Università degli Studi di Napoli Federico II

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Classe di Laurea in Ingegneria dell'Informazione, Classe n. L.8

Corso di Laurea in Ingegneria Elettronica

Elaborato di Laurea

SVILUPPO DI UNA APPLICAZIONE ANDROID PER LA CONDIVISIONE DELLA POSIZIONE TRA MOTOCICLISTI

Relatore:
Ch.mo Prof. Tramontana Porfirio

Candidato:
Giliberti Giulio
Matr. N43/930

Anno Accademico
2017/2018

Indice

1 PREFAZIONE

Il CdL in Ingegneria Elettronica prevede per l'ambito informatico uno studio incentrato maggiormente sulla parte hardware dei sistemi che sulla parte software, con quest'ultima che, laddove affrontata, prevede un approccio di *"basso livello"*, più vicino al lato macchina che all'interfaccia utente.

Il progetto di tesi, consistito nello sviluppo di un Applicativo Android per la condivisione in tempo reale della posizione tra amici e gruppi di amici, pone come scopo per l'autore, un ampliamento delle conoscenze nell'ambito informatico verso un approccio di *"alto livello"*, e in particolar modo verso l'ambito della programmazione Java OOP.

Esso viene articolato in una prima fase dove l'autore concentra la propria attenzione nello studio dell'ambito di lavoro e degli strumenti a disposizione, e in una seconda fase dedicata alla progettazione vera e propria dell'Applicazione e al suo sviluppo.

La tesi è organizzata nel seguente modo: Il capitolo 1 introduce brevemente Android e il suo utilizzo. Il capitolo 2 entra nel merito dello sviluppo delle Applicazioni descrivendone la struttura e il funzionamento. Il capitolo 3 introduce Firebase e ne descrive le funzionalità. Il capitolo 4 si occupa della descrizione del progetto e dei requisiti. Il capitolo 5 illustra il processo di sviluppo. Il capitolo 6 mostra il progetto concluso e ne descrive l'utilizzo da parte dell'utente.

2 ANDROID

2.1 CENNI STORICI

Il sistema Android pone le sue radici nel 2002, quando Andy Rubin, insieme a Rich Miner, Nick Sears e Chris White, fonda la Startup Android Inc. per lo sviluppo di software per dispositivi mobili.

In questo scenario, l'azienda di Rubin, proponeva una nuova tipologia di Mobile O.S. basata su Kernel Linux, gratuito per chiunque volesse farne utilizzo ed open source. L'idea era quella di uniformare verso un unico standard, la miriade di software proprietari delle varie aziende produttrici di dispositivi mobili, in modo da indirizzare tutti gli sviluppatori di applicativi nella stessa direzione.

Figura 2.1

Nel 2005, spinta dall'evoluzione del mercato della telefonia mobile e dall'evoluzione della piattaforma Windows Mobile dell'azienda rivale Microsoft, l'azienda Google Inc. acquista e finanzia la Android Inc. trasformandola nella Google Mobile Division. È in questi anni che il team di Rubin intraprende lo sviluppo vero e proprio di Android (nella figura 2.1 è raffigurato il logo che contraddistingue il S.O. fin dalla sua nascita). Il 5 novembre 2007, ad un consorzio di aziende (Open Handset Alliance) che include la stessa Google Inc. e altre aziende tra cui HTC, Samsung, T-Mobile, Qualcomm e Texas Instrument Incorporated, avviene la presentazione ufficiale della prima versione di Android, Android 1.0. Nello stesso momento, Google pubblica la prima versione dell'Android Software Developer Kit, mettendo a

disposizione circa 10 milioni di dollari per coloro che avrebbero prodotto le migliori Applicazioni per il nuovo sistema. È così che Android comincia a diventare un vero e proprio ecosistema software, che cooperando con le altre realtà proprietarie di Google Inc (YouTube, Gmail, Google Maps, ecc..) diventa sempre più indipendente dall'hardware e aperto al mondo degli sviluppatori, pronto a garantire all'utilizzatore, flessibilità, adattabilità oltre ad un ampio ventaglio di personalizzazioni.

Poco tempo dopo viene rilasciata la prima release di aggiornamento, chiamata: Android 1.5 - *Cupcake*, (*curiosità;* A partire da questa versione di Android, Google comincerà a denominare le release successive con nomi di dolci, che si susseguiranno poi in ordine alfabetico). Questa release migliora l'utilizzo della fotocamera, il rilevamento della posizione GPS e implementa una keyboard virtuale, cambiamento sostanziale che permette agli smartphone di ridurre l'ingombro dovuto alla tastiera, a favore di display più grandi.

Android 1.6 – *Donut*, implementa un box di ricerca veloce e di ricerca vocale, l'indicatore di carica della batteria e la funzione *text-to-speach* multilingua.

Android 2.0 – Eclair, implementa il supporto all'interfaccia Bluetooth 2.1, una nuova interfaccia utente per il browser basata su HTML5, nuove funzionalità per il calendario, la sincronizzazione dei contatti e la gestione di Account email multipli.

Android 2.2 – *Froyo*, implementa il supporto per la creazione di hotspot (condivisione della connessione tramite Wi-Fi) e il supporto per Adobe Flash 10.1.

Android 2.3 – *Gingerbread*, implementa una nuova interfaccia utente e una nuova tastiera virtuale con selezione testo e funzionalità copia/incolla.

Android 3.0 – *Honeycomb*, estende le funzionalità di Android ai Tablet, implementa il tethering tramite Bluetooth ed un supporto per lo scambio rapido dei dati con il PC, inoltre migliora il multitasking, la gestione delle notifiche e i widget.

Android 4.0 – *Ice Cream Sandwich*, implementa funzionalità come la gestione delle cartelle, gli screenshot, lo sblocco con la fotocamera e lo *swipe* sullo schermo, aggiunge inoltre un supporto a Wi-Fi direct e Bluetooth HDP.

Seguono: Android 4.1 – *Jelly Beam*, Android 4.4 – *KitKat*

Android 5.0 – *Lollipop*, introduce una nuova interfaccia basata su Material Design, e la macchina virtuale Dalvik viene sostituita dalla ART.

Android 6.0 – *Marshmallow*, implementa Android Pay, un supporto nativo per schede di memoria SD e per il lettore di impronte digitali e aggiunge la possibilità di selezionare la banda di frequenze Wi-Fi.

Android 7.0 – *Nougat*, implementa il multi-window, nuove API grafiche (Vulkan), il supporto per la realtà virtuale e amplia le possibilità di interagire con le notifiche.

Android 8.0 – *Oreo*, implementa il supporto a display multipli, funzionalità *picture in picture* e *ambient display*, e altre migliorie sul lato grafico.

2.2 ARCHITETTURA

2.2.1 Il Kernel Linux

Come detto nel paragrafo precedente, Android è un sistema operativo *open source*, cioè, i codici sorgenti del progetto "Android" sono pubblici e qualsiasi sviluppatore può studiarli e apportarvi modifiche. Questa filosofia, cioè la filosofia dell'*open* source, viene rispecchiata in pieno nell'utilizzo di Kernel Linux. Il *Kernel*, il nucleo del sistema operativo, come tale, si occupa delle funzioni primarie del sistema, cioè la gestione del processore, della memoria e delle periferiche, la gestione della loro comunicazione, la creazione dei processi e la gestione della loro priorità *detto scheduling* (alla base del Multitasking e della Multiutenza) ecc…

Esso costituisce il livello più basso del sistema operativo ed è la parte software che fornisce un livello di astrazione tra l'hardware del dispositivo e i livelli superiori dell'architettura. Inoltre, tramite i drivers, permette alle *App* di utilizzare correttamente le periferiche hardware messe a disposizione dal dispositivo come il display, la fotocamera, gli speaker ecc…

Figura 2.2

2.2.2 Native Libraries

Subito dopo il *Kernel*, ad un livello superiore, vi sono una serie di librerie native realizzate in C/C++, come ad esempio: il *Surface Manager*, incaricato

della gestione delle View, ossia le componenti dell'interfaccia grafica, e della loro visualizzazione sul display.

L'Open GL ES, per la visualizzazione di grafica 3D.

Il Media Framework, per la gestione multimediale dei diversi CODEC per i vari formati di acquisizione e riproduzione audio e video.

FreeType, per la gestione dei Font (i caratteri).

SQLite, per la memorizzazione e la gestione delle informazioni tramite database.

SSL, libreria per la gestione dei Secure Socket Layer legati alla sicurezza del sistema operativo.

Web Kit, libreria per il browser engine, basato su HTML, CSS, JavaScript ecc...

Altre librerie di fondamentale importanza, sono le *Core Libraries*, assolutamente necessarie per l'esecuzione degli applicativi. Esse vanno a formare quello che è l'ambiente di esecuzione dell'Applicazione. Come avviene per Java, dove il codice per poter essere eseguito dalla *JVM (Java Virtual Machine)* deve essere prima compilato in Bytecode, per Android avviene la stessa cosa: l'applicativo viene prima tradotto in codice intermedio detto *DEX* e poi eseguito dalla *Dalvik Virtual Machine* (nei dispositivi più recenti viene utilizzata solamente la ART, mentre in alcuni precedenti l'utente poteva scegliere quale utilizzare).

Figura 2.3

2.2.3 Application Framework

Dopo le *Native Libraries,* al livello superiore, si trova *l'Application Framework*, che comprende un insieme di componenti di fondamentale importanza per ciascuna Applicazione come ad esempio: *l'Activity Manager*, che organizza il ciclo di vita delle *Activity*, organizzandole in uno *Stack*, a seconda dell'ordine di visualizzazione sul display. Il *Package Manager*, che gestisce il ciclo dell'intera Applicazione, a partire dalla sua installazione. Il *Content Provider*, incaricato della condivisione delle informazioni tra più processi. Il *Resource Manager*, che permette all'Applicazione di gestire una molteplicità di tipi di file diversi: file di configurazione, file di testo, immagini ecc… Il *View System*, per la gestione della renderizzazione dei componenti. Il *Notification Manager* che permette all'Applicazione di eseguire notifiche, sia visive, come l'apparizione di una *View* sulla schermata o l'accensione di un led, che sonore, come un semplice squillo. Il *Location Manager* che gestisce e mette a disposizione varie funzionalità legate al GPS.

Figura 2.4

2.2.4 Le Applicazioni

Nel livello finale vi sono le cosiddette "*App*" o "Applicazioni", cioè degli applicativi software, alcuni pre-installati nel sistema, altri installabili dall'utente a seconda delle proprie necessità. Esempi di *App* di sistema sono:

il *Launcher*, ossia l'applicativo software che gestisce la schermata principale del dispositivo (quella dal quale vengono avviate le altre *App*), i menu, gli stili grafici ecc... Il *Browser*, per la navigazione in internet, il calendario, la calcolatrice, l'*App* per i contatti, per le telefonate, per i messaggi ecc…

Un'*App* importantissima che permette all'utente di scaricare altre applicazioni, e quindi di personalizzare e di aggiungere nuove funzionalità al dispositivo è il *Play Store*. Quest'ultimo mette a disposizione dell'utente un numero elevatissimo di applicativi software, liberamente scaricabili, per svolgere le più disparate funzioni. Qualsiasi sviluppatore è libero quindi di aggiungere a tale *Play Store* la propria realizzazione e quindi di renderla scaricabile dagli altri utenti.

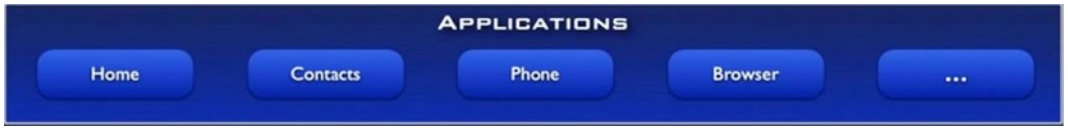


Figura 2.5

2.3 ANDROID THINGS

Da pochi anni l'Elettronica e l'Informatica si stanno evolvendo in una direzione denominata *IOT,* acronimo di *"Internet Of Things"*, o comunemente *"Internet delle cose"*. Le "cose" non sono altro che gli oggetti utilizzati nel quotidiano, come ad esempio le scarpe o l'orologio che indossiamo, che mediante il concetto di *Internet delle cose*, diventano degli *"Smart Objects"* ossia degli oggetti che, dotati di hardware e software interno, vengono dotati di una propria intelligenza. Un semplice paio di scarpe o un orologio, ad esempio, tramite *l'IOT*, diventano degli oggetti "Attivi", possono monitorare gli spostamenti della persona, i chilometri

percorsi, la velocità, ma anche segnalare un'emergenza, magari tramite il monitoraggio delle funzioni vitali.

In questa direzione è proseguito anche lo sviluppo del sistema operativo Android, che si è esteso negli ultimi anni, oltre che al mondo degli smartphone, anche ad altri oggetti. Ad esempio:

- Android Wear: sistema operativo progettato per dispositivi indossabili, come gli Smartwatch.
- Android TV: sistema operativo specifico per le TV, o più precisamente *"Smart TV"*.
- Android Auto: sistema operativo progettato appositamente per le automobili. Equipaggiato alla strumentazione dell'auto garantisce al guidatore alcune funzionalità come il navigatore GPS, la riproduzione di Audio e Video, e se collegato tramite Bluetooth allo Smartphone, invio e ascolto di SMS in arrivo, telefonate ecc.
- Google Glass: un paio di occhiali a realtà aumentata, progettati da Google Inc. e dotati anch'essi di sistema operativo Android.App

3 LO SVILUPPO DI UNA *APP*

3.1 IL LINGUAGGIO JAVA

Nonostante negli ultimi anni gli sviluppatori di Android stiano promuovendo un linguaggio proprietario chiamato *"Kotlin"*, similmente a quanto fatto dalla Apple In.c che da sempre utilizza un linguaggio proprietario chiamato *"Swift"*, il linguaggio preferito dagli sviluppatori Android continua ad essere *"Java"*.

Per sviluppare software in Java, è necessario predisporre il PC con l'ambiente di sviluppo, va cioè installato il *Software Development Kit (SDK)* dal sito ufficiale di *Oracle*. *L'SDK*, oltre a fornire allo sviluppatore alcuni strumenti utilissimi come quelli per il *Debug* del codice, risulta indispensabile in quanto senza di esso *l'IDE* non sarebbe in grado di compilare in *Bytecode,* il codice sorgente e quindi di eseguirlo.

3.2 ANDROID STUDIO

Il secondo passo da intraprendere per poter iniziare a sviluppare applicazioni per Android, è quello di installare l'*Integrated Development Environment (IDE)*.

Inizialmente *l'IDE* più utilizzato per lo sviluppo di *App* Android era Eclipse, lo stesso *IDE* utilizzato per il semplice sviluppo in Java di programmi desktop, con l'aggiunta di alcuni strumenti per lo sviluppo specifici per Android compresi in un kit chiamato *"Android SDK"*, installabile separatamente dallo sviluppatore.

Nel maggio 2013 nasce Android Studio, l'IDE ufficiale di Android, destinato a prendere il sopravvento su Eclipse, e già provvisto all'interno dell'Android SDK e di un emulatore di dispositivi Android *(Android Virtual Device)*.

