ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

APP MOBILE CON L'INTEGRAZIONE DI SOCIAL NETWORK E GOOGLE API PER MIGLIORARE L'ESPERIENZA UTENTE

Relazione finale in
PROGRAMMAZIONE DI SISTEMI MOBILE

Relatore
Dott. MIRKO RAVAIOLI

Presentata da
ANDREA DE CASTRI

Seconda Sessione di Laurea
Anno Accademico 2014 – 2015

PAROLE CHIAVE

Android

Social

HTTP

JSON

Fitness

"Computers have lots of memory but no imagination."

Indice

Introduzione **xi**

1 Stato dell'arte **1**
1.1 Introduzione e cenni storici 1
1.1.1 Linguaggi e Virtual Machine 3
1.2 Pro e contro 3
1.2.1 Varietà dei dispositivi 3
1.2.2 Gestione della memoria 4
1.2.3 Personalizzazione dell'interfaccia utente 5

2 Caso di studio **7**
2.1 Caso di studio 7
2.2 Considerazioni 8

3 Analisi e Modellazione **11**
3.1 Componenti base di Android 11
3.1.1 Activity 11
3.1.2 Manifest 12
3.2 Registrazione e Login 12
3.2.1 Comunicazione con il server 13
3.2.2 Verifica e attivazione dell'account 14
3.2.3 Conclusione della registrazione 14
3.3 Unità di misura 15
3.4 Database e salvataggio file 15
3.4.1 Vantaggi e svantaggi 15
3.4.2 Database online 15
3.4.3 Database locale 18
3.4.4 Salvataggio con SharedPreferences 18
3.4.5 Memorizzazione su file 18
3.5 Threading 19
3.5.1 AsyncTask 20
3.6 Processi in background 21

3.6.1 Service . . . 21
3.6.2 Broadcast Receivers . . . 22
3.7 GPS . . . 22
3.8 Google API . . . 23
3.8.1 Acquisti in App . . . 23
3.9 Facebook API . . . 24
3.10 Lato Server . . . 25

4 Progettazione **27**
4.1 Organizzazione in package . . . 27
4.2 Organizzazione delle risorse . . . 28
4.3 Classi e file . . . 29
4.4 Descrizione delle classi principali . . . 29
4.4.1 Model . . . 29
4.5 Salvataggio dei dati . . . 31
4.6 Le Activity principali . . . 32
4.7 Servizi . . . 40
4.7.1 Gestione delle notifiche da parte dei Servizi . . . 42
4.8 Eccezioni utilizzate . . . 42
4.9 Componenti grafiche utilizzate . . . 42
4.9.1 Adapters . . . 44
4.10 Layout utilizzati e la loro composizione . . . 45

5 Implementazione **47**
5.1 Uso degli AsyncTask . . . 47
5.2 Condivisione tramite Open Graph . . . 48
5.3 Check degli appuntamenti e lancio di una notifica . . . 49
5.4 Calcolo delle calorie e tracciamento del percorso . . . 50
5.5 Gestione delle date degli appuntamenti . . . 51
5.6 Registrazione online . . . 52

6 Sviluppo e Testing **55**
6.1 Computer utilizzato . . . 55
6.2 IDE utilizzato . . . 55
6.2.1 Creazione di un progetto . . . 55
6.3 SDK di Android . . . 56
6.4 Librerie esterne utilizzate . . . 56
6.5 Test dell'applicazione . . . 57

Conclusioni **59**

Ringraziamenti **61**

Bibliografia **63**

Introduzione

Già da diversi anni, grazie al progresso tecnologico, lo sviluppo di software per i dispositivi mobile è aumentato in maniera considerevole; questo perché gli utenti non vogliono più essere vincolati ad utilizzare programmi sempre seduti davanti ad una scrivania, ma vogliono, e a volte necessitano, di utilizzare l'applicativo ovunque si trovino. Infatti, dal 2007 ad oggi, il mercato degli smartphone e dei tablet è cresciuto esponenzialmente:
alcuni dati statistici indicano che, in media, ci sono 1,7 dispositivi pro capite.

Queste tecnologie sono state talmente invasive nelle nostre vite che le usiamo quotidianamente, al punto tale da aver bisogno di accedere ai nostri dati ovunque ci troviamo, interagire con altri utenti, condividere le nostre esperienze sui social network e molto altro.

Un forte contributo allo sviluppo di queste tecnologie è dato dall'evoluzione dei sistemi operativi di questi dispositivi, che hanno permesso ai programmatori di costruire software sempre più complessi su apparecchi di piccole dimensioni.

Questa tesi esamina la progettazione e lo sviluppo di un'applicazione mobile Android che è in grado di gestire l'attività sportiva di un utente. L'applicazione offre numerose funzionalità, che permettono all'utente di eseguire allenamenti per il fitness e allenamenti per la corsa, tenendo sempre sotto controllo i risultati ottenuti e tutte le informazioni necessarie. Oltre ad eseguire allenamenti l'utente può crearne di propri e modificarli a suo piacimento, in più nell'App è inserito lo shop dove l'utilizzatore può comprare allenamenti messi a disposizione direttamente da FitBody.

Gli aspetti visti sopra saranno descritti attraverso un'analisi del problema e un'analisi sulla progettazione architetturale. In particolare verranno sottolineati aspetti riguardanti l'interazione tra utenti e l'utilizzo di API che permetteranno all'utilizzatore di condividere le proprie esperienze sul social network Facebook e di avere un'esperienza completa con l'app.

In questo scritto si parlerà anche della comunicazione tra applicazione e server, che avviene grazie a chiamate HTTP con metodo POST. Attraverso queste chiamate l'applicazione leggerà e scriverà informazioni sul database online, hostato sulla piattaforma Altervista. L'applicazione web, di cui sarà data solamente un'infarinatura, è stata sviluppata utilizzando il linguaggio di programmazione PHP. Ogni response inviata dal server al client è composta da uno o più oggetti JSON.

Un oggetto JSON è un messaggio testuale che rispetta una determinata sintassi. E' stato scelto di utilizzare il JSON poiché è indipendente dal linguaggio dell'applicazione sullo smartphone, offrendo larga scalabilità all'App.

Inoltre sarà posta l'attenzione sulla reattività dell'App. Dato che si sta lavorando su un dispositivo di piccole dimensioni e con risorse limitate è necessario che le operazioni più pesanti siano eseguite su Thread differenti, mantenendo l'interfaccia grafica sempre reattiva agli input dell'utente. Le operazioni che saranno gestite da Thread secondari sono le richieste HTTP e alcune letture e scritture sul database locale.

Il punto di forza dell'App è la sua interfaccia grafica, che si presenta semplice, ben organizzata e di bell'aspetto; caratteristiche di fondamentale importanza per coinvolgere sempre di più l'utente nell'utilizzo dell'applicativo. L'applicazione utilizza un design Material, cioè un design innovativo lanciato da Google, in cui troviamo nuovi temi, nuovi widgets e nuove API che hanno permesso di personalizzare componenti grafiche.

Il software, in tutte le sue parti, è stato strutturato e sviluppato in modo da supportare il multilingua. Infatti tutti i contenuti dell'applicazione e tutte le informazioni prese dal database online possono essere visualizzate sia in italiano che in inglese aumentando il bacino di utenza.

La tesi è strutturata in sei capitoli.

Nel primo sarà fatto un quadro generale sulla piattaforma Android ed un confronto con altri sistemi mobile, mettendo in risalto alcuni pro e contro.

Nel secondo verranno descritte le funzionalità dell'applicazione e l'organizzazione del menu principale.

Il terzo capitolo tratta l'analisi e la modellazione del programma, mettendo in evidenza il salvataggio e il caricamento dei dati, l'utilizzo del multithreading, le API utilizzate e soluzioni delle problematiche riscontrate.

Nel quarto capitolo viene descritta la struttura dell'intero progetto, anche con l'aiuto di opportuni diagrammi UML, soffermandosi sulla struttura delle classi e la loro funzionalità all'interno dell'App.

Successivamente nel quinto capitolo vengono mostrati alcuni frammenti di codice ritenuti più importanti per il funzionamento dell'applicazione.

Infine l'ultimo capitolo contiene le tecnologie utilizzate per lo sviluppo e una descrizione generale dei test effettuati su diversi dispositivi.

Capitolo 1

Stato dell'arte

In questo capitolo verrà effettuata una panoramica generale sulla piattaforma Android, mettendo in evidenza i pro e contro del sistema operativo mobile di Google.

1.1 Introduzione e cenni storici

Android è un sistema operativo open source per dispositivi mobile sviluppato da Google Inc. e basato su kernel Linux. Questo sistema è stato progettato inizialmente per smartphone e tablet, ma negli ultimi anni viene utilizzato anche per dispositivi wearable (Android Wear), come orologi e occhiali, e anche per televisori (Android TV).

La diffusione a larga scala di questo sistema operativo è dovuta al fatto che il Software è **free** e **open source**, a differenza dei rivali iOS e Windows Phone. Inoltre, esiste una comunità di utenti molto vasta che sviluppa applicazioni con l'obiettivo di aumentare e migliorare l'esperienza degli utilizzatori, anche modificando e distribuendo versioni personalizzate di Android, permesse dalla politica adottata da Google. Tutto ciò ha permesso ad Android di essere il sistema mobile più diffuso con, ad oggi, oltre 1 miliardo di dispositivi venduti ogni anno.

Figura 1.1: Logo di Android

Storia di Android

Android Inc. nasce nel 2003 in California. L'azienda fu fondata da Andy Rubin, Rich Minerva, Nick Sears e Chris White con l'obbiettivo di rendere i *cellulari più consapevoli e intelligenti dei loro proprietari.* Agli albori la società lavorò in segreto, rivelando solo di progettare software per dispositivi mobili.

Nel 2005, Google, che voleva entrare nel mercato della telefonia mobile, decise di acquistare l'azienda, assumendo tutto il team di sviluppo. Proprio in quegli anni, Rubin aveva iniziato a sviluppare il sistema operativo basato sul kernel Linux. Oggi il sistema operativo Android è composto in tutto da 12 milioni di righe di codice, in gran parte scritte utilizzando il linguaggio C e C++.

Dal 2008, data di lancio del primo dispositivo equipaggiato con Android, sono stati rilasciati numerosi aggiornamenti. Una delle caratteristiche più evidenti di Android è che le sue diverse release non sono soltanto distinte da un numero di versione, secondo standard informatici, ma anche da un *codename* ispirato a dolci americani in ordine alfabetico, partendo dalla prima versione chiamata *Cupcake* fino ad arrivare alle versioni più recenti *Ice Cream*, *Jelly Bean*, *KitKat*.

La versione attuale, rilasciata nell'Ottobre 2014 è la 5.0 con il nome *Lollipop*. Google ha appena annunciato che è in fase di sviluppo la prossima versione 6.0, identificata ufficialmente con il nome *Marshmallow*.

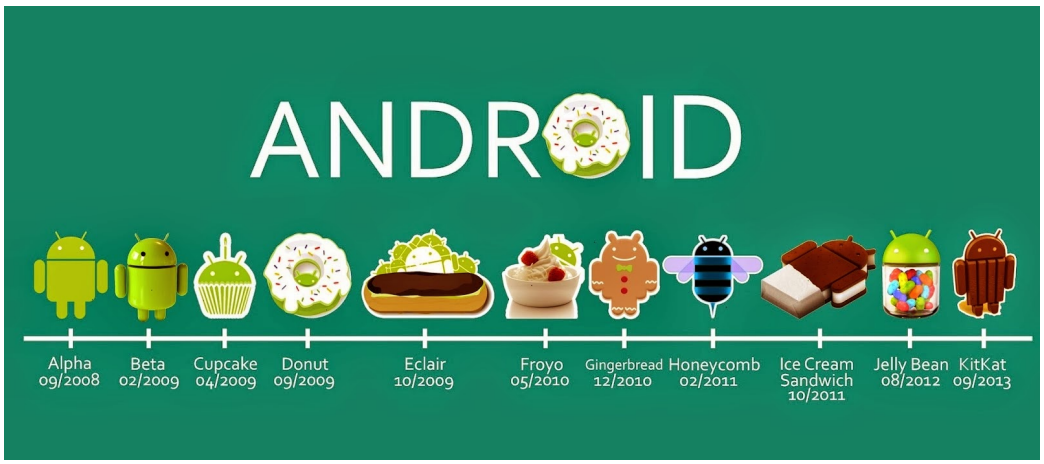

Figura 1.2: Loghi delle varie release di Android

1.1.1 Linguaggi e Virtual Machine

Gli sviluppatori possono implementare le App in diversi linguaggi di programmazione. Il più utilizzato è *Java*, ma Google mette a disposizione altri due linguaggi:

- Android Scripting Environment (ASE): linguaggio semplificato di alto livello;
- Android Native Development Kit (AND): linguaggio che permette di sfruttare tutte le potenzialità hardware del dispositivo.

Android non esegue il bytecode di Java, per questo ha bisogno di una propria *virtual machine* che riesca ad eseguire applicazioni su un dispositivo con risorse limitate. Inizialmente, la Virtual Machine adottata prendeva il nome di *Dalvik* e, a differenza della **JVM**, non gestiva le eccezioni ed aveva un'architettura a registri.

Dalla versione 2.2, per migliorare le prestazioni della macchina virtuale, è stato incluso un compilatore **JIT** (Just in Time), che compila parte dell'App durante il suo utilizzo. Dalla versione 5.0 la *Dalvik Virtual Machine* è stata sostituita dalla *runtime Art* (Android Run Time), basata su una tecnologia **AOT** (*ahead-of-time*), che esegue la compilazione del codice solamente durante l'installazione dell'App nel dispositivo, diversamente dalla precedente **DVM**. In questo modo si hanno vantaggi in termini di prestazioni e di gestione delle risorse.

1.2 Pro e contro

1.2.1 Varietà dei dispositivi

Il sistema operativo Android, data la sua diffusione, viene utilizzato su migliaia di dispositivi diversi, dalle dimensioni dello schermo fino ad arrivare a differenti configurazioni dell'hardware. Una moltitudine di marchi ha adottato Android facendo salire il numero di modelli che lo hanno installato a 18.769.

Il problema della frammentazione, seppur ancora non superato del tutto, è stato un grande problema per gli sviluppatori negli anni passati. Non è semplice adattare un'App su schermi di misure diverse, in termini di pollici, forma e risoluzione. Il programmatore, ancora oggi nella maggior parte dei casi, deve creare più layout, quindi più interfacce utente, che dovranno mostrare la

stessa schermata. Il problema degli schermi è solo uno dei tanti altri problemi della frammentazione dei dispositivi su cui è installato Android.

