currentLocation, in modo tale da poterci poi eseguire delle funzionalità, come ad esempio il gpsImageOnClick().

```
private void getDeviceLocation(){
    try {
        final Task location = fusedLocationProviderClient
                                .getLastLocation();
        location.addOnCompleteListener(new OnCompleteListener() {
            @Override
            public void onComplete(@NonNull Task task) {
                if(task.isSuccessful()){
                    currentLocation = (Location) task.getResult();
                }
            }
        });
    } catch (SecurityException e){
        e.printStackTrace();
    }
}
```

Per controllare invece se il GPS è attivo, creo un istanza di un oggetto Location-Manager nella onCreate() e, nel metodo isLocationEnabled, controllo che il GPS sia attivo, se così non fosse faccio apparire un Dialog(forse da citare) dove spiego all'utente che servono i servizi di localizzazione attivi per fruire delle funzionalità dell'applicazione.

```
private void isLocationEnabled() {

    if(!locationManager.isProviderEnabled(LocationManager
            .GPS_PROVIDER)){
        AlertDialog.Builder alertDialog=new AlertDialog
                .Builder(getContext());
        alertDialog.setTitle("Abilita il GPS");
        alertDialog.setMessage("Il tuo GPS    disabilitato.
                Abilitalo nelle impostazioni.
                Senza GPS non puoi usare l'app.");
        alertDialog.setPositiveButton("Impostazioni GPS",
            new DialogInterface.OnClickListener(){
                public void onClick(DialogInterface dialog,
                    int which){
                    Intent intent=new Intent(Settings
                            .ACTION_LOCATION_SOURCE_SETTINGS);
```

```
                    startActivity(intent);
                    dialog.cancel();
                }
        });
        alertDialog.setNegativeButton("Annulla",
            new DialogInterface.OnClickListener(){
                public void onClick(DialogInterface dialog,
                    int which){
                        dialog.cancel();
                        getActivity().finishAffinity();
                }
        });
        AlertDialog alert=alertDialog.create();
        alert.setCanceledOnTouchOutside(false);
        alert.show();
    }
}
```

4.3 Destinazione

Il punto di destinazione può essere scelto in due modi:

- Cercando la destinazione sulla barra di ricerca e premendo su una delle destinazioni che appaiono sotto come risultati.
- Tendendo premuto per circa due secondi un punto sulla mappa.

In tutti e due i casi apparirà un marker rosso che designerà la destinazione scelta.

Il secondo caso si tratta di un metodo, chiamato clickForMarcker(), che permette all'utente di scegliere una destinazione tenendo premuto sulla mappa il punto da voler raggiungere.
Tale metodo funziona in questo modo: per prima cosa creo un onMapLongClickListener, ovvero un Listener che, al tocco prolungato: richiama il metodo getAddress() per creare una stringa che conterrà lo snippet del marker; in seguito controllerà se esiste già un marker sulla mappa e, in caso affermativo, lo eliminerà; successivamente andrà a creare un markeroption impostandogli la posizione in cui è stato fatto il click prolungato, dandogli come titolo "Destination address", ponendolo draggable, ovvero trascinabile e settare come snippet l'indirizzo prima calcolato dal metodo getAddress().

```
private void clickForMarker(){
    myMap.setOnMapLongClickListener(new GoogleMap
        .OnMapLongClickListener() {
```

```
            @Override
            public void onMapLongClick(LatLng latLng) {
                final LatLng latlngAddress = latLng;
                String snippet;
                Runnable getAddressTask = new Runnable() {
                    @Override
                    public void run() {
                        snippetForClickMarker =
                            getAddress(getContext(),
                            latlngAddress.latitude,
                            latlngAddress.longitude);
                    }
                };
                getAddressTask.run();
                startRoute.setVisibility(View.VISIBLE);
                if(markerOptions!=null) {
                    myMap.clear();
                    markerOptions = null;
                }
                markerOptions = new MarkerOptions()
                        .position(latLng)
                        .draggable(true)
                        .title("Destination address")
                        .snippet(snippetForClickMarker);
                myMap.addMarker(markerOptions);
                setOnClickForPolyline(currentLocation,latLng);
            }
    });
}
```

In più questo metodo esegue altre due funzioni, ovvero startRoute.setVisibility(View.VISIBLE) che farà divenire l'immagine stilizzata dell'uomo che cammina visibile e setOnClickForPolyline(), invece, è il metodo che setterà all'immagine dell'omino stilizzato la funzione di aprire il Fragment MinuteFragment e di far apparire la Polyline sul percorso designato per arrivare a destinazione.
Il primo caso, invece, tratta dell'AutocompleteFragment.

4.3.1 AutocompleteSupportFragment

Come primo step recupero il Fragment statico dal layout xml di MapsFragment e lo assegno ad un oggetto AutocompleteSupportFragment chiamato autocompleteFragment. In seguito setto come impostazione dell'autocompleteFragment che nella lista dei luoghi che appaiono man mano che si scrive sulla barra di ricerca si deve vedere il nome completo della posizione. Infine dichiaro il PlaceSelectedListener, ovvero un Listener che, al click sul luogo scelto come destinazione fra quelli

elencati, farà apparire un marker alla posizione corrispondente e farà un moveCamera della visuale sul luogo scelto, però in questo caso sarà il metodo moveCamera a creare il marker, passandogli la posizione e lo snippet assegnatogli.

```
private void initializeAutocompleteFragment(){
    AutocompleteSupportFragment autocompleteFragment =
        (AutocompleteSupportFragment)getChildFragmentManager()
        .findFragmentById(R.id.autocomplete_fragment);

    autocompleteFragment
        .setPlaceFields(Arrays.asList(
            Place.Field.ID,
            Place.Field.NAME,
            Place.Field.LAT_LNG));

    autocompleteFragment
        .setOnPlaceSelectedListener(
            new PlaceSelectionListener() {
                @Override
                public void onPlaceSelected(Place place) {
                    LatLng positionPlace = place.getLatLng();
                    startRoute.setVisibility(View.VISIBLE);
                    if(markerOptions!=null) {
                        myMap.clear();
                        markerOptions = null;
                    }
                    String snippet = place.getName();
                    moveCamera(positionPlace,DEFAULT_ZOOM,
                                snippet);
                    setOnClickForPolyline(currentLocation,
                                          positionPlace);
                }

        @Override
        public void onError(Status status) {
            Log.i("PROVAPLACE", "An error occurred: " + status);
        }
    });
}
```

```
private void moveCamera(LatLng latLng,
            float zoom, String snippet){
    myMap.moveCamera(CameraUpdateFactory
                        .newLatLngZoom(latLng,zoom));
    if(!snippet.equals("My Location")){
        markerOptions = new MarkerOptions()
                .position(latLng)
                .draggable(true)
```

```
                    .title("Destination address")
                    .snippet(snippet);
            myMap.addMarker(markerOptions);
        }
}
```

Per far si che l'AutocompleteSupportFragment funzioni, ho attivato le Place API (forse da citare) di Google, in modo tale che questa funzionalità riuscisse a trovare i luoghi presenti sulla mappa. Nella onCreate() del MapsFragment controllo che le Place API non siano già inizializzate e, in caso positivo, le vado ad inizializzare.

```
if (!Places.isInitialized()) {
    Places.initialize(getContext(), MY_API_KEY, Locale.ITALY);
}
```

4.4 Polyline

Per quanto riguarda l'implementazione della Polyline mi sono affidato all'implementazione di questa pagina Github [26].
Nel mio applicativo tale funzionalità è stata incaricata al metodo createPolyline(), che prende come parametri: il punto di inizio, ovvero la posizione attuale e la destinazione, ovvero la posizione del marker. Questi due parametri andranno poi passati all'AsyncTask FetchURL che eseguirà le seguenti istruzioni:

- FetchURL.java : è un AsyncTask (forse da citare) al quale, nel costruttore, passo come parametro il contesto dell'applicazione e come parametri per l'AsyncTask prende: l'url generato dal metodo getUrl(), metodo che restituisce una stringa rappresentante l'url necessario alle Directions API (forse da citare) per restituire il percorso corrispondente sulla mappa; e la modalità di trasporto, nel mio caso settata a "walking" visto che la mia applicazione è stata pensata solo per le camminate.
  Nel metodo getUrl() vado a specificare che l'output della risposta delle Directions API deve essere in formato JSON; quindi FetchURL eseguirà la richiesta HTTP e una volta ricevuta la risposta dalle API, nell'onPostExecute() creerà un oggetto PointsParser a cui passa tale risposta.