Scaricato e installato, Android Studio, si presenta come in figura 3.1:

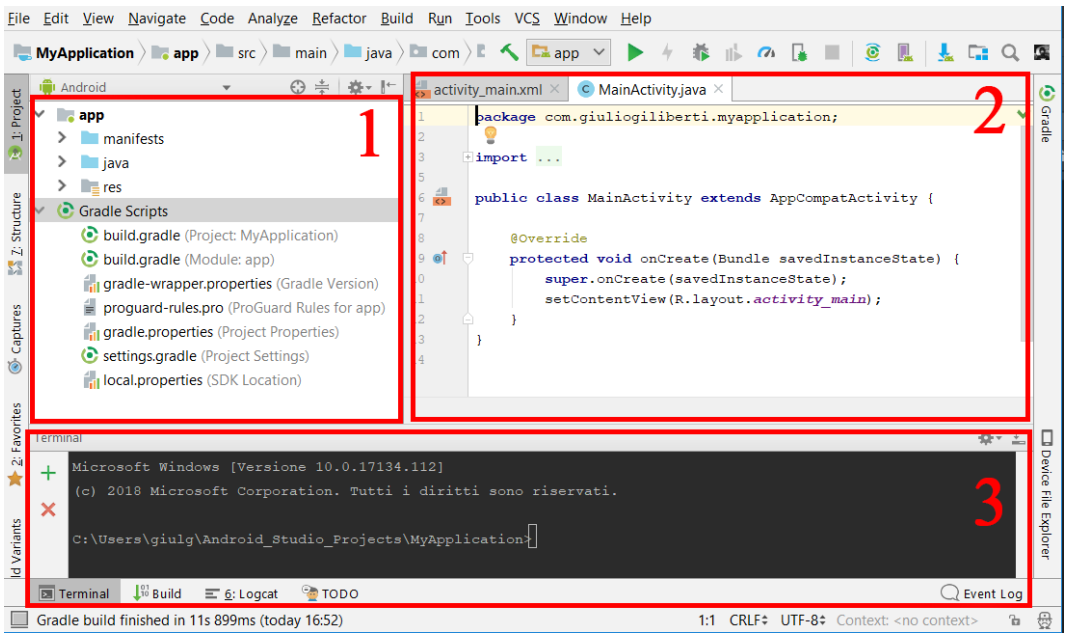


Figura 3.1

Sul lato sinistro, nel riquadro contrassegnato dal numero 1, vi è un pannello che permette allo sviluppatore di navigare all'interno delle cartelle e di visualizzare i file più importanti del progetto. Nel riquadro contrassegnato dal numero 2 vi è, a seconda dei casi, l'editor del codice, l'editor dei layout ecc... Nel riquadro numero 3 si trovano poi altri pannelli molto importanti come il Terminale, il Debugger, e il Logcat.

3.3 LE COMPONENTI DI UNA *APP*

3.3.1 Le Activity

Ciascuna Applicazione può essere composta principalmente da 4 componenti, le *Activity*, i *Service*, il *Broadcast Receiver* e il *Content Provider*. Le *Activity* sono la componente più a contatto con l'utilizzatore dell'applicazione. Esse costituiscono, in un certo senso, le varie schermate che l'utente si ritrova sul display del dispositivo, durante l'interazione con l'Applicazione. Per fare un esempio, basti pensare all'Applicazione messa a disposizione per le email. All'avvio dell'*App*, vi è una schermata principale *(Main Activity)* dove l'utente visualizza le ultime email ricevute, poi, tramite un tap sui vari bottoni, dalla *Main Activity*, si può passare in altre schermate (altre *Activity*): ad esempio la schermata per l'invio di una nuova email, la schermata per la visualizzazione delle bozze, la schermata di ricerca e così via. Generalmente ciascuna Activity è costituita da un file scritto in linguaggio *XML* che va a definire il *layout* della schermata e da un altro file scritto in linguaggio Java che estende la classe *Activity* e contiene tutto il codice relativo al funzionamento dell'applicazione in quella determinata schermata. Facendo ancora riferimento alla figura 3.1, si può accedere ai file *.xml* relativi al layout delle varie *Activity*, nel pannello a sinistra (riquadro numero 2), nella cartella *Res*, sotto la voce *Layouts*, mentre si può accedere al file *.java* nel medesimo pannello, nella cartella *Java*.

Il ciclo di vita delle *Activity* in una *Applicazione*, viene gestito tramite alcuni metodi della classe *Activity*, come i metodi *"onCreate", "onStart" , "onPause" , "onStop" , "onDestroy"*.

Ad esempio, quando l'*Activity* viene lanciata per la prima volta si invoca il codice contenuto nel metodo *"onCreate"*, quando l'*App* viene messa in background dall'utente, viene invocato il metodo *"onPause"*, e così via.

La figura 3.2 illustra il ciclo di vita delle *Activity*, ed i metodi che vengono eseguiti nelle varie circostanze.

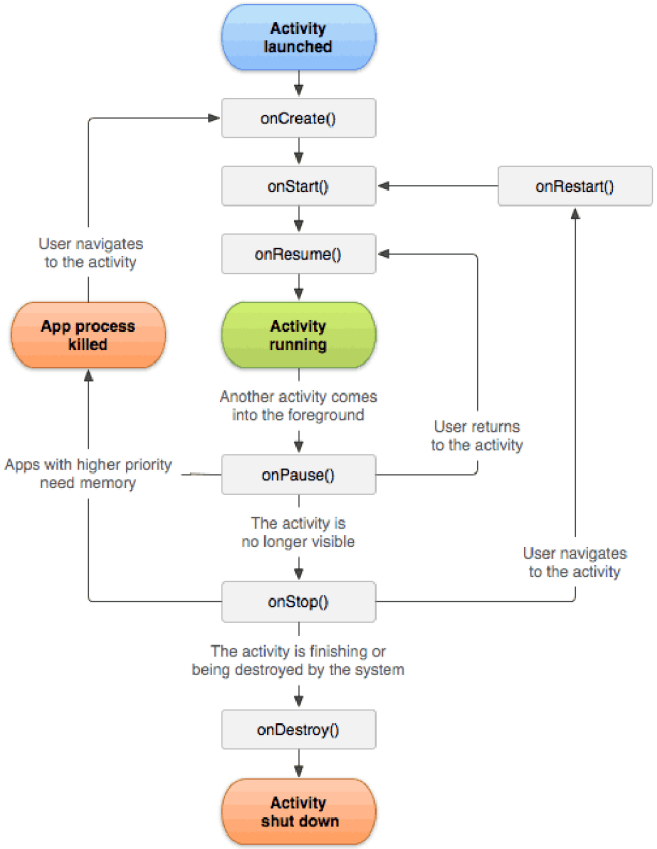


Figura 3.2

3.3.2 I Service

I *Service*, sono, a differenza delle *Activity*, dei processi che lavorano in *background*. Per questo motivo non sono corredati di un file di Layout *.xml*, e non risultano visibili direttamente all'utente. I file *.java* contenenti il codice relativo a ciascun *Service* altro non è che un'estensione della classe Service. Essi si trovano sempre nel pannello sinistro di Android Studio, lo stesso dove possono essere trovate le *Activity*. Un esempio di *Service* è quello utilizzato dalle *App* per la riproduzione musicale. Generalmente quando l'utente

imposta la riproduzione audio, poi è libero di navigare all'interno di altre Applicazioni perché la riproduzione viene gestita in *background* da un *Service*.

Un *Service* si può classificare in 3 tipologie:

- Foreground: l'esecuzione del processo resta in background ma l'utente è al corrente dell'esecuzione del *Service* perché avvertito generalmente da una notifica.
- Background: lavora totalmente in background e non è legato ad alcuna interfaccia grafica. La sua esecuzione è totalmente indipendente dall'utilizzo o meno dell'*App* da parte dell'utente.
- Bound: La sua esecuzione è legata al ciclo di vita dell'*activity* a cui è legato.

3.3.3 Il Broadcast Receiver

Un *Broadcast Receiver* è un ricevitore di informazioni. Esso riveste un ruolo chiave "nell'armonia" del dispositivo, cioè nella collaborazione tra le varie Applicazioni e il sistema. Si spiega il suo funzionamento con un esempio. Si suppone che sul dispositivo in uso, sia in esecuzione *Google Maps*, la famosa *App* per la visualizzazione di carte geografiche. Per mostrare la posizione dell'utente sulla mappa, questa richiede al dispositivo di localizzare l'utente tramite il sensore GPS, e di ottenere quindi le sue coordinate (latitudine e longitudine). Se contemporaneamente l'utente ha in esecuzione altre Applicazioni che richiedono tali coordinate, queste dovrebbero, in assenza del *Broadcast Receiver* e in maniera indipendente dalle altre *App*, interpellare di nuovo il sensore GPS del dispositivo e ciò causerebbe un dispendio inutile di risorse, in quanto le coordinate sono già state rilevate da *Google Maps*. Il *Broadcast Receiver* serve proprio ad ovviare a questa

eventualità. Se un processo richiede delle risorse di sistema, questi condivide i dati ottenuti con le altre Applicazioni che ne necessitano.

3.3.4 Il Content Provider

Il *Content Provider* di una Applicazione, gestisce l'accesso ai dati salvati dall'Applicazione stessa e da altre Applicazioni, secondo un determinato standard di sicurezza. Un esempio può essere l'elenco dei contatti dell'utente, che costituisce un elenco di dati, un database, utile per il funzionamento di più di un'Applicazione, basti pensare alla rubrica telefonica, all'*App* per le telefonate, quella per gli SMS, quella per le email, e così via. Ovviamente non tutti i dati sono condivisibili con le altre *App*, basti pensare alle password di accesso, oppure ai file di configurazione dell'*App* stessa, ecc... La definizione dei criteri di accesso (criteri di sicurezza) ai dati è anch'esso regolato dal *Content Provider.* Le operazioni che è possibile effettuare con i dati memorizzati sono le stesse che vengono utilizzate generalmente nella gestione dei database ossia scrittura, lettura, aggiornamento o cancellazione. I dati vengono individuati nella memoria mediante degli indirizzi univoci detti *URI*.

3.3.5 Le risorse

Oltre alle 4 componenti viste in precedenza, le Applicazioni dispongono in genere di vari tipi di risorse, tutte collocate nella cartella denominata *Res* (pannello di sinistra, figura 3.1).

Nella cartella *Drawable* ad esempio, è possibile collocare le immagini utilizzate dall'Applicazione come i loghi, le icone, gli sfondi, i disegni dei pulsanti ecc… Mentre nella cartella *Values* è possibile trovare dei file come

Colors.xml, *Styles.xml e Strings.xml*. Il primo contiene all'interno la definizione di tutti i codici dei colori utilizzati all'interno della Applicazione. Il secondo definisce lo stile dell'interfaccia grafica dell'Applicazione, come ad esempio permette di impostare la visualizzazione della ;*status bar*, dell'*action bar*, della *navigation bar* e così via. Il terzo file menzionato, riveste un ruolo importantissimo, contenendo al suo interno la definizione di tutti i testi e le scritte che appaiono durante l'utilizzo della *App*. Questa cosa risulta indispensabile sia quando lo sviluppatore deve utilizzare all'interno dell'Applicazione una stringa testuale più volte, sia quando l'Applicazione deve essere poi tradotta in altre lingue. Le stringhe di testo immagazzinate nel suddetto file comprendono anche quelle stringhe utilizzate internamente dallo sviluppatore, dove un errore di digitazione può dare vita ad errori e bug nel funzionamento dell'applicativo. Nel caso in cui il programmatore immette una sola volta la stringa e poi richiama, ogni volta che ne ha bisogno, solo il suo identificatore, viene scongiurata questa problematica (se si sbaglia a scrivere il nome dell'identificatore della stringa definita nel file *Strings.xml*, Android Studio segnala l'errore, cosa che non avverrebbe altrimenti).

3.3.6 I Layout

Già si è parlato dei *layouts* quando si è accennato alle *Activity* nei paragrafi precedenti, ma i *layouts* non sono necessariamente legati a queste ultime, se ne fa infatti uso ogni qual volta bisogna definire un'interfaccia grafica, che sia un menù, che sia una finestra di tipo *Dialog*, una semplice notifica o un *fragment* (se ne parla nei paragrafi successivi). La realizzazione di un *layout* è la medesima sia che si tratti di un *layout* per un'*activity*, sia che si tratti di

un *layout* per uno degli oggetti menzionati prima. Come si è visto per tutti i file .xml visti finora, anche i layout si collocano nella cartella *Res*.

L'interfaccia grafica che si va a realizzare tramite i *layouts*, ha come scopo principale quello di posizionare all'interno della schermata i vari elementi che la compongono, chiamati *View*.

Quando si modifica un *layout*, Android Studio, si pone in aiuto allo sviluppatore mostrando sulla sinistra una lista delle *View* più importanti, al centro un disegno approssimativo dell'interfaccia che si sta realizzando e sulla destra una lista di parametri configurabili per ciascuna *View* (vedi figura 3.2).

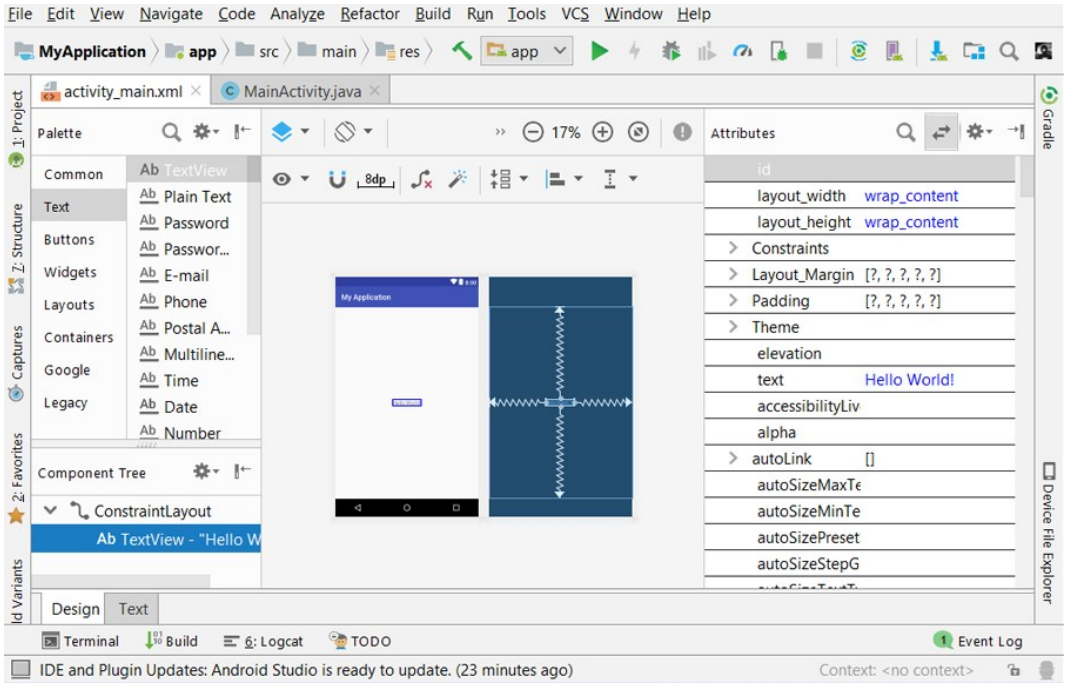


Figura 3.3

L'utente può quindi scegliere di realizzare il layout posizionando sulla schermata centrale le varie *View* o scrivendo manualmente il codice *XML*.

Esempi di *View* sono:

- *TextView*, per visualizzare a schermo delle stringhe di testo;

- *EditText,* dei box per l'immissione di informazioni testuali o numeriche;
- *Button, ToggleButton, Switch*, vari tipi di bottoni;
- *ListView* e *RecyclerView*, delle strutture grafiche a lista;
- *ImageView*, per la visualizzazione di immagini;

E così via.

Come detto più volte, la disposizione delle *View* può avvenire in vari modi attraverso i *layouts*. Anche questi ultimi sono vari, e utilizzabili a seconda dello scopo.

- *Linear Layout*, orizzontale o verticale, dispone le *View* in ordine di inserimento, una dopo l'altra, in una singola riga o colonna;
- *Absolute Layout*, dove le *View* vengono posizionate specificando per ciascuna, le coordinate *x* ed *y* espresse in *px* (pixel) o *dp* (densità di pixel);
- *Table Layout*, definisce il layout tramite una struttura tabellare;
- *Relative Layout*, come dice il nome, ciascuna *View* va posizionata in relazione alle altre;
- *Constraint Layout*, un'evoluzione più user-friendly del *Relative Layout*, più facile da utilizzare con l'interfaccia grafica di Android Studio;

3.3.7 I Fragments

Per aggiungere dinamicità all'Applicazione, lo sviluppatore, può decidere di far uso dei *Fragments,* ossia può decidere di "frammentare" l'*Activity* in più porzioni. Nota bene: un *Fragment* non può esistere se non correlato ad una o più *Activity*.

Per creare un *Fragment* all'interno di un'*Activity*, innanzitutto si posiziona nel layout dell'*Activity* la *View* opportuna per quel tipo di *Fragment*, poi si crea per quest'ultimo sia un file *.java* che estende la classe *Fragment*, sia un file layout *.xml*.

