

ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA

---

CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA  
SCIENZA E INGEGNERIA

CORSO DI LAUREA IN  
INGEGNERIA E SCIENZE INFORMATICHE

**Run/Walk Tracking**  
Progettazione e sviluppo di  
un'applicazione mobile per  
la gestione di allenamenti  
outdoor e indoor

Elaborato in  
Programmazione di sistemi mobile

**Relatore:**  
Prof. Mirko Ravaioli

**Presentata da:**  
Marica Pasquali

Anno Accademico 2018/2019



*Alla mia famiglia, che in questi anni,  
mi ha supportato ma anche sopportato.*



# Indice

<b>Introduzione</b>	<b>5</b>
<b>1 Progettazione</b>	<b>8</b>
1.1 Progettazione della componente dati . . . . .	8
1.1.1 Progettazione concettuale . . . . .	8
1.1.2 Progettazione logica . . . . .	13
1.1.3 Progettazione della persistenza . . . . .	16
1.2 Progettazione della componente applicativa . . . . .	19
1.2.1 Progettazione dei casi d'uso . . . . .	19
1.2.2 Progettazione dei mock-up . . . . .	26
<b>2 Implementazione</b>	<b>31</b>
2.1 Strumenti di Sviluppo . . . . .	31
2.1.1 Sistema Operativo . . . . .	31
2.1.2 Linguaggi . . . . .	33
2.1.3 Database . . . . .	35
2.1.4 Librerie di terze parti . . . . .	38
2.1.5 Versioning . . . . .	39
2.2 Scambio dati fra Client e Server . . . . .	42
2.2.1 Struttura del payload dei pacchetti . . . . .	42
2.2.2 Connessione al Server . . . . .	46
2.3 Persistenza dei dati . . . . .	54
2.3.1 Client . . . . .	54
2.3.2 Server . . . . .	58
2.4 Autenticazione . . . . .	60
2.5 Password Dimenticata . . . . .	64
2.6 Allenamento in Real-Time . . . . .	66
2.6.1 Musica . . . . .	78
2.6.2 Allenatore vocale . . . . .	80
2.7 Statistiche . . . . .	84
2.8 Supporto Multi-Lingua . . . . .	86

2.9	Unità di misura . . . . .	88
<b>Conclusioni</b>		<b>91</b>
<b>Glossario</b>		<b>93</b>
<b>Bibliografia</b>		<b>99</b>

# Introduzione

Il progetto di tesi consiste nello sviluppo di un'applicazione Android dedicata alla gestione di sessioni di allenamento (corsa/camminata) indoor e outdoor.

Negli ultimi anni si sono moltiplicati gli utenti che utilizzano questo genere di applicazioni, per migliorare e monitorare le loro performance. I benefici che questa nuova tendenza sta portando sono molteplici e non solo legati all'allenamento: la spinta motivazionale che queste tecnologie sono in grado di fornire è molto potente.

Lo scopo dell'applicazione sarà quello di garantire l'archiviazione e la visualizzazione degli allenamenti eseguiti. L'utente potrà inserire manualmente un allenamento (data, durata, sport, distanza, consumo calorico) oppure effettuare un allenamento in real-time che include anche il percorso effettuato se il GPS è attivato (allenamento outdoor).

L'applicazione sarà in grado di creare dei grafici dei vari allenamenti in base alla velocità media, alla distanza percorsa e alle calorie consumate. L'utente potrà inoltre visualizzare l'andamento dell'ultima settimana, dell'ultimo mese, dell'ultimo anno o di tutti gli allenamenti effettuati.

L'utente potrà inserire anche il suo peso corporeo e le sue variazioni. L'aggiornamento del peso è strettamente consigliabile per il corretto calcolo delle calorie consumate durante l'allenamento in real-time.

L'applicazione darà la possibilità di creare e modificare delle playlists utilizzando le canzoni dello smartphone. Quindi durante un allenamento l'utente potrà riprodurre o disattivare la playlist che precedentemente ha inserito.

Un'altra funzionalità sarà quella di attivare o disattivare un allenatore vocale che, durante l'allenamento, sintetizzerà in un certo intervallo di tempo (a scelta dell'utente o di default 1 minuto) le informazioni sull'allenamento stesso (tempo trascorso e/o distanza percorsa e/o calorie consumate).

L'applicazione supporterà il multi-lingua (Inglese, Italiano, Spagnolo). Non tutti i paesi adottano le medesime unità di misura, perciò l'applicazione darà la possibilità di cambiarle; per esempio l'unità di misura per le distanze potrà essere il Chilometro (Km) oppure il Miglio (mi), per il peso il Chilogrammo (Kg) oppure la Libbra (lb), per l'altezza il metro (m) oppure il piede (ft).

Prima di usufruire delle funzionalità che offre l'applicazione, l'utente si dovrà registrare seguendo 3 fasi di inserimento:

- inserimento dell'informazioni di base ( nome, cognome, data di nascita, email, città, numero di telefono), e anche un immagine profilo se lo desidera;
- inserimento dell'informazioni riguardanti l'aspetto fisico (sesso, altezza, peso) e l'obiettivo che si vuole raggiungere (dimagrire o allenarsi per una maratona).
- inserimento delle credenziali di accesso (username, password).

Dopo di che gli verrà inviato un codice di verifica tramite email.

Nel caso l'utente dimentichi la password dell'account, potrà richiedere l'invio di un link per la reimpostazione della pas-

sword (non necessariamente tramite l'email inserita in fase di registrazione).

Le informazioni inserite durante la registrazione potranno essere modificate durante l'utilizzo dell'applicazione.

Esistono molte applicazioni che offrono questo tipo di funzionalità come Endomondo, Adidas Running (ex Runtastic), Strava, RunKeeper, ma alcune funzionalità sono a pagamento. Inoltre una funzionalità che queste applicazioni non possiedono è la possibilità di inserire e visualizzare i progressi del peso corporeo.

Questo progetto non è nato solo come progetto di esame per Programmazione di Sistemi Mobile ma anche come motivazione personale al movimento quotidiano.

La tesi è strutturata come di seguito specificato:

- Nel primo capitolo viene analizzata l'intera fase di progettazione del sistema. È trattata la componente dati e la componente applicativa, anche attraverso l'utilizzo di mock-up.
- Nel secondo capitolo viene descritta la fase di implementazione del sistema, presentando gli strumenti di lavoro e descrivendo, con l'ausilio del codice utilizzato, le funzionalità dell'applicazione.
- Infine le conclusioni che analizzano i margini di miglioramento dell'applicazione, delineando alcuni possibili sviluppi futuri.

# **Capitolo 1**

# **Progettazione**

In questo capitolo inizialmente ci si soffermerà sulla componente dati e quindi sulla progettazione logica e concettuale sia del lato client che del lato server. Infine si analizzerà la componente applicativa attraverso i mock-up e lo studio dei casi d'uso.

## **1.1 Progettazione della componente dati**

Un prima fase della progettazione è relativa alla componente dati che si occupa della progettazione di tutte le entità che compongono la realtà considerata. Questa parte della progettazione si suddivide in 3 fasi che saranno esaminate nel seguito di questo paragrafo.

### **1.1.1 Progettazione concettuale**

La prima fase è la progettazione concettuale, nella quale si cerca di rappresentare il contenuto informativo della base di dati senza preoccuparsi né delle modalità di gestione delle informazioni né dell'implementazione.

Un primo strumento da utilizzare è il diagramma Entità-Relazione (E/R), grazie al quale è possibile esprimere il dominio, tenendo conto dei requisiti, come un insieme di entità e di relazioni.

Una volta che i concetti del dominio sono stati modellati possono essere usati come base di una progettazione orientata agli oggetti e implementati in un programma.

### Schemi scheletro

Dopo aver esaminato i requisiti , si presentano i seguenti schemi scheletro:

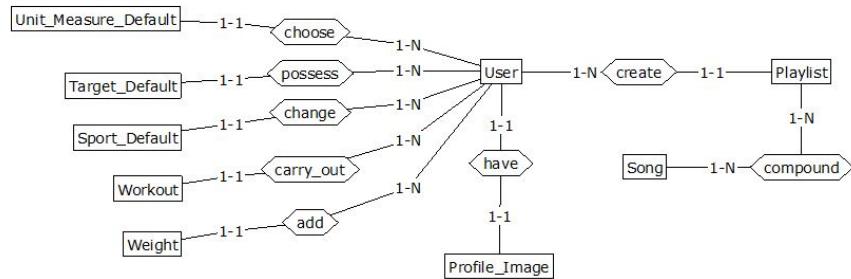


Figura 1.1: Diagramma Scheletro E/R del Client

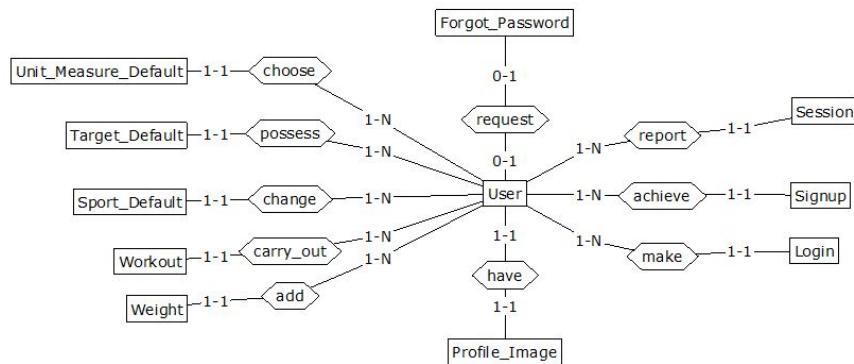


Figura 1.2: Diagramma Scheletro E/R del Server

## Schemi finali

Dopo l'inserimento degli attribuiti necessari per le varie entità, gli schemi finali E/R risultano essere i seguenti :

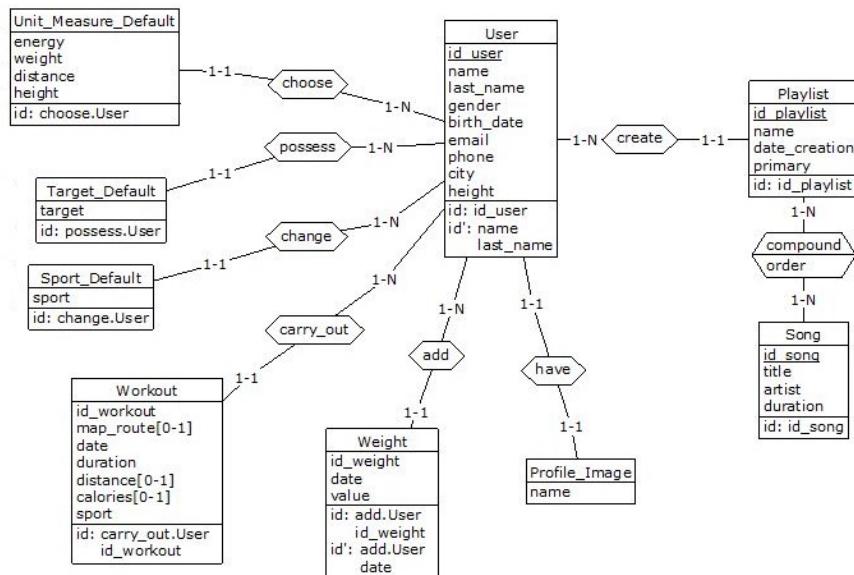


Figura 1.3: Diagramma Finale E/R del Client

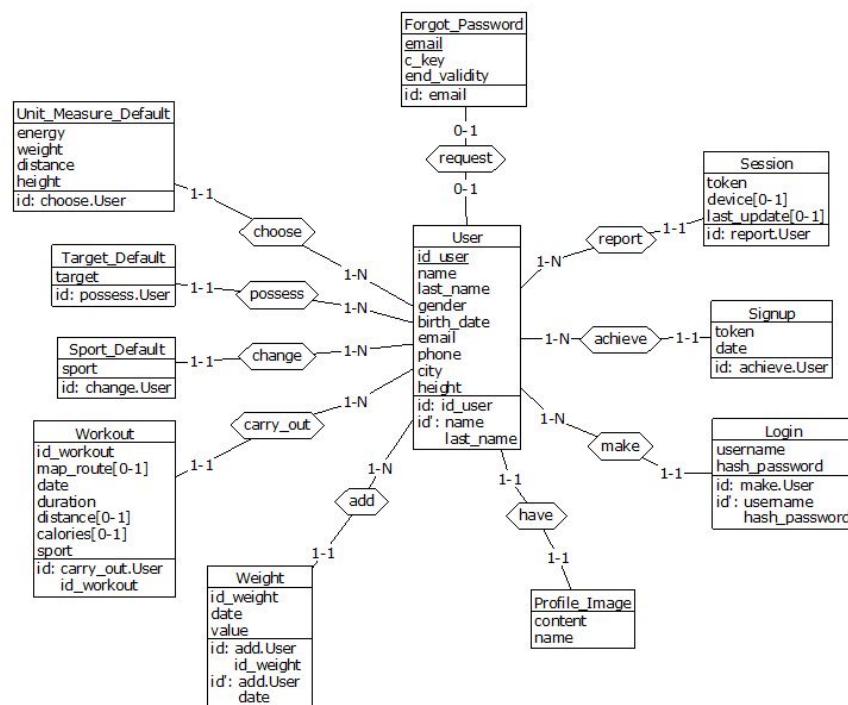


Figura 1.4: Diagramma Finale E/R del Server

## Confronto Client e Server

La base di dati del Server possiede *Forgot\_Password*, *Session*, *Sigup*, *Login* poiché esse rappresentano entità di utilizzo remoto. In particolare l'entità *Session* non è stata inserita nel client perché si è considerato che l'applicazione in uno smartphone viene usata da un utente alla volta e che l'utente, in questo tipo di applicazioni, non ha più di un account.

Mentre la base di dati del Client possiede *Playlist*, *Song*. In conformità con i requisiti, la creazione e la modifica delle playlists è solo modellata sulle canzoni dello smartphone in utilizzo.

### 1.1.2 Progettazione logica

La seconda fase è la progettazione logica, che ha come obiettivo la definizione di uno schema logico capace di rappresentare, in modo fedele ed efficiente, le informazioni contenute nel modello concettuale. Tali dati sono rappresentati facendo uso del concetto matematico di relazione (o di tabella).

La prima operazione da eseguire consiste nella ristrutturazione dello schema E/R; in questo caso non sono presenti generalizzazioni, né attributi multivaleore, per cui tale attività non è necessaria; la seconda, invece, consiste nell'effettiva creazione dello schema logico equivalente ovvero nella traduzione dello schema concettuale ristrutturato in uno schema logico.

Le seguenti traduzioni sono di entità e relazioni comuni al lato server e al lato client.

**USER**( id\_user, name, last\_name, gender, birth\_date, email, phone, city, height )

AK: name, last\_name

**UNIT\_MEASURE\_DEFAULT**( id\_user, energy, weight, distance, height)

FK: id\_user REFERENCES **USER**

**TARGET\_DEFAULT**( id\_user, target)

FK: id\_user REFERENCES **USER**

**SPORT\_DEFAULT**( id\_user, sport)

FK: id\_user REFERENCES **USER**

**WEIGHT**( id\_weight, id\_user, date, value)

FK: id\_user REFERENCES **USER**

AK: id\_user, date

**WORKOUT**( id\_workout, id\_user, map\_route\*, date, duration, distance\*, calories\*, sport)

FK: id\_user REFERENCES **USER**

## Client

**PROFILE\_IMAGE**( id\_user, name)

FK: id\_user REFERENCES **USER**

**PLAYLIST**(id\_playlist, id\_user, name, date\_creation, primary)

FK: id\_user REFERENCES **USER**

**SONG**(id\_song, title, artist, duration, path)

**COMPOUND**(id\_playlist, id\_song, order)

FK: id\_playlist REFERENCES **PLAYLIST**

FK: id\_song REFERENCES **SONG**

## Server

**PROFILE\_IMAGE**( id\_user, content, name)

FK: id\_user REFERENCES **USER**

**LOGIN**( id\_user, username, hash\_password)

FK: id\_user REFERENCES **USER**

AK: username, hash\_password

**SIGNUP(** id\_user, token, date)

FK: id\_user REFERENCES **USER**

**SESSION(** id\_user, token, device\*, last\_update\*)

FK: id\_user REFERENCES **USER**

**FORGOT\_PASSWORD(** email, id\_user\*,c\_key, end\_validity)

FK: id\_user REFERENCES **USER**

### 1.1.3 Progettazione della persistenza

Per l'accesso ai dati si è deciso di utilizzare il pattern Data Access Object (DAO), un pattern architettonale che implementa il meccanismo di accesso richiesto per lavorare con la sorgente dei dati. Il Data Access Object nasconde completamente i dettagli dell'interazione con la sorgente dati. Il livello di applicazione, che contiene la business-logic, utilizza l'interfaccia esposta dal DAO, la quale non cambia quando l'implementazione dell'origine dati sottostante cambia, e questo consente al DAO di adattarsi a diversi schemi di archiviazione senza dover modificare nulla sugli altri livelli. In sostanza, il DAO funge da adapter tra il componente della business logic e l'origine dati.

Il vantaggio relativo all'uso del DAO è dunque il mantenimento di una rigida separazione tra le componenti di un'applicazione.

Per raggiungere lo scopo prefissato si definisce la struttura delle singole classi che accedono al database; in particolare, le classi da progettare in questo passo sono almeno una per ogni tabella.

Oltre alle classi corrispondenti alle tabelle, è buona abitudine progettare un'ulteriore classe DAOFactory che si occupa della gestione del database sottostante. In pratica, DAOFactory ha lo scopo di fare da ponte tra il modello relazionale (tipico dei database) e il paradigma orientato agli oggetti (tipico della programmazione).