- PointsParser.java : una volta ricevuta la risposta, questo AsyncTask andrà a creare un oggetto DataParser per fare il parse del JSON contenuto nella stringa passatagli da FetchURL. Quando il parse sarà concluso, nell'onPostExecute() andremo a creare una lista di oggetti LatLng, dove ogni oggetto rappresenta un punto della Polyline, ed infine a richiamare il metodo

onTaskDone(), passando come parametro la lista sopra citata, dell'oggetto taskCallbak di tipo TaskLoadedCallback.

- DataParser.java : è una classe con all'interno due metodi: parse() che riceve come parametro un oggetto JSON, passatogli dall'AsyncTask PointsParser e restituisce una lista di una lista di un'HashMap (forse da citare); e decodePoly, invece, prende un'array JSON di punti e ne restituisce una lista di LatLng. Sarà il metodo parse() che andrà a richiamare il metodo decodePoly() per avere la lista dei punti della Polyline da poter inserire nella HashMap

- TaskLoadedCallbak.java : interfaccia che implementa un solo metodo: onTaskDone() che riceve come parametro un'array di Objects. Tale metodo, come scritto sopra, viene richiamato alla fine dell'onPostExecute() dell'AsyncTask PointsParser, ma viene implementato nella MainActivity.
  Ciò che questo metodo compie è andarsi a prendere la lista di punti passatagli come parametro, assegnarla ad un oggetto PolylineOptions ed infine andare ad aggiungere sulla mappa tale Polyline.

```
private void createPolyline(Location currentLocation, LatLng
    destination){
    LatLng mCurrentLocation =
    new LatLng(currentLocation.getLatitude(),
                currentLocation.getLongitude());
    LatLng mDestination = destination;
    new FetchURL(getContext())
        .execute(getUrl(mCurrentLocation,mDestination,"walking"),
                    "walking");
}
```

```
@Override
public void onTaskDone(Object... values) {
    this.mMap=MapsFragment.mapForActivity;
    if(currentPolyline!=null)
        currentPolyline.remove();
    PolylineOptions polylineOptions = (PolylineOptions) values[0];
    polylineOptions.width(15).color(Color.BLUE);
    currentPolyline = mMap.addPolyline(polylineOptions);
}
```

Capitolo 5

Implementazione del Metronomo

Il metronomo nella mia applicazione serve per dare un input all'utente sulla cadenza dei passi da tenere per arrivare a destinazione rimanendo nel tempo da lui scelto. Il tempo da impiegare per il viaggio viene selezionato nel MinutesFragment, schermata che appare una volta premuto sull'immagine "START". Tale componente riceve in in input il periodo entro il quale l'utente vuole concludere il suo viaggio; una volta inserito il periodo e l'utente pigia sul pulsante "PARTI" viene avviato un Service, chiamato MetronomeService, tramite il metodo startServiceMetronome() che istanzia un Inetent dalla MainActivity al MetronomeService e inserisce come extra dell'intent i minuti, trasformati in secondi, scelti dall'utente.

```
1 private void startServiceMetronome(){
2     Intent startMetronome = new Intent (getActivity(),
3                         MetronomeService.class);
4     startMetronome.putExtra("seconds",seconds);
5     getActivity().startService(startMetronome);
6 }
```

Una volta avviato, nella onCreate() inizializza tutte le variabili necessarie a far funzionare il metronomo e la regolazione della falcata, in seguito nella onStartCommand()prende il valore passatogli tramite il metodo putExtra() precedentemente da MinutesFragmet e lo assegna ad una variabile denominata seconds.
In seguito crea due PendingIntent, il primo che serve alla notifica in foreground per riaprire l'app sulla MainActivity alla posizione corrente, il secondo che sarà assegnato al pulsante "STOP" della notifica per fermare il metronomo. Quest'ultimo è un Intent verso un BroadcastReciver che rimane in ascolto e, nel momento in cui viene richiamto, attiva il metodo stopService() al quale passo un Intent verso MetronomeService per fermarlo. Dopodiche costruisco la notifica inserendogli come titolo "Il Metronomo è partito" e come descrizione la lunghezza della traversata e la durata scelta dall'utente, aggiungendogli come azione tramite il metodo

addAction() l'azione di "STOP" descritta poc'anzi. Faccio partire la notifica con il metodo startForeground(notification).
Essendo la notifica in Foreground, ovvero in primo piano, non può essere eliminata se non con la distruzione del servizio.

```
Intent notificationIntent = new Intent(this, MainActivity.class);
notificationIntent.setAction(Intent.ACTION_MAIN);
notificationIntent.addCategory(Intent.CATEGORY_LAUNCHER);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
    notificationIntent, 0);

Intent stopServiceI = new Intent(getApplicationContext(),
    StopServiceReceiver.class);
PendingIntent pStopService = PendingIntent.getBroadcast(this, 0,
    stopServiceI, PendingIntent.FLAG_CANCEL_CURRENT);

Notification notification = new NotificationCompat
                .Builder(this, CHANNEL_ID)
        .setContentTitle("Il Metronomo    partito.")
        .setContentText("Tempo totale: " +
            String.valueOf(minutesToShow) +
            " minuti - Distanza totale: " + distanceRemain +
            " metri")
        .setSmallIcon(R.drawable.ic_timer_black_24dp)
        .setContentIntent(pendingIntent)
        .addAction(R.drawable.ic_timer_black_24dp, "STOP",
            pStopService)
        .build();

startForeground(1, notification);
```

Però, per far si che la notifica funzioni, dalla versione O di Android deve essere creato un NotificationChannel [27]. Nel mio applicativo lo vado ad istanziare nella MainActivity richiamando il metodo createNotificationChannel() all'interno della onCreate().

Dopo aver creato la notifica in Foreground controllo che Spotify sia installato nel sistema e, in caso affermativo, setta una variabile booleana, chiamata isSpotifyInstalled, a true altrimenti a false. L'implementazione del metronomo nel caso isSpotifyInstalled sia true la vedremo nella sezione successiva.Controllato Spotify richiamo il metodo startMetronome() nel quale vi sono le istruzioni per avviare il metronomo.

Il metronomo classico funziona con due TimerTask, ovvero un Thread con la possibilità di essere rieseguito dopo un tempo prefissato, uno all'interno dell'altro.

Il più esterno ha un period, ovvero il tempo di attesa prima di essere rieseguito di due minuti ed esegue i seguenti calcoli: per prima cosa prende la distanza rimanente aggiornata nelle SharedPreference, dopodiché calcola la distanza rimanente in passi dividendola per la lunghezza del passo dell'utente, calcola i bps, ovvero i passi al secondo che l'utente deve fare ed infine la variabile timeForBip rappresenta quanti millisecondi devo aspettare tra un passo e l'altro, ovvero ogni quanti millisecondi devo fare un passo, e la trovo facendo mille/bps.

```
float distRemain = sp.getFloat("distancePolyline", 0);
distanceRemain = Float.parseFloat(decimalFormat.format(distRemain)
    .replace(",", "."));
int remainStep = (int) distanceRemain * 100 / cmStep;
double bps = (double) remainStep / remainSeconds;
bpm = bps * 60;
timeForBip = (1000 / bps);
if(timeForBip<150){
    makeToast("BPM troppo alti. Fare un altro percorso.");
    onDestroy();
}
if (bps < 1) {
    timeForBip = 1 / bps * 1000;
}
```

Una volta trovati i millisecondi che distanziano un passo dall'altro avvio il secondo TimerTask tramite il metodo metronomeBeep() a cui passo come parametro timeForBeep, questo perché avrà come periodo di attesa il valore assegnato a alla variabile period, in modo tale che ogni volta che venga riattivato produca un suono simile ad un bip.

Per la riproduzione del suono ho creato un oggetto di tipo MediaPlayer [28] a cui ho assegnato un file mp3 che riproduce il suono di un bip, dopodiché eseguo lo start() di tale oggetto ed infine gli assegno un Listener che controlla la sua conclusione. Nel momento in cui il file audio conclude la sua riproduzione eseguo il release() sull'oggetto MediaPlayer in modo da terminarlo. Questo accade ad ogni ripetizione del Task.