Mentre il resto dell'Activity resta invariata, la zona dedicata al *Fragment* può variare, permettendo allo sviluppatore di caricare all'interno, a seconda dell'evenienza altri *Fragment*. Un vantaggio dell'utilizzo dei *Fragment*, oltre all'aggiungere dinamicità all'interfaccia grafica, è quello di facilitare lo scambio di dati tra le varie parti dell'*App*. Quando non vengono utilizzati questi ultimi, i dati devono essere scambiati tra le *Activity* mediante gli *Intent*[1]. Con i *Fragment* risulta invece tutto più semplice, poiché lo scambio di dati tra più *Fragment*, viene gestito dall'*Activity* dove sono collocati.

3.3.8 Il file *AndroidManifest.xml*

Per ultimo, ma non ultimo in quanto ad importanza, viene trattato il file *AndroidManifest.xml*. Esso raccoglie tutte le informazioni basilari dell'Applicazione, come il nome del Package Java che identifica anche in maniera univoca quest'ultima, dichiara i permessi necessari come l'accesso alla posizione GPS e l'accesso ad Internet, dichiara di quali *API* Android l'*App* necessita per la sua dichiarazione, dichiara e descrive le componenti costitutive dell'*App* come le *Activity*, i *Service*, ecc…

Anch'esso realizzato in linguaggio *.xml*, il file *AndroidManifest.xml* realizza lo scheletro vero e proprio dell'Applicazione.

[1] Un Intent è un componente che può richiedere un'azione da parte di un altro componente come ad esempio quello di avviare o stoppare un'Activity o un Service.

3.4 L'APK

Tutte le componenti descritte nel paragrafo precedente, una volta ultimato lo sviluppo dell'Applicazione, vengono racchiuse e compresse in un archivio detto *Android Application Package*, avente estensione *.apk*. Esso permette l'installazione dell'applicativo sul dispositivo in uso un po' come avviene per i file *.exe* che consentono di installare i programmi sul pc. Tutto il codice scritto in Java su Android Studio viene pre-compilato e posto all'interno dell'*APK* sotto forma di codice. dex, oltre a ciò, poi, nel suddetto file, vi sono i layout e le varie risorse di cui l'*App* fa uso, come immagini, suoni e così via.

Tutte le Applicazioni vengono distribuite tramite i file *APK*.

4 FIREBASE

4.1 DESCRIZIONE

Firebase è una piattaforma online che mette a disposizione degli sviluppatori di applicativi software, sia web che mobili, una serie di servizi. Nata ad opera di James Tamplin e Andrew Lee che nel settembre 2011 fondarono la startup Firebase Inc., tale piattaforma viene successivamente acquistata, nell'anno 2014, da Google Inc., ed entra quindi a far parte della molteplicità di utility offerti da quest'ultima.

I servizi offerti da Firebase, noti come *"backend"*, cioè dei servizi con la quale l'utente non interagisce in maniera diretta, sono molteplici. I più importanti come il *Firebase Auth,* il *Realtime Database* e lo *Storage* vengono trattati nei paragrafi successivi, analizzando per ciascuno le classi e le librerie più importanti, messe a disposizione da *Google Inc*.

4.2 FIREBASE AUTH

Firebase Auth è un servizio messo a disposizione da Firebase che permette allo sviluppatore di gestire l'autenticazione degli utenti. Nella figura 4.1, è mostrata l'interfaccia web di Firebase[2] e in particolar modo la sezione del

[2] https://firebase.google.com/

servizio di autenticazione, denominata *"Metodo di accesso"* che permette allo sviluppatore di decidere quali provider utilizzare per l'autenticazione dell'utente. Si notano nella schermata, i *provider* dei social più famosi come *Facebook, Twitter, Google* e così via. Lo sviluppatore può inoltre implementare un processo di autenticazione con registrazione dell'utente tramite email e password, ed eventualmente inserire anche un processo di verifica dell'indirizzo email.

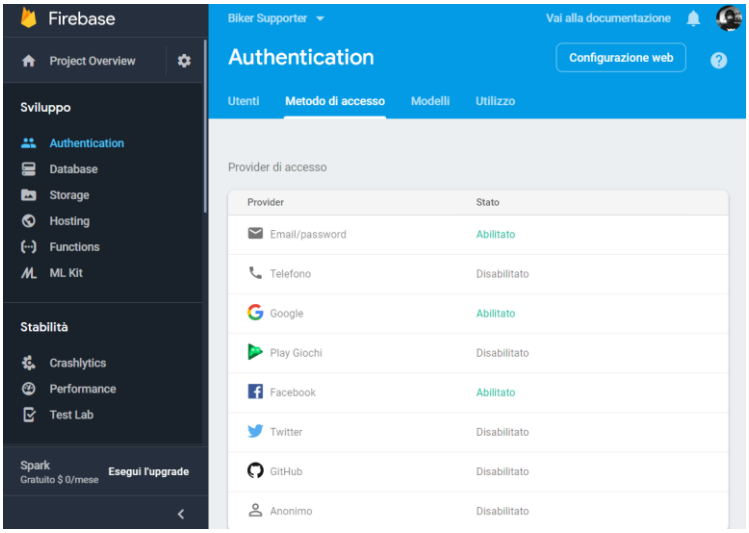


Figura 4.1

Cliccando sul link posto in alto a destra alla voce *"Vai alla documentazione"*, è possibile accedere ad una spiegazione dettagliata di tutto ciò che concerne Firebase per Android, a partire dalle librerie necessarie per il suo utilizzo come:

```
implementation 'com.google.firebase:firebase-auth:16.0.2'
```

da inserire sotto la voce *dependencies* nel file *build.gradle(Module:app)*[3].

[3] Il file build.gradle(Module:app) si trova nel pannello sinistro di Android Studio sotto la voce Gradle Scripts (vedi figura 3.2).

Sempre dalla documentazione ufficiale di Firebase, è possibile consultare una lista di tutte le classi e i relativi metodi presenti in ciascuna libreria, con alcuni esempi dettagliati di utilizzo.

Per implementare gran parte delle funzioni relative all'autenticazione degli utenti, una delle classi che possono essere utilizzate è la classe denominata *FirebaseAuth*. Il primo passo da fare è quindi quello di generare un'istanza di tale classe, tramite la creazione di un nuovo oggetto. Si sfruttano poi i metodi messi a disposizione, come:

CreateUserWithEmailAndPassword(String email, String password);

per la registrazione di un nuovo utente tramite email e password, oppure:

SignInWithEmailAndPassword(String Email, String Password);

per effettuare il *log in* dell'utente.

Anche per quanto riguarda il *log in* tramite *Facebook* o *Google*, vi è un'ampia documentazione sul sito ufficiale di Firebase.

4.3 REALTIME DATABASE

Il secondo servizio messo a disposizione da Firebase (figura 4.2) è un servizio Database basato su *NoSQL*, dove i dati immagazzinati vengono sincronizzati tra tutti gli utenti in *real-time* e restano disponibili sul server anche quando l'*App* viene chiusa. I dati sono organizzati secondo il formato *JSON*.

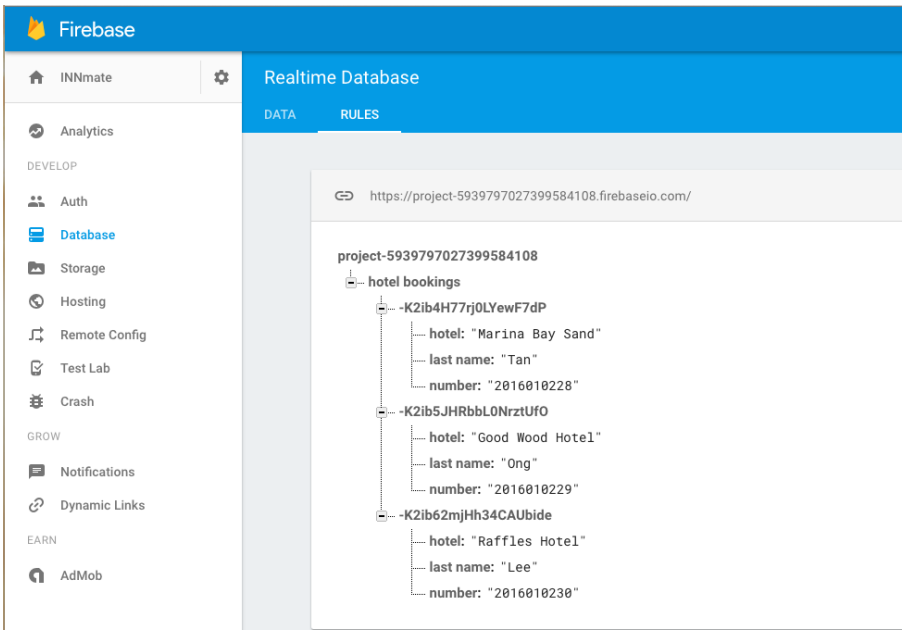


Figura 4.2

Per sfruttare tale servizio, è necessario, all'interno dell'Applicazione, inserire la seguente libreria:

```
implementation 'com.google.firebase:firebase-database:16.0.1'
```

si crea dopodiché un'istanza della classe *DatabaseReference*, i cui metodi permettono allo sviluppatore di leggere, scrivere, aggiornare o cancellare i dati presenti online sul database.

Esempi.

```
DatabaseReference myRef =
FirebaseDatabase.getInstance().getReference("nodo1");
```

Creazione dell'oggetto *myRef*, che punta alla voce *"nodo1"*, all'interno del database.

```
myRef.setValue("Hello, World!");
```

Scrittura nel database del valore da assegnare al nodo identificato da *myRef*. Tale valore può essere una stringa, un intero, una variabile booleana, e così via.

```
myRef.removeValue();
```

per cancellare il valore alla voce interessata.

```
myRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {

    }
    @Override
    public void onCancelled(DatabaseError error) {

    }
});
```

Metodo *addValueEventListener*, utilizzato per realizzare uno *"snapshot"* dell'area dati interessata, ogni qual volta questa presenta delle variazioni.

La struttura del database può essere molto articolata, formata da più nodi e sotto-nodi. Per muoversi all'interno dei livelli del database, si utilizza sempre la classe *DatabaseReference*, tramite il metodo.*child("...")*. Una volta individuato il nodo interessato, si può effettuare una tra le 4 operazioni elencate in precedenza.

4.4 STORAGE

Un altro dei servizi offerti da Firebase è uno *storage* online basato su *Google Cloud Storage*, cioè un contenitore per vari tipi di dati, come immagini,

suoni, video, file di testo e molto altro, messo a disposizione per gli sviluppatori di vari tipi di applicativi, *Android, IOS* o *Web Applications*.

Per implementare le funzionalità di *storage*, messe a disposizione da Firebase, è necessario importare la seguente libreria:

```
implementation 'com.google.firebase:firebase-storage:16.0.1'
```

Di tale libreria si può utilizzare poi la classe *StorageReference*. Nota: anche per lo *storage*, come per il *database real-time*, i dati vengono ordinati gerarchicamente in più livelli o cartelle. Il primo livello lo si può definire all'interno del metodo *getReference("...")*, mentre ai livelli inferiori si può accedere tramite il metodo *child("...")*.

```
StorageReference storageRef =
storage.getReference().child("images");
```

Per effettuare l'upload di dati all'interno dello *storage*, si può utilizzare il metodo *PutFile(uri)*, dove la variabile *"uri"* è una stringa di caratteri che identifica univocamente la risorsa da caricare sul server.

```
storageRef.PutFile(uri);
```

Per effettuare il download di dati dallo *storage*, si può utilizzare il metodo *GetFile(...)*;

```
File localFile = File.createTempFile("images", "jpg");
storageRef.getFile(localFile);
```

Infine, per cancellare un dato dallo *storage*, si può utilizzare il metodo *delete( )*.

```
storageRef.delete();
```

4.5 ANALYTICS

La possibilità di ottenere dei dati analitici dettagliati per la misura e l'analisi del modo in cui gli utenti interagiscono con l'Applicazione, è un altro importante servizio messo a disposizione da Firebase.

Sotto la voce *"Analisi"*, lo sviluppatore può avere accesso ad una notevole mole di strumenti. In *"Dashboard"*, ad esempio, lo sviluppatore può far riferimento ad alcuni grafici che gli permettono di ottenere molteplici informazioni circa l'utilizzo dell'Applicazione da parte degli utenti. Si può osservare di seguito, rispettivamente nelle figure 4.3, 4.4 e 4.5, l'andamento degli utenti attivi giorno per giorno, il grado di coinvolgimento degli utenti e il numero di arresti anomali.

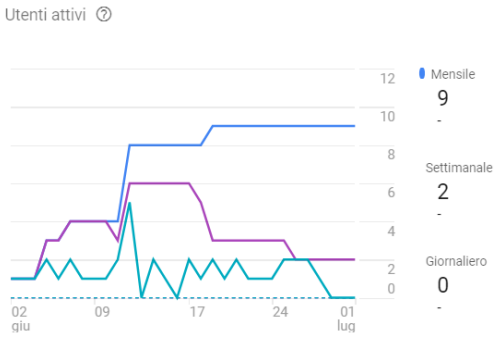


Figura 4.3

Figura 4.4

Figura 4.5

Per quanto riguarda questi ultimi, ossia gli arresti anomali dell'Applicazione, sotto la voce *"Crashlytics"*, vi è la possibilità da parte dello sviluppatore, di

implementare nella propria App, una funzione di *Report degli errori* atta a migliorare l'utilizzo dell'App stessa.

5 SPECIFICA DEI REQUISITI

5.1 INTRODUZIONE

In questo capitolo si inizia ad entrare nel merito dell'Applicazione da sviluppare, oggetto della tesi, denominata "BikersMap". Ed in particolar modo della così detta *"Specifica dei requisiti"* o *"SRS"*, acronimo di *"Software Requirements Specification"*, secondo lo standard *IEEE 830-1993.*

Per meglio comprendere quale possa essere l'utilità dell'*Applicazione* e la sua funzionalità, è doveroso fare un accenno all'ambito nel quale essa si andrebbe a collocare. Si dà innanzitutto una definizione di motociclista, così come inteso dall'autore, cioè, non semplicemente una persona posseditrice di una motocicletta, ma uno sportivo e un viaggiatore, una persona che ama ogni aspetto del motociclismo, da quello agonistico a quello sociale. Il motociclista ad esempio, è colui che la domenica, armato di casco, percorre 200, 300, 400 km per il solo e unico piacere di guidare e di vivere il mondo non da spettatore attraverso un vetro, come avviene in automobile, ma come protagonista, parte integrante della scena.

In questo concetto di motociclismo, si rispecchiano tante persone. Nella regione Campania ad esempio, regione di appartenenza dell'autore, esistono numerosi *Motoclub* e gruppi di motociclisti che puntualmente ogni domenica si riuniscono e trascorrono insieme giornate intere, percorrendo strade su strade ed esplorando ogni volta posti nuovi. L'esempio calzante, è quello dello *"Z Club Campania"*: un gruppo che conta oltre 1.400 motociclisti

provenienti da tutta la regione, nato quasi per caso nel 2014, fondato da semplici ragazzi con la passione comune per la motocicletta e diventato uno dei più importanti gruppi motociclistici del palcoscenico nazionale. Esso, in linea con quanto detto prima, è solito organizzare uscite di gruppo con cadenza settimanale, di 20-30, e talvolta anche 40-50 persone, con tragitti di lunghezza variabile dai 200 ai 600 km.

In tale contesto, l'applicativo mobile da realizzare, *BikersMap*, si pone come obiettivo quello di fornire una serie di servizi utili ai motociclisti, in ambito informativo e ricreativo ma in particolar modo riguardante la loro sicurezza.

Ad esempio, è frequente, durante le uscite di gruppo, specie se coinvolgenti tante persone, che una di queste resti indietro e non sappia la strada, oppure che sbagliando strada, queste si ritrovino sole senza la possibilità di avvisare il resto del gruppo, date le scarse probabilità che il motociclista possa sentir squillare il telefono durante la guida.

BikersMap, propone una soluzione visuale, ossia una mappa che visualizzi in tempo reale, la posizione di tutti gli amici motociclisti connessi.

5.2 DESCRIZIONE GENERALE

5.2.1 Prospettive

L'*Applicazione BikersMap* viene sviluppata con l'intento di essere totalmente autonoma e indipendente ma con la possibilità futura di integrarsi con altri applicativi software e hardware.

