UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea in Informatica

Progetto e Sviluppo di un'Applicazione Android per Aiutare i Pazienti Diabetici

Singh Manroop

Tesi di Laurea

Relatore:

Prof. Riccardo Martoglia

Anno Accademico 2020/2021

Ringraziamenti

Prima di tutto vorrei ringraziare il Prof. Riccardo Martoglia, per disponibilità e l'aiuto che mi ha fornito nell'arco della mia carriera universitaria. Ringrazio Baljinder, Jaskaran, Diletta, Marco, Filippo, Sukhpreet, Jihed, Federico e tutti gli amici e compagni che mi hanno aiutato, incoraggiato e supportato dall'inizio fino alla fine. Non posso non menzionare i miei genitori e mio fratello Jasroop che da sempre mi sostengono nella realizzazione dei miei progetti. Un ringraziamento finale va al mio amico Gagandeep che mi ha consigliato e aiutato nel mio percorso. Non finirò mai di ringraziarvi tutti per avermi permesso di arrivare fin qui

Grazie.

PAROLE CHIAVE

Diabete

Android

Java

FireBase

Applicazione

Sommario

Parte II

Progetto e Sviluppo

Capitolo 3 - La Progettazione

Capitolo 4 - Implementazione

Capitolo 5- Conclusione

Indice figure

Figura 4.17 : Profilo utente

Introduzione

A differenza di qualche decennio fa i moderni telefoni cellulari sono diventati dei veri e propri Pc con processori potenti e caratteristiche per nulla inferiori a certi calcolatori. Inoltre, sono diventati sempre più facili e accessibili a tutti, tanto che al giorno d'oggi chiunque è in grado di utilizzare un telefono cellulare, dai più piccoli fino alle persone anziane, permettendo loro di accedere alla rete ed entrare in contatto con chiunque e da qualsiasi luogo in modo veloce e semplice. Con l'avanzare della tecnologia abbiamo avuto anche un avanzamento di altre materie che man mano stanno cominciando a fondersi con essa, e una di queste è la medicina. Al giorno d'oggi la tecnologia ricopre un ruolo fondamentale nella medicina partendo dai dispositivi tecnologici presenti in ospedale fino alle applicazioni presenti nei nostri cellulari, che ci aiutano a monitorare la salute e la nostra condizione fisica, ad esempio attraverso il contapassi integrato nei cellulari oppure al monitoraggio della frequenza cardiaca attraverso l'utilizzo di Smartwatch connessi allo Smartphone. L'idea di questa tesi nasce da una persona a me cara e molto vicina, mio padre, mio padre convive da diversi anni col diabete ed è costretto ad assumere un medicinale prima di ogni pasto e a una dieta molto ferrea senza nessun zucchero o alimento che possa aumentare il suo livello glicemico. E convivendo anch'io indirettamente con questa malattia ho deciso di sviluppare un'applicazione che potesse essergli utile e aiutarlo nella sua lotta contro il Diabete. E così è nata SmartHealth, un'applicazione che, come richiama il nome, è Smart e permette all'utente di monitorare la salute. La scelta del nome SmartHealth è data dal fatto che l'applicazione permette ai suoi utenti di monitorare la propria salute in modo smart, permettendo agli utenti di qualsiasi età di utilizzare l'app in modo semplice e facile grazie all'interfaccia user-friendly. Il lavoro di tesi è organizzato in cinque capitoli che trattano le fasi di studio delle tecnologie, l'analisi dei requisiti del sistema, la progettazione, l'implementazione dell'applicazione e la conclusione. Il Primo capitolo presenta una panoramica sul Software SmartHealth, la sua struttura generale viene brevemente

introdotto il Diabete. Il Secondo capitolo tratta le tecnologie utilizzate e i motivi delle varie scelte. Il Terzo capitolo analizza le fasi di progettazione del software approfondendo l'analisi dei casi d'uso e i vari diagrammi. Il Quarto capitolo analizza le fasi d'implementazione del software e descrivendo la realizzazione effettiva del progetto studiandone i vari elementi che lo compongono. Infine, il Quinto capitolo conclude la tesi con le possibili implementazioni future che l'applicazione potrebbe avere.

Parte I

Il caso di studio

Capitolo 1

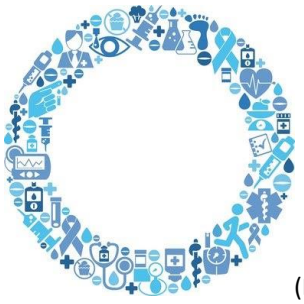
Analisi degli obiettivi

1.1 Scopo Del Progetto

Lo scopo del progetto era quello di realizzare un'applicazione per dispositivi cellulari che potesse aiutare i pazienti Diabetici a ricordarsi di assumere la loro medicina e/o insulina a una certa ora per mantenere il diabete sotto controllo, essendo un software rivolto a persone diabetiche c'era da tener conto anche dell'età media del pubblico al quale era rivolta questa applicazione. Infatti, secondo i dati statistici dell'Istat si è notato che la possibilità che una persona diventi diabetica aumenta con l'aumentare della sua età, e per questo motivo l'applicazione doveva essere sviluppata con un'interfaccia grafica che fosse intuibile e facile da utilizzare, così da permettere il suo facile utilizzo anche ai pazienti più anziani o ai pazienti con poche conoscenze tecnologiche. Per questo il mio obiettivo principale è stato quello di realizzare una app con interfaccia user-friendly. Il funzionamento di SmarthHealth è abbastanza facile e intuitivo, infatti, l'utente potrà scegliere di ricevere una notifica ogni giorno un una certa ora e presa la medicina dovrà poi confermare di aver preso la medicina, inviando così una conferma al medico di base che potrà tener traccia dei suoi pazienti direttamente dall'applicazione e controllare chi sta prendendo la medicina regolarmente e chi no. Oltre al ricordare al paziente di prendere la medicina volevo che l'applicazione potesse essere utile anche in altri modi per l'utente e per questo motivo ho pensato d'implementare ulteriori funzionalità che permettessero di tener traccia dei passi effettuati in un giorno, di poter ricevere consigli su salute e alimentazione inoltre, l'applicazione doveva avere anche una sezione che permettesse all'utente non solo di vedere i propri dati personali ma anche le informazioni di contatto del proprio medico di base il tutto in un'unica applicazione.

1.2 Il Diabete

Prima d'iniziare a parlarvi dello sviluppo del software vorrei soffermarmi a fare una breve introduzione sul diabete. Il diabete è una malattia cronica caratterizzata dall'eccessivo livello di Glucosio (zucchero) nel sangue del paziente. Questa particolare condizione è causata da una disfunzionalità da parte del Pancreas nel creare l'insulina, un ormone con il compito di tener sotto controllo il livello glicemico nell'organismo, in altre parole, se manca l'insulina oppure quella prodotta non funziona a dovere, i livelli di zuccheri nel sangue aumentano fino ad arrivare a una soglia critica, che potrebbe avere conseguenze anche gravi. Spesso quando si parla di diabete si lascia intendere che si tratti di una sola e unica malattia quando invece il diabete si distingue in varie tipologie. E le più note e diffuse sono: il diabete di tipo uno e il diabete di tipo due.



(Figura 1. Simbolo Universale del Diabete.)

1.2.1 Il Diabete di tipo uno

Il primo tipo di diabete è il diabete di tipo uno ed è una malattia autoimmune, viene chiamato anche diabete giovanile per la sua propensione a svilupparsi durante gli anni giovanili di una persona. Il sistema immunitario identifica le cellule del pancreas responsabili della produzione dell'insulina come cellule dannose e si attiva per distruggerle. I motivi per cui ciò accade non sono ancora completamente chiari, ma si pensa che ci siano diverse cause, come la predisposizione genetica e la reazione

spropositata a un'infezione virale. Questo tipo di diabete rappresenta circa il 5% dei casi nel nostro paese ed è irreversibile. Ciò significa che attualmente non è ancora stata individuata una cura. E i pazienti di tale diabete devono rispettare alcuni accorgimenti che aiutano a tenere sotto controllo i livelli glicemici, come ad esempio una dieta equilibrata e una frequente attività fisica. Non essendoci una cura ed essendo una malattia irreversibile il paziente sarà costretto ad assumere l'insulina per il resto della sua vita.

1.2.2 Il diabete di tipo due

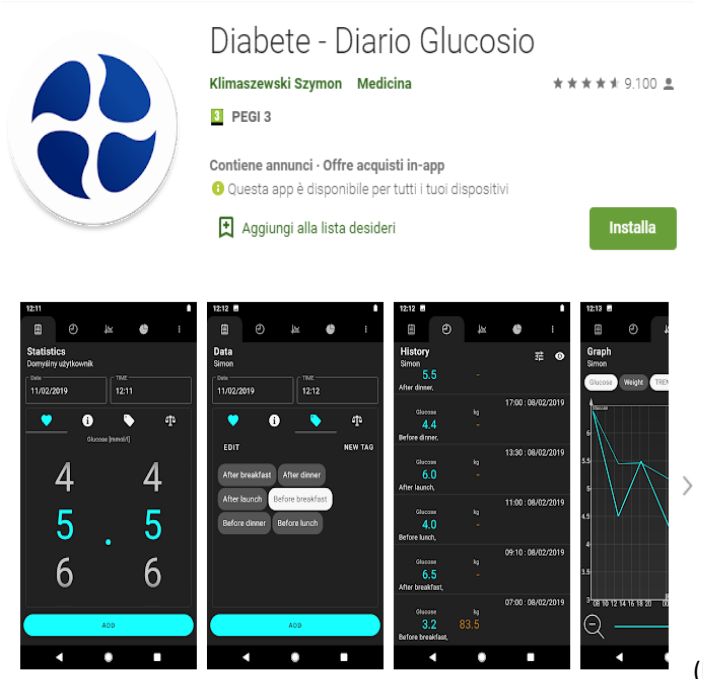
Il diabete di tipo due rappresenta il 90% dei casi in Italia e si verifica quando il pancreas del paziente non riesce a produrre abbastanza insulina oppure l'organismo non riesce a utilizzarla in modo ottimale, causando così un eccesso di glucosio nel sangue. Essendo presente nel 90% dei casi di diabete è la forma più diffusa. Le cause di questo tipo di diabete possono essere multiple e possono essere legate a fattori genetici, ambientali o nella maggior parte dei casi causati da stili di vita scorretti. Ad esempio, l'obesità, la sedentarietà (ovvero uno stile di vita senza esercizio fisico di alcun tipo) o il consumo eccessivo di grassi saturi (formaggi, burro, insaccati, carni grasse) sono fattori di alto rischio per l'insorgere del diabete di tipo due. A differenza dei pazienti di diabete di tipo uno, i pazienti del diabete di tipo due non sono insulinodipendenti, ovvero che non hanno bisogno di assumere insulina da una fonte esterna. La terapia per il diabete di tipo due consiste, in genere, nell'adozione di una sana alimentazione e di un esercizio fisico costante. Se questi accorgimenti non sono sufficienti a controllare i valori glicemici, è necessario assumere dei farmaci orali. E solo nel caso in cui i farmaci non riescano a ottenere un controllo glicemico c'è bisogno di somministrare l'insulina da fonti esterne.

1.3 Ricerca di Software Simili

Prima di cominciare a sviluppare l'applicazione ho voluto fare una ricerca sul Play Store per vedere che tipo di applicazioni sono disponibili per le persone diabetiche, e le applicazioni che ho trovato erano:

- mySugr - App Per diabete e controllo glicemico, un'applicazione sviluppata da mysugr Gmbh che permette di tener traccia del livello glicemico dell'utente e dà la possibilità di collegare un Glucometro.
- Diabete - Diario Glucosio, è un'applicazione sviluppata da Klimaszewski Szymon, permette all'utente di registrare il proprio livello del glucosio nel sangue.
- Diabete Tracker gratuito, sviluppata da Pragma Infotech, permette di tener traccia dei livelli di zuccheri nel sangue.

Quasi tutte le applicazioni presenti sullo store permettono di fare più o meno le stesse cose, ovvero permettono di tenere traccia del livello glicemico dell'utente, o che funzionano con l'utilizzo di un Glucometro o di un Microinfusore d'Insulina, e nessuna di queste applicazioni presenti permette all'utente d'impostare una notifica che gli ricordi di assumere per tempo la medicina o di avere consigli su come mantenere il diabete sotto controllo e né tanto meno di poter segnalare al proprio medico di base di aver assunto la dose di medicina giornaliera in un'unica applicazione. In questo modo ho avuto la possibilità di sviluppare un software che fosse completamente diverso da quelli che si trovano sullo Store, creando così un'applicazione unica nel suo settore.



(Fig. 1.2 Una delle Applicazioni disponibili nello store)

Capitolo 2

Tecnologie utilizzate

2.1 Android

Android è un sistema operativo per smartphone e tablet sviluppato dalla Android.inc. Il sistema operativo Android si basa sul kernel Linux e venne utilizzato per la prima volta nello smartphone T-Mobile G1 della HTC con la prima versione, Android 1.0. Negli anni successivi il sistema Android registrò una progressiva diffusione negli smartphone diventando così una soluzione alternativa ai sistemi operativi della Microsoft e della Apple. Attualmente Android è il principale sistema operativo per gli smartphone, i tablet e anche i computer in tutto il mondo. In un'intervista del 2010 l'Ad di Google, Schmidt, aveva dichiarato che Android era installato su uno smartphone su tre negli Stati Uniti. E dal 2010 ad oggi 2021 l'utilizzo di Android è cresciuto enormemente tanto che oggi tre smartphone su quattro hanno Android come sistema operativo. La crescita di Android non riguarda soltanto l'utente finale ma anche gli sviluppatori, dovuta grazie all'architettura del sistema e dalle salde radici nel mondo open source, un altro dei motivi che spinge gli sviluppatori a scegliere Android rispetto ad altri sistemi è la presenza d'innumerevoli strumenti gratuiti di default e dell'enorme quantità di materiale, tutorial, video lezioni e guide presenti online, e quest'ultime sono le principali ragioni per le quali ho scelto di sviluppare SmartHealth in Android

2.2 Android Studio

Android Studio è un ambiente di sviluppo basato su IntelliJ IDEA e serve per sviluppare facilmente applicazioni Android, è liberamente disponibile per Windows, MacOS e Linux. Android Studio è un editor intelligente che ci offre: completamento avanzato del codice,

refactoring, e analisi. E queste importanti caratteristiche rendono lo sviluppo di app più veloci e produttive. Essendo sviluppato specificamente per la creazione di applicazioni Android, questo ha permesso ad Android Studio di diventare l'IDE primario di Google per lo sviluppo di app native, sostituendo così gli Android Development Tools (ADT) di Eclipse. Una delle sue migliori caratteristiche è quella della portabilità, ovvero essendo disponibile anche su altri sistemi operativi, quali Linux e MacOS, questo permette agli sviluppatori di poter spostare il loro progetto da un sistema all'altro senza avere problemi di compatibilità varie. Un'altra caratteristica degna di nota è il fatto che Android Studio è sempre alla pari con lo sviluppo e l'innovazione globale; infatti, viene costantemente aggiornato con l'aggiunta di nuove features e la compatibilità con le ultime versioni di Android.

