Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Tesi di Laurea

Progettazione e implementazione di un'app Android per la gestione delle attività relative ad una palestra

Design and implementation of an Android app for managing the activities of a gym

Relatore

Prof. Domenico Ursino

Candidato

Valerio Procaccioli

Anno accademico 2019-2020

Indice

Elenco delle figure

Elenco dei listati

Introduzione

Si può dire che l'attività fisica sia nata con l'uomo; a partire dai popoli primitivi, infatti, forza e resistenza erano ritenute indispensabili per poter affrontare le difficoltà della vita, ossia: cacciare, difendersi e costruire dei ripari. L'uomo si accorse rapidamente di come la ripetitività di certe azioni consentiva di eseguirle, poi, con maggiore facilità e sicurezza; dunque, dal movimento sorse l'allenamento. I Greci dettero sempre grandissima importanza all'attività fisica, tanto da porre come riferimento per l'inizio della loro civiltà la data dei primi Giochi Olimpici (776 a.C.). Gli Spartani vedevano l'allenamento come un'ottima preparazione alla guerra; invece, ad Atene sorsero i ginnasi, ovvero edifici dedicati allo sport, nonchè precursori delle odierne palestre. Inoltre, nei luoghi in cui si svolgevano gli incontri sportivi più blasonati vennero costruiti degli stadi, composti da un campo piano per le gare e da gradinate per il pubblico attorno. Persino ai tempi dell'antica Roma la preparazione fisica dei giovani era tenuta in grande considerazione, soprattutto come propedeutica all'addestramento militare.

"*Orandum est ut sit mens sana in corpore sano*"
"*Bisogna pregare affinché ci sia una mente sana in un corpo sano*"
(Giovenale, Satire, X, 356)

Tornando all'attualità, la pandemia da COVID-19 ha drasticamente stravolto tutti i settori della società, tra i quali lo sport non fa eccezione. Le misure di distanziamento sociale, attuate per prevenire la diffusione del virus, hanno portato alla sospensione di ogni forma di attività sportiva, individuale e collettiva, professionistica e dilettantistica. Una scelta sofferta, ma inevitabile, al fine di salvaguardare la salute della popolazione, tanto quanto inevitabili sono state le conseguenze sociali ed economiche.

In questo frangente uno strumento del quotidiano come lo smartphone, il quale occupa un posto fondamentale nelle nostre vite permettendoci di compiere tantissime mansioni, ha assunto un ulteriore ruolo nell'ambito dell'attività motoria. Sono, infatti, sempre più numerose le app scaricabili sul proprio dispositivo utili a soddisfare tale esigenza. Alcune di esse utilizzano funzioni, ad esempio il GPS, per misurare gli spostamenti a piedi, mentre altre si interfacciano con sensori già inte-

grati nel telefono, che misurano i movimenti, i battiti cardiaci o altri dati fisiologici importanti; altre, infine, sono state concepite per svolgere gli esercizi comodamente in casa. In certi casi viene anche offerta la possibilità di inserire informazioni o selezionare difficoltà, allo scopo di programmare durata ed intensità dello sforzo fisico da eseguire. Da un simile contesto il nostro progetto prende le mosse, prefiggendosi l'obiettivo di sviluppare un'applicazione che agevoli, invece di intralciare, la pratica dell'attività fisica. Necessitiamo, quindi, di un programma che massimizzi le motivazioni dell'utente e ne minimizzi le distrazioni, tenendo, inoltre, conto di quanto la componente "social" sia ormai così centrale per i giovani. L'applicazione in seguito descritta nasce per il sistema operativo Android, la cui ampia diffusione su scala mondiale ci consente di raggiungere un bacino d'utenza maggiore. Ulteriori fattori a sostegno della scelta sono, poi, l'enorme versatilità, strumentale ad una più libera creatività lavorativa, nonchè l'elevato supporto fornito ai programmatori attraverso continui aggiornamenti da parte dell'ambiente di sviluppo integrato, chiamato Android Studio.

I passi che contraddistinguono lo sviluppo dell'app sono i seguenti: fase analitica, fase progettuale e fase implementativa. Il processo si esegue a partire dalla fase analitica, in cui vengono inquadrate natura ed origine del problema assieme ai requisiti, che devono, tassativamente, essere garantiti; questo permette di definire gli obiettivi da centrare, in modo da lavorare avendo ben chiare le finalità. Si continua con la fase progettuale, dove vengono illustrate struttura e caratteristiche prioritarie del programma per poi svolgere l'analisi delle azioni che l'utente può intraprendere mediante l'applicazione (i così detti "casi d'uso"). Così facendo, l'elaborazione del prodotto risulterà avere una base solida d'avvio, avendone stabilito sia le funzioni generiche che le peculiarità. Successivamente si procede con la fase implementativa, contrassegnata dallo studio meticoloso del codice che realizza i compiti previsti e dalla stesura completa di un manuale utente, il quale semplificherà l'uso del software una volta sviluppato. In tal senso, portare a termine questo passaggio con successo significa consegnare un'app ben testata, con una remota possibilità di incontrare comportamenti inattesi e, perciò, errati. Inoltre, l'esperienza offerta all'utente deve essere immediata ed efficiente, ossia è necessario garantire servizi tempestivi ed efficaci. A conclusione delle operazioni vi è, infine, l'analisi SWOT, momento in cui si guarda con occhio critico a debolezze e minacce, oltre a prospettare spunti di miglioramento per un eventuale sviluppo futuro; rendendo, pertanto, tale applicazione un punto di partenza concreto.

Questa tesi è strutturata come di seguito specificato:

- Nel Capitolo 1 verrà illustrato Android, il sistema operativo su cui poggia l'applicazione, fornendo, innanzitutto, alcune informazioni riguardo le origini, la rapida diffusione e le problematiche. Successivamente l'attenzione verrà posta sulle controversie legali che lo hanno caratterizzato. Infine, verrà descritta la sua struttura, dando particolare rilevanza all'architettura e all'ambiente di sviluppo specializzato, ossia Android Studio.
- Nel Capitolo 2 ci si occuperà della specifica e dell'analisi dei requisiti, durante la quale verrà inizialmente presentato il contesto di riferimento, evidenziando l'utilità, i vantaggi e le caratteristiche del progetto; subito dopo, a valle dell'attività di analisi, verranno esposti i requisiti che il prodotto dovrà garantire.

- Nel Capitolo 3 verrà riportata nel dettaglio la fase progettuale, delineando lo scheletro dell'applicazione attraverso una mappa ed un percorso dei file. In seguito si porrà l'accento sulle attività e sulle azioni previste per gli utenti. In ultima istanza verrà esaminata la base di dati Firebase, illustrando le sue proprietà.
- Nel Capitolo 4 si affronterà la fase implementativa, soffermandosi sull'enumerazione delle singole funzionalità a livello di codice. Nel prosieguo il capitolo verrà arricchito da un manuale utente che faciliterà al lettore l'utilizzo dell'app.
- Nel Capitolo 5 verrà svolta l'analisi SWOT, evidenziando le qualità del risultato finale e, a seguire, verrà effettuata una comparazione con app correlate. In chiusura verrà prodotta una revisione critica dell'operato rilevando alcuni spunti utili per il futuro.
- Nel Capitolo 6 saranno tratte le conclusioni sul lavoro svolto.

1

Android

Nel primo capitolo introduciamo Android, facendo particolare attenzione alle sue origini e ai motivi per i quali tale piattaforma risulta essere la più utilizzata al mondo.

1.1 Introduzione ad Android

Android è un sistema operativo per dispositivi mobili sviluppato da Google, progettato principalmente per smartphone e tablet. Nel suo continuo evolversi ha successivamente abbracciato altre apparecchiature, permettendo di implementare interfacce utente specifiche come:

- *Android TV*, per particolari televisori.
- *Android Auto*, per avere accessibilità all'interno delle automobili, vuoi per le funzioni di GPS (Global Positioning System), vuoi per finalità manutentive.
- *Wear OS*, per Smart Watches, i quali permettono, oltre che un'elevata portatilità del dispositivo,anche di tenere sotto controllo i propri parametri vitali nell'intero arco della giornata.
- *Google Glass*, per gli occhiali.

1.1.1 Origini e storia recente

Nell'ottobre del 2003 quattro importanti personaggi del settore informatico, ossia Andy Rubin (ex dipendente Apple e cofondatore dell'azienda Danger) [11], Rich Miner (altro cofondatore di Danger), Nick Sears (vicepresidente della società TMobile) e Chris White (autore della User Interface di Web TV) [10] fondarono l'Android Inc. per sviluppare, a detta di Rubin "... dispositivi cellulari piu consapevoli della posizione e delle preferenze del loro proprietario". Inizialmente fu celato il reale obiettivo della società, la quale operava pubblicamente alla progettazione di software per dispositivi mobili. Inoltre, il budget iniziale era assai esiguo, tanto da esaurirsi già nel primo anno. Solo con un fondamentale finanziamento (pari a 10.000 dollari) da parte dell'amico di Rubin, chiamato Steve Perlman, fu possibile continuare lo sviluppo. Data cardine per la storia di Android è stata il 17 agosto 2005, quando

Google LLC acquisì l'azienda per 50 milioni di dollari, con lo scopo di entrare nel mercato della telefonia mobile [8]. Difatti, negli anni successivi, iniziò a prendere forma l'interesse che Rubin aveva nell'implementare e pubblicare un sistema operativo (abbreviato SO, oppure OS in inglese) per dispositivi mobili basato sul kernel Linux. Il 5 novembre 2007 avvenne, invece, la presentazione ufficiale del "robottino verde" (Figura 1.1) da parte della OHA (Open Handset Alliance) [9], un consorzio di aziende del settore Hi Tech che includeva Google, produttori di smartphone come Samsung, operatori di telefonia mobile quali TMobile, Sprint Nextel ed, infine, produttori di microprocessori come Qualcomm.

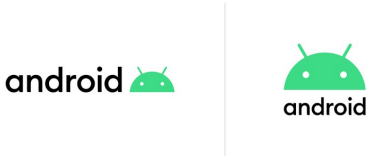

Figura 1.1. Due loghi di Android raffiguranti il robottino verde

1.1.2 Diffusione e problematiche del sistema operativo

Android non riscosse grandi consensi quando venne presentato nel 2007, nonostante molti rimasero colpiti dalle società tecnologiche che avevano collaborato con Google per formare "Open Handset Alliance". In realtà non fu chiaro se i produttori dei telefoni cellulari del tempo sarebbero stati disposti a sostituire i loro sistemi operativi esistenti con Android. L'idea di una piattaforma di sviluppo open source basata su Linux portò ad un grande interesse, a discapito, però, delle preoccupazioni legate ad una forte concorrenza da parte di aziende già affermate nel mercato come Nokia e Microsoft (Figure 1.2 e 1.3), assieme a sistemi operativi mobili concorrenti Linux che erano parallelamente in fase di sviluppo.

Figura 1.2. Il logo di Microsoft

Figura 1.3. Il logo di Nokia

Motivi del successo

Successivamente, anno dopo anno, Android si rivelò sempre più un successo fino a diventare il sistema operativo per smartphone più utilizzato al mondo. Gli analisti studiarono a fondo i motivi di questa grande propagazione, mettendo in evidenza alcune delle caratteristiche che contraddistinguono tale sistema operativo, ovvero:

- *Open Source*: Android permette ad aziende di terze parti, ad esempio Nokia e Amazon, di sviluppare progetti software a partire da un codice sorgente già esistente (FORK) e di rilasciare dispositivi hardware che eseguono al loro interno la versione di Android che gli sviluppatori hanno realizzato.
- *Elevata flessibilità a livello utente*: Android consente ai proprietari di personalizzare ampiamente il proprio dispositivo attraverso le sue schermate (Figura 1.4) e di scaricare app disponibili gratuitamente negli store dedicati (Figura 1.5).

Figura 1.4. Le schermate principali di alcuni dispositivi Android

Figura 1.5. Il logo del Google PlayStore

Frammentazione

C'è, però, la necessità di osservare che, se da una parte abbiamo il sistema operativo più diffuso al mondo, dall'altra abbiamo un'enorme quantità di dispositivi, sviluppatori e produttori per i quali Google non è sempre riuscita a fornire aggiornamenti coerenti a tutto l'ecosistema formatosi, a differenza di piattaforme concorrenti, come iOS, che hanno variazioni software/hardware molto minori. Il fenomeno che si è generato è detto di frammentazione, e riguarda l'impossibilità di raggiungere con le ultime funzionalità la maggior parte dei dispositivi [13]. A controbilanciare tale difetto a detta di OpenSignal, azienda sviluppatrice di app per Android ed iOS, è la più ampia portata del primo rispetto al secondo, che ne aumenta la potenziale ricompensa [1].

1.1.3 Controversie legali legate ad Android

A giudicare dai primi demo di Android si nota come la volontà iniziale della società non fosse quella di progettare un'interfaccia studiata per essere comandata dal tocco delle dita (Figura 1.6), al contrario di Apple. Sono, infatti, ancora più note le controversie legali per violazioni dei brevetti tra questi due marchi di livello mondiale. Negli anni a seguire molte altre cause hanno interessato questa piattaforma:

- *Oracle America, Inc.(Figura 1.7) contro Google Inc. (Figura 1.8)* [2] : è una controversia circa le rivendicazioni di copyright e brevetti di Oracle sul sistema operativo Android di Google. Tale causa si concluse con un verdetto positivo per quest'ultima, dato che la giuria non riscontrò alcuna violazione dei brevetti di Oracle e neppure la protezione da copyright delle API Java impiegate da Google.
- *Apple Inc.(Figura 1.9) contro Samsung Electronics Co., Ltd. (Figura 1.10)* [15]: è stata la prima di una serie di controversie legate alle violazioni dei brevetti sul design e sulle interfacce tra i due più grandi produttori di smarthphone conosciuti. Il tutto si concluse con Samsung che accettò, il 4 dicembre 2015, di pagare al suo concorrente statunitense la somma di 548 milioni di dollari stabilita dai tribunali, ma si riservava il diritto di recuperarla se la sanzione fosse stata successivamente modificata o cancellata in appello.

Figura 1.6. Il primo dispositivo con Android, il TMobile G1

Figura 1.7. Il logo di Oracle

Figura 1.8. Il logo di Google

Figura 1.9. Il logo di Apple

Figura 1.10. Il logo di Samsung

1.2 Caratteristiche del sistema operativo

In questa sezione ci occuperemo delle nozioni relative alle caratteristiche tecniche e strutturali del sistema operativo Android, al fine di introdurre l'IDE di sviluppo delle app chiamato Android Studio.

1.2.1 Architettura

In termini di sviluppo la parte più importante di Android è il suo sistema operativo. Il suo modo di operare è caratterizzato da:

- *Gestione Hardware*, per conto delle applicazioni.

- *Fornitura di servizi alle applicazioni*, principalmente per finalità di sicurezza, di networking e di gestione della memoria.
- *Gestione dell'esecuzione delle applicazioni*, per dare la sensazione di eseguire più applicazioni quasi in contemporanea.

L'architettura di un sistema Android è mostrata nel diagramma logico di Figura 1.11, il quale non presenta esattamente tutte le componenti in quanto ha il solo scopo di mostrare come sono organizzati i suoi livelli.

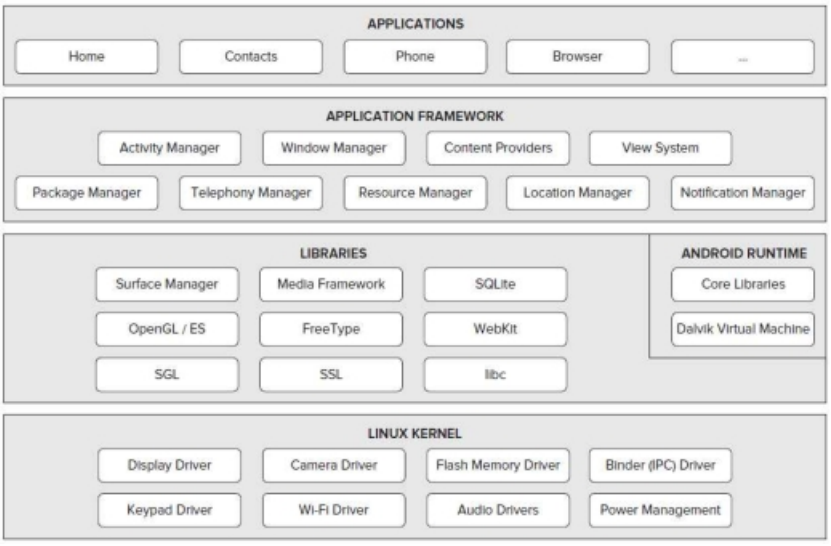

Figura 1.11. La struttura di Android

Il *Linux Kernel* è responsabile dell'interfacciamento con l'hardware e dei servizi di gestione della memoria ed avvio dei processi. Passando al livello successivo troviamo le librerie di basso livello, come SQLite e simili, accompagnate dal *Runtime Android*, che rappresenta l'ambiente nel quale vengono eseguite le applicazioni. Successivamente incontriamo il *Framework delle applicazioni*, con cui gli sviluppatori interagiscono in quanto contiene tutte le librerie utili per scrivere app. In cima abbiamo, invece, il livello delle *Applicazioni*, dove risiedono le app che scriviamo e costruiamo con il sistema operativo in questione.

App native

La maggior parte delle applicazioni che vengono quotidianamente scaricate sono app native. Nel dettaglio parliamo di app sviluppate specificamente per un sistema operativo; quindi, un'app iOS non funzionerà su un sistema Android e viceversa. Questo in virtù del fatto che il linguaggio di programmazione è differente da un sistema operativo all'altro, infatti:

- *iOS* usa Objective C;
- *Android* usa Java (Figura 1.12).

Figura 1.12. Il logo di Java

Vantaggi

Come possiamo immaginare, lo sviluppo di app native porta vari vantaggi tra cui:

- *Maggiore velocità, affidabilità e migliore reattività*, oltre ad una maggiore risoluzione visiva che garantisce una migliore esperienza utente.
- *Accesso facile a tutte le funzionalità del dispositivo*, ad esempio la fotocamera o il microfono.
- *Notifiche Push*, sono notifiche sviluppabili solo nelle app native, volte ad avvisare l'utente ed attirarne l'attenzione (Figura 1.13).
- *Indipendenza da Internet*, questo resta un vantaggio, sebbene attualmente esistono poche zone non coperte da una rete Internet.

In conclusione, le app native hanno ottime performance e sfruttano appieno le funzionalità del dispositivo; tra queste, le migliori rispettano anche le caratteristiche di design di ciascuna piattaforma.

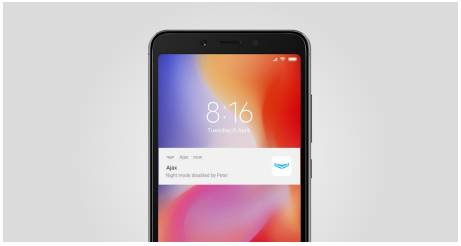

Figura 1.13. Una notifica push

Architettura di un'applicazione Android

Le applicazioni Android non sono uguali a quelle desktop sebbene all'apparenza si possono notare somiglianze; la vera differenza è a livello strutturale. Mentre le applicazioni desktop risultano essere autonome e caratterizzate dal file `.exe`, che contiene le routine e subroutine necessarie al loro interno, quelle Android sono, invece, descritte da un pacchetto di componenti accoppiate in modo lasco in un file `.apk`. Tali componenti sono:

- `AndroidManifest.xml`: un file XML che definisce le proprietà dell'applicazione, ad esempio il nome e la schermata principale che apparirà all'utente.
- *Component*: elementi definibili come i Building Block dell'applicazione, i quali permettono un accoppiamento lasco. Nello specifico ogni singolo Component svolge una funzione, come mostrare una schermata oppure eseguire processi in background o, anche, fornire dati.
- *Intent*: ciò che collega i Component e permette di comunicare anche con altre applicazioni.
- *Resources*: sono le risorse audio, video o immagine che possiamo includere nella nostra applicazione.

Component

Esistono quattro tipi di Component differenti, che sono alla base dello sviluppo delle app Android:

- *Activity*, si possono definire come qualcosa che interagisce a schermo con l'utente con lo scopo di catturarne l'attenzione, in un'applicazione possiamo averne una sola oppure diverse; importante è il fatto che le Activity possono reagire agli eventi prodotti dallo stesso utente, come il click.
- *Service*, rappresentano la parte di codice invisibile all'utente, cioè priva di interfaccia, e sono caratterizzati da operazioni di lunga durata eseguite in background.
- *BroadcastReceiver*, vengono utilizzati quando si desidera implementare una logica di risposta agli eventi generati dal sistema Android all'interno della nostra applicazione.
- *ContentProvider*, sono le componenti che permettono la condivisione dei dati tra le applicazioni.

Linguaggio di programmazione: Java

A differenza della normale programmazione in Java, in Android la compilazione incontra due fasi, in quanto gli eseguibili Android sono file `.dex` e non `.class`. Il codice sorgente viene prima compilato in file `.class` come in Java SE/EE; successivamente i file risultanti sono covertiti mediante lo strumento *DX* in un file `.dex`, il quale verrà eseguito all'interno del *Runtime Android*. Tale file `.dex` viene infine trasformato mediante il processo di "*packaging*" nel file `.apk`, ossia quello che viene installato nei dispositivi e mandato in esecuzione. Questo meccanismo è descritto in maniera esaustiva nella Figura 1.14.

1.2.2 Sviluppo con Android Studio

Introduciamo infine il discorso sull'IDE, cioè l'ambiente di sviluppo integrato con il quale abbiamo lavorato, ovvero Android Studio.

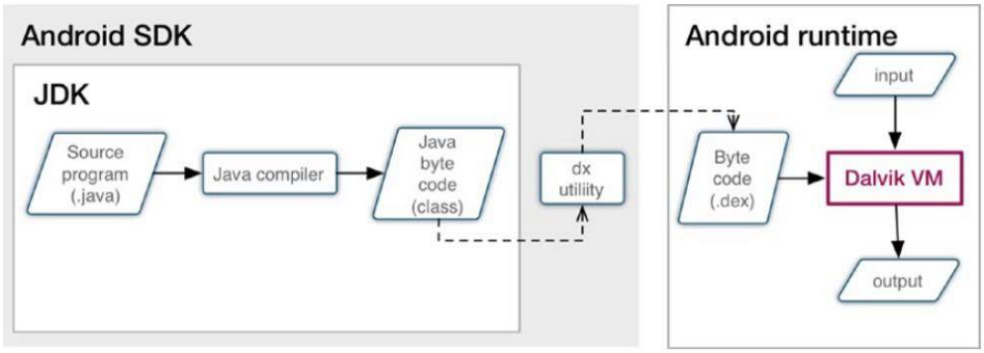

Figura 1.14. La compilazione di un'applicazione Android

Storia dell'IDE

Inizialmente lo sviluppo di app Android non era cosi semplice. Nel 2008 avvenne il rilascio di Android 1.0, con gli sviluppatori che potevano soltanto usufruire di una serie di script predefiniti e di strumenti a riga di comando. Mancavano, quindi, le funzionalità basilari di un ambiente di sviluppo integrato (IDE). Col passare degli anni vennero rilasciati gli strumenti Android per l'IDE Eclipse, fino ad arrivare al 2013 col rilascio, da parte di Google, di Android Studio. Tale ambiente di sviluppo è basato su IntelliJ di Jetbrain, un IDE caratterizzato da una versione free (gestita da una community) che è stata la base da cui si è partiti per realizzare Android Studio (Figura 1.15).

Figura 1.15. Il logo di Android Studio

Caratteristiche dell'IDE

Le caratteristiche che contraddistinguono Android Studio come l'IDE ufficiale per lo sviluppo Android sono:

- *Gestione degli errori in tempo reale*, tale ambiente di sviluppo indica in maniera esplicita gli errori sintattici, mediante input visivi facilmente intuibili, e le eccezioni a runtime, mediante una stampa dell'errore contenente tutte le informazioni utili per poterle gestire.

- *Presenza di numerose librerie online*, che permettono lo sviluppo e l'implementazione di funzionalità sempre più innovative.
- *Grande numero di sviluppatori*, ciò permette di confrontare ed analizzare continuamente i propri lavori con quelli altrui.
- *Ampio sviluppo delle interfacce utente*, dato che ad uno sviluppatore è concesso di scrivere il codice XML manualmente in modalità *Text*, oppure attraverso un'interfaccia grafica dedicata in modalità *Design* (Figura 1.16).

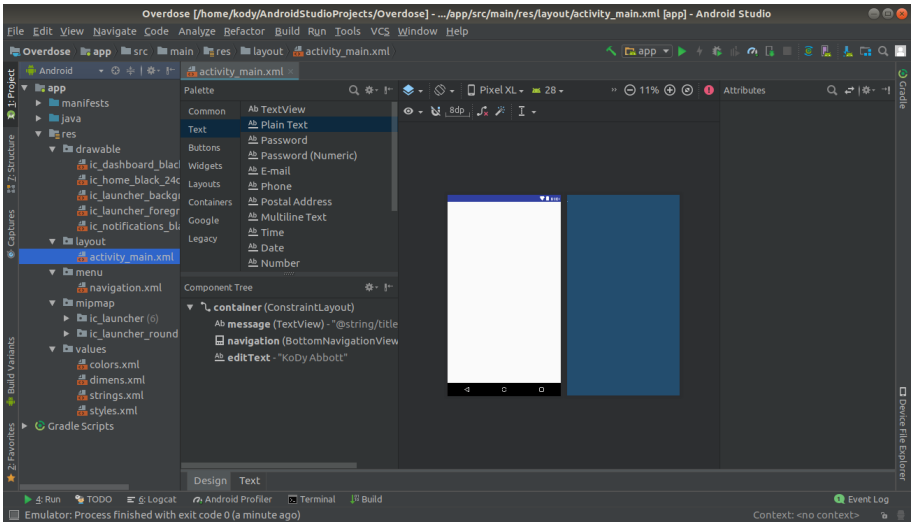

Figura 1.16. La modalità di sviluppo grafico

2

Descrizione del problema e analisi dei requisiti

In questo capitolo introduciamo l'origine del nostro progetto, ossia i motivi per i quali abbiamo reputato opportuno sviluppare questo particolare tipo di applicazione, arrivando a svolgere una completa analisi dei requisti.

2.1 Presentazione del contesto di riferimento

La nostra applicazione (Figura 2.1) è stata sviluppata al fine di poter permettere ai clienti di una palestra di svolgere attività fisica direttamente in casa o nei luoghi appositi. Inoltre, la maggior parte delle funzionalità dell'app sono monitorate dai personal trainer, i quali possono interagire con gli stessi utenti, fornendo loro esercizi mirati e schede di allenamento personalizzate.

2.1.1 Utilità e vantaggi

I vantaggi nell'impiego di questo tipo di applicazione sono i seguenti:

- *Allenamento in remoto*; come già affermato, l'obiettivo è quello di offrire un servizio che possa essere una soluzione al problema dello svolgimento dell'attività fisica, particolarmente necessario in questo periodo di pandemia..
- *Telefono contro carta*; nella maggior parte delle palestre le schede di allenamento vengono fornite in forma cartacea, con una concreta possibilità di smarrirle. L'impiego di un'app intermediaria garantisce, oltre che un minor consumo di carta dato dalla produzione delle stesse schede, anche una maggiore fruibilità all'utente che ha tutto a portata di click.
- *Scelta smart del tipo di allenamento*; per ogni scheda sono presenti vari livelli di difficoltà, i quali permettono all'utente di mettersi sempre alla prova.
- *Monitoraggio dell'attività fisica*; all'interno dell'applicazione è stato implementato un servizio che dà la possibilità di tenere traccia del proprio allenamento e dei propri risultati.
- *Motivazione ed aspetti sociali*; la chiave social è importantissima, in quanto sprona l'utente all'utilizzo dell'app e, allo stesso tempo, i clienti fungono da "*tester*" al fine di aggiornare e migliorare tutte le funzionalità offerte.