Questo TimerTask viene avviato all'interno del primo in modo tale che dopo due minuti si possa aggiornare in base alla distanza rimanente. Per far si che questo sia possibile ho dovuto aggiungere un controllo all'interno del Task più esterno il quale si accerta del fatto che setMetronomeTimer, ovvero il TimerTask interno, è già stato avviato e, in caso positivo, lo distrugge e poi lo ricrea. Questo perché se non facessi tale controllo ci sarebbero più Task in background che riprodurrebbero il suono.

```
private void metronomeBeep(double timeForBip) {
    if(mSpotifyAppRemote!=null)
        mSpotifyAppRemote.getPlayerApi().pause();
        setMetronomeTimer = new Timer();
        setMetronomeTimerTask = new TimerTask() {
        @Override
        public void run() {
            bip = MediaPlayer.create(getApplicationContext(),
                    R.raw.bip);
            bip.start();
            bip.setOnCompletionListener(new MediaPlayer
                    .OnCompletionListener() {
                        public void onCompletion(MediaPlayer mp) {
                            mp.release();
                        };
            });
        }
    };
    setMetronomeTimer.scheduleAtFixedRate(setMetronomeTimerTask,
            0, (long) timeForBip);
}
```

```
if (flagbip && setMetronomeTimer!=null){
    setMetronomeTimer.cancel();
}
```

Qui si conclude l'implementazione della prima variante del metronomo. Nella sezione seguente è spiegato, invece, come funziona il metronomo integrato con Spotify.

5.1 Integrazione con Spotify

Per quello che riguarda l'implementazione di un metronomo basato su Spotify la questione si complica. Per prima cosa ho dovuto eseguire l'accesso a Spotify for Developers [29] e creare un progetto così da vere una chiave univoca, chiamata ClientID, in modo da poter eseguire delle richieste a Spotify. Dopo aver ottenuto la chiave, per prima cosa, all'avvio della MainActivity, nella onCreate() creo un oggetto AuthenticationRequest.Builder specificandogli il ClientID, il tipo di risposta, in questo caso TOKEN, e un RedirectURI, ovvero un uri che ho assegnato al mio progetto nella console di Spotify for Developers. In seguito imposto gli scopes, ovvero dei permessi che l'utente deve dare all'app per interagire con Spotify, si veda la Fig. 2 nel capitolo "Progettazione Architetturale", nel mio caso lo scope è "streaming" [30]. Come riferito da Spotify, l'utente deve avere un account Premium per far si che funzioni. Tale scope è necessario per controllare le riproduzioni

dei brani da parte dell'utente.
Infine eseguo il build() della risposta e apro, tramite il metodo openLoginActivity() l'app di Spotify per eseguire l'accesso e dare i permessi ai vari scopes.

```
AuthenticationRequest.Builder builder =
        new AuthenticationRequest
            .Builder(CLIENT_ID,
                AuthenticationResponse.Type.TOKEN,
                REDIRECT_URI);

builder.setScopes(new String[]{"streaming"});
AuthenticationRequest request = builder.build();

AuthenticationClient.openLoginActivity(this, REQUEST_CODE,
    request);
```

Una volta effettuato l'accesso e aver autorizzato i vari scopes si viene ricondotti alla MainActivity dove verrà richiamato il metodo onActivityResult(). Questo metodo controlla che il REQUEST_CODE, antecedentemente somministrato come parametro al metodo openLoginActivity(), corrisponda al requestcode inserito tra i paramentri del metodo onActivityResult e, se così fosse, viene attivato un switch-case sul codice di risposta dell'autenticazione con due casi:

- TOKEN: ovvero l'autenticazione è andata a buon fine e viene restituita, all'interno della risposta, una stringa,come chiave identificativa del soggetto richiedente, chiamata token, che servirà in seguito per eseguire le richieste http a Spotify. Dopodiché mi salvo questa stringa nelle SharedPreferences e vado a impostare una playlist predefinita, nel mio caso la Top 50 Global, ovvero le 50 canzoni più ascoltate al mondo sulla piattaforma musicale. Infine richiamo un AsyncTask, denominato JsonTask, che mi salverà, in un oggetto di tipo JSONArray le playlist che l'utente ha nella sua libreria personale.
- ERROR: nel caso in cui l'autenticazione non va a buon fine.

```
protected void onActivityResult(int requestCode, int resultCode,
    Intent intent) {
    super.onActivityResult(requestCode, resultCode, intent);

    if (requestCode == REQUEST_CODE) {
        AuthenticationResponse response = AuthenticationClient
                .getResponse(resultCode, intent);

        switch (response.getType()) {
            case TOKEN:
                token = response.getAccessToken();
```

```
                String plid = prefs.getString("plIDSelected",
                                                "none");
                SharedPreferences.Editor editprefs = prefs.edit();
                editprefs.putString("tokenSpotify",token);
                if(plid.equals("none"))
                    editprefs.putString("plIDSelected",
                                        "37i9dQZEVXbMDoHDwVN2tF");
                editprefs.apply();
                new JsonTask().execute(ENDPOINT);
                break;

            case ERROR:
                break;

        }
    }
}
```

Il JsonTask è un AsyncTask che: nel doInBackground esegue una richiesta http di tipo GET all'url Spotify per richiedere la lista delle playlists dell'utente attualmente connesso. Tale lista viene restituita in formato JSON e, una volta ricevuta la risposta, viene attivato il metodo onPostExecute() che andrà a creare un JSONArray delle playlists presenti nella stringa JSON di risposta e vi salverà nome ed id della playlist.

```
private class JsonTask extends AsyncTask<String, String, String> {

    protected String doInBackground(String... params) {
        return actionJSONFromURLConnection(params[0],"GET");
    }

    @Override
    protected void onPostExecute(String result) {
        spotifyPlaylists = result;
        try {
            JSONObject obj = new JSONObject(spotifyPlaylists);
            JSONArray jArr = obj.getJSONArray("items");
            playlists = new JSONArray();
            for (int i = 0; i < jArr.length(); i++) {
                JSONObject item = jArr.getJSONObject(i);
                String name = item.getString("name");
                String id = item.getString("id");
                playlists.put(new JSONObject("{\"name\":\"" + name
                        + "\",\"id\":\"" + id + "\"}"));
            }

        } catch (JSONException e) {
```

```
            e.printStackTrace();
        }
    }
}
```

Tale JSONArray verrà in seguito posto come parametro nel costruttore del PlaylistsFragment, in modo tale che tale componente possa creare la lista delle playlists appartenenti all'utente così da potergliele far vedere e selezionare la più adatta al contesto.

Il PlaylistFragment è un Fragment presente nel menu dell'applicazione dove, se premuto, aprirà una schermata, come visibile nella sezione "Scegli la tua Playlist" della sezione "Navigation Drawer",dal quale l'utente potrà scegliere la playlist che più lo aggrada per riprodurre la musica durante il viaggio.
Questo componente lavora in questo modo:

- Per prima cosa controlla se la playlist sia diversa da null e la sua lunghezza sia maggiore di zero. Dopodiché esegue due cicli:
- Il primo ciclo serve per andare ad inserire all'interno del RadioGroup dichiarato nel layout xml del Fragment tutti i RadioButton corrispondenti alle playlists. Assegna ad ogni RadioButton la posizione i-esima nell' array JSON come id e come testo il nome corrispondente a quell'id.
- In seguito imposto il primo RadioButton come predefinito e la vista dei RadioButton in verticale.
- Infine creo un Listener sul RadioGroup che controlla quale RadioButton è stato scelto. Una volta scelto il RadioButton mi salvo l'id della playlist corrispondente all'interno delle SharedPreference perché in seguito mi servirà per ricavare i brani e i loro metadati.