2.3 Come si sviluppa una app?

Il primo passo per poter iniziare lo sviluppo di applicazioni per Android consiste nel dotarsi degli strumenti adeguati e necessari. Il cuore di questo tipo di programmazione risiede in un ambiente a noi noto come Android SDK (software development kit) ovvero un insieme di strumenti per lo sviluppo e la documentazione di software. Per nostra fortuna, non abbiamo più bisogno d'interfacciarci a esso direttamente ma è stato creato un ambiente di lavoro apposito che è efficiente, intuitivo e multipiattaforma di cui abbiamo già parlato ovvero Android Studio. Android studio ci permette di utilizzare tutte le funzionalità dell'SDK senza alcuna difficoltà, oltre a lavorare al codice, creare interfacce utente e testare le applicazioni. Android Studio è ottenibile gratuitamente dal sito ufficiale di Android che tra l'altro fornisce agli sviluppatori una vasta gamma di guide e tutorial. La realizzazione di un'applicazione Android è composta da due parti:

• La scrittura del codice dinamico, che può essere scritto in Kotlin o Java, che si prende cura della gestione degli eventi. La mia scelta del linguaggio per SmartHealth è ricaduta su Java essendo un linguaggio a me già noto grazie alle lezioni di Programmazione a Oggetti.

• La realizzazione della parte statica, scritta in XML, che definisce le caratteristiche fisse ovvero che non cambiano durante l'esecuzione dell'applicazione, come la disposizione del testo, icone o eventuali titoli che devono restare sempre presenti.

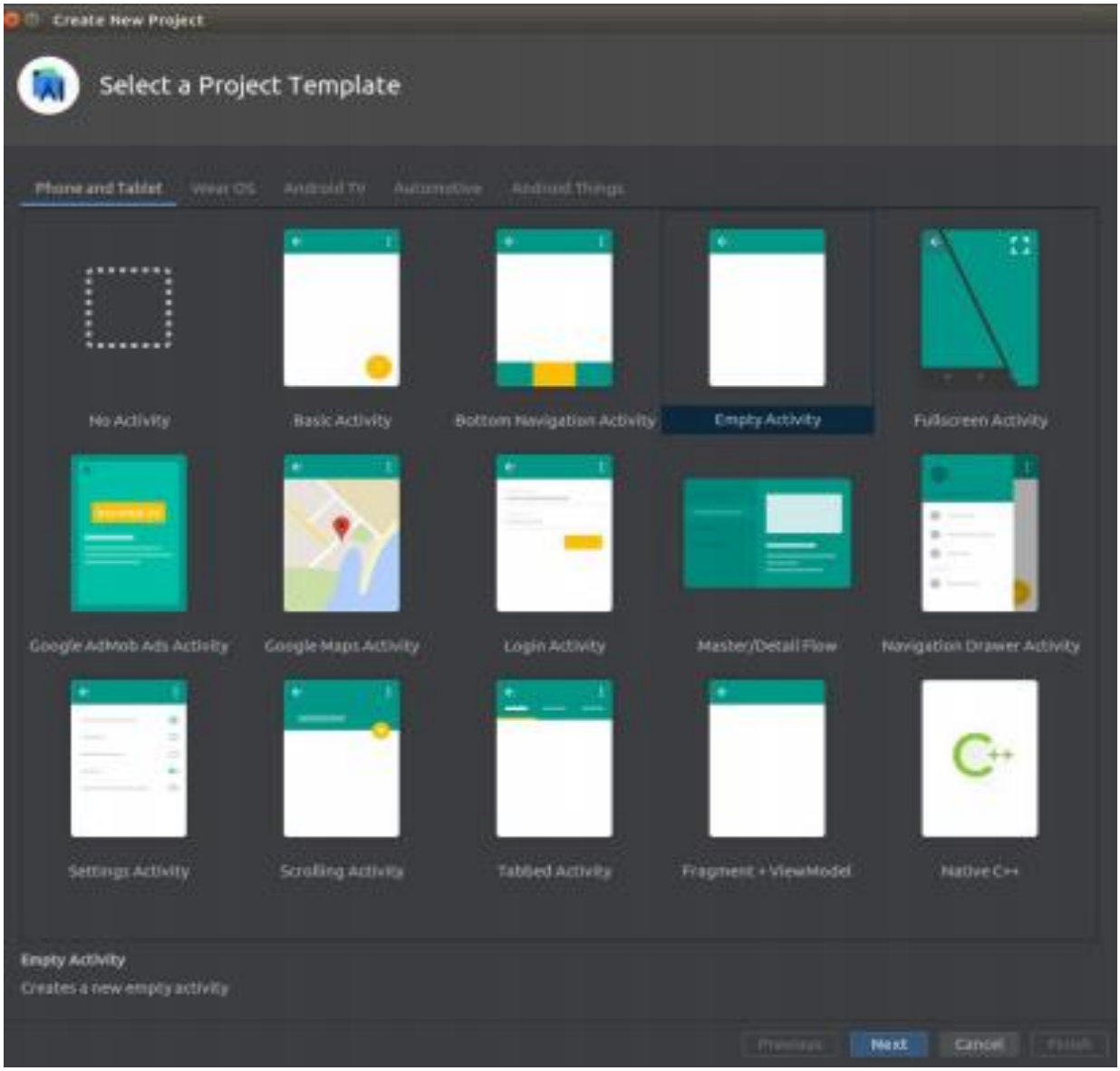
2.4- Android SDK

Android SDK (Software Development Kit) è un pacchetto di sviluppo per applicazioni e per software. Attraverso il software development kit qualsiasi utente, anche il più inesperto, potrà ottenere il codice sorgente di qualsiasi versione di Android e potrà migliorarlo o modificarlo autonomamente. I linguaggi di programmazione alla base delle applicazioni Android sono Java e C/C++. Con l'SDK, scaricato dal sito ufficiale, sarà possibile disporre sul proprio pc un'interfaccia adibita alla modifica e al lancio di applicazioni Android. Inoltre, l'SDK include una serie di strumenti di sviluppo come ad esempio il debugger, un emulatore per le nostre applicazioni basato su QEMU, librerie, documentazioni, codici esemplificativi, tutorial e varie librerie. Per utilizzare conformemente al proprio lavoro gli strumenti e le piattaforme rese disponibili dall'SDK dovremmo accedervi tramite il menu Tools presente su Android studio.

2.5- Iniziare a sviluppare su Android Studio

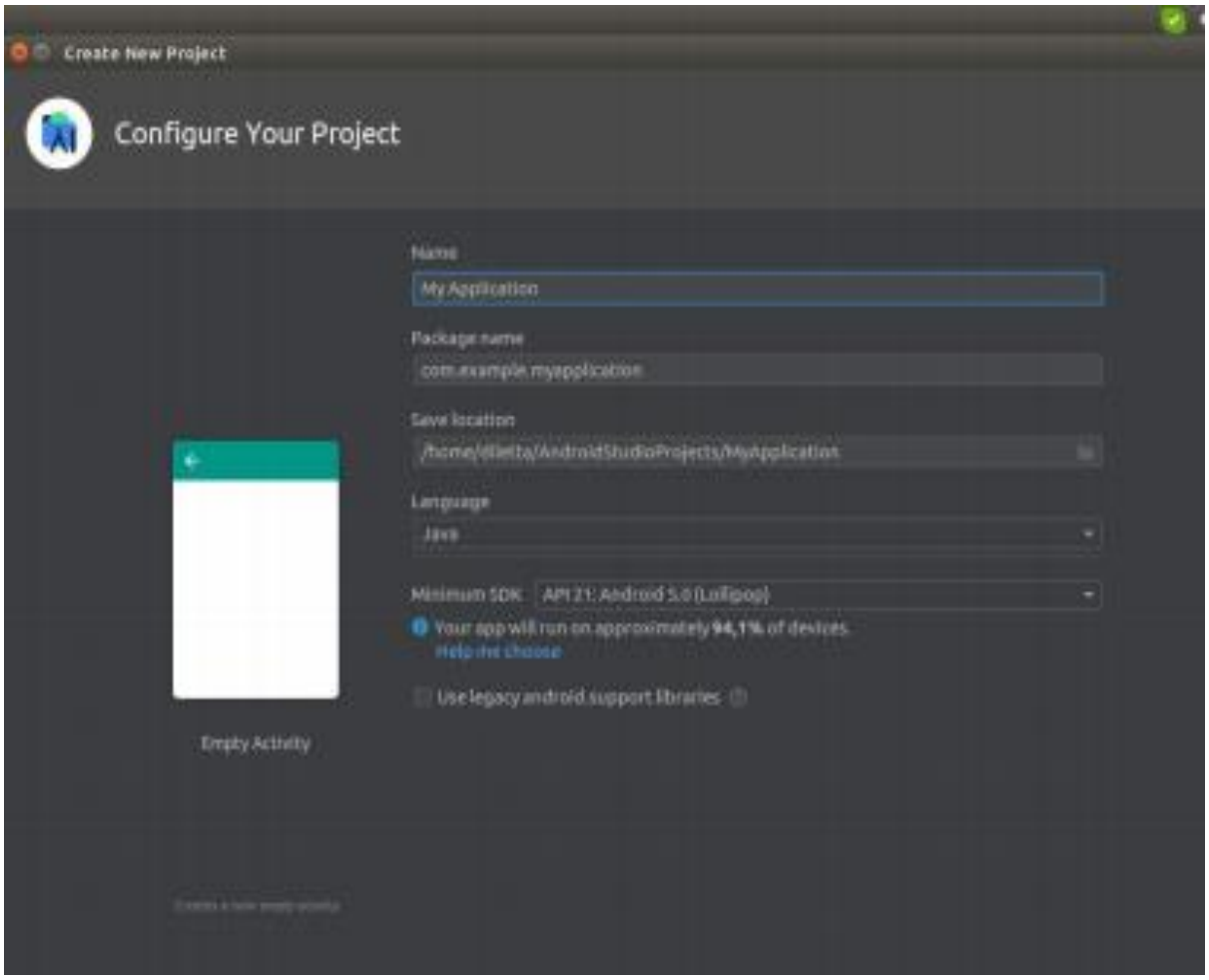
Quando iniziamo a sviluppare un'applicazione Android Studio ci guiderà nei passaggi necessari alla realizzazione della nostra applicazione. Il primo passaggio consiste nello

scegliere la tipologia di dispositivo per il quale verrà sviluppata la nostra applicazione, tra le varie scelte possiamo trovare Phone, Tablet, Wear OS, Tv, etc… Una volta che abbiamo scelto il dispositivo più adatto alle nostre esigenze il passaggio successivo consiste nello scegliere un template di activity tra quelli già esistenti oppure possiamo anche cominciare lo sviluppo senza scegliere un template preesistente scegliendone semplicemente uno vuoto.



(Fig.2.1 Finestra per la scelta del template)

Il terzo passaggio consiste nello scegliere il Package name, ovvero il nome che viene utilizzato per identificare la nostra applicazione in modo univoco, la location nella quale verrà salvato il nostro progetto, il linguaggio da utilizzare, Kotlin o Java. Come già anticipato prima la mia scelta è ricaduta sul linguaggio Java, ma non soltanto perché era un linguaggio a me familiare ma anche perché essendo più utilizzato rispetto a Kotlin sapevo che avrei trovato molte più guide e video tutorial per la risoluzione di bug e problemi di codice.

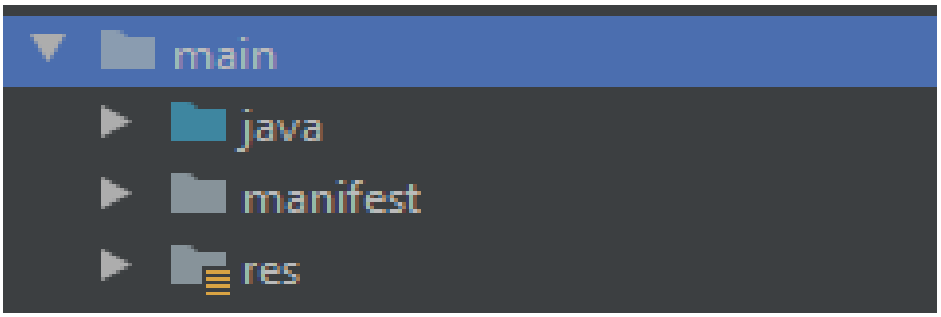


(Fig.2.2 Finestra per la scelta del nome, location e del linguaggio)

2.6- Com'è strutturato SmartHealth su Android Studio

Il nostro progetto sarà formato da tre cartelle principali

- Java, ovvero la cartella contenente il codice Java.
- Manifest, ovvero la cartella contenente il file AndroidManifest.xml
- e infine la cartella Res che contiene i dati esterni all'applicazione.



(Fig. 2.3 Struttura principale del nostro progetto Android)

2.7 Fondamenta di Android

Ogni applicazione sviluppata in Android, a prescindere dalla sua finalità, affida le proprie funzionalità a quattro elementi principali, che sono:

- Activity
- Service
- Content Provider
- Broadcast Receiver
- e Intent

essi permettono all'applicazione di integrarsi alla perfezione nell'ecosistema Android.

2.7.1 Interfaccia grafica (Activity)

L'interfaccia grafica di ogni applicazione viene gestita da un elemento fondamentale di Android ovvero l'Activity, (è doveroso informare che ad una schermata corrisponde una ed una sola Activity). Le applicazioni generalmente vengono sviluppate in modo da avere più pagine; quindi, possiamo dire che nel loro insieme le Activity costituiscono il flusso in cui l'utente entra per poter sfruttare al meglio le funzionalità messe a disposizione. E per questo è importante realizzare una buona interfaccia tanto quanto una corretta progettazione della navigazione tra le nostre pagine, in modo tale da poter scorrere i contenuti in maniera coerente e corretto.

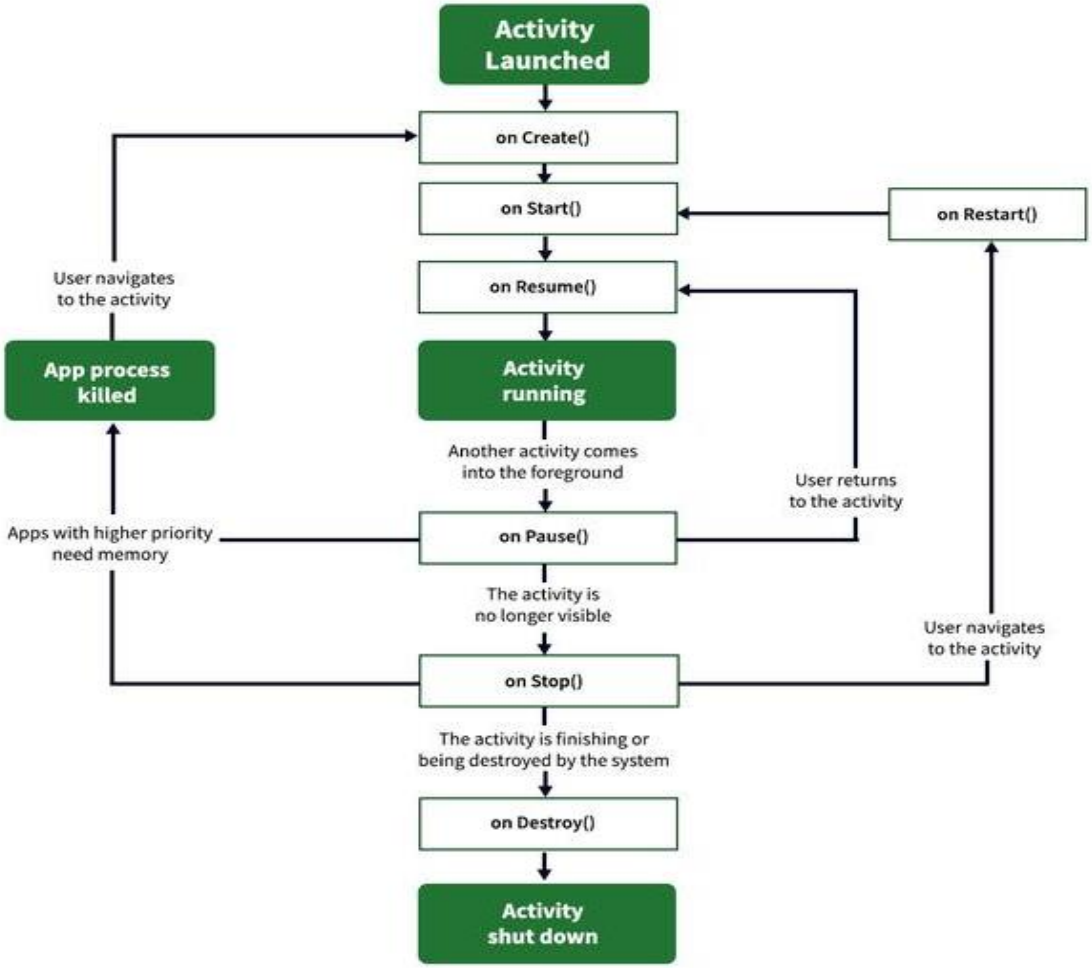
Quando un Activity viene mandata in esecuzione, vengono invocati tre metodi:

- **onCreate**: ovvero l'activity viene creata. Il programmatore deve assegnare le configurazioni di base e definire quale sarà il layout dell'interfaccia;
- **onStart**: ovvero l'activity diventa visibile. È il momento in cui si possono attivare le funzionalità e i servizi che devono offrire informazioni all'utente;
- **onResume**: ovvero l'activity diventa la destinataria di tutti gli input dell'utente.