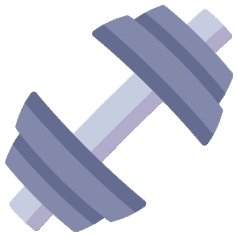

Figura 2.1. L'icona dell'app

2.1.2 Caratteristiche del progetto

L'applicazione consente una gestione ottimale delle schede d'allenamento di ogni singolo utente registrato ad opera dei personal trainer. Introduciamo, quindi, due classi di utenti dell'app:

- *user*, gli utenti di livello 0 che rappresentano i clienti;
- *admin*, gli utenti di livello 1 che rappresentano i personal trainer e il personale autorizzato.

Per queste due categorie vedremo implementate due interfacce grafiche differenti, in quanto le funzionalità richieste a livello *admin* e a livello *user* entrano in conflitto nella maggior parte dei casi. Per questo motivo, è necessaria una modifica dell'interfaccia utente della nostra applicazione, al verificarsi di particolari condizioni. Di seguito forniamo un esempio di questo conflitto, riguardante la prima interfaccia visibile dall'uno e dall'altro. Gli *admin* devono vedere la lista degli *user* allo scopo di gestirne le relative schede di allenamento. Essendo un cliente, lo *user* dovrà, invece, visualizzare il suo calendario personale, con le relative schede di allenamento giornaliere. Il caricamento di una delle due interfacce avviene quando si effettua il login all'interno dell'app, grazie alla distinzione delle due classi di utente fatta in precedenza. Dunque, l'*admin* avrà il compito di creare, modificare ed eliminare gli esercizi disponibili, oltre ad aggiungerli o rimuoverli dalle schede di allenamento degli *user*; questi ultimi potranno, poi, visualizzarle. Inoltre, è stata sviluppata anche un'interfaccia utente "profilo" minimale, che dà la possibilità agli *user* di personalizzare alcuni dei loro dati principali. Infine, sono state implementate funzionalità che rendono l'esperienza utente ancor più stimolante. Esempi di tali funzionalità sono i seguenti:

- *Timer personalizzato*; per ogni scheda di allenamento lo *user*, a suo piacimento, può decidere di svolgere attività in maniera autonoma o seguendo un timer d'esercizio/pausa impostato a seconda della difficoltà selezionata.
- *Conto calorie bruciate*; lo *user* può liberamente stabilire un obiettivo in termini di calorie bruciate e tenere conto dei suoi progressi mediante un'interfaccia specifica.

2.2 Analisi dei requisiti

In questa sezione descriviamo l'analisi dei requisiti, la quale ha inizialmente indirizzato nei giusti binari e, successivamente, caratterizzato la nostra applicazione.

2.2.1 Definizione dei requisiti

In Ingegneria del Software, l'analisi dei requisiti è un'attività necessaria e precedente allo sviluppo vero e proprio del programma, con la necessità di definire le funzionalità che il prodotto deve offrire, ossia i requisiti da soddisfare. Ciò che si ottiene attraverso l'analisi dei requisiti è, pertanto, un documento guida per il team di sviluppo nelle fasi di progettazione ed implementazione dell'app [4]. I due tipi di requisiti sui quali ci soffermiamo sono:

- *requisiti funzionali*;
- *requisiti non funzionali.*

2.2.2 Requisiti funzionali

I requisiti funzionali si possono presentare come delle funzionalità o dei servizi che la nostra applicazione deve necessariamente fornire o, talvolta, come reazioni a seguito di particolari input. Il documento contentente tali requisiti deve rispettare due importanti caratteristiche, ovvero:

- *completezza*: tutti i requisiti richiesti dagli utenti sono ben definiti;
- *coerenza*: non ci sono conflitti tra gli stessi requisiti.

Ovviamente più il nostro software sarà complesso e maggiore sarà lo sforzo richiesto per assicurarsi di garantire i parametri sopra citati.

Descrizione requisiti funzionali

Descriveremo i nostri requisiti funzionali mediante delle voci, ognuna delle quali ha un compito ben specifico (Tabella 2.1):

- *Nome*: il nome del singolo requisito;
- *Descrizione*: un breve testo che spiega il fine del requisito;
- *ID*: un codice di riconoscimento univoco per evitare fenomeni di conflitto;
- *Attore*: esso rappresenta l'utente o il sistema protagonista dell'evento.

Nelle seguenti tabelle definiamo i principali requisiti funzionali della nostra applicazione, apponendo una breve motivazione ad essi relativa.

Registrazione

Motivazione: nella maggior parte delle applicazioni è necessaria una funzionalità di registrazione, la quale permetta di identificare i diversi utenti fruitori del nostro servizio (Tabella 2.2).

Parametro	*Significato*
Nome:	
Descrizione:	
ID:	
Attore:	

Tabella 2.1. Il template della tabella utilizzata

Parametro	*Significato*
Nome:	registrazione
Descrizione:	meccanismo di registrazione per i nuovi utenti
ID:	funz.01
Attore:	utente

Tabella 2.2. Caratteristiche del requisito funzionale “registrazione”

Login

Motivazione: così come per la registrazione, l’app deve consentire il login da parte dello *user* in modo che quest’ultimo possa, poi, usufruire delle funzionalità offerte (Tabella 2.3).

Parametro	*Significato*
Nome:	login
Descrizione:	meccanismo di accesso per l’utente già registrato
ID:	funz.02
Attore:	utente

Tabella 2.3. Caratteristiche del requisito funzionale “login”

Visualizza calendario

Motivazione: dato che vogliamo garantire un allenamento completo in quattro settimane, lo *user* deve poter visualizzare il calendario con le schede di allenamento giornaliere. Anche l’*admin* può visualizzare il calendario personale di ogni singolo utente, con la possibilità di inserire, per ogni giorno libero, una scheda di allenamento (Tabella 2.4).

Visualizza profilo

Motivazione: lo *user* deve poter consultare i propri dati personali e, all’evenienza, modificarli tramite apposite funzionalità; inoltre, gli è anche consentito di cambiare a piacimento l’immagine del profilo, caricandone una dal proprio dispositivo (Tabella 2.5).

Parametro	*Significato*
Nome:	visualizza calendario
Descrizione:	visualizzazione del calendario personale dell'utente
ID:	funz.03
Attore:	sistema

Tabella 2.4. Caratteristiche del requisito funzionale "visualizza calendario"

Parametro	*Significato*
Nome:	visualizza profilo
Descrizione:	visualizzazione dell'interfaccia utente personale
ID:	funz.04
Attore:	sistema

Tabella 2.5. Caratteristiche del requisito funzionale "visualizza profilo"

Visualizza calorie

Motivazione: allo *user* deve essere possibile controllare e calcolare, rispetto all'obiettivo che ha stabilito, le attuali calorie bruciate; a tale scopo è necessario procedere con un'apposita funzionalità (Tabella 2.6).

Parametro	*Significato*
Nome:	visualizza calorie
Descrizione:	visualizzazione dell'interfaccia per il calcolo delle calorie
ID:	funz.05
Attore:	sistema

Tabella 2.6. Caratteristiche del requisito funzionale "visualizza calorie"

Visualizza scheda

Motivazione: per poter iniziare un allenamento lo *user* deve attingere dalla fonte, ossia alla scheda giornaliera. L'*admin*, invece, è colui che può visualizzare la scheda dell'utente per finalità gestionali (Tabella 2.7).

Parametro	*Significato*
Nome:	visualizza scheda
Descrizione:	visualizzazione della singola scheda di allenamento
ID:	funz.06
Attore:	sistema

Tabella 2.7. Caratteristiche del requisito funzionale "visualizza scheda"

Visualizza esercizi scheda

Motivazione: lo *user* ha la necessità di visualizzare gli esercizi all'interno di una scheda d'allenamento per poterli svolgere. L'*admin* invece, tramite questa interfaccia, può inserire e rimuovere gli esercizi dalla singola scheda di allenamento dell'utente (Tabella 2.8).

Parametro	*Significato*
Nome:	visualizza esercizi scheda
Descrizione:	visualizzazione degli esercizi dentro una scheda di allenamento
ID:	funz.07
Attore:	sistema

Tabella 2.8. Caratteristiche del requisito funzionale "visualizza profilo"

Visualizza allenamento

Motivazione: dopo aver selezionato una scheda di allenamento, lo *user* deve poter visualizzare l'interfaccia relativa all'allenamento stesso per poterlo avviare (Tabella 2.9).

Parametro	*Significato*
Nome:	visualizza allenamento
Descrizione:	visualizzazione dell'interfaccia di avvio dell'allenamento
ID:	funz.08
Attore:	sistema

Tabella 2.9. Caratteristiche del requisito funzionale "visualizza allenamento"

Visualizza lista utenti

Motivazione: l'*admin* deve poter visualizzare la lista di utenti registrati per poter interagire con essi e con le loro schede di allenamento (Tabella 2.10).

Parametro	*Significato*
Nome:	visualizza lista utenti
Descrizione:	visualizzazione della lista di utenti registrati
ID:	funz.09
Attore:	sistema

Tabella 2.10. Caratteristiche del requisito funzionale "visualizza lista utenti"

Visualizza lista esercizi

Motivazione: l'*admin* deve poter visualizzare la lista degli esercizi disponibili agli *user* per poterla gestire creando, eliminando o modificando i singoli esercizi (Tabella 2.11).

Parametro	*Significato*
Nome:	visualizza lista esercizi
Descrizione:	visualizzazione della lista di esercizi disponibili
ID:	funz.10
Attore:	sistema

Tabella 2.11. Caratteristiche del requisito funzionale "visualizza lista esercizi"

Creazione scheda

Motivazione: l'*admin* deve poter compilare una form dedicata alla generazione di una scheda d'allenamento per lo *user* (Tabella 2.12).

Parametro	*Significato*
Nome:	creazione scheda
Descrizione:	logica di creazione di una scheda di allenamento giornaliera per l'utente
ID:	funz.11
Attore:	admin

Tabella 2.12. Caratteristiche del requisito funzionale "creazione scheda"

Modifica scheda

Motivazione: l'*admin* deve poter compilare una form dedicata alla modifica dei campi di una scheda d'allenamento di uno *user* (Tabella 2.13).

Parametro	*Significato*
Nome:	modifica scheda
Descrizione:	logica di modifica dei parametri di una scheda di allenamento
ID:	funz.12
Attore:	admin

Tabella 2.13. Caratteristiche del requisito funzionale "modifica scheda"

Eliminazione scheda

Motivazione: l'*admin* deve poter eliminare una o più schede d'allenamento dello *user* (Tabella 2.14). [1]

Parametro	*Significato*
Nome:	eliminazione scheda
Descrizione:	logica d'eliminazione di una scheda di allenamento utente
ID:	funz.13
Attore:	admin

Tabella 2.14. Caratteristiche del requisito funzionale "eliminazione scheda"

Avvio allenamento

Motivazione: una volta che lo *user* ha selezionato la scheda di allenamento e la corrispettiva difficoltà, il sistema deve necessariamente avviare un allenamento coerente agli input forniti (Tabella 2.15).

Parametro	*Significato*
Nome:	avvio allenamento
Descrizione:	funzionalità per l'avvio dell'allenamento a partire della scheda selezionata
ID:	funz.14
Attore:	utente

Tabella 2.15. Caratteristiche del requisito funzionale "avvio allenamento"

Visualizzazione interfaccia video

Motivazione: una volta che lo *user* ha selezionato un esercizio, deve essere visibile l'interfaccia adibita alla riproduzione del video associato. L'*admin* potrà, inoltre, visualizzare i pulsanti utili a caricare un nuovo video direttamente sul database (Tabella 2.16).

Riproduzione video

Motivazione: quando lo *user* accede all'interfaccia video, la nostra applicazione deve permettergli di riprodurre il video associato all'esercizio (Tabella 2.17).

Upload video

Motivazione: l'*admin* deve avere la possibilità di caricare un video per ogni esercizio, oppure deve sovrascriverne uno già presente sul database (Tabella 2.18).

[1] Le stesse funzionalità di creazione, modifica ed eliminazione schede sono state sviluppate per i singoli esercizi, a partire dalla lista completa.

Parametro	*Significato*
Nome:	visualizzazione interfaccia video
Descrizione:	visualizzazione dell'interfaccia video per gli esercizi
ID:	funz.15
Attore:	sistema

Tabella 2.16. Caratteristiche del requisito funzionale "visualizzazione interfaccia video"

Parametro	*Significato*
Nome:	riproduzione video
Descrizione:	meccanismo di riproduzione dei video degli esercizi
ID:	funz.16
Attore:	utente

Tabella 2.17. Caratteristiche del requisito funzionale "riproduzione video"

Parametro	*Significato*
Nome:	upload video
Descrizione:	funzionalità di caricamento del video su database
ID:	funz.17
Attore:	admin

Tabella 2.18. Caratteristiche del requisito funzionale "upload video"

Calcolo calorie bruciate

Motivazione: una volta che lo *user* raggiunge l'interfaccia dedicata al calcolo delle calorie, deve poter effettuare il calcolo stesso sulla base di una logica ben definita (Tabella 2.19).

Parametro	*Significato*
Nome:	calcolo calorie bruciate
Descrizione:	funzionalità di calcolo delle calorie bruciate
ID:	funz.18
Attore:	utente

Tabella 2.19. Caratteristiche del requisito funzionale "calcolo calorie bruciate"

Utente o Admin

Motivazione: quando si effettua l'accesso, la prima schermata visibile varia a seconda del tipo di utente; è, perciò, obbligatorio che tale condizione sia verificata subito dopo il login (Tabella 2.20).

Parametro	*Significato*
Nome:	utente o admin
Descrizione:	meccanismo di controllo della tipologia di utente
ID:	funz.19
Attore:	sistema

Tabella 2.20. Caratteristiche del requisito funzionale "utente o admin"

Modifica profilo

Motivazione: Allo *user* deve essere concessa la funzionalità per modificare alcuni dei suoi dati personali, nel caso di errori in fase di registrazione, o per semplice personalizzazione della sua esperienza (Tabella 2.21).

Parametro	*Significato*
Nome:	modifica profilo
Descrizione:	meccanismo di modifica dei parametri del profilo
ID:	funz.20
Attore:	utente

Tabella 2.21. Caratteristiche del requisito funzionale "modifica profilo"

Aggiunta esercizi in una scheda

Motivazione: l'*admin* deve poter riempire le schede di allenamento riservate agli utenti con gli esercizi disponibili (Tabella 2.22).

Parametro	*Significato*
Nome:	aggiunta esercizi
Descrizione:	logica di aggiunta di uno o più esercizi all'interno di una scheda
ID:	funz.21
Attore:	admin

Tabella 2.22. Caratteristiche del requisito funzionale "aggiunta esercizi"

Rimozione esercizi in una scheda

Motivazione: l'*admin* deve poter eliminare gli esercizi all'interno di una scheda di allenamento (Tabella 2.23).

Parametro	*Significato*
Nome:	rimozione esercizi
Descrizione:	logica di rimozione di un esercizio dall'interno di una scheda
ID:	funz.22
Attore:	admin

Tabella 2.23. Caratteristiche del requisito funzionale "rimozione esercizi in una scheda"

2.2.3 Requisiti non funzionali

I requisiti non funzionali sono caratteristiche e funzionalità del software non richieste dal cliente, ma che influenzano pesantemente il lavoro degli sviluppatori. Essi non descrivono cosa fa il sistema, bensì sotto quali vincoli quest'ultimo esegue le funzionalità di sua pertinenza. Si possono anche definire come dei vincoli ai quali la nostra applicazione deve sottostare per garantirci un corretto funzionamento [5]. Nelle successive tabelle definiamo i requisiti non funzionali della nostra app, assieme alle motivazioni che hanno portanto al loro sviluppo.

Limitazione campo password

Motivazione: per garantire una maggiore sicurezza agli *user* e ai loro dati gioca un ruolo fondamentale la password, la quale deve necessariamente essere abbastanza lunga e complessa (Tabella 2.24).

Parametro	*Significato*
Nome:	limitazione campo password
Descrizione:	limite minimo in lunghezza che la password può avere
ID:	non.01
Attore:	sistema

Tabella 2.24. Caratteristiche del requisito non funzionale "limitazione campo password"

Coerenza dei campi nelle form

Motivazione: al fine di evitare di inserire dati non coerenti alle funzionalità dell'applicazione, ad esempio caratteri all'interno di variabili numeriche, è necessario introdurre appositi controlli, per evitare l'insorgere di eccezioni non gestite (Tabella 2.25).

Modifica dell'interfaccia a seconda dell'utente

Motivazione: dato che la nostra applicazione riserva funzionalità specifiche all'utente a seconda del suo livello (*user* o *admin*), è fondamentale che venga effettuato il controllo su questo specifico parametro. Infatti, in situazioni dove utilizziamo interfacce comuni, tale test evita di fornire ad un utente servizi non coerenti con la conseguente generazione di bug inattesi (Tabella 2.26).

Parametro	*Significato*
Nome:	coerenza campi form
Descrizione:	controllo sulla coerenza dei dati forniti in input
ID:	non.02
Attore:	sistema

Tabella 2.25. Caratteristiche del requisito non funzionale "coerenza campi form"

Parametro	*Significato*
Nome:	modifica interfaccia per utente
Descrizione:	verifica delle condizione dell'utente per garantire un corretto funzionamento
ID:	non.03
Attore:	sistema

Tabella 2.26. Caratteristiche del requisito non funzionale "modifica interfaccia per utente"

Controllo esercizi duplicati in una scheda

Motivazione: questo controllo rappresenta un altro requisito che ci evita la presenza di bug o incoerenze, ovvero la situazione nella quale abbiamo una scheda di allenamento con lo stesso esercizio presente molteplici volte (Tabella 2.27).

Parametro	*Significato*
Nome:	controllo esercizi duplicati
Descrizione:	verifica della presenza dello stesso esercizio nella scheda
ID:	non.04
Attore:	sistema

Tabella 2.27. Caratteristiche del requisito non funzionale "controllo esercizi duplicati"

3

Progettazione dell'app

In questo capitolo saranno affrontati gli argomenti riguardanti la struttura e le proprietà che caratterizzano la nostra app dal punto di vista della progettazione.

3.1 Struttura dell'applicazione

La fase di progettazione dell'app consiste nella messa in atto di tutte le sue funzionalità, le quali sono già state descritte durante l'analisi dei requisiti. In generale questa fase ci dà una visione complessiva di quello che sarà il prodotto finale, del quale l'utente potrà poi usufruire. La necessità primaria è, quindi, quella di progettare:

- la mappa dell'applicazione;
- il diagramma dei casi d'uso;
- il database.

Nelle prossime sottosezioni illustreremo ciascuna di queste fasi.

3.1.1 Mappa dell'applicazione

Una mappa dell'applicazione è un file in cui è possibile elencare le pagine, o meglio, i percorsi del nostro software, con lo scopo di gestirne l'organizzazione dello sviluppo e l'ottimizzazione dei contenuti. Nelle Figure 3.1 e 3.2 riportiamo lo schema del nostro lavoro.

3.1.2 Percorso dell'applicazione

Nella Figura 3.3 mostriamo lo schema delle cartelle e dei file che strutturano il nostro progetto. Caratterizziamo, dunque, alcune di queste componenti:

- `utilities`: prima di svolgere determinate funzioni, i dati sono "modellati" attraverso le classi contenute in questa cartella;
- `fragments`: contiene tutti i fragment con i quali lavoriamo;
- `adapters`: contiene tutti gli adapter associati ai fragment;

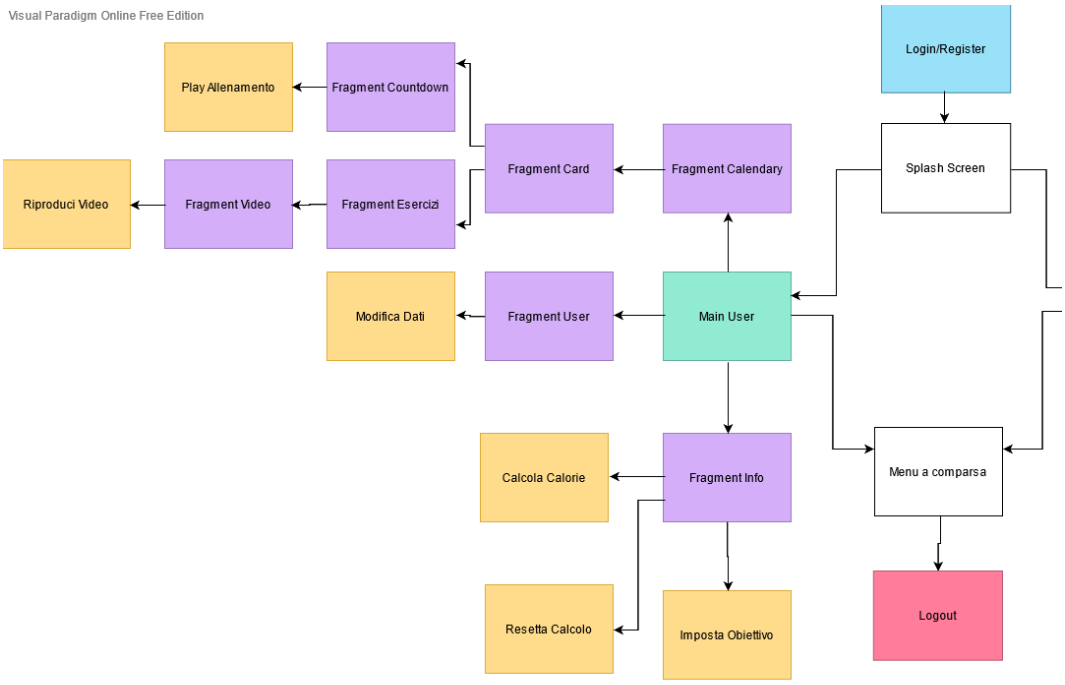

Figura 3.1. La mappa dell'applicazione (1)

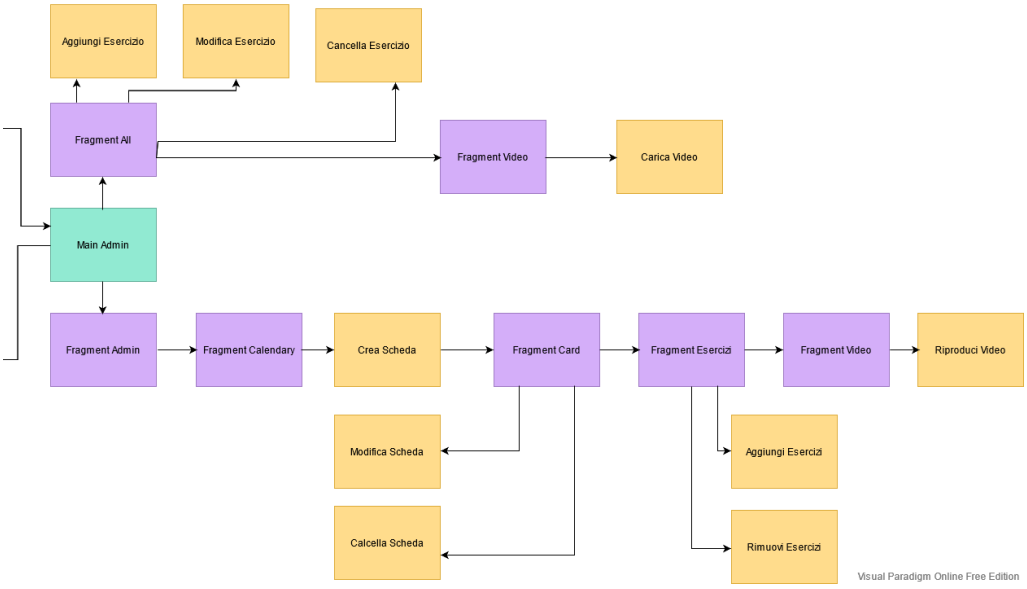

Figura 3.2. La mappa dell'applicazione (2)

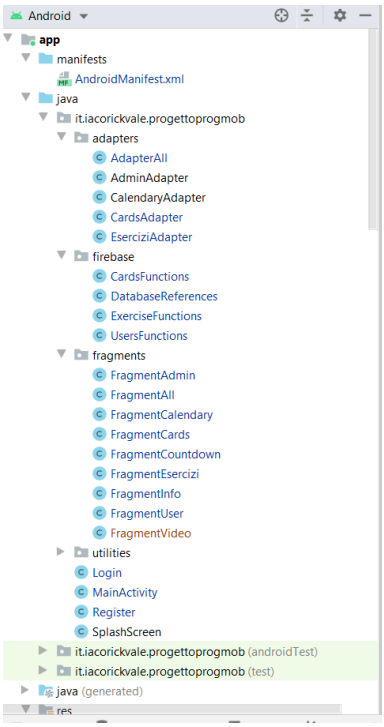

Figura 3.3. Il percorso dell'applicazione

- `firebase`: i dati vengono forniti dal/al database attraverso le funzioni contenute in questa cartella.

I restanti elementi che possiamo notare sono le principali classi Java della nostra applicazione: la `MainActivity`, il `Login`, il `Register` e la `SplashScreeen`.

3.1.3 Uso dei fragment

I fragment nel tempo si sono meritati un posto di spicco nel mondo della programmazione Android, tanto da essere inseriti come parte integrante del framework, nella sua Versione 3.0. La progettazione di un'interfaccia utente incentrata su tale meccanismo ha come primo obiettivo la modularità. Un fragment, infatti, occupa una porzione di Activity, ma non svolge il ruolo di semplice "raggruppamento di View", bensì appare come una sotto-applicazione dotata di un proprio ciclo di vita. Ciò lo rende potenzialmente separabile dagli altri fragment della stessa interfaccia o riutilizzabile, per conto proprio, nonchè congiuntamente a nuove componenti [3]. Oltre al fatto di poter costruire, quindi, interfacce utente con fragment durante la fase di progettazione, è anche possibile aggiungerli in fase d'esecuzione, il che ci permette di sviluppare un'applicazione molto più dinamica (Figura 3.4). Nello specifico, mediante gli oggetti `FragmentManager` e `FragmentTransaction`, metteremo in pratica questo concetto a livello di codice.

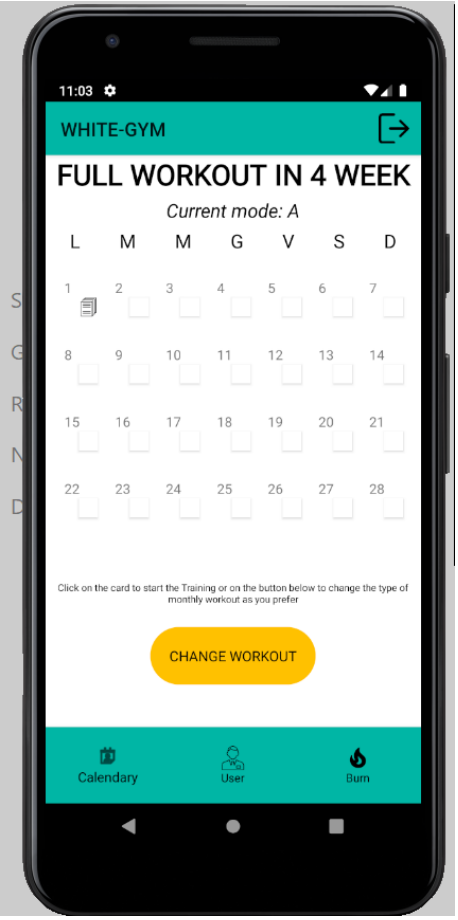

Figura 3.4. Istantanea di un esempio di fragment utilizzato

3.1.4 Adapter e RecyclerView

Le app spesso mostrano dati in sequenza o in qualche altra forma complessa. Qualunque sia il loro significato o la loro provenienza, una delle prime necessità del programmatore è quella di visualizzare tali dati tramite adapter e strumenti come la listview. Un adapter è un oggetto che fa da interfaccia fra un insieme di dati ed una vista. Per quanto riguarda, invece, la recyclerview (una particolare listview), parliamo di un costrutto che permette di mantenere in una coda, chiamata *recycler bin* (cestino per il riciclaggio), alcuni degli elementi precedentemente visualizzati, per riutilizzarli in un secondo momento anzichè creare tutti gli elementi della lista durante lo *scroll*. Infatti, durante lo scorrimento della lista, la recyclerview recupererà dalla coda un elemento e lo popolerà con le nuove informazioni da mostrare all'utente [6]. Forniamo un esempio pratico di questo concetto nella Figura 3.5.