Tenendo conto di queste linee guida si ottengono i seguenti schemi UML (*Figura 1.5* e *Figura 1.6*):

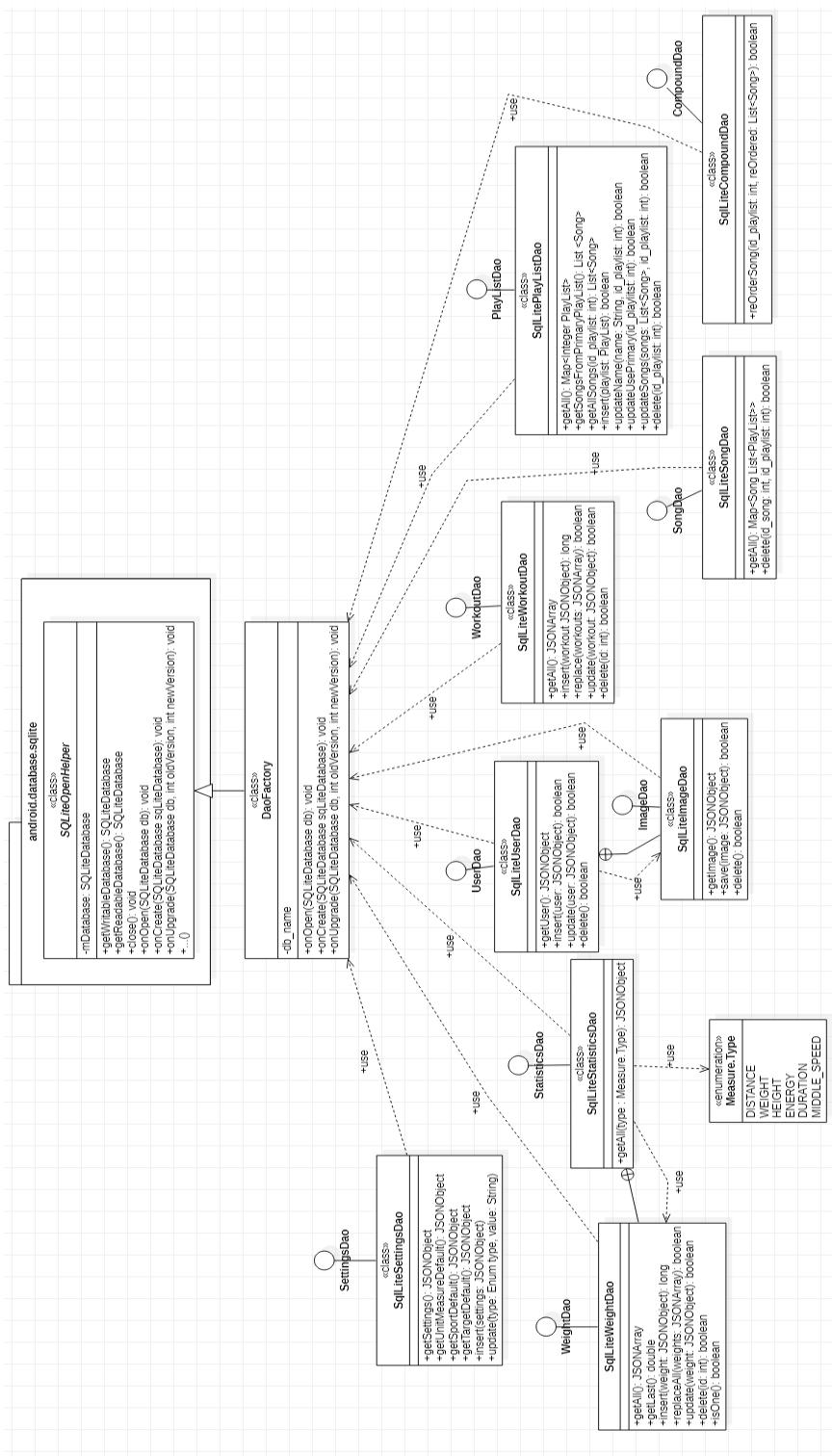


Figura 1.5: Diagramma Persistenza del Client

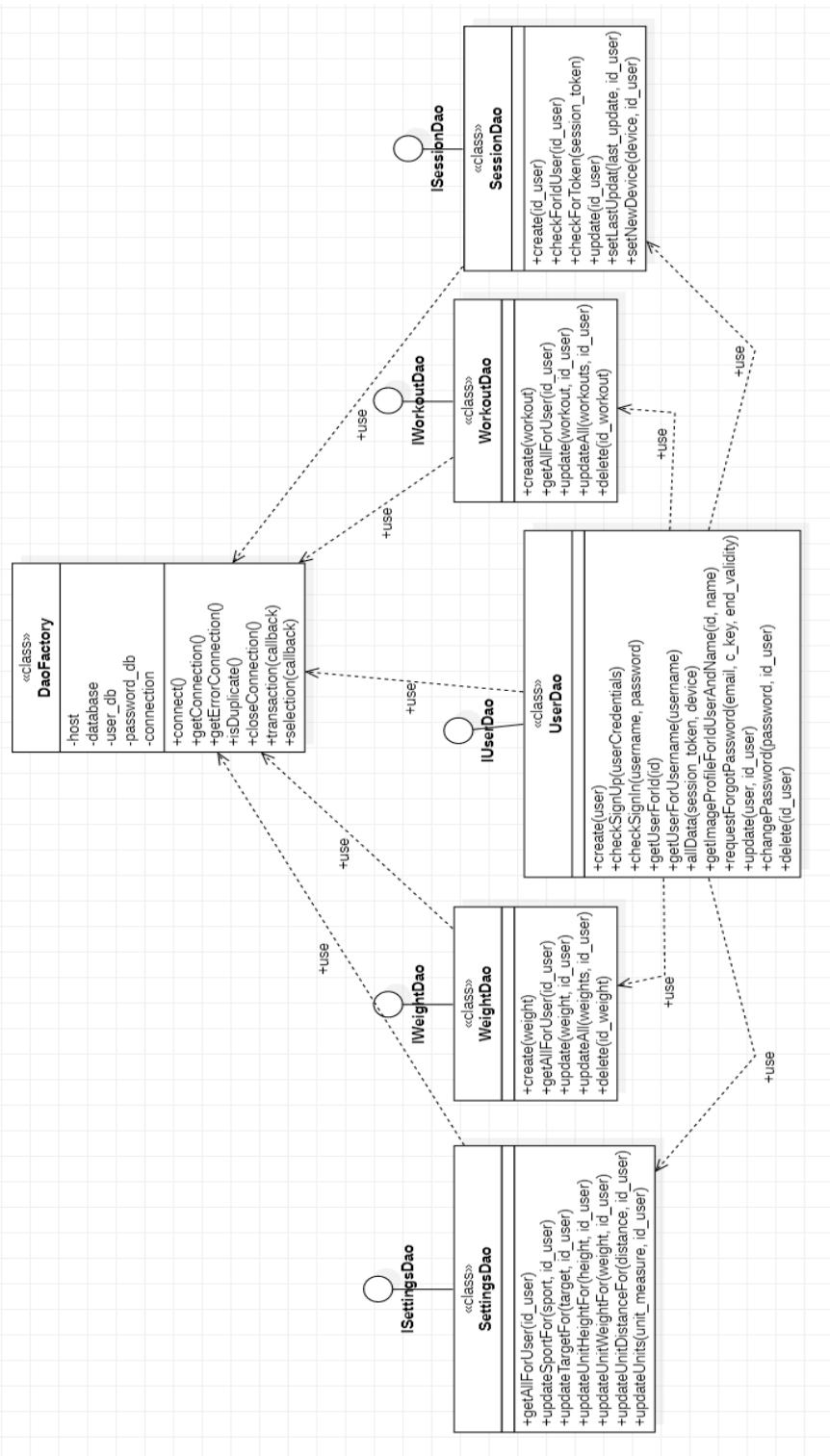


Figura 1.6: Diagramma Persistenza del Server

## 1.2 Progettazione della componente applicativa

Per progettazione della componente applicativa si intende quella fase della progettazione di un sistema informativo che si occupa di definire tutti i programmi che dovranno operare sui dati relativi al sistema informativo stesso. Questa fase comprende tutti quegli strumenti che avvicinano il programmatore al modo di pensare dell'utente finale.

### 1.2.1 Progettazione dei casi d'uso

Il primo strumento che completa la progettazione della componente applicativa è il caso d'uso. Esso rappresenta il ruolo di utilizzo del sistema da parte di uno o più utilizzatori (attori). Un caso d'uso descrive l'interazione tra gli attori e sistemi. Le attività che vengono descritte hanno lo scopo principale di derivare diagrammi dettagliati del sistema in fase di sviluppo. La progettazione di un caso d'uso stabilisce precisamente come la collaborazione tra classi di progettazione permette di raggiungere gli obiettivi del sistema.

Innanzitutto occorre individuare gli attori principali.

Nel caso della realtà presa in esame, l'attore principale è l'utente, ovvero la persona che interagisce con il sistema per attivare i casi d'uso. Per quanto riguarda le funzionalità che il sistema oggetto della presente tesi fornisce all'utente, si ottiene il seguente diagramma dei casi d'uso:

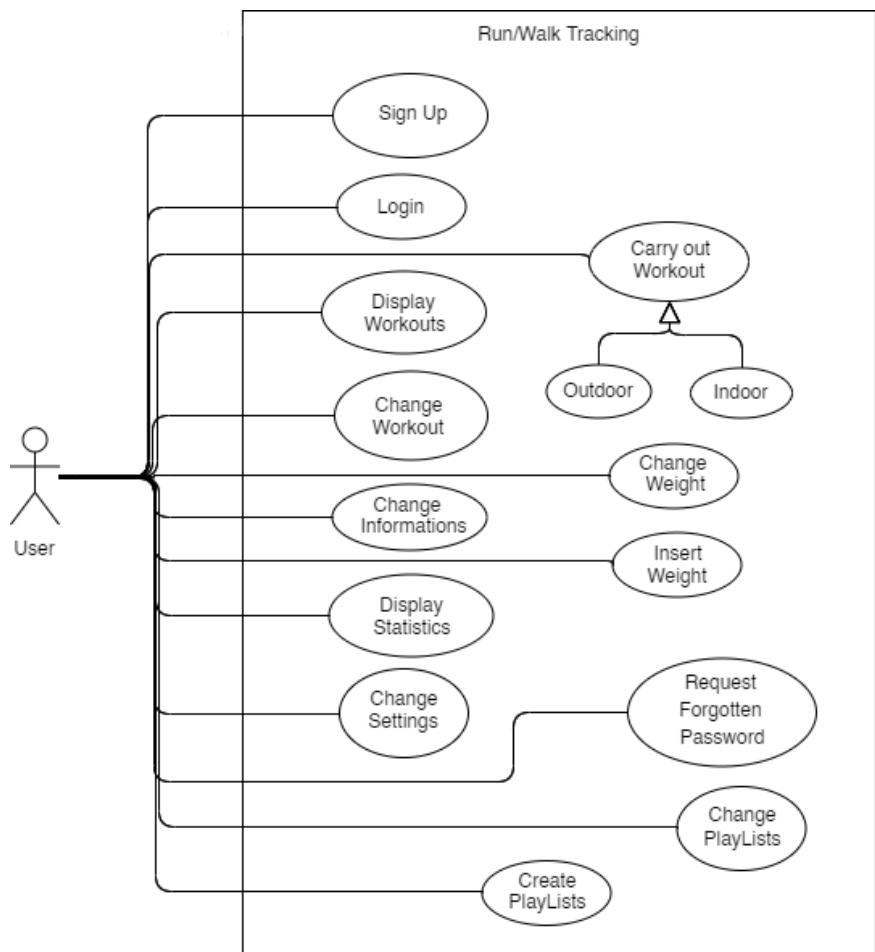


Figura 1.7: Diagramma dei casi d'uso

Per rappresentare in modo chiaro la struttura dei casi d'uso si è deciso di utilizzare i diagrammi di flusso e i diagrammi di stato, molto utili per descrivere in maniera semplice ed intuitiva le varie operazioni e decisioni che sono state previste ed implementate. Di seguito vengono presentati i diagrammi di flusso e diagrammi di stato dei casi d'uso più significativi.

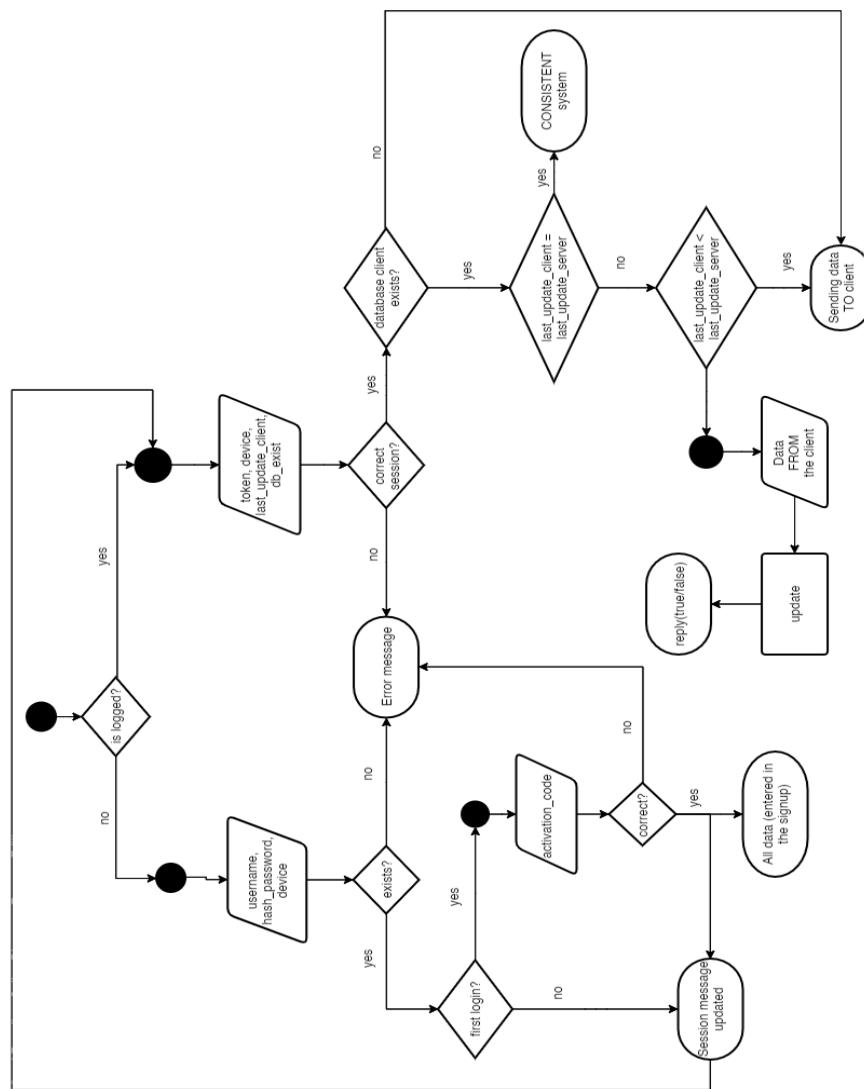


Figura 1.8: Diagramma di flusso del Login

Il diagramma (*Figura 1.8*) è relativo al caso d'uso *Login*. Da esso si evince come il sistema controlla se l'utente abbia già effettuato l'accesso oppure no e la corrispondente sincronizzazione dei dati.

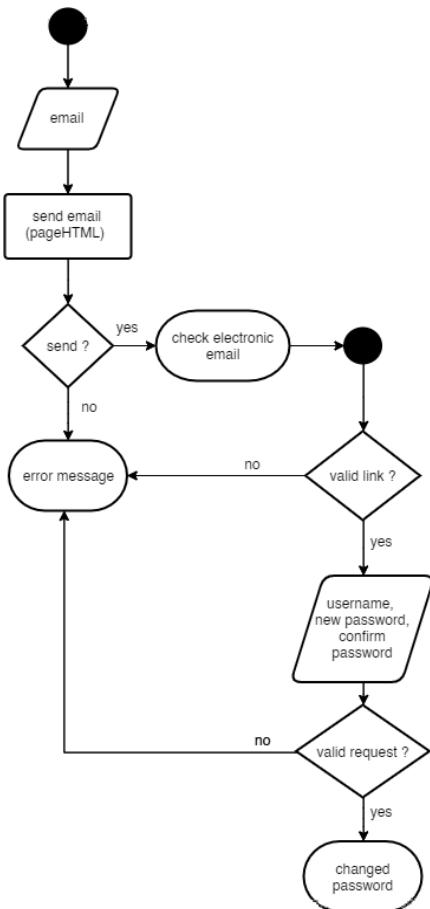


Figura 1.9: Diagramma di flusso *Request Forgotten Password*

Il diagramma (*Figura 1.9*) è relativo al caso d'uso *Request Forgotten Password*. Da esso si deduce come viene gestita la richiesta dell'utente quando dimentica la password.

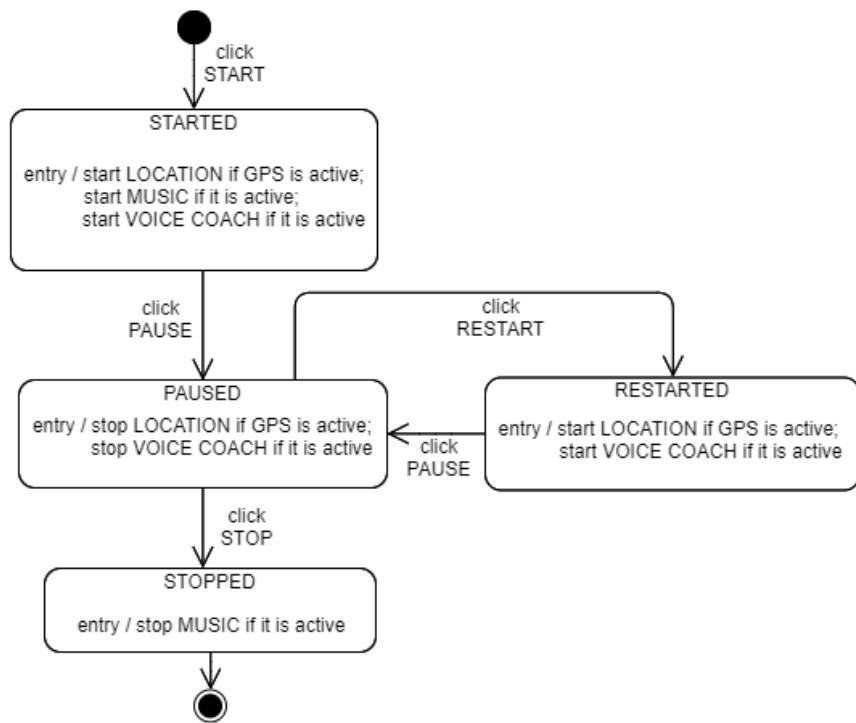


Figura 1.10: Diagramma di stato *Carry out Workout*

Il diagramma (*Figura 1.10*) è relativo al caso d'uso *Carry out Workout*. Da esso si deduce come attraverso pulsanti, l'utente può gestire l'allenamento in Real Time.

I diagrammi (*Figura 1.11 e Figura 1.12*) sono relativi sempre al caso d’uso *Carry out Workout*, ma rappresentano come le componenti dell’“Allenatore vocale” e della “Musica” funzionano durante l’allenamento.