Nel momento in cui l'utente sposta la sua attenzione su un'altra attività del sistema, Android metti in riposo l'Activity corrente, ad esempio se l'utente apre un'applicazione diversa, riceve una telefonata o semplicemente viene attivata un'altra Activity anche nella stessa applicazione, l'Activity corrente che è in funzione viene messa in pausa e anche questo percorso, passa per tre metodi di callback:

- **onPause** (semplicemente possiamo dire che sia l'inverso di onResume) notifica la cessata interazione dell'utente con l'activity;
- **onStop** (contrario di onStart) segna la fine della visibilità dell'activity;
- **onDestroy** (contrapposto a onCreate) segna la distruzione dell'activity.

Possiamo considerare che i metodi di callback sono concepiti a coppie (un metodo di avvio con un metodo di arresto) quindi solitamente il lavoro che viene fatto nel metodo di avvio verrà annullato nel corrispondente metodo di arresto.



(fig. 2.4 il ciclo di vita di un'Activity Android)

2.7.2 Service

Service è un componente di fondamentale importanza nell'architettura Android. Esso è distinto dall'Activity, in quanto ha il compito di restare in esecuzione in background a prescindere dall'interfaccia grafica, un esempio nel quale viene utilizzato può essere in un lettore musicale che continua la sua attività in background anche se l'utente cambia

applicazione. I Service sono classificabili in due tipologie, a seconda del modo in cui vengono avviati, abbiamo i Service Started e i Service Bound, i Service Started vengono avviati tramite il metodo startService() e la loro particolarità è di essere eseguiti in background indefinitamente anche se la componente che li ha avviati viene terminata, mentre i Service Bound vivono in una modalità client-server e hanno senso soltanto se qualche altra componente si collega a loro e vengono interrotti nel momento in cui non vi sono più client collegati a essi. Possiamo notare che i service delle due categorie non sono molto diversi e ciò che li distingue sono il modo nel quale vengono avviati e i metodi di callback che sono implementati al loro interno. Uno stesso service può essere avviato in maniera Started o Bound.

2.7.3 Content Provider e Broadcast Receiver

Ogni applicazione Android deve avere cura dei dati custoditi e gestiti da essa, e considerando che nessuna applicazione può mettere le mani nelle informazioni che sono custodite in un'altra applicazione è previsto un meccanismo ufficiale di sistema per la condivisione dei dati, e questo sistema è il Content Provider. Si tratta di componenti che permettono di leggere e modificare i dati presenti in una determinata sorgente, mediante l'invio di chiamate ad un indirizzo univoco, detto URI. I comandi che vengono passati rispecchiano le quattro operazioni CRUD eseguibili sui database (ovvero Create, Read, Update e Delete). I Content Provider vengono utilizzati per condividere basi di dati con utilità generale presenti nel sistema operativo ad esempio i contatti, il calendario, il dizionario utente e molto altro. Inoltre, vengono utilizzati per fare in modo che più applicazioni possano condividere gli stessi dati, cosa che capita spesso quando più app provengono dallo stesso produttore.

Un Broadcast Receiver invece è un componente che reagisce ad un invio di messaggi a livello di sistema con i quali Android notifica l'avvenimento di un determinato evento, ad

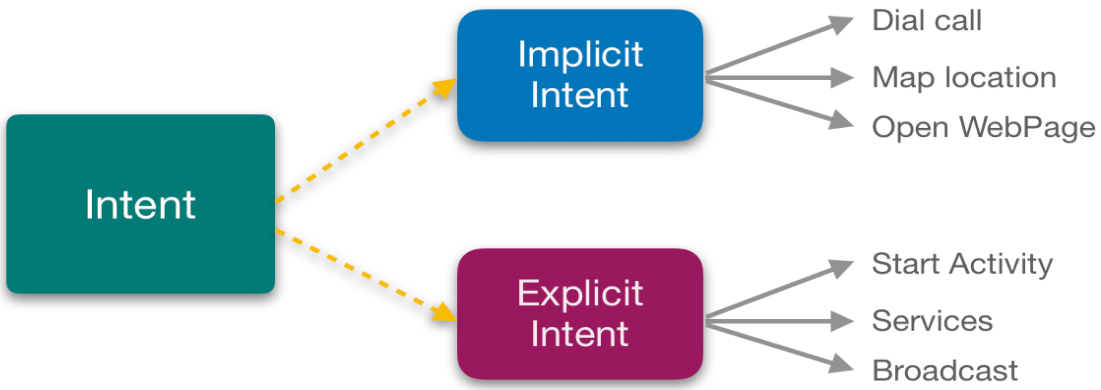
esempio l'arrivo di una notifica WhatsApp, di un SMS, di una chiamata o sollecita l'esecuzione di certe azioni. I Broadcast Receiver non utilizzano un'interfaccia grafica sebbene inoltrino notifiche alla barra di stato per avvisare l'utente dell'avvenimento.

2.7.4 Gli Intent

Un Intent può essere definito semplicemente come un messaggio che viene passato tra le componenti (Activity, Service, Content Provider e Broadcast Reciever) e serve alle componenti per richiedere l'esecuzione di un'azione da parte di altre componenti. Tra gli utilizzi principali dei Intent possiamo trovare l'avvio di un Activity, l'avvio di un Service o l'invio di un messaggio in broadcast che può essere ricevuto da ogni applicazione.

Esistono due tipologie d'Intent:

- Impliciti: ovvero che non specificano una componente da attivare ma quale azione verrà svolta, ad esempio quando premiamo su un link e il dispositivo ci chiede con quale app vogliamo che venga svolta l'azione, nel caso del nostro esempio ci verrà chiesto con quale browser vogliamo che venga aperto il link.
- Espliciti: in questo tipo d'intent viene dichiarato che componente dovrà essere innescata, sono particolarmente utili nell'apertura di una nuova Activity.



(fig. 2.5 tipi di intent)

2.8 Database

Come Database per SmartHealth ho scelto di utilizzare FireBase di Google per lo sviluppo dell'applicazione, poiché, esaminando le diverse tecnologie e gli strumenti per lo sviluppo di un'applicazione, sono giunto alla conclusione che FireBase, è uno degli strumenti più utilizzati nello sviluppo di applicazioni Android, il che lo rende facile da imparare grazie ancora una volta alle innumerevoli guide e tutorial. Inoltre, FireBase è già integrato in Android Studio, quindi non avremmo problemi di o dipendenze o durante la sua installazione.

2.8.1 FireBase Database

FireBase è una piattaforma che permette di creare il backend delle nostre applicazioni, è supportata da Google anche essendo open source, FireBase sfrutta l'infrastruttura di Google per fornirci una suite di strumenti per poter scrivere, analizzare e mantenere le nostre applicazioni. FireBase, infatti, offre differenti funzionalità come analisi, database (usando strutture noSQL), messaggistica e segnalazione di arresti anomali per la gestione di applicazioni Android ma non solo.

Caratteristiche non trascurabili di FireBase sono:

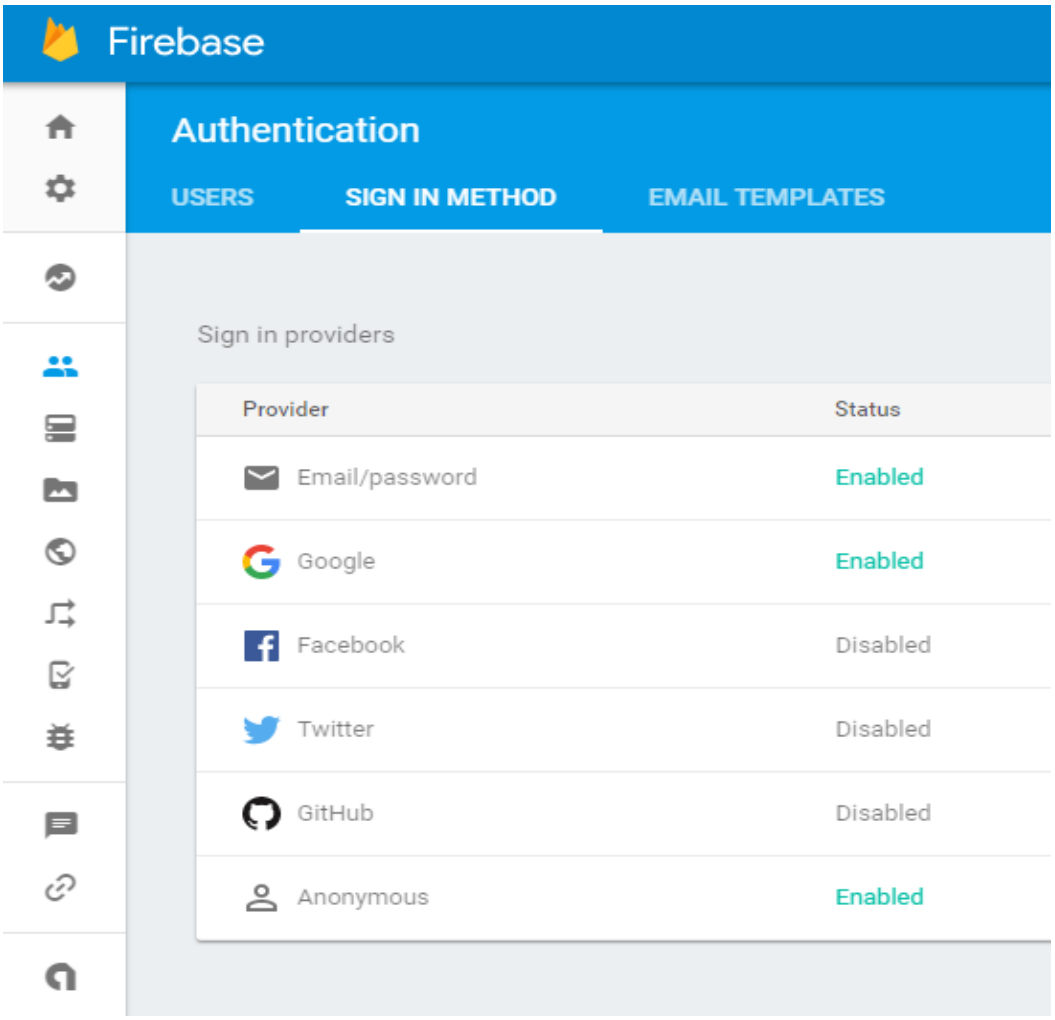
- capacità di sincronizzazione dati e aggiornamento immediato, oltre che di storage.
- differenti librerie client per far integrare FireBase con ogni applicazione Android, JavaScript, Java e con sistemi Apple
- i dati in FireBase sono e sottoposti a backup costantemente e la comunicazione con il cliente succede sempre in modalità crittografata.
- è possibile usufruire di piani con costi diversi. Per il nostro progetto abbiamo utilizzato il piano gratuito.

Inoltre, FireBase ci rende disponibili diversi strumenti per semplificarci il lavoro, e tra i più importanti abbiamo: FireBase Authentication, FireBase Realtime Database, FireBase Storage, FireBase Functions, Push Notification, FireBase Analytics e tanti altri strumenti.

2.8.2 FireBase Authentication

FireBase Authentication ci permette d'implementare funzioni per l'autenticazione degli utenti nella nostra applicazione attraverso l'uso di password, numeri di telefono, o providers d'identità come Facebook, Twitter, Google e altri molti altri.

È possibile integrare manualmente uno o più metodi di accesso e registrazione nella nostra app, come si vede nella Figura d'esempio qui sotto:



(fig. 2.6 esempio di metodi di autenticazione)

Nel nostro progetto si è scelto di basare l'autenticazione solo sull'email e sulla password dell'utente. Quindi per prima cosa bisognerà inserite le credenziali ovvero e-mail e una password nel nostro caso, che verranno poi passate alla SDK di FireBase Authentication. I servizi back-end di FireBase poi controlleranno le credenziali immesse e dopodiché sarà possibile accedere alle informazioni di base dello user e controllare il suo accesso ai dati che sono immagazzinati in FireBase. Inoltre, attraverso la modifica delle regole di sicurezza sarà anche possibile limitare le autorizzazioni a un certo utente, che di default ha i permessi di lettura e scrittura.

2.8.3 Perché è stato scelto un Database NoSQL

Prima di iniziare a sviluppare SmartHealth ho effettuato delle ricerche per valutare quali sono i pro e i contro di un database SQL/NoSQL ed è emerso, che non sempre i DBMS (Database Management System) tradizionali sono in grado di gestire quantità di dati sufficientemente grandi, se non al costo di performance molto limitate e con costi piuttosto elevati.

Qui di seguito ho raccolto alcuni punti di debolezza che ho trovato nell'uso dei DBMS (Database Management System) relazionali:

- Join: nonostante siano molto efficaci, queste operazioni però coinvolgono spesso molte più righe del necessario, limitando così le performance delle interrogazioni eseguite;
- struttura rigida delle tabelle: la struttura delle tabelle si rivela utile fino a quando si ha bisogno di dover introdurre informazioni caratterizzate sempre dalle stesse proprietà, ma sono molto meno efficiente per informazioni di natura eterogenea;

- il conflitto di impedenza consiste nella diversità costitutiva dei record delle tabelle e degli oggetti generalmente utilizzati per la gestione dei dati nei software che interagiscono con le basi di dati.