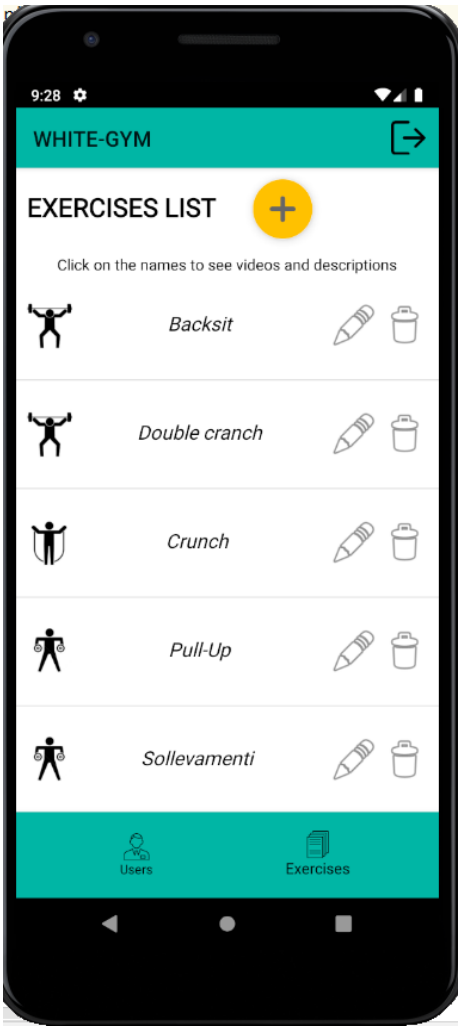

Figura 3.5. La lista di esercizi disponibili sviluppata con adapter e recyclerview

3.1.5 Bottom Navigation Bar

I menù sono fondamentali per quanto concerne la progettazione delle interfacce, poichè consentono all'utente di avere un rapido accesso alla maggior parte delle funzionalità dell'applicazione. A partire dalla actionbar, per app costruite a partire dalla Versione 11 dell'SDK di Android, si ha accesso ad aree all'interno della nostra schermata che possiamo definire come dei menù a tendina (Figura 3.6); noto che questi ultimi risultano essere molto più puliti dei menù classici di Figura 3.7. Nel nostro caso abbiamo optato per una BottomNavigationBar, ossia una barra di navigazione visibile sul fondo dell'interfaccia, volta a guidare in maniera molto intuitiva l'utente tra i servizi offerti.

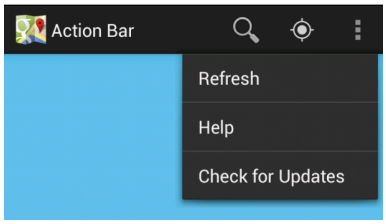

Figura 3.6. L'actionbar di Android

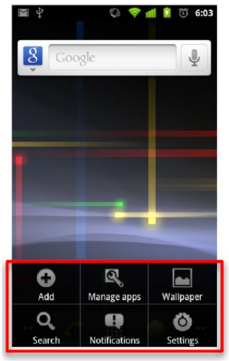

Figura 3.7. Un menù classico di Android

3.2 Diagramma dei casi d'uso

Il diagramma dei casi d'uso fa parte dei diagrammi UML (Unified Modelling Language), con i quali rappresentiamo sistemi e processi della programmazione orientata ad oggetti. UML quindi non è un linguaggio di programmazione, ma di modellazione; esso può anche essere visto come un metodo standardizzato che descrive un sistema pianificato o già esistente, mediante diagrammi in cui tutti i singoli oggetti sono inclusi, strutturati e relazionati tra di loro. Con un diagramma dei casi d'uso vengono rappresentate le funzionalità di un'applicazione, dal punto di vista dell'utilizzatore (trattato in UML come "attore"). Questo attore non deve necessariamente essere un utente umano; il ruolo può anche essere attribuito al sistema esterno in determinate situazioni. Tale diagramma rappresenta, quindi, la relazione tra un attore e le sue richieste o aspettative rispetto al sistema. Il diagramma dei casi d'uso della nostra applicazione è mostrato nelle Figure 3.8 e 3.9. Dal punto di vista pratico, l'impiego di tali diagrammi è finalizzato a rappresentare le funzioni e gli obiettivi principali di un sistema, nella maniera più chiara possibile. Per questo motivo, nello sviluppo dei software o nella pianificazione di nuovi processi aziendali, spesso uno dei primi passi è quello di creare un diagramma dei casi d'uso. Esso, infatti, evidenzia in maniera semplice ed intuitiva quali "casi d'uso" debbano essere tenuti in considerazione nella fase di sviluppo, in modo che gli attori raggiungano il proprio obiettivo senza tener conto della fattibilità in chiave tecnica [16].

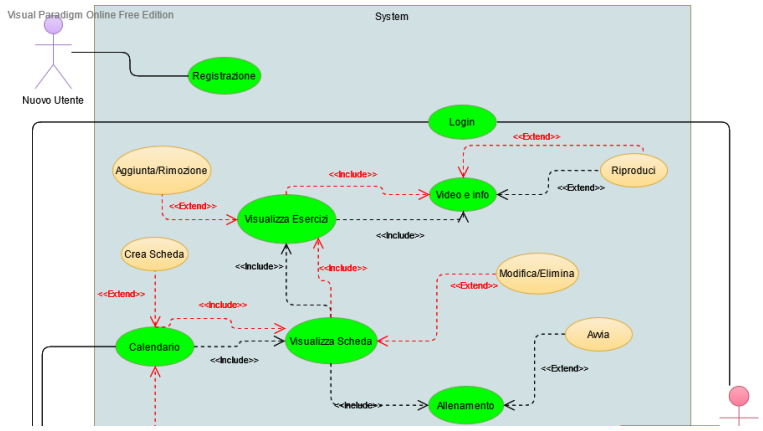

Figura 3.8. Il diagramma dei casi d'uso dell'applicazione (1)

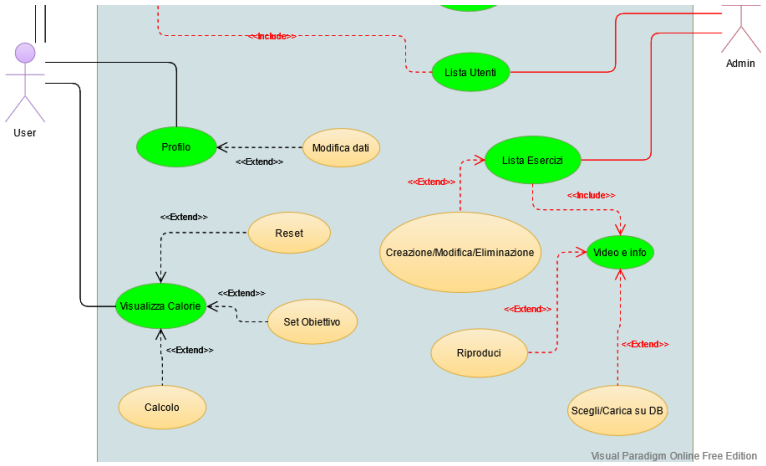

Figura 3.9. Il diagramma dei casi d'uso dell'applicazione (2)

3.2.1 Descrizione dei casi d'uso

Dopo aver introdotto il diagramma dei casi d'uso approfondiamo il concetto di singolo caso d'uso, ossia l'azione che può essere svolta dall'utilizzatore della nostra applicazione. Le caratteristiche principali di un caso d'uso correttamente individuato sono le seguenti:

- *coerenza con gli attori*: questa azione deve avere un senso compiuto per gli attori, nel senso che consente loro di raggiungere un obiettivo significativo;
- *semplicità*: un caso d'uso deve essere elementare, cioè non scomponibile in casi d'uso più semplici che abbiano ancora senso compiuto per gli attori coinvolti;
- *chiarezza*: non devono sorgere conflitti tra i singoli casi d'uso.

3.2.2 Rappresentazione delle funzionalità in casi d'uso

Per poter specificare un singolo caso d'uso necessitiamo di alcuni parametri fondamentali, ovvero:

- *Nome*: nome del caso d'uso.
- *Attori*: principali e (all'evenienza) secondari.
- *ID*: codice univoco per evitare conflitti tra i singoli elementi.
- *Obiettivo*: motivo di avvio del caso d'uso da parte dell'attore.
- *Precondizione*: situazione nella quale è eseguibile questa azione.
- *Postcondizioni*: guardie aggiuntive nell'eventualità di controlli su determinate componenti.
- *Flusso degli eventi*: sequenza delle azioni svolte dagli attori e dal sistema, visto come una scatola nera, cioè senza entrare nel dettaglio del suo funzionamento intrinseco.

A questo punto, analizziamo in dettaglio ciascuno dei casi d'uso relativi al nostro sistema.

Registrazione

- *Nome*: Registrazione.
- *Attori*: utente non registrato.
- *ID*: n1.
- *Obiettivo*: permettere ad un nuovo utente di registrarsi all'interno della nostra applicazione.
- *Precondizione*: il cliente deve aver installato l'app sul suo dispositivo.
- *Postcondizioni*: l'utente viene prima registrato sul database e poi accede alla schermata iniziale.
- *Flusso degli eventi*:
  1. L'applicazione è stata correttamente avviata.
  2. L'utente ha cliccato sul pulsante "Registrati".
  3. L'utente ha compilato senza errori la form di registrazione.
  4. L'utente ha cliccato sul pulsante "Conferma" (Figura 3.10).

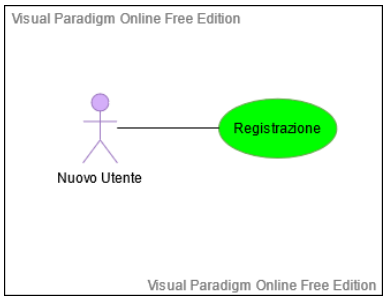

Figura 3.10. Caso d'uso relativo alla registrazione

Login

- *Nome*: Login.
- *Attori*: user e admin.
- *ID*: n2.
- *Obiettivo*: permettere ad un utente registrato di accedere alla nostra applicazione.
- *Precondizione*: il cliente deve aver completato la registrazione.
- *Postcondizioni*: l'utente viene riconosciuto e successivamente accede alla schermata iniziale.
- *Flusso degli eventi*:
  1. L'applicazione è stata correttamente avviata.
  2. L'utente ha cliccato sul pulsante "Accedi".
  3. L'utente ha compilato senza errori i campi email e password.
  4. L'utente ha cliccato sul pulsante "Conferma" (Figura 3.11).

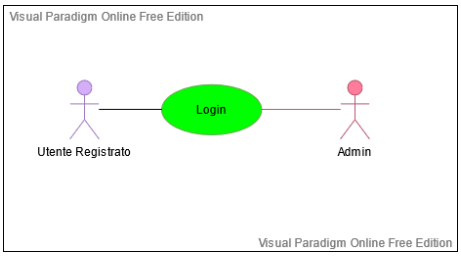

Figura 3.11. Caso d'uso relativo al login

Profilo

- *Nome*: Profilo.
- *Attori*: user.
- *ID*: n3.
- *Obiettivo*: permettere all'utente di accedere alla sua area privata e modificare, all'evenienza, alcuni dei suoi dati.
- *Precondizione*: l'utente deve aver effettuato il login.
- *Postcondizioni*: l'utente visualizza la schermata profilo con i suoi dati principali.
- *Flusso degli eventi*:
  1. L'utente ha effettuato il login correttamente.
  2. L'utente ha visualizzato la schermata relativa al calendario.
  3. L'utente ha cliccato sull'icona "Profilo" della BottomBar (Figura 3.12).

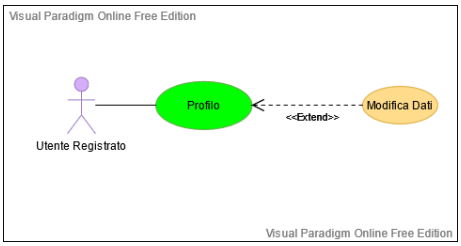

Figura 3.12. Caso d'uso relativo al profilo

Calorie

- *Nome*: Calorie.
- *Attori*: user.
- *ID*: n4.
- *Obiettivo*: permettere all'utente di accedere alla schermata dedicata alle calorie bruciate.
- *Precondizione*: l'utente deve aver effettuato il login.
- *Postcondizioni*: l'utente visualizza la schermata relativa alle calorie bruciate con i valori aggiornati, recupati dal database.
- *Flusso degli eventi*:
  1. L'utente ha effettuato il login correttamente.
  2. L'utente ha visualizzato la schermata calendario.
  3. L'utente ha cliccato sull'icona "Burn" della BottomBar (Figura 3.13).

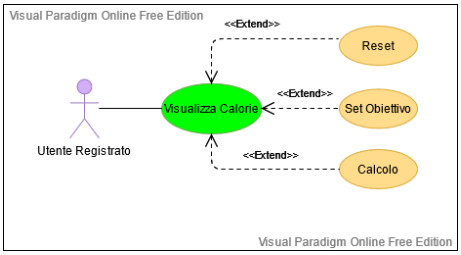

Figura 3.13. Caso d'uso relativo alle calorie

Calendario

- *Nome*: Calendario.
- *Attori*: user e admin.
- *ID*: n5.
- *Obiettivo*: permettere all'utente/admin di accedere alla schermata calendario.
- *Precondizione*: l'utente/admin deve aver effettuato il login.

- *Postcondizioni*: l'utente visualizza la schermata calendario con tutte le schede di allenamento. L'admin visualizza la stessa schermata, con la possibilità di aggiungere schede all'interno delle giornate.
- *Flusso degli eventi*:
  - Nel caso in cui l'utente sia uno *user*:
    1. L'utente ha effettuato il login correttamente.
    2. L'utente ha visualizzato la schermata calendario (Figura 3.14).
  - Nel caso in cui l'utente sia un *admin*:
    1. L'admin ha visualizzato la lista degli utenti.
    2. L'admin ha selezionato un utente dalla lista.
    3. L'admin ha visualizzato la schermata calendario.

*L'admin estende funzionalità diverse da quelle dello *user* come l'aggiunta delle schede (Figura 3.15).

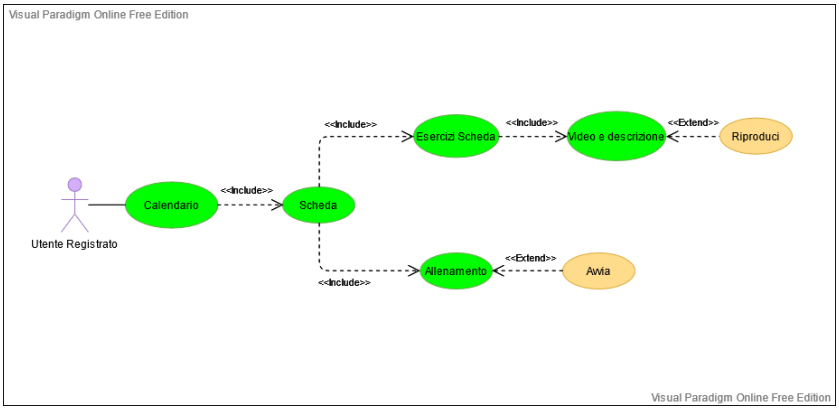

Figura 3.14. Il diagramma dei casi d'uso a disposizione dell'utente

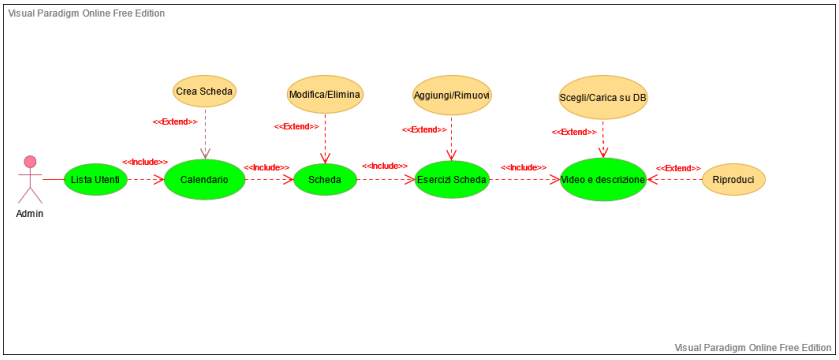

Figura 3.15. Il diagramma dei casi d'uso a disposizione dell'admin

Scheda giornaliera

- *Nome*: Visualizza scheda.
- *Attori*: user e admin.
- *ID*: n6.
- *Obiettivo*: permettere all'utente/admin di accedere all'interfaccia relativa alla scheda che ha selezionato.
- *Precondizione*: l'utente/admin deve aver selezionato e cliccato su una scheda presente all'interno della schermata calendario.
- *Postcondizioni*: l'utente/admin visualizza la scheda selezionata e i relativi pulsanti d'interazione (differenti).
- *Flusso degli eventi*:
  - Nel caso in cui l'utente sia uno *user*:
    1. L'utente ha visualizzato il calendario.
    2. L'utente ha cliccato su una scheda (Figura 3.14).
  - Nel caso in cui l'utente sia un *admin*:
    1. L'admin ha visualizzato la lista degli utenti.
    2. L'admin ha selezionato un utente dalla lista.
    3. L'admin ha visualizzato il calendario dell'utente.
    4. L'admin ha cliccato su una scheda dell'utente.

*L'admin estende funzionalità diverse da quelle dello *user* come la creazione/-modifica/eliminazione delle schede. (Figura 3.15)

Esercizi in una scheda

- *Nome*: Visualizza esercizi all'interno di una scheda.
- *Attori*: user e admin.
- *ID*: n7.
- *Obiettivo*: permettere all'utente/admin di visualizzare gli esercizi presenti all'interno di una scheda.
- *Precondizione*: l'utente/admin deve aver cliccato sul nome della scheda aperta.
- *Postcondizioni*: l'utente visualizza la lista degli esercizi dentro alla scheda.
- *Flusso degli eventi*:
  - Nel caso in cui l'utente sia uno *user*:
    1. L'utente ha visualizzato il calendario.
    2. L'utente ha cliccato su una scheda.
    3. L'utente ha cliccato sul nome della scheda (Figura 3.14).
  - Nel caso in cui l'utente sia un *admin*:
    1. L'admin ha visualizzato la lista degli utenti.
    2. L'admin ha selezionato un utente dalla lista.
    3. L'admin ha visualizzato il calendario dell'utente.
    4. L'admin ha cliccato su una scheda dell'utente.
    5. L'admin ha cliccato sul nome della scheda.

*L'admin estende funzionalità diverse da quelle dello *user* come l'aggiunta/rimozione degli esercizi (Figura 3.15).

Video e descrizione

- *Nome*: Visualizza video e descrizione di un esercizio.
- *Attori*: user e admin.
- *ID*: n8.
- *Obiettivo*: permettere all'utente/admin di visualizzare la pagina di dettaglio del singolo esercizio.
- *Precondizione*: l'utente/admin deve aver cliccato sul nome dell'esercizio.
- *Postcondizioni*: l'utente visualizza la descrizione e un video dell'esercizio.
- *Flusso degli eventi*:
  - Nel caso in cui l'utente sia uno *user*:
    1. L'utente ha visualizzato il calendario.
    2. L'utente ha cliccato su una scheda.
    3. L'utente ha cliccato sul nome della scheda.
    4. L'utente ha cliccato sul nome dell'esercizio all'interno della scheda (Figura 3.14).
  - Nel caso in cui l'utente sia un *admin*:
    1. L'admin ha visualizzato la lista degli utenti.
    2. L'admin ha selezionato un utente dalla lista.
    3. L'admin ha visualizzato il calendario dell'utente.
    4. L'admin ha cliccato su una scheda dell'utente.
    5. L'admin ha cliccato sul nome della scheda.
    6. L'admin ha cliccato sul nome dell'esercizio all'interno della scheda.

*L'admin estende funzionalità diverse da quelle dello *user* come il caricamento di un video sul database (Figura 3.15).

Allenamento

- *Nome*: Visualizza allenamento.
- *Attori*: user.
- *ID*: n9.
- *Obiettivo*: permettere all'utente di visualizzare l'interfaccia d'allenamento, relativa alla scheda selezionata.
- *Precondizione*: l'utente deve aver cliccato su una scheda all'interno del calendario.
- *Postcondizioni*: l'utente visualizza la schermata di allenamento desiderata.
- *Flusso degli eventi*:
  1. L'utente ha visualizzato il calendario.
  2. L'utente ha cliccato su una scheda.
  3. L'utente ha cliccato sull'icona di avvio dell'allenamento (Figura 3.14).

Lista degli utenti registrati

- *Nome*: Visualizza la lista degli utenti registrati.
- *Attori*: admin.
- *ID*: n10.

- *Obiettivo*: permettere all'admin di visualizzare la lista aggiornata degli utenti caricati sul database.
- *Precondizione*: l'admin deve aver effettuato il login come tale.
- *Postcondizioni*: l'admin visualizza la lista integrale degli utenti sui quali può interagire, semplicemente cliccando sui nomi.
- *Flusso degli eventi*:
  1. L'admin ha effettuato correttamente il login (Figura 3.15).

Lista degli esercizi disponibili

- *Nome*: Visualizza la lista degli esercizi disponibili.
- *Attori*: admin.
- *ID*: n11.
- *Obiettivo*: permettere all'admin di visualizzare la lista aggiornata di esercizi caricati sul database.
- *Precondizione*: l'admin deve aver effettuato il login come tale.
- *Postcondizioni*: l'utente visualizza la lista integrale degli esercizi, sui quali può effettuare operazioni di modifica/eliminazione, oppure crearne di nuovi.
- *Flusso degli eventi*:
  1. L'admin ha visualizzato la lista degli utenti (la sua schermata principale).
  2. L'utente ha cliccato sull'icona "Esercizi" della BottomNavigationBar (Figura 3.15).

Logout

- *Nome*: Logout.
- *Attori*: user e admin.
- *ID*: n12.
- *Obiettivo*: permettere all'utente/admin di effettuare il logout dal sito.
- *Precondizione*: l'utente deve aver effettuato il login o la registrazione.
- *Postcondizioni*: nessuna.
- *Flusso degli eventi*:
  1. L'utente/admin clicca sull'icona di logout (Figura 3.16).

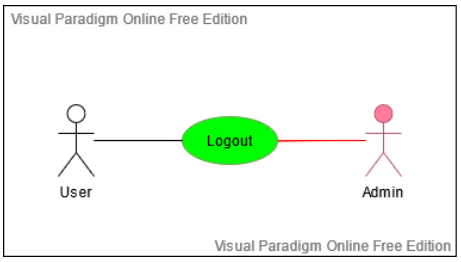

Figura 3.16. Caso d'uso relativo al logout

3.3 Firebase Database

Firebase offre un servizio online che permette di salvare e sincronizzare i dati elaborati da applicazioni web e mobile. Si tratta di un database NoSQL dalle grandissime risorse, ad alta disponibilità ed integrabile in tempi rapidissimi all'interno dei nostri progetti software, semplicemente sottoscrivendo un account al servizio. Nello specifico, i database NoSQL sono appositamente realizzati per modelli di dati specifici e hanno schemi flessibili per creare applicazioni moderne [12]. Tali database si sono affermati per la facilità di sviluppo, la funzionalità e l'ottimizzazione delle prestazioni. Lo sviluppo della nostra applicazione è avvenuto, quindi, in parallelo alla gestione dei dati da parte del Firebase Database e del suo Storage. Il database dell'app è stato strutturato in due collection, ovvero:

- `users`.
- `Esercizi`.

Nelle prossime sottosezioni esaminiamo più nel dettaglio tali collection.

3.3.1 Utenti

La collection `users`, come suggerisce il nome, contiene tutti gli utenti registrati all'interno dell'app. Per ciascuno di essi, Firebase genera automaticamente un codice identificativo che caratterizza la singola instanza d'utente, della quale verranno infine creati e popolati i relativi campi. Nel seguito forniamo una rappresentazione schematica di tali campi, per avere davanti agli occhi una struttura chiara sulla quale lavorare (Figura 3.17 e 3.18):

- `admin`: è la stringa che permette di scindere un utente amministratore (con valore "admin") da un cliente (con valore "noadmin").
- `email`: è la stringa che rappresenta l'email dell'utente, sia user che admin; la quale verrà utilizzata come chiave d'accesso al momento del login.
- `firstname`: nome dell'utente registrato.
- `lastname`: cognome dell'utente registrato.
- `phone`: stringa relativa al numero di telefono inserito dall'utente.
- `uri`: è la stringa che definisce l'URI estratto dall'immagine profilo associata all'utente (di default si fa riferimento alla stringa di un'immagine salvata nello storage di Firebase).
- `goal`: è la grandezza numerica che indica l'obiettivo, in termini di calorie bruciate, che un utente vuole raggiungere (di default viene posto al valore 500 Kcal).
- `totcal`: è un'altra grandezza numerica la quale descrive il progresso, in termini di calorie bruciate, che l'utente ha perseguito con gli esercizi svolti.

3.3.2 Schede di allenamento

Ogni utente ha, inoltre, una serie di raccolte che rappresentano i giorni di allenamento. Per ogni raccolta abbiamo più documenti i quali, presi singolarmente, rappresentano la scheda giornaliera che è identificata col suo campo `path`. Per ogni scheda abbiamo, infatti, tre campi (Figura 3.19):

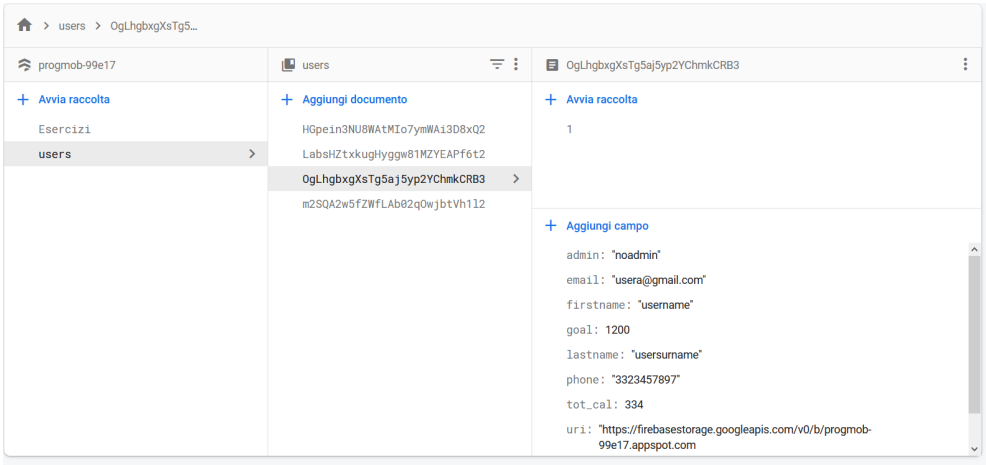

Figura 3.17. Istantanea del database rappresentante l'user

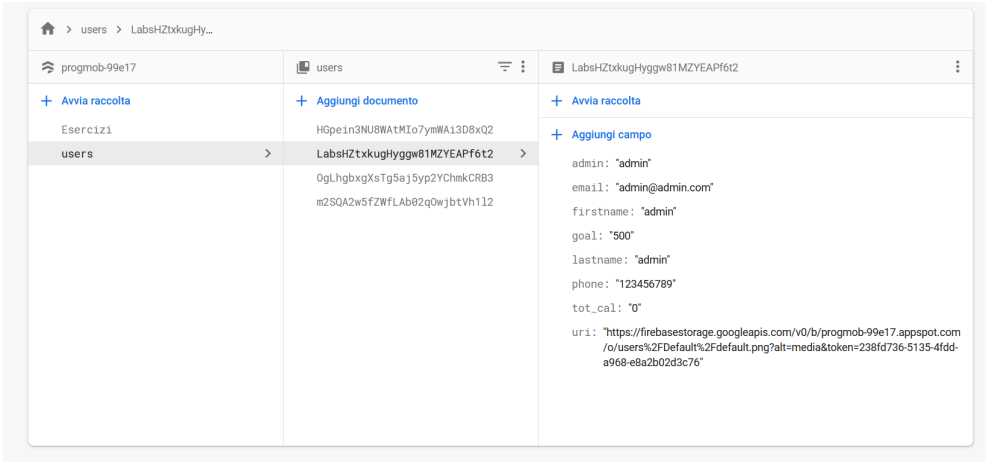

Figura 3.18. Istantanea del database rappresentante l'admin

- `path`: il nome della scheda.
- `ref`: il riferimento all'ID dell'utente associato alla scheda.
- `type`: la tipologia di allenamento della scheda.