5.2.2 Funzionalità

BikersMap deve permettere all'utente di:

- Registrarsi ed effettuare *log in* e *log out*
- Creare un proprio profilo utente inserendo informazioni quali:
  - Nickname
  - Nome
  - Cognome
  - Età
  - Sesso
  - Città
  - Informazioni relative alla motocicletta posseduta
  - Foto profilo
- Gestire la propria cerchia di amicizie.
  - Ricercare un particolare utente registrato all'*Applicazione* per nome, cognome o nickname
  - Inviare richieste di amicizia
  - Eliminare un utente dalle proprie amicizie
  - Decidere se accettare o rifiutare una richiesta di amicizia in arrivo
  - Bloccare un altro utente
- Visualizzare su di una mappa alcune informazioni come:
  - La propria posizione
  - Lo status, *online* o *offline,* dei propri amici
  - La posizione degli amici *online*
  - Una piccola e compatta visuale del profilo di ciascun amico

5.2.3 Assunzioni e dipendenze

Il sistema software *BikersMap*, dovrà essere utilizzato su dispositivi dotati di sistema operativo Android.

5.3 ELENCO DEI REQUISITI FUNZIONALI

5.3.1 Registrazione dell'utente

Introduzione: Consente all'utente di creare il proprio account personale.

Input: Doppia immissione al fine di evitare errori di digitazione di un indirizzo email e di una password di almeno 6 caratteri, al massimo di 18 caratteri, che contenga almeno un numero, una lettera minuscola ed una maiuscola.

Elaborazione: Bisogna verificare che l'utente abbia inserito un indirizzo email valido, che i due indirizzi email corrispondano, che la password soddisfi i criteri di sicurezza descritti infine che le due password inserite coincidano. Dopodiché il sistema invia all'indirizzo indicato dall'utente, un email, al fine di verificare l'indirizzo stesso.

Output: Invio di un'email di verifica all'indirizzo indicato dall'utente. Immagazzinamento sul server dei dati relativi all'accesso (email e password) e registrazione nel database, sotto una chiave che contraddistingua univocamente l'utente, l'indirizzo email.

5.3.2 Verifica dell'email

Introduzione: Fornisce un livello di sicurezza maggiore all'*Applicazione*, verificando che l'utilizzatore sia realmente una persona fisica e impedendo che il database si riempia di account fittizi.

Input: Click dell'utente sul link ipertestuale ricevuto per email.

Elaborazione: Se l'utente ha verificato l'account, il sistema gli consente di proseguire con l'utilizzo dell'applicazione. Se l'utente non ha verificato l'account, il sistema dà due opzioni: re-inviare l'email contenente il link per la verifica, cancellare la registrazione.

Output: Accesso alle funzionalità dell'*Applicazione*.

5.3.3 *Log in* dell'utente tramite email e password

Introduzione: Consente all'utente di eseguire l'accesso, utilizzando l'indirizzo email e la password inseriti durante il processo di registrazione

Input: Indirizzo email, password.

Elaborazione: Il sistema verifica che l'utente abbia inserito un indirizzo email valido, poi fa altrettanto per la password. Se questi dati risultano validi, il sistema verifica l'esistenza sul server e la corrispondenza dei dati per effettuare il *log in*.

Output: Il *log in* dell'utente, cioè l'accesso alle funzionalità dell'*Applicazione*.

5.3.4 *Log in* dell'utente tramite i providers "Facebook" e "Google".

Introduzione: Consente all'utente di effettuare l'accesso o la registrazione in maniera rapida, tramite l'utilizzo dei suddetti providers.

Input: Il sistema autentica l'utente richiedendo il token di accesso ai due providers, *facebook* e *google*, tramite il tap, da parte dell'utente, sul corrispettivo bottone.

Elaborazione: Se il token di accesso è disponibile, il sistema verifica l'esistenza sul server dell'utente. Se l'utente non esiste, il sistema registra il token di accesso e l'indirizzo email associato al provider scelto, dopodiché registra nel database, sotto una chiave che identifichi univocamente l'utente, il suddetto indirizzo email. Se l'utente esiste, oppure dopo che questi si è registrato, il sistema gli consente di proseguire con l'utilizzo dell'Applicazione e di accedere alle sue funzionalità. Se il token di accesso non è disponibile, il sistema rimanda l'utente al sito web o all'Applicazione del provider richiesto per l'immissione dei dati di accesso.

Output: Il *log in* dell'utente, cioè l'accesso alle funzionalità dell'Applicazione.

5.3.5 Gestione dello smarrimento della password dell'utente.

Introduzione: Consente all'utente di accedere all'*Applicazione*, nonostante lo smarrimento della password.

Input: Indirizzo email dell'utente.

Elaborazione: Il sistema verifica che l'indirizzo inserito sia un indirizzo email valido, dopodiché invia all'utente, all'indirizzo email indicato, una nuova password.

Output: Invio all'indirizzo email indicato dall'utente, di una nuova password.

5.3.6 Modifica del profilo utente

Introduzione: Consente all'utente di modificare o completare il proprio profilo, con nickname, nome, cognome, foto profilo, età, sesso, città, marca, modello e cilindrata della moto posseduta.

Input: nickname, nome, cognome, foto profilo, età, sesso, città, marca, modello e cilindrata della moto posseduta

Elaborazione: Il sistema verifica la validità dei dati inseriti e ne effettua l'upload sul database soltanto in caso di risposta affermativa.

Output: Upload sul database dei dati inseriti.

5.3.7 Gestione del form di ricerca utenti

Introduzione: Consente all'utente di ricercare degli altri utenti tramite l'immissione del nickname, del nome o del cognome, o parte di essi.

Input: Immissione di almeno tre lettere costituenti nickname, nome o cognome dell'utente che si sta ricercando.

Elaborazione: Non appena l'utente inserisce almeno tre caratteri, il sistema ricerca all'interno del database l'esistenza di utenti registrati,

i cui nickname, nomi o cognomi contengano al loro interno l'input immesso dall'utente.

Output: Lista di utenti i cui nickname, nomi o cognomi, corrispondono alla stringa immessa in input.

5.3.8 Gestione del form di ricerca degli amici

Introduzione: Consente di navigare velocemente tra i propri amici, per facilitare l'utilizzo dell'*App* nel caso di liste amici notevolmente numerose.

Input: Stringa di caratteri senza nessuna restrizione particolare.

Elaborazione: Il sistema ricerca la stringa di caratteri immessa in input dall'utente, all'interno del nickname, del nome o del cognome di tutti gli utenti appartenenti alla lista amici.

Output: Lista degli utenti i corrispondenti alla ricerca.

5.3.9 Visualizzazione del profilo utente

Introduzione: Consente all'utente di visualizzare il profilo di altri utenti, contenente informazioni come nickname, nome, cognome, foto profilo, sesso, età, città, marca, modello e cilindrata della motocicletta posseduta. Poi consente di inviare una richiesta di amicizia, se l'utente non appartiene alla lista amici, e consente di rimuovere lo stesso se invece appartiene alla lista amici. Consente anche all'utente che

utilizza l'Applicazione di bloccare l'altro utente, impedendogli l'invio della richiesta di amicizia.

Input: richiesta dell'utente di visualizzare un determinato profilo, tramite un tap su determinati bottoni o collegamenti.

Elaborazione: Il sistema preleva dal database tutte le informazioni relative all'utente, cioè, nickname, nome, cognome, foto profilo, sesso, età, città, marca, modello e cilindrata della motocicletta posseduta dall'utente. Preleva la lista di amici dell'utente utilizzante l'Applicazione, e le informazioni circa gli utenti bloccati o che hanno bloccato l'utente stesso. Il sistema verifica poi se l'utente richiesto risulta essere presente nella lista degli amici e verifica la presenza di blocchi da parte dell'uno o dell'altro. A seconda dei casi il sistema permette all'utilizzatore dell'Applicazione di inviare una richiesta di amicizia o di eliminare/bloccare l'utente.

Output: Visualizzazione del profilo dell'utente.

5.3.10 Visualizzazione della lista amici

Introduzione: Gestisce la visualizzazione della lista degli amici dell'utente.

Input: nessuno.

Elaborazione: Il sistema preleva dal database la lista di tutti gli utenti appartenenti alla lista amici dell'utilizzatore dell'*App*, e alcune loro informazioni, cioè nickname, nome, cognome, foto profilo e status (online o offline).

Output: Visualizzazione su schermo della lista di tutti gli amici dell'utente, dando precedenza a quelli il cui status risulta essere "online".

5.3.11 Notifica di nuove richieste di amicizia

Introduzione: Consente all'utente di essere messo al corrente qual ora riceva da parte di un altro utente una richiesta di amicizia.

Input: Richiesta di amicizia da parte di un altro utente.

Elaborazione: Il sistema monitora il database, e appena questo registra una richiesta di amicizia da parte di un altro utente, riporta una notifica all'utente, che potrà quindi scegliere se accettare o rifiutare tale richiesta.

Output: Notifica su schermo della ricezione di una nuova richiesta di amicizia.

5.3.12 Invio della posizione al server

Introduzione: Consente all'Applicazione di salvare i dati relativi alla posizione dell'utente sul server, per far sì che altri utenti possano accedervi.

Input: Dati relativi alla posizione dell'utente, cioè latitudine e longitudine.

Elaborazione: Il sistema, una volta verificati i permessi per l'accesso e l'utilizzo del sensore GPS, riceve da quest'ultimo i valori di

latitudine e longitudine dell'utente e li registra sul server, in modo tale che altri utenti possano usufruirne.

Output: Registrazione sul database dei valori di latitudine e longitudine dell'utente.

5.3.13 Visualizzazione elementi della mappa

Introduzione: Consente all'utente di visualizzare sulla mappa la posizione degli utenti online, e su richiesta quelle degli utenti offline.

Input: Dati relativi alla posizione di ciascun utente, ricavati dal database.

Elaborazione: Il sistema scarica dal database i dati degli utenti presenti nella lista degli amici ogni qual volta questi presentano delle modifiche, (nickname, latitudine e longitudine). Dopodiché genera un marker personalizzato per ciascun utente contenente il relativo nickname.

Output: Disegno sulla mappa di tutti gli utenti online, nelle relative posizioni geografiche.

5.3.14 Spostamento della mappa sulla posizione di un amico

Introduzione: Consente all'utilizzatore di spostare rapidamente la mappa sulla posizione dell'utente individuato e di scegliere se fare in modo che tale posizionamento risulti permanente o no.

Input: Tap semplice o tap prolungato sul nome dell'utente.

Elaborazione: Il sistema elabora la richiesta calcolando la posizione in cui spostare la mappa e il relativo valore di zoom. Se l'utilizzatore effettua un tap prolungato, la mappa risulterà sensibile agli spostamenti dell'utente in questione ogni qualvolta ne si registrerà un cambiamento sul database.

Output: Spostamento della mappa.

5.3.15 Spostamento della mappa sulla posizione dell'utente

Introduzione: Consente all'utente di scegliere se la mappa debba far visualizzare o meno la propria posizione, seguendo gli eventuali spostamenti.

Input: Tap su di un bottone specifico.

Elaborazione: Il sistema elabora la richiesta calcolando la posizione in cui spostare la mappa e il relativo valore di zoom.

Output: Spostamento della mappa.

5.4 ELENCO DEI REQUISITI PRESTAZIONALI

- Far sì che la velocità di aggiornamento della posizione dei *markers* sulla mappa sia tale da consentire una corretta visualizzazione di quest'ultima. Si sceglie per tanto una modalità di aggiornamento asincrona.

- Far sì che i consumi della batteria risultino contenuti.
- Far sì che i consumi relativi alla connessione dati risultino contenuti.

5.5 ELENCO DEI REQUISITI DI SICUREZZA

- Far sì che soltanto gli utenti registrati possano utilizzare l'Applicazione.
- Far sì che soltanto gli utenti che risultano tra gli amici possano accedere alla posizione dell'utente.

6 PROGETTO E SVILUPPO

6.1 GLI STRUMENTI UTILIZZATI

Per lo sviluppo dell'Applicazione *"BikersMap"*, si è scelto di utilizzare:

- l'*IDE* ufficiale di Android, cioè *Android Studio* (paragrafo 3.2), come detto già dotato di un emulatore interno di dispositivi mobili.
- Si è scelto di adottare come dispositivo da emulare, il dispositivo *Nexus 5* sviluppato da *Google Inc.* e prodotto da *LG* nel 2013.
- Firebase, per le svariate funzionalità offerte e ampiamente descritte nel corso del capitolo 4.
- I metodi e le classi offerte dalle seguenti librerie:

```
'com.google.firebase:firebase-auth:16.0.1' //FIREBASE AUTH
'com.facebook.android:facebook-android-sdk:[4,5)' //FACEBOOK AUTH
'com.google.android.gms:play-services-auth:15.0.1' //GOOGLE AUTH
'com.google.firebase:firebase-core:16.0.0' //FIREBASE DATABASE
'com.google.firebase:firebase-database:16.0.1'
'com.google.firebase:firebase-storage:16.0.1' //FIREBASE STORAGE
'com.squareup.picasso:picasso:2.5.2' //IMAGES MANIPULATION
'com.google.maps.android:android-maps-utils:0.4+' //GOOGLE MAPS
'com.google.android.gms:play-services-maps:15.0.1'
'com.google.android.gms:play-services-places:15.0.1'
'com.android.support:design:27.1.1 //ANDROID VIEWS
```

NB. si è scelto di destinare l'Applicazione a dispositivi dotati di *"minSdkVersion"* cioè *API Level* pari a 21 (*Android 5.0)*.

6.2 LE ACTIVITY

BikersMap, viene strutturata in 5 Activity: *MainActivity, LoginActivity, CreateNewAccountActivity, ProfileActivity e FriendsActivity*.

6.2.1 La MainActivity

È l'Activity principale dell'Applicazione, ed è la prima ad essere eseguita quando quest'ultima viene aperta. Permette all'utente di usufruire delle maggiori funzionalità dell'*App*.

In essa, sarà presente centralmente la mappa dove l'utente può visualizzare tramite dei markers la posizione dei propri amici, aggiornata in tempo reale.

Sarà presente un menù laterale, accessibile con uno *swipe* da sinistra verso destra, sul bordo sinistro dello schermo, in cui vi è la lista degli amici, ed un bottone che se premuto rimanda alla *FriendsActivity*.

Se l'utente effettua poi un tap sul nome di un amico, in tale lista, la mappa si sposta nella posizione dove si trova l'amico in questione.

Se l'utente effettua invece un tap prolungato, l'amico in questione viene selezionato, e la mappa si centrerà sulla sua posizione ogni qual volta venga rilevato un suo spostamento nel database.

Sarà presente, un bottone (si tratterà di un *ToggleButton)*, nell'angolo superiore destro della mappa, che se attivato, centrerà quest'ultima sulla posizione dell'utente che sta utilizzando l'Applicazione ogni qual volta il sensore del *GPS* rilevi uno spostamento.

La mappa dovrà essere gestita in modo tale che più utenti, compreso l'utilizzatore, possano essere selezionati contemporaneamente, cioè si dovrà

calcolare tramite una funzione, la latitudine, la longitudine e lo zoom opportuno della mappa, in modo tale da farvi rientrare i markers di tutti gli utenti selezionati. Tale funzione dovrà entrare in gioco ogni qual volta si registri uno spostamento di uno qualsiasi di essi.

6.2.2 La LoginActivity

Permetterà all'utente di effettuare il *log in*. L'interfaccia sarà strutturata con due EditText per l'immissione dei dati di accesso dell'utente (email e password), un pulsante per effettuare il *log in*, un pulsante per la creazione di un nuovo account, un pulsante per il recupero della password e altri due pulsanti per il *log in* tramite *Facebook* e *Google*. Quando l'utente tocca il pulsante *"Log in"*, si esegue una funzione che dovrà determinare se i dati immessi dall'utente sono validi, e in seguito se vi è una corrispondenza sul server di Firebase. Quando l'utente tocca il pulsante "*Crea nuovo account*", viene rimandato alla *CreateNewAccountActivity*.