Dopo le svariate ricerche effettuate ho raccolto anche le informazioni sul perché sia conveniente utilizzare un Database NoSQL piuttosto che un Database relazionale. In generale, è opportuno scegliere l'approccio NoSQL quando:

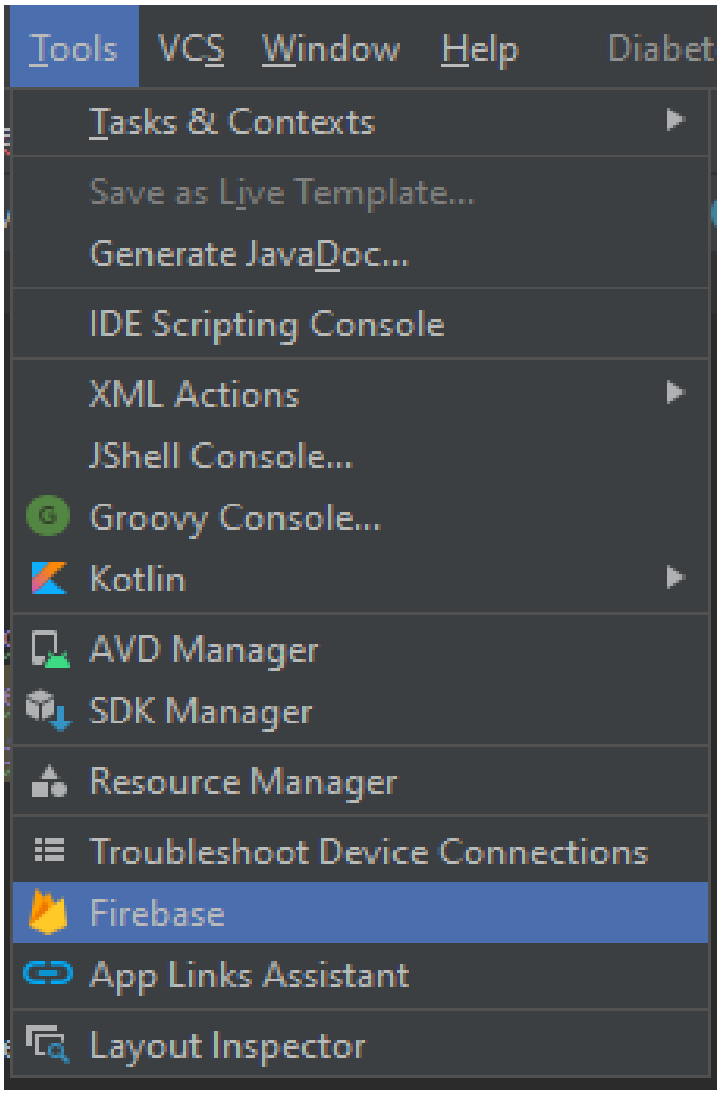
- la struttura dei dati non è definibile a priori, ad esempio proviamo a pensare alla gestione dei contenuti che impongono oggi i social network per la gestione dei contenuti;
- i dati disposti nei vari oggetti sono molto uniti fra di loro e il Join rischia di non essere più uno strumento ottimale, in questo caso sarebbe da preferire la navigazione tra oggetti sfruttando i legami tra i vari nodi di informazione;
- quando è necessario interagire molto frequentemente con il database.
- Oppure quando abbiamo bisogno di prestazioni più elevate.

	NoSQL	SQL
Model	Non-relational	Relational
	Stores data in JSON documents, key/value pairs, wide column stores, or graphs	Stores data in a table
Data	Offers flexibility as not every record needs to store the same properties	Great for solutions where every record has the same properties
	New properties can be added on the fly	Adding a new property may require altering schemas or backfilling data
	Relationships are often captured by denormalizing data and presenting it in a single record	Relationships are often captured in a using joins to resolve references across tables
	Good for semi-structured data	Good for structured data
Schema	Dynamic or flexible schemas	Strict schema
	Database is schema-agnostic and the schema is dictated by the application. This allows for agility and highly iterative development	Schema must be maintained and kept in sync between application and database
Transactions	ACID transaction support varies per solution	Supports ACID transactions
Consistency	Consistency varies per solution, some solutions have tunable consistency	Strong consistency supported
Scale	Scales well horizontally	Scales well vertically

(fig. 2.7 schema riassuntivo che confronta i Database NoSQL e SQL)

2.8.4 Come aggiungere FireBase al nostro progetto

Integrare FireBase nella propria applicazione è molto facile, infatti in Android Studio basta cliccare Tools e poi su FireBase e seguendo le istruzioni che ci vengono date FireBase verrà aggiunto al nostro progetto insieme alle dipendenze essenziali dello strumento di FireBase che si vuole utilizzare.



(fig. 2.8 Come aggiungere FireBase)

Parte II

Progetto e Sviluppo

Capitolo 3

La progettazione

3.1 Analisi dei Requisiti

L'analisi dei requisiti è una fase necessaria per la nascita dell'applicazione. Rappresenta un'attività preparatoria per procedere allo sviluppo di un sistema software. Lo scopo che si prefigge questa analisi principalmente è quello di definire le funzionalità che la nostra applicazione deve offrire, cioè in poche parole i requisiti necessari da soddisfare durante lo sviluppo del software. Quindi bisogna specificare che cosa deve fare e quali sono gli obiettivi della nostra applicazione, indipendentemente dal linguaggio di programmazione, l'architettura e la tecnologia che utilizziamo.

3.2 Funzionalità di base

Il primo passaggio per iniziare è stato quello di analizzare gli obbiettivi principali che l'applicazione dovrà svolgere, ovvero le funzionalità offerte per raggiungere uno scopo.

Le operazioni fondamentali dell'applicazione sono le seguenti:

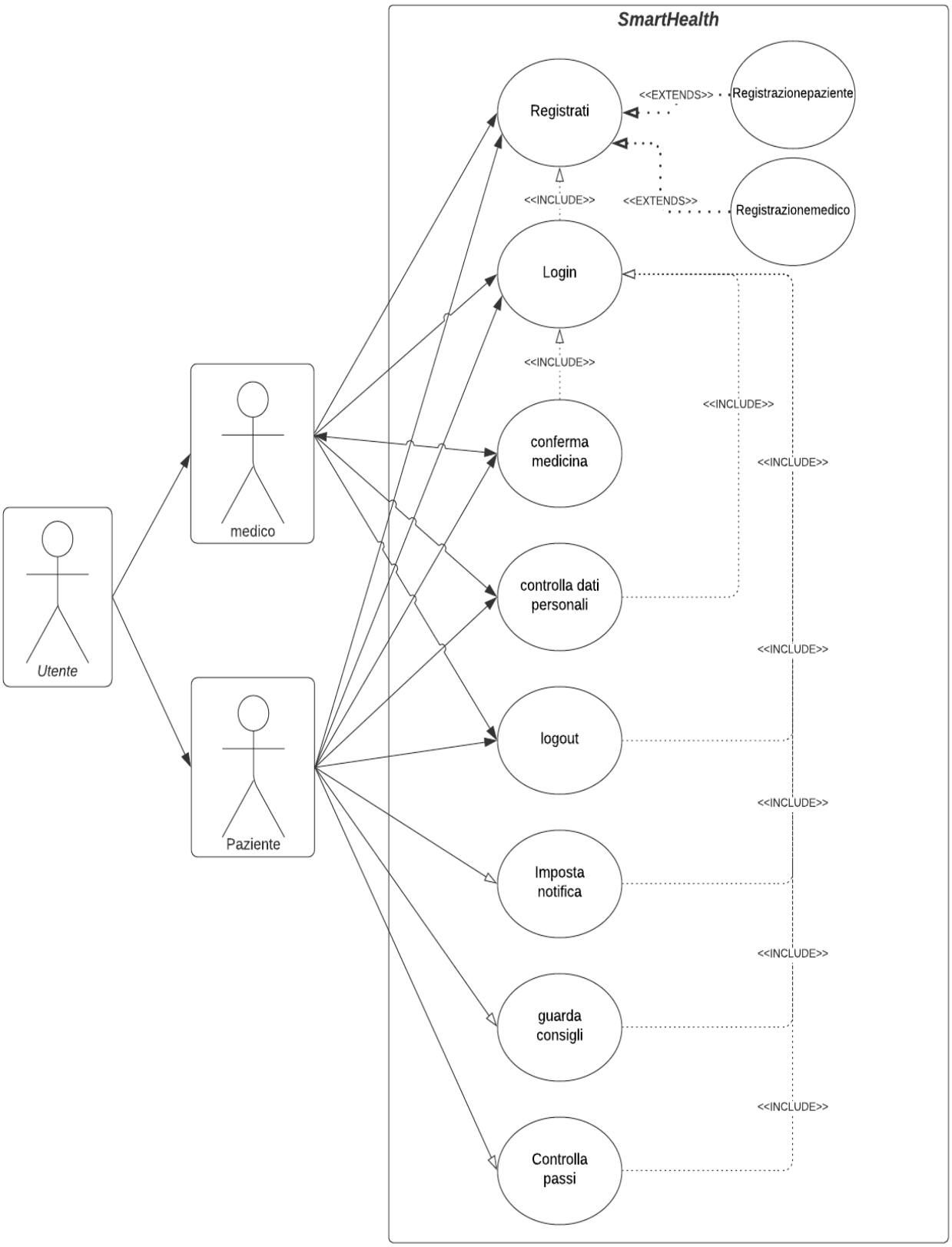
- Registrazione: La fase di registrazione è fondamentale sia per gli utenti Pazienti che per l'utente Medico. Verrà mostrata una schermata dove l'utente avrà la possibilità di scegliere se è un paziente oppure un medico, e a seconda della sua scelta verrà reindirizzato alla schermata di registrazione del medico o quella del paziente
- Login: L'applicazione ovviamente dà la possibilità agli utenti già registrati di fare il login e accedere alle funzioni adibite al tipo di utente.

- Impostare notifica: gli utenti registrati come pazienti possono impostare una notifica che ricordi loro di assumere la medicina o l'insulina.
- Conferma di aver preso medicina: i pazienti una volta dopo aver assunto il medicinale o l'insulina dovranno dare conferma sull'applicazione, tale conferma verrà inviata al medico di base
- Attività fisica: gli utenti pazienti potranno scegliere un obbiettivo di passi giornaliero, e potranno controllare i propri progressi grazie al contapassi di SmartHealth.
- Consigli: gli utenti di tipo paziente in questa sezione potranno trovare consigli utili su alimentazione e stile di vita.
- Controllo pazienti: l'utente di tipo medico potrà controllare giornalmente se i suoi pazienti hanno assunto la medicina oppure no
- Controllo dati: l'utente indifferentemente dal suo tipo potrà controllare i propri dati personali.
- Logout: Gli utenti hanno possibilità di effettuare il logout dall'applicazione in qualsiasi momento.

3.3 Caso d'uso

Il diagramma dei casi d'uso rientra nell'ambito dell'analisi, è un metodo per realizzare in modo non confuso la raccolta dei requisiti, focalizzandoci sugli attori (ovvero chi ha a che fare con il sistema) e su che cosa viene fatto (il caso d'uso), Il risultato finale che otterremo sarà una descrizione degli scenari di utilizzo dell'applicazione da parte degli attori che la utilizzano. Avremmo così una visione ad alto livello di quello che l'applicazione può fare, senza dettagli implementativi.

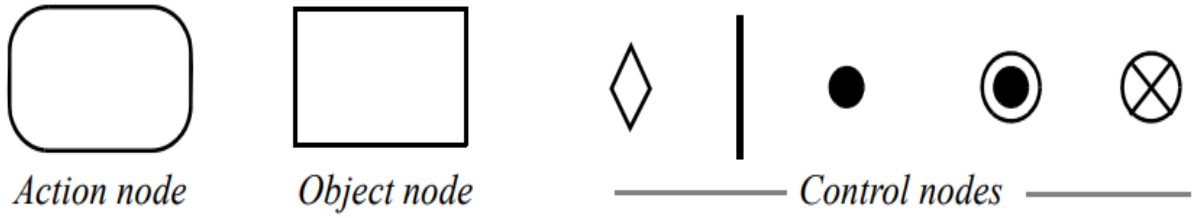
Nella nostra applicazione l'attore è l'utente che si suddivide in medico e paziente invece i casi d'uso sono raffigurati dalle funzionalità citate nel paragrafo scorso.



(fig. 3.1 Diagramma dei Casi d'uso)

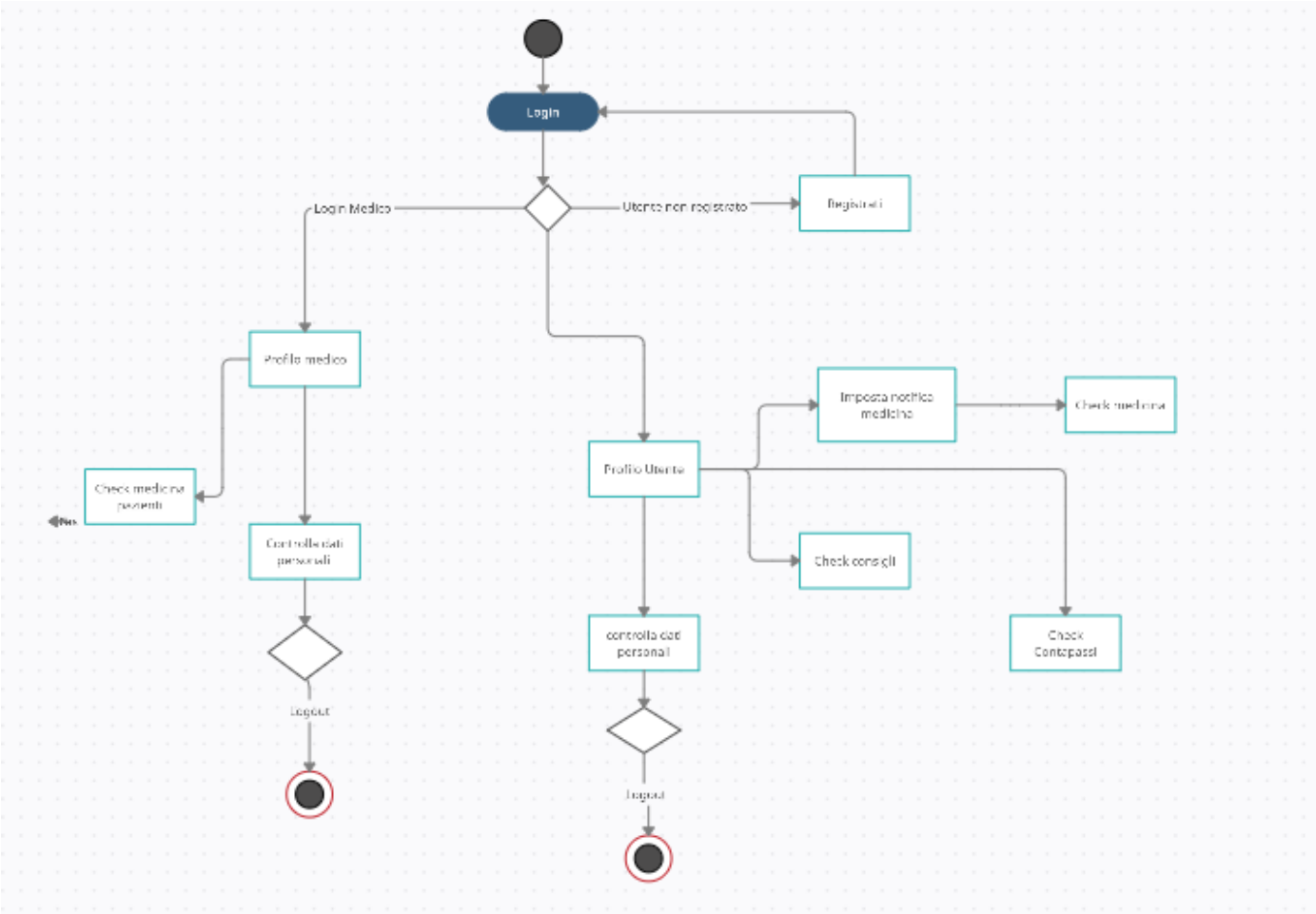
3.4 Diagramma delle attività

Il diagramma delle attività UML ci aiuta a visualizzare a un livello più dettagliato un certo caso d'uso, è un diagramma che illustra il flusso delle attività attraverso un sistema. Nel nostro caso vogliamo andare a modellare il flusso di un caso d'uso dell'applicazione. I principali elementi che compongono un diagramma delle attività sono i seguenti:

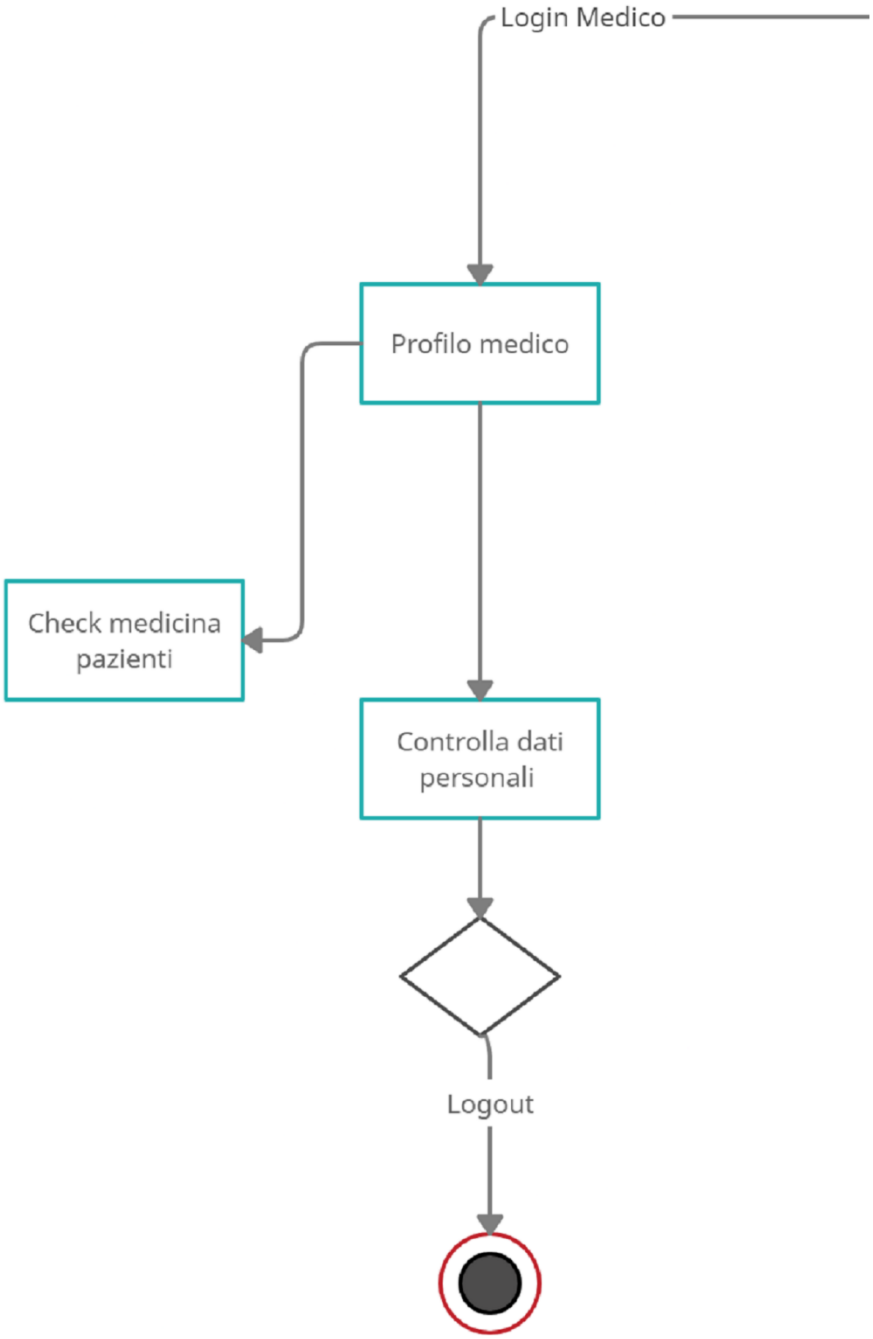


(fig. 3.2 Legenda del Diagramma delle attività)

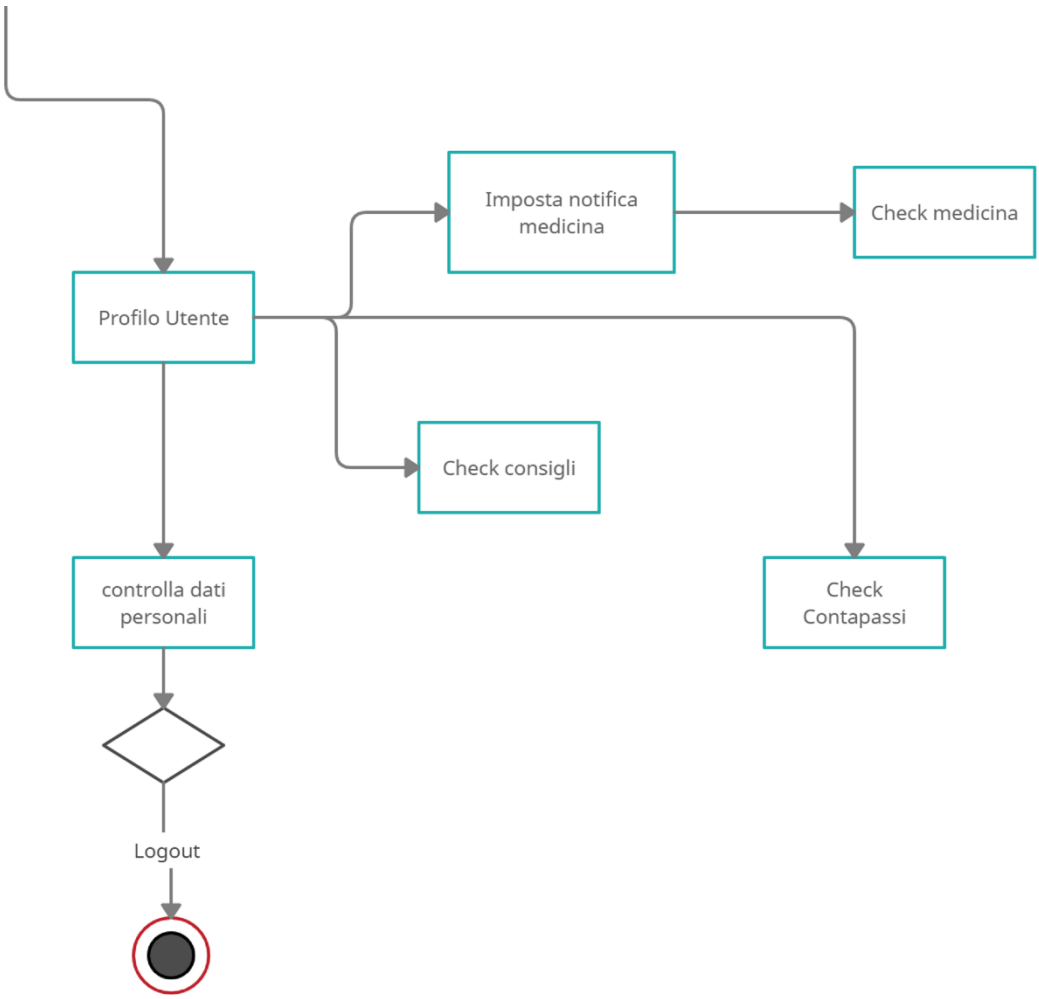
Con l'utilizzo delle componenti riportare nella figura 3.2 possiamo rappresentare le transizioni del flusso di un caso d'uso, andando a descrivere quindi il comportamento dinamico del nostro sistema considerando i flussi alternativi. La figura 3.3 nella pagina seguente mostra il diagramma delle attività relativo alla nostra applicazione.



(fig. 3.3 Diagramma delle attività)



(Figura 3.4: Diagramma delle attività zoom prima parte)

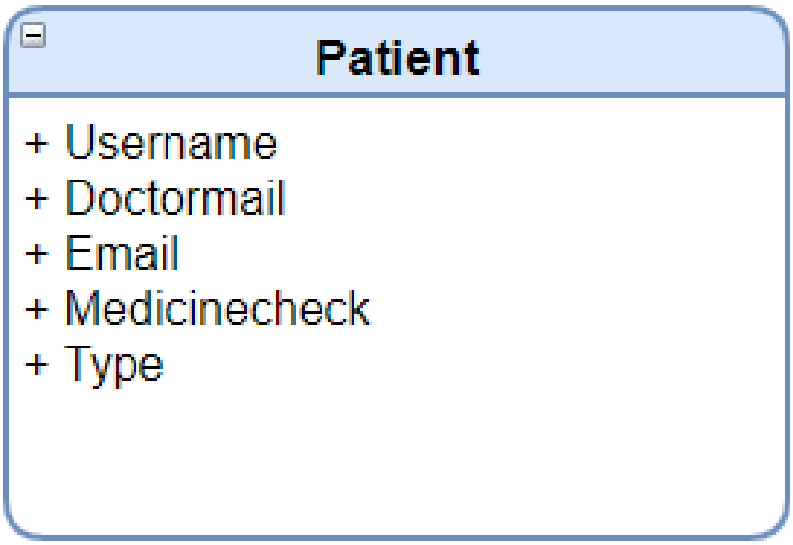


(Figura 3.5: Diagramma delle attività zoom seconda parte)

3.5 Progettazione Database

Come è già stato spiegato precedentemente nella sezione delle tecnologie utilizzate, il database scelto per il nostro progetto è stato FireBase. Quindi la fase di progettazione del database non è avvenuta attraverso l'ausilio di uno schema Entità-Relazioni, strumento che risulta incongruente con la filosofia di un database NoSQL. La progettazione, nel nostro caso, è stata effettuata seguendo la logica NoSQL, nel quale i dati sono immagazzinati all'interno dei file JSON (file esportabile direttamente dalla console di FireBase del progetto).

3.6 Costruzione dell'entità



(Figura 3.6: Entità patient)

Quando un utente si registra su SmartHealth, FireBase salva i suoi dati all'interno di un'entità comune chiamata patient, a seconda dell'utente scelto FireBase salverà i dati

utili per quel tipo di utente, l'attributo Type è l'attributo più importante perché esso determina se l'utente è un Paziente o un Medico, se l'utente sceglie di registrarsi come paziente allora gli attributi che verranno salvati saranno:

- Username – contiene il nome utente scelto durante la registrazione
- Doctormail – l'email del medico curante
- Email – l'indirizzo mail utilizzato dall'utente per registrarsi
- Medicinecheck – un booleano che indica se l'utente ha preso la medicina o no
- Type – per determinare che tipo di utente è

Mentre se l'utente scegli di registrarsi come medico avrà come attributi soltanto: l'username, la sua email personale e il valore che indica il tipo di utente, tutti gli altri attributi non verranno compilati in questo caso. L'applicazione effettuerà un controllo durante il login per verificare il type dell'user e a seconda di questo risultato caricherà le varie schermate.

Di seguito possiamo notare due esempi di utenti che sono registrati nel nostro database.

```
doctormail: "doctor@manroop.com"
email: "manroopsingh@hotmail.co.uk"
medicineCheck: true
name: "manroop singh"
type: "patient"
```

(Figura 3.7: esempio di utente Paziente)

```
email: "doctor@manroop.com"
name: "Manroop Singh"
type: "doctor"
```

(Figura 3.8: esempio di utente Medico)

Capitolo 4

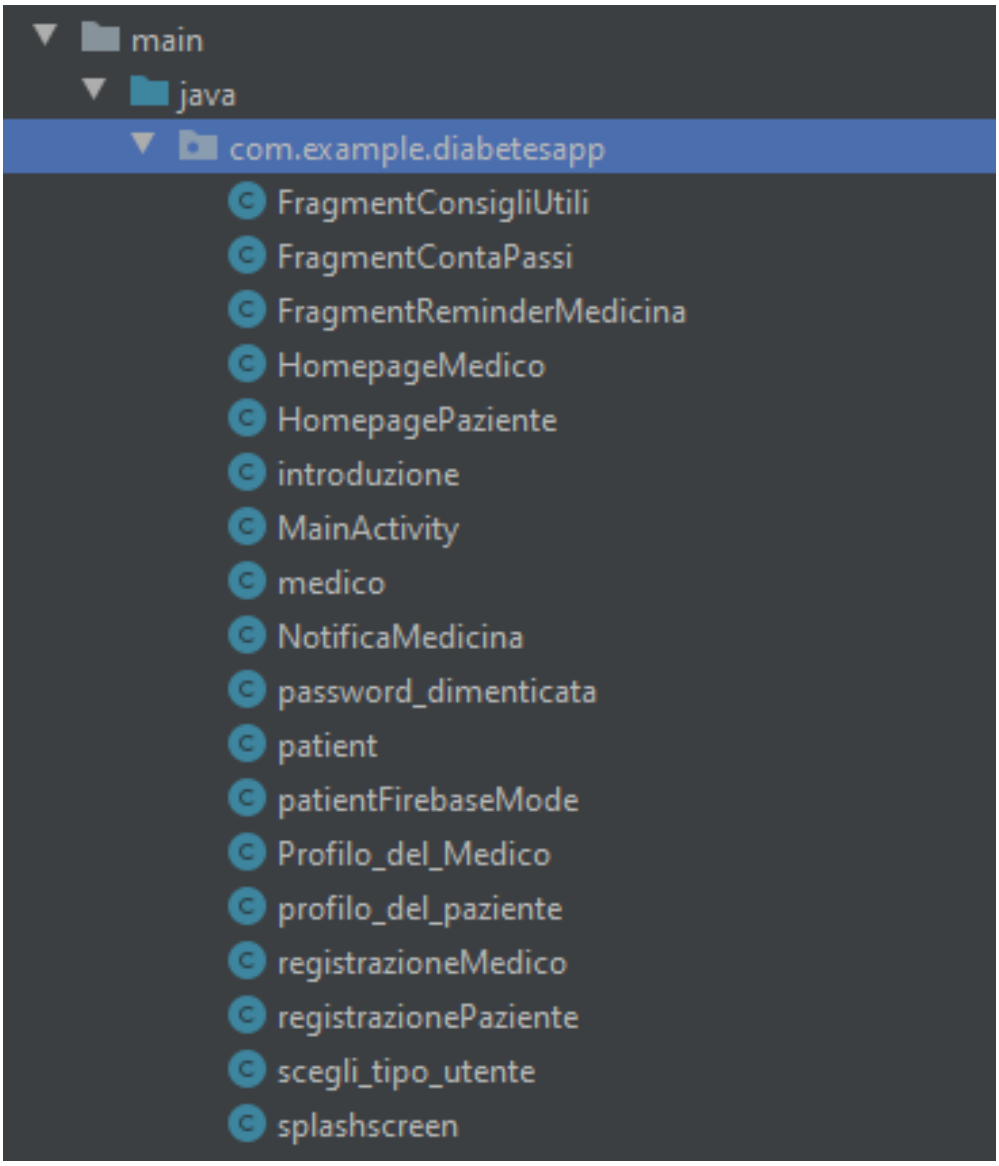
Implementazione

4.1- Introduzione all'implementazione

Questo capitolo tratterà in dettaglio tutte le fasi implementative dello sviluppo dell'applicazione. Si analizzeranno uno alla volta tutti i file utilizzati nella realizzazione del progetto, verranno studiate e spiegate tutte le componenti, le interfacce e le loro Activity. Come è stato già detto in precedenza, si è utilizzato Android Studio come IDE, Java come linguaggio di programmazione e FireBase per la gestione dei dati. Le operazioni di controllo e i test per verificare la correttezza delle funzionalità implementate nell'applicazione sono stati effettuate attraverso l'utilizzo dell'emulatore virtuale fornitoci da Android Studio, il quale però è risultato uno strumento molto time-consuming causando anche rallentamenti al nostro sistema operativo. Ho quindi approfittato ancora una volta delle funzionalità di Android Studio che mi hanno permesso di svolgere queste operazioni direttamente su un dispositivo Android prestatomi gentilmente da mia madre, e questa funzionalità di Android si è rivelata molto utile velocizzando lo sviluppo dell'applicazione. Verranno mostrati per primi la struttura del progetto e il file Manifest, dopodiché il capitolo navigherà tra le varie activity soffermandosi sulle componenti, l'interfaccia e le funzionalità offerte.