3.3.3 Esercizi di una scheda

All'interno di ogni documento scheda abbiamo la raccolta "ExSchede", la quale rappresenta l'insieme degli esercizi presenti nella scheda; ogni esercizio ha i medesimi campi degli elementi della collection `Esercizi` (Figura 3.20).

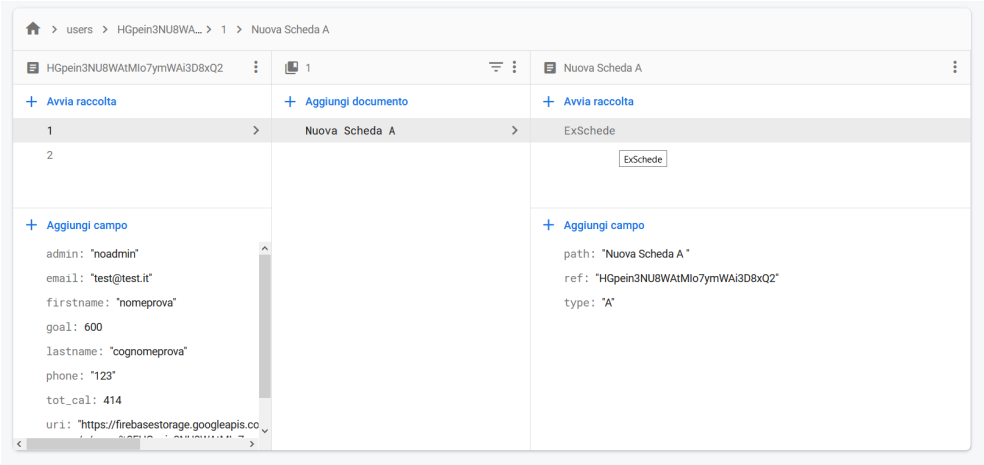

Figura 3.19. Istantanea del database rappresentante una scheda di allenamento

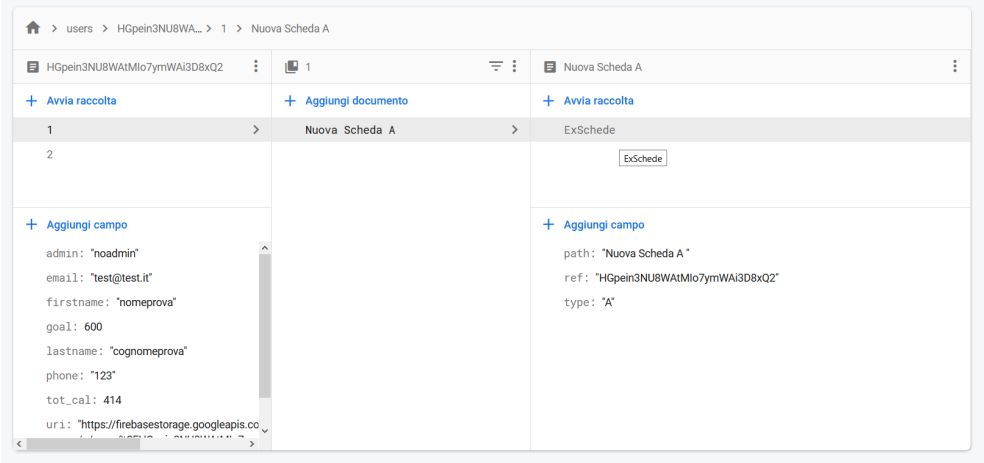

Figura 3.20. Istantanea del database rappresentante `ExSchede` con esercizi all'interno

3.3.4 Esercizi

Infine, la collection `Esercizi` viene utilizzata per catalogare gli esercizi, i quali verranno inseriti all'interno dell'app, a seguito della creazione da parte dell'admin. Ciascun esercizio è caratterizzato da cinque campi (Figura 3.21):

- `name`, il nome dell'esercizio.
- `difficulty`, la tipologia e difficoltà dell'esercizio.
- `description`, una breve descrizione riassuntiva.
- `cal`, il numero di calorie bruciate svolgendo tale esercizio.
- `uri`, il riferimento all'URI relativo al video associato all'esercizio.

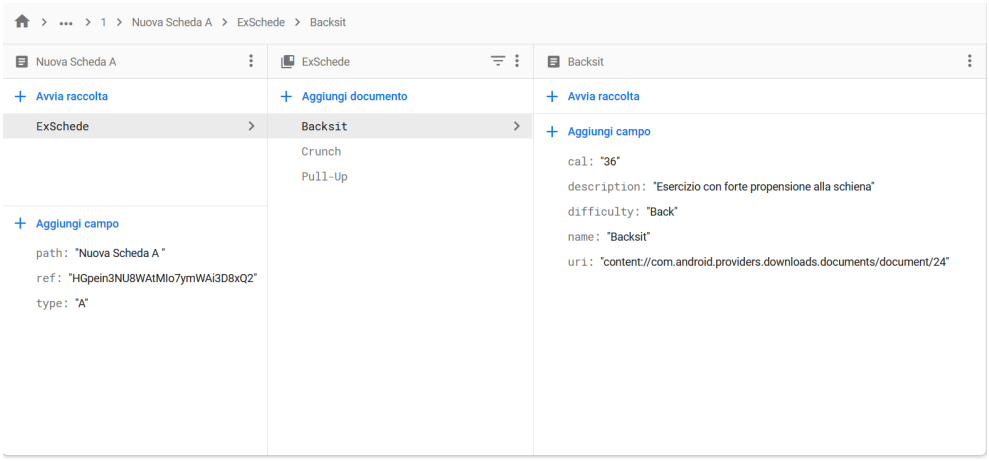

Figura 3.21. Istantanea del database rappresentante un esercizio dentro `ExSchede`

3.3.5 Firebase Storage

Firebase Storage offre un modo semplice per salvare i file. Nel nostro caso tali file consistono in immagini e video tuttavia, Firebase è in grado di gestire la maggior parte dei file multimediali esistenti. Firebase Storage ha il proprio sistema di regole per proteggere l'archivio associato all'applicazione, garantendo, allo stesso tempo, anche i privilegi di scrittura, a particolari utenti autenticati (Figura 3.22). Nel nostro caso abbiamo utilizzato due cartelle per i nostri file, ovvero:

- `exercises`: è la cartella contenente i video associati agli esercizi (Figura 3.23).
- `users`: è la cartella contentente le immagini profilo associate agli utenti (Figura 3.24).

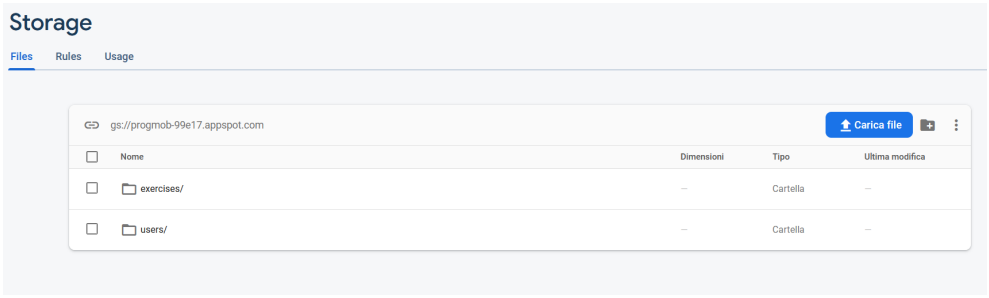

Figura 3.22. Istantanea dello storage di Firebase

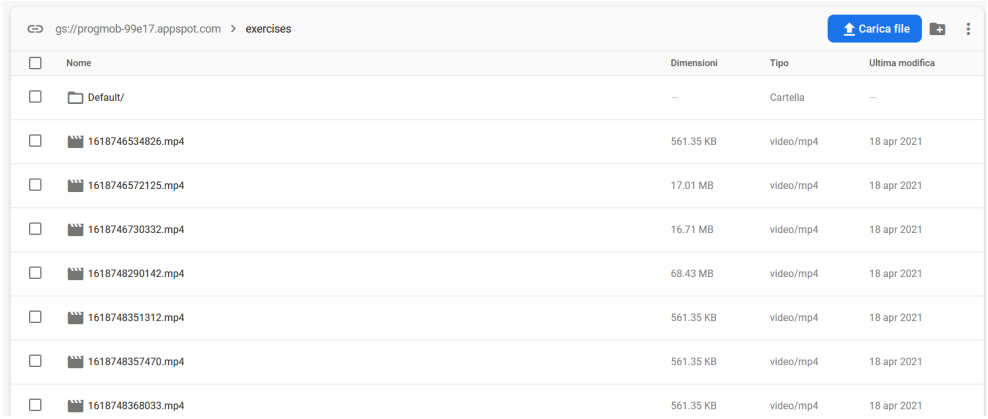

Figura 3.23. Istantanea dell'interno della cartella `exercises`

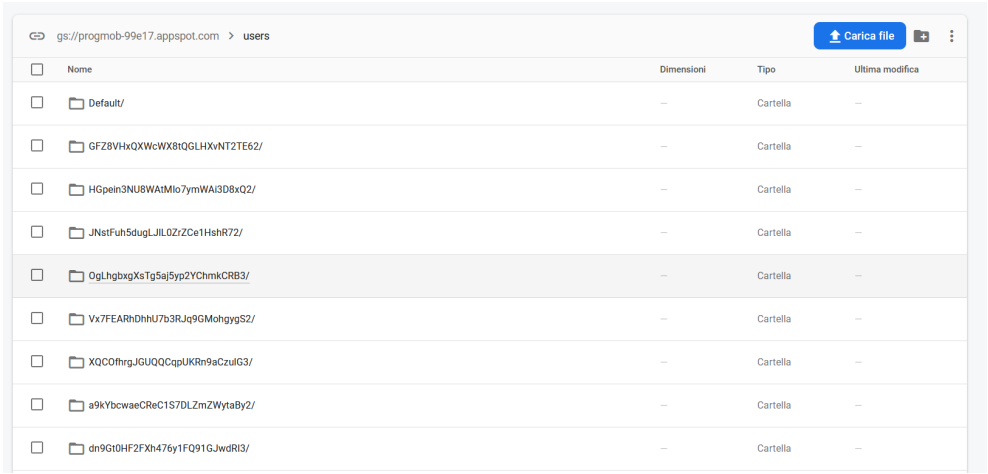

Figura 3.24. Istantanea dell'interno della cartella `users`

4

Implementazione dell'app e manuale utente

In questo capitolo illustreremo le modalità e i costrutti adoperati per implementare le funzionalità precedentemente descritte. A seguire presenteremo un manuale utente dettagliato, il quale fungerà da guida per coloro che vogliono affacciarsi allo studio o all'utilizzo della nostra app.

4.1 Implementazione dell'app

In questa prima parte ci soffermiamo sull'implementazione delle funzionalità richieste. Per ogni servizio mostreremo le parti salienti del codice, accompagnate da un'esaustiva spiegazione.

4.1.1 Registrazione

La prima funzionalità che incontriamo, avviando per la prima volta la nostra applicazione, è quella di registrazione. La sua implementazione è stata possibile mediante l'impiego delle seguenti librerie di Firebase:

- `FirebaseAuth`.
- `FirebaseUser`.
- `UserProfileChangeRequest`.
- `FirebaseFirestore`.
- `AuthResult`.

Il metodo che è stato invocato è `createUserWithEmailAndPassword(email,password)`, il quale come suggerisce il nome, permette di creare un nuovo utente fornendo in input un'email ed una password da una form dedicata (Listato 4.1).

```
fAuth.createUserWithEmailAndPassword(email,password).addOnCompleteListener(new OnCompleteListener<AuthResult>(){
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
    if(task.isSuccessful()) {
        final FirebaseUser user = fAuth.getCurrentUser();
            UserProfileChangeRequest profileChangeRequest = new UserProfileChangeRequest.Builder()
            .setDisplayName(mfirstname + "␣" + mlastname)
            .build();
            user.updateProfile(profileChangeRequest).addOnCompleteListener(new OnCompleteListener<Void>() {
                @Override
```

```
                public void onComplete(@NonNull Task<Void> task) {
                writeUserToDb(firstname, lastname, phone, email, user.getUid(), goal, real_goal);
                Intent intent = new Intent();
                intent.putExtra("firstname", mfirstname.getText().toString());
                intent.putExtra("lastname", mlastname.getText().toString());
                setResult(RESULT_OK, intent);
                finish();
                }
            });
        Toast.makeText(Register.this, "User Created", Toast.LENGTH_SHORT).show();
        startActivity(new Intent(getApplicationContext(), SplashScreen.class));
        finish();
    } else {
            Toast.makeText(Register.this, "Error !" + task.getException().getMessage(), Toast.LENGTH_SHORT).show();
            }
        }
    });
```

Listato 4.1. Codice del metodo per la registrazione

Il metodo successivo `writeUserToDb(String firstname, String lastname, String phone, String email, final String uid, String totcal,String g)` permette di creare, a livello di database, una mappa (stringa, oggetto) con un numero di campi pari a quelli inseriti in fase di registrazione dall'utente stesso, assieme ad altri campi predefiniti e popolati di default (`admin`, `totcal`, `goal`, `uri`). Di particolarmente notevole c'è il meccanismo di assegnazione dell'immagine del profilo ai nuovi utenti, il quale fa riferimento al file del percorso `"/users/Default/default.png"` dello storage di Firebase (Listato 4.2).

```
private void writeUserToDb(String firstname, String lastname, String phone, String email, final String uid,
String tot_cal, String goal) {
        final Map<String, Object> user = new HashMap<>();
        user.put("firstname", firstname);
        user.put("lastname", lastname);
        user.put("phone", phone);
        user.put("email", email);
        user.put("admin", "noadmin");
        user.put("tot_cal", 0);
        user.put("goal", 500);
        StorageReference storageRef = FirebaseStorage.getInstance().getReference();
        StorageReference uriStorage = storageRef.child("/users/Default/default.png");
        uriStorage.getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
            @Override
            public void onSuccess(Uri uri) {
                Log.d("URI", uri.toString());
                user.put("uri", uri.toString());
                FirebaseFirestore db = FirebaseFirestore.getInstance();
                db.collection("users")
                        .document(uid)
                        .set(user);
            }
        });
}
```

Listato 4.2. Codice del metodo per registrare l'utente sul database

4.1.2 Login

Logicamente, se si ha la possibilità di registrarsi al primo avvio dell'applicazione, si ha anche quella di accedere come utente (non *admin*) per le volte a seguire. Il metodo più significativo è `fAuth.signInWithEmailAndPassword(email, password)`, il quale permette all'utente, fornendo l'email e la password inseriti in fase di registrazione, di eseguire correttamente l'accesso al fragment di competenza (Listato 4.3).

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

```
                    if (task.isSuccessful()) {
                        Toast.makeText(Login.this, "Logged␣in␣successfully", Toast.LENGTH_SHORT).show();
                        startActivity(new Intent(getApplicationContext(), SplashScreen.class));
                        finish();
                    } else {
                        Toast.makeText(Login.this, "Error␣!" + task.getException().getMessage(),
                        Toast.LENGTH_SHORT).show();
                    }
                }
            });
```

Listato 4.3. Codice del metodo per il login

Abbiamo anche garantito il servizio di recupero della password in caso di smarrimento, mediante la funzione `fAuth.sendPasswordResetEmail(mail)`, la quale invierà automaticamente un'email di reset all'indirizzo di posta elettronica inserito (Listato 4.4).

```
final EditText resetMail = new EditText(v.getContext());
            AlertDialog.Builder passwordResetDialog = new AlertDialog.Builder(v.getContext());
            passwordResetDialog.setTitle("Reset␣password?");
            passwordResetDialog.setMessage("Enter␣your␣Email␣to␣receive␣Reset␣Link");
            passwordResetDialog.setView(resetMail);

            passwordResetDialog.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    // extract the email and reset link
                    String mail = resetMail.getText().toString();
                    fAuth.sendPasswordResetEmail(mail).addOnSuccessListener(new OnSuccessListener<Void>() {
                        @Override
                        public void onSuccess(Void aVoid) {
                            Toast.makeText(Login.this, "Reset␣Link␣sent␣to␣your␣Email",
                            Toast.LENGTH_SHORT).show();
                        }
                    }).addOnFailureListener(new OnFailureListener() {
                        @Override
                        public void onFailure(@NonNull Exception e) {
                            Toast.makeText(Login.this, "Error!␣Reset␣Link␣is␣not␣sent" +
                            e.getMessage(),Toast.LENGTH_SHORT).show();
                        }
                    });
                }
            });
```

Listato 4.4. Codice del metodo per il reset della password

4.1.3 SplashScreen

Mediante l'impiego della libreria `FirebaseAuth` abbiamo sviluppato una schermata di attesa, alla quale si giunge ogni volta che vengono completate le operazioni di accesso/registrazione (Listato 4.5). Nel caso in cui viene rilevata l'istanza di un utente registrato da `fAuth = FirebaseAuth.getInstance()` e da `if(fAuth.getCurrentUser() != null)`, allora tale schermata ci indirizza verso la `MainActivity`; in caso contrario, incontriamo la form di login. Invece, nel caso in cui il reindirizzamento fallisse, il sistema ci restituirà un'eccezione del tipo `SPLASH TIME OUT`.

```
public class SplashScreen extends AppCompatActivity {
    FirebaseAuth fAuth;
    private static int SPLASH_TIME_OUT = 4000;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.splash_screen);

       fAuth = FirebaseAuth.getInstance();

            new Handler().postDelayed(new Runnable(){
            @Override
```

```
            public void run() {
                if(fAuth.getCurrentUser() != null) {
                    startActivity(new Intent(SplashScreen.this,MainActivity.class));
                    finish();
                }
                else {
                    startActivity(new Intent(SplashScreen.this,Login.class));
                    finish();
                }
            }
        },SPLASH_TIME_OUT);
    }
}
```

Listato 4.5. Codice che definisce la classe Java `SplashScreen`

4.1.4 MainActivity

Per lo sviluppo della `MainActivity` ribadiamo di aver utilizzato i fragment, ossia delle porzioni di schermata con un proprio ciclo di vita che possono essere inseriti o rimossi sia in fase di progettazione sia, dinamicamente, in fase di esecuzione. Quindi, le prime librerie che sottolineiamo sono quelle per l'utilizzo degli stessi frammenti, ovvero:

- `androidx.fragment.app.Fragment`.
- `androidx.fragment.app.FragmentManager`.
- `androidx.fragment.app.FragmentTransaction`.

Utilizziamo `MainActivity` come contenitore dei frammenti (o `Fragment Container`), mentre trattiamo tutte le altre interfacce come singoli frammenti con un proprio adapter, i quali saranno, poi, caricati all'interno della schermata principale a seconda di vari fattori (Listato 4.6). Le librerie di Firebase importate sono:

- `FirebaseAuth`.
- `DocumentReference`.
- `FirebaseFirestore`.

Con molta facilità possiamo osservare l'implementazione del controllo sulla tipologia d'utente (*user* o *admin*), a partire dal campo ricavato dalla riga `final String user type = String.valueOf(task.getResult().get("admin"))`. Abbiamo, infatti, sviluppato una guardia di controllo `if((user type.equals("admin")))`, la quale, nel caso in cui incontra un valore *admin*, predispone i frammenti per l'amministratore; altrimenti, prepara quelli per lo *user*. Inoltre, per quanto riguarda la barra di navigazione, abbiamo importato la libreria `BottomNavigationView`, che ci garantisce tutte le funzionalità necessarie. Questa barra, a seconda del tipo di utente, viene modificata con i frammenti che possono essere raggiunti dall'uno e dall'altro.

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        context = getApplicationContext();
        setContentView(R.layout.activity_main);
        actionBar=getSupportActionBar();
        actionBar.setBackgroundDrawable(new ColorDrawable(Color.parseColor("#01b6a5")));

        if(Build.VERSION.SDK_INT>=21){
            window=this.getWindow();
            window.setStatusBarColor(this.getResources().getColor(R.color.black));
            //window.setStatusBarColor(this.getResources().getColor(R.color.primaryDark));
        }
```

```
fragmentCards = new FragmentCards();
fragmentUser = new FragmentUser();
fragmentInfo = new FragmentInfo();
fragmentAdmin = new FragmentAdmin();
fragmentAll = new FragmentAll();
fragmentCalendary = new FragmentCalendary();

fAuth = FirebaseAuth.getInstance();
fStore = FirebaseFirestore.getInstance();
userId = fAuth.getCurrentUser().getUid();

DocumentReference docRef = fStore.collection("users").document(fAuth.getCurrentUser().getUid());
docRef.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {
    @Override
    public void onComplete(@NonNull Task<DocumentSnapshot> task ) {
        if(task.isComplete()){
            final String user_type = String.valueOf(task.getResult().get("admin"));
            final String user_goal = String.valueOf(task.getResult().get("tot_cal"));
            final String user_name = String.valueOf(task.getResult().get("firstname"));
            final String user_lastname = String.valueOf(task.getResult().get("lastname"));
            final String user_phone = String.valueOf(task.getResult().get("phone"));
            final String user_uri = String.valueOf(task.getResult().get("uri"));
            final String user_real_goal = String.valueOf(task.getResult().get("goal"));
            final String user_email = String.valueOf(task.getResult().get("email"));

            if((user_type.equals("admin"))){
                getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container,
                fragmentAdmin).commit();
                bottomNav = findViewById(R.id.bottom_nav);
                bottomNav.inflateMenu(R.menu.bottom_menu_admin);
                bottomNav.setOnNavigationItemSelectedListener(new
                BottomNavigationView.OnNavigationItemSelectedListener() {
                    @Override
                    public boolean onNavigationItemSelected(@NonNull MenuItem item) {
                        Fragment selectedFragment = null;
                        switch(item.getItemId()) {
                            case R.id.menu_cards:
                                selectedFragment = fragmentAll;
                                break;
                            case R.id.menu_admin:
                                selectedFragment = fragmentAdmin;
                                break;
                        }
                        if(selectedFragment != null) {

                        getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container
                        ,selectedFragment).addToBackStack(null).commit();
                        }
                        return true;
                    }
                });
            } else {
                Bundle args = new Bundle();
                args.putString("type", "noadmin");
                args.putString("ABC", "A");
                args.putString("SelectedDay", "all");
                fragmentCalendary.setArguments(args);
                getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container,
                fragmentCalendary).commit();
                bottomNav = findViewById(R.id.bottom_nav);
                bottomNav.inflateMenu(R.menu.bottom_menu);
                bottomNav.setOnNavigationItemSelectedListener(new
                BottomNavigationView.OnNavigationItemSelectedListener() {
                    @Override
                    public boolean onNavigationItemSelected(@NonNull MenuItem item) {
                        Fragment selectedFragment = null;
                        switch(item.getItemId()) {
                            case R.id.menu_calendary:
                                selectedFragment = fragmentCalendary;
                                break;
                            case R.id.menu_user:
                                selectedFragment = fragmentUser;
                                break;
                            case R.id.menu_info:
                                selectedFragment = fragmentInfo;
                        }
                        if(selectedFragment != null) {
                            Bundle args = new Bundle();
                            args.putString("admin", user_type);
                            args.putString("type", "noadmin");
                            args.putString("ABC", "A");
                            args.putString("goal", user_real_goal);
                            args.putString("tot_cal", user_goal);
                            args.putString("firstname", user_name);
                            args.putString("lastname", user_lastname);
                            args.putString("uri", user_uri);
                            args.putString("phone", user_phone);
                            args.putString("email", user_email);

                            selectedFragment.setArguments(args);
                            getSupportFragmentManager().beginTransaction().replace(R.id.fragment_conta
                            iner, selectedFragment).addToBackStack(null).commit();
                        }
```

```
                        return true;
                    }
                });
            }

        } else {
            Log.d("tag", "onEvent: Document do not exists");
        }
    }
});
}
```

Listato 4.6. Codice del metodo `onCreate` della classe Java `Main Activity`

Infine, per quanto riguarda il passaggio da una schermata ad un'altra in termini di frammenti, tutto ciò è possibile grazie al metodo `getSupportFragmentManager() .beginTransaction().replace(R.id.fragment container,selectedFragment) .addToBackStack(null).commit()`, il quale dà la possibilità all'utente di recuperare il frammento precedente semplicemente premendo sul tasto *back* del dispositivo.

4.1.5 FragmentCalendary

La prima schermata visibile ad uno *user* è quella riguardante il suo calendario, all'interno del quale potrà accedere alle schede, oppure cambiare la tipologia di allenamento (Listato 4.7). L'*admin*, invece, ne avrà accesso passando prima per il frammento della lista degli utenti. Dal punto di vista del calendario, inizialmente effettuiamo il controllo sulla tipologia d'utente; questa viene recuperata a livello di database nella `MainActivity` e, successivamente, passata all'interfaccia con l'uso dei `Bundle`. Il recupero di tale informazione avviene mediante `this.getArguments().getString("type")`. Inoltre, lo sviluppo della lista di schede dello *user* avviene attraverso il metodo `DatabaseReferences.listCards(uid, String.valueOf(d+1))`, il quale si trova all'interno di un "for each" che scorre tutti i giorni del calendario. Infine viene popolata e resa visibile la recyclerview, assieme all'adapter associato, tutto ad opera di `recyclerView.setAdapter(calendaryAdap ter)`.

```
controlIfAd = this.getArguments().getString("type");
       if( controlIfAd.equals("admin")){ uid = this.getArguments().getString("u_id"); }
       else{
           uid = DatabaseReferences.getUser().getUid();
           Log.d("TAG" , uid);
       }

       type = this.getArguments().getString("ABC");
       days = fillDays(days, nday);
       curr_mode = view.findViewById(R.id.curr_mode);
       curr_mode.setText("Current mode: "+type);

       for(final Integer d : days){
           DatabaseReferences.listCards(uid, String.valueOf(d+1)).get().addOnCompleteListener(
               new OnCompleteListener<QuerySnapshot>() {
                   @Override
                   public void onComplete(@NonNull Task<QuerySnapshot> task) {
                       if (task.isSuccessful()) {
                           for (QueryDocumentSnapshot document : task.getResult()) {
                               Log.d("ATTUALE CICLO " + String.valueOf(d+1) , " document: " +
                               document.getData());
                               if(document.get("type").toString().equals(type)) {
                                   ifset[d] = true;
                                   break;
                               }else{
                                   ifset[d] = false;
                               }
                           }
                       }
                   }
               }).addOnSuccessListener(new OnSuccessListener<QuerySnapshot>() {
```

```
                @Override
                public void onSuccess(QuerySnapshot queryDocumentSnapshots) {
                    calendaryAdapter.notifyDataSetChanged();
                }
            });
        }
    changeType = view.findViewById(R.id.changeABC);
    recyclerView = view.findViewById(R.id.rv_calendary);
    recyclerView.setLayoutManager(new GridLayoutManager(getContext() , 7));
    calendaryAdapter = new CalendaryAdapter(getContext(), type , ifset ,
    this.getArguments().getString("type"), uid) ;
    recyclerView.setAdapter(calendaryAdapter);
```

Listato 4.7. Codice dell'`if` e del `for each` nel fragment calendario

In definitiva, uno *user* e l'*admin* visualizzano entrambi lo stesso calendario con una sola differenza: il secondo può creare una scheda d'allenamento per il primo semplicemente cliccando sull'icona di un giorno che deve inderogabilmente essere libero. La funzione che genera un nuovo elemento di tipo card è `CardsFunctions.createCard(name card.getText().toString(), type , uid, new Integer(position+1).toString())` dove i parametri passati sono il nome della card, il tipo di allenamento, l'ID dello *user* ed, infine, la posizione dell'elemento all'interno calendario. Una volta completato il processo, la nuova scheda sarà visibile ad entrambe le tipologie di utente (Listato 4.8).