6.2.3 La CreateNewAccountActivity

Permetterà all'utente di creare un nuovo account. L'interfaccia grafica sarà strutturata con quattro EditText, per l'inserimento dell'indirizzo email, la verifica del corretto inserimento di quest'ultimo, la password e la verifica del corretto inserimento di quest'ultima, ed un pulsante *"Crea account"*. L'utente inserirà i dati, dopodiché pigerà il suddetto pulsante. In quel momento, si richiamerà una funzione che dovrà determinare se i dati inseriti rispettano i dovuti criteri. In caso di risposta affermativa verrà eseguita un'altra funzione per la creazione dell'account e l'invio di una email di verifica. Quindi l'interfaccia si aggiornerà mostrando un messaggio nel

quale si chiederà all'utente di effettuare un click su di un link ricevuto tramite email, e due pulsanti, uno per re-inviare l'email ed un altro per proseguire. Al tocco del pulsante *"Prosegui"*, l'utente, se verificato, sarà rimandato alla *ProfileActivity*, e verrà salvato sul database, sotto una chiave univoca identificata con l'*UID* dell'utente, l'indirizzo email.

6.2.4 La ProfileActivity

Permetterà all'utente di inserire i propri dati del profilo. Sarà strutturata con una ImageView che se toccata permetterà all'utente di selezionare un'immagine profilo, varie EditText per l'inserimento di informazioni come Nickname, Nome, Cognome, ecc... Ed un pulsante per proseguire. Al click su tale pulsante, una funzione verificherà la validità dei dati inseriti, ed in caso di esito positivo, eseguirà l'upload dei dati sul database, dopodiché rimanderà l'utente alla *MainActivity*.

6.2.5 La FriendsActivity

Permetterà all'utente di gestire le proprie amicizie. Sarà presente una ListView per la visualizzazione dell'elenco degli amici, un EditText per eseguire una ricerca tra questi, ed un pulsante che se cliccato lancerà una finestra *Dialog* per la ricerca di altri utenti, chiamata *SearchUsersDialog*. La visualizzazione degli elementi della ListView, sarà sensibile alle variazioni della stringa inserita nell'EditText per la ricerca. Se la stringa sarà vuota, verranno visualizzati tutti gli utenti. Se la stringa non sarà vuota, verranno visualizzati soltanto quelli i cui nickname, nomi o cognomi contengono la stringa stessa. Se l'utilizzatore dell'*App* toccherà il nome di un utente nella ListView, si aprirà un altro Dialog, l'*UserProfileDialog*, che mostrerà il

profilo dell'utente scelto. In caso arrivi una richiesta di amicizia, comparirà all'utente un testo di avviso. Se egli vi cliccherà sopra gli si aprirà un ulteriore Dialog, il *FriendReqsDialog*.

6.3 I DIALOG

6.3.1 Il SearchUsersDialog

Consentirà all'utente di effettuare la ricerca di altri utenti nel database che non appartengono alla propria lista amici. Sarà strutturato con un EditText per l'immissione del nickname, del nome o del cognome e con una ListView per la visualizzazione dei risultati della ricerca. Anche in questo caso, se l'utente effettuerà un click sul nome di un utente, gli si aprirà il Dialog, *UserProfileDialog*.

6.3.2 L'UserProfileDialog

Mostrerà all'utente il profilo di un altro utente. Sarà strutturato con una ImageView per la visualizzazione della foto profilo, una serie di TextView per la visualizzazione di alcune informazioni come nickname, nome, cognome, età ecc., da un pulsante per l'invio di una richiesta di amicizia o per l'eliminazione dell'utente dalle proprie amicizie, a seconda dei casi, e da un bottone per il blocco.

6.3.3 Il FriendReqsDialog

Il FriendReqsDialog mostrerà all'utente la lista di richieste di amicizia ricevute. Sarà strutturato con una ListView che conterrà la suddetta lista, con la possibilità per ciascuna di esse di scegliere se accettare o rifiutare.

6.4 I LAYOUT

Nel seguente paragrafo verranno illustrati tutti i *layout* adottati nello sviluppo dell'*App*.

- **MainActivity**

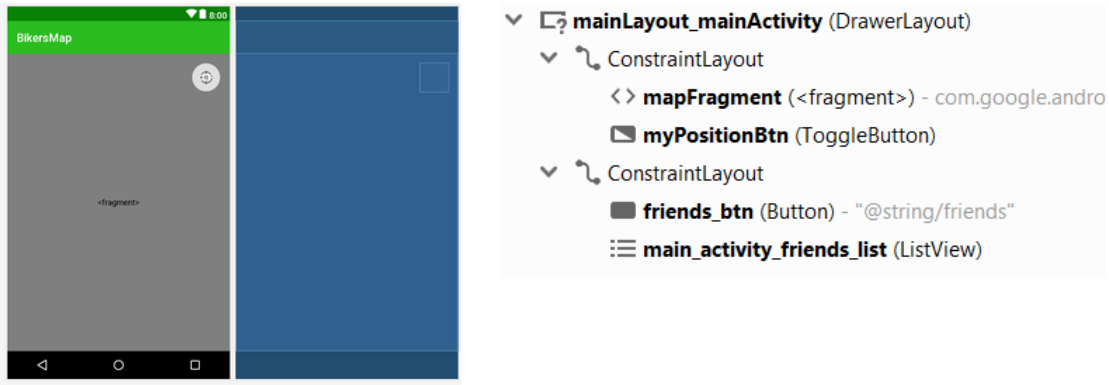
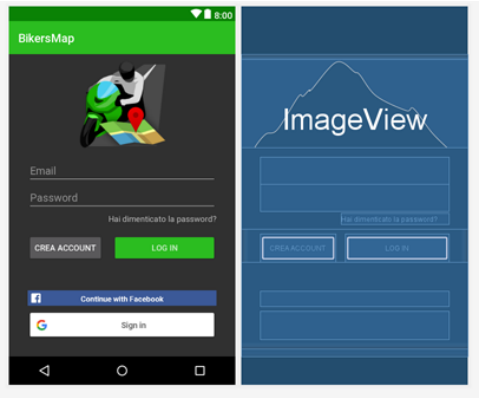


Figura 6.1

Per quanto riguarda la *MainActivity* (figura 6.1) si è utilizzato come layout principale, un DrawerLayout, che consente l'implementazione del side menù. Il DrawerLayout si utilizza specificando altri due layout al suo interno, uno per la schermata principale e l'altro per il side menù. Per entrambi si è scelto un ConstraintsLayout. In quello relativo alla schermata vi è un MapFragment per la visualizzazione della mappa ed il ToggleButton

di cui si è parlato nei paragrafi precedenti. In quello relativo alla side bar vi è invece un bottone semplice ed una ListView.

- **LoginActivity**



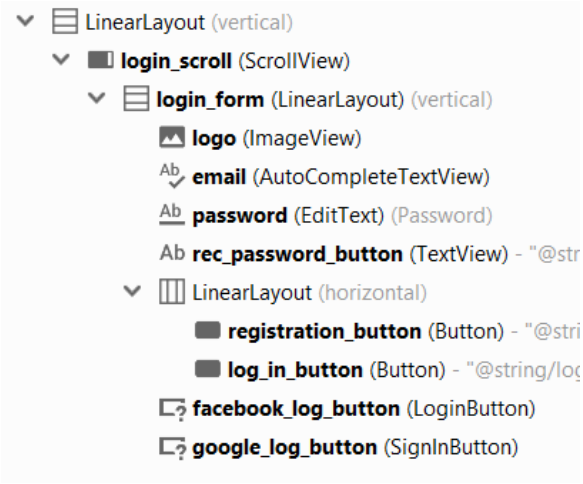
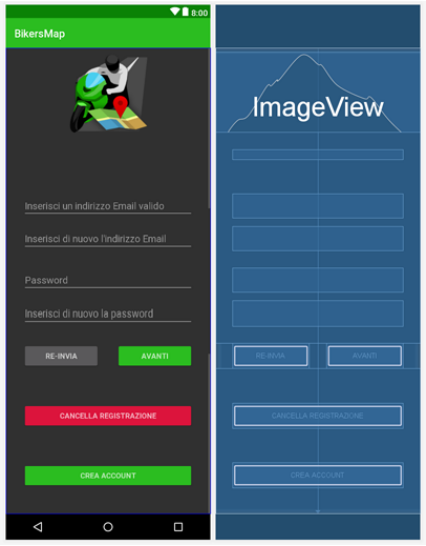


Figura 6.2

Per quanto riguarda invece la *LoginActivity* (figura 6.2), si è scelto di utilizzare un LinearLayout con all'interno una ScrollView.

- **CreateNewAccountActivity**



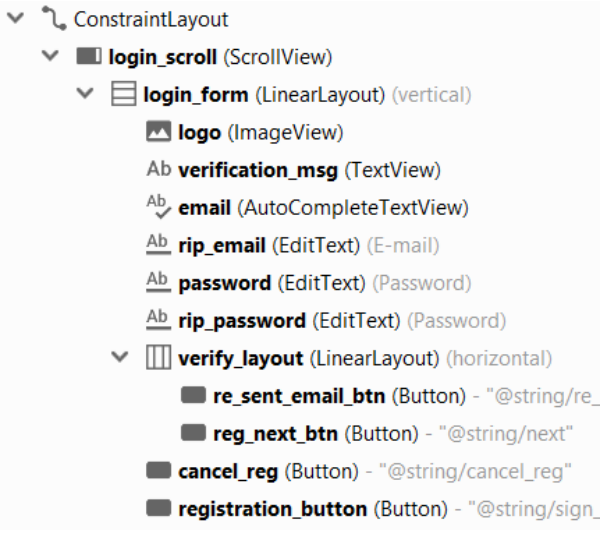


Figura 6.3

Dalla figura 6.3, si può notare come nella *CreateNewAccountActivity* ci siano sia gli elementi relativi alla creazione di un nuovo account sia quelli necessari alla verifica dell'indirizzo email. La visibilità dei bottoni relativi alla verifica dell'indirizzo email verrà gestita programmaticamente.

- **ProfileActivity**

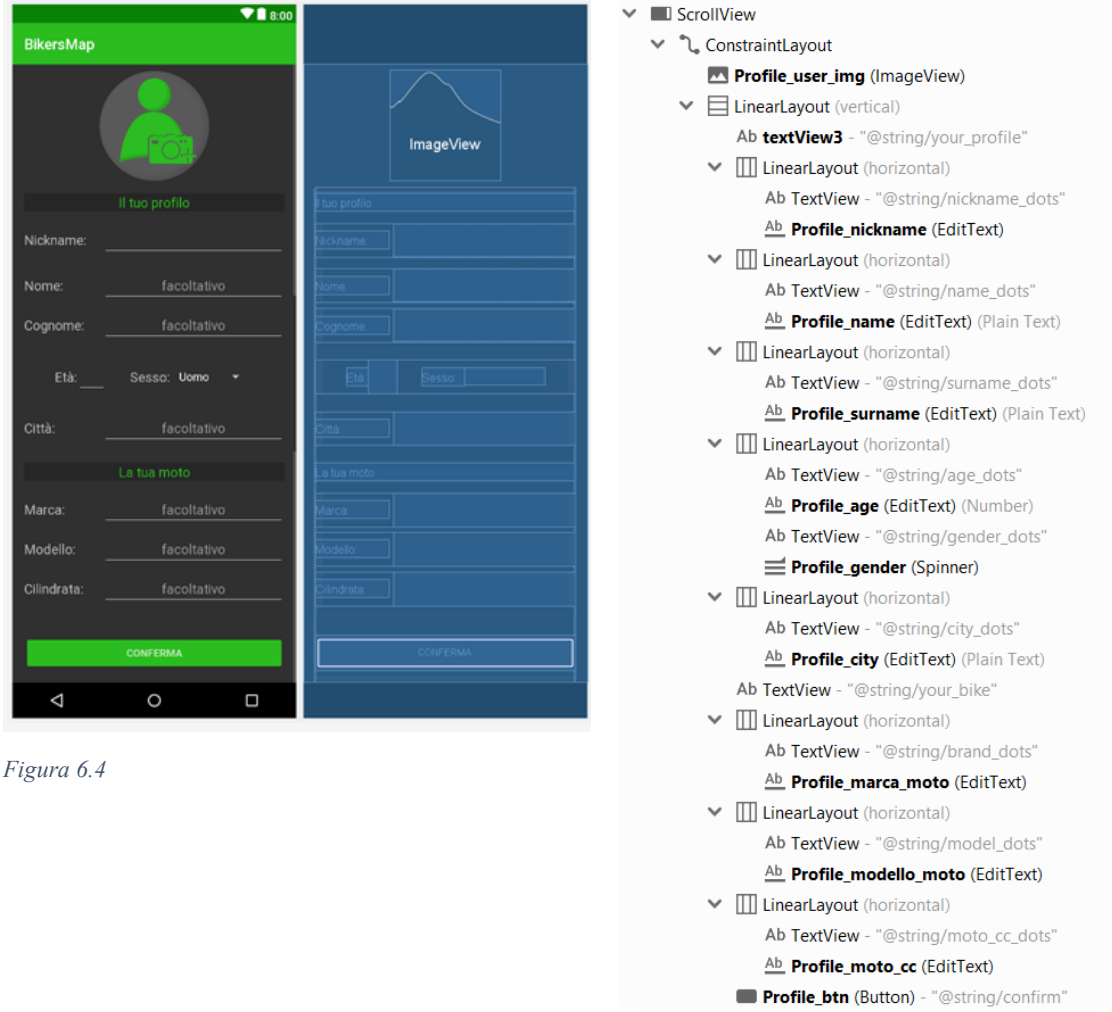


Figura 6.4

- **FriendsActivity**

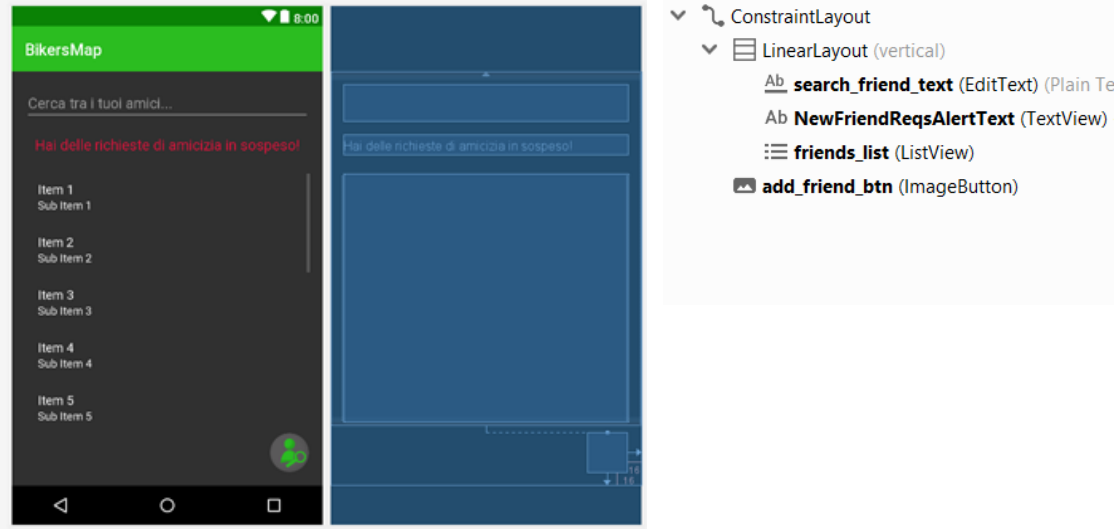
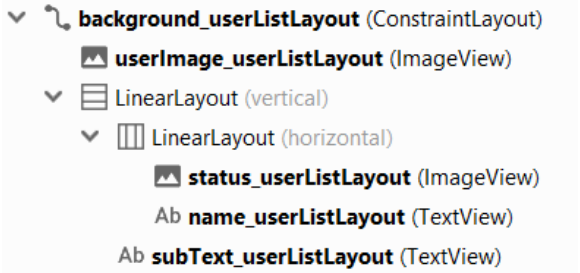


Figura 6.5

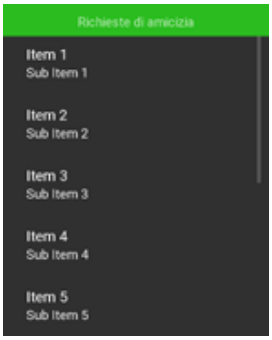
- **FriendListLayout**

Figura 6.6



Questo layout viene utilizzato per definire la struttura degli elementi delle liste presenti nella *FriendsActivity*, nella side bar della *MainActivity* e quella presente nel *SearchUserDialog*.