4.2- Struttura del progetto

SmartHealth è composta da più file .java che sono contenuti nella cartella main\java\com.example.diabetesapp in Android Studio



(Figura 4.1: file java contenuti in com.example.diabetesapp)

4.3- Cartella Manifest

Ogni applicazione che viene sviluppata in Android deve contenere un file chiamato AndroidManifest.xml nella sua cartella principale. Il Manifest raccoglie le informazioni basilari sull'applicazione, informazioni che sono necessarie al sistema per far girare

qualsiasi porzione di codice della stessa. Tra le varie proprietà il Manifest si occupa anche di dare un nome al package Java dell'applicazione, di descrive le componenti dell'applicazione (attività, servizi, provider, ecc.), dichiara i permessi dell'app, e i permessi necessari alle altre applicazioni per poter interagire con la stessa ed elenca le

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.smarthealth">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACTIVITY_RECOGNITION" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/applogo"
        android:label="Smart Health"
        android:roundIcon="@drawable/applogo"
        android:supportsRtl="true"
        android:theme="@style/Theme.DiabetesApp">
        <activity android:name="com.example.smarthealth.password_dimenticata" />
        <activity android:name="com.example.smarthealth.patient" />
        <activity android:name="com.example.smarthealth.medico" />
        <activity android:name="com.example.smarthealth.scegli_tipo_utente" />
        <activity android:name="com.example.smarthealth.registrazioneMedico" />
        <activity android:name="com.example.smarthealth.registrazionePaziente" />
        <activity...>
        <activity android:name="com.example.smarthealth.MainActivity" />
        <activity...>
        <receiver android:name="com.example.smarthealth.NotificaMedicina" />
    </application>
</manifest>
```

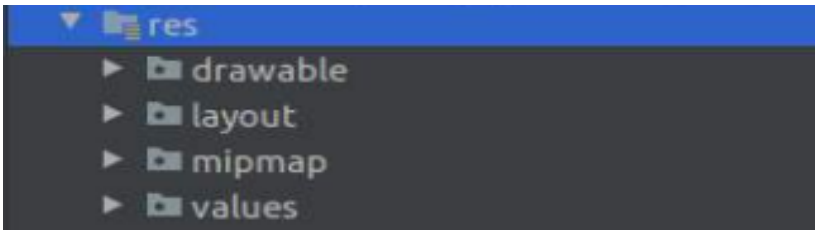
librerie necessarie all'applicazione per un corretto funzionamento.

(Fig. 4.2: Struttura del AndroidManifest.xml del nostro progetto Android)

4.4- Cartella Res

La cartella res contiene i file delle interfacce grafiche e i file esterni dell'applicazione. Le cartelle principali di res sono le seguenti:

- la cartella layout che serve a raggruppare tutti i file XML utilizzati per l'interfaccia grafica, ovvero il design della nostra applicazione.
- La cartella values che raggruppa tutti i dati come le stringhe, gli interi, i colori e gli array che vengono definiti dal programmatore tramite dei tag specifici in XML e verranno utilizzati poi direttamente nella nostra applicazione.
- Drawable invece contiene tutte le immagini e icone che utilizziamo.

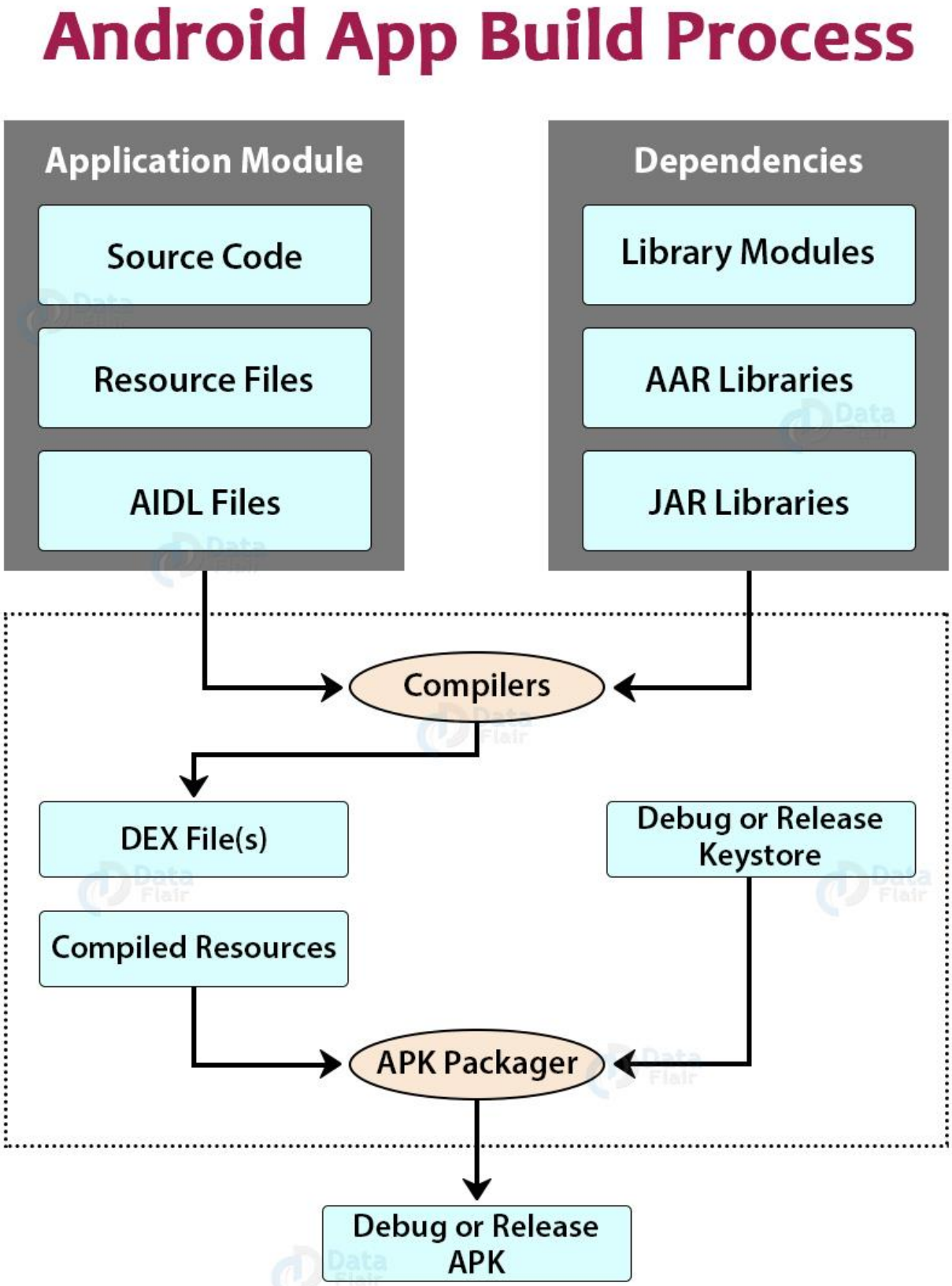


(Fig.4.3 La cartella res e le sue sottocartelle)

4.5- Gradle

Gradle è un sistema di build in esecuzione su Android Studio. Viene creato automaticamente da Android Studio quando avviamo un nuovo progetto, all'interno della cartella Gradle è possibile trovare tutte le impostazioni e gli altri file utilizzati per creare il progetto. Il suo scopo principale è quello di controllare e supervisionare l'operazione di compilazione, l'esecuzione dei casi di test e del raggruppamento del codice all'interno di un file apk per la nostra applicazione. Ogni progetto Android contiene due file build.gradle di cui uno è per l'applicazione, nel quale il programmatore definisce le configurazioni personalizzate per l'implementazione dell'applicazione, mentre l'altro file è per le build che sono a livello di progetto. Tra le sezioni più importanti abbiamo quella delle dependencies (dipendenze) che serve per poter espandere le funzionalità del nostro progetto grazie all'inclusione di librerie e di file binari.

Durante il processo di compilazione, il compilatore andrà a prendere il file sorgente assieme alle risorse, i file Jar delle librerie esterne e il file AndroidManifest.xml e lo convertirà in un file con estensione .dex (Dalvik Executables files), che al suo interno includerà il bytecode. A questo punto quindi APK Manager combinerà i file .dex e ogni altra risorsa all'interno di un unico file APK.



(fig. 4.4 come lavora un Build process)

4.6- Implementazione

Nel seguente paragrafo verranno illustrati i file principali che compongono il nostro progetto, soffermandoci in particolare su pezzi di codice fondamentali e sulla funzionalità delle activity.

4.6.1- Permission

Ogni applicazione Android vive all'interno di un sandbox, ovvero un ambiente chiuso dentro al quale l'applicazione opera in maniera principalmente separata dal resto del sistema. Talvolta, però, delle volte risulta necessario che l'applicazione debba "uscire" da questa gabbia per accedere a funzionalità, informazioni o apparecchiature hardware del dispositivo, e per far sì che l'applicazione acceda a tali servizi, essa ha bisogno di possedere alcuni "permessi" speciali, le cosiddette permission che è necessario dichiarare appositamente nel file *AndroidManifest.xml*. Ogni permission viene dichiarata tramite il tag `<uses-permission>`, da collocare al di fuori del nodo `<application>`. Una volta inseriti i permission, il sistema operativo saprà quali attività particolari l'app dovrà svolgere, e potrà concedere il permesso di effettuarle. Prima dell'installazione dell'applicazione, infatti, l'utente verrà sempre avvisato delle permission richieste dall'app: se l'utente accetta di installarla, accetta anche che il suo sistema operativo conceda tali permessi all'app.

4.6.2- Tipi di Permission

Le permission Android sono suddivise in tre famiglie in base al loro livello di protezione:

1. **Permission Normal**, ovvero le permission che non mettono a rischio la privacy dell'utente, tra le quali troviamo ad esempio:

    - Accesso alla rete:
      android.permission.INTERNET, consente all'applicazione di aprire socket di rete.
      android.permission.ACCESS_NETWORK_STATE il quale verifica lo stato della rete.

    - Accesso al Bluetooth:
      android.permission.BLUETOOTH,
      android.permission.BLUETOOTH_ADMIN.

    - Per l'impostazione di un allarme:
      com.android.alarm.permission.SET_ALARM.

2. **Permission Dangerous,** permission che sono potenzialmente più lesive della riservatezza degli utenti, ad esempio:

    - La localizzazione:

      android.permission.ACCESS_FINE_LOCATION

    - Il calendario:

android.permission.READ_CALENDAR e

android.permission.WRITE_CALENDAR

- I contatti:

  android.permission.READ_CONTACTS e

  android.permission.WRITE_CONTACTS

- Accesso ai file:

  android.permission.READ_EXTERNAL_STORAGE e

  android.permission.WRITE_EXTERNAL_STORAGE

3. **Permission signature:** queste permission vengono gestite a livello d'installazione ma sono utilizzabili solo se l'applicazione che ne richiede il permesso è firmata con lo stesso certificato di quella che ha definito la permission.

4.6.3- Gestione delle Permission

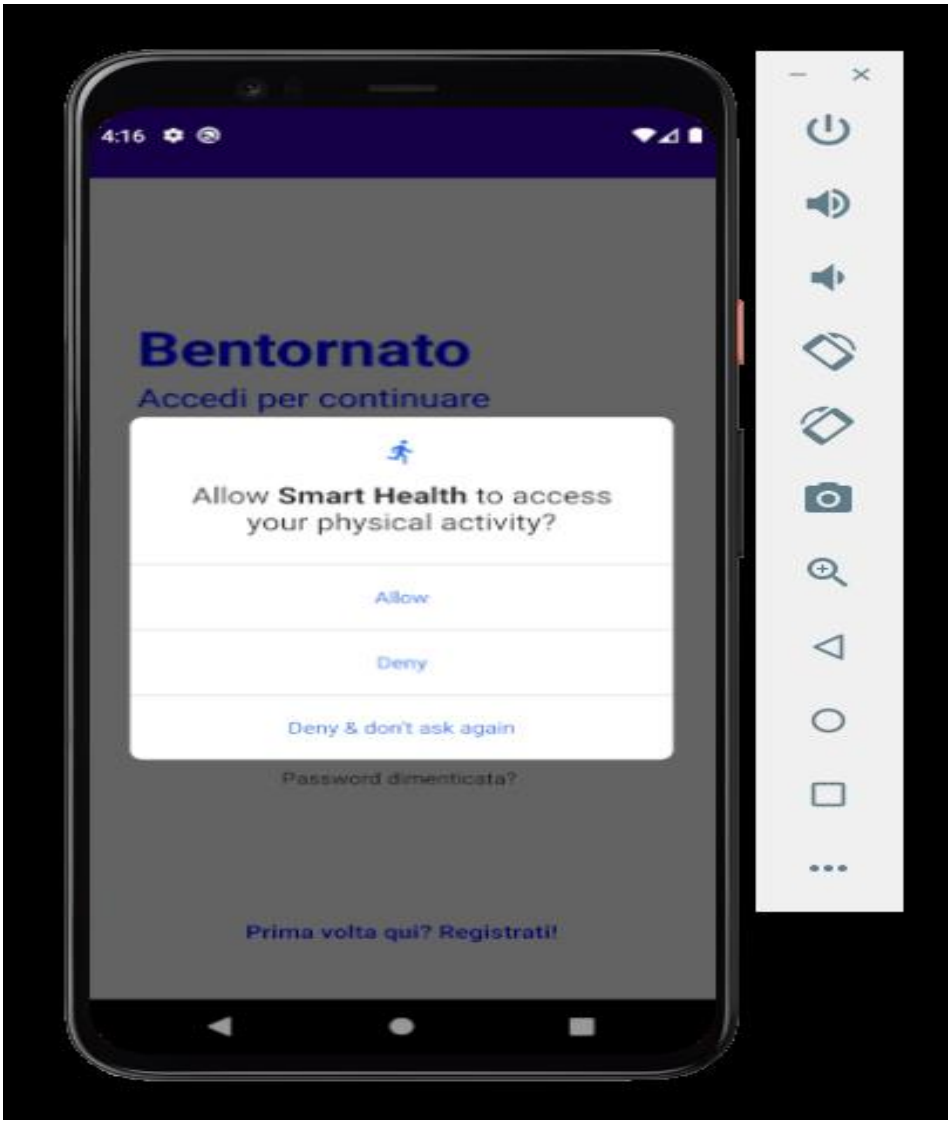
Le permission Dangerous vanno accettate a Runtime dall'utente durante l'utilizzo delle funzionalità che richiedono queste permission, mentre per le permission normal è bastante dichiarare l'utilizzo al momento dell'installazione dell'applicazione. Inoltre, una permission Dangerous che è già stata concessa, potrà esser revocata in qualsiasi istante

4.6.4- Permission di SmartHealth

Le permission richieste da SmartHealth sono le seguenti:

- android.permission.INTERNET
- android.permission.ACTIVITY_RECOGNITION

Il permesso a INTERNET serve per poter effettuare l'accesso, la registrazione e per inviare il check al database dopo aver confermato di aver preso la medicina, mentre l'ACTIVITY_RECOGNITION serve per monitorare i passi.



(fig. 4.5 richiesta di permission di SmartHealth)

4.6.5- AndroidManifest

Il file AndroidManifest.xml è il primo file che andremmo ad analizzare e contiene le informazioni di base del nostro progetto oltre ai permessi definiti nelle prime righe. Nella sezione application possiamo trovare i dettagli dell'applicazione come il nome dell'applicazione in android:label= , il logo che verrà visualizzato sul dispositivo e infine le varie activity.