```
if (controlAd.equals("admin")){
            holder.btnDay.setBackground(ResourcesCompat.getDrawable(context.getResources(),
            R.drawable.ic_plus, null));
            holder.btnDay.setOnClickListener(new View.OnClickListener() {
                   @Override
                   public void onClick(View v) {
                       // Creation of a new card in the database
                       try {
                           AlertDialog.Builder builder = new AlertDialog.Builder(context);

                           builder.setTitle("Create␣New␣Card");
                           View dialogView = inflater.inflate(R.layout.dialog_card_mod, null);

                           builder.setView(dialogView);

                           final EditText name_card = dialogView.findViewById(R.id.rename_card);

                           builder.setPositiveButton("Confirm", new DialogInterface.OnClickListener() {
                               @Override
                               public void onClick(DialogInterface dialog, int which) {
                                   if(name_card.getText().toString().length() > 30)
                                   {
                                       Toast.makeText(context, "Impossible␣to␣Create" , Toast.LENGTH_SHORT).show();
                                   }

                                   else{
                                       try {
                                           CardsFunctions.createCard(name_card.getText().toString(), type
                                           , uid, new Integer(position+1).toString());
                                           ifset[position] = true;
                                           notifyItemChanged(position,true);
                                           notifyItemChanged(position);
                                           notifyDataSetChanged();
                                           Toast.makeText(context, "Your␣card␣has␣been␣created",
                                           Toast.LENGTH_SHORT).show();
                                       } catch (Exception e) {
                                           Toast.makeText(context , "path␣" +
                                           name_card.getText().toString() + "␣type␣" + type + "␣uid" +
                                           uid + e.getMessage(), Toast.LENGTH_SHORT).show();
                                       }
                                   }
                               }
                           });
                           builder.setNegativeButton("Delete", new DialogInterface.OnClickListener() {
                               @Override
                               public void onClick(DialogInterface dialog, int which) {
                                   // Close dialog
                               }
                           });

                           builder.create().show();

                       } catch (Exception e) {
                           Toast.makeText(context, "Error!␣" + e.getMessage(),
                           Toast.LENGTH_SHORT).show();
                       }
```

```
                }
            });


        }
```

Listato 4.8. Codice dell'adapter associato al fragment calendario

4.1.6 FragmentUser

Attraverso la BottomNavigationBar lo *user* può passare al frammento relativo al profilo; nello specifico egli potrà visualizzare e modificare:

- *l'immagine del profilo*, dalla libreria `com.squareup.picasso.Picasso`;
- *i dati personali*, grazie all'importazione delle librerie di Firebase.

Immagine profilo

Lo *user* visualizzerà l'immagine del profilo in automatico, dopo che essa è stata scaricata dal database (Listato 4.10).

```
fAuth = FirebaseAuth.getInstance();
fStore = FirebaseFirestore.getInstance();
StorageReference profileRef = storageReference.child("users/"+fAuth.getCurrentUser().getUid()+"/profile.jpg");
try {
       profileRef.getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
           @Override
           public void onSuccess(Uri uri) {
               Picasso.get().load(uri).into(profileImage);
               fStore.collection("users")
                       .document(userId)
                       .update("uri", uri.toString());
           }
       });
       } catch (Exception e){
           Log.d("Do nothing", "");
       }
```

Listato 4.9. Codice del metodo per visualizzare l'immagine del profilo tramite la libreria `Picasso`

La modifica dell'immagine del profilo, invece, avviene mediante l'Override del metodo `onActivityResult(int requestCode, int resultCode, @Nullable Intent data)`. Una volta che l'immagine è stata selezionata, il suo `uri` viene caricato come stringa sul database, grazie ad `uploadImageToFirebase(Uri imageUri)`, come mostrato nel Listato 4.10.

```
profileImage.setOnClickListener(new View.OnClickListener(){
          @Override
          public void onClick(View v){
              // open gallery
              Intent openGalleryIntent = new Intent(Intent.ACTION_PICK,
              MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
              startActivityForResult(openGalleryIntent, 1000);

          }
      });

@Override
   public void onActivityResult(int requestCode, int resultCode, @Nullable Intent data){
       super.onActivityResult(requestCode, resultCode, data);

       if(resultCode != RESULT_CANCELED) {
       if(requestCode == 1000) {
           try{
               assert data != null;
```

```
                Uri imageUri = data.getData();
                uploadImageToFirebase(imageUri);
            } catch(Exception e){
                Toast.makeText(getContext(), "Failed.", Toast.LENGTH_SHORT).show();
            }

        }
        }else{
            Toast.makeText(getContext(), "Failed", Toast.LENGTH_SHORT).show();
        }

    }

    private void uploadImageToFirebase(Uri imageUri) {
        // upload image to firebase storage
        final StorageReference fileRef =
        storageReference.child("users/"+fAuth.getCurrentUser().getUid()+"/profile.jpg");
        fileRef.putFile(imageUri).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
            @Override
            public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
                fileRef.getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
                    @Override
                    public void onSuccess(Uri uri) {
                        Picasso.get()
                                .load(uri)
                                .into(profileImage);
                    }
                });
            }
        }).addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Toast.makeText(getContext(), "Failed.", Toast.LENGTH_SHORT).show();
            }
        });

    }
```

Listato 4.10. Codice dei metodi per modificare l'immagine del profilo con la libreria `Picasso`

Dati personali

A partire dalla collection `users` del database viene recuperato l'ID dello *user* corrente; da quel parametro si ottengono gli altri dati grazie a `documentSnapshot.getString("db param")`; questi, infine, si iniettano dentro alle relative textview (Listato 4.11).

```
DocumentReference docRef = fStore.collection("users").document(userId);
        try {
            docRef.addSnapshotListener(new EventListener<DocumentSnapshot>() {
                @Override
                public void onEvent(@Nullable DocumentSnapshot documentSnapshot, @Nullable
                FirebaseFirestoreException e) {
                    if(documentSnapshot != null ? documentSnapshot.exists() : false){
                        phone.setText(documentSnapshot.getString("phone"));
                        fullName.setText(documentSnapshot.getString("firstname") + "␣" +
                        documentSnapshot.getString("lastname"));
                        email.setText(documentSnapshot.getString("email"));
                    } else {
                        Log.d("tag", "onEvent:␣Document␣do␣not␣exists");
                    }
                }
            });
        } catch (Exception e){
            Log.d("Do␣nothing", "");
        }
```

Listato 4.11. Codice del metodo per visualizzare i dati dell'utente

Per quanto riguarda, invece, la modifica dei dati, viene mostrato allo *user* un alertdialog che funge da form. Una volta che i campi sono stati compilati correttamente, viene creata una nuova mappa con i dati aggiornati; quest'ultima sovrascriverà la precedente grazie al metodo `DatabaseReferences.getUserById(FirebaseAuth.getI`

`nstance().getCurrentUser().getUid()).set(mp)`, come mostrato nel Listato 4.12.

```
modify.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View v) {
             try {
                AlertDialog.Builder builder = new AlertDialog.Builder(getContext());
                builder.setTitle("Update␣your␣data");
                LayoutInflater inflater = getActivity().getLayoutInflater();
                View dialogView = inflater.inflate(R.layout.dialog_user, null);

                builder.setView(dialogView);

                final EditText name = dialogView.findViewById(R.id.name_user);
                final EditText surname = dialogView.findViewById(R.id.surname_user);
                final EditText phone = dialogView.findViewById(R.id.phone_user);

                builder.setPositiveButton("Confirm", new DialogInterface.OnClickListener() {
                   @Override
                   public void onClick(DialogInterface dialog, int which) {
                      if( name.getText().toString().equals("")||
                             surname.getText().toString().equals("")||
                             phone.getText().toString().equals("")

                      ) {
                         Toast.makeText(getContext(), "Impossible␣to␣Modify:" + "\nNo␣empty␣spaces",
                         Toast.LENGTH_SHORT).show();
                      }
                      else if(name.getText().toString().length() > 20)
                      {
                         Toast.makeText(getContext(), "Impossible␣to␣Create",
                         Toast.LENGTH_SHORT).show();
                      }

                      else if(surname.getText().toString().length() > 25)
                      {
                         Toast.makeText(getContext(), "Impossible␣to␣Create", Toast.LENGTH_SHORT).show();
                      }

                      else if(phone.getText().toString().length() < 8)
                      {
                         Toast.makeText(getContext(), "Impossible␣to␣Create", Toast.LENGTH_SHORT).show();
                      }

                      else if(isDoubleOrInt(phone.getText().toString())== -1)
                      {
                         Toast.makeText(getContext(), "Impossible␣to␣Create" , Toast.LENGTH_SHORT).show();
                      }
                      else {
                         final Map<String, Object> mp = new HashMap<>();
                            mp.put("firstname", name.getText().toString());
                            mp.put("lastname", surname.getText().toString());
                            mp.put("phone", phone.getText().toString());
                            mp.put("uri", mod_uri);
                            mp.put("email", mod_email);
                            mp.put("admin", mod_admin);
                            mp.put("goal", Integer.parseInt(mod_goal));
                            mp.put("tot_cal" , Integer.parseInt(mod_totcal));

                            DatabaseReferences.getUserById(FirebaseAuth.getInstance().getCurrentUser()
                            .getUid()).set(mp);
                            Toast.makeText(getContext(), "Update␣Done", Toast.LENGTH_SHORT).show();
                      }
                   }
                });

                builder.setNegativeButton("Delete",
                       new DialogInterface.OnClickListener() {
                          public void onClick(DialogInterface dialog, int which) {

                          }
                       });
                builder.show();

                } catch (Exception e){
                Toast.makeText(getContext(), "Error!␣" + e.getMessage(), Toast.LENGTH_SHORT).show();
                }
             }
```

Listato 4.12. Codice del metodo `setOnClickListener` associato al pulsante di modifica dati utente

4.1.7 FragmentInfo

L'ultima schermata raggiungibile dallo *user*, tramite la BottomNavigationBar, è quella relativa al calcolo delle calorie bruciate. Inizialmente è visibile l'andamento dell'allenamento con una ProgressBar, grazie a due parametri che la caratterizzano, ovvero:

- `goal`; esso rappresenta l'obiettivo che lo *user* deve raggiungere;
- `totcal`; esso rappresenta l'attuale progresso dello *user*.

Tali dati vengono visualizzati sullo schermo attraverso i metodi `progressbar.setProgress(int progress)` e `progressbar.setMax(int max)`, osservabili nel Listato 4.13.

```
calories_bar = (ProgressBar) view.findViewById(R.id.progress_bar_calories);
       if (riprova == 0){
           user_cal = Integer.parseInt(this.getArguments().getString("tot_cal"));
       }
       else {
           user_cal = riprova;
       }

       user_goal = Integer.parseInt(this.getArguments().getString("goal"));


       Log.d("VERIFICA␣VALORI␣PASSATI", this.getArguments().getString("tot_cal"));
       if (prova == null){
           goal_view.setText("Your␣Goal␣is:␣" + this.getArguments().getString("goal") + "␣calories.");
           calories_bar.setMax(Integer.parseInt(this.getArguments().getString("goal")));
       }
       else {
           goal_view.setText("Your␣Goal␣is:␣" + prova + "␣calories.");
           calories_bar.setMax(Integer.parseInt(prova));
       }
           act_cal.setText("Curr.Cal.:" + user_cal);
           calories_bar.setProgress(user_cal);
```

Listato 4.13. Codice del metodo per l'iniezione dei dati dal database alla ProgressBar

Reset progresso e set obiettivo

Cliccando sul tasto di reset si genera un alertdialog, il quale permette di azzerare il valore di `totcal` sul database, mediante il metodo booleano `isTotCalChanged()`, e, di conseguenza, anche il valore iniettato e visibile sulla ProgressBar. Discorso analogo vale per il set, con un altro alertdialog che interviene richiedendo il valore dell'obiettivo da settare. Questa volta il metodo booleano che effettua la modifica sul database è `isGoalChanged()`, con successivo aggiornamento del massimo sulla barra di progresso (Listato 4.14).

```
reset.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View view) {
              final AlertDialog.Builder builder = new AlertDialog.Builder(getContext());
              builder.setTitle("Reset␣the␣progress?");
              builder.setView(inflater.inflate(R.layout.reset_alert, null));
              builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                  @Override
                  public void onClick(DialogInterface dialog, int which) {
                      user_cal = 0;
                      riprova = user_cal;
                      calories_bar.setProgress(user_cal);
                      act_cal.setText("Curr.Cal.:␣" + 0);
                      if (isTotCalChanged()){
                          Toast.makeText(getContext(),"Update␣Done", Toast.LENGTH_SHORT).show();
                      }
                      else {
                          Toast.makeText(getContext(),"No␣Update", Toast.LENGTH_SHORT).show();
                      }
```

```
                }
                });
            builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    //close dialog
                }
            });
            builder.create().show();
        }
    });

    goal.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

            try {
                //qui va settato il valore dell'obiettivo di calorie dell'utente
            final AlertDialog.Builder builder = new AlertDialog.Builder(getContext());
            builder.setTitle("Set your Goal");
            View dialogView = inflater.inflate(R.layout.goal_alert, null);

            builder.setView(dialogView);

            final EditText cal_goal = dialogView.findViewById(R.id.cal_goal);


            builder.setPositiveButton("Confirm", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    try {
                    if (cal_goal.getText().toString().equals(""))
                    {
                        Toast.makeText(getContext(), "Impossible to set:" + "\nNo empty values",
                        Toast.LENGTH_SHORT).show();
                    }
                    else if (isDoubleOrInt(cal_goal.getText().toString())== -1)
                    {
                        Toast.makeText(getContext(), "Impossible to set:" + "\nGoal must be an Int",
                        Toast.LENGTH_SHORT).show();
                    }
                    else {
                        int calories = Integer.parseInt(cal_goal.getText().toString());
                        //calories_bar.setProgress(0);
                        user_goal= calories;
                        prova = cal_goal.getText().toString();
                        calories_bar.setMax(calories);
                        calories_bar.setProgress(user_cal);
                        goal_view.setText("Your goal is now: " + cal_goal.getText().toString();


                        if (isGoalChanged()){
                            Toast.makeText(getContext(),"Update Done", Toast.LENGTH_SHORT).show();
                        }
                        else {
                            Toast.makeText(getContext(),"No Update", Toast.LENGTH_SHORT).show();
                        }
                    }

                }catch (Exception e){
                        Toast.makeText(getContext(), "No empty spaces:  " + e.getMessage(),
                        Toast.LENGTH_SHORT).show();
                    }
                }

            });

            builder.setNegativeButton("Delete", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    //close dialog
                }
            });
            builder.create().show();
        }
        catch (Exception e){
            Toast.makeText(getContext(), "Error! " + e.getMessage(), Toast.LENGTH_SHORT).show();
            }
        }

    });

    public boolean isTotCalChanged()
{
    if (!this.getArguments().getString("tot_cal").equals(user_cal)) {
        final Map<String, Object> mp = new HashMap<>();
        mp.put("firstname", this.getArguments().getString("firstname"));
        mp.put("lastname", this.getArguments().getString("lastname"));
        mp.put("phone", this.getArguments().getString("phone"));
        mp.put("uri", this.getArguments().getString("uri"));
        mp.put("email", this.getArguments().getString("email"));
        mp.put("admin", this.getArguments().getString("admin"));
        mp.put("tot_cal" , user_cal);
```

```
            if (prova == null){
                mp.put("goal", user_goal);
            }else{
                mp.put("goal", Integer.parseInt(prova));
            }

            DatabaseReferences.getUserById(FirebaseAuth.getInstance().getCurrentUser().getUid()).set(mp);
            //setTot_cal(String cal_goal)
            return true;
        } else {
            return false;
        }
    }
```

Listato 4.14. Codice del metodo `setOnClickListener` associato al pulsante per il reset del progresso e il set dell'obiettivo e del metodo `isTotCalChanged`

Calcolo delle calorie bruciate

Questa funzionalità serve per calcolare le calorie che sono state bruciate dallo *user* svolgendo gli esercizi. Dopo aver cliccato sul pulsante dedicato, appare un alert-dialog, mediante il quale lo *user* può selezionare gli esercizi che ha svolto (anche più di una volta). Successivamente il sistema recupererà il parametro dell'esercizio rigurdante le calorie `ex.getCal()` e lo sommerà a quelli degli altri esercizi selezionati. Il risultato di tale calcolo verrà aggiunto all'attuale progresso dello *user*, sovrascrivendone anche il valore sul database, attraverso `isTotCalChanged()`, e sulla ProgressBar, con `caloriesbar.setProgress(usercal)`, come mostrato nel Listato 4.15.

```
choose.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                allEsercizi = new String[listEserciziAdd.size()];
                index = 0;
                for (Esercizi e : listEserciziAdd) {
                    allEsercizi[index] = e.getName();
                    index++;
                }
                checkedItemsArray = new boolean[allEsercizi.length];
                exList = Arrays.asList(allEsercizi);

                final AlertDialog.Builder builder = new AlertDialog.Builder(getContext());
                builder.setTitle("Calculate␣how␣many␣calories␣you␣burned␣on␣exercises!");
                builder.setView(inflater.inflate(R.layout.dialog_list, null));


                builder.setMultiChoiceItems(allEsercizi, checkedItemsArray, new
                DialogInterface.OnMultiChoiceClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which, boolean isChecked) {

                        checkedItemsArray[which] = isChecked;
                        String currentItem = exList.get(which);
                        builder.setCancelable(false);
                        builder.setTitle("Choose␣Exercises␣to␣add");
                    }
                });
                builder.setPositiveButton("Calculate", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        int counter = 0;
                        for (int i = 0; i < checkedItemsArray.length; i++) {
                            boolean checked = checkedItemsArray[i];
                            if (checked){
                                try {
                                    for (final Esercizi ex : listEserciziAdd) {
                                        if(allEsercizi[i].toString().equals(ex.getName())) {
                                            counter = counter + Integer.parseInt(ex.getCal());
                                        }
                                    }
                                }catch (Exception e){
                                    Toast.makeText(getContext(), "Error!␣" + e.getMessage(),
                                    Toast.LENGTH_SHORT).show();
                                }
                            }
```

```
                    }
                    user_cal += counter;
                    riprova = user_cal;
                    calories_bar.setProgress(user_cal);
                    act_cal.setText("Curr.Cal.:" + user_cal);

                    if (isTotCalChanged()){
                        Toast.makeText(getContext(),"Update␣Done", Toast.LENGTH_SHORT).show();
                    }
                    else {
                        Toast.makeText(getContext(),"No␣Update", Toast.LENGTH_SHORT).show();
                    }
                }
                });

            builder.setNegativeButton("Delete", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {

                }
            });
            builder.create().show();
        }
    });
```

Listato 4.15. Codice del metodo `setOnClickListener` associato al pulsante per il calcolo delle calorie bruciate

4.1.8 FragmentAdmin

La prima schermata visibile all'*admin* riguarda la lista degli utenti registrati, con i quali egli può interagire, semplicemente cliccando sul nome specifico. Essendo una lista, abbiamo implementato una recyclerview e l'adapter associato. Al fine di popolarla prima ricaviamo i dati di ogni singolo *user* e poi li inseriamo al suo interno, uno ad uno, attraverso il metodo `listUsers.add(users)`; infine, configuriamo l'adapter con `recyclerView.setAdapter(adminAdapter)`, come mostrato nel Listato 4.16.

```
@Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_admin, container, false);

        recyclerView = view.findViewById(R.id.rv_admin);
        recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));

        final RecyclerView.ItemDecoration divider = new DividerItemDecoration(getActivity(),
        DividerItemDecoration.VERTICAL);
        recyclerView.addItemDecoration(divider);

        // Name current user
        userId = DatabaseReferences.getUser().getUid();

        // Fill listUser with all the Users on firebase
        try {
            DatabaseReferences.getUsers()
                    .get()
                    .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                        @Override
                        public void onComplete(@NonNull Task<QuerySnapshot> task) {
                            if (task.isSuccessful()) {
                                for (final QueryDocumentSnapshot document : task.getResult()) {
                                    // The current user's name does not appear in the list
                                    if(!document.getId().equals(userId)) {
                                        DatabaseReferences.getUserById(document.getId())
                                                .get()
                                                .addOnCompleteListener(new
                                                OnCompleteListener<DocumentSnapshot>() {
                                                    @Override
                                                    public void onComplete(@NonNull Task<DocumentSnapshot>
                                                    task) {
                                                        if(!Objects.equals(document.get("admin"), "admin")) {
                                                            Log.d("USERS␣DATA",
                                                            document.get("firstname").toString());
                                                            Log.d("USERS␣DATA",
                                                            document.get("lastname").toString());
                                                            Log.d("USERS␣DATA", document.getId());
                                                            Log.d("USERS␣DATA",
                                                            document.get("uri").toString());
```

```
                                                Log.d("USERS_DATA",
                                                document.get("tot_cal").toString());
                                                Log.d("USERS_DATA",
                                                document.get("goal").toString());

                                                Users users = new
                                                Users(document.get("firstname").toString(),
                                                document.get("lastname").toString(),
                                                document.getId(), document.get("uri").toString(),
                                                document.get("tot_cal").toString(),
                                                document.get("goal").toString());
                                                listUsers.add(users);
                                                adminAdapter.notifyDataSetChanged();
                                            }
                                        }
                                    });
                                }
                            }
                        }
                    }).addOnSuccessListener(new OnSuccessListener<QuerySnapshot>() {
                @Override
                public void onSuccess(QuerySnapshot queryDocumentSnapshots) {
                    adminAdapter.notifyDataSetChanged();
                }
            });
        } catch (Exception e) {
            Toast.makeText(getContext(), "Error!␣" + e.getMessage(), Toast.LENGTH_SHORT).show();
        }

        adminAdapter = new AdminAdapter(getContext(), listUsers);
        recyclerView.setAdapter(adminAdapter);

        adminAdapter.notifyDataSetChanged();
        listUsers.clear();

        return view;
    }
```

Listato 4.16. Codice del metodo `onCreateView` del fragment lista utenti

L'adapter sviluppa la lista, permettendo di visualizzare tutti gli utenti registrati attraverso una recyclerview già configurata. Inoltre, al click dell'*admin* su di un nome, si ottiene l'accesso al fragment calendario dello specifico *user*; il tutto grazie all'`onClickListener()` dedicato. Da notare che, con l'importazione della libreria `com.bumptech.glide.Glide`, è possibile visualizzare, per ogni *user*, l'immagine di profilo personale, grazie al metodo `Glide.with(context).load(struttura.get(position).getUri()).into(holder.proPic)`. Ciò rende molto più facile la ricerca e il riconoscimento del singolo all'interno della lista (Listato 4.17).

```
public void onBindViewHolder(@NonNull AdminAdapter.CViewHolder holder, final int position) {
        holder.firstname.setText(struttura.get(position).getFirstname());
        holder.lastname.setText(struttura.get(position).getLastname());
        if(struttura.get(position).getUri() != null) {
        Glide.with(context).load(struttura.get(position).getUri()).into(holder.proPic); }
        holder.parentLayout.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                try {
                    fragmentCalendary = new FragmentCalendary();
                    String id = struttura.get(position).getId();
                    Bundle args = new Bundle();
                    args.putString("ABC", "A");
                    args.putString("u_id", id);
                    args.putString("type", "admin");
                    fragmentCalendary.setArguments(args);
                    FragmentManager fm = ((MainActivity) context).getSupportFragmentManager();
                    FragmentTransaction ft = fm.beginTransaction();
                    ft.replace(R.id.fragment_container, fragmentCalendary).addToBackStack(null).commit();
                }catch (Exception e){
                    Toast.makeText(context.getApplicationContext(), "Error!␣" + e.getMessage(),
                    Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
```

Listato 4.17. Codice del metodo `onBindViewHolder` dell'adapter per l'admin

4.1.9 FragmentAll

In questa particolare schermata l'*admin* ha a disposizione l'intera lista di esercizi; a partire da essa, egli può:

- aggiungerne nuovi esercizi.
- eliminarne esercizi vecchi.
- modificare gli esercizi già presenti.

La lista degli esercizi è resa visibile, assieme alle relative icone, grazie alla configurazione di una recyclerview e dell'adapter associato; il tutto avviene mediante i metodi `listEsercizi.add(new Esercizi(desc, diff, name, cal, uri))` e `recyclerView.setAdapter(adapterAll)`, come mostrato nel Listato 4.18.

```
RecyclerView.ItemDecoration divider = new DividerItemDecoration(getActivity(), DividerItemDecoration.VERTICAL);
        recyclerView.addItemDecoration(divider);


        adapterAll = new AdapterAll(getContext(), listEsercizi);
        recyclerView.setAdapter(adapterAll);

        try {
            DatabaseReferences.getExercises()
                    .get()
                    .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                        @Override
                        public void onComplete(@NonNull Task<QuerySnapshot> task) {
                            if (task.isSuccessful()) {
                                for (final QueryDocumentSnapshot document : task.getResult()) {
                                    Log.d("AO", document.getId());
                                    DatabaseReferences.getExById(document.getId())
                                            .get()
                                            .addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {
                                                @Override
                                                public void onComplete(@NonNull Task<DocumentSnapshot> task) {
                                                    String name = new String(document.get("name").toString());
                                                    String desc = new String(document.get("description").toString());
                                                    String diff = new String(document.get("difficulty").toString());
                                                    String cal = new String(document.get("cal").toString());
                                                    String uri = new String(document.get("uri").toString());
                                                    Log.d(name, uri);
                                                    listEsercizi.add(new Esercizi(desc, diff, name, cal, uri));
                                                    adapterAll.notifyItemInserted(listEsercizi.size());
                                                }
                                            });
                                }
                            }
                        }
                    }).addOnSuccessListener(new OnSuccessListener<QuerySnapshot>() {
                @Override
                public void onSuccess(QuerySnapshot queryDocumentSnapshots) {
                    adapterAll.notifyDataSetChanged();
                }
            });
        } catch (Exception e) {
            Toast.makeText(getContext(), "Error!␣" + e.getMessage(), Toast.LENGTH_SHORT).show();
        }
```

Listato 4.18. Codice del `try catch` nel metodo `onCreateView` del fragment lista esercizi

Oltre alla lista, è presente, come abbiamo già affermato, un pulsante per l'aggiunta di nuovi esercizi. Completando con valori coerenti all'interno dei campi, l'*admin* può creare a piacimento gli esercizi che saranno, poi, disponibili agli utenti (Listato 4.19). La funzione di creazione dell'esercizio sul database è `ExerciseFunctions.createEx(String desc, String diff, final String nm, String cal, String uri)`; questa riceve in input i dati necessari ossia: una descrizione, una difficoltà/tipologia, un nome, le calorie bruciate svolgendolo (in stringa) e l'URI del video associato (in stringa). Infine, dopo essere stato creato, l'esercizio viene aggiunto alla lista complessiva attraverso `listEsercizi.add(new Esercizi`

`(String desc, String diff, String name, String cal, String u ri))` e viene successivamente visualizzato sullo schermo (Listato 4.19).

```
boolean aux = false;
for (Esercizi e : listEsercizi) {
    if (e.getName().equals(name_ex.getText().toString())) {
    aux = true;
    }
}
    if (!aux) {
        try {
        // Write to db the new exercise
        ExerciseFunctions.createEx(desc_ex.getText().toString(), diff_ex.getSelectedItem().toString() ,
        name_ex.getText().toString(), cal_ex.getText().toString(), uri_ex.getText().toString());
        Toast.makeText(getContext(), "The␣exercise␣has␣been␣created", Toast.LENGTH_SHORT).show();
        listEsercizi.add(new Esercizi (desc_ex.getText().toString() , diff_ex.getSelectedItem().toString(),
        name_ex.getText().toString(), cal_ex.getText().toString(), uri_ex.getText().toString())) ;
        adapterAll.notifyItemInserted(listEsercizi.size());
        } catch (Exception e) {
        Toast.makeText(getContext(), "Error!␣" + e.getMessage(), Toast.LENGTH_SHORT).show();
        }
        } else {
        Toast.makeText(getContext(), name_ex.getText().toString(), Toast.LENGTH_SHORT).show();
    }
}
```

Listato 4.19. Codice del metodo per la creazione degli esercizi

Cliccando, invece, sull'icona di modifica dell'esercizio, si genera un alertdialog che permette, una volta compilati correttamente i campi, di portare a compimento la richiesta. Principalmente ciò che avviene è l'eliminazione dell'esercizio con i vecchi dati e la creazione di un nuovo esercizio con i nuovi. Tutto ciò avviene grazie ai metodi `ExerciseFunctions.deleteEx(String name)` e `ExerciseFunctions.createEx(n ewDesc , newDiff , newName, newCal, Uri)`. Una particolarità da far notare assolutamente è il metodo `ExerciseFunctions.upd ateExCard(defName, newName, newDesc, newDiff, newCal, defUri)` il quale, alla modifica di un esercizio all'interno della lista, propaga tale risultato anche all'interno delle schede che lo contengono. Non si avrà, quindi, mai una situazione nella quale degli esercizi obsoleti sono ridondanti dentro una qualsiasi scheda di allenamento (Listato 4.20).