- **FriendReqsDialog**



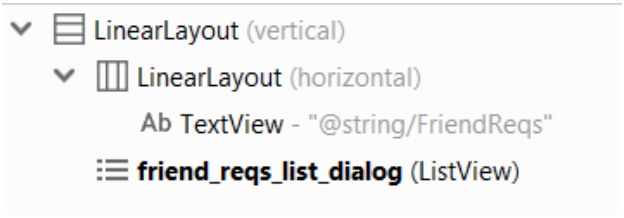
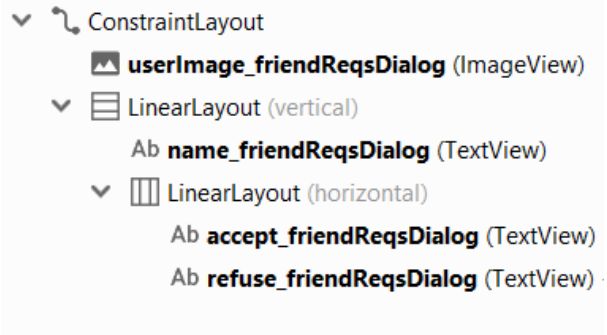


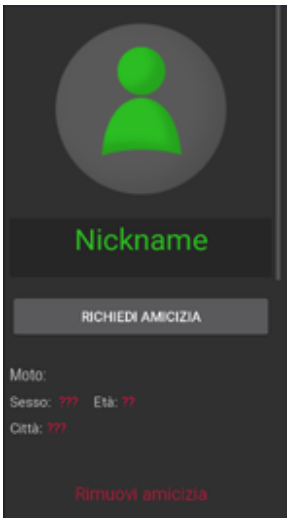
Figura 6.7

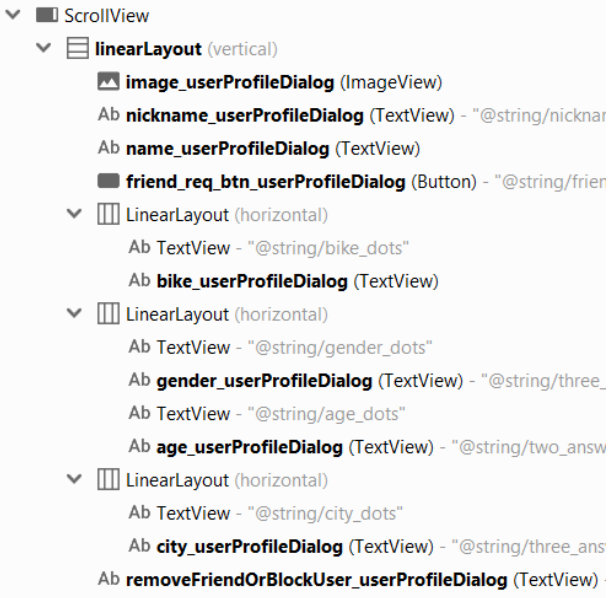
Per gli elementi della lista, questo Dialog utilizza il seguente layout:

Figura 6.8



- **UserProfileDialog**





- **SearchUserDialog**

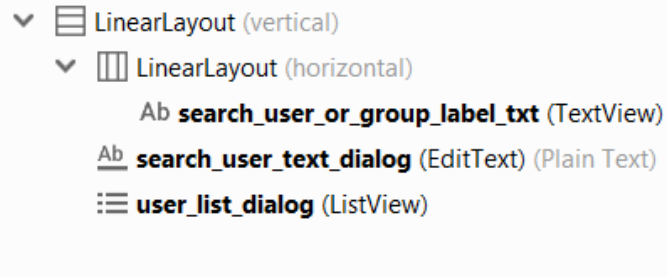


Figura 6.9

- **InfoWindowMapMarker**

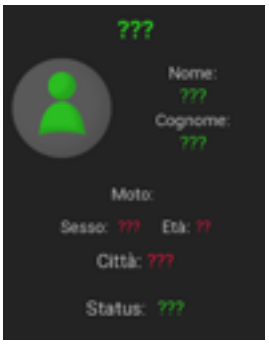
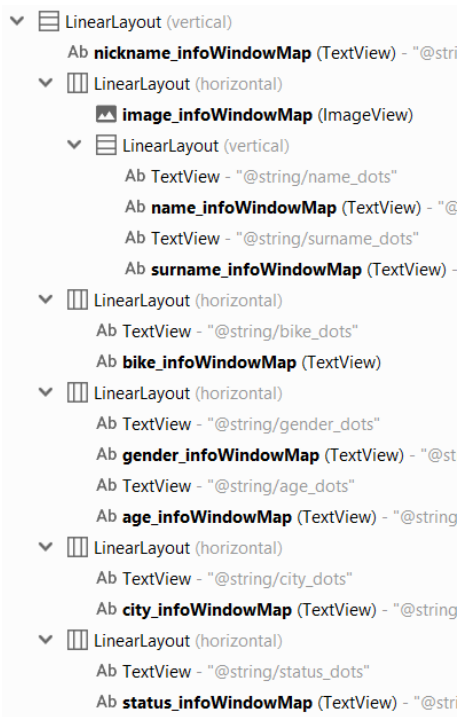


Figura 6.10



Il Layout di figura 6.10, viene utilizzato per la *Custom InfoWindow* relativa ai markers della mappa.

6.5 STRUTTURA DEL DATABASE

Figura 6.11

La figura a lato (figura 6.11), presenta la struttura dei dati immagazzinati sul database messo a disposizione da Firebase.

Ciascun utente ha una propria sezione sotto la voce *"Users"*, identificata da una chiave univoca detta *UID*. Essa consiste in una stringa alfanumerica che viene assegnata da Firebase a ciascun utente, al momento della loro registrazione.

È possibile in questo modo distinguere in maniera univoca qualsiasi utente. Anche due utenti con lo stesso nickname, lo stesso nome e lo stesso cognome.

All'interno del database, vengono registrate tutte le informazioni degli utenti. Nella voce *"friends"* viene salvata la lista degli amici. Quando l'utente riceve poi una richiesta di amicizia, verrà creata una nuova voce detta *"friendreqs"* che conterrà l'*UID* dell'utente che l'ha inviata.

6.6 I METODI E LE CLASSI PRINCIPALI

Si riportano nel seguito alcuni dei metodi e delle classi che costituiscono l'applicativo, che risultano essere più significativi.

6.6.1 Classe UserDB

Essa definisce l'oggetto UserDB, ossia un oggetto che contiene al suo interno tutte le informazioni relative all'utente. Sono inoltre presenti alcune variabili di gestione che non verranno salvate sul database come la variabile booleana *hooked* che sta ad indicare se l'utente è selezionato o meno per la visualizzazione della mappa e l'oggetto *marker*.

```
public class UserDB
{
    private String name, surname, nickname, email, city, moto_brand,
moto_model, moto_cc, UID, status, age, gender, imageURL;
    private Byte[] image;
    private boolean hooked = false;
    private double latitude, longitude;
    private Marker marker;

    private final DatabaseReference UserRef =
FirebaseDatabase.getInstance().getReference("Users");
    private DatabaseReference CurrentUserRef;

    private static List<onChangeListener> listeners = new
ArrayList<>();
...
...
}
```

La classe UserDB, implementa vari costruttori, i metodi *Setter* e i metodi *getter* per la maggior parte delle variabili, e altri metodi come quelli usati per caricare soltanto una determinata informazione sul database, ad esempio:

```
public void updatePosition(double Latitude, double Longitude) {
    CurrentUserRef.child("latitude").setValue(Latitude);
    CurrentUserRef.child("longitude").setValue(Longitude);
    this.latitude = Latitude;
    this.longitude = Longitude;
}
```

Vi è poi un metodo chiamato *checkUserData( )* di cui non si allega il codice per ragioni di spazio, il quale verifica che le informazioni più importanti, salvate sul database, siano valide, altrimenti le corregge.

6.6.2 Classe MapPlot

```
public class MapPlot
{
    private double LatMedia, LngMedia;
    private float zoom = 0;
    private boolean isAtLeastOne;
    private List<UserDB> friendsList;
    private UserDB CurrentUserDB;
    private ToggleButton myPositionBtn;

    public MapPlot()
    {}
...
...
}
```

Ogni istanza della classe MapPlot, contiene al suo interno una lista del tipo *UserDB*, inizializzabile tramite il rispettivo metodo *setter*, con la quale implementa alcuni metodi. Il più importante è il seguente:

```
public void UpdateValues()
{
    double LatMedia = 0, LngMedia = 0, LatMin = 0, LatMax = 0, LngMin
= 0, LngMax = 0;
    int countVisible = 0;
    isAtLeastOne = false;

    if (myPositionBtn.isChecked())
    {
        LatMedia = CurrentUserDB.getLatitude();
        LatMin = CurrentUserDB.getLatitude();
        LatMax = CurrentUserDB.getLatitude();
        LngMedia = CurrentUserDB.getLongitude();
        LngMin = CurrentUserDB.getLongitude();
        LngMax = CurrentUserDB.getLongitude();

        countVisible++;
    }


    for (int j = 0; j < friendsList.size(); j++)
    {
        if (friendsList.get(j).isHooked())
        {
```

```
            if (countVisible == 0)
            {
                LatMedia = friendsList.get(j).getLatitude();
                LatMin = friendsList.get(j).getLatitude();
                LatMax = friendsList.get(j).getLatitude();
                LngMedia = friendsList.get(j).getLongitude();
                LngMin = friendsList.get(j).getLongitude();
                LngMax = friendsList.get(j).getLongitude();
            }
            else
            {
                LatMedia =
(LatMedia+friendsList.get(j).getLatitude())/2;
                LatMin = Math.min(LatMin,
friendsList.get(j).getLatitude());
                LatMax = Math.max(LatMax,
friendsList.get(j).getLatitude());
                LngMedia =
(LngMedia+friendsList.get(j).getLongitude())/2;
                LngMin = Math.min(LngMin,
friendsList.get(j).getLongitude());
                LngMax = Math.max(LngMax,
friendsList.get(j).getLongitude());
            }

            countVisible++;
        }
    }

    this.LatMedia = LatMedia;
    this.LngMedia = LngMedia;

    if (countVisible > 0)
    {
        isAtLeastOne = true;
        double MaxLatDifference = LatMax - LatMin, MaxLngDifference =
LngMax - LngMin;
        double AbsoluteDis = Math.sqrt(MaxLatDifference *
MaxLatDifference + MaxLngDifference * MaxLngDifference);
        float AbsoluteDistance = (float) AbsoluteDis;

        if (AbsoluteDistance == 0)
            this.zoom = 18;
        else {
            this.zoom = (float) 7.5 + (float) 4 * ((float)
Math.log10(1 / AbsoluteDistance));

            if (this.zoom > 18)
                this.zoom = 18;
        }


    }


}
```

Esso utilizza la suddetta lista di elementi del tipo UserDB, la *friendsList*, per calcolarne, soltanto di quelli selezionati, la latitudine media, la longitudine media ed il livello di zoom.

La formula utilizzata per il calcolo dello zoom, è una formula ricavata empiricamente. Il valore così ottenuto, è tale da far mostrare contemporaneamente sulla mappa tutti gli utenti selezionati.

All'interno della classe MapPlot vi sono anche altri metodi, come:

```
public Bitmap CreateMarkerIcon(Context c, String name, int markerID);
```

che crea un marker personalizzato con il nickname dell'utente in formato Bitmap.

E la classe:

```
public class infoWindowAdapter implements GoogleMap.InfoWindowAdapter
```

necessaria per visualizzare la InfoWindow all'interno della mappa.

6.6.3 Classe ListsPopulation

Tale classe, viene utilizzata per il popolamento di tutte le ListView presenti all'interno dell'*App*, per tale scopo, essa possiede al suo interno un *Custom ArrayAdapter*, modificabile a seconda dei casi tramite alcuni metodi *setters*.

```
private class UserListAdapter extends ArrayAdapter<UserDB>
```

I metodi principali contenuti all'interno della classe, sono:

```
public void SearchUsersListPopulation(
        final Context context,
        final List<UserDB> usersList,
        final ListView usersListView,
        final TextView usersSearchText);

public void FriendsListPopulationAndSearch(
        final Context context,
        final List<UserDB> friendsList,
        final ListView friendsListView,
        final TextView friendSearchText);

public void FriendsListPopulationAndMapPlotting(
        final Activity context,
        final ListView friendsListView,
        final List<UserDB> friendsList,
        final GoogleMap googleMap);
```

Il primo popola la ListView identificata come *usersListView* presente nel Dialog *SearchUsersDialog*, prelevando dal database tutte le informazioni degli utenti che soddisfano la ricerca.

Il secondo popola la ListView identificata come *friendsListView* presente nella *FriendsActivity*, andando a prelevare dal database, sotto la voce Users/Utente/Friends, gli *UID* degli utenti che appartengono alla lista degli amici, e poi andando a prelevare in maniera mirata, per ciascuno di essi, le informazioni necessarie.

Il terzo agisce in maniera simile al secondo ma al posto di gestire la ricerca degli utenti, ne gestisce il disegno sulla mappa.

Non si riportano i suddetti metodi anche se significativi, per ragioni di spazio.

6.6.4 Metodo void UpdateUserPosition()

Il presente metodo contenuto nella *MainActivity*, rileva la posizione dell'utente e ad ogni sua variazione, ne aggiorna i valori di latitudine e longitudine sul database.

```
private void UpdateUserPosition()
{
    locationListener = new android.location.LocationListener() {
        @Override
        public void onLocationChanged(Location location)
        {

CurrentUserDB.updatePosition(location.getLatitude(),location.getLongitude());

CurrentUserDB.setPosition(location.getLatitude(),location.getLongitude());
        }

        @Override
        public void onStatusChanged(String provider, int status, Bundle
extras) {
        }

        @Override
        public void onProviderEnabled(String provider) {
        }

        @Override
        public void onProviderDisabled(String provider) {
        }
    };


    locationManager = (LocationManager)
getApplicationContext().getSystemService(Context.LOCATION_SERVICE);
    if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED
&& ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED)
        return;
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
POSITION_UPLOAD_TIME, 10, locationListener);
}
```

6.6.5 Metodo OnMapReady

Esso è il metodo principale della *MainActivity*, contenuto nella classe OnMapReadyCallback. In esso vengono gestite tutte le funzionalità della mappa, come ad esempio gli spostamenti di quest'ultima a seguito del click su determinati elementi dell'interfaccia, come il bottone myPosition:

```
myPositionBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v)
    {
        if(myPositionBtn.isChecked())

myPositionBtn.setBackgroundDrawable(getDrawable(R.drawable.my_location_btn_on)
);
        else

myPositionBtn.setBackgroundDrawable(getDrawable(R.drawable.my_location_btn_off
));

        mapPlot.setCurrentUserDB(CurrentUserDB);
        mapPlot.UpdateValues();
        listsPopulation.setMapPlot(mapPlot);

        if(mapPlot.isAtLeastOne())
        {
            CameraUpdate cameraUpdate = CameraUpdateFactory.newLatLngZoom(new
LatLng(

mapPlot.getLatMedia(),mapPlot.getLngMedia()),mapPlot.getZoom());
            mGoogleMap.animateCamera(cameraUpdate);
        }

    }
});
```

6.7 DESCRIZIONE OPERATIVA

Una volta installata ed aperta l'Applicazione, si presenta la schermata visibile nelle figure da 6.12 a 6.15, dove l'utente può effettuare il *log in* tramite email e password, se già registrato, registrarsi ed effettuare il *log in* tramite *Google* o *Facebook,* o decidere di creare un nuovo account tramite un tap sull'apposito bottone.