```
<application
    android:allowBackup="true"
    android:icon="@drawable/applogo"
    android:label="@string/Nome_app"
    android:roundIcon="@drawable/applogo"
    android:supportsRtl="true"
    android:theme="@style/Theme.DiabetesApp">
    <activity android:name=".password_dimenticata" />
    <activity android:name=".patient" />
    <activity android:name=".medico" />
    <activity android:name=".scegli_tipo_utente" />
    <activity android:name=".registrazioneMedico" />
    <activity android:name=".registrazionePaziente" />
    <activity
        android:name=".introduzione"
        android:theme="@style/Theme.AppCompat.DayNight.NoActionBar" />
    <activity android:name=".MainActivity" />
    <activity
        android:name=".splashscreen"
        android:theme="@style/Theme.AppCompat.DayNight.NoActionBar"
        android:noHistory="true">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <receiver android:name=".NotificaMedicina" />
</application>
```

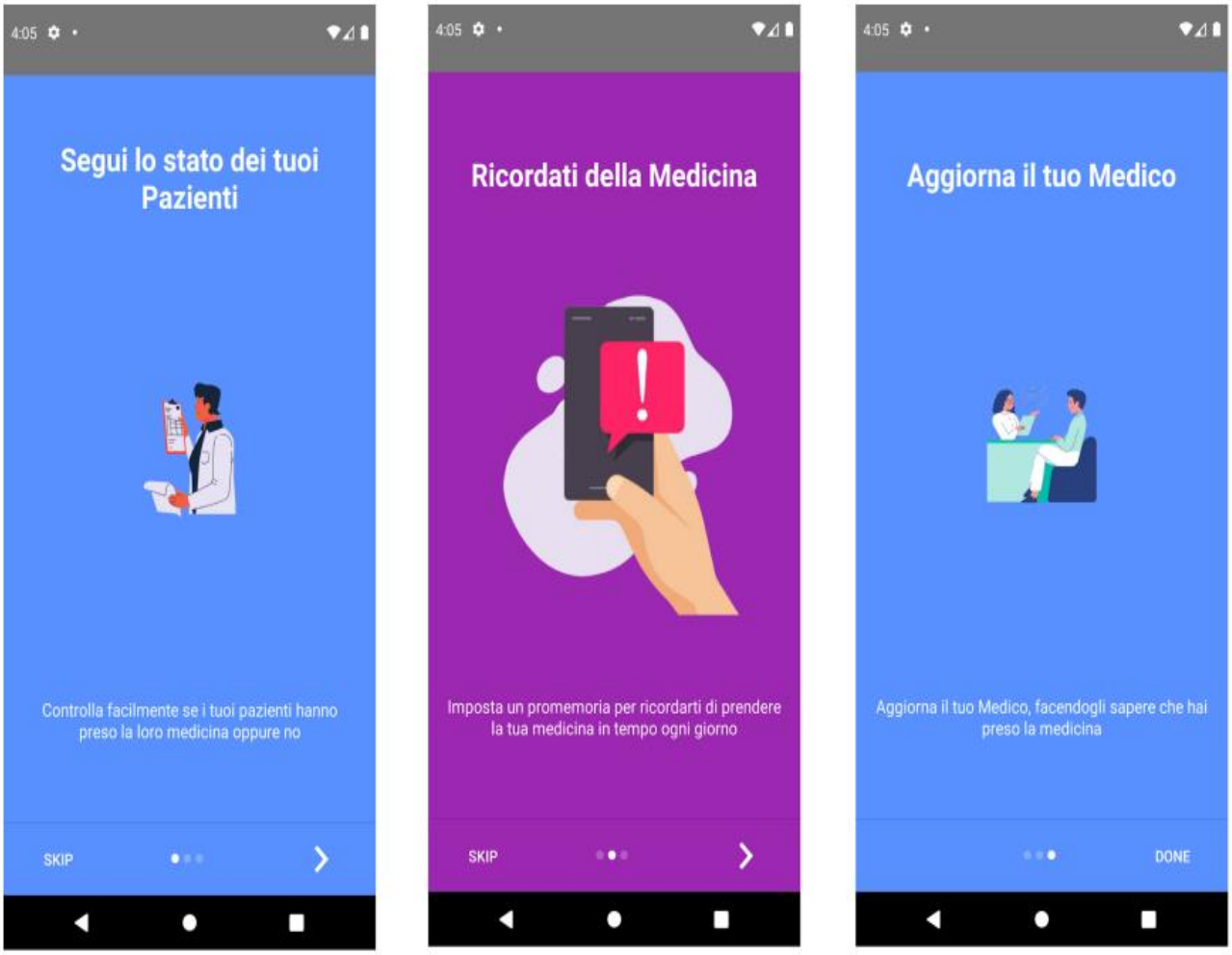
4.6.6- SplashScreen

La prima activity che verrà lanciata all'avvio di SmartHealth è la SplashScreen. Essa rappresenta la prima schermata visualizzata, mostrando un layout contenente il logo e il nome dell'applicazione. La schermata iniziale dura 3 secondi e viene mostrata a ogni avvio dell'applicazione, passati 3 secondi dalla splashscreen viene richiamata la classe introduzione.

(fig. 4.6 schermata iniziale, il logo e il nome dell'app visualizzato all'inizio)

4.6.7- Introduzione

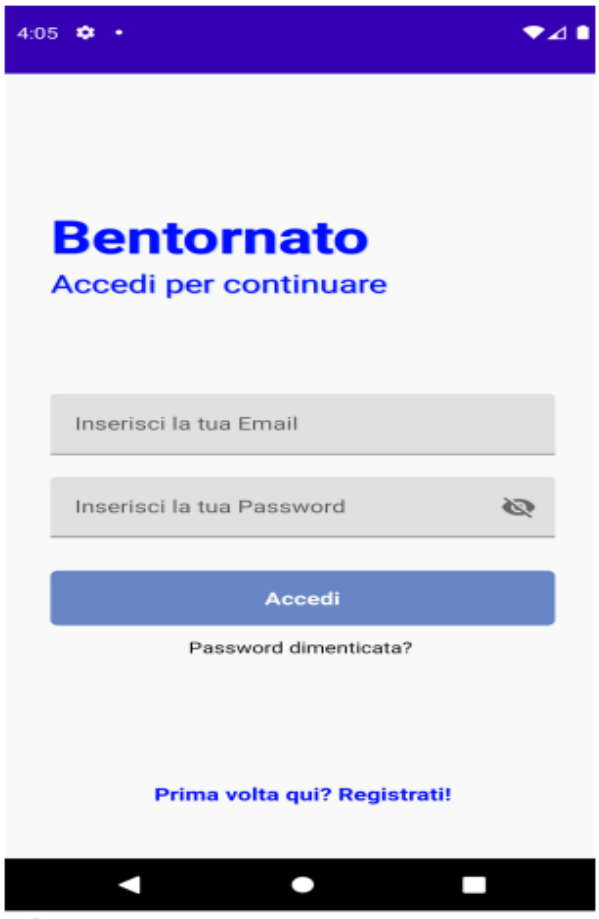
Dopo la splashscreen, solo durante il primo avvio dell'applicazione verranno mostrate 3 slide contenenti informazioni per capire meglio come utilizzare l'applicazione, l'utente volendo potrà saltare l'introduzione, mandare avanti o tornare indietro nella slide precedente, queste schermate d'introduzione non avranno una durata massima o minima.



(fig. 4.7 schermate di introduzione)

4.6.8- MainActivity

La MainActivity viene mostrata una volta che l'utente ha finito di vedere l'introduzione e ci permette di accedere se l'utente fosse già registrato in precedenza oppure di recuperare la password qualora non si ricordasse e infine di creare un nuovo account. Una volta inserite le credenziali, l'applicazione controllerà nel database se l'utente risulta registrato oppure no, e qualora risulti registrato effettuerà un ulteriore controllo nel Database riguardante la tipologia dell'utente, medico o paziente, e a seconda che sia l'uno o l'altro ci porterà nelle varie Homepage dedicate. Se l'utente non risultasse registrato l'applicazione ritornerebbe un errore facendo notare all'utente che esso non è registrato. Se l'utente dovesse cliccare sul pulsante password dimenticata verremo portati alla classe: password_dimenticata dove sarà possibile recuperare la password. E come ultimo caso possibile se l'utente dovesse cliccare su registrati verremo portati alla classe: scegli_tipo_utente.



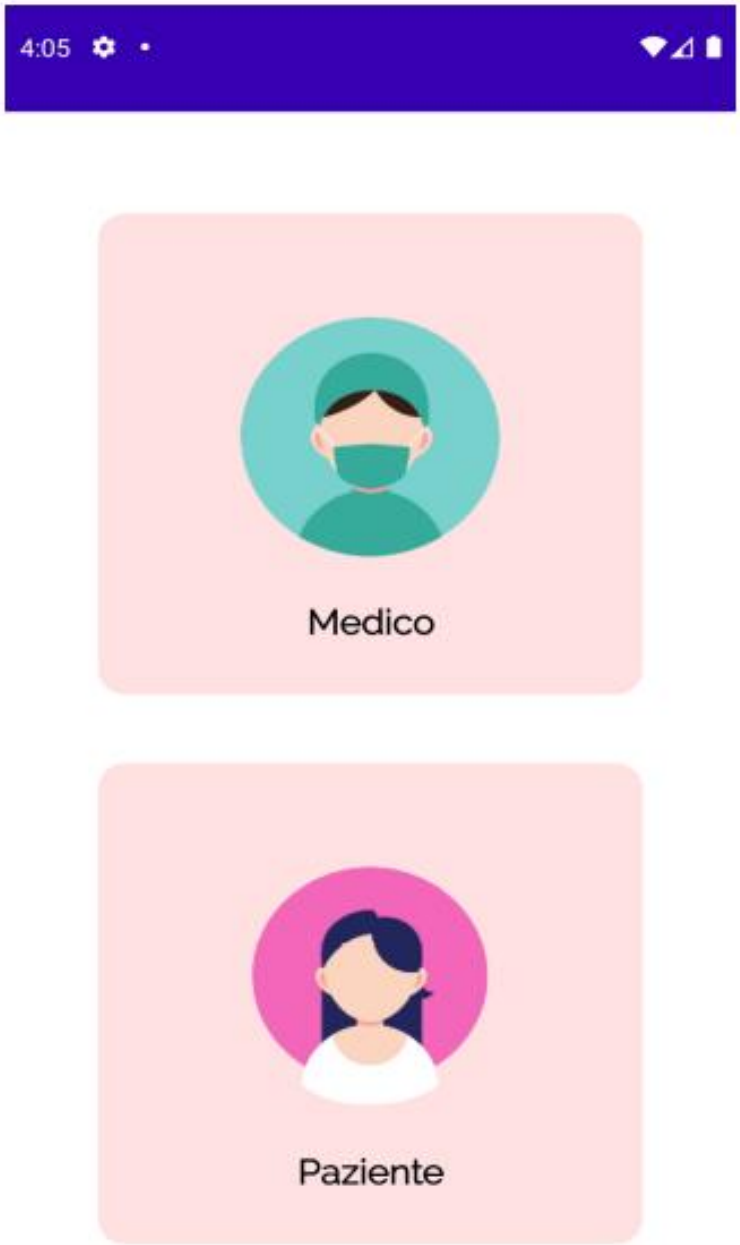
(fig. 4.8 MainActivity)

4.6.9- Scegli il tipo di utente

In questa schermata ci sarà possibile selezionare il tipo di utente che vogliamo registrare: Medico o Paziente. Scegliendo Medico verremo portati alla schermata di registrazione del Medico (classe: registrazioneMedico). Scegliendo Paziente invece verremmo portati alla schermata di registrazione del Paziente (classe: registrazionePaziente). Le due classi sono esteticamente uguali con l'unica differenza che l'utente Paziente avrà la possibilità d'inserire l'indirizzo mail del proprio medico di base, l'inserimento della mail del medico non è obbligatoria ma facoltativa: l'applicazione, infatti, è utilizzabile anche solo come un contapassi oppure per avere dei consigli per mantenere il diabete sotto controllo, i quali sono già presenti nell'applicazione. Una volta inserite le credenziali verrà inviata una mail di conferma all'indirizzo email inserito. L'E-mail di conferma viene inviata utilizzando il sistema di autenticazione di FireBase. E dopo aver confermato la mail attraverso il link ricevuto potremmo finalmente accedere, se invece provassimo ad accedere senza aver confermato la nostra mail, ci verrà restituito un errore.

```
public class scegli_tipo_utente extends AppCompatActivity {

    RelativeLayout mgotopatientsignin,mgotodoctorsignin;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sceglitipoutente);
        mgotodoctorsignin=findViewById(R.id.gotodoctorsignin);
        mgotopatientsignin=findViewById(R.id.gotopatientsignin);
        getSupportActionBar().hide();

        //se viene scelto medico apri la classe registrazionePaziente per far
registrazione del paziente
        mgotopatientsignin.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent=new Intent(scegli_tipo_utente.this,
registrazionePaziente.class);
                startActivity(intent);
            }
        });

        //se viene scelto medico apri la classe registrazionemedico per far
```

```
registrazione del medico
        mgotodoctorsignin.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent=new Intent(scegli_tipo_utente.this,
registrazioneMedico.class);
                startActivity(intent);
            }
        });
    }
}
```



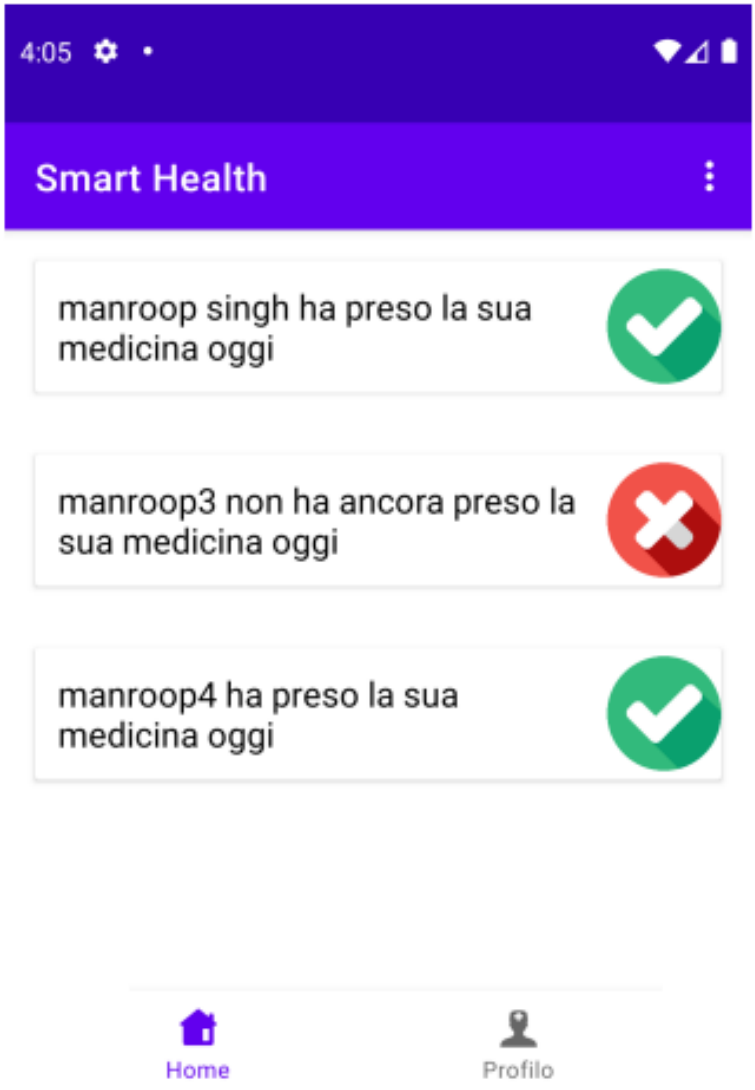
(fig. 4.9 La schermata che ci permette di scegliere il tipo di utente)

4.6.10 - Accesso come medico

Effettuando l'accesso come Medico, ci ritroveremo nella homepage contenente una lista con il nome dei nostri pazienti e un'icona che ci indicherà se l'utente quel giorno ha preso la medicina oppure no, l'icona della medicina viene aggiornata in tempo reale e resettata ogni giorno cosicché il medico possa tener traccia dei suoi pazienti e controllare se stanno prendendo la medicina regolarmente come stabilito oppure no. Come possiamo notare nella figura 4.10 nella parte inferiore della schermata oltre all'icona home abbiamo anche l'icona profilo, la quale se verrà cliccata ci porterà nella schermata profilo (classe: profilo_del_medico) dove troveremo le nostre informazioni. Mentre cliccando in alto a destra sui 3 puntini si aprirà un menù a tendina che ci darà la possibilità di fare Logout.