Se non si è eseguito l'accesso a Spotify o non si hanno playlists nella propria libreria, al posto della lista di RadioButtons, verrà visualizzata una frase con su scritto "No results found". Per far si che una playlist sia resa visibile all'interno dell'applicazione, tale playlist deve essere resa pubblica.

```
if(playlists!=null && playlists.length()>0) {
    for (int i = 0; i < playlists.length(); i++) {
        RadioButton rb = new RadioButton(getContext());
        try {
            JSONObject pl = playlists.getJSONObject(i);
            rb.setText(pl.getString("name"));
            rb.setId(i);
        } catch (JSONException e) {
```

```
            e.printStackTrace();
        }
        rg.addView(rb);
    }
    rg.check(plSelected);
    rg.setOrientation(RadioGroup.VERTICAL);

    rg.setOnCheckedChangeListener(
        new RadioGroup.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(RadioGroup group,
                                         int checkedId) {
                RadioButton radioButton =
                    (RadioButton) view.findViewById(checkedId);
                String plid = "";
                for (int i = 0; i < playlists.length(); i++) {

                    try {
                        JSONObject pl = playlists.getJSONObject(i)
    ;
                        if (radioButton.getText().toString()
                                .equals(pl.getString("name"))) {
                            plid = pl.getString("id");
                        }
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            editsp.putString("plIDSelected", plid);
            editsp.putInt("plSelected", checkedId);
            editsp.apply();
        }
    });
}
```

Dopo aver impostato tutti i componenti base per l'utilizzo di Spotify nel metronomo, possiamo procedere alla spiegazione riguardante il funzionamento della riproduzione di brani in base al bpm.
Come detto prima, nel MetronomeService, controllo che Spotify sia installato, e, in caso affermativo, imposto la variabile booleana isSpotifyInstalled uguale a true. Ma non basta avere Spotify installato, è necessario anche aver eseguito l'accesso. Questo controllo viene eseguito nel metodo startMetronome() nel quel vedo se è stato eseguito l'accesso. La variabile nella SharedPreference viene modificata nell'onResume() della MainActivity dove ho un if la quale condizione controlla se la stringa token sia uguale a null. In caso affermativo imposto la variabile nella SharedPreferences a false e in questo modo non verranno riprodotti i brani da

Spotify, bensi verrà azionato il metronomo classico descritto antecedentemente.

Una volta superati tutti i controlli entreremo nell'if isSpotifyInstalled uguale a true nel quale troveremo un altro if. Quest'ultimo servirà nel momento in cui il metronomo si aggiornerà dopo due minuti per vedere quanti brani sono state messe in coda e fare lo skipNext(), ovvero passare alla prossima canzone, tante volte quanto il numero di brani in coda. Ho dovuto adottare questa soluzione perché, purtroppo, le Spotify API non hanno un metodo che da la possibilità di pulire la coda del playback.
In seguito verrà richiamato il JsonTask passandogli come parametro un uri, chiamato ENDPOINTPLAYLISTTRACKS, che servirà per avere come
risposta tutte le track presenti nella playlist selezionata precedentemente dall'utente nel PlaylistFragment. Vado a ricavarmi l'id della playlist da mettere nell'uri nelle SharedPreferences alla chiave corrispondete nel quale lo avevo salvato.

```
playlistID = sp.getString("plIDSelected", "");
ENDPOINTPLAYLISTTRACKS = "https://api.spotify.com/v1/playlists/" +
    playlistID + "/tracks";
```

```
if(countTrackInQueue>0) {
    int i=0;
    while(i<countTrackInQueue){
        mSpotifyAppRemote.getPlayerApi().skipNext();
        i++;
    }
    countTrackInQueue=0;
}
new JsonTask().execute(ENDPOINTPLAYLISTTRACKS);
```

JsonTask è un AsyncTask che:

- Nel doInBackground() fa una richiesta http di tipo GET tramite il metodo prima citato posizionato nella MainActivity.
- Nell'onPostExecute(), una volta ricevuta la stringa di risposta in formato JSON dal metodo actionJSONFromURLConnection(), eseguo un for sul JSONArray nella stringa e vado a creare l'uri contenente gli id di tutte le tracks presenti nella playlist con il quale andrò a chiedere i metadati relativi di ogni brano così da scoprire i bpm di ogni canzone.

Una volta creata la stringa che rappresenta il mio uri per richiedere le audio features delle track presenti nella playlist richiamo il secondo AsyncTask, denominato

JsonTaskBPM passando come parametro la stringa creata poc'anzi.

JsonTaskBPM è l'AsyncTask più lungo e complesso che ho nel mio applicativo. In ordine, esegue le seguenti mansioni:

- Come per JsonTask, esegue il metodo actionJSONFromURLConnection() per fare una richiesta GET dei metadati relativi alle track presenti nella playlist.

- Dopodiché, una volta ricevuta la risposta, parte il metodo onPostExecute. In primis trasforma la risposta da stringa in un oggetto JSON e, in seguito, ne ricava l'array delle tracks.
  Successivamente legge il tempo relativo ad ogni track, tramite un ciclo che scorre l'array, e lo salva in un ArrayList di double chiamato songsTempo. In seguito controlla il bpm calcolato precedentemente dal TimerTask semplicemente facendo bps*60, e controllando che sia maggiore del bpm minimo accettato, da me impostato ad 80. In caso contrario viene trovata la canzone con il bpm minore e viene impostato per il viaggio. Invece, se maggiore di MINBPM ricadiamo nell'else dove controlleremo se il bpm è maggiore della canzone con il bpm maggiore fra tutte; in questo caso avvieremo il metronomo normale.
  Se nessuno dei due casi si presentasse, procederemo con la creazione di un ArrayList di stringe, chiamato trackToPlay, dove ogni cella contiene l'id di una canzone che possiede un tempo compreso fra bpm meno 5 e bpm più 5. Se tale ArrayList dovesse essere vuoto perché non ci sono canzoni che rientrano in quell'intervallo allora troverò la canzone con il tempo più alto e più vicino al bpm calcolato, e lo riassegno alla variabile bpm, in modo tale da avere almeno una canzone da riprodurre. In questo caso farò apparire un Toast dove scriverò che, essendo stato aumentato il tempo, si arriverà probabilmente in anticipo.
  Come ultimo step, non avendo la lista di tracks da riprodurre vuota, metterò in riproduzione la prima track, tramite il metodo connectSpotify(), che spiegherò successivamente, della lista e chiamerò il JsonTaskQueue.

Di seguito sono riportate le foto del codice di questo AsyncTask per capirne meglio i meccanismi.

```
1 private class JsonTaskBPM extends AsyncTask<String, String, String
    > {
2
3     boolean canSpotifyPlay = true;
4
5     protected String doInBackground(String... params) {
```

```
6           return MainActivity
7                   .actionJSONFromURLConnection(params[0],"GET");
8       }
9
10      @Override
11      protected void onPostExecute(String s) {
12          super.onPostExecute(s);
13          try {
14              JSONObject result = new JSONObject(s);
15              JSONArray audioFeatures = result
16                              .getJSONArray("audio_features");
17              for (int i = 0; i < audioFeatures.length(); i++) {
18                  if (i == audioFeatures.length() - 1)
19                      break;
20                  JSONObject track = null;
21                  if (audioFeatures.getJSONObject(i) != null)
22                      track = audioFeatures.getJSONObject(i);
23                  double tempo = track.getDouble("tempo");
24                  songTempo.add(tempo);
25              }
26              if(bpm<MINBPM){
27                  bpm = findMinBPM(songTempo);
28                  makeToast("BPM troppo basso, ora    stato
29                      impostato a "+bpm+". Arriverai in anticipo.");
30              } else {
31                  double maxbpm = findMaxBPM(songTempo);
32                  if(bpm>maxbpm) {
33                      canSpotifyPlay = false;
34                      if(mSpotifyAppRemote!=null)
35                          mSpotifyAppRemote.getPlayerApi().pause();
36                      metronomeBeep(timeForBip);
37                      makeToast("BPM troppo alto,
38                          nessuna canzone in grado di raggiungerlo.
39                          Partir  il metronomo classico");
40                  }
41              }
42              if (canSpotifyPlay) {
43                  for (int i = 0; i < audioFeatures.length(); i++) {
44                      if (i == audioFeatures.length() - 1)
45                          break;
46                      JSONObject track = null;
47                      if (audioFeatures.getJSONObject(i) != null)
48                          track = audioFeatures.getJSONObject(i);
49                      double tempo = track.getDouble("tempo");
50                      if (bpm - 5 <= tempo && bpm + 5 >= tempo) {
51                          int duration = track
52                                  .getInt("duration_ms");
53                          tracksToPlay
54                              .add(track.getString("id"));
```

```
                        if(tracksToPlay.size()>40)
                            break;
                        trackDuration.add(duration);
                    }
                }
                if(tracksToPlay.size()>0) {
                    connectSpotify(tracksToPlay.get(0));
                    (new Handler()).postDelayed(
                        new Runnable() {
                            @Override
                            public void run() {
                                new JsonTaskQueue().execute();
                            }
                        },
                        2000
                    );
                }
                else {
                    makeToast("Nessuna canzone con questo bpm,
                        arrotondato con la canzone con il bpm pi
                        alto subito dopo. Arriverai in anticipo");
                    bpm=findNearHighBPM(audioFeatures,bpm);
                    for (int i = 0;
                            i < audioFeatures.length(); i++) {
                        if (i == audioFeatures.length() - 1)
                            break;
                        JSONObject track = null;
                        if (audioFeatures.getJSONObject(i)
                            != null)
                            track = audioFeatures
                                        .getJSONObject(i);
                        double tempo = track.getDouble("tempo");
                        if (bpm - 5 <= tempo && bpm + 5 >= tempo)
                        {
                            int duration = track
                                .getInt("duration_ms");
                            tracksToPlay
                                .add(track.getString("id"));
                            if(tracksToPlay.size()>40)
                                break;
                            trackDuration.add(duration);
                        }
                    }
                    connectSpotify(tracksToPlay.get(0));
                    (new Handler()).postDelayed(
                            new Runnable() {
                                @Override
                                public void run() {
                                    new JsonTaskQueue().execute();
```

```
                                    }
                                },
                                2000
                        );
                    }
                }
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }
```