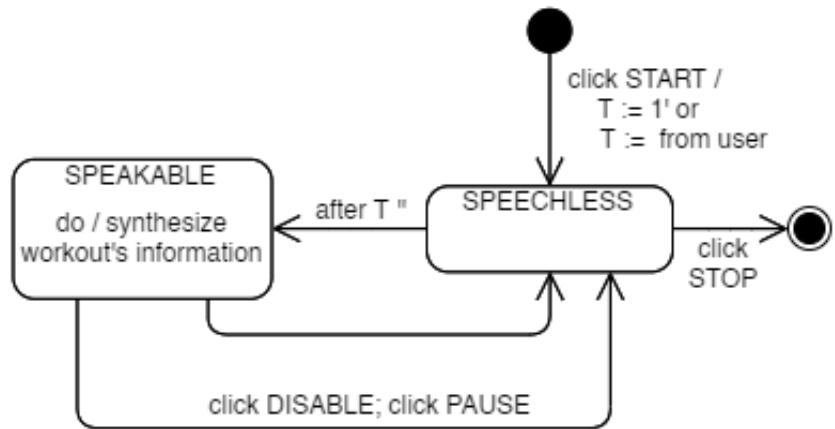


Figura 1.11: Diagramma di stato dell’ “Allenatore vocale”

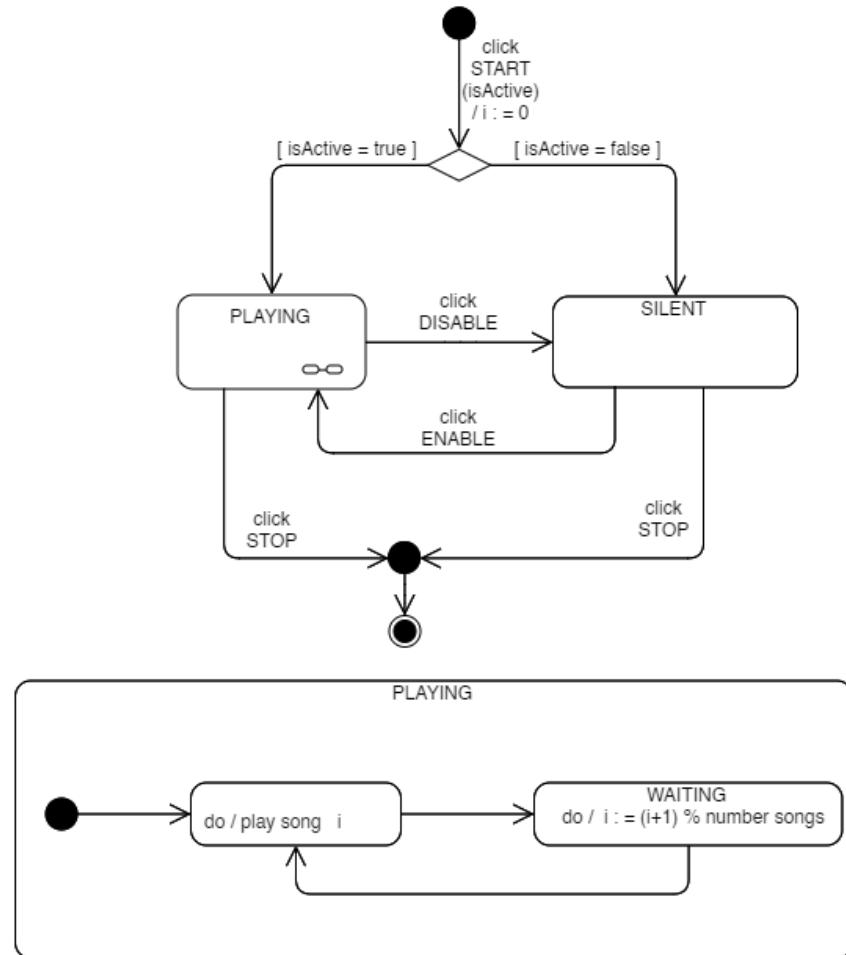


Figura 1.12: Diagramma di stato della “Musica”

### **1.2.2 Progettazione dei mock-up**

Il secondo strumento è quello dei mock-up. Con questo termine si intende comunemente la riproduzione di un oggetto originale ad uso didattico, dimostrativo, scenografico o di comunicazione visiva. In genere i mock-up vengono utilizzati come modelli o prototipi. Si realizza un mock-up quando è necessario disporre di una copia verosimile dell'oggetto, che attratta l'attenzione, che sia in grado di rappresentare un progetto completo, oppure una sua parte. In ambito informatico il significato di tale termine non si discosta molto; infatti un mock-up è la rappresentazione grafica, pulita e stilizzata, delle interfacce di un sistema informativo. Il mock-up viene sottoposto al cliente, prima di passare alla realizzazione del template vero e proprio, per fargli capire la strategia di comunicazione del suo nuovo sistema e mostrargli la disposizione dei principali elementi.

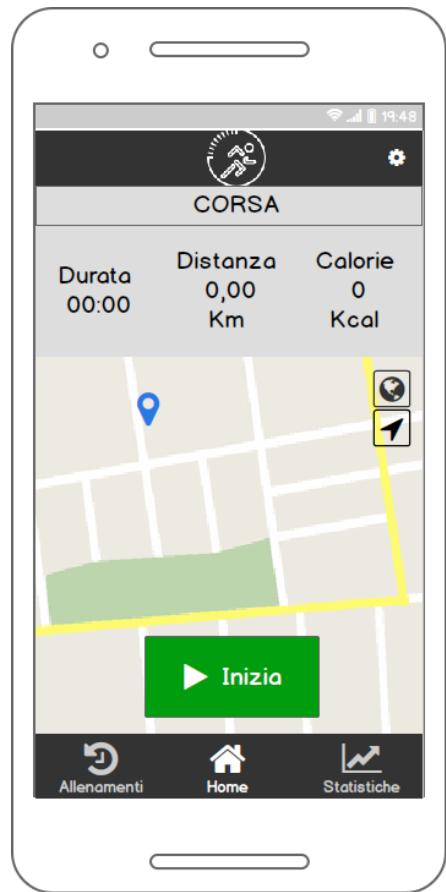


Figura 1.13: Mock-Up della Home

Lo scopo della schermata (*Figura 1.13*) è quello di permettere all'utente di iniziare un nuovo allenamento outdoor. Se il GPS è disattivato, verrà effettuato un allenamento indoor ovvero il percorso effettuato non verrà memorizzato.

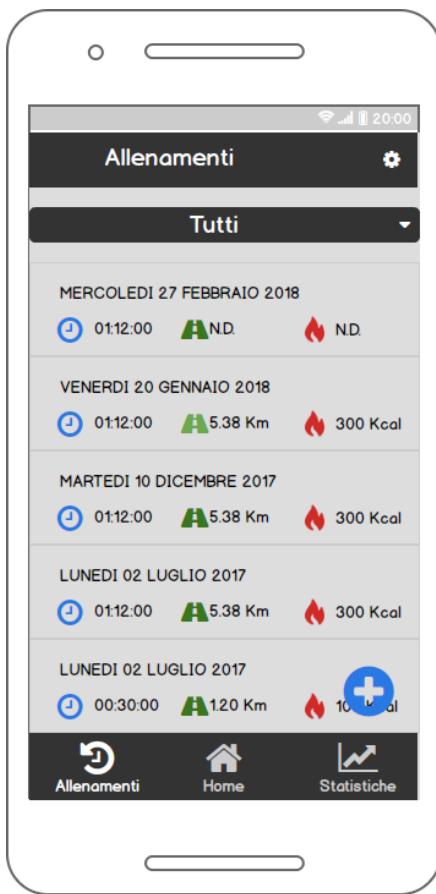


Figura 1.14: Mock-Up degli Allenamenti

Lo scopo della schermata (*Figura 1.14*) è quello di permettere all’utente di visualizzare gli allenamenti effettuati e inseriti. Inoltre, attraverso il menù a tendina, si può filtrare gli allenamenti in base all’anno o al mese. Il pulsante in basso a destra permette di memorizzare un nuovo allenamento in modo manuale.

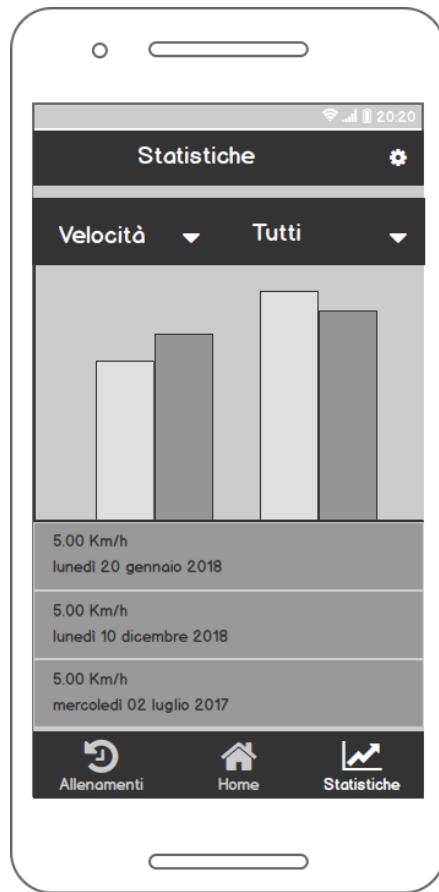


Figura 1.15: Mock-Up delle Statistiche

Lo scopo della schermata (*Figura 1.15*) è quello di permettere all’utente di visualizzare le statistiche degli allenamenti e del peso inserito, utilizzando un grafico.

Il menù a tendina a sinistra permette di visualizzare le varie statistiche ( velocità media, calorie, distanza e peso ) e quello a destra permette di visualizzare le statistiche filtrate per l’ultimo anno, l’ultimo mese o l’ultima settimana.



Figura 1.16: Mock-Up delle Impostazioni

Lo scopo della schermata (*Figura 1.16*) è quello di permettere all'utente di visualizzare e modificare le informazioni inserite o le impostazioni di default.

# Capitolo 2

## Implementazione

In questo capitolo verrà descritta la fase di implementazione del sistema. Inizialmente si presenteranno gli strumenti utilizzati definendo una panoramica generale, per poi passare allo studio delle funzionalità dell'applicazione, esaminando e approfondendo sezioni di codice e utilizzando screenshot dell'applicazione.

### 2.1 Strumenti di Sviluppo

#### 2.1.1 Sistema Operativo

**Android** è un sistema operativo per dispositivi mobili sviluppato da Google LLC e basato sul kernel Linux, da considerarsi propriamente una distribuzione embedded Linux e non un sistema unix-like né una distribuzione GNU/Linux (dato che la quasi totalità delle utilità GNU è sostituita da software in Java), progettato principalmente per sistemi embedded quali smartphone e tablet, con interfacce utente specializzate per televisori (Android TV), automobili (Android Auto), orologi da polso (Wear OS), occhiali (Google Glass), e altri.

Le applicazioni sono la forma più generica per indicare i software applicativi installabili su Android. Esse possono essere sca-

ricate sia dal catalogo ufficiale Google Play, sia da altri cataloghi, come l'Amazon Appstore di Amazon.com, o F-Droid che contiene solo software su licenza libera. Le applicazioni Android possono essere installate direttamente a partire da un file APK fornito dal distributore del software. Per motivi di sicurezza informatica, le applicazioni sono fornite di un sistema di certificazione che verifica l'integrità del pacchetto da installare e la paternità rispetto a un distributore di software accreditato presso Google.

Le applicazioni vengono eseguite tramite la Dalvik Virtual Machine, una macchina virtuale adattata per l'uso su dispositivi mobili.

Poiché i dispositivi Android sono generalmente alimentati a batteria, Android è progettato per gestire i processi per ridurre al minimo il consumo di energia. Quando un'applicazione non è in uso, il sistema ne limita l'uso di risorse in modo tale che, sebbene disponibile per l'uso immediato, non utilizzi quantitativi significativi di batteria o risorse della CPU. Android gestisce automaticamente le applicazioni archiviate in memoria: quando la memoria è insufficiente, il sistema inizia, in modo invisibile, a chiudere automaticamente i processi inattivi, a partire da quelli che sono rimasti inattivi più a lungo.

Android 8.0 (livello API 26) e successivi impongono limitazioni su ciò che le app possono fare durante l'esecuzione in background, nel tentativo di ridurre il consumo energetico. Le app sono limitate in tre modi:

- **Limitazioni del servizio in background.** Mentre un'app è inattiva, ci sono limiti all'utilizzo dei servizi in background. Ciò non si applica ai servizi in foreground, che sono più evidenti per l'utente.



- **Limitazioni della trasmissione.** Con limitate eccezioni, le app non possono utilizzare il proprio manifest per registrare broadcast impliciti. Possono comunque registrare broadcast in fase di runtime e possono utilizzare il manifest per registrare broadcast esplicativi specificamente per loro.
- **Limitazioni della localizzazione.** Viene limitata la frequenza con cui le app in background possono recuperare la posizione corrente dell’utente. Le app possono ricevere aggiornamenti sulla posizione solo poche volte ogni ora.

Ad aprile 2017 è il sistema operativo per dispositivi mobili più diffuso al mondo, con una fetta di mercato attestarsi a quota 62,94% sul totale, seguito da iOS con il 33,9%; nello stesso mese supera anche il marketshare come sistema operativo più utilizzato per navigare in rete, superando di fatto Windows che, fino ad allora, aveva il più alto marketshare al mondo.

### 2.1.2 Linguaggi

**Java** è un linguaggio di programmazione ad alto livello, orientato agli oggetti e alla tipizzazione statica, che si appoggia sull’omonima piattaforma software di esecuzione, specificatamente progettato per essere il più possibile indipendente dalla piattaforma hardware di esecuzione (tramite compilazione in bytecode prima e interpretazione poi da parte di una JVM), sebbene questa caratteristica comporti prestazioni in termini di computazione inferiori a quelle di linguaggi direttamente compilati come C e C++ ovvero dunque perfettamente adattati alla piattaforma hardware.



Java venne creato per soddisfare cinque obiettivi primari:

- essere ”semplice, orientato agli oggetti e familiare”;
- essere ”robusto e sicuro”;
- essere indipendente dalla piattaforma;
- contenere strumenti e librerie per il networking;
- essere progettato per eseguire codice da sorgenti remote in modo sicuro.

**PHP** è un linguaggio per lo scripting server-side, ovvero un linguaggio che risiede in un server remoto e che in fase di esecuzione interpreta le informazioni ricevute da un client grazie al Web server, le elabora e restituisce un risultato al client che ha formulato la richiesta.

Quello di cui si occupa PHP è appunto elaborare i dati veicolati tramite una richiesta HTTP e fornire un’adeguata risposta al client.

PHP si differenzia poi dai linguaggi di scripting client side, come per esempio JavaScript, per via del fatto che la sorgente viene eseguita nel server, generando markup HTML da inviare al client; quest’ultimo riceverà i risultati dell’esecuzione di un’applicazione ma il codice eseguito non sarà mai visibile.

PHP è multipiattaforma, cioè può essere utilizzato sia in ambienti unix-based (Linux, Mac OSX) che su Windows. La combinazione più utilizzata, però, resta quella LAMP ovvero Linux come sistema operativo, Apache come Web server, MySQL per i database e PHP.

### 2.1.3 Database

**SQLite** è una libreria software scritta in linguaggio C che implementa un DBMS SQL di tipo ACID incorporabile all'interno di applicazioni.



Permette di creare una base di dati (comprese tabelle, query, form, report) incorporata in un unico file.

SQLite non è un processo stand-alone utilizzabile di per sé, ma può essere incorporato all'interno di un altro programma.

La libreria offre molte interessanti caratteristiche:

- è compatta (meno di 500KB per l'intera libreria alla versione 3.6.14);
- è molto veloce; in molti casi più di MySQL e PostgreSQL;
- il codice sorgente è liberamente disponibile, chiaro e ben commentato;
- è in grado di interpretare stringhe SQL; a differenza di altre librerie simili, supporta buona parte dello standard SQL92;
- ha la possibilità di interpretare campi in JSON;
- l'API è semplice da utilizzare;
- ha transazioni atomiche, consistenti, isolate e durabili (ACID), anche in caso di crash di sistema o blackout;
- è multipiattaforma;

- include un programma di utilità a riga di comando per accedere al database anche manualmente (come su MySQL, Postgresql e tanti altri DB) o tramite scripting;
- supporta database che possono essere anche molto grandi; attualmente il limite è 140TB con una riga di dimensione massima pari a 1Gb;
- un database consiste di un unico file, il cui formato interno è indipendente dalla piattaforma e dal relativo ordine dei byte;
- non ha dipendenze esterne;
- normalmente non richiede alcun lavoro di amministrazione, ma supporta il comando "SQL VACUUM" nel caso si renda necessario compattare esplicitamente i dati del database.

SQLite presenta dei limiti legati in parte alla finalità di semplice database da incorporare in altre applicazioni:

- non offre le stored procedure;
- non prevede la gestione dei permessi d'accesso, demandata al software con cui si interagisce con il database e/o al meccanismo dei permessi del file system;
- non ha una vera gestione della concorrenza: le applicazioni che lo utilizzano, se necessario, devono implementarla;
- per garantire la coerenza del file del database sono usati i lock del file system, e quindi vi possono essere problemi qualora quest'ultimo non li implementi correttamente, ad esempio con file system di rete (come NFS);
- non offre alcuna cache per le query (e non ne ha la possibilità, non esistendo un processo server centrale);

- non ha protocolli di rete, non essendo utilizzabile come programma a sé; è possibile utilizzare un database remoto, ma solo tramite file system di rete del sistema operativo, con prestazioni difficilmente accettabili;
- non supporta alcuni importanti costrutti SQL quali RIGHT JOIN e FULL OUTER JOIN;
- non supporta le sottoquery variabili.
- non supporta la scrittura diretta nelle viste (occorre usare trigger "INSTEAD OF");
- non consente di modificare, cancellare o rinominare le colonne di una tabella: il comando ALTER TABLE è infatti limitato alla modifica del nome della tabella e all'aggiunta di colonne in coda alla stessa. Molti front-end di terze parti aggirano comunque il limite rigenerando in modo trasparente la tabella (perdendo però trigger e indici ad essa correlati);
- non supporta trigger di tipo "FOR EACH STATEMENT" (solo "FOR EACH ROW", eventualmente combinato con "INSTEAD OF");
- il supporto ai trigger ricorsivi ed ai vincoli sulle chiavi esterne, introdotto rispettivamente nelle versioni 3.6.18 e 3.6.19, deve essere attivato dal programmatore; tale comportamento, dovuto alla compatibilità con le versioni precedenti, sarà modificato a partire dalla versione 3.7.

Una peculiarità di SQLite è la gestione flessibile dei tipi di dati: ogni campo può contenere qualsiasi tipo di dato o quasi; gestito differentemente nella versione 2 e 3.

**MySQL o Oracle MySQL** MySQL è il Relational database management system (RDBMS) più diffuso al mondo. Esso è composto da un client a riga di comando e un server; entrambi i software sono disponibili sia per sistemi Unix e Unix-like sia per Windows; le piattaforme principali di riferimento sono Linux e Oracle Solaris. MySQL è un software open source e libero, rilasciato a doppia licenza, compresa la GNU General Public License, ed è sviluppato per essere il più possibile conforme agli standard ANSI SQL e ODBC SQL.

I sistemi e i linguaggi di programmazione che supportano MySQL sono molto numerosi: ODBC, Java, Mono, .NET, PHP, Python e molti altri.