```
holder.btnModify.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(final View v) {
              AlertDialog.Builder alertDialog = new AlertDialog.Builder(v.getContext());//Here I have to use
              v.getContext() istead of just cont.
              alertDialog.setTitle("Update␣Exercise");
              View dialogView = inflater.inflate(R.layout.dialog_ex_mod, null);
              alertDialog.setView(dialogView);
              final EditText name = dialogView.findViewById(R.id.name_ex_mod);
              final EditText description = dialogView.findViewById(R.id.desc_ex_mod);
              final EditText calories = dialogView.findViewById(R.id.cal_ex_mod);
              final Spinner difficulty = dialogView.findViewById(R.id.diff_ex_mod);
              Log.d("KKKKKKK:␣id␣" , struttura.get(position).getName());
              DatabaseReferences.getExById(struttura.get(position).getName()).get().addOnCompleteListener(
                      new OnCompleteListener<DocumentSnapshot>() {
                          @Override
                          public void onComplete(@NonNull Task<DocumentSnapshot> task) {
                              defName = new String(task.getResult().get("name").toString());
                              defDesc = new String(task.getResult().get("description").toString());
                              defDiff = new String(task.getResult().get("difficulty").toString());
                              defCal = new String(task.getResult().get("cal").toString());
                              defUri = new String(task.getResult().get("uri").toString());
                              Log.d("1)" , defName+ defDesc + defDiff + defCal + defUri);
                          }
                      }
              );
              alertDialog.setPositiveButton("Confirm",
                      new DialogInterface.OnClickListener() {
                          public void onClick(DialogInterface dialog, int which) {

                              if( name.getText().toString().length() > 20)
                              {
                                  Toast.makeText(v.getContext(), "Impossible␣to␣Update:",
                                  Toast.LENGTH_SHORT).show();
                              }
```

```
                        else if( description.getText().toString().length() > 50)
                        {
                            Toast.makeText(v.getContext(), "Impossible␣to␣Update:", Toast.LENGTH_SHORT).show();
                        }
                        else if(isDoubleOrInt(calories.getText().toString())== -1)
                        {
                            Toast.makeText(v.getContext(), "Impossible␣to␣Create:", Toast.LENGTH_SHORT).show();
                        }
                        else {
                        if(ExerciseFunctions.verifyName(name.getText().toString()) && !name.equals(defName)){
                            Log.d( "1)" , "lo␣Modifico");
                            String newName;
                            if (name.getText().toString().equals("")) {
                                newName = new String(defName);
                            } else {
                                newName = new String(name.getText().toString());
                            }

                            String newDesc;
                            if (description.getText().toString().equals("")) {
                                newDesc = new String(defDesc);
                            } else {
                                newDesc = new String(description.getText().toString());
                            }

                            String newDiff;
                            if (difficulty.getSelectedItem().toString().equals("")) {
                                newDiff = new String(defDiff);
                            } else {
                                newDiff = new String(difficulty.getSelectedItem().toString());
                            }

                            String newCal;
                            if (calories.getText().toString().equals("")){
                                newCal = new String(defCal);
                            } else {
                                newCal = new String(calories.getText().toString());
                            }

                            Log.d("AA:␣", newDiff);
                            ExerciseFunctions.deleteEx(struttura.get(position).getName());
                            ExerciseFunctions.createEx(newDesc , newDiff , newName, newCal, defUri);
                            ExerciseFunctions.updateExCard(defName , newName, newDesc, newDiff, newCal,
                            defUri);
                            struttura.set(position , new Esercizi(newDesc , newDiff , newName, newCal, defUri
                            ));
                            notifyItemChanged(position , new Esercizi(newDesc , newDiff , newName, newCal,
                            defUri));
                            notifyDataSetChanged();
                            Toast.makeText(v.getContext(), "Update␣Done", Toast.LENGTH_SHORT).show();
                        }else{
                            Log.d( "onClick:␣" , "non␣lo␣Modifico");
                        }
                        notifyDataSetChanged();

                        }
                    }
                });
            alertDialog.setNegativeButton("Negate",
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int which) {

                        }
                    });
            alertDialog.show();
            notifyDataSetChanged();
        }
    });
```

Listato 4.20. Codice del metodo `setOnClickListener` associato al pulsante di modifica degli esercizi

Inoltre se l'*admin* clicca sull'icona di cancellazione dell'esercizio, attraverso un alertdialog specifico, avviene l'effettiva rimozione ad opera del metodo `ExerciseFun ctions.deleteFromExCard(String name , String desc, String, String cal, String uri)`. Al fine di evitare bug o eccezioni non gestite intervengono i metodi `DatabaseReferences.getExById(document.getId()).delete();/ removeAt(po sition)`, i quali ricercano ed eliminano l'esercizio da ogni scheda nella quale è presente, così da evitare situazioni in cui ci sono schede d'allenamento con esercizi ormai rimossi.

```
holder.btnDelete.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View v) {
              AlertDialog.Builder alertDialog = new AlertDialog.Builder(v.getContext());
              v.getContext() istead of just cont.
              alertDialog.setTitle("Delete␣the␣Exercise?");
              View dialogView = inflater.inflate(R.layout.delete_ex, null);
              alertDialog.setView(dialogView);
              alertDialog.setPositiveButton( "Confirm", new DialogInterface.OnClickListener() {
                  @Override
                  public void onClick(DialogInterface dialog, int which) {
                      DatabaseReferences.getExercises()
                          .get()
                          .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                              @Override
                              public void onComplete(@NonNull Task<QuerySnapshot> task) {
                                  if(task.isSuccessful()){
                                      for(DocumentSnapshot document : task.getResult()){

                                          if(document.get("name").toString().equals(struttura.get(position).getName(
                                          ))){
                                              Log.d("esercizio␣rimosso:␣", struttura.get(position).getName());
                                              ExerciseFunctions.deleteFromExCard(struttura.get(position).getName(),
                                              defName, defDesc, defDiff, defCal, defUri);
                                              DatabaseReferences.getExById(document.getId())
                                                      .delete();
                                              removeAt(position);
                                              break;
                                          }
                                      }
                                  }
                              }
                          }).addOnSuccessListener(new OnSuccessListener<QuerySnapshot>() {
                      @Override
                              public void onSuccess(QuerySnapshot queryDocumentSnapshots) {
                                  notifyDataSetChanged();
                              }
                          });
                      }
                  });
                  alertDialog.setNegativeButton("Negate", new DialogInterface.OnClickListener() {
                      @Override
                      public void onClick(DialogInterface dialog, int which) {
                          // Close dialog
                      }
                  });

                  alertDialog.create().show();

          }
      });
```

Listato 4.21. Codice del metodo `setOnClickListener` associato al pulsante di cancellazione degli esercizi

Per poter accedere al fragment contenente i dettagli di un esercizio, basterà che l'*admin* clicchi sul nome specifico. Ciò è reso possibile dall'ennesimo impiego degli oggetti `FragmentManager` e `FragmentTransaction` per passare da un frammento ad un altro trasferendo i dati attraverso `Bundle`.

```
try {
    fragmentVideo = new FragmentVideo();
    Bundle args = new Bundle();
    args.putString("name", struttura.get(position).getName());
    args.putString("description", struttura.get(position).getDescription());
    args.putString("difficulty", struttura.get(position).getDifficulty());
    args.putString("cal", struttura.get(position).getCal());
    args.putString("uri", struttura.get(position).getUri());

    args.putString("type", "admin");

    //controllo dati
    Log.d("DENTRO␣ADAPTER_ALL", struttura.get(position).getName());
    Log.d("DENTRO␣ADAPTER_ALL", struttura.get(position).getDescription());
    Log.d("DENTRO␣ADAPTER_ALL", struttura.get(position).getDifficulty());
    Log.d("DENTRO␣ADAPTER_ALL", struttura.get(position).getCal());
    Log.d("DENTRO␣ADAPTER_ALL", struttura.get(position).getUri());

    fragmentVideo.setArguments(args);
    FragmentManager fm = ((MainActivity) context).getSupportFragmentManager();
    FragmentTransaction ft = fm.beginTransaction();
    ft.replace(R.id.fragment_container, fragmentVideo).addToBackStack(null).commit();
}catch (Exception e){
    Toast.makeText(context.getApplicationContext(), "Error!␣" + e.getMessage(), Toast.LENGTH_SHORT).show();
            }
        }
```

Listato 4.22. Codice dell'`onBindViewHolder` dell'adapter relativo agli esercizi

4.1.10 FragmentVideo

Attraverso questo fragment, l'*admin* può caricare un video all'interno di ogni singolo esercizio, grazie al metodo `ChooseVideo(View view)`, il quale richiama al suo interno `onActivityResult(int requestCode, int resultCode, @Nullable Intent data)`. Lo stesso *admin* potrà, poi, effettuare l'upload sul database attraverso il metodo `UploadVideo()`, come mostrato nel Listato 4.23.

```
up.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View v) {
              UploadVideo();
          }
      });

      choose.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View v) {
              ChooseVideo(v);
          }
      });

 @Override
    public void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if(resultCode != RESULT_CANCELED) {
            if (requestCode == PICK_VIDEO || resultCode == Activity.RESULT_OK || data != null || data.getData() !=
            null) {
                videoUri = data.getData();
                mExoPlayerView.setVisibility(View.INVISIBLE);
                videoView.setVisibility(View.VISIBLE);

                videoView.setVideoURI(videoUri);
                mediaController = new MediaController(this.getActivity());
                videoView.setMediaController(mediaController);
                mediaController.setAnchorView(videoView);
                videoView.start();
                //upload su firebase poi dopo lo modifichi
                //ExerciseFunctions.deleteEx(this.getArguments().getString("name"));
                //ExerciseFunctions.createEx(this.getArguments().getString("description"),
                this.getArguments().getString("difficulty"), this.getArguments().getString("name"),
                this.getArguments().getString("cal"), videoUri.toString());
                //ExerciseFunctions.modifyEx(this.getArguments().getString("difficulty"),
                this.getArguments().getString("description") , this.getArguments().getString("difficulty")
                ,this.getArguments().getString("name") , videoUri.toString());

            }
        }else{
            Toast.makeText(getContext(), "Failed", Toast.LENGTH_SHORT).show();
        }
    }

public void ChooseVideo(View view) {
        Intent intent = new Intent();
        intent.setType("*/*");
        intent.setAction(Intent.ACTION_OPEN_DOCUMENT);
        startActivityForResult(intent, PICK_VIDEO);
    }


    private String getExt(Uri uri) {
        ContentResolver contentResolver = getActivity().getContentResolver();
        MimeTypeMap mimeTypeMap = MimeTypeMap.getSingleton();
        return mimeTypeMap.getExtensionFromMimeType(contentResolver.getType(uri));
    }


    private void UploadVideo(){
        if (videoUri != null) {

            upload.setVisibility(View.VISIBLE);
            final StorageReference reference = storageReference.child(System.currentTimeMillis() + "." +
            getExt(videoUri));
            uploadTask = reference.putFile(videoUri);

            Task<Uri> urltask = uploadTask.continueWithTask(new Continuation<UploadTask.TaskSnapshot, Task<Uri>>() {
```

```
                @Override
                public Task<Uri> then(@NonNull Task<UploadTask.TaskSnapshot> task) throws Exception {
                    if (!task.isSuccessful()) {
                        throw task.getException();
                    }
                    return reference.getDownloadUrl();
                }
            })
                    .addOnCompleteListener(new OnCompleteListener<Uri>() {
                        @Override
                        public void onComplete(@NonNull Task<Uri> task) {

                            if (task.isSuccessful()){
                                Uri downloadUrl = task.getResult();
                                upload.setVisibility(View.INVISIBLE);
                                Toast.makeText(getContext(), "VideoSaved", Toast.LENGTH_SHORT).show();
                                ExerciseFunctions.deleteEx(pre_name);
                                ExerciseFunctions.createEx(pre_desc, pre_diff, pre_name, pre_cal,
                                downloadUrl.toString());
                                //String i = databaseReferences.push().getKey();
                                //databaseReferences.child(i).setValue();
                            }
                            else{
                                Toast.makeText(getContext(), "Failed", Toast.LENGTH_SHORT).show();
                            }
                        }
                    });
        }
    }
```

Listato 4.23. Codice dei metodi per visualizzare e caricare i video dal dispositivo sul database

Al fine di poter riprodurre contenuti salvati su Firebase, non potendo usufruire delle funzionalità di *streaming*, abbiamo optato per l'implementazione di `ExoPlayer`. Questa soluzione permette, inizialmente, di scaricare il video sul dispositivo e, successivamente, di riprodurlo, quando sono stati recuperati abbastanza byte di dati per farlo (Listato 4.24).

```
storageReference = FirebaseStorage.getInstance().getReference("exercises");
        database = FirebaseDatabase.getInstance();
        databaseReferences = FirebaseDatabase.getInstance().getReference("Esercizi");
        mExoPlayerView = view.findViewById(R.id.video_view);
        videoView = view.findViewById(R.id.video_two);

        try {
            BandwidthMeter bandwidthMeter = new DefaultBandwidthMeter.Builder(getContext()).build();
            TrackSelector trackSelector = new DefaultTrackSelector(new
            AdaptiveTrackSelection.Factory(bandwidthMeter));
            exoPlayer = (SimpleExoPlayer) ExoPlayerFactory.newSimpleInstance(getContext());
            Uri video = Uri.parse(pre_uri);
            DefaultHttpDataSourceFactory dataSourceFactory = new DefaultHttpDataSourceFactory("Esercizi");
            ExtractorsFactory extractorsFactory = new DefaultExtractorsFactory();
            MediaSource mediaSource = new ExtractorMediaSource(video, dataSourceFactory, extractorsFactory, null,
            null);
            mExoPlayerView.setPlayer(exoPlayer);
            exoPlayer.prepare(mediaSource);
            exoPlayer.setPlayWhenReady(false);
        }catch (Exception e){
            Toast.makeText(getContext(), "Failed to play", Toast.LENGTH_SHORT).show();
        }
```

Listato 4.24. Codice dei metodi per la configurazione dell'ExoPlayer

4.1.11 FragmentCards

Si raggiunge il frammento relativo alla scheda d'allenamento quando lo *user* o l'*admin* cliccano sull'icona all'interno del calendario. Qui sarà possibile visualizzare il nome ed i pulsanti, i quali variano a seconda della tipologia d'utenza. Lo *user* visualizzerà il tasto per l'allenamento, mentre per l'*admin* avremo quelli di modifica ed eliminazione; tale controllo viene effettuato dalla guardia

`if(controlifAd)`. I pulsanti di cancellazione e di modifica funzionano analogamente al caso della lista di esercizi. Ci soffermiamo nello specifico su quello di allenamento, in quanto prima permette di selezionare una difficoltà e poi, a seconda della scelta, fornisce tempi di svolgimento diversi. Un esempio di ciò è dato dal metodo `bDifficulty.setOnClickListener(new View.OnClickListener()`. Inoltre, attraverso gli oggetti `FragmentManager` e `FragmentTransaction`, è possibile passare al fragment di allenamento cliccando sull'icona specifica, oppure al fragment degli esercizi disponibili, cliccando sul nome (Listato 4.25).

```
public void onBindViewHolder(@NonNull final CViewHolder holder, final int position) {
       holder.textPath.setText(struttura.get(position).getPath());
       if(control_ifAd){
           holder.btnMod.setVisibility(View.VISIBLE);
           holder.btnMod.setOnClickListener(new View.OnClickListener() {
               @Override
               public void onClick(View v) {
                   AlertDialog.Builder alertDialog = new AlertDialog.Builder(v.getContext());//Here I have to use
                   v.getContext() istead of just cont.
                   alertDialog.setTitle("Enter␣new␣card␣name");
                   final EditText name = new EditText(v.getContext());
                   alertDialog.setView(name);
                   alertDialog.setPositiveButton("confirm",
                           new DialogInterface.OnClickListener() {
                               public void onClick(DialogInterface dialog, int which) {
                                   if( name.getText().toString().equals("")){
                                       Toast.makeText(context, "Impossible␣to␣Update:" + "\nInsert␣new␣name",
                                       Toast.LENGTH_SHORT).show();
                                   }
                                   else if( name.getText().toString().length() > 20)
                                   {
                                       Toast.makeText(context, "Impossible␣to␣Update:" + "\nName␣must␣be␣<20",
                                       Toast.LENGTH_SHORT).show();
                                   }
                                   else {
                                       String newName = name.getText().toString();
                                       String type = struttura.get(position).getType();
                                       String ref = struttura.get(position).getRef();

                                       CardsFunctions.modifyCard(ref, struttura.get(position).getPath(), newName,
                                       type, currentDay);

                                       struttura.set(position, new Cards(newName, ref, type));
                                       notifyItemChanged(position, new Cards(newName, ref, type));
                                       notifyDataSetChanged();
                                   }
                               }
                           }
                           );
                   alertDialog.setNegativeButton("delete",
                           new DialogInterface.OnClickListener() {
                               public void onClick(DialogInterface dialog, int which) {

                               }
                           });
                   alertDialog.show();
                   notifyDataSetChanged();
               }
           });
           holder.btnDel.setVisibility(View.VISIBLE);
           holder.btnDel.setOnClickListener(new View.OnClickListener() {
               @Override
               public void onClick(View v) {
                   final String uid = struttura.get(position).getRef();
                   AlertDialog.Builder alertDialog = new AlertDialog.Builder(v.getContext());//Here I have to use
                   v.getContext() istead of just cont.
                   alertDialog.setTitle("Delete␣the␣Card?");
                   View dialogView = inflater.inflate(R.layout.delete_ex, null);
                   alertDialog.setView(dialogView);
                   alertDialog.setPositiveButton("Confirm", new DialogInterface.OnClickListener() {
                       @Override
                       public void onClick(DialogInterface dialog, int which) {
                   //holder.btnDel.
                   CardsFunctions.deleteCard(struttura.get(position).getRef() , struttura.get(position).getPath(),
                   currentDay);
                   struttura.remove(holder.getAdapterPosition());
                   notifyItemRemoved(holder.getAdapterPosition());
                   notifyDataSetChanged();
                   if (struttura.isEmpty()){
                       fragmentCalendary = new FragmentCalendary();
                       Bundle args = new Bundle();
                       if(control_ifAd) {
                           args.putString("type", "admin");
                           args.putString("u_id" , uid);
                       }else{
                           args.putString("type" , "noadmin");
```

```
                }
                args.putString("ABC", ABC);
                fragmentCalendary.setArguments(args);
                FragmentManager fm = ((MainActivity) context).getSupportFragmentManager();
                FragmentTransaction ft = fm.beginTransaction();
                ft.replace(R.id.fragment_container, fragmentCalendary).addToBackStack(null).commit();
            }

        }
    });
            alertDialog.setNegativeButton("Negate", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    // Close dialog
                }
            });
            alertDialog.create().show();
        }

    });

    holder.btnPlay.setVisibility(View.INVISIBLE);
} else {
    holder.btnDel.setVisibility(View.INVISIBLE);
    holder.btnMod.setVisibility(View.INVISIBLE);
    holder.btnPlay.setVisibility(View.VISIBLE);
    holder.btnPlay.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
    if(struttura.size() > 0){
        final AlertDialog.Builder builder = new AlertDialog.Builder(v.getContext());
        View dialogView = inflater.inflate(R.layout.difficulty_card, null);
        builder.setTitle("CHOOSE␣DIFFICULTY:");
        builder.setView(dialogView);
        final Button bEasy = dialogView.findViewById(R.id.easy_btn);
        final Button bMedium = dialogView.findViewById(R.id.medium_btn);
        final Button bHard = dialogView.findViewById(R.id.hard_btn);
        /*final Button bGoto = dialogView.findViewById(R.id.goto_btn);*/
        bEasy.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                timeLeftInMilliseconds = 5000;
                timePauseInMilliseconds = 15000;
                bEasy.setBackground(ResourcesCompat.getDrawable(v.getResources(),
                R.drawable.whiteline_background, null));
                bHard.setBackgroundColor(v.getResources().getColor(android.R.color.white));
                bMedium.setBackgroundColor(v.getResources().getColor(android.R.color.white));
            }
        });
        bMedium.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                timeLeftInMilliseconds = 10000;
                timePauseInMilliseconds = 10000;
                bEasy.setBackgroundColor(v.getResources().getColor(android.R.color.white));
                bMedium.setBackground(ResourcesCompat.getDrawable(v.getResources(),
                R.drawable.whiteline_background, null));
                bHard.setBackgroundColor(v.getResources().getColor(android.R.color.white));
            }
        });
        bHard.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                timeLeftInMilliseconds = 15000;
                timePauseInMilliseconds = 5000;
                bEasy.setBackgroundColor(v.getResources().getColor(android.R.color.white));
                bMedium.setBackgroundColor(v.getResources().getColor(android.R.color.white));
                bHard.setBackground(ResourcesCompat.getDrawable(v.getResources(),
                R.drawable.whiteline_background, null));
            }
        });

        builder.setPositiveButton("confirm",
        new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            if (timePauseInMilliseconds!=0 && timeLeftInMilliseconds!=0){
                try {
                    fragmentCountdown = new FragmentCountdown();
                    String path_scheda = struttura.get(position).getPath();
                    String ref= struttura.get(position).getRef();
                    Bundle args = new Bundle();
                    Log.d("1␣left:␣"+String.valueOf(timeIntLeft), "pause:␣"+String.valueOf(timeIntPause));
                    args.putLong("timeTot", timeIntLeft);
                    args.putLong("timeLeftInMilliseconds", timeLeftInMilliseconds);
                    args.putLong("timePauseInMilliseconds", timePauseInMilliseconds);
                    args.putString("path", path_scheda);
                    args.putString("ref", ref);
                    args.putString("currentDay", currentDay);
                    fragmentCountdown.setArguments(args);
                    FragmentManager fm = ((MainActivity) context).getSupportFragmentManager();
                    FragmentTransaction ft = fm.beginTransaction();
                    ft.replace(R.id.fragment_container, fragmentCountdown).addToBackStack(null).commit();
                }catch (Exception e){
                    Toast.makeText(context.getApplicationContext(), "Error!␣" + e.getMessage(),
```

```
                        Toast.LENGTH_SHORT).show();
                    }
                }
            }
            });
            builder.setNegativeButton("Negate", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                                // Close dialog
                }
            });
            builder.show();
        }
    }
});
```

Listato 4.25. Codice del metodo `onBindViewHolder` dell'adapter relativo alla scheda

4.1.12 FragmentEsercizi

Questo frammento contiene gli esercizi all'interno di una scheda, con un'unica peculirità. Mentre lo *user* può solamente visualizzare gli elementi presenti, l'*admin* può, invece, aggiungere esercizi nella scheda attraverso i metodi `ExerciseFunctions.add ExToCard(ref, path, ex, currentDay ); listEserciziScheda.add(ex)`, oppure rimuovere da essa esercizi già presenti, grazie a `struttura.remove(position)`, come mostrato nei Listati 4.26 e 4.27.

```
btnAdd.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                  allEsercizi = new String[listEserciziAdd.size()];
                  index = 0;
                  for (Esercizi e : listEserciziAdd) {
                      allEsercizi[index] = e.getName();
                      index++;
                  }
                  checkedItemsArray = new boolean[allEsercizi.length];
                  exList = Arrays.asList(allEsercizi);

                  final AlertDialog.Builder builder = new AlertDialog.Builder(getContext());
                  builder.setTitle("Choose␣Exercises␣to␣add");
                  builder.setMultiChoiceItems(allEsercizi, checkedItemsArray, new
                  DialogInterface.OnMultiChoiceClickListener() {
                      @Override
                      public void onClick(DialogInterface dialog, int which, boolean isChecked) {
                          checkedItemsArray[which] = isChecked;
                          String currentItem = exList.get(which);
                          builder.setCancelable(false);
                      }
                  }).setPositiveButton("ADD", new DialogInterface.OnClickListener() {
                      @Override
                      public void onClick(DialogInterface dialog, int which) {
                          for (int i = 0; i < checkedItemsArray.length; i++) {
                              boolean checked = checkedItemsArray[i];
                              if (checked) {
                                  for (final Esercizi ex : listEserciziAdd) {
                                      if (ex.getName().equals(allEsercizi[i])) {
                                          boolean aux = false;
                                          for (Esercizi e : listEserciziScheda) {
                                              if (e.getName().equals(ex.getName())) {
                                                  aux = true;
                                              }
                                          }
                                          if(!aux){
                                              try {
                                                  ExerciseFunctions.addExToCard(ref, path, ex, currentDay );
                                                  listEserciziScheda.add(ex);
                                                  EserciziAdapter.notifyDataSetChanged();
                                              } catch (Exception e) {
                                                  Toast.makeText(getContext(), "Error!␣" + e.getMessage(),
                                                  Toast.LENGTH_SHORT).show();
                                              }
                                          }else{
                                              Toast.makeText(getContext(), "This␣Exercise␣already␣exists",
                                              Toast.LENGTH_SHORT).show();
                                          }
                                      }
                                  }
                              }
                          }
                      }
```

```
                }
            });
            builder.create().show();
        }
    });
```

Listato 4.26. Codice del pulsante per l'aggiunta degli esercizi in una scheda

```
holder.btnDel.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                AlertDialog.Builder alertDialog = new AlertDialog.Builder(v.getContext());//Here I have to use
                v.getContext() istead of just cont.
                alertDialog.setTitle("Delete␣the␣Exercise?");
                View dialogView = inflater.inflate(R.layout.delete_ex, null);
                alertDialog.setView(dialogView);
                alertDialog.setPositiveButton("Confirm", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        final CollectionReference colRef = DatabaseReferences.listExCard(ref, path, currentDay);
                        colRef.get().addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                            @Override
                            public void onComplete(@NonNull Task<QuerySnapshot> task) {
                                if(task.isSuccessful()) {
                                    for (final QueryDocumentSnapshot document : task.getResult()) {
                                        if  (document.getId().equals(struttura.get(position).getName())) {
                                            Log.d("delete:", struttura.get(position).getName());
                                            colRef.document(document.getId())
                                                    .delete();
                                            struttura.remove(position);
                                            notifyItemRemoved(position);
                                            notifyDataSetChanged();
                                            notifyItemRangeChanged(position, getItemCount());
                                            break;
                                        }
                                    }
                                }
                            }
                        }).addOnSuccessListener(new OnSuccessListener<QuerySnapshot>() {
                            @Override
                            public void onSuccess(QuerySnapshot queryDocumentSnapshots) {
                                notifyDataSetChanged();
                            }
                        });
                    }
                });
                        alertDialog.setNegativeButton("Negate", new DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int which) {
                                // Close dialog
                            }
                        });

                        alertDialog.create().show();
                    }
                });
```

Listato 4.27. Codice del pulsante per la cancellazione degli esercizi all'interno di una scheda

Infine, sia lo *user* che l'*admin* possono cliccare sul nome dell'esercizio accedendo al frammento di riproduzione del video, sempre grazie ai soliti oggetti `FragmentManager` e `FragmentTransaction`.

4.1.13 FragmentCountDown

L'ultimo frammento che analizziamo è quello relativo all'allenamento. Ad esso si giunge dopo aver cliccato sul pulsante specifico all'interno del frammento relativo alla scheda, e dopo aver selezionato un livello di difficoltà fra quelli proposti. All'avvio si genera un timer attraverso `startStop()` ed una ProgressBar grazie a `progressBar.setVisibility(V iew.VISIBLE)`, i quali scorrono per indicare lo svolgimento corretto dell'esercizio. Inoltre, si ha la possibilità di sospendere momentaneamente l'allenamento con il tasto di pausa implementato dal metodo

`stopTimer()`, il quale ogni volta effettua il reset del progresso della barra. Realizzare l'effettivo scorrere del tempo è, invece, possibile attraverso funzioni di aggiornamento continuo di timer e Bar, come `onTick(long l)`, che contiene all'interno il metodo di *update* `updateTimer()`, come evidenziato nel Listato 4.28.