Il metodo connectSpotify() è il cuore per il funzionamento del collegamento con l'app. Tramite questo metodo riesco a congiungere la mia app a Spotify utilizzando delle classi importate dalle api di Spotify. Per poter usare tali classi ho dovuto integrare un modulo, seguendo questa guida [31], chiamato spotify-app-remote, all'interno del quale vi sono presenti i file per poter scrivere il codice con cui connettere la propria app ad essa.
Il metodo funziona così: prima di tutto vado ad instaurare un collegamento con l'app attraverso un oggetto ConnecionParams, a cui passo come parametri i valori necessari per far funzionare la connessione, come il CLIENT_ID ed il REDIRECT_URI, dopodiché instauro la connessione tramite un ConnectionListener dell'oggetto Connector all'interno del metodo connect() della classe SpotifyAppRemote. Tale Listener ha due possibili risultati:

- onConnected() che ha come parametro un oggetto spotifyAppRemote il quale lo assegno alla mia variabile globale mSpotifyAppRemote per istaziarla e poterne poi andare a richiamare i metodi, come skipNext() per passare alla prossima canzone, o, come in questo caso, usare il metodo play() di quest'oggetto per andare a riprodurre la track che passo come parametro a connectSpotify()
- onFailure nel caso in cui l'app non dovesse riuscire a raggiungere Spotify.

```
private void connectSpotify(String firstTrack) {
    ConnectionParams connectionParams =
            new ConnectionParams.Builder(CLIENT_ID)
                    .setRedirectUri(REDIRECT_URI)
                    .showAuthView(true)
                    .build();
    SpotifyAppRemote.connect(this, connectionParams,
        new Connector.ConnectionListener() {

            public void onConnected(
                    SpotifyAppRemote spotifyAppRemote) {
```

```
                mSpotifyAppRemote = spotifyAppRemote;
                mSpotifyAppRemote.getPlayerApi()
                    .play("spotify:track:" + firstTrack);
            }

            public void onFailure(Throwable throwable) {
                Log.e("MyActivity", throwable.getMessage(),
                    throwable);
            }
        });
}
```

JsonTrackQueue è l'AsyncTask che si occupa di mettere in coda tutte le canzoni trovate da JsonTaskBPM e salvate nell'ArrayList trackToPlay.
Qui utilizzo solo il metodo doInBackground() dove: inizializzo una variabile che corrisponde alla lunghezza dell'ArrayList trackToPlay, dopodiché, in base a tale lunghezza, decido una cifra, assegnata alla variabile n, che corrisponderà al numero di volte in cui vado a ripetere l'operazione di messa in coda dei brani presenti in tracksToPlay.
Infine creo un doppio ciclo for dove il più esterno verrà ripetuto n volte mentre il più interno verrà ripetuto tante volte quante sono le canzoni all'interno di trackToPlay. All'interno di quest'ultimo secondo ciclo richiamo il metodo actionJSONFromURLConnection() dove nell'uri specifico il comando per inserire la track corrispondente alla cella i-esima in coda. Questa volta il tipo della richiesta http sarà POST.

```
private class JsonTaskQueue extends AsyncTask<Void, Void, Void> {
    @Override
    protected Void doInBackground(Void... voids) {
        int length = tracksToPlay.size();
        int n = 0;
        if(length<5)
            n = 8;
        else if(length<10)
            n = 4;
        else if(length<20)
            n = 2;
        else
            n = 1;
        for (int k = 0; k < n; k++) {
            for (int i = 0; i < tracksToPlay.size(); i++) {
                countTrackInQueue++;
                MainActivity
                    .actionJSONFromURLConnection(
                        "https://api.spotify.com/v1/me/player/
```

```
                            queue?uri=spotify%3Atrack%3A" +
                            tracksToPlay.get(i),
                            "POST");
                }
            }
            return null;
        }
}
```

Nel momento in cui il Service viene distrutto, viene pulita la coda nello stesso modo in cui è descritto nel metodo startMetronome(), ovvero eseguendo il metodo skipNext() tante volte quante sono i brani presenti in coda e, in seguito, viene disconnesso il collegamento tra l'app e Spotify.

Per approfondire, queste sono le API di Spotify da me utilizzate per eseguire tutte le funzionalità sopra descritte:

- Per sapere le playlists nella libreria dell'utente.
- Per avere la lista delle tracks di una determinata playlist.
- Per avere i metadati dei brani musicali.
- Per mettere le canzoni in coda.

Possono essere trovati tutti alla pagina [32] sotto le voci Player, Playlists e Tracks.

Capitolo 6

Regolazione Adattiva della Falcata

In questo capitolo tratterò gli argomenti sul quale ho compiuto i miei studi per questa tesi.
Il io studio è stato quello di usare i sensori di uno smartphone per cercare di approssimare la lunghezza del passo dell'utente, in modo tale che, più la lunghezza fosse precisa, più il calcolo dei bpm sopra riportato sarebbe stato ottimale.
Per far si che questa funzionalità riuscisse mi sono servito di due sensori, il sensore GPS ed il sensore contapassi. Per quanto riguarda il GPS ho già spiegato come sono andato ad utilizzarlo, per quel che riguarda il contapassi, invece, ne espongo l'impiego qui di seguito:

- In primis, al MetronomeService, ho fatto implementare l'interfaccia SensorEventListener, la quale richiede l'implementazione di due metodi: onSensorChanged() e onAccuracyChanged(). Il primo viene attivato nel momento in cui il sensore rivela dei cambiamenti, il secondo viene attivato nel momento in cui l'accuratezza del sensore cambia, però, nel mio caso, non ho usufruito di quest'ultimo metodo.

- Dopodiché, nella onCreate(), ho inizializzato un oggetto SensorManager, oggetto con il quale si possono accedere ai servizi di sistema relativi ad i sensori, dal quale ho inizializzato il sensore corrispondente alle mie necessità, ovvero lo STEP_COUNTER.
  In seguito controllo che il dispositivo ne sia dotato, ed in caso positivo registro il sensore impostandogli come ritardo di risposta il minimo possibile tramite SENSOR_DELAY_FASTEST.
  Nel caso in cui, invece, lo smartphone non fosse dotato di tale sensore stampo un Toast in cui scrivo "Sensore non trovato" e non procedo alla computazione della lunghezza del passo. La registrazione del sensore avviene nel metodo startMetronome().

```
sensorManager = (SensorManager) getSystemService(Context.
    SENSOR_SERVICE);

```

```
Sensor counterSensor = sensorManager.getDefaultSensor(Sensor.
    TYPE_STEP_COUNTER);
if(counterSensor!=null){
    makeToast("registro il sensore.");
    sensorManager.registerListener(this,counterSensor,
                SensorManager.SENSOR_DELAY_FASTEST);
    running = true;
} else {
    makeToast("Sensore non trovato!");
}
```

- Come ultimo step nell'onDestroy() del service vado a scollegare la registrazione del sensore tramite il metodo unregister().

```
sensorManager.unregisterListener(this);
```

Il metodo da me adottato per approssimare la lunghezza è lo sliding windows, ovvero crearmi delle finestre di una dimensione prestabilita, nel mio caso di 100 metri, che variano fra di loro per una lunghezza δ, ovvero 20 metri. Quindi avrò un comportamento da 0 a 100 metri, un altro da 20 a 120, un altro ancora da 40 a 140 e così via, fino al momento in cui l'utente o l'applicativo non termineranno il Service.
Per far operare tale funzionalità mi sono servito solo del metodo onSensorChanced che procede svolgendo le seguenti funzioni:

- Come primo passaggio controllo che la variabile running sia true, viene settata a true solo nel momento in cui la registrazione del sensore avviene con successo.