I programmatori che volevano creare App con l'utilizzo della camera dovevano avere la certezza che il dispositivo su cui andava installato il loro applicativo avesse la fotocamera. Ci sono molte altre difficoltà come queste, che, mano a mano, si stanno assottigliando grazie ad alcuni standard introdotti da Google; infatti tutti i produttori di telefonia mobile o tablet, che vorranno usare nei loro dispositivi Android, dovranno rispettare gli standard dettati come, ad esempio, "tutti i telefoni dovranno avere almeno una fotocamera".

Queste problematiche, invece, non hanno mai disturbato i programmatori iOS, poiché Apple produce una mole ridotta di modelli, i cui diversi smartphone, allo stato attuale, possono presentare solo tre risoluzioni differenti dello schermo.

Figura 1.3: Varietà di modelli Android

1.2.2 Gestione della memoria

I dispositivi Android hanno installata molta memoria RAM rispetto ai dispositivi che utilizzano iOS. Questo non vuol dire, come ci vogliono far credere, che un dispositivo con 2GB di RAM è sempre migliore, come prestazioni, di un altro che ne ha solamente 1GB. Le specifiche non sono sufficienti a raccontare tutta la verità, infatti, nonostante alcuni dispositivi hanno meno memoria, sono migliori in prestazioni.

Uno dei motivi per cui Android è più lento in alcune operazioni è che utilizza applicazioni scritte in linguaggio di programmazione Java, eseguite attraverso una macchina virtuale. Inoltre Java ha bisogno del processo *Garbage Collection* per la gestione della memoria. Questo meccanismo gestisce in modalità automatica la memoria, liberando porzioni di essa che non saranno più utilizzate successivamente dall'applicativo, nel momento in cui lo riterrà più opportuno.

Sistemi senza *Garbage Collection* hanno un consumo di memoria ridotto, in più, consumando poca memoria riesco ad avere un consumo energetico limitato aumentando la durata e la vita della batteria. Anche se il Garbage Collector limita le prestazioni di un dispositivo, questo ha anche aspetti positivi: infatti la sua presenza esonera il programmatore dall'eseguire manualmente l'allocazione e la deallocazione di aree di memoria, riducendo o eliminando alcune categorie di bug.

1.2.3 Personalizzazione dell'interfaccia utente

Il punto di forza di Android è sempre stato, oltre la sua natura free e open source, la sua personalizzazione. Infatti Google permette alle aziende produttrici di smartphone di personalizzare la grafica di Android e offre all'utente la possibilità di customizzare le impostazioni del sistema operativo.

Il *Software Development Kit* (SDK) di Android mette a disposizione un ampio set di componenti grafiche per interfacciarsi all'utente e mostrare diversi tipi di dati. A differenza di iOS, che ha un'interfaccia utente *povera* ma più leggera, la *GUI* di Android, a discapito delle prestazioni, è molto più coinvolgente e personalizzabile. Infatti, attraverso il linguaggio *XML*, è possibile creare layout semplici con pochi elementi oppure layout elaborati con molte componenti grafiche. Alcune componenti grafiche, che hanno permesso agli utenti di facilitare l'interazione con le App, sono state le *notifiche* ed i *widget*.

Per quanto riguarda le notifiche, Android è decisamente in testa grazie alla sua compresione del loro ruolo chiave, dato che è molto più semplice vedere un messaggio sulla schermata di blocco che aprire un'applicazione per ricevere aggiornamenti. Dall'ultima release di Android il programmatore può settare anche un preset alle notifiche, per esempio è possibile scegliere il preset *MediaPlayer* per indicare una notifica dove saranno presenti i tasti indispensabili per interagire con un lettore musicale.

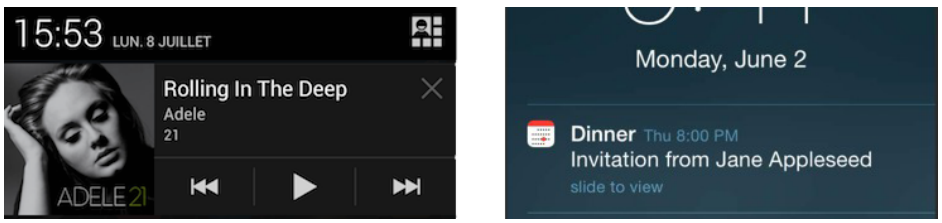

Figura 1.4: Differenza tra notifiche - Android a sinistra iOS a destra

Capitolo 2

Caso di studio

In questo capitolo verranno trattate le funzionalità più importanti dell'App descritta in questa tesi.

2.1 Caso di studio

FitBody è un'App Android che permette di gestire allenamenti di fitness e controllare i progressi durante una corsa. L'idea di fondo è quella di creare un'applicazione in grado di competere con altre App nel campo delle applicazioni per il fitness.

Le funzionalità che fornisce l'applicativo sono molteplici:
prima di tutto l'utente deve registrarsi nell'apposito form, dove gli vengono richiesti alcuni dati obbligatori, in modo da poter accedere al menu principale, dopo aver effettuato il login. Vista la diffusione dei social network, è stato aggiunto anche il login attraverso Facebook, che risulta più veloce e più gradito dagli utenti. Una volta effettuato il login, l'utente si troverà nel menu principale, in cui l'*user* potrà muoversi nelle diverse sezioni dell'App. Oltre a muoversi tra le sezioni, elencate di seguito, l'utente può vedere il suo nome e quanti Km e Kcal ha consumato dal primo utilizzo dell'applicazione.

Le sezioni presenti nel menu principale sono:

- Fitness;
- Running;
- Schedule;
- Shop;

- Feeding;
- Setting;

Ogni sezione o schermata avrà accesso a specifiche funzionalità, che verranno spiegate nel dettaglio nei capitoli successivi.

La schermata *Fitness* permette di creare/modificare/eliminare allenamenti di fitness, controllare i progressi dei workout e di accedere alla schermata di allenamento.

La seconda sezione dell'applicazione ha il compito di mostrare all'utente i suoi obbiettivi settimanali e mensili in termini di Km percorsi e calorie bruciate. Si può accedere alla schermata di allenamento, dove verranno visualizzate le informazioni della corsa.

Nella terza sezione l'utente può visualizzare i suoi appuntamenti con la palestra, se risulta iscritto ad una, e richiederne degli altri.

Nello *Shop*, ovvero la quarta schermata, l'utilizzatore può acquistare con soldi reali degli allenamenti che riguardano il fitness. I pagamenti dei vari prodotti vengono gestiti attraverso *Google Market*.

Nella penultima schermata si può misurare la propria percentuale di massa magra e massa grassa in modo da tenersi sempre aggiornati sulla forma fisica. Queste percentuali sono calcolate attraverso formule individuate dopo un'accurata documentazione online.

Infine nell'ultima sezione l'utente ha accesso alle impostazioni dell'App; qui può cambiare le unità di misura da utilizzare, iscriversi ad una palestra che faccia parte di *FitBody*, associare o dissociare un suo social account, vedere i suoi dati personali ed effettuare il logout.

2.2 Considerazioni

Alcune caratteristiche importanti, descritte nei capitoli seguenti, sono la possibilità di condividere sulla propria bacheca *Facebook* i progressi e gli allenamenti conclusi. La maggior parte di App sfrutta la condivisione per farsi pubblicità nei social network. La condivisione, in molti casi, dà visibilità all'applicazione, dandole l'opportunità di avere una diffusione su vasta scala.

Un'altra importante *feature* dell'App è il supporto multilingua, fondamentale per ampliare il bacino di utenza. La lingua predefinita dell'applicazione è

l'inglese. Supporta però la lingua italiana, che verrà automaticamente mostrata agli utenti se l'applicazione viene eseguita su uno smartphone con lingua impostata all'italiano.

Capitolo 3

Analisi e Modellazione

In questo capitolo verranno descritte le basi dell'applicazione attraverso modelli semplificati, che saranno delineati nel dettaglio nel prossimo capitolo.

3.1 Componenti base di Android

3.1.1 Activity

Dato che l'App descritta in questa tesi è stata sviluppata per la piattaforma Android, non si può non parlare del componente fondamentale di un'applicazione Android: l'*Activity*. Un App Android è strutturata in modo tale da seguire il pattern di programmazione **MVC** [1] (model-view-controller), rendendo più leggibile e modulare il codice.

Oltre a quanto detto, Android presenta una vera e propria separazione tra il codice dell'interfaccia utente e codice della gestione degli eventi. Le *Activity*, infatti, sono dei veri e propri *Controller* a cui è associata una **GUI**. L'interfaccia utente è descritta da file *XML*, mentre le Activity sono implementate utilizzando il linguaggio di programmazione *Java*. Gli input generati dall'utente nell'interfaccia dedicatagli sono gestiti dal controller.

Ogni Activity ha un proprio ciclo di vita, che va dalla creazione alla sua distruzione. Il sistema per lanciare una nuova Activity fa uso di oggetti chiamati *Intent*, che sono utilizzati per gestire la navigazione all'interno dell'App. Per avviare la nuova Activity basta chiamare il metodo *startActivity(Intent intent)* che accetta un Intent come parametro, avente riferimento l'Activity di partenza e l'Activity da visualizzare.

[1] MVC è un pattern architetturale di programmazione utilizzato nell'ambito della programmazione orientata agli oggetti. Questo pattern è in grado di separare la logica di presentazione dei dati dalla logica di gestione degli stessi.

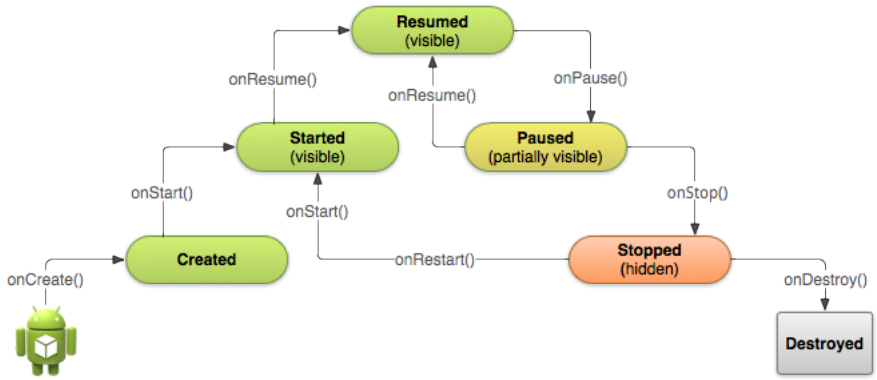

Figura 3.1: Ciclo di vita di un'Activity

3.1.2 Manifest

Ogni applicazione Android deve avere un file *AndroidManifest.xml* nella cartella principale del progetto. Il file *Manifest* contiene le informazioni essenziali dell'App e del sistema Android. Nello specifico contiene:

- il **Package** principale dell'applicazione che funge da identificativo dell'App nello *Store*;
- le **Activity**, i **Service** ed i **BroadcastReceiver** che compongono l'applicativo;
- un **Intent-filter** che ha il compito di avviare la prima Activity.

Oltre a quanto descritto, nel file del manifest vengono dichiarati i permessi che sono richiesti dal sistema per interagire con le componenti dell'applicazione. Attraverso queste **uses-permission** l'applicazione può accedere alle funzionalità fornite dal sistema e dal dispositivo, come la possibilità di utilizzare la connessione internet, il *GPS* oppure di salvare i file all'interno del dispositivo. Tutte queste operazioni, per poter essere effettuate, devono necessariamente essere descritte dell'interno del Manifest. La mancata descrizione di un permesso farà *crashare* l'applicativo.

3.2 Registrazione e Login

Per aver accesso a tutte le funzionalità dell'App è necessario avere un account. Per questo motivo l'utente, prima di poter effettuare il primo login, deve registrarsi attraverso un apposito form. I dati che sono richiesti alla registrazione sono l'indirizzo e-mail, che funge da ID dell'utente, ed una password. La

password che viene fornita dall'utente, prima di essere salvata, viene criptata in modo da proteggere la privacy dello stesso.

Nel caso in cui l'utente decida di accedere all'applicazione tramite *Facebook* non verrà richiesta nessuna password.

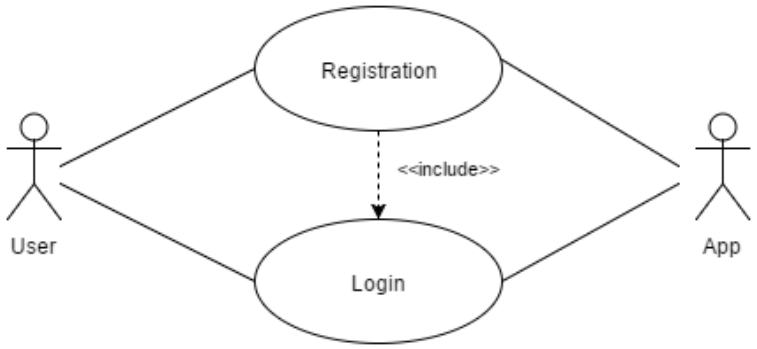

Figura 3.2: Diagramma dei casi d'uso per accedere all'App

3.2.1 Comunicazione con il server

La comunicazione tra applicazione e server avviene attraverso richieste *HTTP* con metodo *POST*, poiché si preferisce inserire i dati delle form nel corpo del messaggio. La risposta del server nella maggior parte dei casi è un oggetto *JSON*, che successivamente sarà convertito in oggetto Java.

Per quanto riguarda la registrazione, la risposta che riceve il *Client* è un valore *booleano*, che indica se la richiesta di registrazione sia andata a buon fine. Nell'immagine sottostante possiamo vedere un esempio di comunicazione tra App e server per la registrazione di un utente.