```
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState){
        View view = inflater.inflate(R.layout.countdown_ex, container, false);
        countdown = view.findViewById(R.id.countdown);
        currentExDiff = view.findViewById(R.id.currentExDiff);
        currentExName = view.findViewById(R.id.currentExName);
        end = view.findViewById(R.id.end_txt);
        StartAndStopBtn = view.findViewById(R.id.StartAndStopBtn);
        progressBar = view.findViewById(R.id.progress_bar);

        assert this.getArguments() != null;
        timeLeftInMilliseconds = this.getArguments().getLong("timeLeftInMilliseconds");
        timePauseInMilliseconds = this.getArguments().getLong("timePauseInMilliseconds");
        k = 60000/timeLeftInMilliseconds;
        h = 60000/timePauseInMilliseconds;
        hok = k;
        path = this.getArguments().getString("path");
        ref = this.getArguments().getString("ref");
        currentDay = this.getArguments().getString("currentDay");
        StartAndStopBtn.setText("START");
        timerRunning = true;
        try {
        //
            lenght = 0;
            listExCard(ref , path,currentDay).get()
                    .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                        @Override
                        public void onComplete(@NonNull Task<QuerySnapshot> task) {
                            if(task.isSuccessful()){
                                for(final QueryDocumentSnapshot document : task.getResult()){
                                    listEserciziScheda.add(new
                                    Esercizi(Objects.requireNonNull(document.get("description")).toString(),
                                            document.get("difficulty").toString(),
                                            document.get("name").toString(),
                                            document.get("cal").toString(),
                                            document.get("uri").toString())
                                    );
                                }
                            }
                        }
                        //
                    }).addOnSuccessListener(new OnSuccessListener<QuerySnapshot>() {
                @Override
                public void onSuccess(QuerySnapshot queryDocumentSnapshots) {
                    exCountdown = new String[listEserciziScheda.size()];
                    diffCountdown = new String[listEserciziScheda.size()];
                    int i = 0;
                    for (Esercizi e : orderListByDiff(listEserciziScheda)){
                        exCountdown[i] = e.getName();
                        diffCountdown[i] = e.getDifficulty();
                        Log.d(exCountdown[i], "onSuccess: ");
                        i+=1;
                    }
                    index = -1;
                    currentExDiff.setText("");
                    currentExName.setText("READY?");
                    countdown.setText("");
                    if (listEserciziScheda.size()!=0){
                        StartAndStopBtn.setVisibility(View.VISIBLE);
                        StartAndStopBtn.setOnClickListener(new View.OnClickListener() {
                            @Override
                            public void onClick(View view) {
                                if (firstTime) {
                                    firstTime = false;
                                    progressBar.setVisibility(View.VISIBLE);
                                    index += 1;
                                    currentExDiff.setText("Current Type: " + diffCountdown[index]);
                                    currentExName.setText(exCountdown[index]);
                                }
                                timerRunning = !timerRunning;
                                startStop();
                            }
                        });
                    }else {
                    //no exercises
                        currentExName.setText("NO EXERCISE");
                        StartAndStopBtn.setVisibility(View.INVISIBLE);
                    }
                }
            });
            //catch
        }catch (Exception e){
            Toast.makeText(getContext(), "Error! " + e.getMessage(), Toast.LENGTH_SHORT).show();
        }
```

```
    return view;
}


public void startStop(){
//stop e start
    if(timerRunning) {
        Toast.makeText(getContext(), "Pause: new Max!" , Toast.LENGTH_SHORT).show();
        progressBar.setMax((int) ((hok*timeLeftInMilliseconds/1000)));
        stopTimer();
    }else{
        progressBar.setMax((int) ((hok*timeLeftInMilliseconds/1000)));
        startTimer();
    }
}

public void startTimer(){
    countDownTimer = new CountDownTimer(timeLeftInMilliseconds , 1000){
        @Override
        public void onTick(long l){
            int progress = (int) ((hok*l/1000)+1);
            progressBar.setProgress(progress);
            timeLeftInMilliseconds = l;
            updateTimer();
        }

        @Override
        public void onFinish(){
            progressBar.setProgress(0);
            index += 1;
            if ((index < exCountdown.length)) {
                if(!diffCountdown[index-1].equals(diffCountdown[index]) && !inPause){
                    inPause = true;
                    currentExDiff.setText("BREAK TIME");
                    currentExName.setText("next exercise: " + exCountdown[index]);
                    hok = h;
                    index -= 1;
                }

                else{
                    inPause = false;
                    hok = k;
                    currentExDiff.setText(diffCountdown[index]);
                    currentExName.setText(exCountdown[index]);
                }

                countdown.setText("");
                timeLeftInMilliseconds = timePauseInMilliseconds;
                startStop();
            }else{
                progressBar.setVisibility(View.VISIBLE);
                progressBar.setMax(100);
                progressBar.setProgress(0);
                countdown.setText("END");
                end.setVisibility(View.VISIBLE);
                end.setText("Thank you for training with us!");
                currentExDiff.setText("");
                currentExName.setText("Training is ended");
                StartAndStopBtn.setVisibility(View.INVISIBLE);
            }
        }
    }.start();
    StartAndStopBtn.setText("Pause");
}

public void stopTimer(){
    StartAndStopBtn.setText("RESUME");
    countDownTimer.cancel();
}