- In seguito salvo in una variabile la distanza rimanente, calcolatami dalla Polyline e salvata nelle SharedPreferences, e controllo che tale distanza non sia maggiore della distanza iniziale, perché se così fosse vorrebbe dire che l'utente non sta seguendo la Polyline e l'applicativo poi non riesce a computare in maniera ottimale la lunghezza del passo.

- Una volta che l'utente inizia a seguire la Polyline si entra nell'else del controllo precedentemente eseguito ed istanzio due variabili, la prima è la mia window la seconda è il mio δ.

- in seguito mi salvo in una variabile chiamata initialStep, il valore dei passi contati fino a quel momento, perché il sensore salva il numero totale dei passi fatti e li somma con i nuovi passi compiuti ogni volta che viene azionato. Per far si che ad ogni viaggio inizi da 0 devo salvarmi il valore dei passi totali situati nella memoria del sensore e sottrargli il valore iniziale posizionato in initialStep.

- totalStep sarà la mia variabile in cui memorizzerò i passi totali di quel viaggio, prendendo il valore totale dei passi contati nella memoria del sensore e sottraendogli i passi inizialmente calcolati. la variabile initialStep viene assegnata solo la prima volta in cui si entra nel sensore grazie a un controllo su un booleano il quale, una volta compiuto, non vi entro più perché imposto il booleano a false.

- Successivamente trovo la distanza che è stata percorsa semplicemente sottraendo alla distanza iniziale la distanza restante.

```
@Override
public void onSensorChanged(SensorEvent event) {
    if(running) {
        float distRemain = sp.getFloat("distancePolyline", 0);
        if(distRemain>initialDistance+20) {
            makeToast("Segui la polyline se vuoi
                che l'app misuri i tuoi passi.");
            eventFirst = true;
        }
        else {
            double minMeters = 100;
            int deltaM = 20;
            if (eventFirst) {
                initialStep = (int) event.values[0];
                eventFirst = false;
            }

            totalStep = (int) (event.values[0] - initialStep);

            float differenceDistance = initialDistance-distRemain;
```

Da questo momento inizia l'implementazione delle sliding windows. I controlli inizio a compierli solo nel momento in cui l'utente percorre almeno cinque metri. Da qui entro nel primo if e controllo se la distanza percorsa è compresa in un intervallo

$$distanzaPercorsa\%\delta - 3, distanzaPercorsa\%\delta + 3$$

. Ho deciso di inserire un intervallo invece che una distanza ben precisa perché la Polyline non si aggiorna perfettamente. Entrato in questo controllo arrotondo

la distanza percorsa, chiamata differeceDistance, al multiplo di δ più vicino per poi dividere tale modulo per δ, in modo tale da avere l'indice del δ in cui sono passato e, se la Polyline si dovesse aggiornare di nuovo all'interno di quell'intervallo, l'applicazione non copie di nuovo i calcoli. Successivamente ho un ispezione, sempre tramite if, che verifica se l'ArrayList in cui mi salvo tali indici, contiene l'indice precedentemente calcolato. Nel caso in cui lo contenesse non eseguo niente, altrimenti aggiungo l'indice all'ArrayList e proseguo implementando le seguenti funzionalità: stepsInDelta è l'ArrayList contenente in ogni cella i passi corrispondenti fatti dalla distanza δ*(i-1) alla distanza δ*i, dove i corrisponde alla cella i-esima dell'Arraylist. All'interno dell'if popolo questa struttura dati in base ai passi rilevati dal sensore.

```
boolean enterInDelta = differenceDistance % deltaM < 3 ||
   differenceDistance % deltaM > deltaM - 3;
if(differenceDistance>5) {
    if (enterInDelta) {
        int roundOfDiffDist =
            (int) round(differenceDistance,deltaM);
        int moduleOfRoundAndDelta = roundOfDiffDist/deltaM;
        makeToast("in delta: moduleOfRoundAndDelta:
            "+moduleOfRoundAndDelta+" roundOfDiffDist:"+
            roundOfDiffDist);
        if(deltaVisited.contains(moduleOfRoundAndDelta)){
            checkDeltaOnlyOnce = false;
        } else {
            checkDeltaOnlyOnce = true;
            deltaVisited.add(moduleOfRoundAndDelta);
        }
        if(checkDeltaOnlyOnce) {
            if (stepsInDelta.size() > 0) {
                int totStepsDelta = 0;
                for (int i = 0; i < stepsInDelta.size(); i++) {
                    totStepsDelta += stepsInDelta.get(i);
                }
                if (totalStep - totStepsDelta < 10) {
                    int stepToAdd = totalStep - totStepsDelta;
                    int stepsInDeltaLastPosition = stepsInDelta
                            .get(stepsInDelta.size() - 1);
                    stepsInDelta.set(stepsInDelta.size() - 1,
                            stepToAdd + stepsInDeltaLastPosition);
                } else
                    stepsInDelta.add(totalStep - totStepsDelta);
                makeToast("stepsInDelta-1 con for: " +
                    stepsInDelta.get(stepsInDelta.size() - 1));
            } else {
                stepsInDelta.add(totalStep);
                makeToast("stepsInDelta-1: " +
```

```
                        stepsInDelta.get(stepsInDelta.size() - 1));
            }
        }
    }
}
```

Nel secondo controllo invece verifico che la distanza percorsa sia maggiore di minMeters, ovvero 100 metri, e, se così fosse, di nuovo controllo che la distanza sia in un intervallo

$$distanzaPercorsa\%\delta - 3, distanzaPercorsa\%\delta + 3$$

. Rieseguo i controlli eseguiti sopra, ovvero arrotondo la distanza ad un multiplo di δ e divido il risultato per ottenere l'indice della posizione del δ in cui mi trovo. Verifico che tale posizione non sia stata inserita nell'ArrayList deltaVisitedInWindow e, nel caso in cui fosse già presente, non eseguo niente, altrimenti, aggiungo tale posizione all'ArrayList e proseguo con i calcoli. Ho creato questa struttura dati per gli stessi motivi spiegati sopra.
Nel momento in cui non trovo la posizione all'interno dell'ArrayList posso proseguire implementando le seguenti istruzioni: trovo tutti i passi compiuti nella window andando a sommare le ultime 5 posizioni dell'ArrayList stepsInDelta, le quali corrispondono ad una distanza totale di 100 metri. Successivamente calcolo la lunghezza del passo di quella window facendo minMeters, ovvero 100, diviso i passi totali compiuti in quella window e vado a salvare il risultato in un altro ArrayList chiamato stespArray e crea una stringa, chiamata valuesInStepsArray, che rappresenta la stringa visualizzata nella schermata SLH riportante:lunghezza del passo per quei 100 metri_passi totali in quei 100 metri.

```
if (differenceDistance >= minMeters) {
    if (enterInDelta) {
        int roundOfDiffDist =(int) round(differenceDistance,deltaM
    );
        int moduleOfRoundAndDelta = roundOfDiffDist/deltaM;
        makeToast("in window: moduleOfRoundAndDelta: "+
            moduleOfRoundAndDelta+" roundOfDiffDist:"+
            roundOfDiffDist);
        if(deltaVisitedInWindow.contains(moduleOfRoundAndDelta)){
            checkDeltaOnlyOnce = false;
        } else {
            checkDeltaOnlyOnce = true;
            deltaVisitedInWindow.add(moduleOfRoundAndDelta);
        }
        if(checkDeltaOnlyOnce) {
            double totStepSlidingWindow = 0;
            for (int i = k; i < stepsInDelta.size(); i++) {
```

```
                totStepSlidingWindow += stepsInDelta.get(i);
            }
            double sldouble = (minMeters / totStepSlidingWindow);
            makeToast("sldouble: " + sldouble);
            int stepLength = (int) (sldouble * 100);
            makeToast("stepLength: " + stepLength);
            stepsArray.add(stepLength);
            makeToast("stepsArray: " +
                stepsArray.get(lengthStepsArray) +
                " totStepSlidingWindow: " + totStepSlidingWindow);
            valuesInStepsArray +=
                stepsArray.get(lengthStepsArray) + "_" +
                totStepSlidingWindow + " - ";
            makeToast("stepsArray size:" + lengthStepsArray);
            k++;
            lengthStepsArray++;
        }
    }
}
```