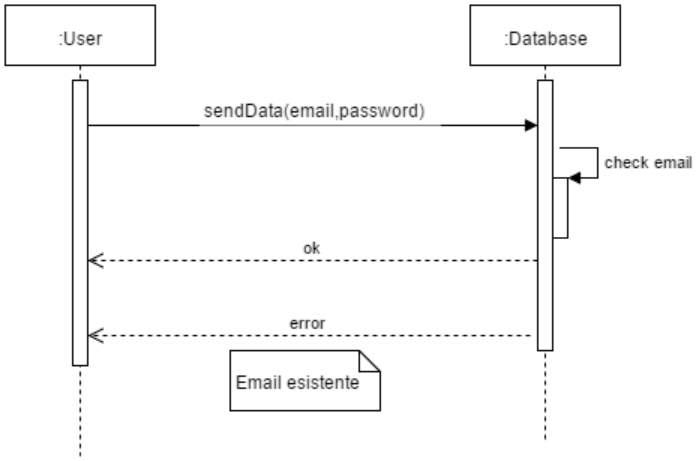

Figura 3.3: Diagramma di sequenza per registrarsi all'App

3.2.2 Verifica e attivazione dell'account

Per diminuire lo spam nelle registrazioni è stata inserita una verifica dell'e-mail: infatti, una volta completata la registrazione, l'account non è ancora attivo. Il server genererà un codice per ogni utente registrato, successivamente spedirà al nuovo utente un'e-mail con un link. Cliccando sul link si effettuerà una richiesta HTTP con metodo *GET* che avrà come dati due coppie nome-valore. I dati che saranno inviati dalla richiesta sono l'e-mail dell'utente e il codice generato dal server. Infine il server effettuerà un *check* dei dati e se ci sarà corrispondenza tra i dati l'account sarà attivato.

Esistono due stati in cui si può trovare un account:

- Inattivo;
- Attivo;

Se un utente provasse ad effettuare il login, ma il suo account non fosse ancora attivato, gli verrebbe notificata la mancata verifica dell'account. Nell'immagine che seguirà viene mostrato il ciclo di vita di un account utente attraverso un diagramma a stati.

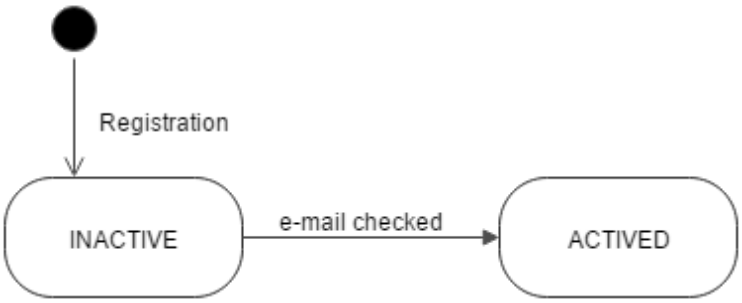

Figura 3.4: Diagramma degli stati di un account

3.2.3 Conclusione della registrazione

Non possiamo ancora dire che la registrazione sia conclusa, dato che una volta effettuato il primo login viene richiesto all'utente di inserire altri dati obbligatori quali il suo nome completo, il sesso, la sua altezza e il suo peso. Questi dati sono indispensabili per calcolare con più precisione il consumo energetico durante un'attività sportiva. Una volta riempiti tutti i campi obbligatori verrà effettuato un *update* del database online e si avrà accesso al menu principale dell'App.

3.3 Unità di misura

In un'App dedicata allo sport è importante e soprattutto utile l'utilizzo di unità di misura. Le unità di misura di default sono quelle adottate dal sistema internazionale, ovvero il metro per la lunghezza, il chilogrammo per la massa e il secondo per il tempo. Viene data la possibilità di utilizzare unità di misura differenti da quelle predefinite, ad esempio per la massa possono essere usate le libbre. Per gestire con più semplicità i dati, essi vengono salvati sempre con valori corrispondenti alle unità di misura del sistema internazionale. Successivamente verranno convertiti, se necessario, per essere mostrati all'utente.

3.4 Database e salvataggio file

Per quanto riguarda la persistenza dei dati, *FitBody* utilizza due database, uno locale e uno online. Inoltre viene utilizzato anche il salvataggio su file per memorizzare piccoli oggetti, come lo user corrente e le sue preferenze.

3.4.1 Vantaggi e svantaggi

L'applicazione poteva essere strutturata usando un unico database online, ma per diversi motivi si è optato per l'utilizzo di due database. Grazie ad un database locale il funzionamento dell'App non è vincolato dalla presenza di connessione internet, altrimenti, se il database fosse stato solo online, l'applicazione non sarebbe stata utilizzabile in mancanza di rete. Un ulteriore vantaggio è dato dalla distribuzione del carico di lavoro del server. Usando un unico database il server avrebbe dovuto gestire tutte le richieste inviate dagli utenti, in questo caso invece le richieste al server sono di gran lunga inferiori.

3.4.2 Database online

Per permettere all'utente di connettersi su più dispositivi differenti è stato necessario utilizzare un database online. In questo modo l'utente riesce a reperire i suoi dati in qualunque momento. Il database online è hostato sulla piattaforma *Altervista*. La banca dati è composta da 8 tabelle:

- USER;
- GYM;
- SUBSCRIBE;

- PROGRAM;
- EXERCISE;
- EXERCISE IN PROGRAM;
- APPOINTMENT;
- PURCHASE;

Tabella User In questa tabella vengono salvati tutti gli utenti registrati a *FitBody*, vengono memorizzate le e-mail, le password criptate, il nome completo, il sesso, il peso, l'altezza, i Km totali percorsi, le calorie totali bruciate e la data di registrazione. Inoltre viene salvato il codice di attivazione dell'account e un valore boolean che sta ad indicare se l'account è stato attivato o meno.

Tabella Gym Questa tabella contiene le tuple che rappresentano le palestre registrate su *FitBody*. Per il momento viene memorizzato solamente l'ID, che deve necessariamente essere un'email, e l'indirizzo in cui si trova la palestra.

Tabella Subscribe Nella tabella *Subscribe* vengono salvate le iscrizioni degli utenti alle palestre. Per il momento contiene *foreign keys* riferite alla tabella *User* e *Gym*.

Tabella Program I record contenuti in questa tabella rappresentano gli allenamenti che sono stati offerti da *FitBody* agli utenti, oppure gli allenamenti creati dalle palestre e inviati ad uno specifico utente. I campi salvati per ogni tupla sono l'e-mail dell'utente, l'e-mail della palestra e il nome del programma.

Tabella Exercise La tabella di cui parleremo adesso è la più importante per l'App, poiché ogni utente al primo avvio dell'applicazione scaricherà tutti gli esercizi per il fitness attingendo da questa tabella. Ogni esercizio è composto da un ID, un nome, la zona muscolare allenata dall'esercizio, una descrizione dei muscoli allenati, una descrizione dell'esecuzione dell'esercizio, un link e il consumo orario in Kcal. Tutti i campi descrittivi sono duplicati, poiché i dati sono salvati sia in italiano sia in inglese.

Tabella Exercise in Program In quest'altra tabella vengono salvati gli esercizi contenuti dagli allenamenti. L'ID di questa tabella è composto dall'e-mail dell'utente e della palestra, dal nome del programma, dall'ID dell'esercizio e da un numero progressivo. I campi restanti serviranno per memorizzare i dati

di ogni esercizio, cioè le ripetizioni, il tempo di recupero, la durata e il peso da utilizzare. Oltre ai soliti dati è stato aggiunto anche il tipo di serie da effettuare per l'esercizio, ovvero una serie potrebbe essere con peso crescente, decrescente o semplicemente utilizzando un peso costante. Alcuni campi come la durata o il peso possono essere lasciati vuoti, dato che molti esercizi non ne hanno bisogno.

Tabella Appointment Qui verranno salvate le richieste di appuntamento degli utenti. Un utente può richiedere un appuntamento con un istruttore alla palestra a cui è scritto. Ogni tupla è composta da una data di richiesta, che servirà a comporre l'ID insieme all'e-mail dell'utente e della palestra, una data di appuntamento, un messaggio e un risultato.

Tabella Purchase Potendo effettuare acquisti in App è opportuno salvarsi gli acquisti effettuati dagli utenti. Un acquisto è composto dall'e-mail dell'utente e il programma a cui si riferisce l'acquisto.

Alcune tabelle hanno altri campi che non sono stati menzionati, perché ancora non utilizzati. L'applicazione è stata sviluppata per essere ampliata successivamente, per questo alcune tabelle sono state predisposte per modifiche future. I valori testuali delle tabelle sono stati salvati con la codifica *UTF-8 Unicode-ci*, poiché il *Server* risponderà al *Client* con oggetti *JSON*, il quale supporta solamente la codifica UTF-8.

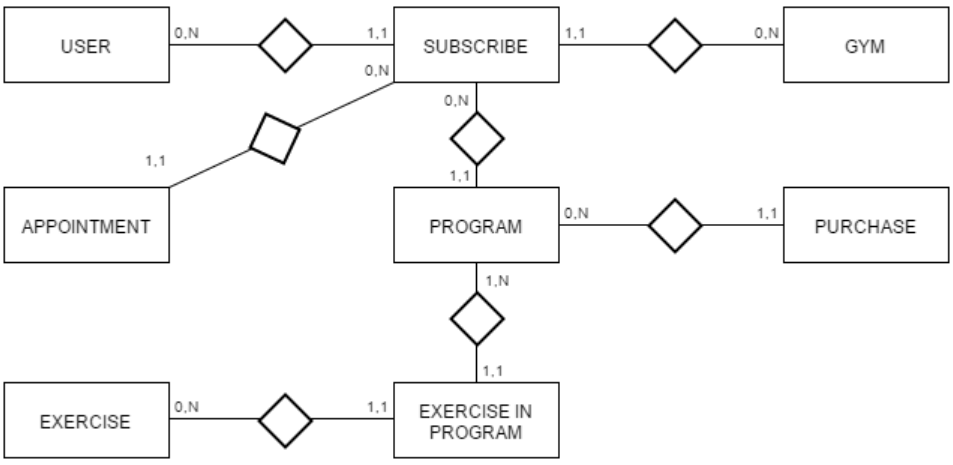

Figura 3.5: Database online semplificato

3.4.3 Database locale

Android fornisce diversi metodi e strumenti per salvare i dati in modo persistente. I dati in locale vengono salvati su un database *SQLite*, che a differenza degli altri database SQL è molto leggero ed occupa poco spazio su disco. E' una piattaforma perfetta per gestire database in un ambiente come quello degli applicativi mobile, dove le risorse sono limitate. SQLite scrive direttamente su file ordinari sul disco e possiede tutti gli strumenti principali, che caratterizzano i più importanti database.

Il database locale è composto da 4 tabelle:

- EXERCISE;
- PROGRAM;
- EXERCISE IN PROGRAM;
- PROGRESS;

Queste rispecchiano le tabelle che sono hostate nel Server, tranne per la tabella *Exercise* che non ha campi duplicati per entrambe le lingue, ma contiene valori di una sola lingua. Nella tabella *Progress* vengono memorizzati i record che indicano i progressi dell'utente per ogni esercizio effettuato. In quest'ultima tabella si salva l'esercizio, la data di esecuzione dell'esercizio e il peso sollevato.

3.4.4 Salvataggio con SharedPreferences

Un altro metodo per la persistenza dei dati offerto da Android è il salvataggio attraverso *SharedPreferences*. Questo metodo permette di salvare piccole quantità di dati che possono essere ad esempio una parola, oppure un qualsiasi valore. La gestione di salvataggio e di caricamento del dato viene effettuata totalmente da Android. Android ci lascia la possibilità di scegliere solamente la modalità di accesso ai dati; possiamo salvare i dati in modalità private ***(MODE PRIVATE)***, modalità pubblica ***(MODE WORLD)*** oppure altre modalità poco usate. Nell'App è stata usata esclusivamente la modalità privata, in modo da bloccare la lettura dei dati da altre applicazioni.

3.4.5 Memorizzazione su file

Per salvare oggetti leggermente più complessi è stato utilizzato anche il classico metodo di salvataggio su file. Android, con la versione KitKat, ha messo a disposizione dei metodi che fanno uso di *InputStream e OutputStream*

per il salvataggio e il caricamento dei dati. In questa applicazione vengono usati per salvare l'ultimo utente che ha utilizzato l'App, i progressi settimanali e mensili dell'utente per quanto riguarda la corsa e un immagine di profilo dell'utente.

3.5 Threading

Molte operazioni svolte dalle applicazioni sono presumibilmente *lente*, come ad esempio le operazioni di salvataggio viste precedentemente. Lavorando solamente con il *Thread* principale, che si occupa principalmente di gestire l'interfaccia utente, si rischierebbe di rendere poco reattiva la *UI*. Per rendere l'esperienza utente più gradevole e per impedire che si verifichino rallentamenti dell'App abbiamo bisogno di utilizzare più *Thread* per svolgere tutto il lavoro.

Tutte le chiamate *HTTP* e alcune operazioni di prelevamento dei dati dal database sono gestite da *Thread*, più specificatamente da *AsyncTask*. In questo modo il processo principale riesce a mantenere fluida e reattiva l'interfaccia utente.

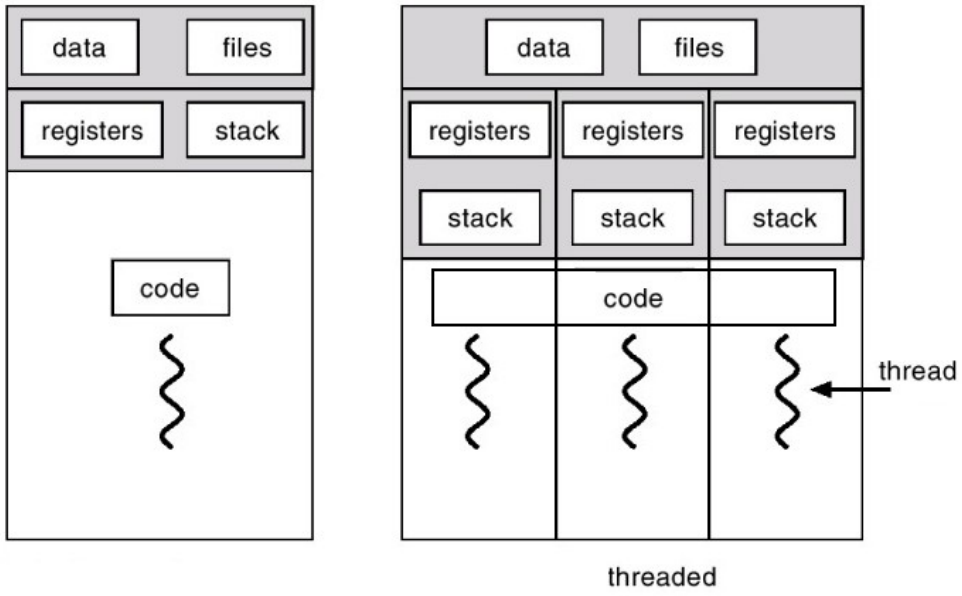

Figura 3.6: Rappresentazione di un processo leggero (Thread)

3.5.1 AsyncTask

Per sfruttare al meglio il *multi-threading* e per non gestire manualmente i Thread, Android mette a disposizione la classe *AsyncTask*, la quale fa largo uso dei generici. Questa classe ci semplifica molto il lavoro, grazie ai suoi metodi:

- **onPreExecute** inizializza le operazioni prima che avvenga l'esecuzione di doInBackground. Solitamente prepara *ProgressDialog* e le mostra all'utente;
- **doInBackground** è l'unico metodo che viene eseguito su un Thread secondario;
- **onProgressUpdate** serve a fornire aggiornamenti periodici all'interfaccia utente ed in molti casi a spostare la barra di progresso in avanti;
- **onPostExecute** viene eseguito alla fine di doInBackground ed anch'esso svolge operazioni collegate all'interfaccia utente, facendo scomparire la ProgressDialog;

La classe *AsyncTask*, come detto nel paragrafo precedente, prende in ingresso tre generici. Il primo indica il tipo dei parametri presi in ingresso, il secondo il tipo del valore restituito durante l'esecuzione del processo e il terzo è il tipo del risultato. Nella porzione di codice che segue viene mostrata una classe che estende AsyncTask.

```
public class RegisterAsyncTask extends AsyncTask<String, Void,
    String>{

    private IRegisterActivity activity;

    public RegisterAsyncTask(IRegisterActivity activity){
        super();
        this.activity = activity;
    }

   @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }

    @Override
    protected String doInBackground(String... params) {
```

```
        return null;
    }

   @Override
    protected void onProgressUpdate(Void... values) {
        super.onProgressUpdate(values);
    }

    @Override
    protected void onPostExecute(String s) {
        super.onPostExecute(s);
    }
}
```

3.6 Processi in background

Nella sezione precedente si è parlato degli AsyncTask, che permettono di avviare attività asincrone a supporto dell'interfaccia utente. I task gestiti non dovrebbero essere troppo lunghi (pochi secondi al massimo). Per lavori di lunga durata o addirittura indeterminata, si deve ricorrere ad un componente appropriato che ci offre Android, i *Service*.