Le piattaforme LAMP e WAMP incorporano MySQL per l'implementazione di server per gestire siti web dinamici.

I motivi di tanto successo risiedono nella sua capacità di mantenere fede agli impegni presi sin dall'inizio:

- alta efficienza nonostante le moli di dati affidate;
- integrazione di tutte le funzionalità che offrono i migliori DBMS: indici, trigger e stored procedure;
- altissima capacità di integrazione con i principali linguaggi di programmazione, ambienti di sviluppo e suite di programmi da ufficio.

#### 2.1.4 Librerie di terze parti

Android Graph Library :

<https://github.com/jjoe64/GraphView>

Android ListView with drag and drop reordering :

<https://github.com/bauerca/drag-sort-listview>

Android Image Picker Library :

<http://myhexaville.com/2017/05/19/andorid-simplest-image-picker-library/>

Android Custom SeekBar Library :

<https://github.com/warkiz/IndicatorSeekBar>

Android Image loading and caching Library :

<https://github.com/bumptech/glide>

Maps SDK for Android Utility Library:

<https://github.com/googlemaps/android-maps-utils>

Google Api :

<https://developers.google.com/android/guides/setup>

Material Design:

<https://github.com/material-components/material-components-android>

### 2.1.5 Versioning

**Git** è un software di controllo versione distribuito utilizzabile da interfaccia a riga di comando, creato da Linus Torvalds nel 2005.



Nacque per essere un semplice strumento per facilitare lo sviluppo del kernel Linux ed è diventato uno degli strumenti di controllo versione più diffusi.

Il progetto di Git è la sintesi dell'esperienza di Torvalds nel mantenere un grande progetto di sviluppo distribuito, della sua intima conoscenza delle prestazioni del file system e di un bisogno urgente di produrre un sistema soddisfacente in poco tempo. Queste influenze hanno condotto alle seguenti scelte implementative:

- forte supporto allo sviluppo non lineare. Git supporta diramazione e fusione (branching and merging) rapide e comode, e comprende strumenti specifici per visualizzare e navigare una cronologia di sviluppo non lineare. Un’assunzione centrale in Git è che una modifica verrà fusa più spesso di quanto sia scritta, dato che viene passata per le mani di vari revisori;
- sviluppo distribuito. Git dà a ogni sviluppatore una copia locale dell’intera cronologia di sviluppo, e le modifiche vengono copiate da un tale repository a un altro. Queste modifiche vengono importate come diramazioni aggiuntive di sviluppo, e possono essere fuse allo stesso modo di una diramazione sviluppata localmente;
- i repository possono essere pubblicati facilmente tramite HTTP, FTP, ssh, rsync, o uno speciale protocollo git;
- gestione efficiente di grandi progetti. Git è molto veloce e scalabile. È tipicamente un ordine di grandezza più veloce degli altri sistemi di controllo versione, e due ordini di grandezza più veloce per alcune operazioni;
- autenticazione crittografica della cronologia. La cronologia di Git viene memorizzata in modo tale che il nome di una revisione particolare (secondo la terminologia Git, una ”commit”) dipende dalla completa cronologia di sviluppo che conduce a tale commit. Una volta che è stata pubblicata, non è più possibile cambiare le vecchie versioni senza che ciò venga notato;
- progettazione del toolkit. Git è un insieme di programmi di base scritti in linguaggio C, e molti script di shell che

forniscono comodi incapsulamenti. È facile concatenare i componenti per fare altre cose utili;

- strategie di fusione intercambiabili. Come parte della progettazione del suo toolkit, Git ha un modello ben definito di una fusione incompleta, e ha più algoritmi per tentare di completarla. Se tutti gli algoritmi falliscono, tale fallimento viene comunicato all’utente e viene sollecitata una fusione manuale. Pertanto è facile sperimentare con nuovi algoritmi di fusione;
- la ”spazzatura” si accumula fino a quando viene raccolta. Quando si abortisce un comando o si scartano delle modifiche si lasciano degli oggetti inutilizzabili nel database. Tipicamente, questi sono solo una piccola parte della cronologia continuamente crescente degli oggetti utili, ma il comando di recupero dello spazio, git gc –prune, può richiedere parecchio tempo.

## 2.2 Scambio dati fra Client e Server

L'applicazione presenta una architettura Client-Server. Il sistema è presentato come un insieme di servizi forniti da un server. I client sono gli utenti di questi servizi e accedono al server per utilizzare tali servizi. I servizi e il server possono essere modificati senza influire su altre parti del sistema. I client potrebbero aver bisogno di conoscere il nome del server disponibile e i servizi che può offrire. Il server, invece, non ha bisogno di conoscere l'identità del client o il numero di client che stanno utilizzando i suoi servizi. I client accedono a servizi offerti dal server attraverso chiamate di procedure remote, usando un protocollo richiesta-risposta (come HTTP), dove il client invia una richiesta al server e resta in attesa finché non riceve una risposta . Il vantaggio più importante del modello Client-Server è che si tratta di un'architettura distribuita. L'uso più efficiente si realizza nei sistemi in rete con molti processori distribuiti.

Il *Server* è allocato nel Web Hosting Gratuito *000webhost*. Precedentemente è stato progettato e sviluppato in locale con l'aiuto del software XMAPP.

### 2.2.1 Struttura del payload dei pacchetti

Il formato utilizzato per i payload è il formato JSON (JavaScript Object Notation), che è completamente indipendente dal linguaggio di programmazione. Per le persone è facile da leggere e scrivere e per le macchine risulta facile da generare e analizzarne la sintassi. Questa caratteristica fa di JSON un linguaggio ideale per lo scambio di dati.

Ogni script del server ha una struttura tutta sua dedicata ai payload. Di seguito sono illustrati i più significativi.

```
{  
    "name": "...",  
    "last_name": "...",  
    "gender": "...",  
    "birth_date": "...",  
    "email": "...",  
    "city": "...",  
    "phone": "...",  
    "height": "...",  
    "target": "...",  
    "weight": "...",  
    "username": "...",  
    "password": "...",  
    "image": {  
        "name": "...",  
        "content": "..."  
    }  
}
```

Figura 2.1: Struttura payload di input della Registrazione

```
{  
    "username": "...",  
    "password": "...",  
    "device": "..."  
}
```

Figura 2.2: Struttura payload di input del Login

```
{  
    "username": "...",  
    "password": "...",  
    "token": "...", // codice di attivazione  
    "device": "...",  
}
```

Figura 2.3: Struttura payload di input del Primo Login

```
{  
    "token": "...",  
    "last_update": "...",  
    "filter": "...",  
    "data": {  
        ...  
    }  
}
```

Figura 2.4: Struttura payload di input delle operazioni CRUD

```
{  
    "state":{  
        "code": "0/2"  
        "description":"consistent/no consistent : send data to server",  
    }  
}  
{  
    "state":{  
        "code": "1"  
        "description":"no consistent : receive data from server",  
        "data":{  
            "user":{  
                "name":"...","  
                "last_name":"...","  
                "gender":"...","  
                "birth_date":"...","  
                "email":"...","  
                "city":"...","  
                "phone":"...","  
                "height":"...","  
                "username":"...","  
                "image":{...}  
            },  
            "weights": [  
                {...},  
                ...  
            ],  
            "workouts": [  
                {...},  
                ...  
            ],  
            "settings":{  
                "sport":"...","  
                "target":"...","  
                "unit_measure":{...}  
            }  
        }  
    }  
}
```

Figura 2.5: Struttura payload di output della sincronizzazione dati

## 2.2.2 Connessione al Server

Un punto importante per lo sviluppo dell'applicazione è stato quello di riuscire a connettere il device al server per il mantenimento dei dati in uno stato sincronizzato.

Sono richiesti i permessi *android.permission.INTERNET* e *android.permission.ACCESS\_NETWORK\_STATE*.

Fortunatamente Android mette a disposizione *Volley*, una libreria HTTP che rende la rete più semplice e, soprattutto, più veloce.

*Volley* offre i seguenti vantaggi:

- pianificazione automatica delle richieste di rete,
- più connessioni di rete simultanee,
- supporto per priorità delle richieste,
- API di richiesta di cancellazione : è possibile annullare una singola richiesta oppure è possibile impostare blocchi o ambiti di richieste da annullare,
- facilità di personalizzazione, ad esempio, per riprovare e backoff,
- buon ordinamento che semplifica la corretta compilazione dell'interfaccia utente con i dati recuperati in modo asincrono dalla rete,
- strumenti di debug e tracciabilità.

Si integra facilmente con qualsiasi protocollo ed è pronto all'uso con supporto per stringhe, immagini e JSON non elaborati. Fornendo supporto integrato per le funzionalità di cui hai bisogno, *Volley* ti libera dalla stesura del codice del boilerplate e ti

consente di concentrarti sulla logica specifica della tua app.

Ad alto livello, Volley si utilizza creando un *RequestQueue* e passandogli oggetti *Request*. RequestQueue gestisce i thread di lavoro per l'esecuzione delle operazioni di rete, la lettura e la scrittura nella cache e l'analisi delle risposte. Le richieste eseguono l'analisi delle risposte non elaborate e Volley si occupa di inviare la risposta analizzata al thread principale per la consegna. In questo contesto come oggetto *Request* si è implementata una classe *CustomRequest* (*Figura 2.6*) che estende *StringRequest*.

In effetti esiste anche una classe *JsonRequest*, ma non viene utilizzata perché si vuole gestire gli errori del server tramite Alert Dialog.

```

public class CustomRequest extends StringRequest {

    private static final String BODY_CONTENT_TYPE = "application/json; charset=utf-8";
    private final static int INITIAL_TIMEOUT_MS = 50000;
    private final static int MAX_NUM_RETRIES = 3;

    private JSONObject bodyJson;

    CustomRequest(Context context, int method, String url, JSONObject bodyJson, Response.Listener<JSONObject> responseJsonListener) {
        super(method, url, responseJsonListener);
        Log.d(TAG, "onResponse");
        try {
            final JSONObject JSONResponse = new JSONObject(response);
            Log.d(TAG, JSONResponse.toString());
            if(JSONResponse.has(ERROR)) {
                final JSONObject error = JSONResponse.getJSONObject(ERROR);
                switch (error.getInt(NetworkHelper.Constant.CODE)){
                    case NetworkHelper.HttpResponse.Code.Error.TOKEN_NOT_VALID:
                        throw new TokenException(context);
                }
                throw new SomeErrorHttpException(context, error.getString(NetworkHelper.Constant.DESCRIPTION));
            }
            responseJsonListener.onResponse(JSONResponse);
        } catch (Exception e) {
            Log.e(TAG, "Error = " + e + ", (response) = " + response);

            BackgroundException ex;
            if(e instanceof TokenException)
                ex = (TokenException) e;
            else if(e instanceof SomeErrorHttpException)
                ex = (SomeErrorHttpException)e;
            else{
                ex = SomeErrorHttpException.create(context, e instanceof JSONException? response: e.getMessage());
            }
            ex.alert();
        }
        }, error -> {
            if(error.networkResponse!=null){
                String errorHandler = String.valueOf(error.networkResponse.statusCode);

                if(error instanceof TimeoutError)
                    errorHandler = " " + context.getString(R.string.connection_slow);
                else if(error instanceof ServerError)
                    errorHandler = " " + context.getString(R.string.internal_server_error);

                SomeErrorHttpException ex = SomeErrorHttpException.create(context, errorHandler);
                ex.alert();
            });
        this.bodyJson = bodyJson;
    }

    @Override
    public String getBodyContentType() {
        return BODY_CONTENT_TYPE;
    }

    @Override
    public byte[] getBody() {
        return bodyJson.toString().getBytes();
    }

    @Override
    public Request<> setRetryPolicy(RetryPolicy retryPolicy) {
        return super.setRetryPolicy(new DefaultRetryPolicy( INITIAL_TIMEOUT_MS, MAX_NUM_RETRIES,
                DefaultRetryPolicy.DEFAULT_BACKOFF_MULT));
    }

    @Override
    public Request<> setTag(Object tag) {
        return super.setTag(bodyJson);
    }
}

```

Figura 2.6: *CustomRequest.java*

Le richieste vengono gestite dalla classe utility *NetworkHelper* che delega alla classe innestata *HttpRequest* (*Figura 2.7*) l’invio delle richieste, mentre delega alla classe innestata *HttpResponce* la ricezione delle risposte e l’aggiornamento dell’interfaccia grafica, quando richiesto.

Per quanto riguarda la sincronizzazione dei dati dovuti a delle modifiche in locale, avviene tramite il servizio started *NetworkService* (*Figura 2.8*), il quale aggiorna il database del server in background.

```

public static class HttpRequest{

    private static HttpRequest httpRequest;
    private Context context;
    private HttpRequest(Context context){=}

    public static synchronized HttpRequest getInstance(Context context){=}

    private static final String TAG = NetworkHelper.class.getName();

    private static final String SERVER = "https://runwalktracking.e000webhostapp.com/";
    private static final String ACCOUNT = SERVER + "account/";

    private static final String INSERT = SERVER + "insert.php";
    private static final String UPDATE = SERVER + "update.php";
    private static final String DELETE = SERVER + "delete.php";
    private static final String UPDATE_ALL = SERVER + "updateAll.php";
    private static final String SYNC = SERVER + "sync.php";

    private static final String FORGOT_PASSWORD = SERVER + "request_change_password.php";
    private static final String CONTINUE_HERE = SERVER + "continuehere.php";
    private static final String DOWNLOAD_IMAGE = ACCOUNT + "download_image.php";
    private static final String SIGN_UP = ACCOUNT + "signup.php";
    private static final String FIRST_SIGN_IN = ACCOUNT + "first_signin.php";
    private static final String SIGN_IN = ACCOUNT + "signin.php";
    private static final String CHANGE_PASSWORD = ACCOUNT + "change_password.php";

    private static RequestQueue queue;

    public boolean requestCUD(String action, String filter, JSONObject data, Response.Listener<JSONObject> responseJsonListener)
        throws NullPointerException, JSONException {
        try {
            if(action!=null){
                switch (action){
                    case Constant.INSERT:
                        return insert(filter, data, responseJsonListener);
                    case Constant.UPDATE:
                        return update(filter, data, responseJsonListener);
                    case Constant.DELETE:
                        return delete(filter, data, responseJsonListener);
                    default:
                        return false;
                }
            }
            }catch (InternetNoAvailableException e){
                e.alert();
            }
            return false;
        }

        public static boolean request(AppCompatActivity activity, String action , JSONObject bodyJson) {
        try {
            switch (action){
                case Constant.SIGN_UP:
                    return requestSignUp(activity, bodyJson);
                case Constant.SIGN_IN:
                    return requestSignIn(activity, bodyJson);
                case Constant.FIRST_LOGIN:
                    return requestFirstSignIn(activity, bodyJson);
                case Constant.FORGOT_PASSWORD:
                    return requestForgotPassword(activity, bodyJson);
                case Constant.CHANGE_PASSWORD:
                    return requestChangePassword(activity, bodyJson);
                case Constant.DELETE_ACCOUNT:
                    return requestDeleteAccount(activity);
                case Constant.CONTINUE_HERE:
                    return requestContinueHere(activity);
                default:
                    return false;
            }
            } catch (InternetNoAvailableException e) {
                e.alert();
            } catch (IllegalArgumentException e){
                Log.e(TAG, e.getMessage());
            } catch (JSONException e) {
                e.printStackTrace();
            }
            return false;
        }

        public static boolean syncInForeground(AppCompatActivity activity) {
            return sync(activity, null, HttpResponse.onResponseSyncInForeground(activity));
        }

        public boolean syncInBackground(OnUpdateGuilistener onUpdateGuilistener) {
            return sync(null, context, onUpdateGuilistener);
        }

    } ...
}

```

Figura 2.7: Classe *HttpRequest*

```

public class NetworkService extends Service {
    private final String TAG = NetworkService.class.getName();
    private Context context;
    @Override
    public void onCreate() {
        super.onCreate();
        this.context = getApplicationContext();
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Thread threadCUO= new Thread(()->{
            if(intent!=null){
                try {
                    final String filter = intent.getStringExtra(KeysIntent.FILTER);
                    final JSONObject data = new JSONObject(intent.getStringExtra(KeysIntent.DATA_REQUEST));
                    if(data.has(UserDescriptor.ID_USER)) data.remove(UserDescriptor.ID_USER);
                    if(filter.equals(NetworkHelper.Constant.USER) && !data.isNull(NetworkHelper.Constant.IMAGE)){
                        // COMPRESSIONE IMMAGINE
                        final String name_img = data.getJSONObject(NetworkHelper.Constant.IMAGE).getString(ImageProfileDescriptor.NAME);
                        final String path = ImagefileHelper.create(context).getImage(name_img).getPath();
                        String encode = BitmapUtilities.BitmapToString(Bitmapfactory.decodeFile(path));
                        data.getJSONObject(NetworkHelper.Constant.IMAGE).put(NetworkHelper.Constant.IMG_ENCODE, encode);
                    }
                    Log.d(TAG, data.toString());
                    if(intent.getAction()!=null){
                        NetworkHelper.HttpRequest.getInstance(context).requestCUO(intent.getAction(), filter , data , response -> {
                            Log.d(TAG, "Response after "+ intent.getAction() +" : "+ response.toString());
                            NetworkService.this.stopSelf();
                        });
                    }
                } catch (JSONException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        });
        threadCUO.start();
        return super.onStartCommand(intent, flags, startId);
    }
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}

```

Figura 2.8: *NetworkService.java*

Se il servizio non viene concluso per cause esterne (connessione debole o assente), la sincronizzazione si può rimandare a quando si ha una “buona connessione”, premendo per la seconda volta uno dei pulsanti della *Bottom Navigation Bar* (*Figura 2.9*).

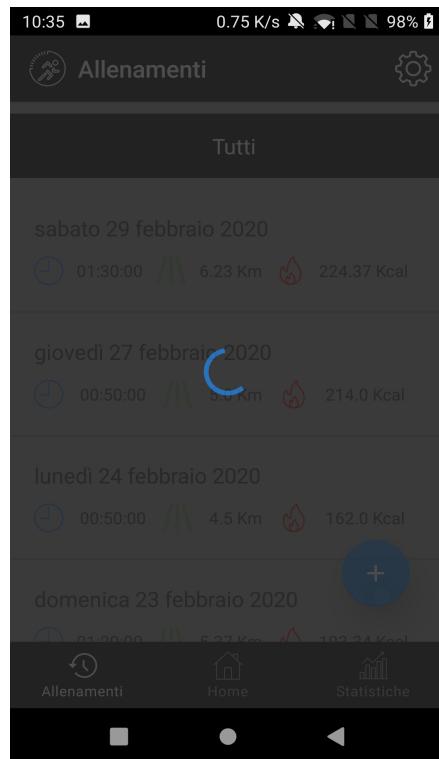


Figura 2.9: *Sincronizzazione dati attraverso Bottom Navigation Bar*

Un controllo avviene anche all'avvio dell'applicazione (*Figura 2.10*).