Non appena il Service viene terminato, o dall'utente o dall'applicativo, controllo, nel metodo onDestroy() di MetronomeService, che la stringa sopra citata non sia vuota, e nel caso non lo fosse, vado a calcolare la deviazione standard sui valori ottenuti dalla sliding windows in modo tale da poter vedere se, durante il viaggio, i valori ottenuti sono stati più o meno simili. Dopodiché salvo sul file, visibile alla schermata SLH del menu, i vari dati ottenuti dalla corsa e richiamo l'AsyncTask InsertStepLengthTask per andare ad inserire la media della lunghezza del passo tenuta durante la corsa e la deviazione standard relativa ad essa nel database.

```
if(valuesInStepsArray != ""){
    int sum = 0;
    for(int i = 0; i<stepsArray.size();i++){
        sum += stepsArray.get(i);
    }
    int stepAVG = sum / stepsArray.size();
    for (int i = 0; i<stepsArray.size();i++){
        if(!(stepsArray.get(i)>stepAVG+50)
            || !(stepsArray.get(i)<stepAVG-50)){
            sANoOutliers.add(stepsArray.get(i));
        }
    }

    double sd = standardDeviation(sANoOutliers,
                    sANoOutliers.size());
    String writeOnFile = "";
    if(sANoOutliers.size()>0)
        writeOnFile = "Corsa del: "+dataDellaCorsa+
```

```
            "\nMedia di stepsArray: "+stepAVG+
            "\nArray dei valori SW con outliesrs: "
            +valuesInStepsArray+"\nSD no outliers: "+sd+"\n\n";
    else
        writeOnFile = "Corsa del: "+dataDellaCorsa+
            "\nI calcoli hanno fallito. Nessun valore riscontrato"
    ;
    writeToFile(writeOnFile,getApplicationContext());
    String stepsInDeltaString = "";
    for(int i=0;i<stepsInDelta.size();i++){
        stepsInDeltaString +=
            stepsInDelta.get(i).toString() + " - ";
    }
    writeToFile(stepsInDeltaString,getApplicationContext());
    if(sANoOutliers.size()>0)
        new InsertStepLengthTask(stepAVG,(int)sd).execute();
}
```

Ad InsertStepLengthTask vanno iniettati due valori interi nel costruttore che corrispondono alla deviazione standard e alla lunghezza media del passo corrispondente a quella corsa. Dopodiché crea due oggetti, uno di tipo StepLengthHistory e l'altro di tipo stringa, il quale corrisponde alla data corrente in formato italiano. Come penultimo passaggio entra nel doInBackground(), va ad assegnare la lunghezza del passo, la deviazione standard, le quali sono state passate tramite il costruttore, e la data corrente all'oggetto slh tramite i suoi metodi setStepLength(), setRegistrationDate() e setStandardDeviation() e lo va a salvare nel database come nuovo record della tabella StepLengthHistory. Infine, dopo aver inserito il nuovo SLH record nel database, nell' onPostExecute richiamo un altro AsyncTask, chiamato SLHTask, nel quale, all'interno del metodo doInBackground, vado a prendere dalla base di dati tutti i valori inseriti nella tabella StepLenthHistory e li salvo all'interno di una lista chiamata slhList. Dopodiché nell'onPostExecute() vedo quali record, tramite un ciclo for, hanno una deviazione standard minore di 15, e salvo la lunghezza del passo corrispondente al record all'interno di un ArrayList stepLengthUnderSD. Infine controllo che ci sia almeno un dato all'interno di quest'ultima struttura dati e calcolo la media per avere un risultato finale da andare poi a salvare come lunghezza ufficiale del passo dell'utente richiamando l'AsyncTask ChangeStepLengthTask.

```
private class InsertStepLengthTask extends AsyncTask<Void, Void,
    Void> {

    private int stepLength;
    private int sd;
```

```
5
6       public InsertStepLengthTask(int stepLength, int sd){
7           this.stepLength = stepLength;
8           this.sd = sd;
9       }
10
11      StepsLengthHistory slh = new StepsLengthHistory();
12      String currentDate =
13          new SimpleDateFormat("dd-MM-yyyy HH:mm",
14              Locale.getDefault()).format(new Date());
15
16      @Override
17      protected Void doInBackground(Void... voids) {
18          slh.setStepLength(stepLength);
19          slh.setRegistrationDate(currentDate);
20          slh.setStandardDeviation(sd);
21          MainActivity.mydb.mydaoSH().addStep(slh);
22          return null;
23      }
24
25      @Override
26      protected void onPostExecute(Void aVoid) {
27          new SLHTask().execute();
28          super.onPostExecute(aVoid);
29      }
30  }
```

```
1   private class SLHTask extends AsyncTask<Void, Void, Void>{
2
3       List<StepsLengthHistory> slhList;
4       List<Integer> stepLengthUnderSD = new ArrayList<>();
5       private static final int SD_LIMIT = 15;
6
7       @Override
8       protected Void doInBackground(Void... voids) {
9           slhList = MainActivity.mydb.mydaoSH().getSLH();
10          return null;
11      }
12
13      @SuppressLint("SetTextI18n")
14      @Override
15      protected void onPostExecute(Void aVoid) {
16          for(int i=0;i<slhList.size();i++){
17              if(slhList.get(i).getStandardDeviation()<SD_LIMIT){
18                  stepLengthUnderSD
19                      .add(slhList.get(i).getStepLength());
20              }
21          }
```

```
        if(stepLengthUnderSD.size()>0) {
            int meanStepLength = 0;
            int sum = 0;
            for (int i = 0; i < stepLengthUnderSD.size(); i++) {
                sum += stepLengthUnderSD.get(i);
            }
            meanStepLength =
                (int) (sum / stepLengthUnderSD.size());
            new ChangeStepLength().execute(meanStepLength);
        }
        super.onPostExecute(aVoid);
    }
}
```

```
private class ChangeStepLength
    extends AsyncTask<Integer,Void,Void> {

    List<User> users;
    User oldUser;

    @Override
    protected Void doInBackground(Integer... integers) {
        users = MainActivity.mydb.mydao().getUser();
        oldUser = users.get(0);
        MainActivity.mydb.mydao().removeOldUser();
        String leaveName = oldUser.getName();
        int newCmStep = integers[0];
        User newUser = new User();
        newUser.setCmStep(newCmStep);
        newUser.setName(leaveName);
        MainActivity.mydb.mydao().addUser(newUser);
        return null;
    }

    @Override
    protected void onPostExecute(Void aVoid) {
        super.onPostExecute(aVoid);
    }
}
```

Capitolo 7

Risultati e Conclusioni

In questa tesi abbiamo parlato dell'implementazione di un'applicazione in grado di, dato un luogo di destinazione e un tempo per percorrere la distanza tra il dispositivo e tale destinazione, fornirci un input sonoro, sotto forma di metronomo o di canzone, che indica la cadenza dei passi da tenere calcolata in base alla distanza e al tempo per terminarla.
Il mio studio si è basato sul cercare di approssimare quanto più vicina alla realtà, la lunghezza del passo dell'utente sfruttando i sensori GPS e contapassi del telefono.
Dopo aver parlato della struttura dell'applicazione e della sua implementazione, in quest'ultimo capitolo discuterò dei risultati ottenuti dai test compiuti sul calcolo della lunghezza del passo con l'applicativo e ne trarrò delle conclusioni.

7.1 Risultati

Nelle immagini in Fig. 7.1 che seguono ho reso noti i dataset che sono riuscito a procurare per testare come si comporta l'applicazione nel calcolo della falcata. Qui riportate vi sono 15 corse compiute da me nella fase di testing dell'applicazione. La distanza percorsa in ogni corsa é variata dai 250 ai 550 metri.
Come possiamo notare, 11 corse su 15 posseggono una deviazione standard minore della soglia, ovvero 15, e quindi possono rientrare nel calcolo della media della lunghezza del passo. La media di queste misurazioni è di circa 74, ovvero 3 centimetri in meno rispetto alla mia lunghezza del passo calcolata inserendo il mio sesso e la mia altezza, che è risultata 77. Da questo possiamo dedurre che la il calcolo falcata é stato approssimativamente accurato.