Nell'applicazione sono stati usati due *Service*, uno dei quali gestisce il lettore multimediale, mentre l'altro è in ascolto per rilevare risposte alle richieste di appuntamento effettuate dall'utente. Il primo servizio è *startato* all'avvio dell'App, mentre il secondo è avviato nel momento in cui il *boot* dello smartphone è concluso. E' stato necessario utilizzare questi servizi, poiché le operazioni svolte devono essere effettuate di continuo anche quando il cellulare è bloccato. Per esempio l'utente deve avere la possibilità di ascoltare musica durante i suoi allenamenti avendo il telefono bloccato.

3.6.1 Service

Possiamo definire un *Service* come un'*Activity* senza interfaccia grafica. A differenza degli AsyncTask il loro funzionamento è sincrono. La chiamata per avviare un servizio è simile a quella per *startare* una nuova Activity, infatti basta chiamare il metodo *startService(Intent intent)* a cui viene passato un *Intent*. Le operazioni svolte dal servizio sono implementate all'interno del metodo *onStartCommand*. Solitamente i servizi servono per gestire download, oppure operazioni molto dispendiose in tempo e risorse.

3.6.2 Broadcast Receivers

I *Broadcast Receivers* sono uno dei componenti fondamentali nell'architettura di Android. Questi componenti sono costantemente in ascolto di determinati messaggi, chiamati *Intent Broadcast*. Gli *Intent Broadcast* sono particolari Intent spediti attraverso chiamate al metodo *sendBroadcast()*. Solitamente vengono utilizzati per notificare l'avvenuta di determinati eventi. A questi eventi, successivamente, corrisponderà una determinata azione.

Nell'applicazione, *Service* e *Broadcast Receiver* lavorano insieme, questo per aumentare la modularità del codice e la reattività dell'App. Ogni operazione effettuata sul lettore musicale viene notificata ad un certo Broadcast Receiver, che in base al messaggio ricevuto chiamerà un determinato metodo del servizio che si occupa della riproduzione dei brani musicali. Un altro Broadcast Receiver si occupa dell'avvio del servizio, che monitora l'arrivo di una risposta ad una richiesta di appuntamento. Quest'ultimo Broadcast viene notificato nel momento in cui il boot dello smartphone è concluso; al ricevimento del messaggio avvierà il servizio.

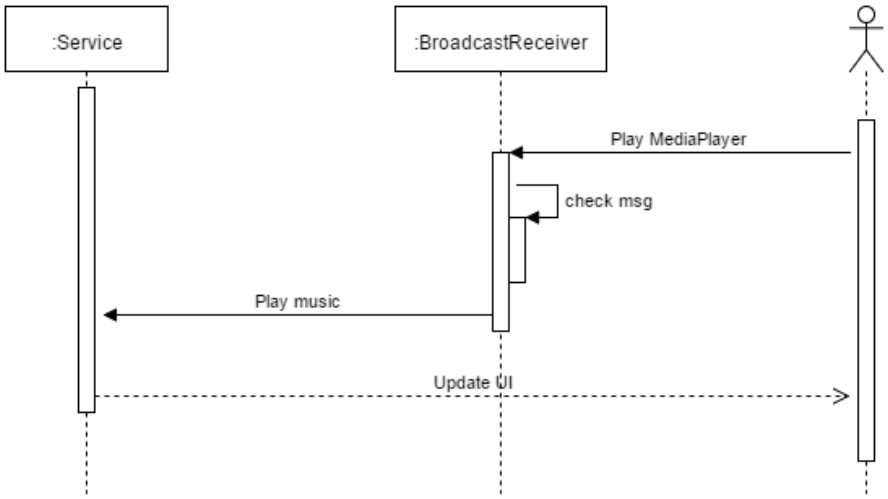

Figura 3.7: Diagramma di sequenza per interazione tra Service e Broadcast Receiver del MediaPlayer

3.7 GPS

Il *GPS* è di fondamentale importanza per gestire correttamente l'attività della *Corsa*, sia per permettere l'analisi dei dati sia per tracciare il percorso effettuato dall'utente durante la sua attività fisica. Il *sistema di posizionamento* dovrà necessariamente essere gestito da un *Service*, poiché il processo

di rilevamento deve rimanere sempre attivo in background. L'utilizzo di questo strumento permette di calcolare con precisione i metri percorsi e di conseguenza anche le calorie bruciate.

Utilizzando questo sistema il consumo di batteria del dispositivo aumenta notevolmente, ma grazie ad alcune API è possibile richiedere la propria posizione dopo un certo intervallo di tempo. Scegliendo un intervallo ragionevole si avrà un consumo ridotto della batteria e una buona precisione del rilevamento.

3.8 Google API

Questa applicazione fa largo utilizzo delle API di Google. Infatti queste sono usate per la mappa geografica mostrata durante l'attività della corsa, per il lettore video di *YouTube* la cui funzione è quella di mostrare il video dell'esecuzione degli esercizi fitness, per la gestione degli acquisti in App e per visualizzare pubblicità, chiamate *Ad*, all'interno dell'applicazione.

Per poter usufruire delle API di Google è necessario avere un account sviluppatore. Una volta ottenuto l'account sviluppatore è possibile abilitare, per una determinata applicazione, le API desiderate. Una volta abilitata una API ci viene fornita una *key* relativa alla nostra applicazione, che è indispensabile per l'utilizzo della funzionalità.

3.8.1 Acquisti in App

Nello *Shop* l'utente può acquistare gli allenamenti per il fitness. L'utente sceglie da una lista l'allenamento che vuole acquistare, successivamente effettuerà il pagamento attraverso **Google Wallet**. Google salva automaticamente gli acquisti effettuati, ma l'applicazione deve aggiornare il suo database online per gestire gli effetti di un acquisto di un prodotto. Nel caso di *FitBody* si deve aggiungere un record dove viene dichiarato che un utente ha acquistato un determinato prodotto, che è identificato da un ID.

Un problema, che potrebbe incorrere durante un acquisto, per esempio a causa di un bug, è la mancata *consegna* del prodotto dopo il pagamento. Per non incorrere in questo problema l'App, una volta andato a buon fine il pagamento, cercherà di effettuare l'aggiornamento del database online e se l'operazione non va a buon fine si salverà le informazioni dell'acquisto in locale. Successivamente ogni volta che verranno aggiornati i programmi di allenamento verrà effettuato un check per gli acquisti in sospeso; se ci dovesse essere qualche

acquisto non ancora evaso si effettuerà la chiamata *HTTP*, che aggiornerà il database online.

3.9 Facebook API

Le API di *Facebook* permettono all'utente di interagire con il proprio account *Social*. Nell'App è possibile effettuare il login tramite *Facebook* e condividere alcuni contenuti sulla propria bacheca, come i risultati ottenuti dopo un allenamento. Attraverso il login vengono recuperati alcuni dati dell'utente, come la sua e-mail e il suo *Facebook* ID. Questo Social permette di effettuare diversi tipi di condivisione; in quest'applicazione è stata utilizzata la condivisione tramite *Open Graph Stories*.

Open Graph Stories Le persone usano le *Open Graph Stories* per condividere le attività che svolgono durante la giornata, con chi le svolgono e il luogo in cui si trovano. Le *Open Graph* sono costituite da una *Action* e un *Object*.

La Action è la descrizione di un'azione attraverso un verbo, ad esempio *ha percorso*. Mentre l'Object è l'oggetto a cui si riferisce la Action, in questo caso *corsa*. L'oggetto è composto da diversi attributi, come la durata, il consumo energetico etc... Facebook tiene traccia di tutte le attività svolte in modo da mostrare all'utente la sua storia nella sua bacheca.

Figura 3.8: Post Open Graph di una corsa effettuata

3.10 Lato Server

Per gestire le operazioni su un database online abbiamo bisogno di un'applicazione lato Server, che si interesserà di rispondere alle chiamate *HTTP* e di effettuare le query sul database richieste dal *Client*. L'applicazione lato server sarà composta da file *PHP* hostate sempre sulla piattaforma di *Altervista*.

PHP è un linguaggio di programmazione interpretato, concepito per pagine web dinamiche. E' stato scelto di sviluppare l'applicazione server con PHP al posto di Java, poiché non necessita di un web server per essere eseguito. In più PHP mette a disposizione delle API che permettono di convertire automaticamente un array di oggetti in un altro array di oggetti JSON.

Capitolo 4

Progettazione

In questo capitolo vengono descritte le classi del progetto, motivando le scelte progettuali adottate.

4.1 Organizzazione in package

L'applicazione è stata suddivisa in vari *package*, raggruppando le classi in modo da avere una buona organizzazione e permettendo di muoversi più velocemente nell'applicazione per eventuali modifiche. Qui sotto è riportata un'analisi dell'organizzazione in package dell'applicativo:

- **adapters**: contiene i sorgenti dei vari *Adapters* delle componenti grafiche che li richiedono;
- **asynctask**: questo package contiene classi che estendono *AsyncTask*, che eseguiranno richieste *HTTP* e alcune operazioni sul database;
- **database**: all'interno del package *database* troviamo due classi che si occupano del salvataggio e del caricamento dei dati. Salvataggio e caricamento vengono effettuati sia da file che da database *SQLite*;
- **exceptions**: qui troviamo eccezioni, che potrebbero essere lanciate dall'applicazione;
- **exceptions.interfaces**: la sotto-cartella di *exceptions* contiene l'interfaccia che sarà implementata dalle eccezioni;
- **fragment**: contiene i sorgenti dei vari *Fragment* utilizzati all'interno dell'App;

- **it.deca.fitbody**: questo è il package principale dell'applicazione, contiene tutte le *Activity* dell'App;
- **it.deca.fitbody.interfaces**: qui sono contenute tutte le interfacce delle varie *Activity*;
- **model**: contiene i sorgenti delle classi che rappresentano gli oggetti principali usati dall'applicazione;
- **model.interfaces**: contiene i sorgenti delle interfacce implementate dalle classi contenute in model;
- **services**: contiene tutti i *Service* e *Broadcast Receiver* lanciati dall'App;
- **services.interfaces**: contiene le interfacce dei servizi dell'applicativo;
- **utility**: ha al suo interno due classi che permettono di fare determinate operazioni sulle stringhe;
- **utilpurchase**: in quest'ultimo package sono contenute le classi fornite da Google per effettuare e verificare gli acquisti in App.

4.2 Organizzazione delle risorse

Tutte le risorse di un'applicazione Android sono contenute nella cartella **res**, che a sua volta si divide in:

- **drawable**: qui sono contenute tutte le immagini nei formati *PNG* e *JPG* insieme ad elementi grafici personalizzati scritti utilizzando il linguaggio *XML*;
- **layout**: contiene tutti i *layout* delle varie schermate e degli oggetti mostrati all'interno delle *ListView*[1];
- **menu**: contiene file XML che definiscono la struttura dei menu visualizzati;
- **values**: in *values* troviamo tutte le stringhe visualizzate all'interno dell'App, colori personalizzati e temi.

[1]ListView è una componente grafica che permette di visualizzare un numero elevato di elementi che non riescono ad essere mostrati su una singola schermata

4.3 Classi e file

L'intero progetto si compone di 90 classi *Java* e 62 file *XML*. Tutto il programma segue l'architettura **MVC**, come detto nel capitolo precedente. Un pattern molto utilizzato è quello del *Singleton*, usato per esempio nelle classi *Session* e *Manager*, poiché si trattano di oggetti che devono essere istanziati una sola volta per sessione, facilmente reperibili grazie ai vari getter statici.

4.4 Descrizione delle classi principali

4.4.1 Model

Il model si basa su poche classi essenziali:

- **User**;
- **Gym**;
- **Session**;
- **Program**;
- **ExerciseInProgram**.

User La classe *User* modella un utente ed è caratterizzata da diversi campi:

- email;
- nome;
- sesso;
- peso;
- altezza;
- chilometri totali percorsi;
- calorie totali bruciate.

Questa classe implementa le interfacce *IUser* e *Serializable*. Quest'ultima viene usata per salvare l'oggetto user su file. La memorizzazione dell'utente permette di effettuare il login automatico all'avvio dell'applicazione.

Gym Questa classe modella una palestra a cui è iscritto l'utente. Per il momento è caratterizzata solamente da un nome, ma è stato scelto di modellarla come classe, poiché in futuro verranno aggiunti ulteriori dati che si riferiscono alle palestre. Queste classe, come tutte le altre, implementa l'interfaccia corrispondente: *IGym*.