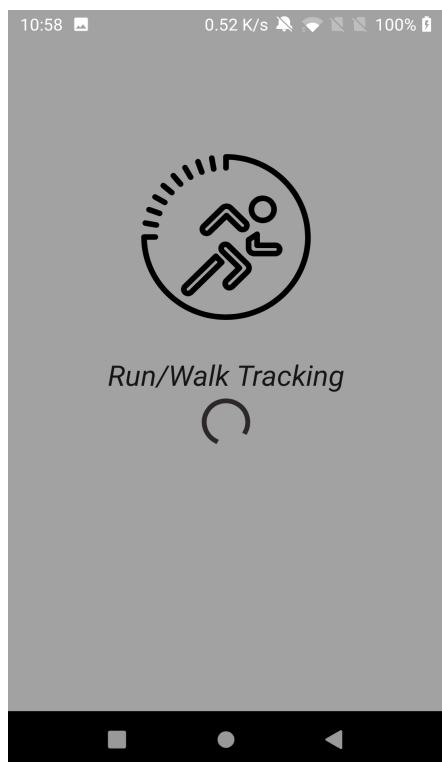


Figura 2.10: *Splash Screen*

## 2.3 Persistenza dei dati

Nei sistemi informatici, le informazioni vengono rappresentate per mezzo di *dati*, che hanno bisogno di essere interpretati per fornire informazioni.

L'approccio “convenzionale” alla gestione dei dati sfrutta la presenza di archivi o file per memorizzare i dati in modo persistente sulla memoria di massa. Un file consente di memorizzare e ricercare i dati, ma fornisce solo semplici meccanismi di accesso e di condivisione. Secondo questo approccio, le procedure scritte in un linguaggio di programmazione sono completamente autonome; ciascuna di esse definisce e utilizza uno o più file “privati”. Eventuali dati di interesse per più programmi sono replicati tante volte quanti sono i programmi che li utilizzano, con evidente ridondanza e possibilità di incoerenza.

Le basi di dati sono state concepite in buona misura per superare questo tipo di inconvenienti, gestendo in modo integrato e flessibile le informazioni di interesse per diversi soggetti, limitando i rischi di ridondanza e incoerenza.

In poche parole si può definire un *sistema di gestione di basi di dati* un sistema software in grado di gestire collezioni di dati che siano *grandi, condivise e persistenti*, assicurando la loro *affidabilità e privatezza*.

### 2.3.1 Client

Android offre un leggerissimo (ma potente) database open-source, SQLite. SQLite è fondamentalmente una libreria che implementa un motore di database transazionale autonomo. A differenza di altri database, SQLite non si basa su un processo separato ma accede direttamente ai propri file contenenti i dati.

Le caratteristiche peculiari di SQLite sono le seguenti:

- non richiede un server o sistema separato per funzionare,
- non necessita di configurazioni,
- viene gestito da un singolo file,
- non dipende da applicazioni esterne,
- è database transazionale che rispetta le proprietà ACID (Atomicità, Coerenza, Isolamento e Durabilità).

Insomma, leggere e scrivere un database, in Android, è un'operazione nativa e incorporata, come lo è accedere ad un file o connettersi alla rete. L'unico prerequisito richiesto è la conoscenza, anche basilare, di SQL, il linguaggio principe per l'interazione con i DBMS .

Per gestire la creazione del database e la gestione delle versioni, Android mette a disposizione una classe astratta di supporto, *SQLiteOpenHelper*. Questa si occupa di aprire il database se esiste, crearlo in caso contrario e aggiornarlo se necessario .

*SQLiteOpenHelper*, essendo una classe astratta, deve essere implementata attraverso una sottoclassificazione che estenda quest'ultima (*Figura 2.11*). La sottoclassificazione deve sovrascrivere almeno i metodi *onCreate(SQLiteDatabase)*, *onUpgrade(SQLiteDatabase, int, int)* e in questo caso anche *onOpen(SQLiteDatabase)*.

```

public class DaoFactory extends SQLiteOpenHelper implements Dao {

    private static final String TAG = DaoFactory.class.getName();
    private static DaoFactory DaoFactoryInstance = null;
    private static final String DB_NAME = "RUN_WALK_TRACKING_DB.db";
    private static final int DB_VERSION = 4;
    private Context context;

    public static synchronized DaoFactory getInstance(Context context){
        if(DaoFactoryInstance == null)
            DaoFactoryInstance = new DaoFactory(context.getApplicationContext());
        return DaoFactoryInstance;
    }

    private DaoFactory(Context context) {
        super(context);
        Log.d(TAG, "onOpen");
        db.execSQL("PRAGMA FOREIGN_KEYS=ON");
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        Log.d(TAG, "onCreate");
        // CREAZIONE TABELLE
        sqLiteDatabase.execSQL(UserDescriptor.CREATE_TABLE_USER);
        sqLiteDatabase.execSQL(ImageProfileDescriptor.CREATE_TABLE_PROFILE_IMAGE);
        sqLiteDatabase.execSQL(SettingsDescriptor.SportDefault.CREATE_TABLE_SPORT_DEFAULT);
        sqLiteDatabase.execSQL(SettingsDescriptor.TargetDefault.CREATE_TABLE_TARGET_DEFAULT);
        sqLiteDatabase.execSQL(SettingsDescriptor.UnitMeasureDefault.CREATE_TABLE_UNIT_MEASURE_DEFAULT);
        sqLiteDatabase.execSQL(WeightDescriptor.CREATE_TABLE_WEIGHT);
        sqLiteDatabase.execSQL(WorkoutDescriptor.CREATE_TABLE_WORKOUT);
        sqLiteDatabase.execSQL(PlaylistDescriptor.CREATE_TABLE_PLAYLIST);
        sqLiteDatabase.execSQL(SongDescriptor.CREATE_TABLE_SONG);
        sqLiteDatabase.execSQL(CompoundDescriptor.CREATE_TABLE_COMPOUND);

        // CREAZIONE TRIGGER
        sqLiteDatabase.execSQL(PlaylistDescriptor.CREATE_TRIGGER_BEFORE_INSERT);
        sqLiteDatabase.execSQL(PlaylistDescriptor.CREATE_TRIGGER_BEFORE_UPDATE);
        sqLiteDatabase.execSQL(CompoundDescriptor.CREATE_TRIGGER_BEFORE_DELETE);

        // CREAZIONE INDICI
        sqLiteDatabase.execSQL(UserDescriptor.CREATE_INDEX_USER);
        sqLiteDatabase.execSQL(UserDescriptor.CREATE_INDEX_NAME);
        sqLiteDatabase.execSQL(ImageProfileDescriptor.CREATE_INDEX_IMAGE);
        sqLiteDatabase.execSQL(SettingsDescriptor.SportDefault.CREATE_INDEX);
        sqLiteDatabase.execSQL(SettingsDescriptor.TargetDefault.CREATE_INDEX);
        sqLiteDatabase.execSQL(SettingsDescriptor.UnitMeasureDefault.CREATE_INDEX);
        sqLiteDatabase.execSQL(WeightDescriptor.CREATE_INDEX_WEIGHT);
        sqLiteDatabase.execSQL(WeightDescriptor.CREATE_INDEX_DATE);
        sqLiteDatabase.execSQL(WorkoutDescriptor.CREATE_INDEX);
        sqLiteDatabase.execSQL(PlaylistDescriptor.CREATE_INDEX_PLAYLIST);
        sqLiteDatabase.execSQL(PlaylistDescriptor.CREATE_INDEX_USER);
        sqLiteDatabase.execSQL(SongDescriptor.CREATE_INDEX_SONG);
        sqLiteDatabase.execSQL(CompoundDescriptor.CREATE_INDEX_COMPOUND);
        sqLiteDatabase.execSQL(CompoundDescriptor.CREATE_INDEX_PLAYLIST);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.d(TAG, "onUpgrade");
        if(newVersion > oldVersion){
            onCreate(db);
        }
    }
    ...
}

```

Figura 2.11: *DaoFactory.java*

Come già visto nel capitolo della progettazione, per accedere ai dati si utilizza il pattern DAO.

La creazione degli oggetti Dao è delegata alla classe *DaoFactory* che implementa l’interfaccia *Dao* (*Figura 2.12*).

```
public interface Dao {  
  
    WorkoutDao getWorkoutDao();  
  
    UserDao getUserDao();  
  
    UserDao.ImageDao getImageDao();  
  
    StatisticsDao getStatisticsDao();  
  
    StatisticsDao.WeightDao getWeightDao();  
  
    SettingsDao getSettingDao();  
  
    PlayListDao getPlayListDao();  
  
    SongDao getSongDao();  
  
    CompoundDao getCompoundDao();  
}
```

Figura 2.12: *Dao.java*

### 2.3.2 Server

Per quanto riguarda il lato server, il DBMS utilizzato è *MySQL*. Si è scelto questo DBMS perchè è presente un strumento software gratuito, *phpMyAdmin*, destinato a gestire l'amministrazione di MySQL sul Web. *phpMyAdmin* supporta una vasta gamma di operazioni su MySQL. Le operazioni utilizzate di frequente (gestione di database, tabelle, colonne, relazioni, indici, utenti, autorizzazioni, ecc.) possono essere eseguite tramite l'interfaccia utente, mentre si ha ancora la possibilità di eseguire direttamente qualsiasi istruzione SQL.

PHP 5 e successivi mettono a disposizione la classe *mysqli*, la quale rappresenta una connessione tra PHP e un database MySQL.

La connessione è eseguita utilizzando la funzione *connect()* e le varie transazioni utilizzando le funzioni *transation(\$callback)* o *selection(\$callback)*, a seconda che sia una transazione di scrittura o lettura. Queste funzioni sono implementate dalla classe *DaoFactory* (*Figura 2.13*).

```

class DaoFactory {

    private $host = "localhost";
    private $database = "id11865186_runwalktracking";
    private $user_db = "id11865186_sec_user_tracking";
    private $password_db = "owUpTYWuQEagbc9J";

    private $connection;

    private static $instance;
    public static function instance(){
        if(self::$instance==null) self::$instance = new self();
        return self::$instance;
    }

    public function connect() {
        error_reporting(E_ERROR | E_PARSE);
        $this->connection = new mysqli($this->host, $this->user_db,
                                         $this->password_db, $this->database);
        return !$self::getErrorConnection();
    }

    public function getConnection(){
        return $this->connection;
    }

    public function getErrorConnection(){
        return $this->connection->error;
    }

    public function isDuplicate(){
        return $this->connection->errno==1062;
    }

    public function closeConnection(){
        $this->connection->close();
    }

    public function transaction($callback){
        try {
            if(self::connect())
            {
                $this->connection->autocommit(false);

                if (!is_callable($callback))
                    throw new Exception(PARAMETER_NOT_VALID);

                $callback();

                $this->connection->commit();
            }
        }catch (Exception $e) {
            $this->connection->rollback();
            self::closeConnection();
            throw $e;
        }
        self::closeConnection();
        return true;
    }

    public function selection($callback){
        if(self::connect()){

            if (!is_callable($callback))
                throw new Exception(PARAMETER_NOT_VALID);

            $callback();
        }
        self::closeConnection();
        return true;
    }
}

```

Figura 2.13: *DaoFactory.php*

## 2.4 Autenticazione

L'autenticazione dell'utente avviene ogni volta che si esegue l'accesso al sistema dalla schermata *Accedi* (*Figura 2.14*).

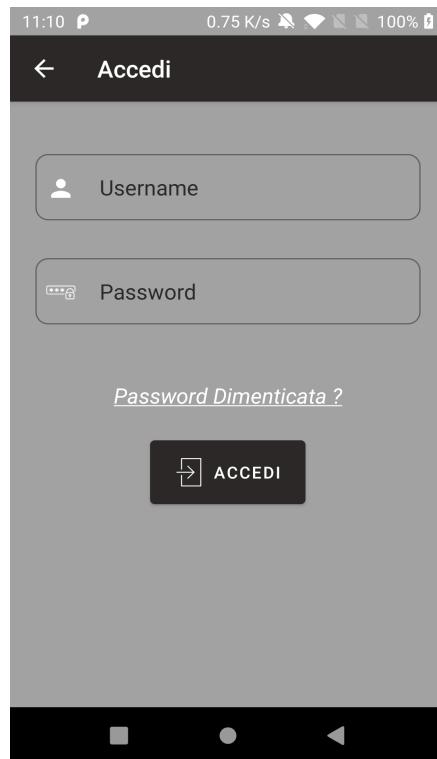


Figura 2.14: *Schermata di Login*

Ciò comporta la creazione di un nuovo token di accesso (sessione codificata) da parte del Server usando la funzione *getEncodedSession(\$session)* riportata nella *Figura 2.15*, dopo opportune verifiche.

```

define("SEPARATOR", ".");

function encode($val){
    return base64_encode(base64_encode($val));
}

function getEncodedSession($session){
    return encode(implode(SEPARATOR,
        array(encode($session[ID_USER]), encode($session[TOKEN]),
            encode($session[LAST_UPDATE]), encode($session[DEVICE]))));
}

```

Figura 2.15: *Funzione di codifica della sessione del Server*

Dopo che l’utente si è autenticato, il client decodifica la sessione con la funzione *getDecodedSession(String encodeSession)* (*Figura 2.16*) e la memorizza nelle SharedPreferences (session.xml).

```

private static final Base64.Decoder decoder = Base64.getDecoder();

private static String decode(Object i){
    return new String(decoder.decode(decoder.decode(String.valueOf(i).getBytes())));
}

public static JSONObject getDecodedSession(String encodeSession) {
    String session_string = decode(encodeSession);
    String[] tmp_session_value = session_string.split("\\.");
    String[] tmp_session_key = Stream.of(NetworkHelper.Constant.ID_USER, NetworkHelper.Constant.TOKEN,
        NetworkHelper.Constant.LAST_UPDATE, NetworkHelper.Constant.DEVICE).toArray(String[]::new);
    Map<String, Object> session =
        IntStream.range(0, tmp_session_key.length).boxed().collect(
            Collectors.toMap(i -> tmp_session_key[i], i -> tmp_session_value[i]));

    session.entrySet().forEach(e -> e.setValue(NumberUtilities.cast(decode(e.getValue()))));
    return new JSONObject(session);
}

```

Figura 2.16: *Funzione di decodifica della sessione del Client*

Successivamente ad ogni richiesta di “aggiornamento”, viene inviato al server anche l’id di sessione (nelle funzioni chiamato TOKEN) creato dal server stesso con la funzione *getToken(100)* (*Figura 2.17*).

```

function getToken($length){
    $token = "";
    $codeAlphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    $codeAlphabet .= "abcdefghijklmnopqrstuvwxyz";
    $codeAlphabet .= "0123456789";
    $max = strlen($codeAlphabet);

    for ($i=0; $i < $length; $i++) {
        $token .= $codeAlphabet[random_int(0, $max-1)];
    }
    return $token;
}

```

Figura 2.17: Funzione di creazione dell' id sessione (token)

## Funzionalità “Continua Qui”

Se l'utente accede all'applicazione da un secondo smartphone senza avere effettuato il logout nel primo, in questo appare il seguente Alert Dialog :

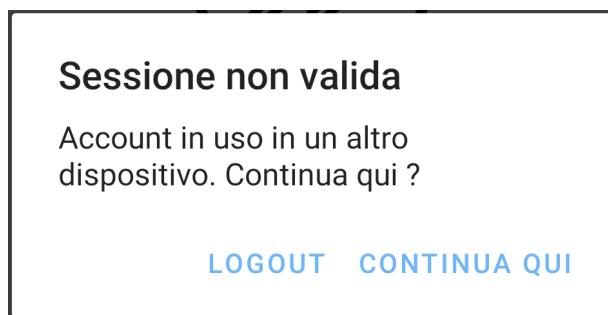


Figura 2.18: Dialog Token non valido

Se l'utente clicca “Continua Qui”, il sistema fa una chiamata al server con il token di accesso “vecchio” solo per ricavare l'id dell' utente e del device, in modo che non siano in chiaro nel payload del pacchetto in uscita.

Quindi sia il Server che il Client hanno bisogno rispettivamente delle funzioni di decodifica (*Figura 2.19*) e di codifica (*Figura 2.20*) della sessione.

```
define("SEPARATOR", ".");  
  
function decode($val){  
    return base64_decode(base64_decode($val));  
}  
  
function getDecodedSession($token_session){  
    return array_combine(array(ID_USER, TOKEN, LAST_UPDATE, DEVICE),  
                        array_map(function($value){  
                            return cast(decode($value));  
                        }, explode(SEPARATOR, decode($token_session))));  
}
```

Figura 2.19: *Funzione di decodifica della sessione del Server*

```
private static final Base64.Encoder encoder = Base64.getEncoder();  
  
private static String encode(Object i){  
    return encoder.encodeToString(encoder.encode(String.valueOf(i).getBytes()));  
}  
  
public static String getEncodedSession(int id_user_value, String token_value,  
                                       long last_update_value, String device_value ){  
  
    String id_user = encode(id_user_value);  
    String token = encode(token_value);  
    String last_update = encode(last_update_value);  
    String device = encode(device_value);  
  
    List<String> session = Arrays.asList(id_user, token, last_update, device);  
    String session_string = String.join(".", session);  
    return encode(session_string);  
}
```

Figura 2.20: *Funzione di codifica della sessione del Client*

## 2.5 Password Dimenticata

Una funzionalità importante è quello di poter reimpostare la password dell'account se dimenticata.

Dalla schermata di login, si può accedere alla schermata “Ripristina Password” nella quale si richiede inserimento dell'email, nella quale si vuole ricevere il link di reset. Se l'invio va a buon fine, apparirà un messaggio che invita a controllare la posta elettronica (*Figura 2.21*).

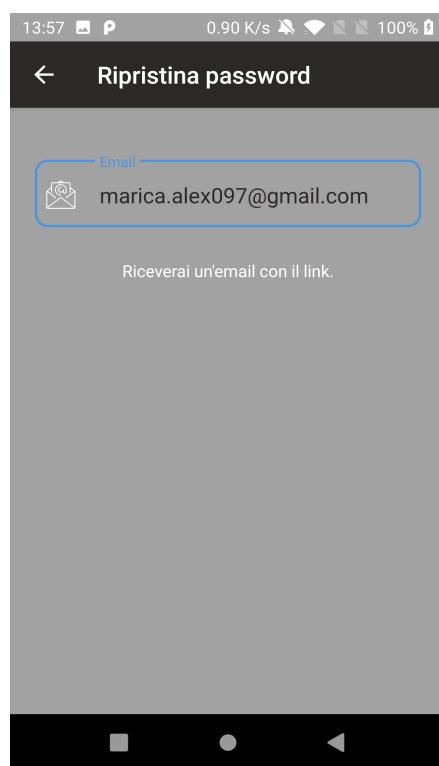


Figura 2.21: *Schermata di Password Dimenticata*

Successivamente è illustrata l'email e la pagina web che permette di reimpostare la password.