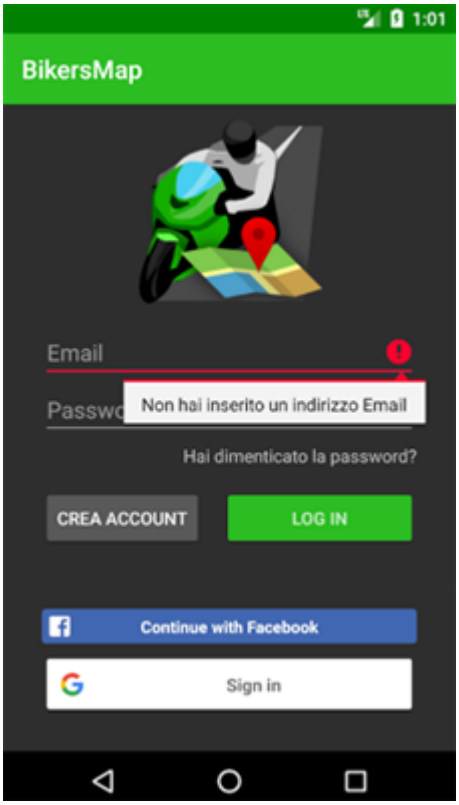


Figura 6.12

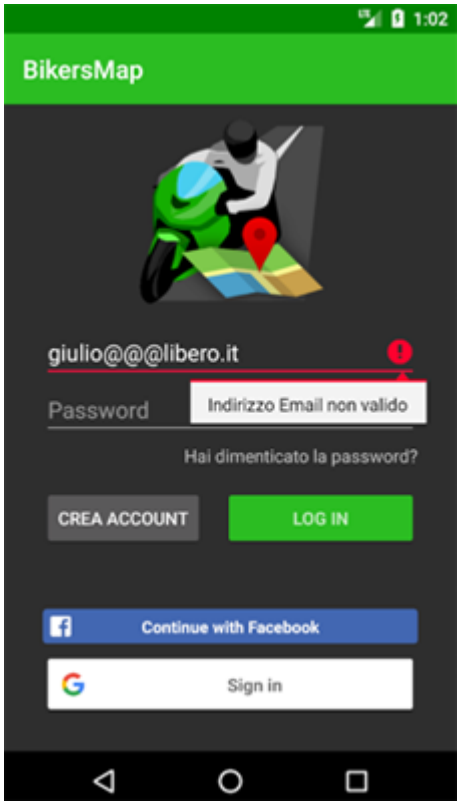


Figura 6.13

Nelle seguenti schermate si notano i controlli di validità, da parte dell'Applicazione dei dati inseriti dall'utente.

Figura 6.14

Figura 6.15

Nelle seguenti figure è mostrata la schermata di creazione di un nuovo account:

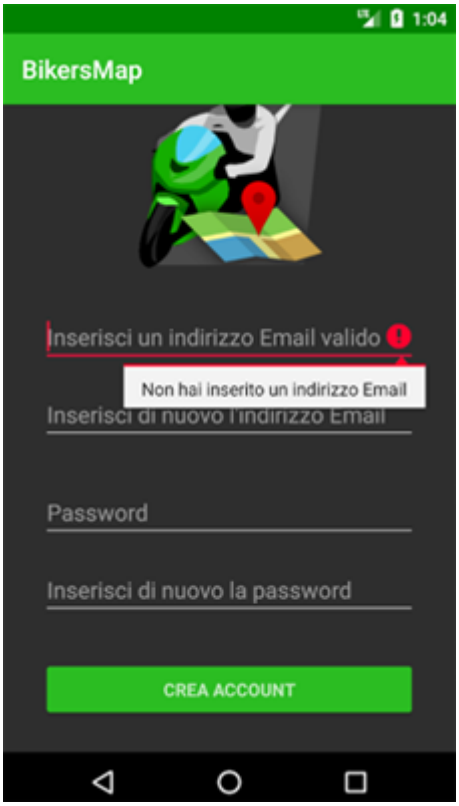


Figura 6.16

Figura 6.17

Se i dati inseriti risultano essere validi, si entra in una schermata dove viene richiesta la verifica dell'indirizzo email.

Figura 6.18

Figura 6.19

Verificato l'indirizzo email, si accede alla schermata relativa alla modifica del profilo:

Figura 6.20

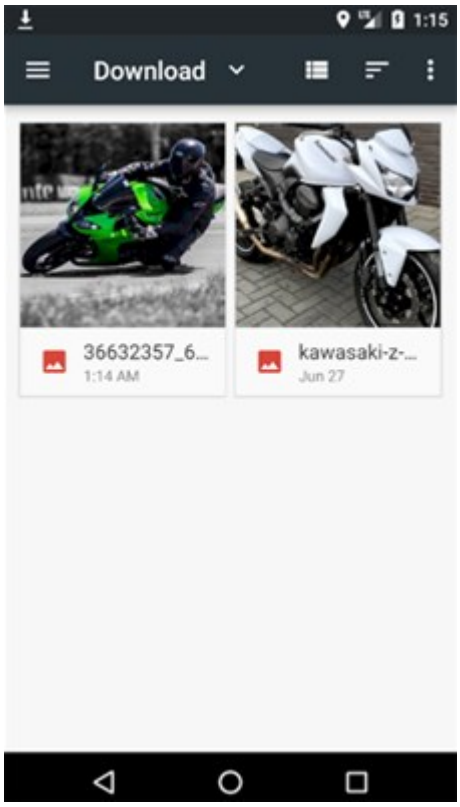


Figura 6.21

Tramite il click sull'immagine in alto, l'utente può impostare una foto del profilo.

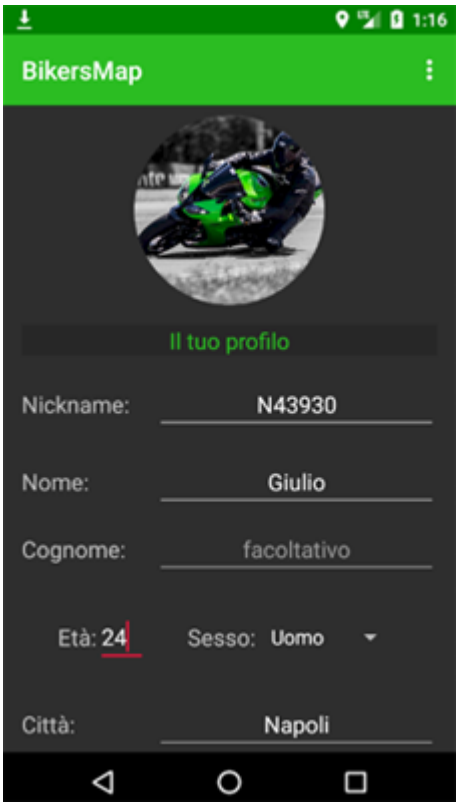


Figura 6.22

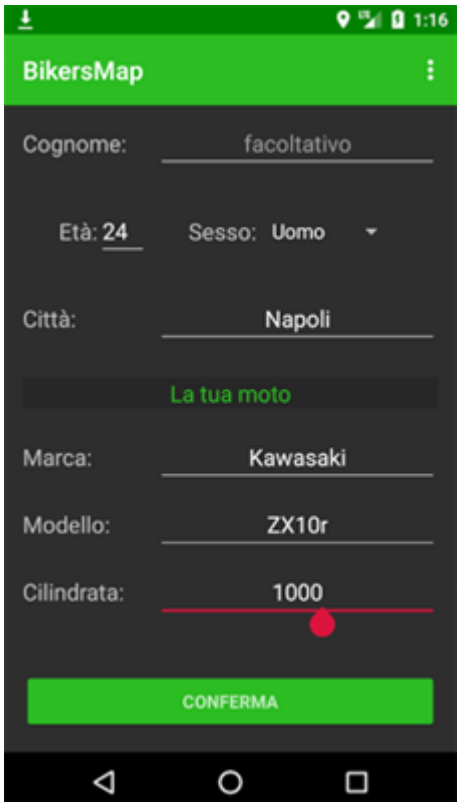


Figura 6.23

Confermati i dati del profilo, l'utente accede alla schermata figura 6.24 e 6.25, dove vi è una mappa:

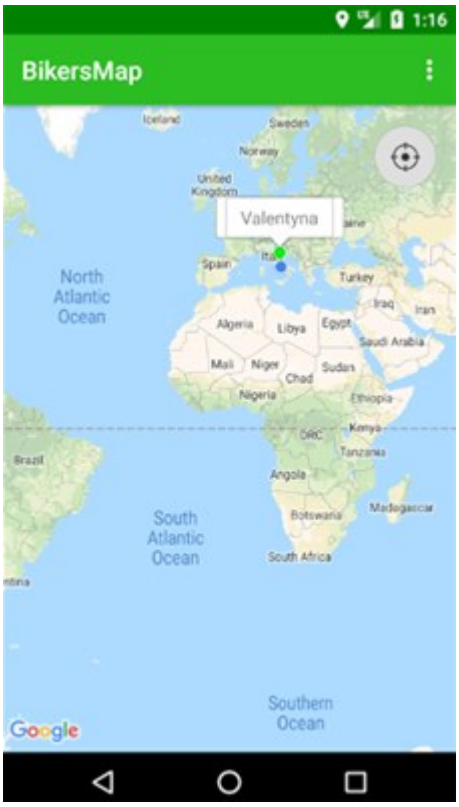


Figura 6.24

Figura 6.25

Facendo uno swipe da sinistra verso destra, si apre il side menù. Effettuando un tap sul nome di un amico, la mappa si sposta sulla sua posizione:

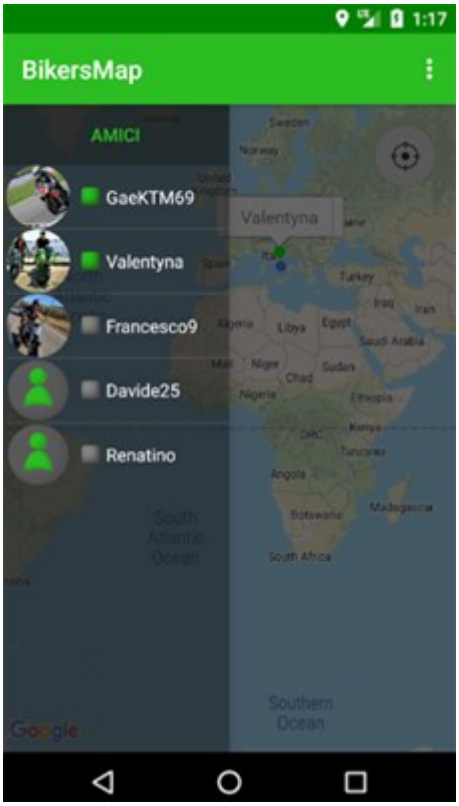


Figura 6.26

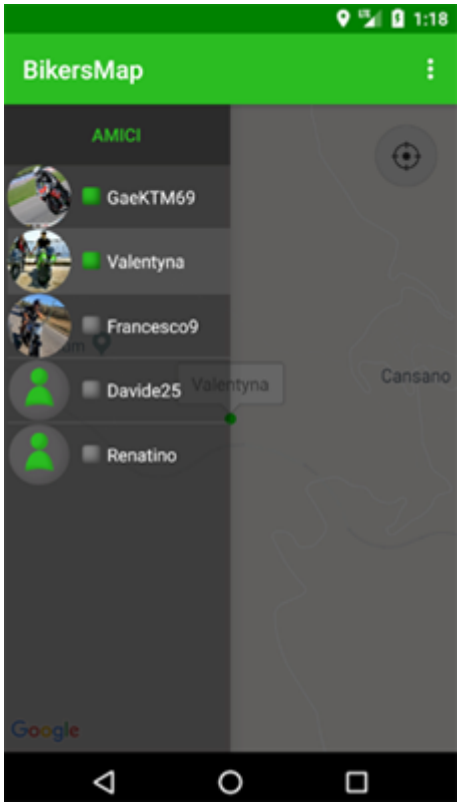


Figura 6.27

Effettuando un tap prolungato su uno o più amici, questi vengono selezionati. La mappa si sposterà quindi in modo da mostrare tutti gli amici selezionati, ogni qual volta la posizione sulla mappa di uno qualsiasi di loro, cambierà.

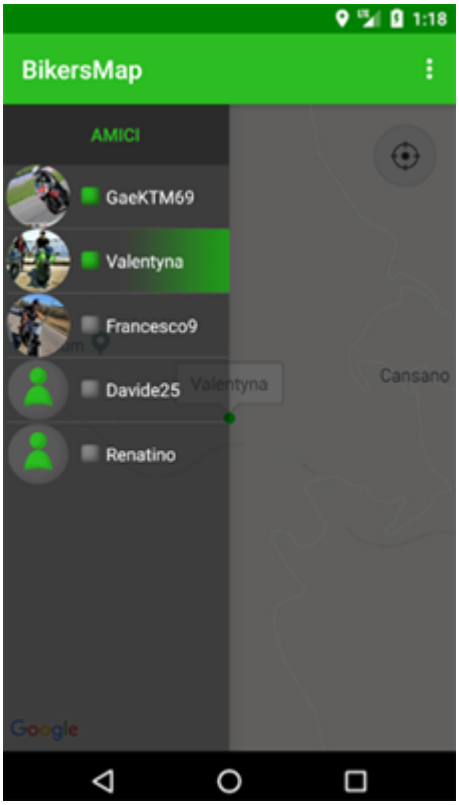


Figura 6.28

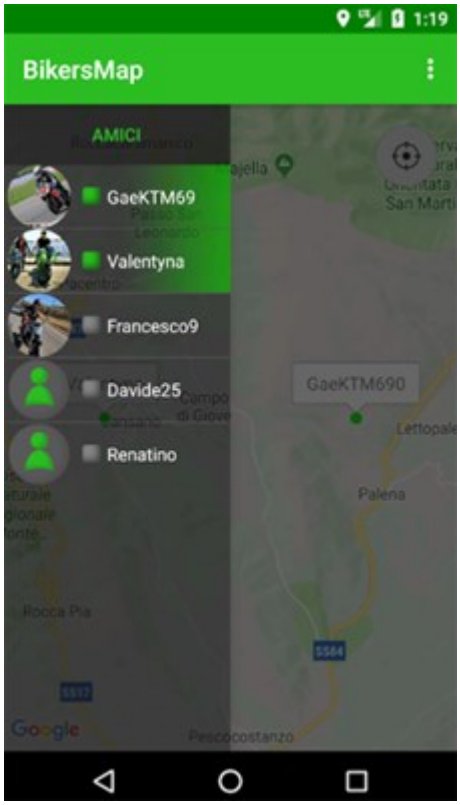


Figura 6.29

Nello stesso modo funziona il tasto in alto a destra che aggancia la posizione dell'utente che sta utilizzando l'Applicazione.

Figura 6.30

Figura 6.31

Quando l’utente effettua il tap su un marker, gli si apre la InfoWindow:

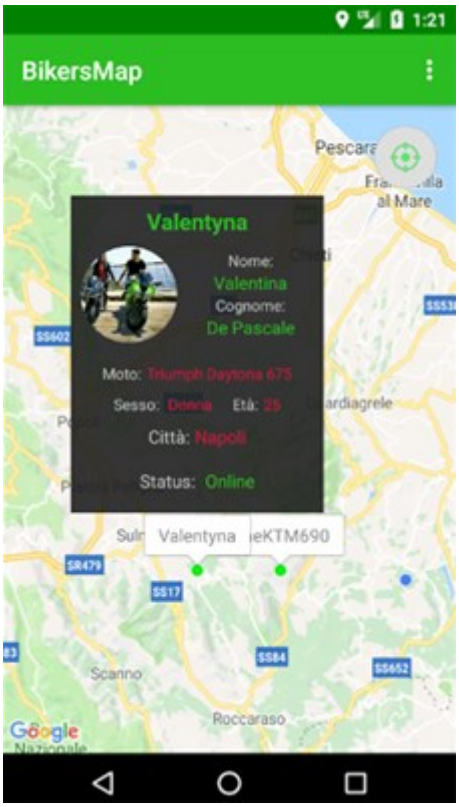


Figura 6.32

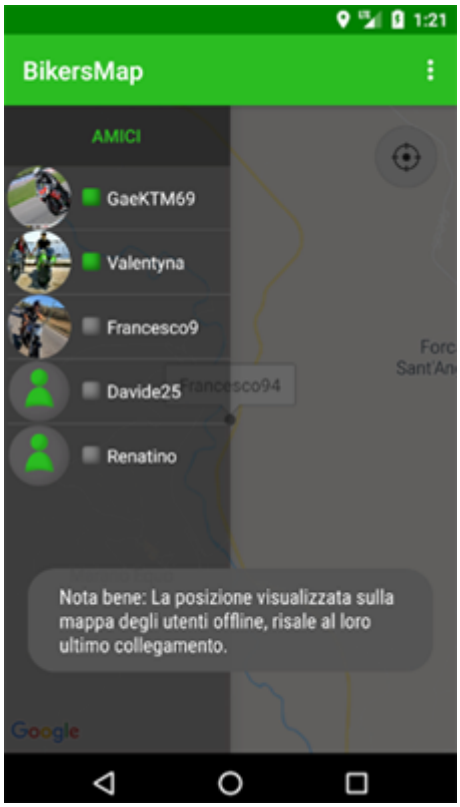


Figura 6.33

Effettuando un click sulla scritta amici situata nell'action bar menù o nel side menù, l'utente viene rimandato nella *FriendsActivity*. Se questi effettua un click sul nome di un amico, gli si apre una Dialog che ne mostra il profilo.