Session La classe portante del progetto è *Session*, che si occupa di gestire i dati della sessione corrente. In questa vengono memorizzati l'utente corrente, la palestra, una *RunStory* ed una lista degli appuntamenti dell'utente.
RunStory è un'altra classe del progetto, che ha il compito di rappresentare le statistiche della corsa. Implementa l'interfaccia *IRunStory* e *Serializable*. Questa classe si occupa di tenere memorizzati i dati relativi all'ultima corsa effettuata, ai chilometri percorsi nella settimana corrente ed i chilometri e le calorie bruciate nell'ultimo mese.

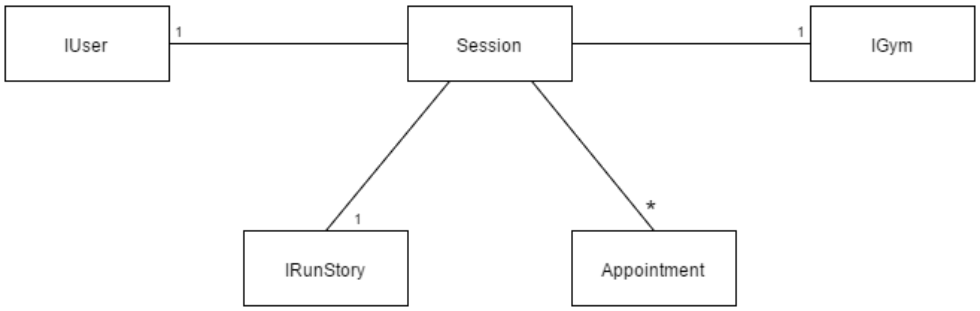

Figura 4.1: Diagramma delle classi semplificato che mostra una sessione

Program Questa classe modella un programma fitness ed è caratterizzato da un nome, un tipo ed una sezione. La sezione indica se questo è un programma creato dall'utente oppure se è stato inviato all'utente da *FitBody* o dalla sua palestra. Il tipo indica se si tratta di un programma oppure se è il titolo di ogni singola sezione. Questa distinzione è stata necessaria poiché l'oggetto *Program* è mostrato all'utente attraverso una *ListView*; in questo modo programmi e titoli sono rappresentati con layout differenti.

ExerciseInProgram Gli esercizi che compongono un programma sono rappresentati dalla classe *ExerciseInProgram*. In questa classe troviamo l'ID dell'esercizio, il nome, la sua posizione in lista, le ripetizioni, la durata, il tempo di recupero, il peso da utilizzare e il tipo di serie da effettuare.

Le ripetizioni degli esercizi sono gestiti attraverso una stringa composta nel seguente modo:

String repetitions = 0x 0x 0x 0x 0x 0x;

Ogni valore, separato da un doppio spazio, rappresenta il numero di ripetute della serie corrispondente. Quando il valore delle ripetute è uguale a 0, la serie non viene mostrata all'utente. Il tipo della serie da effettuare indica se il peso deve essere costante, crescente o decrescente durante lo svolgimento dell'esercizio.

4.5 Salvataggio dei dati

Le classi che si occupano del salvataggio e del caricamento dei dati sono due:

- DBFitBody;
- Manager;

DBFitBody La classe *DBFitBody* estende *SQLiteOpenHelper* e gestisce tutte le query che vengono effettuate nel database locale. Il database è caratterizzato da un nome ed una versione. La prima volta che si istanzia l'oggetto della classe verranno chiamate in automatico le query per la creazione delle tabelle necessarie al funzionamento dell'applicazione. La classe contiene numerosi metodi che andranno a gestire il salvataggio, il prelevamento e l'aggiornamento dei dati. I metodi più usati o i più importanti sono:

- addProgram(email, name, gym): questo metodo aggiunge al database i programmi degli utenti;
- getPrograms(): restituisce tutti i programmi dell'utente corrente;
- addExercise(...): aggiunge alla tabella *Exercise* tutti gli esercizi prelevati dal database online;
- getExerciseInProgram(email, gym, name): restituisce tutti gli esercizi che compongono il programma selezionato.

Manager La classe *Manager* si occupa del salvataggio e del caricamento dei file. Molti dati vengono salvati su file, poiché è un metodo più veloce per salvare o reperire piccole quantità di byte. La maggior parte dei dati che sono gestiti da questa classe vengono caricati all'avvio dell'applicazione, come ad esempio l'User e la sua RunStory. Oltre a questi due oggetti vengono salvati in memoria l'immagine profilo nel formato *Bitmap*, gli appuntamenti

dell'utente e tutti gli acquisti effettuati dallo stesso che sono rimasti in sospeso.

Per il caricamento e il salvataggio dei dati sono stati utilizzati i metodi forniti da Android, dato che facilitano lo sviluppatore nel compito. Questi metodi sono:

- **openFileOutput** per il salvataggio dei dati;
- **openFileInput** per il caricamento dei dati.

4.6 Le Activity principali

Tutta l'applicazione è composta da 16 *Activity*, che controllano e triggherano gli eventi generati dall'interfaccia grafica. Queste Activity sono:

- **AppointmentActivity** permette all'utente di visualizzare o richiedere nuovi appuntamenti alla palestra;
- **BaseActivity** è un'Activity di base che viene estesa da altre Activity. Al suo interno troviamo dei metodi per il controllo della presenza/assenza della connessione ad internet;
- **CompleteRegistrationActivity** consente all'utente di terminare la propria registrazione inserendo nome, cognome, sesso, altezza e peso;
- **CreateEditProgramActivity** questa Activity permette di navigare all'interno di 3 Fragment [2] che consentono di aggiungere, modificare, eliminare esercizi dal programma di fitness;
- **DetailExerciseActivity** in questa Activity l'utente può vedere il dettaglio dell'esercizio completo di tutte le descrizioni. Inoltre in questo *Controller* troviamo il lettore di *YouTube* che mostrerà all'utente il video dell'esecuzione dell'esercizio selezionato;
- **FitnessActivity** si occupa di gestire la schermata di allenamento. Qui l'utente può seguire gli esercizi contenuti nel programma selezionato, spuntare gli esercizi conclusi, controllare il tempo di esecuzione e le calorie bruciate durante l'allenamento. A fine allenamento l'utente può condividere i suoi risultati sul proprio profilo Facebook;

[2] I Fragment sono sostanzialmente dei pannelli che hanno un proprio layout. Un'Activity può contenere più Fragment e può mostrarli uno alla volta, oppure più alla volta in porzioni di schermo differenti.

- **LoginActivity** permette all'utente di loggare all'applicazione tramite e-mail e password, oppure attraverso il login di Facebook. Se l'utente non possiede un account può accedere alla schermata di registrazione;
- **MainActivity** è l'Activity centrale dell'App. Da qui si può navigare, attraverso un menu, in tutte le sezioni dell'applicazione;
- **MaximalActivity** consente all'utente di registrare o calcolare i propri massimali per ogni esercizio;
- **ProgramActivity** mostra all'utente tutti i programmi. Da qui è possibile accedere alla schermata per modificare i programmi o crearne di nuovi;
- **PurchaseActivity** permette all'utente di fare acquisti in App. In particolare l'utente può acquistare allenamenti per il fitness;
- **RegisterActivity** permette all'utente di creare un nuovo account, inserendo nella form i dati richiesti;
- **RunningActivity** gestisce la corsa dell'utente e consente, a fine allenamento, di condividere i risultati ottenuti;
- **SettingActivity** permette all'utente di personalizzare alcuni parametri dell'App. Da qui è possibile aggiungere una foto del profilo, iscriversi ad una palestra ed effettuare il logout.
- **StatsActivity** mostra all'utente i suoi progressi riguardo gli allenamenti del fitness;
- **StretchingActivity** mostra all'utente tutti gli esercizi che possono essere eseguiti per lo stretching.

Le Activity principali sono *LoginActivity*, *MainActivity*, *CreateEditProgramActivity*, *FitnessActivity* e *RunningActivity*.

LoginActivity Questa Activity estende da BaseActivity e implementa l'interfaccia *ILoginActivity*. In questa schermata l'utente può effettuare il login oppure accedere alla pagina di registrazione. La schermata è formata da due *EditText* [3], nel primo deve essere inserita l'e-mail dell'utente, mentre il secondo prende in ingresso la password dell'utente. Sotto ai TextField ci sono due bottoni. Cliccando il primo bottone si avvia la procedura di accesso all'App,

[3]Gli EditText sono dei *text field* dove è possibile inserire del testo.

quindi vengono controllati i parametri inseriti all'interno degli EditText e se tutto va a buon fine si accede al menu principale dell'applicazione. Il secondo bottone invece avvia il processo di Login attraverso le API di Facebook.

Le password degli utenti vengono criptate durante la registrazione, quindi non sono mai visibili in chiaro all'amministratore del database. La libreria utilizzata per criptare le password si chiama **Jasypt**.

Procedura di accesso Una volta riempiti gli EditText l'utente potrà premete il pulsante di Login. A questo punto viene creato un **LoginAsyncTask** che prende in ingresso una *ILoginActivity*. Successivamente vengono passati i dati inseriti dall'utente e viene avviato l'AsyncTask, che effettuerà una richiesta *HTTP* al server e attenderà una sua risposta. Una volta ricevuta la risposta verrà chiamato un metodo dell'interfaccia *ILoginActivity* che effettuerà delle operazioni in base al risultato ricevuto.

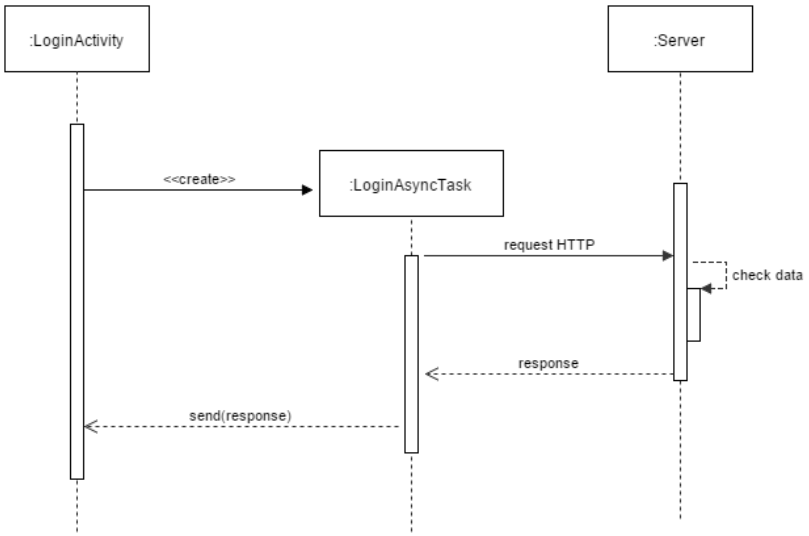

Figura 4.2: Diagramma di sequenza della procedura di accesso

MainActivity La MainActivity è il cuore dell'applicazione. Questa classe estende *MaterialNavigationDrawer*, cioè un'Activity che ha un menu a scomparsa chiamato *NavigationDrawer*. Attraverso questo menu è possibile accedere alla sezione Fitness, Running, Schedule, Shop, Feeding e Setting. La schermata di ogni sezione consiste in un Fragment. In base alla sezione selezionata viene visualizzato un Fragment al posto di un altro. All'avvio di questa Activity viene controllata l'esistenza del database. Se quest'ultimo non

dovesse esistere vengono create tutte le tabelle e vengono scaricati dal database online tutti gli esercizi per il fitness.

Procedura di download L'operazione di download è simile alla procedura di login. Infatti viene avviato un AsyncTask che effettua una richiesta HTTP per scaricare tutti gli esercizi. La risposta positiva del server consiste in un array di oggetti JSON. Una volta ricevuto questo array viene effettuato un parsing per salvare gli esercizi nel database locale.

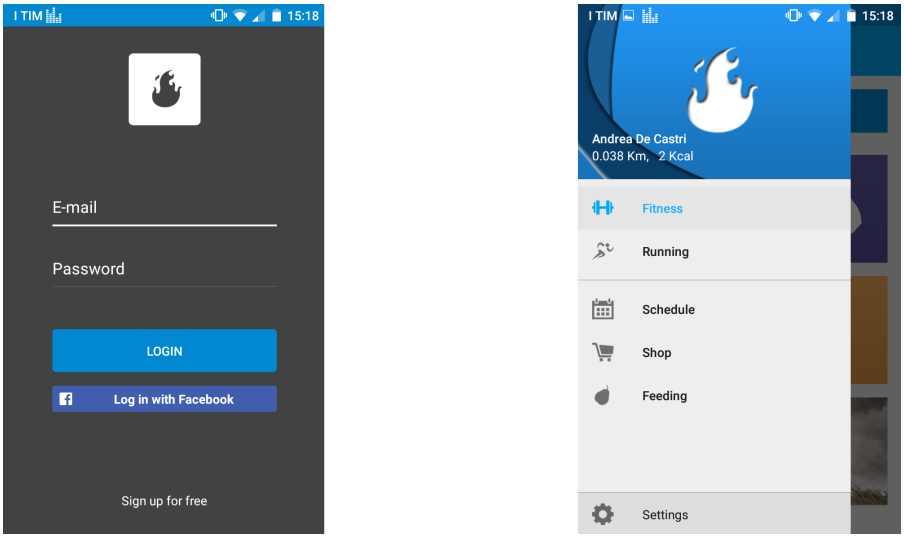

Figura 4.3: Schermata del Login e del menu principale

CreateEditProgramActivity L'Activity ha al suo interno un *ViewPager*, composto da tre *Fragment*, selezionabili dall'utente tramite swipe verso destro o verso sinistra. Questi Fragment sono:

- AddExerciseFragment;
- EditExerciseFragment;
- PreviewProgramFragment.

AddExerciseFragment In questo Fragment l'utente può visualizzare il dettaglio degli esercizi e aggiungerli al suo programma. La schermata è composta da uno *Spinner*[4], da un'*EditText* e da una *ListView*. Lo *Spinner* contiene

[4]Lo Spinner è una componente grafica di Android che mostra una lista di elementi in una tendina scorrevole.

le varie parti del corpo; selezionando un valore, vengono mostrati solo gli esercizi che allenano quella specifica parte del corpo. Inoltre, gli esercizi possono essere filtrati per il nome con l'inserimento del testo all'interno dell'*EditText*. La *ListView* mostra all'utente gli esercizi richiesti.

Ogni elemento visualizzato dalla lista è composto da un nome e da un bottone. Cliccando su un *item* della lista, l'utente accede alla schermata di dettaglio dell'esercizio, mentre facendo click sul bottone *Aggiungi*, l'esercizio selezionato viene aggiunto al programma.

EditExerciseFragment Qui l'utente può personalizzare ogni esercizio aggiunto alla scheda, settando i valori di ogni singola serie, il tempo di recupero, la durata dell'esercizio e il tipo della serie da eseguire. Anche questa Activity mostra ogni esercizio attraverso una *ListView*, ma ogni elemento della lista è composto da 9 *Spinner*.