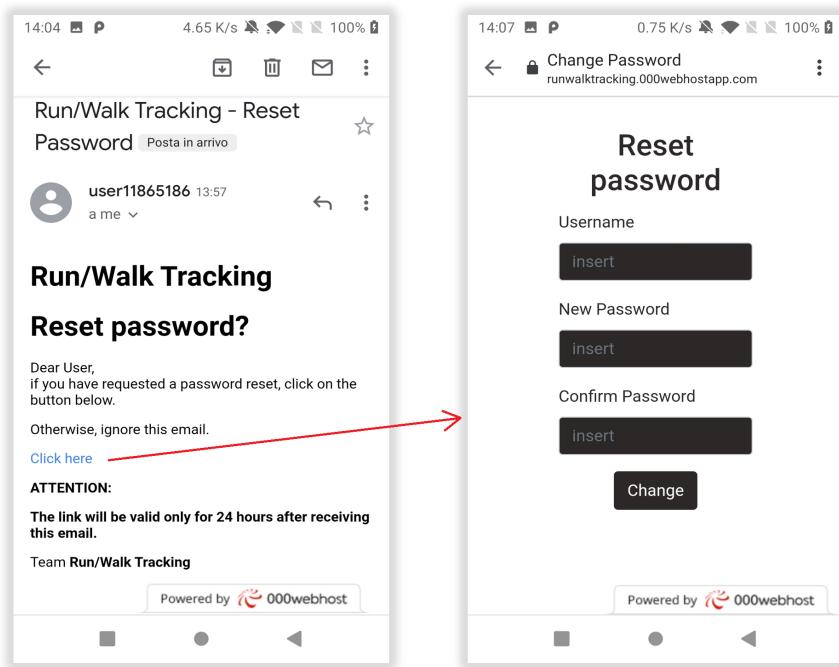


Figura 2.22: *Email e Link valido*

Se il link presente nell'email è scaduto, viene visualizzato il seguente messaggio di errore:

```
"Error": {  
    "code": 14,  
    "description": "Link for the reset you clicked is EXPIRED !! Please request a new one."  
}
```

Figura 2.23: *Messaggio di errore del link scaduto*

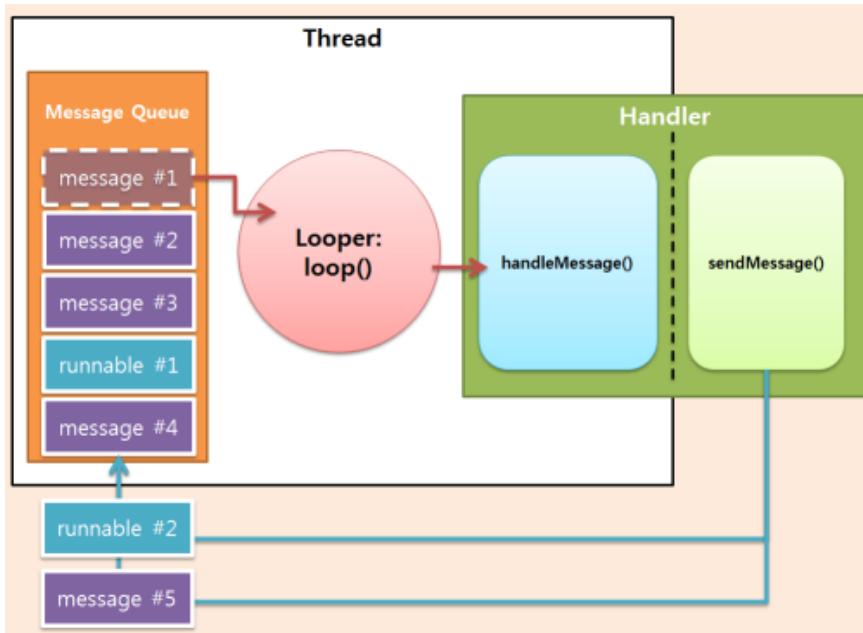
## 2.6 Allenamento in Real-Time

La funzionalità principale è quella di poter memorizzare i dati di un allenamento in real-time outdoor o indoor.

Questa funzionalità è delegata al servizio *WorkoutService* (servizio sia started che bound), che utilizza un thread associato ad un Looper, il quale a sua volta è associato a un Handler.

I thread normali non possiedono un looper ma si può associare attraverso le funzioni *Looper.prepare()* e *Looper.loop()*.

Il Looper è una classe che viene utilizzata per eseguire i messaggi in una coda, infatti è responsabile della sua creazione nel thread.



Per inviare ed elaborare oggetti Message e Runnable associati a MessageQueue di un thread esiste la classe Handler. Ogni istanza del Handler è associata a un singolo thread e alla coda dei messaggi di quel thread. Quando si crea un nuovo Handler, viene associato a un Looper. Esso consegnerà gli oggetti Message

e Runnable alla MessageQueue di quel Looper ed eseguirli sul thread di quel Looper.

In questo caso si è creato un Handler (*Figura 2.24*) che elabora messaggi della componente che gestisce la localizzazione e della componente che gestisce il movimento.

```
private Handler.Callback handlerCallback = new Handler.Callback() {
    @Override
    public boolean handleMessage(Message msg) {

        if(msg!=null && msg.getData()!=null){
            Bundle bundle = msg.getData();
            Log.d(TAG, bundle.toString());

            String distance = bundle.getString(KeysIntent.DISTANCE);
            String energy = bundle.getString(KeysIntent.ENERGY);
            if(distance!=null && energy!=null){
                notificationWorkout.updateDistanceAndEnergy(distance, energy);
                Log.d(TAG, "Update Notification");
                if(binder!=null) {
                    Log.d(TAG, "Send to activity");
                    sendBroadcast(new Intent(ActionReceiver.DISTANCE_ENERGY_ACTION)
                        .putExtra(KeysIntent.DISTANCE, distance)
                        .putExtra(KeysIntent.ENERGY, energy));
                }
            }
            ArrayList<Location> locations = bundle.getParcelableArrayList(KeysIntent.ROUTE);
            if(locations!=null){
                Preferences.WorkoutInExecution.MapLocation.addAll(context, locations);
                Log.d(TAG, "Update Preferences MAP ROUTE");
                if(binder!=null){
                    Log.d(TAG, "Send to activity");
                    sendBroadcast(new Intent(ActionReceiver.DRAWING_MAP_ACTION));
                }
            }
        }
        return false;
    };
};
```

Figura 2.24: *Handler.Callback*

Per avviare il servizio si è utilizzata la funzione *startForegroundService(Intent service)*, che prevede il permesso *android.permission.FOREGROUND\_SERVICE*.

Una funzionalità, che si è voluta aggiungere, è quella di ritardare l'inizio dell'allenamento di 15 secondi tramite la classe *DelayedStartWorkoutDialog* (*Figura 2.25*).

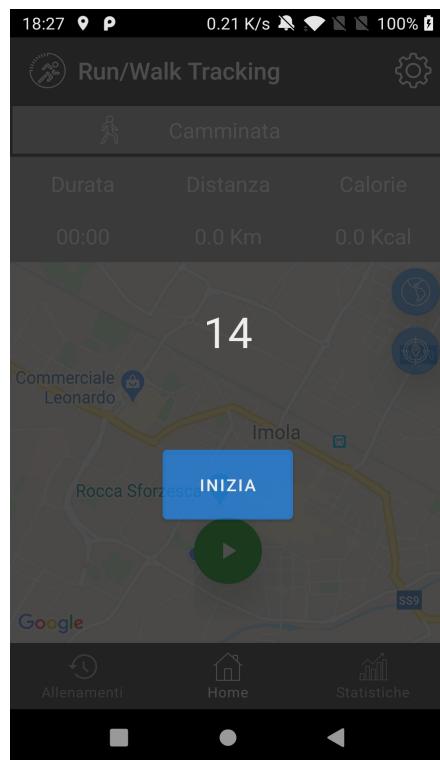


Figura 2.25: Schermata di inizio ritardato dell'allenamento

## Sensore di movimento : STEP\_COUNTER

Il sensore STEP\_COUNTER viene utilizzato per ottenere il numero totale di passi eseguiti dall'utente dall'ultimo riavvio del telefono. Al riavvio del telefono, il valore del sensore contapassi viene ripristinato a zero.

In questo progetto si è fatto in modo che ogni volta che l'allenamento in Real Time viene avviato, il valore del sensore parta da zero. Nel metodo `onSensorChanged()`, il numero di passi è dato da `event.value[0]`; sebbene sia un valore float, la parte frazionale è sempre zero.

Questo sensore funziona in batch e in modalità continua.

Se si specifica 0 o nessuna latenza nel metodo `SensorManager.registerListener()`, il sensore funziona in modalità continua; se invece si specifica una latenza, raggruppa gli eventi in batch e li riporta alla latenza specificata. Per un uso prolungato di questo sensore, si consiglia di utilizzare la modalità batch, poiché consente di risparmiare energia.

Per poterlo utilizzare è richiesto il permesso `android.hardware.sensor.STEPCOUNTER`.

La *Figura 2.26* rappresenta il *SensorEventListener* il quale è utilizzato per ricevere notifiche da parte del *SensorManager* quando il sensore rileva un cambiamento del valore. Il Listener deve essere associato al sensore STEP\_COUNTER affinché venga chiamata la funzione `onSensorChanged()`. Quest'associazione si effettua utilizzando la seguente funzione `registerListener(sensorEventListener, stepCounterSensor, SensorManager.SENSOR_DELAY_UI, LATENCY, handler)`.

Nella funzione `onSensorChanged()` viene convertito il "passo" in chilometro e calcolato il consumo calorico in base allo sport che si sta praticando (`sport.getConsumedEnergy(weight, km)`).

Se l'unità di misura di default non è il chilometro, il valore memorizzato sarà sempre in chilometri ma nell'interfaccia grafica viene convertito (approfondito nella Sezione *Unità di misura*).

```
private SensorEventListener sensorEventListener = new SensorEventListener() {

    @Override
    public void onSensorChanged(SensorEvent event) {
        Log.d(TAG, "onSensorChanged");

        if (isRunning && !isInPause) {

            if (firstTime) {
                milestoneStep = (int) event.values[0];
                Log.d(TAG, "Reset milestoneStep : " + milestoneStep);
                firstTime = false;
            }

            int steps = (int) event.values[0] - milestoneStep;
            // distance and energy
            double km = steps * STEPS_TO_KM;
            double kcal = sport.getConsumedEnergy(weight, km);
            workout.getDistance().setvalue(true, km);
            workout.getCalories().setvalue(true, kcal);

            String distance = workout.getDistance().toString(true);
            String energy = workout.getCalories().toString(true);

            if(powerManager.isInteractive()) {
                Message msg = Message.obtain();
                Bundle bundle = new Bundle();
                bundle.putString(KeysIntent.DISTANCE, distance);
                bundle.putString(KeysIntent.ENERGY, energy);
                msg.setData(bundle);
                handler.sendMessage(msg);
            }
            else{
                Log.d(TAG, "Steps = "+ steps +
                      ", Distance = " + distance +
                      " , Calories = " + energy);
            }
        }
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        Log.d(TAG, "onAccuracyChanged");
    }
};
```

Figura 2.26: SensorEventListener di *WorkoutService*

L'aggiornamento dell'interfaccia grafica è gestita dal Handler, che invia al Looper le informazioni ( distanza e calorie ) utilizzando l'oggetto Message. Nel caso il device sia nello stato SCREEN OFF, l'invio non viene effettuato, altrimenti viene aggiornata la notifica del servizio in foreground e anche l'activity se l'applicazione è in foreground (*Figura 2.27*).

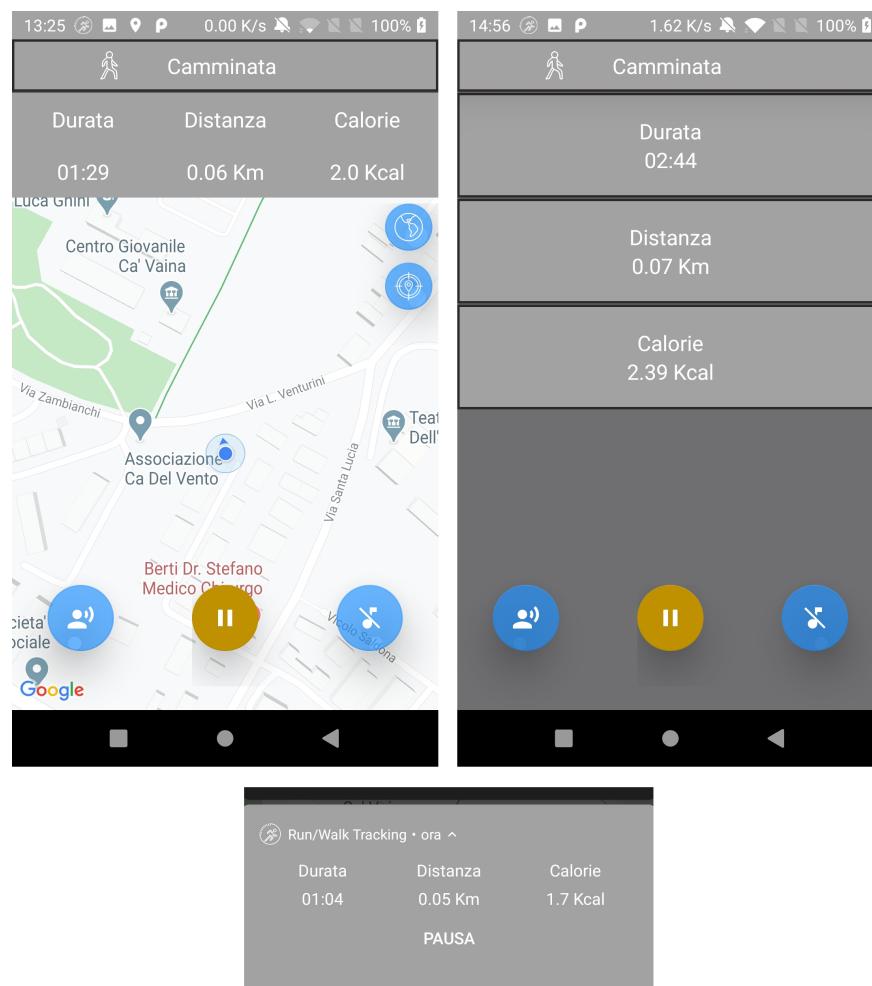


Figura 2.27: Aggiornamento distanza e calorie

## Localizzazione

Questa funzionalità è attiva solo se il GPS del device è attivo, come si osserva nell'interfaccia grafica (*Figura 2.28*) dove è presente una mappa per tracciare il percorso effettuato.

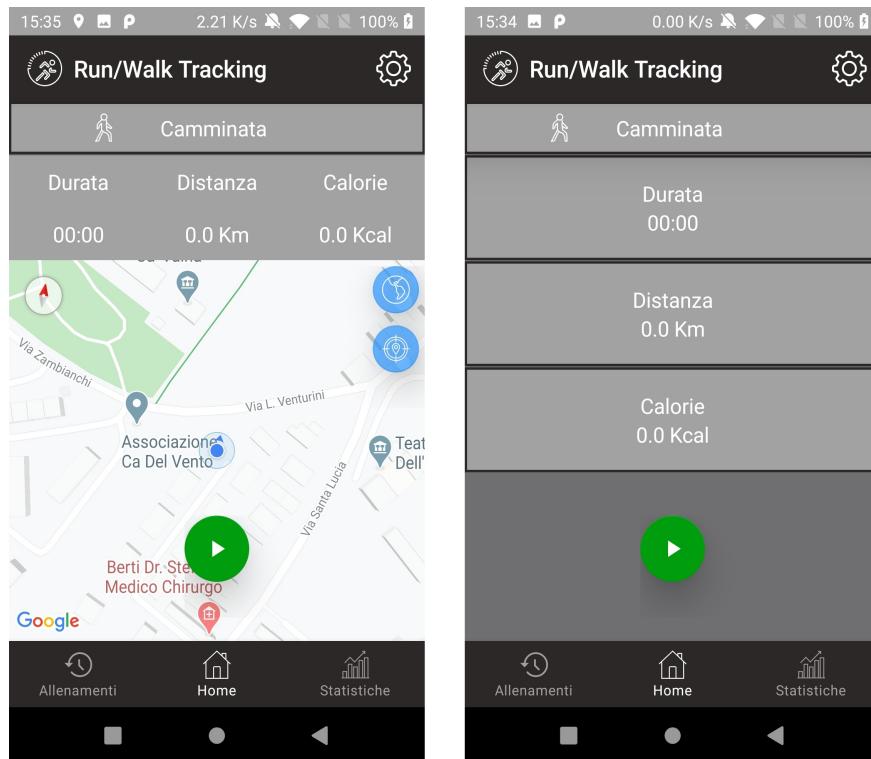


Figura 2.28: *Interfaccia grafica della Home*

Questa funzionalità richiede i seguenti permessi  
*android.permission.ACCESS\_COARSE\_LOCATION* e  
*android.permission.ACCESS\_FINE\_LOCATION*.

Si è scelto di utilizzare il servizio più famoso e facilmente integrabile con i dispositivi Android: Google Maps. Per poter iniziare ad utilizzare le mappe di Google durante la programmazione Android, è stato necessario, dopo aver ottenuto la google maps key, creare un opportuno fragment. All'interno del fragment Java è implementata l'interfaccia *OnMapReadyCallback* ed

è istanziato un oggetto di tipo GoogleMap. Quest'ultimo sarà l'oggetto principale che viene utilizzato per il tracciamento del percorso, richiamando i suoi metodi e attributi. La callback principale fornita da questa interfaccia è *onMapReady*. Essa permette di eseguire codice non appena la mappa è pronta all'utilizzo.

Il percorso effettuato durante l'allenamento non viene mantenuto in memoria, ma si è deciso di memorizzarlo temporaneamente nelle SharedPreferences (WorkoutInExecution.xml), utilizzando una funzione di utility della libreria di Google Maps: *PolyUtil.encode(List<LatLng> locations)*, che codifica una lista di coordinate in una stringa di caratteri.

Per visualizzare il percorso nella mappa, si utilizza la classe *Polyline* che grazie al Builder, *PolylineOptions*, aggiunge la lista di coordinate ottenuta dalla stringa memorizzata precedentemente, utilizzando la funzione utility *PolyUtil.decode(stringLocations)*.

Come chiarito precedentemente Android ha posto dei limiti sulla localizzazione in background. Quindi si deve gestire le richieste in background in modo diverso da quelle in foreground (*Figura 2.29*).