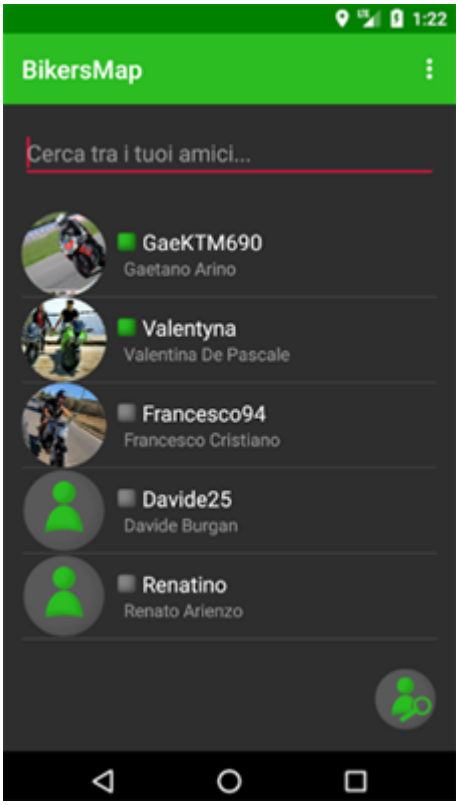


Figura 6.34

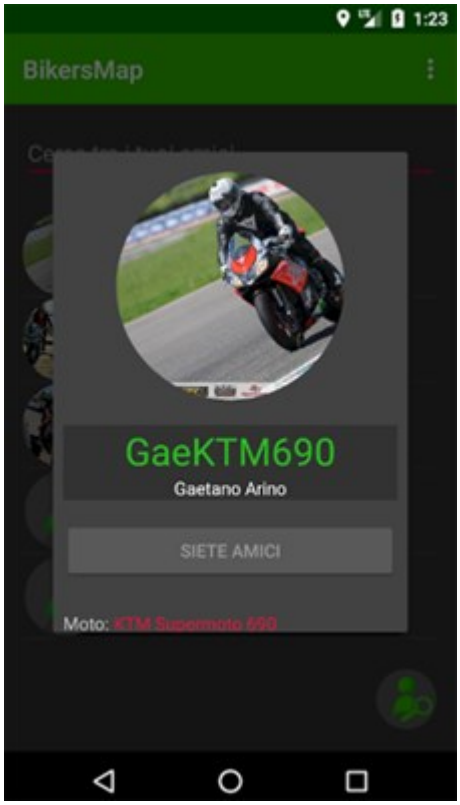


Figura 6.35

Tramite il click sul testo *Rimuovi amicizia*, l'utente può decidere di eliminare dalla lista amici un determinato utente.

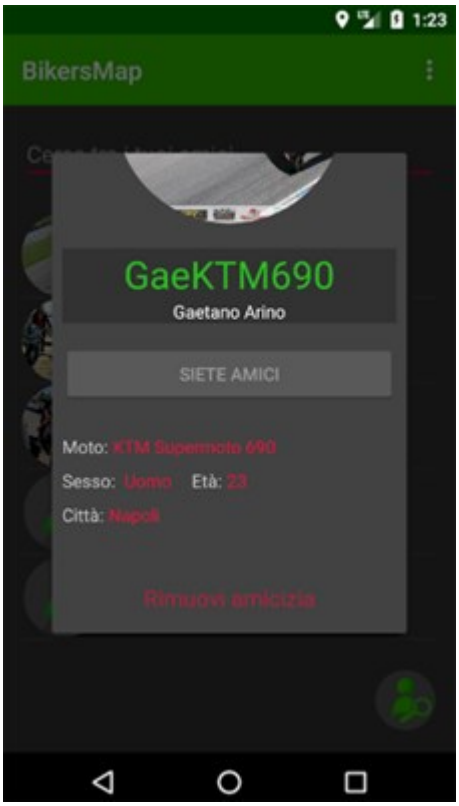
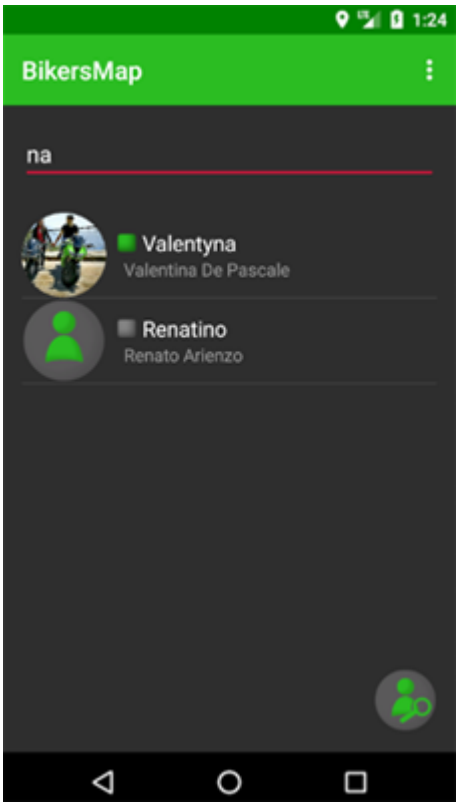
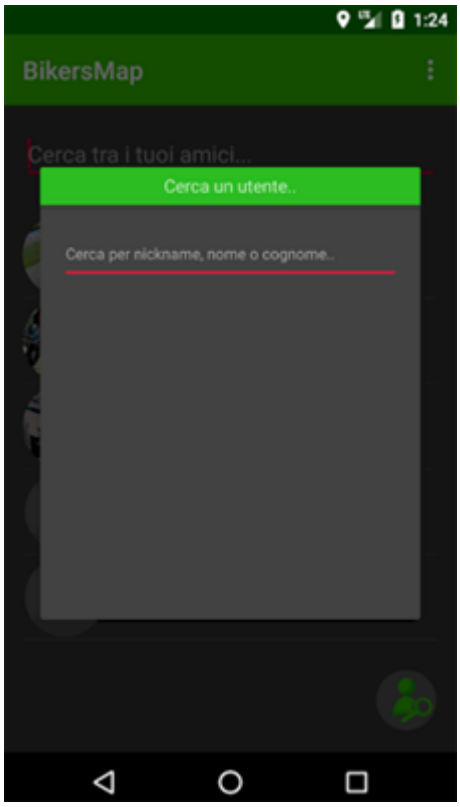


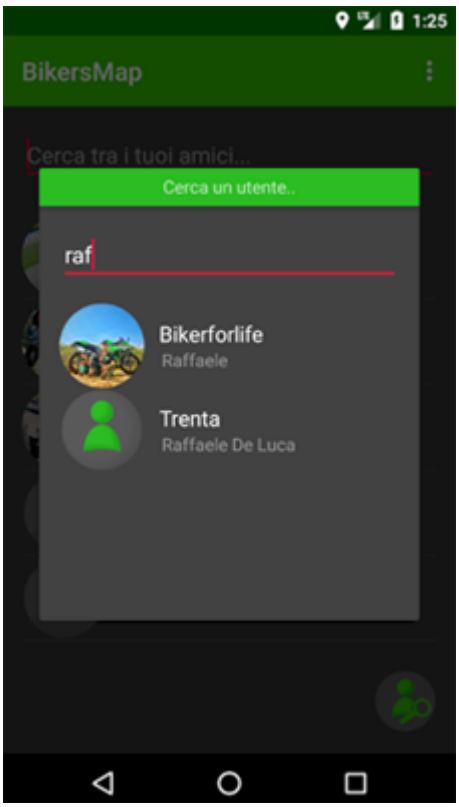
Figura 6.36

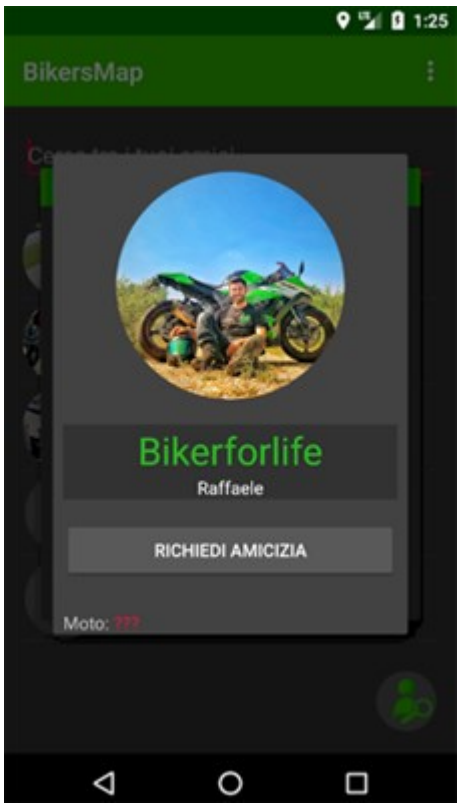
Figura 6.37

La casella di testo in alto serve per cercare tra i propri amici. Il pulsante in basso a destra serve per cercare, tramite una Dialog, altri utenti.









Quando si ricevono delle richieste di amicizia, compare la seguente scritta (in rosso).

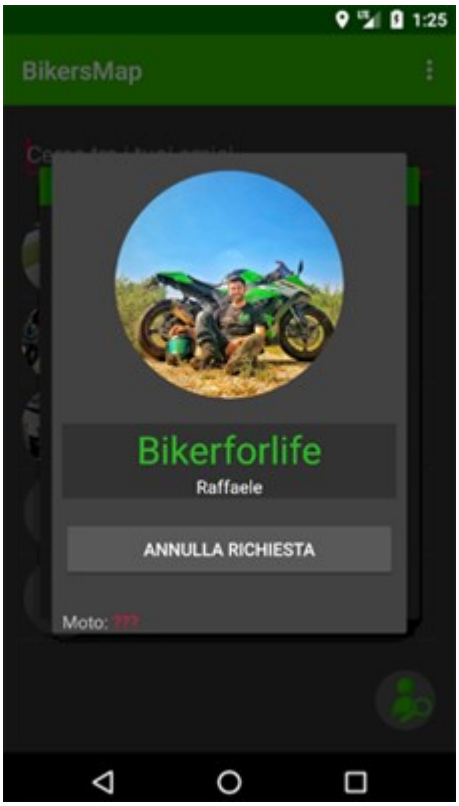


Figura 6.38

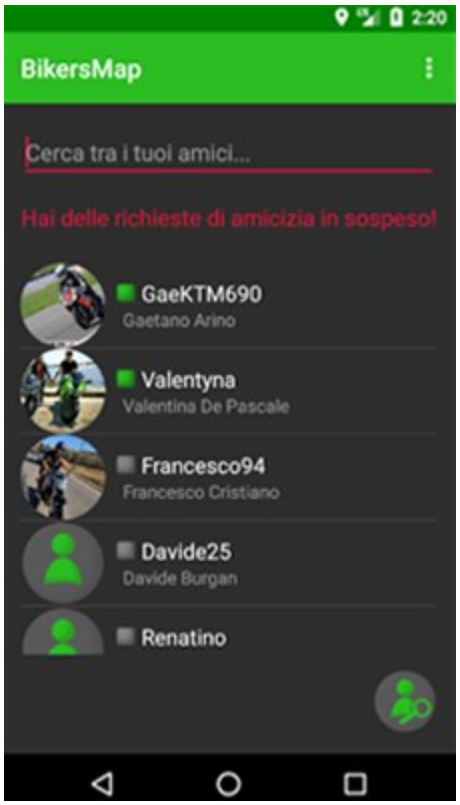


Figura 6.39

Al click, da parte dell'utente, su tale scritta, gli si apre la schermata di figura 6.40 e 6.41, dove può scegliere se accettare o rifiutare la richiesta.

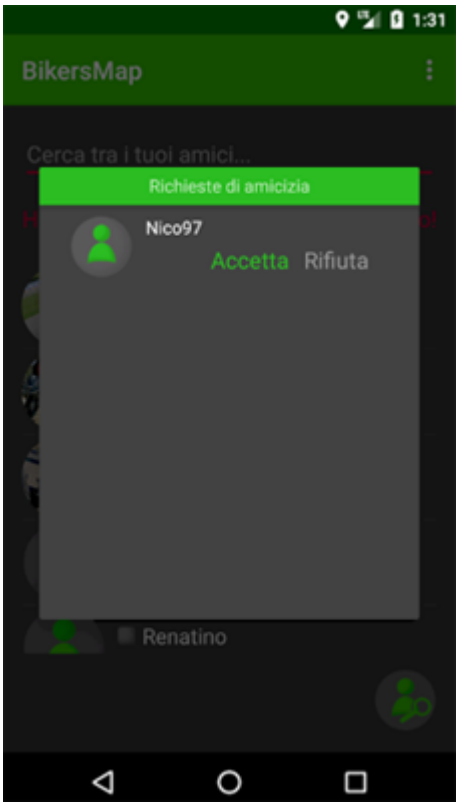


Figura 6.40

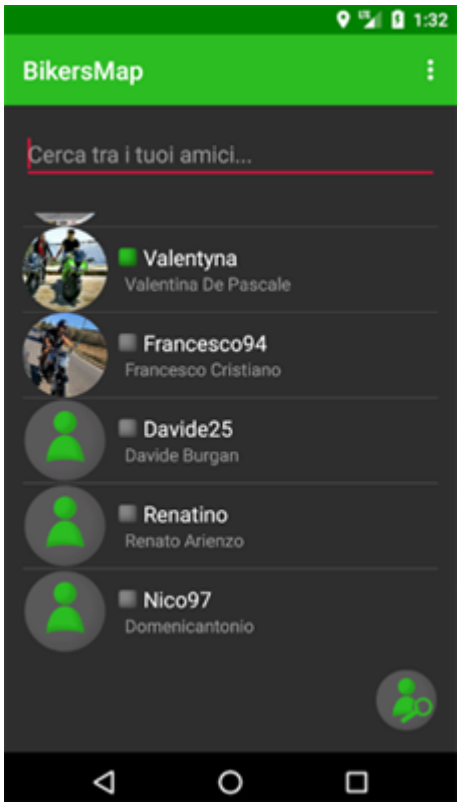


Figura 6.41

7 CONCLUSIONI

La realizzazione dell'applicativo software per Android *BikersMap*, ha soddisfatto tutte le attese, sia sul piano progettuale sia sul piano didattico. Realizzando la suddetta *App*, infatti, l'autore, si è trovato dinanzi alla necessità di dover approfondire gli studi, sia per quanto riguarda alcuni linguaggi di programmazione, come il linguaggio Java OOP e il linguaggio XML, sia per quanto riguarda la struttura del sistema operativo Android, sia per quanto riguarda l'utilizzo dei database.

L'Applicazione *BikersMap*, inoltre, seppur realizzata a scopo didattico, risulta essere funzionale, essendo riuscito l'autore a soddisfare tutti i requisiti richiesti.

Ulteriori sviluppi futuri dell'Applicazione potrebbero essere l'inserimento di alcune funzionalità come l'implementazione di un servizio di navigazione, l'aggiunta di un pulsante per segnalare eventuali emergenze, la creazione di un lato social per l'organizzazione di eventi ed uscite di gruppo ed un servizio di messaggistica istantanea per comunicare con i propri amici.

8 RIFERIMENTI

8.1 BIBLIOGRAFIA

8.2 SITOGRAFIA

http://www.storiainformatica.it/