PreviewProgramFragment Nell'ultimo Fragment l'utente vede un'anteprima del programma che ha creato, sempre attraverso una *ListView*. L'utente, attraverso un click prolungato su un esercizio, visualizzerà una lista di comandi che può effettuare, ad esempio l'eliminazione dell'esercizio oppure la modifica. Nel caso in cui l'utente decida di modificare l'esercizio, viene effettuato un cambio automatico del Fragment e viene posizionato come primo elemento l'esercizio che l'utente vuole modificare. Con lo swipe dal Fragment *EditExercise* si abiliterà il pulsante salva, che permette di salvare nel database il programma creato.

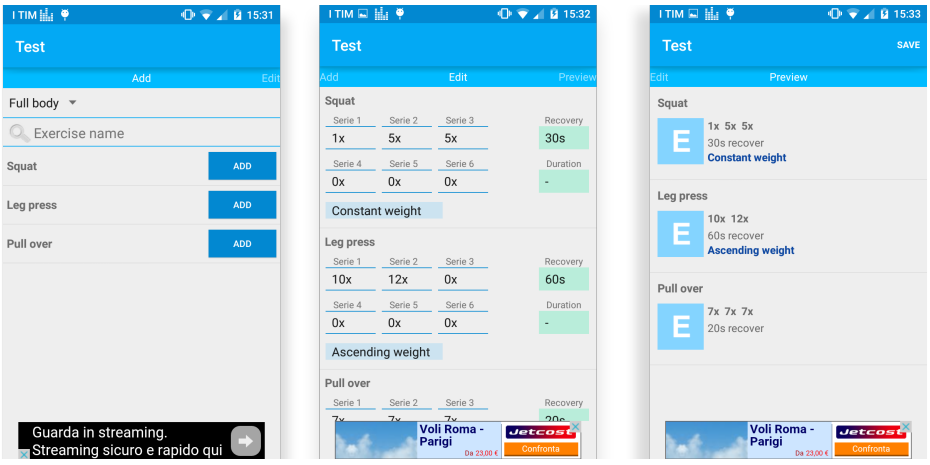

Figura 4.4: Schermate per la creazione e modifica di un programma di allenamento

Comunicazione tra Activity e Fragment I *Fragment* non possono comunicare direttamente tra loro, ma devono avere un intermediario. Questo intermediario è proprio l'*Activity* che hanno in comune.

I Fragment implementano il metodo di libreria **onAttach**, che restituisce l'Activity a cui è agganciato il Fragment. Per una buona programmazione, l'Activity in discussione implementa l'interfaccia **ICreateEditProgramActivity**, in questo modo nei Fragment, l'Activity, passata come parametro, verrà castata al tipo dell'interfaccia, in modo da rendere accessibili pubblicamente soltanto i metodi dell'interfaccia.

L'*ArrayList* di *ExerciseInProgram* sarà istanziata solamente nell'Activity principale e tutti i Fragment avranno accesso, attraverso l'interfaccia, alla lista. I metodi di aggiunta, eliminazione e salvataggio sono stati implementati tutti nell'Activity *CreateEditProgramActivity*.

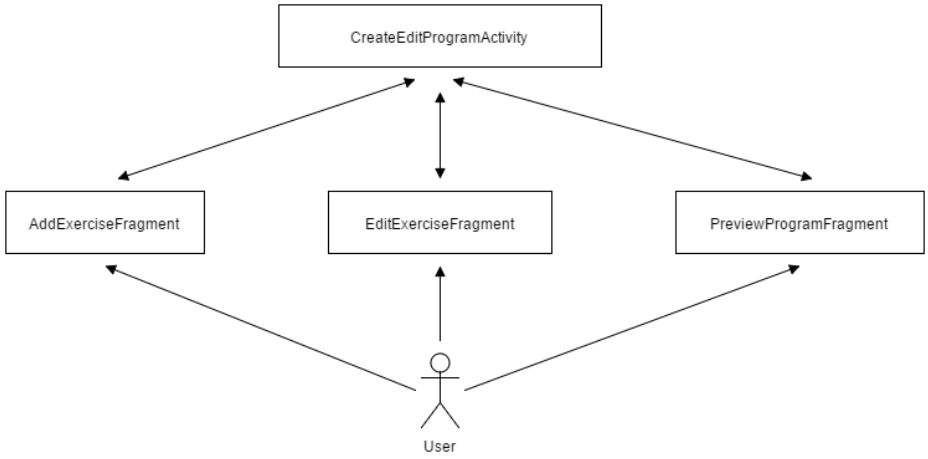

Figura 4.5: Comunicazione tra Fragment e Activity

FitnessActivity La *FitnessActivity* si presenta con un layout più articolato rispetto alle Activity precedenti. Nella parte superiore vengono indicate le calorie bruciate durante l'allenamento, gli esercizi mancanti e viene visualizzato il cronometro, avviabile tramite il pulsante presente nel menu in alto a destra. Subito sotto queste informazioni viene visualizzata la lista degli esercizi che compongono il programma.

Per ogni esercizio viene visualizzata un'immagine esplicativa, oltre alle varie informazioni descritte precedentemente. In sovra-impressione all'immagine è posizionato un quadratino di colore rosso, che sta ad indicare la mancata

esecuzione dell'esercizio. Tenendo premuta a lungo l'icona dell'immagine, il quadratino passa dal colore rosso al colore verde, inoltre viene effettuato un calcolo per le calorie bruciate durante l'esecuzione dell'esercizio. Facendo un click prolungato sul singolo esercizio, l'utente potrà aggiornare il peso dell'esercizio, invece cliccando semplicemente un esercizio si avvierà l'Activity per la visualizzazione del dettaglio dell'esercizio.

Quando il contatore degli esercizi mancanti arriverà a zero l'allenamento è considerato concluso e viene mostrato all'utente il risultato ottenuto. A questo punto l'utente può decidere se condividere sulla propria bacheca Facebook il risultato conseguito oppure chiudere l'Activity corrente. La condivisione è abilitata solo per gli utenti che hanno l'account Facebook collegato all'applicazione. Per associare un account Facebook basta andare nelle impostazioni dell'App.

RunningActivity L'Activity che gestisce l'attività della corsa è la *RunningActivity*. Nella parte superiore vengono mostrate le informazioni principali che gli utenti controllano durante una corsa, cioè le calorie bruciate, il tempo trascorso e la distanza percorsa. Subito sotto abbiamo una mappa che mostra all'utente il percorso che sta effettuando. Il percorso effettuato viene indicato sulla mappa con una traccia di colore blu.

La posizione del corridore viene ottenuta attraverso il *GPS* ogni 4 secondi, ma le statistiche vengono aggiornate solo se l'utente ha percorso almeno 10 metri dal rilevamento precedente. Per terminare l'attività, l'utente deve prima sbloccare il pulsante di stop tenendo premuto il bottone *Hold*; una volta sbloccato ha 3 secondi per premere il pulsante che terminerà l'attività, altrimenti dovrà ripetere l'operazione. Conclusa anche questa attività viene chiesto all'utente se vuole condividere l'attività su Facebook.

Scelta dell'attività e attivazione del GPS All'avvio di questa Activity viene controllato lo stato del *GPS*, qualora non fosse attivo, viene chiesto all'utente di attivarlo. Non attivando il GPS, sarà impossibile calcolare la distanza, il percorso e le calorie bruciate.

Una volta attivato il GPS, viene mostrato all'utente un menu *BottomSheet*[5], che dà l'opportunità all'utente di scegliere diverse attività, quali:

- Corsa;
- Camminata;
- Ciclismo.

In base all'attività selezionata verrà effettuato un calcolo diverso per il consumo calorico.

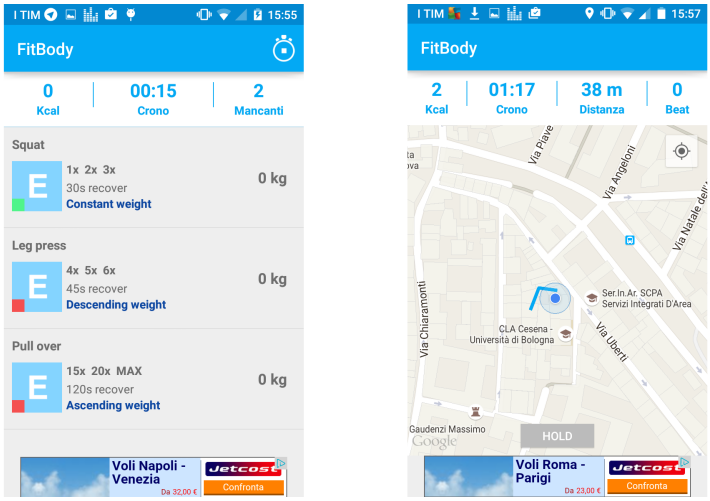

Figura 4.6: Schermate di allenamento fitness e corsa

Altre Activity Un'altra Activity di cui abbiamo parlato è la *SettingActivity*. Il suo *layout* è costituito da una *ScrollView*. La *ScrollView* è un layout che può contenere solamente un figlio. Questo permette di mostrare informazioni che superano la lunghezza dello schermo. Qui l'utente può cambiare la propria immagine di profilo, scegliendola tra le immagini salvate in galleria, vedere i suoi dati, configurare le unità di misura attraverso due *Spinner*, collegare o scollegare il proprio account Facebook, aggiungere una palestra tramite un *EditText* ed effettuare il logout dall'applicazione.

Un'altra Activity con un layout particolare è la *DetailExerciseActivity*, composto da uno *SlidingUpPanelLayout*, che contiene all'interno due layout figli.

[5]Il BottomSheet è un menu a scomparsa verso il basso, dove è possibile selezionare diverse opzioni.

Il primo mostra il contenuto principale della schermata, in questo caso la descrizione completa dell'esercizio e l'immagine, mentre il secondo contiene il lettore *YouTube* per visualizzare l'esecuzione dell'esercizio. Inizialmente il secondo layout è nascosto, ma può essere mostrato cliccando sull'icona della telecamera posizionata sopra l'immagine dell'esercizio.

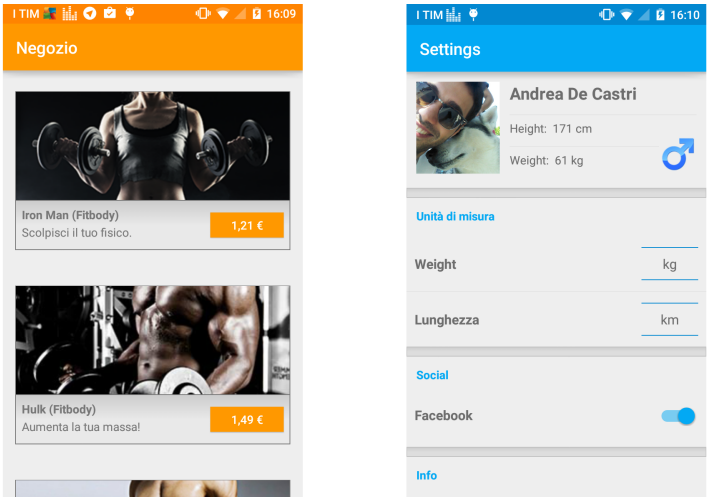

Figura 4.7: Schermata per l'acquisto di allenamenti e menu delle impostazioni

4.7 Servizi

I servizi sono implementati dalle classi **MusicService** e **AppointmentService**. Il primo lavora in background solamente quando l'App è attiva, mentre l'altro è sempre attivo dall'avvio dello smartphone.

MusicService Nel costruttore di questo servizio viene creata un'istanza del **MediaPlayer**, poiché il servizio si occupa dell'esecuzione dei brani che l'utente ascolta durante i suoi allenamenti. Una volta creato il *player* vengono cercati tutti i brani musicali con estensione *.mp3* presenti all'interno del dispositivo. Quando viene trovato un file con la giusta estensione, viene creato un oggetto **Song** e viene aggiunto ad un *ArrayList*. L'oggetto *Song* contiene le informazioni necessarie per il recupero del brano all'interno del dispositivo e il titolo della canzone. Infine viene registrato un *BroadcastReceiver* che gestirà gli input inviati dall'utente.

L'interfaccia del servizio mette a disposizione tutte le operazioni che possono essere effettuate con un *MediaPlayer*, cioè avviare o mettere in pausa una

canzone, scegliere il brano successivo o il brano precedente. Lo scorrimento delle canzoni viene gestito attraverso l'utilizzo di un indice.

MediaPlayer Il player offerto da Android demanda la gestione totalmente allo sviluppatore, che dovrà occuparsi del caricamento dei brani e di tutte le funzionalità di base di un MediaPlayer. I principali metodi messi a disposizione dal player sono:

- **reset**: deve essere chiamato ad ogni caricamento di una canzone nel player. Questo rilascia le risorse e pulisce il buffer del player;
- **setDataSource**: questo metodo permette di settare il brano da eseguire, prendendo in ingresso un oggetto di tipo *Uri*, che indicherà in percorso del file all'interno del dispositivo;
- **prepare**: caricherà il file della canzone all'interno di un buffer;
- **release**: questo metodo deve essere chiamato al momento della chiusura del player. Questo effettuerà un rilascio di tutte le risorse utilizzate dal riproduttore musicale.

Notifica del MediaPlayer Le operazioni che possono essere effettuate sul player sono mostrate all'interno di una notifica lanciata all'avvio dell'applicazione. Con la versione 5.0 di Android è possibile settare un tema per le notifiche. Questa notifica ha come tema quello del MediaPlayer, che mostrerà i tre pulsanti necessari per avviare, stoppare o cambiare canzone. Inoltre, vengono mostrate le informazioni della canzone corrente e un'icona del player.

Quando l'utente cliccherà su uno dei pulsanti mostrati, la notifica invierà un *PendingIntent*[6] al *MusicReceiver*, che, in base al bottone premuto, chiamerà un determinato metodo dell'interfaccia del *MusicService*.

AppointmentService Questo servizio si occupa di effettuare periodicamente delle chiamate *HTTP* al server, per verificare se l'utente ha ricevuto una risposta ad una richiesta di appuntamento. La richiesta viene effettuata con un periodo di 5 minuti. Nel momento in cui l'utente riceverà una risposta, verrà effettuato un download dell'appuntamento. Il server invierà un oggetto JSON che verrà parsato e trasformato in un oggetto di tipo *Appointment*. L'appuntamento viene automaticamente salvato in memoria e successivamente viene inviata una notifica.