In background, si utilizza la posizione passiva chiamando il metodo *setFastestInterval()* passando un valore inferiore a quello che si è passato a *setInterval()*. Utilizzando come priorità l'opzione **PRIORITY\_NO\_POWER**, la posizione passiva può fornire opportunamente la posizione calcolata da altre app senza costi aggiuntivi.

Per il foreground non ci sono problemi, quindi si può usare **PRIORITY\_HIGH\_ACCURACY** e un intervallo di pochi secondi.

```
private static final long UPDATE_INTERVAL_IN_MILLISECONDS = 10000;
private static final long FASTEST_UPDATE_INTERVAL_IN_MILLISECONDS = UPDATE_INTERVAL_IN_MILLISECONDS /2;

private static final long UPDATE_INTERVAL_BACK_IN_MILLISECONDS = 3600000;
private static final long FASTEST_UPDATE_INTERVAL_BACK_IN_MILLISECONDS = 6000;

public static boolean isGpsEnable(Context context) {
    return ((LocationManager) context.getSystemService(Service.LOCATION_SERVICE))
        .isProviderEnabled(LocationManager.GPS_PROVIDER);
}

public static LocationRequest createLocationRequestForeground() {
    return LocationRequest.create()
        .setInterval(UPDATE_INTERVAL_IN_MILLISECONDS)
        .setFastestInterval(FASTEST_UPDATE_INTERVAL_IN_MILLISECONDS)
        .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
}

public static LocationRequest createLocationRequestBackground() {
    return LocationRequest.create()
        .setInterval(UPDATE_INTERVAL_BACK_IN_MILLISECONDS)
        .setMaxWaitTime(2*UPDATE_INTERVAL_BACK_IN_MILLISECONDS)
        .setFastestInterval(FASTEST_UPDATE_INTERVAL_BACK_IN_MILLISECONDS)
        .setPriority(LocationRequest.PRIORITY_NO_POWER);
}
```

Figura 2.29: *LocationRequest*

Per poter richiedere gli aggiornamenti sulla posizione si deve richiamare la funzione *requestLocationUpdates()*, della quale ne esistono diverse varianti (*Figura 2.30*).

```

public void startDrawing(Handler handler) {
    try {
        LocationRequest locationRequest ;
        this.handler = handler;
        if(isBackground){
            registerBroadcastReceiver();
            locationRequest = LocationUtilities.createLocationRequestBackground();
            pendingLocation = PendingIntent.getBroadcast(context, REQUEST_LOCATION_BACKGROUND,
                new Intent(ActionReceiver.UPDATE_CHANGE_LOCATION)
                .setPackage(context.getPackageName()),
                PendingIntent.FLAG_UPDATE_CURRENT);
            fusedLocationClient.requestLocationUpdates(locationRequest ,pendingLocation);
            Log.d(TAG, "START LOCATION BACKGROUND!");
        }else{
            locationRequest = LocationUtilities.createLocationRequestForeground();
            fusedLocationClient.requestLocationUpdates(
                locationRequest, locationCallback, handler.getLooper());
            Log.d(TAG, "START LOCATION FOREGROUND!");
        }
    } catch (SecurityException e) {
        Log.e(TAG, "NON HAI I PERMESSI X LA LOCALIZZAZIONE");
        e.printStackTrace();
    }
}

```

Figura 2.30: Funzione di inizio localizzazione

Per le richieste in foreground si è utilizzata la funzione :

*requestLocationUpdates(LocationRequest, LocationCallback, Looper)* , che richiede aggiornamenti sulla posizione con un callback sul Thread Looper specificato. Eventuali precedenti LocationRequests registrati verranno sostituiti. Questa chiamata manterrà attiva la connessione dei servizi di Google Play, quindi quando non serve più ( ovvero quando l’allenamento termina ) si chiama la funzione *removeLocationUpdates(LocationCallback)*.

Per le richieste in background si è utilizzata la funzione :

*requestLocationUpdates(LocationRequest, PendingIntent)* , che richiede aggiornamenti di posizione, anche quando l’app è stata interrotta dal sistema. Il LocationResult, che nelle richieste foreground è gestito dalla LocationCallback, viene inviato a un BroadcastReceiver associato all’ Handler specificato nella sua registrazione (*Figura 2.31*).

```

private void registerBroadcastReceiver() {
    IntentFilter intentFilter = new IntentFilter(ActionReceiver.UPDATE_CHANGE_LOCATION);
    if(!isRegister()){
        context.registerReceiver(broadcastReceiver, intentFilter, null, this.handler);
        isRegister =true;
    }
}

```

Figura 2.31: *Registrazione del BroadcastReceiver per la localizzazione in background*

Sia in foreground che in background quando viene ricevuto l’aggiornamento della posizione, Handler invia al proprio Looper il risultato ottenuto (*Figura 2.32*).

```

private void sendMessage(List<Location> locations){
    Message message = Message.obtain();
    Bundle bundle = new Bundle();
    bundle.putParcelableArrayList(KeysIntent.ROUTE, new ArrayList<>(locations));
    message.setData(bundle);
    handler.sendMessage(message);
}

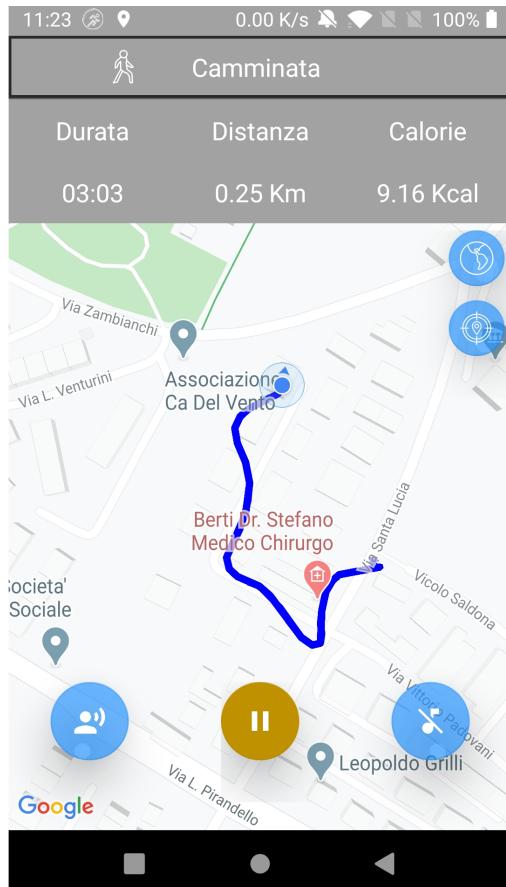
/* BACKGROUND */
public class MapRouteBackgroundReceiver extends BroadcastReceiver{
    @Override
    public void onReceive(Context context, Intent intent) {
        if(intent!=null && intent.getAction()!=null) {
            switch (intent.getAction()) {
                case ActionReceiver.UPDATE_CHANGE_LOCATION:
                    if(LocationResult.hasResult(intent)){
                        LocationResult locationResult = LocationResult.extractResult(intent);
                        if(locationResult!=null){
                            sendMessage(locationResult.getLocations());
                        }
                    }
                    break;
            }
        }
    }
}

/* FOREGROUND */
private LocationCallback locationCallback = new LocationCallback() {
    @Override
    public void onLocationResult(LocationResult locationResult) {
        if (locationResult != null) {
            sendMessage(locationResult.getLocations());
        }
    }
};

```

Figura 2.32: Callback (Foreground) e PendingIntent/Receiver (Background) della localizzazione

Dopo aver ricevuto il messaggio, Handler gestisce l'aggiornamento del percorso nelle SharedPreferences e anche nell'activity se in foreground (*Figura 2.33*).



```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="map_route">caumGuamffACDFKBZB^DHJHRDL@HAH@@JGNQRM
        JKLINCNENGJKBKEQMOGMQAUDSJQBKGKOEOCKIEEC</string>
</map>
```

Figura 2.33: Aggiornamento percorso effettuato

## 2.6.1 Musica

Una funzionalità dell'allenamento in Real-Time è di poter riprodurre una playlist di canzoni, scelte precedentemente nelle impostazioni (*Figura 2.34*). Richiede il permesso *android.permission.READ\_EXTERNAL\_STORAGE* per visualizzare le canzoni nel device.

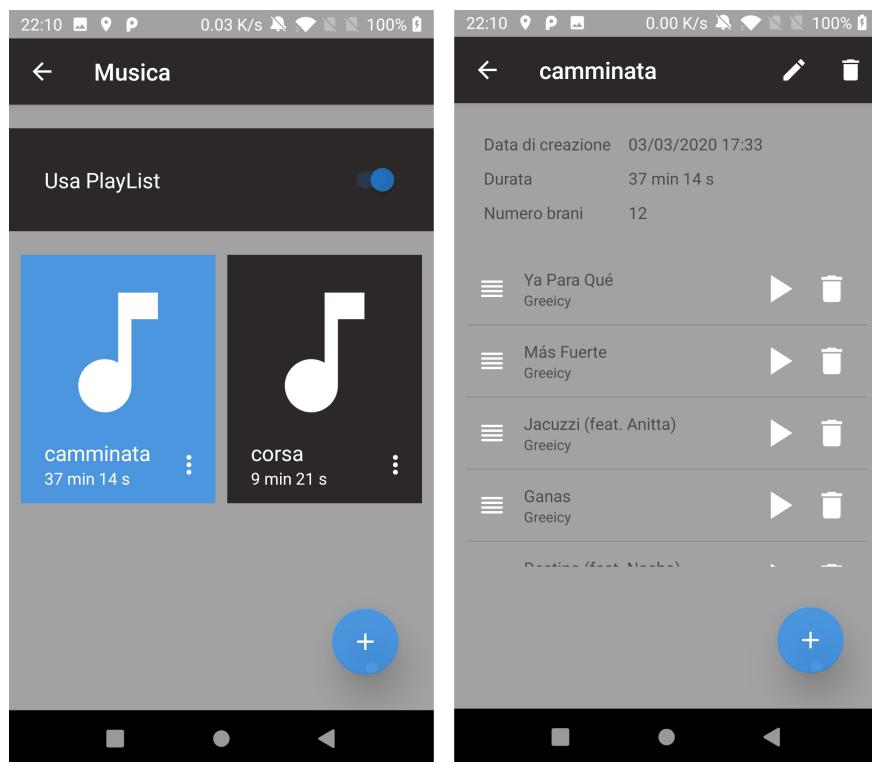


Figura 2.34: Impostazioni della Musica

La playlist viene riprodotta fino a quando l’allenamento non finisce o l’utente non lo disattiva.

Questa funzionalità è gestita dalla classe *MusicCoach* (*Figura 2.35*), che istanzia un oggetto *MediaPlayer*, utilizzato per controllare e riprodurre file e flussi audio.

```
private MusicCoach(Context context){
    this.context = context;
    this.resetPrimaryPlayList();
    this.isActive = Preferences.Music.isActive(context);
    this.mediaPlayerHelper = MediaPlayerHelper.getInstance(context);
    this.mediaPlayerHelper.setOnCompleteSong(mp -> {
        this.songPlaying = (songPlaying + 1) % songs.size();
        this.mediaPlayerHelper.reset();
        this.next();
    });
}

private void next() {
    this.mediaPlayerHelper.startMedia(
        this.songs.get(songPlaying).getPath());
}

public void start(){
    if(this.isActive()) {
        this.next();
    }else{
        release();
    }
}

public void stop(){
    if(this.isActive()) {
        this.mediaPlayerHelper.stopMedia();
    }else{
        release();
    }
}
```

Figura 2.35: *MusicCoach.java*

## 2.6.2 Allenatore vocale

Un'altra funzionalità dell' allenamento in Real-Time è quello di poter avviare un "allenatore vocale" che fornisce informazioni sul tempo trascorso, sulla distanza percorsa e sulle calorie consumate.

Questa funzionalità è gestita dalla classe *VoiceCoach*, la quale istanzia un oggetto *TextToSpeech*, classe fornita da Android, che sintetizza il testo per la riproduzione vocale immediata. Un'istanza TextToSpeech può essere utilizzata per sintetizzare il testo solo dopo aver completato l'inizializzazione (*Figura 2.36*).

```
private void init(){
    if(this.isActive && this.tts==null){
        this.tts = new TextToSpeech(context, status -> {
            switch (status){
                case TextToSpeech.SUCCESS:
                    tts.setLanguage(Language.getLocale(context));
                    tts.setOnUtteranceProgressListener(new UtteranceProgressListener(){
                        @Override
                        public void onStart(String utteranceId) {
                            Log.d(TAG, "onStart");
                            MusicCoach.getInstance(context).downVolume();
                        }

                        @Override
                        public void onDone(String utteranceId) {
                            Log.d(TAG, "onDone");
                            MusicCoach.getInstance(context).restoreVolume();
                        }
                        @Override
                        public void onError(String utteranceId) {

                        }
                    });
                    break;
            }
        });
    }
}
```

Figura 2.36: Inizializzazione istanza TextToSpeech (TTS)

Per sintetizzare il testo si chiama la funzione *speak(CharSequence text, int queueMode, Bundle params, String utteranceId)* (*Figura 2.37*)

```

private boolean isNowToSpeak(String time){
    int timeSec = Measure.Utilities.toSeconds(time);
    return lastSpeak + interval*60 <= timeSec ;
}

public void speakIfIsActive(String time, String distance, String calories){
    if(isNowToSpeak(time)){
        StringBuilder speak = new StringBuilder()
            .append(speakString(Measure.Type.DURATION, time))
            .append(speakString(Measure.Type.DISTANCE, distance))
            .append(speakString(Measure.Type.ENERGY, calories));

        if (speak.length() > 0)
        {
            lastSpeak = Measure.Utilities.toSeconds(time);
            tts.speak(speak.toString(), TextToSpeech.QUEUE_FLUSH,
                      null, TextToSpeech.ACTION_TTS_QUEUE_PROCESSING_COMPLETED);
        }
    }
}

```

Figura 2.37: Funzione di Speaking

Al termine dell'utilizzo dell'istanza TextToSpeech, si deve chiamare il metodo shutdown() per rilasciare le risorse native utilizzate dal motore TextToSpeech (*Figura 2.38*).

```

private void releaseTTS(){
    if(this.isActive && this.tts!=null){
        tts.stop();
        tts.shutdown();
        tts=null;
    }
}

```

Figura 2.38: Liberazione delle risorse di TextToSpeech (TTS)

Durante l'allenamento, viene gestito dall'AlarmManager (*Figura 2.39*), utilizzando la funzione *setRepeating(int type, long triggerAtMillis, long intervalMillis, PendingIntent operation)* settando come *intervalMillis* quello impostato dall'utente o quel-

lo di default (1 minuto). Per disattivarlo, viene utilizzata la funzione *cancel(PendingIntent operation)*.

Se è attivata anche la playlist, il volume della canzone in riproduzione viene ridotto e poi ripristinato quando il TTS è stato completato, come mostrato nella *Figura 2.36*.

```
public void start(){
    if(this.isActive){
        init();
        alarmManager.setRepeating(AlarmManager.RTC_WAKEUP,
            System.currentTimeMillis(),
            this.interval*60000, voicePendingIntent);
    }
}

public void stop(){
    if(this.isActive){
        alarmManager.cancel(voicePendingIntent);
        releaseTTS();
        release();
    }
}
```

Figura 2.39: Start e Stop dell’allenatore vocale

La Figura 2.40 è la schermata per attivare o disattivare l’allenatore vocale, prima di effettuare l’allenamento.

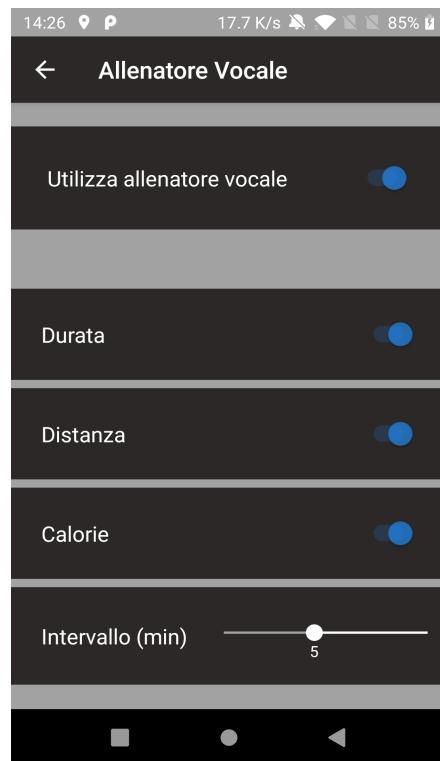


Figura 2.40: Impostazioni dell’Allenatore vocale

## 2.7 Statistiche

Un'altra funzionalità, che è stata implementata, è la visualizzazione di statistiche degli allenamenti e dell'andamento del peso corporeo.

Si possono filtrare i valori del grafico in base :

- al tipo di misura (velocità media, calorie, distanza e peso);
- al tempo: ultimo anno, mese, settimana o semplicemente tutti i valori (*Figura 2.41*).