public void updateTimer(){
    int seconds = (int) timeLeftInMilliseconds % 60000 / 1000;
    seconds += 1;
    String timeLeftText;
    if(seconds < 10){
        timeLeftText= "0:0"+seconds;
    }else{
        timeLeftText= "0:"+seconds;
    }
    countdown.setText(timeLeftText);
}
```

Listato 4.28. Codice del metodo `onCreateView` del fragment d'allenamento

4.2 Manuale utente

In questa sezione ci occupiamo di presentare un manuale che esponga in maniera chiara il funzionamento della nostra applicazione. L'obiettivo è quello di offrire una guida quanto più completa possibile, la quale permetta al lettore di conoscere a fondo tutte le funzionalità dell'app per poterne usufruire. Inoltre, per ciascun passo della descrizione saranno allegate una o più istantanee delle schermate visibili, al fine di rendere il tutto molto più immediato ed intuitivo.

4.2.1 Registrazione

Al primo avvio dell'applicazione saremo indirizzati alla pagina di login; non avendo, però, alcun account registrato dovremo raggiungere la schermata di registrazione cliccando sul pulsante "Create account". Una volta che tutti i campi sono stati compilati correttamente, tenendo conto del fatto che quello dedicato al telefono deve contenere solo quantità numeriche e quello relativo all'email ha una sua struttura particolare, si può procedere con la conferma mediante il pulsante "Register" (Figura 4.1). Nel caso in cui venisse rilevata un'imprecisione nella scrittura, il sistema avviserà immediatamente l'utente con una spiegazione degli errori commessi e lo inviterà a riprovare (Figura 4.2).

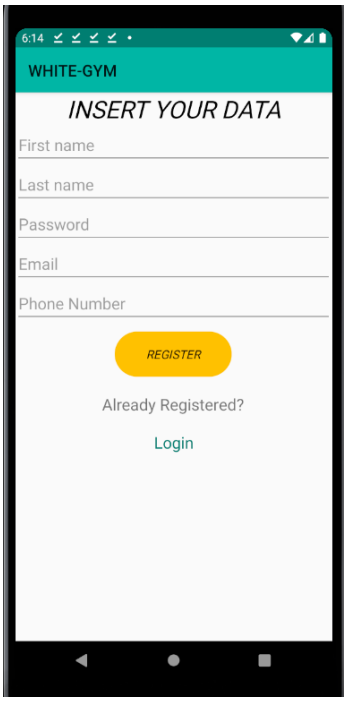

Figura 4.1. Istantanea della schermata di registrazione

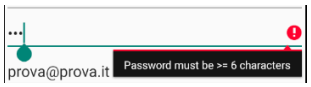

Figura 4.2. Istantanea dell'input visivo d'errore

4.2.2 Login

Nel caso in cui l'utente si fosse già registrato nell'applicazione, sarà opportuno indirizzarlo alla schermata di login. Tale schermata è molto intuitiva, anche grazie al frequente uso di `android:hint` o meglio "Placeholder", i quali evitano la maggior parte delle incomprensioni che possono sorgere. Compilati i due campi (email e password), mediante il pulsante di conferma, si avvia la fase di autenticazione con Firebase (Figura 4.3). Dunque, se l'utente ha digitato correttamente le sue informazioni, riceverà un toast di conferma dell'avvenuto accesso (Figura 4.4); altrimenti ne riceverà uno di errore (Figura 4.5).

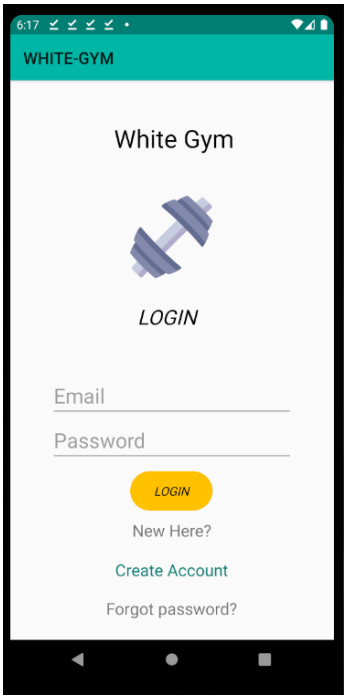

Figura 4.3. Istantanea della schermata di login

Figura 4.4. Istantanea del toast di conferma

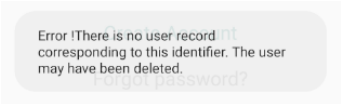

Figura 4.5. Istantanea del toast di errore

4.2.3 Calendario

Una volta completato l'accesso, se l'utente non è un *admin*, si ha il reindirizzamento nel frammento calendario di Figura 4.6. Al suo interno è possibile visualizzare il calendario dell'utente (composto da 4 settimane) con tutte le schede sviluppate dal personal trainer. Inoltre, viene anche data la possibilità allo *user* di cambiare tipologia di allenamento, potendo, quindi, consultare fino a tre calendari diversi, ciascuno dei quali presenta schede con esercizi differenti a seconda delle esigenze. L'*admin*, invece, ha accesso a questa schermata soltanto passando prima per la lista di utenti e cliccando su uno dei nomi. Tale interfaccia si differenzia dalla prima per la possibilità di creare una scheda di allenamento usando l'icona d'aggiunta (Figura 4.7). Successivamente sarà possibile completare la form sullo schermo, confermando o annullando, al fine di ottenere il risultato atteso (Figura 4.8). Infine, per visualizzare i dettagli di una singola scheda, lo *user* o l'*admin* deve cliccare sull'icona corrispondente.

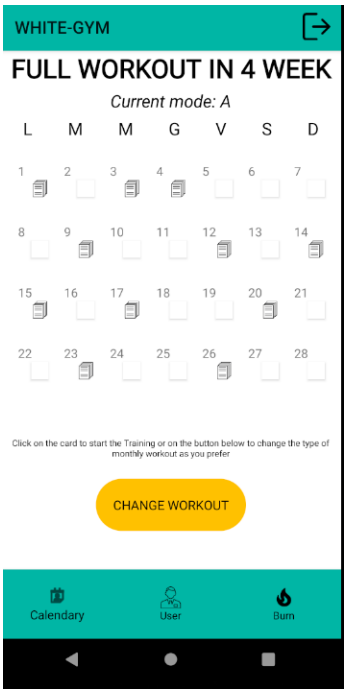

Figura 4.6. Istantanea del fragment relativo al calendario dell'utente

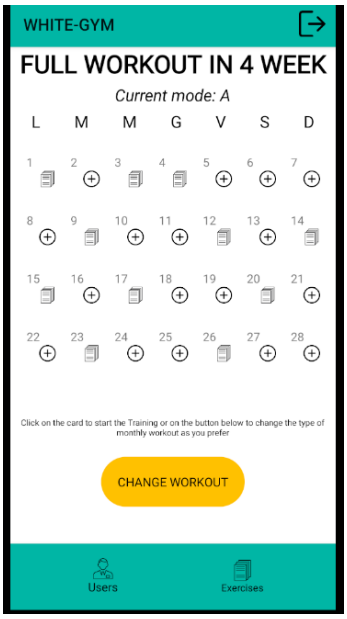

Figura 4.7. Istantanea del fragment relativo al calendario dell'amministratore

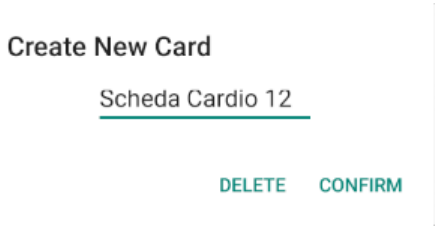

Figura 4.8. Istantanea del popup per la creazione della scheda

4.2.4 Scheda

Il frammento scheda è una sorta di bivio per entrambe le tipologie di utente, in quanto permette di raggiungere:

- *il fragment di allenamento*: allo *user*, cliccando sull'icona di riproduzione;
- *la modifica/elimina della scheda*: all'*admin*, cliccando sulle icone dedicate;
- *la lista di esercizi della scheda*: ad entrambi gli utenti.

Inoltre, già da subito è visibile il nome della scheda, con un testo che invita a cliccare per poter accedere alla lista degli esercizi. Notiamo anche che la schermata visibile allo *user* e all'*admin* è la medesima, con l'unica differenza di avere accesso ad icone con varie funzionalità (Figura 4.9 e 4.11). Lo *user* avvia la schermata

d'allenamento cliccando sull'icona di riproduzione e selezionando una difficoltà fra quelle proposte dal popup (Figura 4.10). L'*admin*, invece, può modificare o cancellare la scheda sulla quale si trova, semplicemente cliccando sull'icona specifica e completando le richieste del popup che si genera (Figura 4.12 e 4.13).

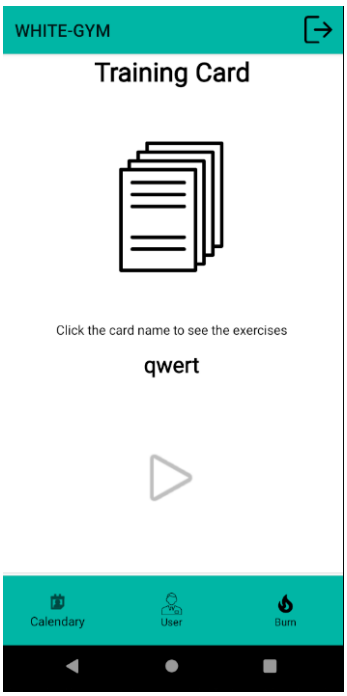

Figura 4.9. Istantanea del fragment relativo alla scheda dell'utente

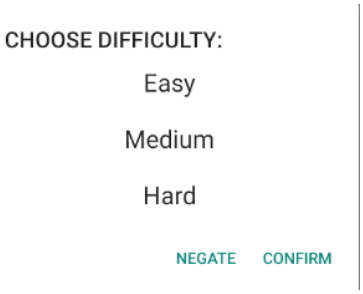

Figura 4.10. Istantanea del popup per la selezione della difficoltà

Figura 4.11. Istantanea del fragment relativo alla scheda vista dall'amministratore

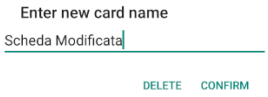

Figura 4.12. Istantanea del popup di modifica della scheda

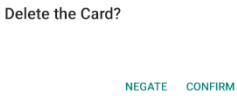

Figura 4.13. Istantanea del popup di cancellazione della scheda

4.2.5 Lista degli esercizi all'interno di una scheda

Questo frammento, come il precedente, riserva funzionalità diverse a seconda dell'utente; esso, infatti, consente di:

- *vedere la lista di esercizi*: ad entrambi;
- *leggere i dettagli di un esercizio*: ad entrambi, cliccando sul nome dell'esercizio;
- *aggiungere esercizi alla lista*: all'*admin*, cliccando sull'icona in alto;
- *rimuovere esercizi dalla lista*: all'*admin*, cliccando sull'icona dell'esercizio.

L'idea di visualizzare la lista di esercizi è nata per permettere allo *user* di allenarsi autonomamente, se lo preferisce, oppure di seguire un attività personalizzata (Figura 4.14 e 4.16). L'*admin*, a differenza dello *user*, ha la possibilità di aggiungere

o rimuovere esercizi dalla scheda tramite gli appositi popup di (Figura 4.15) e (Figura 4.17). L'aggiunta è, quindi, possibile selezionando gli esercizi tramite apposite checkbox. Chiaramente, questi ultimi, nel caso in cui risultino già presenti all'interno di una scheda, non possono essere inseriti una seconda volta. Per questo motivo abbiamo sviluppato un toast specifico, il quale avvisa l'utente quando si verifica tale eccezione (Figura 4.18).

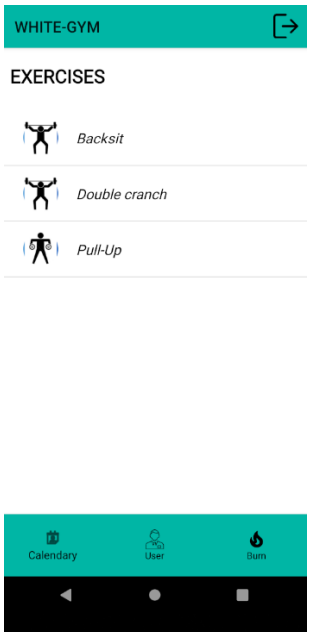

Figura 4.14. Istantanea del fragment relativo alla lista degli esercizi in una scheda user

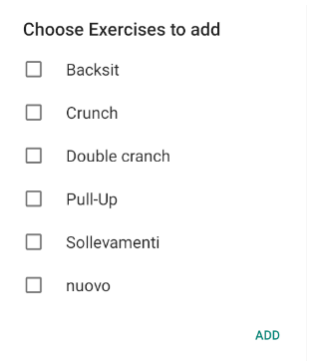

Figura 4.15. Istantanea del popup di aggiunta degli esercizi in una scheda

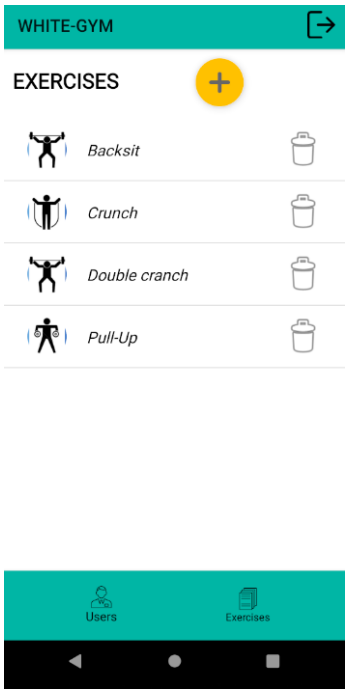

Figura 4.16. Istantanea del fragment relativo alla lista degli esercizi in una scheda admin

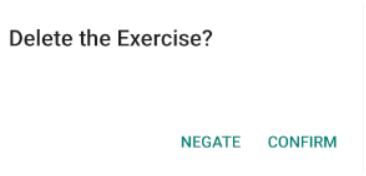

Figura 4.17. Istantanea del popup di cancellazione degli esercizi all'interno di una scheda

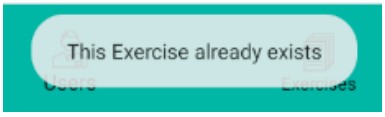

Figura 4.18. Istantanea del toast di errore

4.2.6 Allenamento

Solo lo *user* può accedere a questo frammento e, a differenza dell'*admin*, può iniziare e completare l'allenamento a seconda della difficoltà selezionata. Per iniziare sarà sufficiente cliccare sul pulsante "Start", che, una volta attivato, rende visibile un timer ed una ProgressBar, i quali accompagnano il cliente e ne migliorano l'esperienza.
Ogni esercizio ha una determinata durata; quando uno di essi termina si ha un inter-

vallo di tempo pensato per il recupero ed, infine, si svolgono gli esercizi rimanenti. Il tutto viene scandito dai due contatori che abbiamo sviluppato. Inoltre, una volta che viene avviato l'allenamento, il pulsante di "Start" viene sostituito da quello di pausa. Ogni volta che ci troviamo in modalità pausa il progresso del timer viene configurato al valore massimo della barra, per permettere allo *user* di avere maggior chiarezza in tutto e per tutto. Di seguito alleghiamo, nelle Figure 4.19, 4.20 e 4.21, delle istantanee che rappresentano al meglio il funzionamento di questo servizio.

Figura 4.19. Istantanea del fragment d'allenamento prima dell'avvio

4.2.7 Profilo

Cliccando sull'icona relativa al profilo, all'interno della BottomNavigationBar (Figura 4.22), lo *user* ha accesso alla sua area risevata. Al suo interno sono iniettati i principali dati personali con la possibilità di modifica e, a piacimento, di caricamento dell'immagine di profilo. La modifica di questi dati richiede la compilazione di una form specifica (Figura 4.24) che, se completata correttamente, permetterà in tempo reale di visualizzare i cambiamenti effettuati (Figura 4.23).

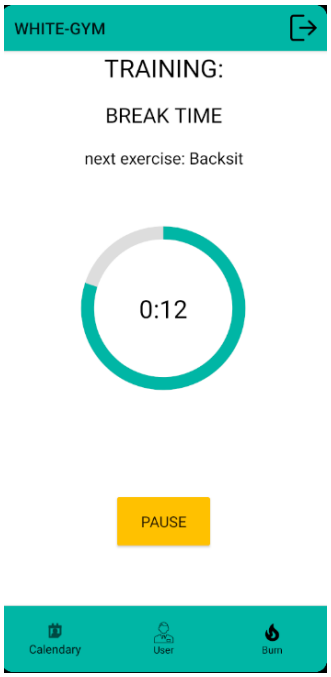

Figura 4.20. Istantanea del fragment d'allenamento dopo l'avvio

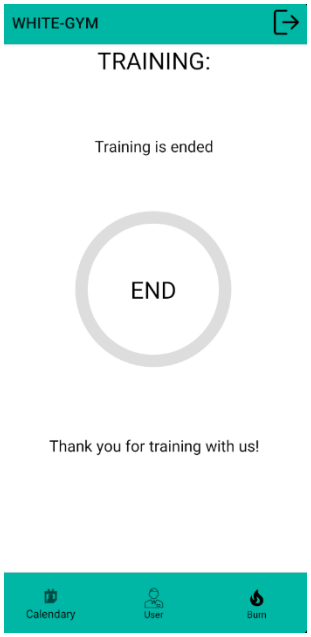

Figura 4.21. Istantanea del fragment d'allenamento terminato

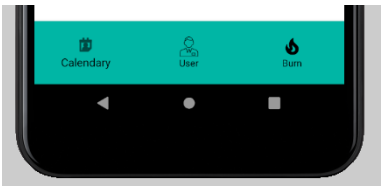

Figura 4.22. Istantanea della BottomNavigationBar dell'utente

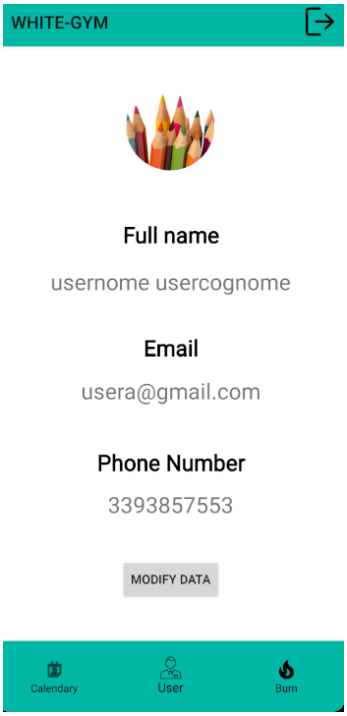

Figura 4.23. Istantanea del fragment del profilo

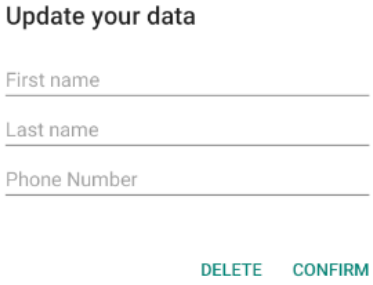

Figura 4.24. Istantanea del popup per la modifica dei dati nel profilo dell'utente

4.2.8 Calorie bruciate

Anche questo frammento è raggiungibile mediante la barra di navigazione, selezionando l'icona corretta. Questa schermata è accessibile solo allo *user*. Essa fornisce a quest'ultimo vari servizi, ovvero:

- *Visualizzare gli attuali progressi*; ciò avviene semplicemente avviando questa schermata, in quanto i dati saranno iniettati in automatico.
- *Resettare l'attuale progresso*, nel caso in cui si avesse la necessità di ricominciare daccapo.
- *Impostare un nuovo obiettivo*; se, eventualmente, viene raggiunto l'obiettivo o c'è la necessità di fissarne uno nuovo.
- *Calcolare il nuovo progresso*; lo *user*, sulla base degli esercizi svolti, potrà calcolare quante calorie ha bruciato.

Inoltre, tutte queste funzionalità sono disponibili con modifiche in tempo reale, sia a livello di interfaccia visibile (con textview dedicate), sia a livello di database. Le istantanee delle Figure 4.26-4.28 mostrano questa schermata e, più nello specifico, i popup dedicati alle modifiche e agli aggiornamenti.

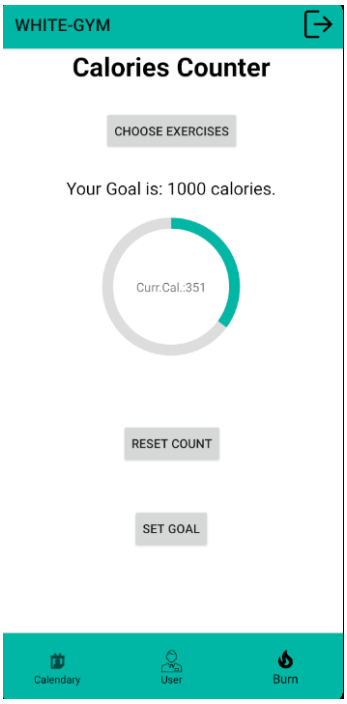

Figura 4.25. Istantanea del fragment relativo al calcolo delle calorie bruciate

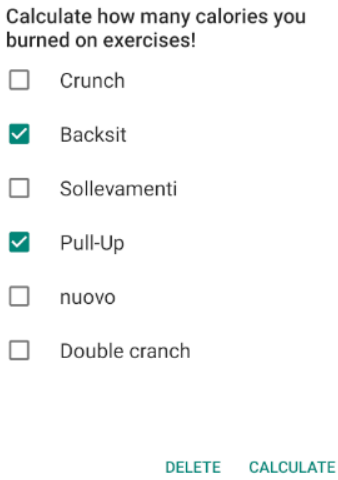

Figura 4.26. Istantanea del popup relativo al calcolo delle calorie con gli esercizi

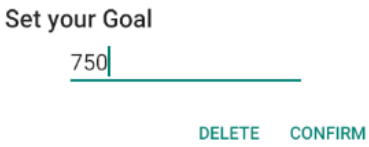

Figura 4.27. Istantanea del popup per impostare un nuovo obiettivo

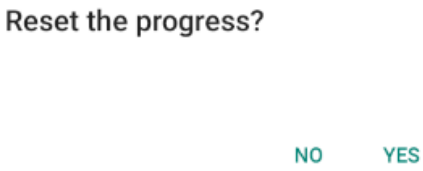

Figura 4.28. Istantanea del popup per azzerare l'attuale progresso

4.2.9 Lista degli utenti

La schermata principale dell'*admin* è quella che mostra la lista integrale degli utenti registrati all'applicazione con i quali l'amministratore stesso può interagire a piacimento cliccando sul loro nome. La ricerca all'interno di questo elenco è stata resa più facile dall'iniezione, mediante una recyclerview, del nome e del cognome dello *user*, assieme all'immagine del profilo. La spiegazione è resa molto più semplice e chiara dall'allegato in Figura 4.29.

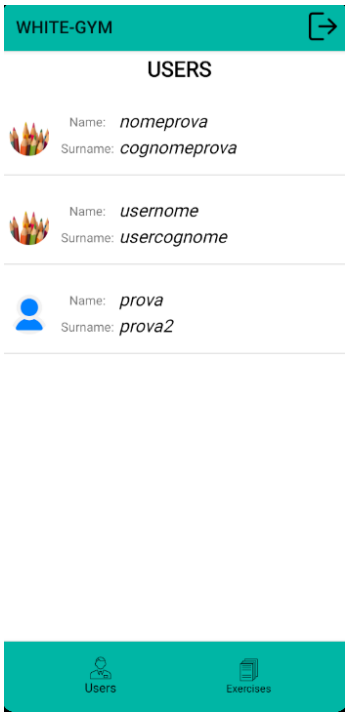

Figura 4.29. Istantanea del fragment relativo alla lista degli utenti registrati

4.2.10 Lista degli esercizi disponibili

Attraverso la BottomNavigationBar, configurata per l'*admin* (Figura 4.30), è possibile raggiungere il frammento che contiene l'intera lista degli esercizi disponibili. Da qui lo stesso amministratore può effettuare operazioni di creazione/modifica ed eliminazione dei singoli esercizi, attraverso dei popup specializzati per tali funzionalità. La logica utilizzata in questa interfaccia è molto simile a quella del frammento riguardante la lista degli esercizi all'interno di una scheda. Per una maggiore chiarezza, nella Figura 4.31, mostriamo la corrispettiva schermata.

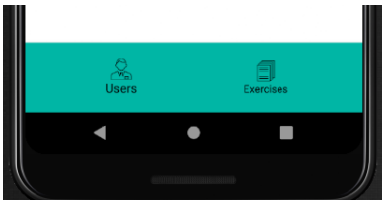

Figura 4.30. Istantanea della BottomNavigationBar dell'admin

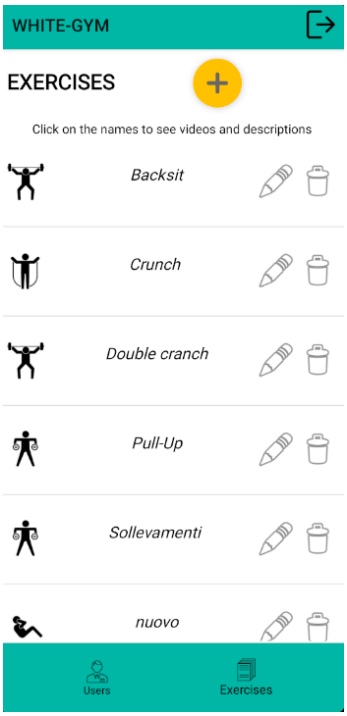

Figura 4.31. Istantanea del fragment relativo alla lista degli esercizi disponibili

4.2.11 Video e descrizione

Questa schermata, in particolare, contiene i dettagli di un singolo esercizio, ovvero descrizione e video esplicativo. La prima viene iniettata automaticamente su una textview dedicata, mentre il secondo può essere riprodotto cliccando sul pulsante di avvio riproduzione del Media Controller che è stato implementato (Figura 4.32). Il frammento è raggiungibile sia dallo *user* che dall'*admin*; ma quest'ultimo può selezionare un video dal dispositivo ed effettuarne l'*upload* direttamente sul database attraverso dei pulsanti specifici. Nell'intervallo di tempo in cui avviene questo caricamento sarà visibile una barra di progresso lineare, assieme ad un toast specifico. Questi ultimi permetteranno di capire quando l'operazione è ancora in fase di esecuzione, oppure è andata a buon fine, oppure, ancora, è fallita (Figura 4.33) e (Figura 4.35).

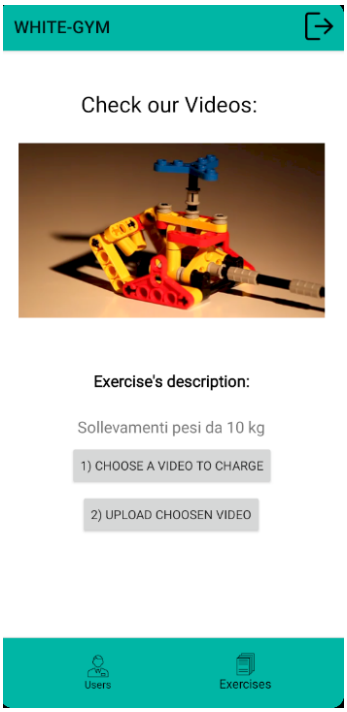

Figura 4.32. Istantanea del fragment relativo al video e alla descrizione

Figura 4.33. Istantanea della barra di caricamento

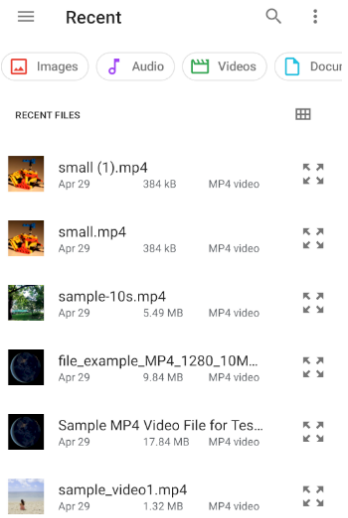

Figura 4.34. Istantanea dell'`onActivityResult`

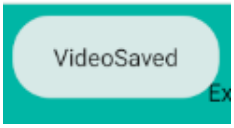

Figura 4.35. Istantanea del popup di conferma del caricamento

5

Analisi SWOT e confronto con app correlate

In questo capitolo affronteremo l'analisi conclusiva dell'applicazione, evidenziando i punti a favore e a sfavore della stessa, assieme alle opportunità di miglioramento e alle minacce che potrebbero presentarsi in futuro.

5.1 Definizione di analisi SWOT

L'analisi SWOT (Figura 5.1) è una tecnica di pianificazione strategica usata per lo studio dell'ambiente interno o esterno di un progetto su quattro aspetti: Strengths, Weaknesses, Opportunities e Threats [14]. La fase preliminare di quest'analisi si basa sul determinare:

- *Punti di forza*, ovvero le caratteristiche dell'applicativo che sono utili a raggiungere l'obiettivo prefissato;
- *Debolezze*, cioè gli attributi del prodotto che sono dannosi al fine di raggiungere l'obiettivo;
- *Opportunità*, ossia le condizioni esterne che sono utili a raggiungere l'obiettivo;
- *Minacce*, quindi le condizioni esterne che potrebbero recare danni alle performance.

Figura 5.1. Immagine rappresentativa dell'analisi SWOT

A partire dalla combinazione di questi singoli elementi sono definite le strategie da intraprendere per il raggiungimento dell'obiettivo. Gli analisti stabiliscono se quest'ultimo è raggiungibile rispetto ad una data matrice SWOT (Figura 5.2). Nel caso in cui non fosse raggiungibile con i mezzi disponibili allora deve esserne selezionato uno diverso e il processo và ripetuto nuovamente, se invece fosse raggiungibile, le SWOT sono utilizzate come input per la generazione delle possibili strategie con le quali ci è possibile lavorare.

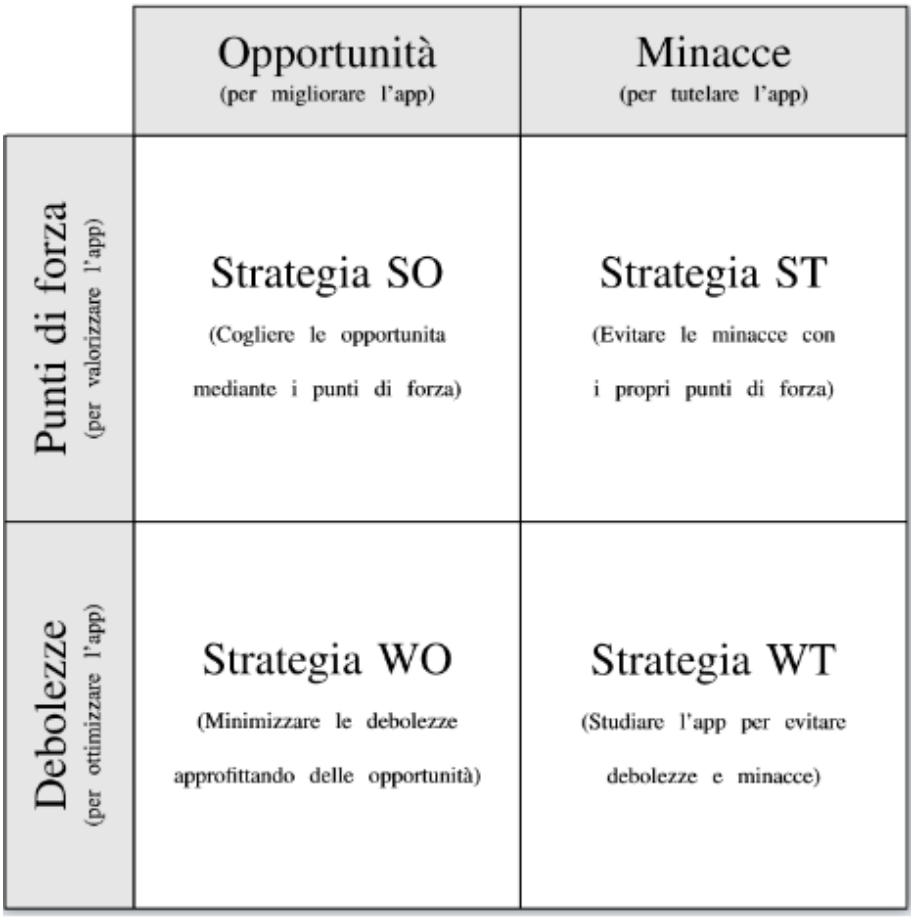

Figura 5.2. Matrice SWOT dell'applicazione

5.1.1 Strategia Strenghts-Opportunites

Questa strategia ha come unico obiettivo quello di conseguire le migliori opportunità con i propri punti di forza. Iniziamo definendo questi ultimi all'interno della nostra applicazione, ovvero:

- *Intuitività*, l'utilizzo di questa applicazione è possibile anche senza un vero e proprio manuale, ovvero è alla portata di tutti;
- *Rapidità*, l'impiego dei frammenti rende l'esperienza utente più dinamica che mai;
- *Non richiede sottoscrizioni a pagamento*, è un'applicazione totalmente gratuita, quindi fruibile dal grande pubblico;
- *Trasparenza ed autonomia*, ogni utente può decidere se svolgere l'allenamento seguendo le direttive e le funzionalità offerte, oppure lavorare autonomamente con la lista degli esercizi che gli è fornita;
- *Sicurezza*, l'impiego di Firebase mantiene i dati salvati in sicurezza.

Le prime opportunità che si potrebbero cogliere, a partire dai parametri sopra citati, sono quelle di collaborazione con una palestra. Infatti, quest'app è stata progettata per una singola attività e non per una pluralità di esse. Cosi facendo lo staff potrà monitorare tutta la clientela in maniera molto più diretta e semplice. Da questo tipo di legame si potranno poi sviluppare funzionalità nuove e molto significative; forniamo di seguito alcune idee che sono sorte durante lo studio:

- *Gestione degli abbonamenti*; realizzare un servizio di controllo degli abbonamenti, con la possibilità di rinnovo in app attraverso logiche di pagamento definite.
- *Monitoraggio attività fisica*; realizzare un'interfaccia in cui l'*admin* può esaminare l'andamento dell'attività fisica di ogni *user*.
- *Interfaccia per l'alimentazione*; la progettazione e lo sviluppo di questa funzionalità possono portare un ingente numero di fruitori dato che, vista la situazione di pandemia, la maggior parte della popolazione sta mostrando grande interesse verso la salute e, di conseguenza, verso l'alimentazione.
- *Comunicazione con lo staff*; la possibilità di comunicare con lo staff o con un personal trainer (via chat dedicata) rende contatto e supporto di gran lunga più immediati.
- *Registrazione e login con account social*; potersi registrare o accedere con account già definiti e conosciuti rende l'app molto più accogliente.

5.1.2 Strategia Strenghts-Threats

A partire dai punti di forza abbiamo la necessità di evitare le minacce che possono influenzare lo sviluppo e la pubblicazione della nostra applicazione. La più grande minaccia che si presenta in questi casi riguarda la sicurezza dei dati. Infatti, un utente non affiderebbe mai i suoi dati sensibili, come la carta di credito o altro, ad un applicativo non sicuro. La scelta di utilizzare Firebase è legata, di fatto, anche a questo concetto. Le regole di sicurezza dei database di Firebase permettono di tutelare i dati da terza parti non autorizzate e salvaguardare la stessa struttura che li contiene. Infine, nel caso in cui venissero implementate registrazione e login con account social, riscontreremmo sicuramente maggior protezione degli estremi di ciascun utente.

5.1.3 Strategia Weaknesses-Opportunities

Incentriamo adesso lo sguardo sulle opportunità di cui possiamo approfittare, al fine di minimizzare le debolezze che affliggono la nostra app. Possiamo rilevare questi difetti effettuando un esame molto critico del nostro lavoro. Di seguito riportiamo quelle che, secondo noi, possono dirsi caratteristiche sfavorevoli:

- *Sezioni molto elementari*; all'interno del progetto sono presenti alcune schermate o interfacce relativamente semplici e non molto convincenti.
- *Logica di avvio allenamento molto ripetitiva*; al di là della funzione di selezione difficoltà non rileviamo altri aspetti che possano interessare l'*user* o invogliarlo ad un continuo utilizzo dell'app.
- *Calcolo calorie con poca motivazione*; l'interfaccia di calcolo delle calorie svolge solamente le operazioni matematiche e non offre compagnia o motivazione all'*user*.
- *App concorrenti all'avanguardia*; nel mercato globale sono presenti applicazioni con funzioni sempre più immediate, intuitive e rapide, quindi c'è l'obbligo impellente di restare al passo con gli altri.

Successivamente andiamo a vedere come le opportunità possono intervenire per aiutarci. Una collaborazione può permetterci di sviluppare interfacce ancor più specifiche, con lo stile e le tematiche proprie della palestra. Tutto questo può fornire una buona iniezione di creatività e aggiornamento al nostro lavoro. L'implementazione del sistema di pagamento abbonamenti è un servizio che raramente si vede all'interno di un'app per la palestra; pertanto, questa novità porta anche identità ed unicità al complesso. Anche lo sviluppo della possibilità di comunicazione con lo staff può giovare e portare enormi migliorie, date dal contatto diretto con coloro che gestiscono l'applicazione e di conseguenza lavorano per migliorarla. C'è infine da osservare un aspetto in particolare, ovvero che molte di queste debolezze sono gestibili. Pertanto, in futuro, non nascondiamo la possibilità di un ulteriore sviluppo dell'app, a patto di fornire un servizio all'altezza delle aspettative. Solamente l'ultimo aspetto è da tenere sempre presente, ossia l'intero mercato concorrente. Dunque, le nostre funzionalità dovranno essere sempre innovative e mai riciclate, per poter raggiungere i migliori risultati possibili.

5.1.4 Strategia Weaknesses-Threats

In quest'ultima strategia elaboriamo i piani di difesa volti ad evitare che le minacce provenienti dall'esterno vadano ad alimentare quelli che sono i punti di debolezza interni. Per fare ciò conviene, quindi, confrontare la nostra applicazione con le altre e predisporre, a partire dai risultati ottenuti, una o più contromisure (se necessarie). Sviluppiamo tali argomentazioni in maniera esaustiva nella sezione successiva.

5.2 Confronto con app correlate

Quest'ultima parte interesserà quindi il confronto della nostra applicazione con i corrispettivi concorrenti del mercato attuale, col fine di studiare affondo le differenze

e le mancanze da poter poi levigare. Concluderemo infine con delle considerazioni sul percorso che abbiamo svolto.

5.2.1 Comparazioni con app delle grandi aziende

Durante lo studio riguardante l'analisi SWOT, nello specifico la strategia Weaknesses Threats, abbiamo analizzato varie app per comprenderne potenzialità e valori che le contraddistinguono. Riportiamo quelle che più ci hanno colpito:

- *MyFitness Pal*; app gratuita per Android e iOS di proprietà di Under Armour, che contiene in realtà tre app in una: dieta, contacalorie e coaching. Possiede un database con circa 5 milioni di alimenti da poter analizzare singolarmente per eleborare il miglior diario alimentare possibile, tenendo conto delle calorie assimilate.
- *Sworkit*; app freemium per Android e iOS, ossia permette un utilizzo gratis ma fornisce molti più servizi nel caso in cui venisse sottoscritto un account a pagamento. Essa offre oltre duecento tipologie di esercizi spiegati, in video, da veri personal trainer; inoltre, presenta anche una versione per bambini. Infine permette sia di stabilire la durata dell'allenamento sia le aree del corpo da allenare.
- *Nike Training Club*; app gratuita per Android e iOS, che propone workout realizzati da professionisti per tutti i gusti e le esigenze, con video guide e aiuti vocali di personal trainer virtuali. Inoltre, è possibile scegliere tra un'ampia serie di programmi di allenamento, a seconda del tempo che abbiamo a disposizione, dell'attrezzatura e del gruppo muscolare che vogliamo allenare.

Andiamo adesso ad analizzare le relazioni che ci sono tra la nostra applicazione e quelle citate, sempre tenendo conto delle possibilità di sviluppo che si hanno. La prima osservazione da fare è che il nostro progetto cerca di condensare le funzionalità delle tre app studiate, in un'unica. Il nostro obiettivo è quindi quello di fornire un'unica sorgente di dati, che però contenga tutte le funzionalità utili al pubblico. Per quanto riguarda l'impiego di personal trainer dedicati (reali o virtuali), lo sviluppo di un servizio di chat con lo staff permette una comunicazione dello stesso livello; pertanto, anche da questo punto di vista siamo difesi. Ennesimo aspetto da tener conto riguarda il fatto che la nostra app è totalmente gratuita ma, essendo progettata per una palestra in particolare, richiederà sicuramente l'abbonamento nei confronti di quest'ultima. In conclusione il nostro progetto non ha grandi differenze, dal punto di vista concettuale, con le opere delle grandi aziende. Per questa ragione, possiamo valutare il nostro percorso come totalmente positivo, in quanto ci ha permesso di lavorare con uno sguardo rivolto verso il mercato globale delle app di questa tipologia; di conseguenza, ci ha anche dato la possibilità di occuparci, dal punto di vista "pratico", di ciò che ci attenderà nel caso in cui decidessimo di intraprendere una carriera di questo tipo.

6

Conclusioni

In questa tesi sono stati affrontati gli argomenti relativi alla progettazione e all'implementazione di un'applicazione Android che permetta di svolgere attività fisica a distanza, dato l'impedimento di recarsi in palestra a causa delle restrizioni legate al COVID-19. Abbiamo approfondito in maniera esaustiva la situazione che stiamo vivendo e come ci siamo adoperati per lo sviluppo di questo applicativo, tenendo conto del fatto che, oggigiorno, l'attività fisica risulta essere una condizione fondamentale nella vita.

Inizialmente è stata presentata una panoramica su Android, partendo dalle sue origini e la sua evoluzione nel tempo, sino ad esaminarne la struttura e le particolarità. Il fine di questo capitolo è stato quello di introdurre un qualsiasi individuo al sistema operativo in questione riservando particolare attenzione ai benefici derivanti dall'impiego, alla sua diffusione a macchia d'olio nel mondo e al modo con cui è entrato nelle vite quotidiane di milioni di persone. Inoltre, sono state dedicate alcune sezioni all'ambiente di sviluppo Android Studio, permettendoci di vestire anche i panni di un generico programmatore che vuole avvicinarsi al contesto trattato.

Nel capitolo successivo ci siamo soffermati sugli studi preliminari alla progettazione e all'implementazione dell'app, quali la descrizione del problema e l'analisi dei requisiti (funzionali e non). Questo passo è stato cruciale ai fini della buona riuscita del lavoro, in quanto ci ha permesso di raggruppare le generalità e le necessità, oltre agli obiettivi, che ci siamo preposti di conseguire. Siamo partiti da una premessa sulle motivazioni che ci hanno spinto a lavorare su questa tematica, definendone utilità, vantaggi e caratteristiche, per approdare, infine, all'analisi vera e propria. Quest'ultima è stata portata a compimento dopo aver descritto e caratterizzato ampiamente i requisiti che l'applicazione, a nostro avviso, doveva inderogabilmente garantire.

Successivamente ci siamo concentrati sulla progettazione illustrando la mappa e un percorso dei file. Al fine di mettere in luce la logica di funzionamento del prodotto sono state, inoltre, specificate le sue nozioni cardine, ossia fragment, adapter e recyclerview. Oltre a ciò, tutte le azioni dell'utente contemplate dalla stessa applicazione sono state rappresentate in un diagramma dei casi d'uso e, a seguire, sono state analizzate una ad una. Infine ci siamo serviti della piattaforma Firebase per conservare e salvaguardare i dati sensibili con i quali abbiamo lavorato.

Una volta terminata l'esposizione della fase progettuale, abbiamo intrapreso

l'implementazione del software, mostrando il codice di tutte le funzioni che permettono la realizzazione dei requisiti. Per ciascuna di esse è stato poi esplicato in maniera esauriente il funzionamento delle istruzioni, studiate riga per riga, per avere la migliore chiarezza possibile. In prosieguo è stato presentato il manuale utente, che indica al lettore le istruzioni da eseguire per completare determinate azioni, allegando le istantanee relative alle interfacce dell'applicazione.

Infine, la tesi si conclude con un'analisi SWOT, attraverso la quale abbiamo sintetizzato gli aspetti più significativi del programma, ovvero: punti di forza, debolezze, opportunità e minacce. Le combinazioni di questi quattro aspetti ci ha permesso di effettuare una valutazione fortemente autocritica mirata a comprendere al meglio il mondo delle mobile app.

Riferimenti bibliografici

1. C. Arthur. Android fragmentation "worse than ever" – but OpenSignal says that's good. *The Guardian*, 30 luglio 2013.
2. P.R. Catena. Il caso Oracle vs. Google: tutelabilità e corretto utilizzo delle Application Programming Interfaces. In *Ciberspazio e Diritto*, pages 383–404. Enrico Mucchi Editore, 2017.
3. J. Chen, G. Han, S. Guo, and W. Diao. Fragdroid: Automated user interface interaction with activity and fragment analysis in android applications. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 398–409. IEEE, 2018.
4. L. Chung and J.C.S. do Prado Leite. On non-functional requirements in software engineering. In *Conceptual modeling: Foundations and applications*, pages 363–379. Springer, 2009.
5. L. Chung, B. Nixon, E. Yu, and J. Mylopoulos. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012.
6. N.S. Fatima, D. Steffy, D. Stella, and S.N. Devi. Enhanced Performance of Android Application Using RecyclerView. In *Advanced Computing and Intelligent Engineering*, pages 189–199. Springer, 2020.
7. N. Gandhewar and R. Sheikh. Google Android: An emerging software platform for mobile devices. *International Journal on Computer Science and Engineering*, 1(1):12–17, 2010.
8. P. Gilski and J. Stefanski. Android os: a review. *Tem Journal*, 4(1):116, 2015.
9. E. Grøtnes. Standardization as open innovation: two cases from the mobile industry. *Information Technology & People*, 2009.
10. S. Gunasekera. Android Architecture. In *Android Apps Security*, pages 1–12. Springer, 2012.
11. P. Kaur and S. Sharma. Google Android a mobile platform: A review. In *2014 Recent Advances in Engineering and Computational Sciences (RAECS)*, pages 1–5. IEEE, 2014.
12. L. Moroney. The firebase realtime database. In *The Definitive Guide to Firebase*, pages 51–71. Springer, 2017.
13. J. Park, Y.B. Park, and H.K. Ham. Fragmentation problem in Android. In *2013 International Conference on Information Science and Applications (ICISA)*, pages 1–2. IEEE, 2013.
14. D.W. Pickton and S. Wright. What's swot in strategic analysis? *Strategic change*, 7(2):101–109, 1998.
15. P. Samuelson. Apple v. Samsung and the upcoming design patent wars? *Communications of the ACM*, 59(7):22–24, 2016.

16. S. Sengupta and S. Bhattacharya. Formalization of UML use case diagram-a Z notation based approach. In *2006 International Conference on Computing & Informatics*, pages 1–6. IEEE, 2006.
17. H. Trung. Requirements analysis. *Openstax CNX*, 2009.

Ringraziamenti

In conclusione, desidero menzionare tutte le persone senza le quali questa tesi non esisterebbe nemmeno.

Un sentito grazie al Professore Domenico Ursino per la sua infinita disponibilità e la tempestività nelle risposte ad ogni mia incertezza; mi ha aiutato moltissimo ad affrontare nella giusta maniera la parte conclusiva del mio percorso di laurea, oltre che fornirmi prontamente la documentazione utile alla stesura di questo elaborato.

Un ringraziamento speciale va a mia madre, che ha sempre creduto in me, nonostante le mille problematiche che sono sorte durante questo lungo viaggio. Un saluto a mio padre, col quale dovrei scusarmi di molte cose dato che non sono stato proprio un figlio modello e spero di averlo fatto, in parte, così. Grazie anche a mia sorella; nonostante il nostro rapporto non sia sempre stato roseo, in cuor mio le voglio bene e sono sicuro che riuscirà a raggiungere i suoi obiettivi indipendentemente dai problemi. Non mi avete fatto mancare mai nulla, sacrificandovi ogni volta per me. Se voi siete felici lo sono anche io. Spero, sinceramente, di avervi reso fieri di me!

Un grazie infinite va a Giorgia, che ha saputo valorizzare ogni aspetto del mio carattere ed apprezzarmi per ciò che sono, una persona fragile ed imperfetta. L'ho conosciuta quando eravamo due ragazzini e, a quel tempo, non avrei mai pensato di arrivare fin qui, immaginiamo di arrivarci assieme a lei. Ti faccio un augurio: sebbene la tua strada sembri lunga e tortuosa sono sicuro che, col tuo fantastico carattere, supererai ogni tipo di ostacolo.

Un ringraziamento anche a Riccardo, amico di una vita e compagno di questo breve ma intenso percorso; ha saputo sempre ascoltare i miei problemi e parlarmi, a cuore aperto, dei suoi. Mi ha sempre aiutato e sono sicuro che troverà esattamente ciò che cerca; nel mentre, gli auguro le migliori felicità possibili.

Una menzione dovuta è al mio amico Daniele, col quale condivido un rapporto molto speciale che spero duri ancora per molto. É stato presente in ogni periodo della mia vita e mi ha aiutato in qualsiasi situazione; per questo motivo lo ringrazio enormemente, la sua compagnia è stata essenziale.

Ringrazio Marco, Iacopo, Davide, Nicola, Momo ed Eugenio per essere stati capaci di strapparmi un sorriso in qualsiasi momento della giornata, dentro e fuori dall'università. Un grazie anche a Mattia, che nell'ultimo periodo è sempre stato

ben disposto ad aiutarmi o a chiarirmi certi concetti con una “calma olimpica”. È anche merito vostro se sono arrivato fin qui.

Voglio ringraziare, in particolar modo, mia nonna Luciana, la quale sognava da tanto di vedere un nipote laureato e mi auguro di averle dato la gioia che si merita, in un periodo così arduo per tutti. Ti voglio tanto bene!

Inoltre ringrazio tutti i membri della mia famiglia a partire dai miei zii fino ad arrivare ai miei cugini, oltre a Claudia e Valeria, per il costante supporto e l’affetto che mi avete sempre trasmesso. Siete e sarete sempre parte della mia vita.

Spendo poi due parole per tutte le persone che non possono essere qui con me oggi, alcune delle quali non si sarebbero mai aspettate di vedermi raggiungere un simile traguardo. Volevo farvi sapere che vi penso sempre.

Infine ringrazio chiunque, nel corso della sua vita, abbia condiviso con me dal pensiero più superficiale a quello più profondo. Ognuno di voi ha recitato un ruolo fondamentale affinchè io arrivassi esattamente dove sono e, per questo, ve ne sarò eternamente grato.

Spero con tutto il mio cuore che questo possa essere solo uno dei tanti traguardi che mi aspettano e confido che ognuno di voi mi sarà vicino, qualsiasi strada io decida di intraprendere nel futuro. Grazie.