[6]Il PendingIntent è un particolare tipo di Intent.

Questa notifica, a differenza della precedente, non ha un tema settato. Mostra solamente l'icona dell'applicazione e il messaggio che indicherà all'utente di aver ricevuto una risposta dalla palestra. Cliccando sulla notifica si accederà direttamente alla schermata che mostra tutti gli appuntamenti.

4.7.1 Gestione delle notifiche da parte dei Servizi

Le notifiche in Android sono create partendo da un *NotificationBuilder*. Il NotificationBuilder setterà tutte le informazioni della notifica, come il titolo, il contenuto testuale, l'icona e l'azione che deve essere eseguita al click della notifica. Oltre a queste informazioni indispensabili, è possibile indicare al *Builder* se la notifica può essere cancellata dall'utente oppure essere cancellata solo dal sistema.

Una volta settati tutti i valori, è possibile creare la notifica chiamando il metodo **build** sull'oggetto Builder. La comparsa delle notifiche e la loro cancellazione viene gestita dal **NotificationManager**, che è un servizio messo a disposizione da Android. Infine per lanciare una notifica basta chiamare il metodo **notify**, che prende in ingresso l'ID della notifica e la notifica creata.

4.8 Eccezioni utilizzate

Le eccezioni, che possono essere lanciate dall'applicazione, estendono la classe **CustomException** che, a sua volta, implementa l'interfaccia **IException**. Il metodo implementato da tutte le eccezioni è *getErrorMessage*, che ritorna un intero. Questo numero intero rappresenta l'ID della stringa da visualizzare a schermo. In questo modo è possibile gestire facilmente la traduzione dei messaggi delle eccezioni, poiché ad un singolo ID sono associate stesse stringhe, ma di lingue differenti.

4.9 Componenti grafiche utilizzate

Sono state utilizzate una moltitudine di componenti grafiche indispensabili per il funzionamento dell'applicazione. Con il loro utilizzo, l'organizzazione delle varie schermate ha reso l'interfaccia grafica *user-friendly*. Le componenti utilizzate sono:

- **ListView** sono state utilizzate per visualizzare molti elementi in una lista verticale scorrevole. Ogni *ListView* ha associato un *Adapter*, che ha il compito di convertire un *ArrayList* di oggetti in una lista di *View*;

- **EditText** usati per l'inserimento di un testo da parte dell'utente. Il contenuto accettato dall'EditText può essere definito attraverso l'attributo **inputType**; i valori utilizzati per l'attributo sono: *number*, *emailAddress* e *text*. In base al tipo di valore scelto viene visualizzato un layout della tastiera diverso.

- **Button** vengono utilizzati per scatenare eventi alla loro pressione. Ad ogni bottone è stato associato un *selector*, ovvero un file XML che mostra un determinato background del bottone in base al suo stato: rilasciato, premuto.

- **TextView** sono semplicemente delle stringhe visualizzate a schermo;

- **Spinner** sono menu a tendina utilizzate per scegliere un opzione all'interno di una lista. Usata ad esempio per la scelta del sesso alla registrazione.

- **ImageView** mostrano all'utente le immagini, ad esempio le immagini degli esercizi.

- **Switch** è un vero e proprio interruttore che serve per attivare o disabilitare qualche opzione. Ad esempio viene utilizzato per l'associazione di un account Facebook.

- **Chronometer** mostra a schermo un cronometro. Usato nella schermata per gli allenamenti di fitness per calcolare il tempo di recupero.

- **AlertDialog** sono delle piccole finestre che appaiono in primo piano per mostrare all'utente un messaggio importante.

- **ProgressDialog** sono simili alle AlertDialog, ma non sono cancellabili attraverso la pressione di qualche bottone. Mostrano soltanto la barra di caricamento per indicare all'utente che lo smartphone è in fase di caricamento. Viene utilizzato durante il download dei contenuti necessari all'applicazione.

- **Toast** sono dei piccoli messaggi che vengono visualizzati all'utente per una breve durata di tempo.

4.9.1 Adapters

Ogni volta che è necessario visualizzare una lista scorrevole di elementi, ad esempio una *ListView*, dobbiamo popolare questa lista attraverso un *Adapter*. Un Adapter prende in ingresso un *ArrayList* e lo converte in oggetti da visualizzare contenuti nella lista scorrevole.

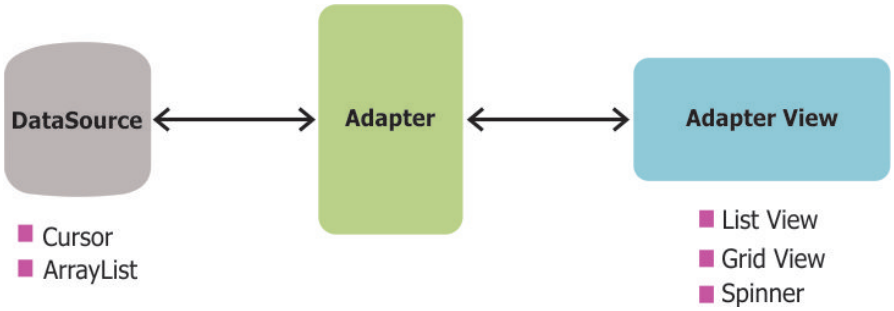

Figura 4.8: Trasformazione di una lista in una lista visibile di oggetti

Nell'applicazione tutte le classi contenute nel package *adapters* estendono la classe **BaseAdapter** e nel loro costruttore prendono in ingresso una lista di elementi e un *Context*. I metodi che devono essere implementati sono:

- **getCount** che ritorna il numero di elementi presenti nella lista;
- **getItem(int position)** che restituisce l'oggetto della lista in una determinata posizione;
- **getItemId(int position)** ritorna l'ID di un oggetto in una determinata posizione;
- **getView(...)** ritorna la rappresentazione grafica di un oggetto della lista. In questo metodo viene settato il layout dell'oggetto e vengono settati tutti i suoi valori.

Gli *Adapter*, oltre ad occuparsi della visualizzazione degli oggetti, sono indispensabili per migliorare l'utilizzo della memoria. Infatti, quando si crea un Adapter, viene riservata la memoria per un certo numero di elementi grafici, che basta a riempire lo schermo e per avere un certo margine di errore. In questo modo, anche se la lista da visualizzare supera le 1000 unità, in memoria saranno tenuti solamente quei pochi elementi grafici, che saranno riciclati con lo scorrere della lista. Per capire meglio basta dare un'occhiata all'immagine che segue.

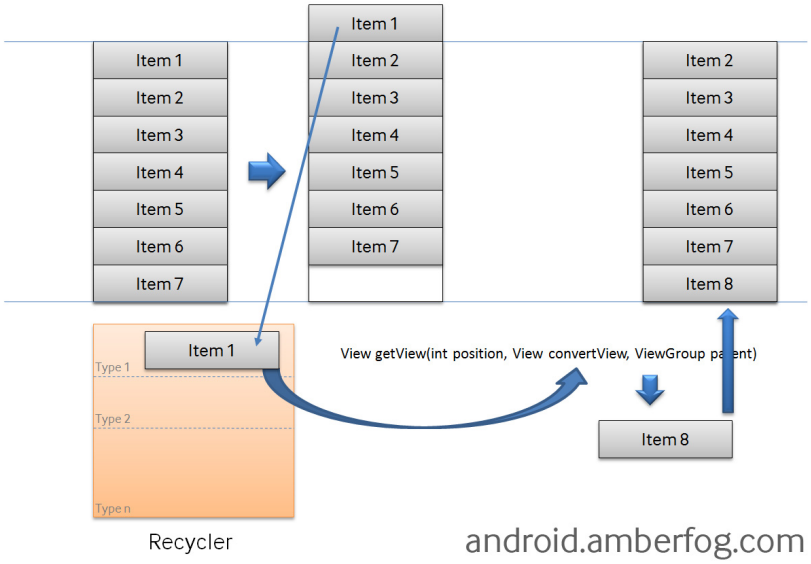

Figura 4.9: Riciclo delle celle utilizzate in una ListView

4.10 Layout utilizzati e la loro composizione

Nell'intero progetto sono stati utilizzati principalmente due tipi di layout: **RelativeLayout**, **LinearLayout**. I *LinearLayout* sono utilizzati per posizionare elementi uno accanto all'altro oppure uno sotto l'altro. Il posizionamento nel Linear viene dichiarato attraverso l'attributo *orientation* che può avere come valore *vertical* oppure *horizontal*. Per avere più libertà nel posizionamento delle componenti grafiche all'interno di un layout è necessario utilizzare il *RelativeLayout*, che permette di posizionare elementi rispetto ad altri. Ad esempio, si può posizionare una *ListView* sotto una determinata *EditText* e così via.

Capitolo 5

Implementazione

In questo capitolo verranno mostrate alcune implementazioni delle classi più importanti.

5.1 Uso degli AsyncTask

L'uso degli AsyncTask nell'App è molto frequente; questo la rende reattiva agli input dell'utente. Di seguito viene riportata l'implementazione del metodo che viene eseguito da un Thread secondario, in modo da non bloccare il Thread principale che si occupa della *GUI*.

AsyncTask per il Login Nel costruttore dell'AsyncTask viene passata l'interfaccia implementata dall'Activity. Successivamente viene passato come parametro l'e-mail dell'utente. Nel metodo *doInBackground* viene creata una lista di coppie nome-valore, successivamente viene creato un *Client HTTP*, che invierà una richiesta *POST* al server tramite un *URL* passato precedentemente. La risposta del server viene bufferizzata e caricata su una stringa. Al completamento dell'esecuzione viene chiamato, attraverso l'interfaccia, il metodo dell'Activity **onCompleteLogin** a cui viene passato il risultato in formato JSON, che sarà parsato dall'Activity. L'oggetto JSON descrive tutte le informazioni dell'utente, compresa la sua password criptata. Prima di proseguire, viene effettuato un controllo per verificare la corrispondenza della password inserita con la password criptata. Il controllo della password viene effettuato solo ed esclusivamente nel caso in cui l'utente non abbia effettuato il login tramite Facebook.

```
@Override
protected String doInBackground(String... params) {
    List<NameValuePair> list = new ArrayList<>();
    list.add(new BasicNameValuePair(FIELD_EMAIL, params[0]));

    DefaultHttpClient httpClient = new DefaultHttpClient();
    HttpPost httpPost = new HttpPost(URL);
    try {
        httpPost.setEntity(new UrlEncodedFormEntity(list));
        HttpResponse httpResponse = httpClient.execute(httpPost);
        BufferedReader reader = new BufferedReader(new
            InputStreamReader(httpResponse.getEntity()
                                                    .getContent()));

        StringBuilder sb = new StringBuilder();
        String content;
        while ((content = reader.readLine()) != null) {
            sb.append(content);
        }

        return sb.toString();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}

@Override
protected void onPostExecute(String result) {
    super.onPostExecute(result);
    this.activity.onCompletedLogin(result, false);
}
```

5.2 Condivisione tramite Open Graph

Per poter condividere un post attraverso Open Graph si deve creare un oggetto di tipo *ShareOpenGraphObject*, a cui vengono settate coppie nome-valore: ad esempio, vengono settati i valori della distanza percorsa, il tempo impiegato e la velocità media mantenuta. Successivamente si deve creare una *Action* attraverso l'oggetto di tipo *ShareOpenGraphAction*, a cui viene passato

lo *ShareOpenGraphObject*, creato precedentemente, e l'ID dell'azione a cui è associato. Dopo di che si crea un oggetto di tipo *ShareOpenGraphContent*, a cui verrà passata l'azione creata in precedenza. Infine viene mostrata una *ShareDialog* che prende in ingresso il contenuto appena creato.

```
ShareOpenGraphObject object = new ShareOpenGraphObject.Builder()
                .putString("og:type", "fitness.course")
                .putString("og:title", "Corsa con FitBody")
                .putString("og:description", "Questa corsa mi ha
                   distrutto!")
                .putInt("fitness:duration:value", seconds)
                .putString("fitness:duration:units", "s")
                .putDouble("fitness:distance:value", totalMeters /
                   1000)
                .putString("fitness:distance:units", "km")
                .putInt("fitness:calories", (int) totalCal)
                .putDouble("fitness:speed:value", totalMeters/seconds)
                .putString("fitness:speed:units", "m/s")
                .build();
        ShareOpenGraphAction action = new
            ShareOpenGraphAction.Builder()
                .setActionType(getActionType())
                .putObject("fitness:course", object)
                .build();
        ShareOpenGraphContent shareContent = new
            ShareOpenGraphContent.Builder()
                .setPreviewPropertyName("fitness:course")
                .setAction(action)
                .build();
       ShareDialog dialog = new ShareDialog(this);
      dialog.show(shareContent);
```

5.3 Check degli appuntamenti e lancio di una notifica

Il servizio che si occupa degli appuntamenti, al suo avvio, lancia un Thread che controllerà periodicamente la presenza di risposte. Quando è presente una nuova risposta viene effettuato il download dell'appuntamento. L'appuntamento viene immediatamente salvato in memoria e successivamente viene inviata una notifica all'utente. Il download dell'appuntamento avviene attra-

verso una richiesta *HTTP* analoga a quella usata per effettuare il login. Di seguito viene mostrato il codice che serve per creare e lanciare una notifica sullo smartphone, con l'attivazione della vibrazione e del LED.

```
private void notifyUser(){
     Bitmap icon = BitmapFactory.decodeResource(getResources(),
                          R.drawable.icon);

       Notification.Builder builder = new Notification.Builder(this);
       builder.setContentTitle("FitBody");
       builder.setSmallIcon(R.drawable.icon_schedule);
       builder.setLargeIcon(icon);
       builder.setVibrate(new long[] { 1000, 500 });
       builder.setLights(Color.WHITE, 3000, 3000);
       builder.setContentText("Hai ricevuto una risposta dalla tua
           palestra");
       builder.setAutoCancel(true);

       NotificationManager manager = (NotificationManager)
             getSystemService(Context.NOTIFICATION_SERVICE);
       manager.notify(ID_NOTIFICATION, builder.build());
}
```

5.4 Calcolo delle calorie e tracciamento del percorso

Durante la corsa vengono effettuati i rilevamenti della posizione dell'utente ogni 4-5 secondi. Ad ogni aggiornamento sulla posizione le *API* di Google ci restituiscono un oggetto location, con cui è possibile calcolare la distanza percorsa comparando l'oggetto con un'altra location. Successivamente la distanza calcolata viene passata ad una funzione che calcolerà, attraverso delle formule, il consumo energetico. Per calcolare le calorie bruciate vengono moltiplicati i chilometri percorsi per il peso dell'utente, il tutto moltiplicato per una costante. Infine viene stampato a schermo il numero di calorie consumate dall'utente nella sessione corrente.