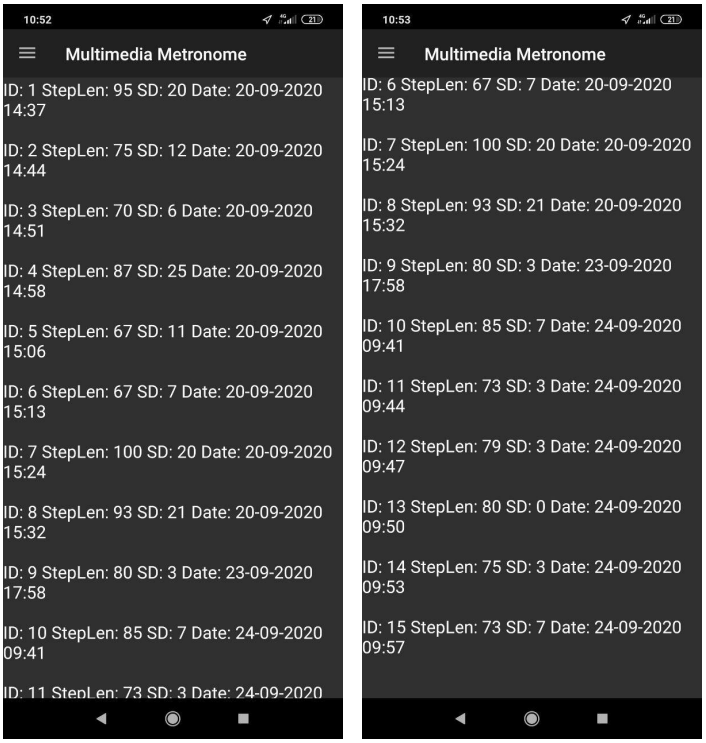


Figura 7.1: Dataset creato testando l'applicazione.

7.2 Conclusioni

Le principali difficoltà riscontrate nello studio da me compiuto sono state la non possibilità di avere molto tempo a disposizione per fare più test sul comportamento dell'app nel momento in cui il dataset diventi grande. La non disposizione di hardware estremamente precisi; mi riferisco al GPS e al contapassi, questo perché ho notato durante i test che l'accuratezza di entrambi è molto variabile. Infatti nelle corse dove troviamo una deviazione standard molto alta nel 100% dei casi, l'inaccuratezza dei calcoli era dovuta alla posizione GPS, che aveva circa 100 metri di errore, errore misurato tramite un'applicazione chiamata Gps Essentials [33], oppure ad un conteggio dei passi sbagliato. Oltre all'hardware ho avuto problemi con il calcolo delle distanze perché vengono calcolate sulla base della lunghezza della Polyline che ha un problema di una leggera latenza nell'aggiornamento rispetto all'aggiornamento della posizione, ed alcune volte é capitato che non si aggiornasse affatto.

In conclusione posso affermare che i test effettuati sono soddisfacenti perché, anche con il rischio che l'accuratezza dei sensori sia bassa, o l'aggiornamento delle

richieste effettuate sia in latenza, l'applicazione si è comportata molto bene anche andando a considerare i singoli casi. Se presi singolarmente i record a Fig. 7.1 possiamo notare che coloro che hanno una deviazione standard sotto la soglia si discostano di molto poco dalla lunghezza del passo calcolata con sesso e altezza, 5-10 centimetri al massimo.

Le possibili estensioni future possono essere molteplici. Alcuni esempi potrebbero essere:

- Introdurre un sistema di monitoraggio dei mezzi pubblici in modo tale che l'utente riesca a raggiungere una fermata di un mezzo in tempo.
- Introdurre un sistema di predizione delle tempistiche basata sulla distanza da percorrere e sulla lunghezza del passo inserita inizialmente.
- Introdurre un algoritmo che sceglie le canzoni da riprodurre in base alle preferenze di genere musicale dell'utente e non solo dalla playlist selezionata.

Qui presentate vi sono alcune delle possibili estensioni che possono essere applicate a questo applicativo.

Bibliografia

[1] Iqbal H Sarker. Context-aware rule learning from smartphone data: survey, challenges and future directions. *Journal of Big Data*, 6(1):95, 2019.

[2] Pei Zheng and L. Ni. Spotlight: the rise of the smart phone. *IEEE Distributed Systems Online*, 7:3–3, 2006.

[3] International telecommunication union. *Measuring the information society. Technical report,* *http://www.itu.int/en/itu-d/statistics/documents/publications/misr2015/misr2015-w5.pdf*, 2015.

[4] Google trends. *https://trends.google.it/trends/*, 2020.

[5] Buildfire. *https://buildfire.com/app-statistics/#:~:text=There%20are%202.8%20million%20apps,on%20the%20Google%20Play%20Store*, 2020.

[6] Enciclopedia treccani. *https://www.treccani.it/enciclopedia/contesto_(Enciclopedia-dell'Italiano)/*, 2020.

[7] G.D. Dey, A.K. Abowd. Towards a better understanding of context and context-awareness. chi 2000 workshop on the what, who, where, when, and how of context-awareness. 2000.

[8] Anind K Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

[9] Crina Grosan and Ajith Abraham. Rule-based expert systems. In *Intelligent Systems*, pages 149–185. Springer, 2011.

[10] Martin Halvey, Mark T Keane, and Barry Smyth. Time-based segmentation of log data for user navigation prediction in personalization. In *The 2005 IEEE/-WIC/ACM International Conference on Web Intelligence (WI'05)*, pages 636–640. IEEE, 2005.

[11] Katayoun Farrahi and Daniel Gatica-Perez. A probabilistic approach to mining mobile phone data sequences. *Personal and ubiquitous computing*, 18(1):223–238, 2014.

[12] Jiyoung Lee, Seungryong Kim, Sunok Kim, Jungin Park, and Kwanghoon Sohn. Context-aware emotion recognition networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 10143–10152, 2019.

[13] Markus Raab and Gergö Barany. Introducing context awareness in unmodified, context-unaware software. *arXiv preprint arXiv:1702.06806*, 2017.

[14] Tom Rodden, Keith Cheverst, K Davies, and Alan Dix. Exploiting context in hci design for mobile systems. In *Workshop on human computer interaction with mobile devices*, volume 12. Citeseer, 1998.

[15] Jason Pascoe. Adding generic contextual capabilities to wearable computers. In *Digest of papers. second international symposium on wearable computers (cat. no. 98ex215)*, pages 92–99. IEEE, 1998.

[16] Alejandra Pensabe-Rodriguez, Eduardo Lopez-Dominguez, Yesenia Hernandez-Velazquez, Saul Dominguez-Isidro, and Jorge De-la Calleja. Context-aware mobile learning system: Usability assessment based on a field study. *Telematics and Informatics*, 48:101346, 2020.

[17] Ricardo Colomo-Palacios, Francisco José García-Peñalvo, Vladimir Stantchev, and Sanjay Misra. Towards a social and context-aware mobile recommendation system for tourism. *Pervasive and Mobile Computing*, 38:505–515, 2017.

[18] Kevin Ashton et al. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009.

[19] Runners Pub. Calcolatore per la corsa. `https://play.google.com/store/apps/details?id=pub.runners.runnerspub&hl=it`.

[20] StepsApp GmbH. Stepsapp. `https://steps.app/it`.

[21] csomorbalazs.hu. Runbeat for spotify. `https://play.google.com/store/apps/details?id=hu.csomorbalazs.runbeat&hl=it`.

[22] Pacing Technologies. Pacedj. `https://www.pacedj.com/`.

[23] StepsApp GmbH. How calcolate step length from height and gender. https://steps.app/en/support/android/how-does-stepsapp-calculate-the-distance.

[24] Google Developers. Save data in local database using room. https://developer.android.com/training/data-storage/room.

[25] Google. Google cloud platform. https://cloud.google.com/?hl=it.

[26] Vysh01. Direction helpers. https://github.com/Vysh01/android-maps-directions/tree/master/app/src/main/java/com/thecodecity/mapsdirection/directionhelpers.

[27] Android Developers. Notification channel. https://developer.android.com/reference/android/app/NotificationChannel.

[28] Android Developers. Mediaplayer. https://developer.android.com/reference/android/media/MediaPlayer.

[29] Spotify. Spotify for developers. https://developer.spotify.com/.

[30] Spotify. Authentication scopes. https://developer.spotify.com/documentation/general/guides/scopes/#streaming.

[31] Spotify. Android sdk quick start. https://developer.spotify.com/documentation/android/quick-start/.

[32] Spotify. Web api reference. https://developer.spotify.com/documentation/web-api/reference/.

[33] mictale.com. Gps essentials. https://http://www.gpsessentials.com/.