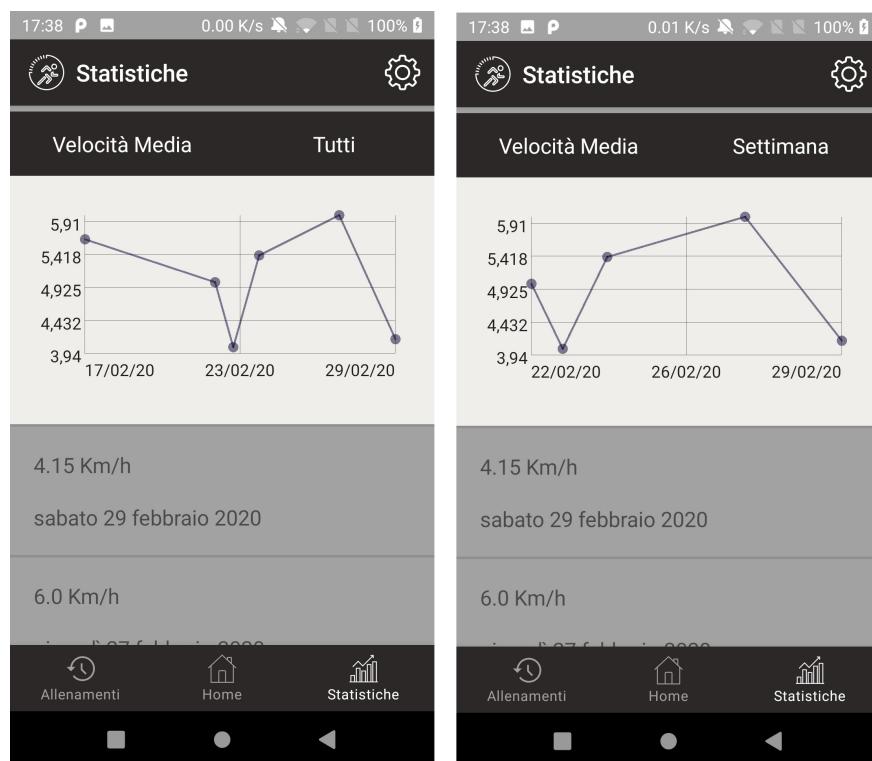


Figura 2.41: *Grafico filtrato della velocità*

Per visualizzare i grafici, è stata utilizzata una libreria di terze parti che mette a disposizione la classe *GraphView*.

```
final GraphView graphView = (GraphView) convertView.findViewById(R.id.statistics_graph);

final LineGraphSeries<DataPoint> seriesLine = new LineGraphSeries<>();
final PointsGraphSeries<DataPoint> seriesPoint = new PointsGraphSeries<>();

seriesLine.setColor(R.color.colorPrimaryDark);

seriesPoint.setSize(13);
seriesPoint.setColor(R.color.colorPrimaryDark);
final List<StatisticsData> statisticsFilteredGraphic = getStatisticsFilteredGraphic();
statisticsFilteredGraphic.stream().sorted((d1, d2) -> d1.getDate().compareTo(d2.getDate()))
    .forEach(data -> seriesPoint.appendData(new DataPoint(data.getDate(), data.getValue()),true, getCount()-1));
statisticsFilteredGraphic.stream().sorted((d1, d2) -> d1.getDate().compareTo(d2.getDate()))
    .forEach(data -> seriesLine.appendData(new DataPoint(data.getDate(),data.getValue()),true, getCount()-1));

graphView.addSeries(seriesPoint);graphView.addSeries(seriesLine);
// set date label formatter
graphView.getGridLabelRenderer().setPadding(100);
graphView.getGridLabelRenderer().setLabelFormatter(new DateAsXAxisLabelFormatter(context));
graphView.getGridLabelRenderer().setNumHorizontalLabels(3); // only 4 because of the space

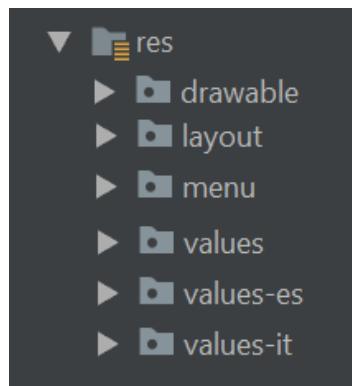
// set manual x bounds to have nice steps
final List<Date> x = statisticsFilteredGraphic.stream().map(StatisticsData::getDate).collect(toList());
if(!x.isEmpty()){
    graphView.setViewport().setMinX(x.get(statisticsFilteredGraphic.size()-1).getTime());
    graphView.setViewport().setMaxX(x.get(0).getTime());
    graphView.setViewport().setXAxisBoundsManual(true);

    // as we use dates as Labels, the human rounding to nice readable numbers
    graphView.getGridLabelRenderer().setHumanRounding(false);
}
```

Figura 2.42: Utilizzo della classe *GraphView*

## 2.8 Supporto Multi-Lingua

Il progetto è stato creato utilizzando gli strumenti SDK Android, che creano una directory res/ nel livello superiore del progetto. All'interno di questa directory res/ ci sono le sottodirectory per vari tipi di risorse. Ci sono anche alcuni file predefiniti, come res/values/strings.xml che contiene i valori di stringa.



L'applicazione supporta più lingue : Inglese (*Figura 2.43*), Italiano (*Figura 2.44*) e Spagnolo (*Figura 2.45*).  
La lingua di default è l'Inglese.

```
<resources>
    <string name="sport">Sport</string>
    <string name="walk">Walk</string>
    <string name="run">Run</string>
    <string name="profile">Profile</string>
    <string name="location">Location</string>
    <string name="measure_unit">Measure Unit</string>
    <string name="language">Language</string>
    <string name="vocal_coach">Voice Coach</string>
    <string name="info">Information</string>
    <string name="exit">Exit</string>
    ...
</resources>
```

Figura 2.43: *Stringhe in lingua inglese (values/strings.xml)*

```
<resources>
    <string name="sport">Sport</string>
    <string name="walk">Camminata</string>
    <string name="run">Corsa</string>
    <string name="profile">Profilo</string>
    <string name="location">Localizzazione</string>
    <string name="measure_unit">Unità di misura</string>
    <string name="language">Lingua</string>
    <string name="vocal_coach">Allenatore Vocale</string>
    <string name="info">Informazioni</string>
    <string name="exit">Esci</string>
    ...
</resources>
```

Figura 2.44: *Stringhe in lingua italiana (values-it/values.xml)*

```
<resources>
    <string name="sport">Deporte</string>
    <string name="walk">Caminata</string>
    <string name="run">Correr</string>
    <string name="profile">Perfil</string>
    <string name="location">Localización</string>
    <string name="measure_unit">Unidad de dimensión</string>
    <string name="language">Idioma</string>
    <string name="vocal_coach">Entrenador de voz</string>
    <string name="info">Información</string>
    <string name="exit">Sales</string>
    ...
</resources>
```

Figura 2.45: *Stringhe in lingua spagnola (values-es/values.xml)*

## 2.9 Unità di misura

L'applicazione prevede più unità di misura per la distanza, il peso, l'altezza e velocità (*Figura 2.46*).

```
public enum Unit {
    // DISTANCE
    KILOMETER("Km"),
    MILE("mi"),

    // WEIGHT
    KILOGRAM("Kg"),
    POUND("lb"),

    // HEIGHT
    METER("m"),
    FEET("ft"),

    // ENERGY
    KILO_CALORIES("Kcal"),

    // DURATION
    HOURS("h"),
    MINUTES("min"),
    SECOND("s"),

    // MIDDLE SPEED
    KILOMETER_PER_HOUR("Km/h"),
    MILE_PER_HOUR("mi/h");

    private final String str;

    Unit(String str) {
        this.str = str;
    }

    ...
}
```

Figura 2.46: Enumerazione *Measure.Unit*

Nei database vengono utilizzate le unità di misura internazionali ( Km, Kg, m ). Se l'utente cambia unità non cambia il valore memorizzato, ma viene convertito a run-time dall'applicazione, utilizzando la funzione *conversionTo(Unit unit, Double value)* (*Figura 2.47*).

```
public static Double conversionTo(Unit unit, Double value){
    switch (unit){
        case MILE:
            return ConversionUnitUtilities.kilometerToMile(value);
        case KILOMETER:
            return ConversionUnitUtilities.mileToKilometer(value);
        case POUND:
            return ConversionUnitUtilities.kilogramToPound(value);
        case KILOGRAM:
            return ConversionUnitUtilities.poundToKilogram(value);
        case FEET:
            return ConversionUnitUtilities.meterToFeet(value);
        case METER:
            return ConversionUnitUtilities.feetToMeter(value);
        case MILE_PER_HOUR:
            return ConversionUnitUtilities.kilometerForHoursToMileForHours(value);
        case KILOMETER_PER_HOUR:
            return ConversionUnitUtilities.mileForHoursToKilometerForHours(value);
    }
    return null;
}
```

Figura 2.47: Funzione di conversione di unità

La *Figura 2.48* è la schermata per cambiare le diverse unità di misura.

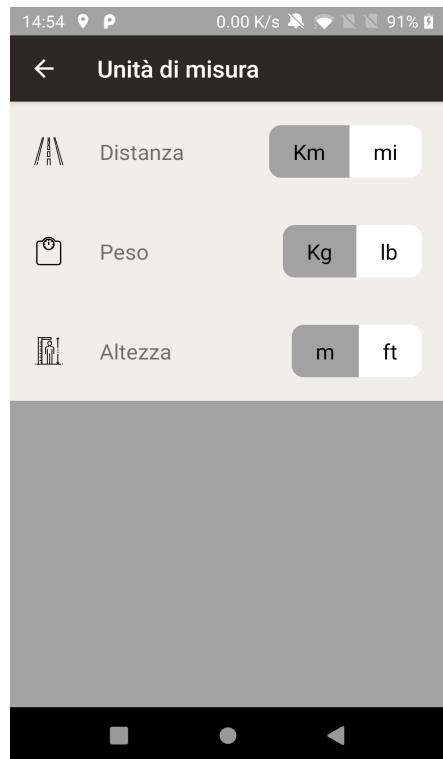


Figura 2.48: Impostazioni dell'Unità di misura

# Conclusioni e sviluppi futuri

Il progetto ha illustrato la creazione di un'applicazione Android e della relativa applicazione server per la gestione di allenamenti quotidiani, come una semplice camminata o una corsa.

L'architettura della piattaforma si compone di un'applicazione Client-Side, che permette di archiviare allenamenti, visualizzare le statistiche di quest'ultimi e di effettuare opportune chiamate all'applicazione Server-Side per sincronizzare i dati.

L'applicazione Server-side si occupa quindi di mettere a disposizione dei Client degli script PHP, che ricevono le richieste ed elaborano una risposta.

L'applicazione *Run/Walk Tracking* attualmente non si può definire come una versione definitiva, bensì come una prima versione funzionante e pronta a modifiche e a integrazioni con nuove funzionalità che la rendano affidabile e completa sotto tutti gli aspetti.

In futuro potrebbero essere introdotte le seguenti funzionalità o miglioramenti:

- Playlists in remoto.
- Versione Web (visualizzazione e modifiche se possibili)
- Statistiche più dettagliate (per esempio per ogni allenamento una statistica delle velocità raggiunte ).
- Integrazione con Smart Watch.
- Piani di allenamento in base al target inserito.
- Miglioramento delle prestazioni.
- Miglioramento dell’interfaccia grafica.

Oltre tutto il progetto potrebbe essere realizzato anche per altri sistemi operativi mobile, come iOS, in modo da raggiungere il maggior numero di utenti possibili.

# Glossario e sintesi delle convenzioni di notazione

## Glossario

**Attributo:** una delle proprietà significative di una relazione ai fini della descrizione della realtà applicativa di interesse.

**Key (chiave):** un vincolo intrarelazionale che interessa una sola tabella, che nel modello relazionale viene detta relazione.

**Primary Key (chiave primaria):** insieme di attributi che permette di individuare univocamente un record o tupla o ennupla in una tabella o relazione.

**Foreign Key (chiave esterna):** un insieme di attributi che fa riferimento a una chiave di un'altra tabella, permettendo in tal modo di esplicitare relazioni di tipo uno a molti tra tabelle attraverso quello che è chiamato vincolo di integrità referenziale.

**Alternate Key (chiave secondaria):** una chiave che non è stata selezionata come chiave primaria, ma è chiave candidata. Tuttavia, è considerata una chiave candidata per la chiave primaria.

**Web Hosting:** servizio di rete che consiste nell'allocare su un server web delle pagine di un sito web o di un'applicazione web, rendendolo così accessibile dalla rete Internet e ai suoi utenti. Tale "server web", definito "host", è connesso ad Internet in modalità idonea a garantire l'accesso alle pagine del sito mediante

il web browser dell'host client dell'utente, con identificazione dei contenuti tramite dominio ed indirizzo IP. Il servizio può essere gratuito o a pagamento, tipicamente a qualità maggiore nel secondo caso.

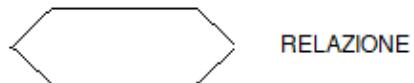
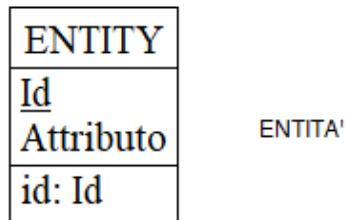
**Servizio Started:** un servizio che viene avviato quando un componente dell'applicazione lo avvia chiamando `startService()`. Una volta avviato, un servizio può essere eseguito in background a tempo indeterminato, anche se il componente che lo ha avviato viene distrutto.

**Servizio Bound:** un servizio il quale è associato quando un componente dell'applicazione si associa ad esso chiamando `bindService()`. Un servizio associato offre un'interfaccia Client-Server che consente ai componenti di interagire con il servizio, inviare richieste, ottenere risultati e persino farlo attraverso processi con comunicazione tra processi (IPC).

**Builder:** è uno dei più importanti pattern, meglio conosciuti come GoF design pattern. Nella programmazione ad oggetti tale pattern è molto di voga poiché separa la costruzione di un oggetto complesso dalla sua rappresentazione, cosicché il processo di costruzione stesso possa creare diverse rappresentazioni. In questo modo l'algoritmo per la creazione di un oggetto complesso è indipendente dalle varie parti che costituiscono l'oggetto e da come vengono assemblate. Ciò ha l'effetto immediato di rendere più semplice la classe, permettendo a una classe builder separata di focalizzarsi sulla corretta costruzione di un'istanza e lasciando che la classe originale si concentri sul funzionamento degli oggetti.

## Notazioni

### Progettazione Concettuale



### Progettazione Logica

**NOME** : nome della tabella

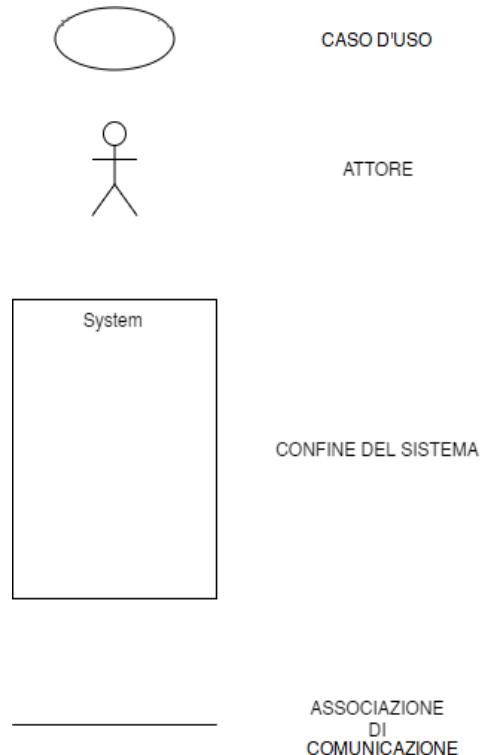
attributo: chiave primaria (se più attributi sono sottolineati si tratta di una chiave composita)

attributo\* : attributo opzionale

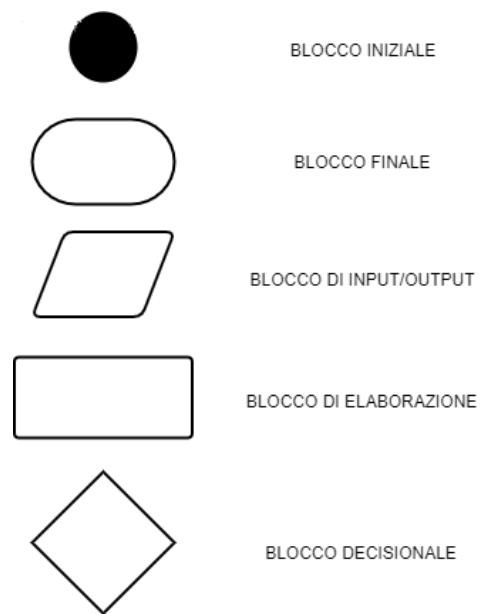
FK : Foreign Key

AK : Alternate Key

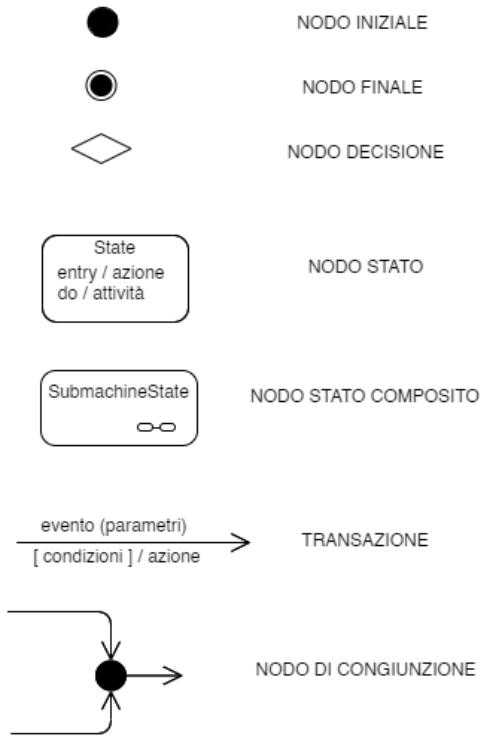
## Diagramma dei casi d'uso



## Diagramma di flusso



## Diagramma di stato



**entry/azione**: ogni volta che si entra in questo stato si compie "azione".

**do/attività**: ogni volta che ci si trova in questo stato si compie "attività".

**azione**: operazione istantanea, atomica e non interrompibile.

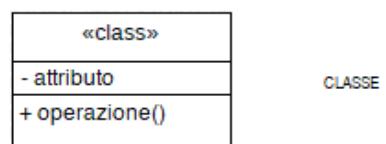
**attività**: operazione che richiede un certo tempo per essere completata e può quindi essere interrotta da un evento.

**evento**: operazione che provoca la transazione tra uno stato e l'altro.

**condizione**: espressione booleana che deve risultare vera affinchè la transazione possa avvenire.

Nota: una transazione che esce da uno stato e non riporta alcun evento indica che la transazione avviene al termine dell'attività.

## Diagramma delle classi



INTERFACCIA



PACKAGE



GENERALIZZAZIONE



DIPENDENZA



CONTENIMENTO

- : indica un attributo o un'operazione privata
- + : indica un attributo o un'operazione pubblica

# Bibliografia

- [1] Android. <https://it.wikipedia.org/wiki/Android>.
- [2] Builder Pattern un ottima alternativa al costruttore. <https://italiancoders.it/builder-pattern-un-ottima-alternativa-al-costruttore/>.
- [3] Developer Guides. <https://developer.android.com/guide>.
- [4] Formula per il calcolo delle calorie consumate con la corsa. <https://www.pesi-e-misure.it/site/formule/formula-calcolo-calorie-nella-corsa>.
- [5] Git(software). [https://it.wikipedia.org/wiki/Git\\_\(software\)](https://it.wikipedia.org/wiki/Git_(software)).
- [6] Google Api for Android. <https://developers.google.com/android/>.
- [7] Hosting. <https://it.wikipedia.org/wiki/Hosting>.
- [8] Introduzione a PHP. <https://www.html.it/pag/16673/cos-php/>.
- [9] Introduzione agli RDBMS e a MySQL. <https://www.html.it/pag/32137/introduzione-ai-rdbms/>.
- [10] Java (linguaggio di programmazione). [https://it.wikipedia.org/wiki/Java\\_\(linguaggio\\_di\\_programmazione\)](https://it.wikipedia.org/wiki/Java_(linguaggio_di_programmazione)).

- [11] MD5 hash in Java. <https://www.geeksforgeeks.org/md5-hash-in-java/>.
- [12] SQLite. <https://it.wikipedia.org/wiki/SQLite>.
- [13] P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, and R. Torlone. *Basi di dati Modelli e linguaggio di interrogazione*. McGraw-Hill, 2013.
- [14] Ian Sommerville. *Ingegneria del Software*. Pearson, 2017.