Oltre all'aggiornamento delle calorie bruciate, vengono salvati in un *ArrayList* tutte le coordinate della posizione dell'utente ad ogni rilevamento. Questo permette di tracciare, tramite una linea, il percorso che effettua il corridore.

Inoltre, il centro della mappa sarà fatto corrispondere sempre con la posizione corrente dell'utente, infatti la telecamera viene spostata continuamente sulle nuove coordinate.

```
private void updateMap(Location location){
    LatLng newPosition = new LatLng(location.getLatitude(),
        location.getLongitude());
    routePoints.add(newPosition);

    map.addPolyline(new PolylineOptions());
    map.moveCamera(CameraUpdateFactory.newLatLng(newPosition));

    route.setPoints(routePoints);

    if(lastLocation != null){
        float distance = location.distanceTo(lastLocation);
        incTotalMeters(distance);
        updateCal(distance);
    }
    this.lastLocation = location;
}

private void updateCal(float distance){
    float gap = 0;
    switch (typeActivity){
        case CYCLING:
            gap = calculateCalBurnedInCycling();
            break;
        default:
            gap = this.constant * userWeight * (distance/1000);
    }
    this.totalCal += gap;
    this.textViewCal.setText((int) this.totalCal + "");
}
```

5.5 Gestione delle date degli appuntamenti

Per quanto riguarda gli appuntamenti è importante gestire le loro date. Gli appuntamenti sono salvati in memoria quando si riceve una risposta, ma devo essere cancellati quando sono ormai scaduti. Per prima cosa, quando un appuntamento viene mostrato sullo schermo la data viene formattata in un for-

mato più leggibile. La data apparirà seguendo questo pattern: *dd-MM-yyyy*. Ogni volta che viene effettuato il caricamento degli appuntamenti viene eseguito un controllo sulla loro scadenza. La data dell'appuntamento, espressa in millisecondi, viene comparata con la data attuale. Se l'appuntamento dovesse essere più vecchio di due ore allora viene subito cancellato.

Di seguito sono riportate le porzioni di codice che si occupano di formattare la data e di verificare se quell'allenamento è da cancellare o meno. I metodi mostrati fanno parte della classe **UString** (*utility-string*), in cui sono implementati tutti metodi di classe statici.

```
public static String getDateToString(Date date){
    if(date == null){
        return "-/-/-";
    }
    String DATE_FORMAT_NOW = "dd-MM-yyyy";
    SimpleDateFormat sdf = new SimpleDateFormat(DATE_FORMAT_NOW);
    return sdf.format(date);
}

public static boolean isOldAppointment(String date){
    try {
        Date dateTime = formatAppointment.parse(date);
        if (System.currentTimeMillis() > (dateTime.getTime() +
            7200000)) {
            return true;
      }
   } catch (ParseException e) {
        e.printStackTrace();
    }
    return false;
}
```

5.6 Registrazione online

Essendo FitBody un'App che comunica con un server esterno, non si può non mostrare una pagina *PHP* che gestisce le richieste *HTTP*. Prendiamo, come esempio, la registrazione effettuata dall'utente. L'utente invierà una richiesta *HTTP* con metodo *POST* ad un certo *URL*. Il primo controllo verifica la presenza di tutti i dati richiesti, attraverso i metodi **isset**, ovvero e-mail e

password. Subito dopo viene effettuata una connessione con il database e viene creato un codice alfa-numerico che servirà per l'attivazione dell'account. Successivamente viene eseguita la query per l'inserimento dell'utente nella banca dati. Se l'operazione di inserimento va a buon fine viene inviata un e-mail con il link di attivazione all'utente, altrimenti viene inviato al *Client* un messaggio che indicherà la non riuscita dell'operazione. Nel codice riportato viene mostrata la generazione del codice alfa-numerico e l'invio dell'email all'utente.

```
<?

   // Generazione del codice alfa-numerico
   $code = '';
    $characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
   for ($i = 0; $i < 5; $i++) {
      $code .= $characters[rand(0, strlen($characters) - 1)];
   }

   // Viene effettuata la query in inserimento
   ...

   // Invio dell'e-mail per attivare l'account
   $to  = '$email';
    $subject = 'FitBody - Attivazione';
   $link = "http://xxxxxxxxx.altervista.org/xxxx.php" . "?email=" .
       $email . "&code=" . $code;
   $message = 'Gentile utente, per poter attivare il proprio
       account' .
            'e completare la registrazione basta cliccare il seguente
               link:' . '$link';

    $headers = 'MIME-Version: 1.0' . "\r\n";
    $headers .= 'Content-type: text/html; charset=iso-8859-1' .
        "\r\n";
    $headers .= 'From: FitBody <fitbody@email.com>' . "\r\n";
    mail($to, $subject, $message, $headers);

?>
```

Capitolo 6

Sviluppo e Testing

Per la realizzazione dell'App sono stati utilizzati vari strumenti di sviluppo.

6.1 Computer utilizzato

Il PC con cui è stata realizzata l'applicazione è un PC portatile Asus del 2012, con processore Intel i7, con 8 GB di RAM DDR3. Il sistema operativo installato sulla macchina è Microsoft Windows 8.1.

6.2 IDE utilizzato

L'applicazione è stata realizzata interamente utilizzando l'ambiente di sviluppo **Android Studio** versione 1.1.0. *Android Studio* è l'IDE ufficiale per lo sviluppo di applicazioni Android.

6.2.1 Creazione di un progetto

Quando viene creato un progetto Android la piattaforma crea automaticamente tutti i file necessari per il funzionamento di un'applicazione. Il progetto iniziale contiene il package principale con la *MainActivity*, tutte le cartelle delle risorse ed altri file importanti per il funzionamento dell'App come il **Manifest** e il **Gradle**.

Il *Gradle* è un file di configurazione dell'App, in cui vengono descritti il package principale dell'applicazione, il numero di versione ed altre informazioni. E' molto importante, poiché al suo interno devono essere indicate tutte le librerie che l'applicazione utilizza. Se non si dovesse inserire il riferimento ad

una libreria, questa non verrebbe riconosciuta dall'ambiente di sviluppo.

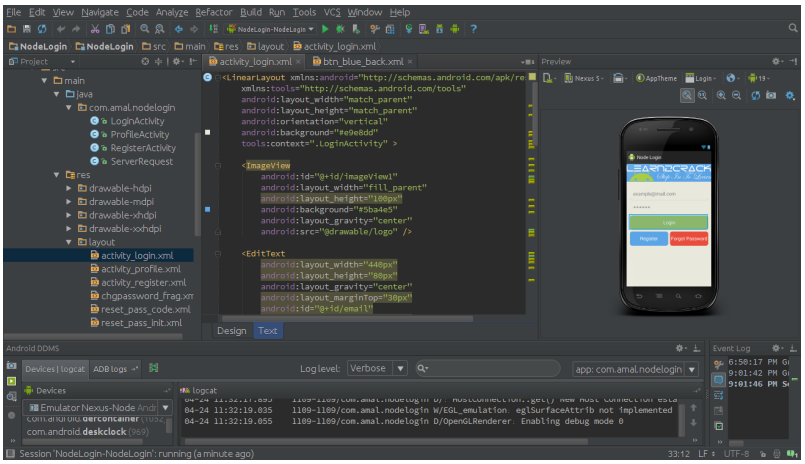

Figura 6.1: IDE di sviluppo con schermata per la progettazione di un layout

6.3 SDK di Android

Android SDK è il pacchetto di strumenti che permette di realizzare applicazioni per gli smartphone, tablet e tutti gli altri dispositivi che hanno Android. Questo viene installato automaticamente durante l'installazione dell'IDE sopra descritto e può essere aggiornato attraverso **Android SDK Manager**.

Oltre a strumenti per la costruzione di App, mette a disposizione un emulatore per testare le applicazioni direttamente su dispositivi virtuali con una versione di Android scelta dallo sviluppatore.

6.4 Librerie esterne utilizzate

Le librerie di default offerte da Android non sono state sufficienti per sviluppare l'App, infatti sono state utilizzate molte librerie esterne, molte delle quali offerte dalla comunità di Android.

Le librerie usate sono:

- **MaterialNavigationDrawer** per creare il menu principale a scomparsa;
- **MaterialEditText** per visualizzare EditText con funzionalità aggiuntive;

- **Facebook-sdk** ha permesso l'integrazione dell'App con il social network;
- **Jasypt** per criptare e decriptare le password degli utenti;
- **MPAndroidChart** per disegnare grafici. I grafici disegnati mostrano agli utenti i progressi conseguiti;
- **play-services** è una libreria di Google per la visualizzazione delle pubblicità;
- **MaterialDataPicker** ha permesso di creare un *date-time picker* in modo semplice;
- **BottomSheet** è la libreria che si occupa di visualizzare il menu a scomparsa verso il basso;
- **YouTubeAndroidPlayerApi** gestisce la visualizzazione dei video caricati su YouTube.

6.5 Test dell'applicazione

Le funzionalità dell'applicazione sono state implementate con successo ed è stato possibile verificare l'effettivo funzionamento grazie ai test effettuati su vari dispositivi con diverse versioni di Android e diversi tipi di schermo.

I dispositivi su cui è stata testata l'App sono:

- **Motorola Moto G 2013** con installato Android 5.0. Lo schermo adottato da questo dispositivo ha una densità di 256 pixel per pollice;
- **OnePlus One** e **Nexus 5** aventi la versione di Android 5.1.1 e con schermi di elevata densità di pixel. Il primo con una densità pari a 401 pixel per pollice, mentre il secondo con una densità di 445 pixel per pollice.
- **Samsung Galaxy Core** con installata una versione precedente di Android: la 4.3. Il suo schermo ha una densità di circa 190 pixel per pollice.

L'applicazione funziona perfettamente su tutti i dispositivi elencati precedentemente. Su smartphone di ultima generazione, con processori che superano i *2.2 GHz* l'applicazione è più performante quando vengono eseguite operazioni pesanti come il caricamento di numerosi esercizi con le relative immagini.

Il consumo della batteria è minimo anche con un utilizzo prolungato dell'applicazione. Il consumo aumenta leggermente durante l'attività della corsa,

dove viene richiesta l'attivazione del GPS. L'intervallo di tempo utilizzato per ricevere aggiornamenti sulla posizione consente di avere un'ottima precisione con un utilizzo moderato della batteria. Il percorso tracciato sulla mappa corrisponde al percorso reale effettuato dal corridore.

Inoltre, durante i test, il calcolo del consumo energetico delle varie attività è risultato molto accurato e conforme alla realtà.

Conclusioni

Lo sviluppo di questa applicazione è stato interessante e costruttivo, soprattutto quando ho dovuto risolvere alcuni problemi importanti, che non permettevano l'utilizzo dell'App. I problemi affrontati mi hanno permesso di ampliare e migliorare il mio bagaglio culturale. Tutte le nozioni imparate in aula sono state di grande aiuto e mi hanno permesso di realizzare un'applicazione completa in tutti i suoi dettagli.

La realizzazione del progetto è stata progressiva. Per quanto riguarda l'interfaccia grafica, si è andata a sviluppare progressivamente, seguendo lo stato di progettazione dell'applicazione. Tutte le schermate sono state, prima di essere implementate, disegnate su carta, in modo da rendere la GUI più intuitiva e di facile utilizzo.

L'App non è stata ancora publicata sullo Store, poichè, per motivi di tempo, non è stato possibile riempire il database con tutti gli esercizi per il fitness. Prima che l'applicazione possa essere publicata trascorrerranno, almeno, un paio di mesi.

Sviluppi Futuri

Nonostante l'applicazione abbia molteplici funzionalità, queste dovranno essere ampliate in futuro. Una modifica necessaria sarà l'aggiunta di un rilevatore di battiti cardiaci, avviabile durante una corsa. Quindi, sarà necessario implementare una comunicazione *Bluetooth Low Energy* con un dispositivo *Embedded* che rileverà i battiti del corridore. Successivamente, i dati ricevuti dal dispositivo dovranno essere elaborati e visualizzati a schermo.

Grazia alla struttura del progetto, le modifiche da effettuare richiederanno una minima modifica nel codice, velocizzando i lavori di ampliamento e miglioramento.

Ringraziamenti

Prima di tutto ringrazio i miei genitori, che mi hanno permesso di frequentare questa Università. Un grazie va al professore Mirko Ravaioli, che mi ha dato delle dritte per migliorare il mio progetto. Un grazie speciale ai miei coinquilini, Alex e Matteo, che hanno dovuto sopportare le mie ansie. Infine un grazie a tutti i miei amici che mi hanno aiutato a testare l'applicazione su diversi dispositivi; in particolare Aldo, Berlo, Brando, Paolo, Laura e Francesca.

Bibliografia

[1] Stefano Rizzato - *Un sistema, 19 mila schermi: l'incredibile frammentazione del mondo Android* - http://www.lastampa.it

[2] http://developer.android.com

[3] https://en.wikipedia.org

[4] https://developers.facebook.com/

[5] https://bitbucket.org

Elenco delle figure

1.1 Logo di Android 1
1.2 Loghi delle varie release di Android 2
1.3 Varietà di modelli Android 4
1.4 Differenza tra notifiche - Android a sinistra iOS a destra 5

3.1 Ciclo di vita di un'Activity 12
3.2 Diagramma dei casi d'uso per accedere all'App 13
3.3 Diagramma di sequenza per registrarsi all'App 13
3.4 Diagramma degli stati di un account 14
3.5 Database online semplificato 17
3.6 Rappresentazione di un processo leggero (Thread) 19
3.7 Diagramma di sequenza per interazione tra Service e Broadcast Receiver del MediaPlayer 22
3.8 Post Open Graph di una corsa effettuata 24

4.1 Diagramma delle classi semplificato che mostra una sessione . . 30
4.2 Diagramma di sequenza della procedura di accesso 34
4.3 Schermata del Login e del menu principale 35
4.4 Schermate per la creazione e modifica di un programma di allenamento 36
4.5 Comunicazione tra Fragment e Activity 37
4.6 Schermate di allenamento fitness e corsa 39
4.7 Schermata per l'acquisto di allenamenti e menu delle impostazioni 40
4.8 Trasformazione di una lista in una lista visibile di oggetti 44
4.9 Riciclo delle celle utilizzate in una ListView 45

6.1 IDE di sviluppo con schermata per la progettazione di un layout 56