```
public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup
container, @Nullable Bundle savedInstanceState) {
    View v= inflater.inflate(R.layout.fragment_homepagemedico,container,false);

    getUserId();
    getPatientReference();

    Query
query=firebaseFirestore.collection("patient").whereEqualTo("doctormail",FirebaseAuth.ge
tInstance().getCurrentUser().getEmail());
    FirestoreRecyclerOptions<patientFirebaseMode> allpatientname= new
FirestoreRecyclerOptions.Builder<patientFirebaseMode>().setQuery(query,
patientFirebaseMode.class).build();


    patientAdapter=new FirestoreRecyclerAdapter<patientFirebaseMode,
NoteViewHolder>(allpatientname) {
        @Override
        protected void onBindViewHolder(@NonNull NoteViewHolder noteViewHolder, int i,
@NonNull patientFirebaseMode patientFirebaseMode) {

            noteViewHolder.patientName.setText(patientFirebaseMode.getName());

            if(patientFirebaseMode.getMedicineCheck()){
                noteViewHolder.ivPatientMedCheck.setImageResource(R.drawable.checked);
                noteViewHolder.patientName.setText(patientFirebaseMode.getName() + " ha
preso la sua medicina oggi");
            }
            else {
                noteViewHolder.ivPatientMedCheck.setImageResource(R.drawable.cancel);
                noteViewHolder.patientName.setText(patientFirebaseMode.getName() + "
non ha ancora preso la sua medicina oggi");
            }
        }
```

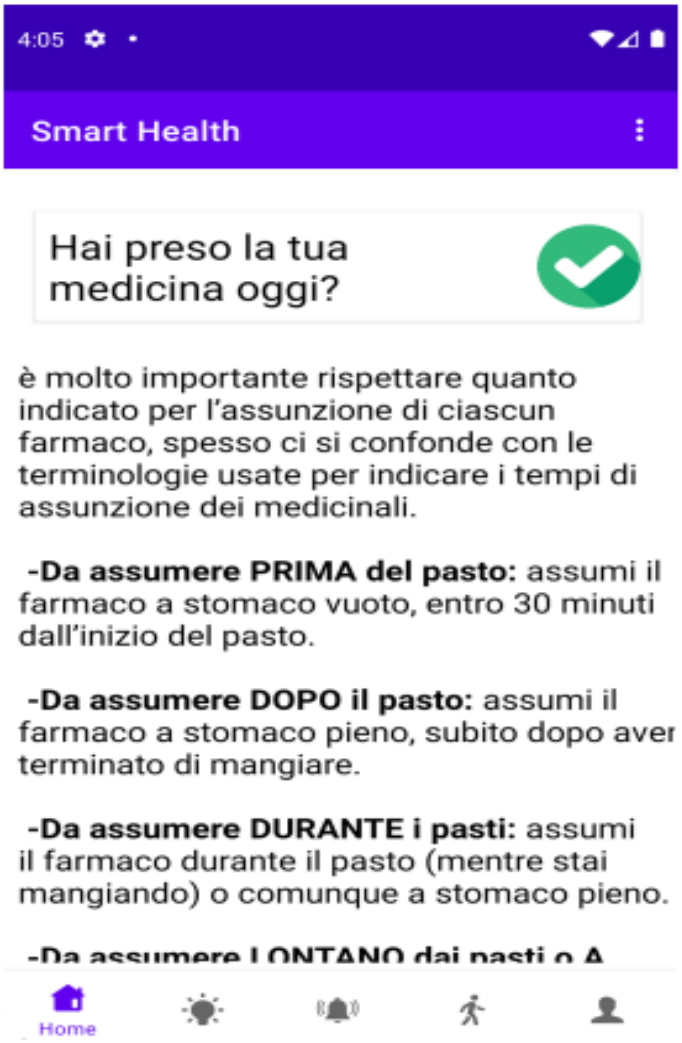
Nella pagina precedente è presente il codice della HomepageMedico che controlla nei database di FireBase se l'utente è un medico e i controlli in tempo reale per poter aggiornare lo stato del paziente.



(fig. 4.10 Homepage del Medico)

4.6.11 - Accesso come paziente

Effettuando L'accesso come paziente ci troveremo nella schermata Home del paziente dove possiamo trovare un check box che una volta premuto ci permetterà di aggiornare il nostro medico di base e segnalargli che abbiamo preso la medicina in quel giorno. Sotto al check box ci sono dei consigli utili su quando bisogna prendere la medicina, spesso in farmacia ci viene detto di prendere una certa medicina prima/dopo i pasti e può succedere che molte persone, specialmente in età avanzata abbiano problemi e dubbi su quando bisogna prendere la medicina, per questo motivo è stata inserita una sezione scrollabile che contiene informazioni su quando bisogna assumere la medicina. Come possiamo notare nella figura 4.11 nella parte inferiore del menù possiamo vedere che sono presenti diversi fragment fra i quali è possibile scegliere e sono (da sinistra verso destra): la Homepage, i Consigli, il Promemoria, il Contapassi e il Profilo.



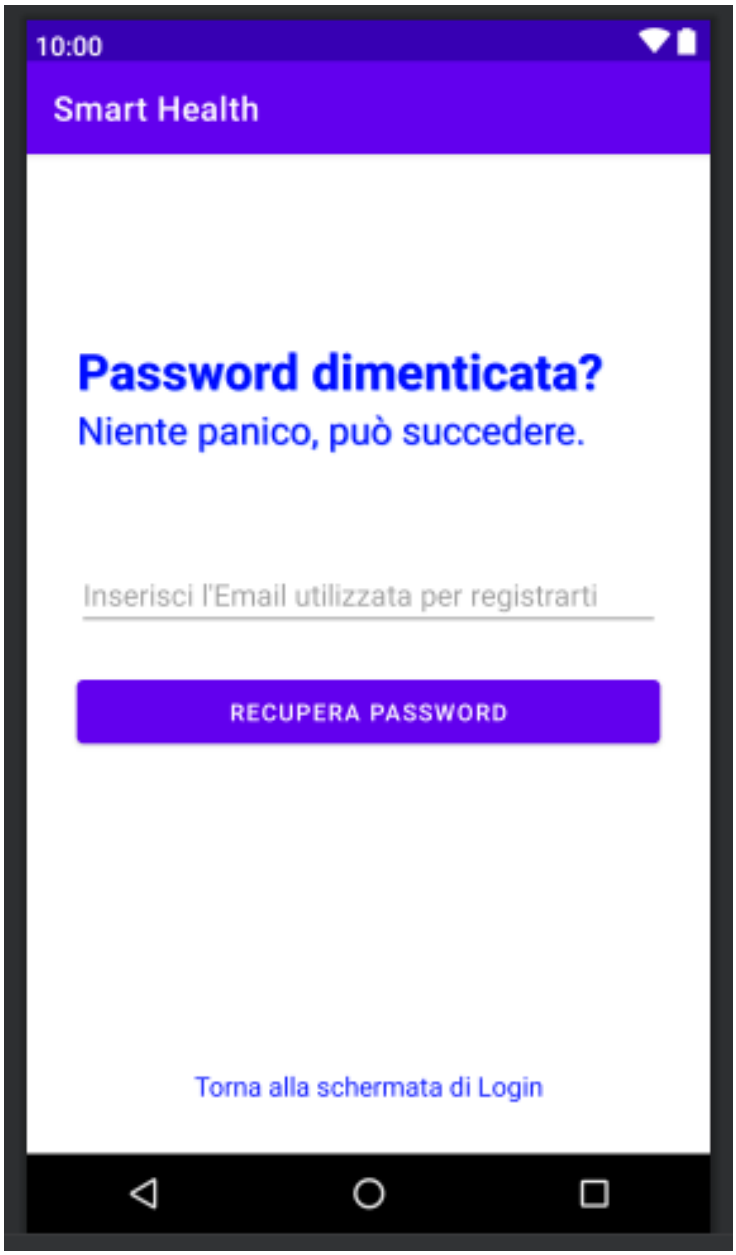
(fig. 4.11 Homepage del Paziente)

4.6.12 – Password dimenticata

Nella MainActivity se dovessimo cliccare su password dimenticata invece di accedere, verremmo portati alla schermata 4.12 dove sarà possibile recuperare la password automaticamente inserendo la nostra mail, l'applicazione poi controllerà se l'utente è un utente registrato oppure no, e se l'utente risultasse registrato FireBase invierà in automatico una mail di recupero tramite la quale l'utente potrà resettare la propria password, se per caso dovesse tornarci in mente la nostra vecchia password cliccando sulla scritta in fondo verremmo riportati alla schermata di login.

```
public class password_dimenticata extends AppCompatActivity {

    private EditText mforgotpassword;
    private Button mpasswordrecoverbutton;
    private TextView mgobacktologin;

    FirebaseAuth firebaseAuth;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_password_dimenticata);
        mforgotpassword=findViewById(R.id.forgotpassword);
        mpasswordrecoverbutton=findViewById(R.id.passwordrecoverbutton);
        mgobacktologin=findViewById(R.id.gobacktologin);

        firebaseAuth= FirebaseAuth.getInstance();


        mgobacktologin.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent=new Intent(password_dimenticata.this,MainActivity.class);
                startActivity(intent);
            }
        });

        mpasswordrecoverbutton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String mail=mforgotpassword.getText().toString().trim();
                //se l'utente non inserisce una mail
                if(mail.isEmpty())
                {
                    Toast.makeText(getApplicationContext(),"Inserisci la tua Email
perfavore",Toast.LENGTH_SHORT).show();
                }

//se l'utente inserisce una mail
                else
                {
                    firebaseAuth.sendPasswordResetEmail(mail).addOnCompleteListener(new
OnCompleteListener<Void>() {
                        @Override
                        public void onComplete(@NonNull Task<Void> task) {
                            //se la mail è corretta e l'utente è registrato
                            if (task.isSuccessful())
                            {
                                Toast.makeText(getApplicationContext(),"L'Email è stata
inviata controlla la casella postale",Toast.LENGTH_SHORT).show();
                                finish();
                                startActivity(new
Intent(password_dimenticata.this,MainActivity.class));
                            }
                            //se l'email è incorretta o l'utente non è registrato
                            else
                            {
                                Toast.makeText(getApplicationContext(),"L'Email non è
corretta, riprova perfavore.",Toast.LENGTH_SHORT).show();
                            }
```

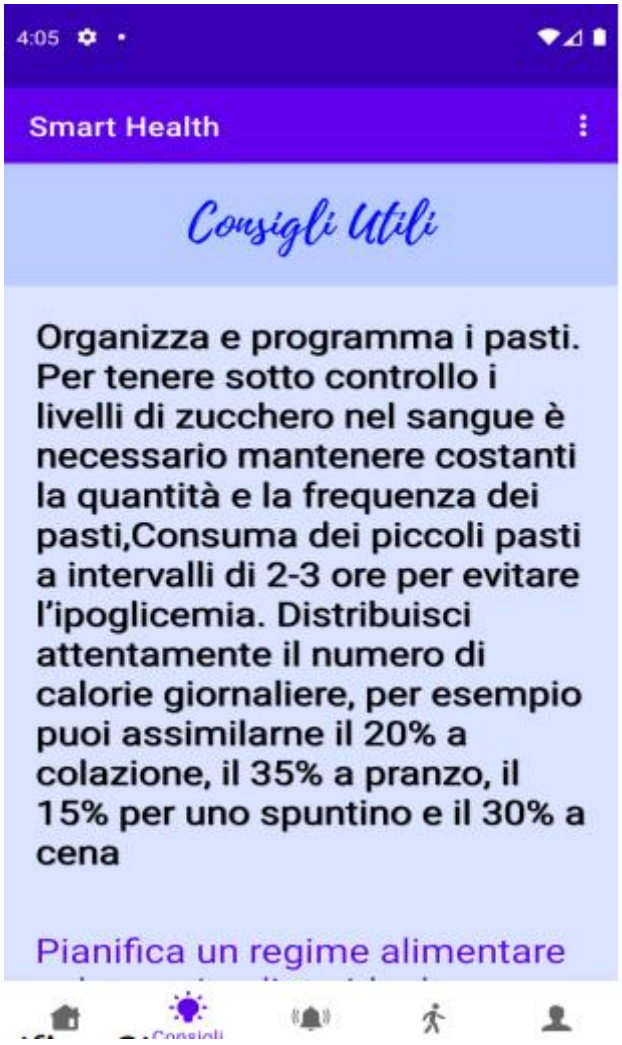
```
                    }
                });

            }
        }
    });


    }
}
```



(fig. 4.12 Password dimenticata)

4.6.13 – Consigli utili

Il Fragment Consigli Utili contiene una serie di consigli che possono essere utili per i pazienti diabetici, questa sezione contiene una raccolta di consigli e varie informazioni sulla tipologia di dieta da seguire e sull'attività fisica consigliata.

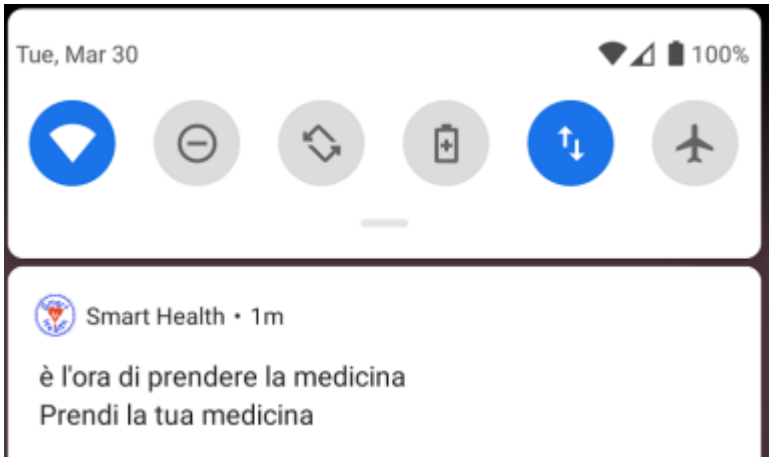


(fig. 4.13 Consigli utili)

4.6.14 – Promemoria

Il Fragment Promemoria (fig. 4.14) ci permette di selezionare l'orario nel quale vogliamo ricevere la notifica che ci ricorda di assumere la medicina. Una volta selezionato un orario riceveremo una notifica ogni giorno alla stessa ora a meno che non modifichiamo l'ora. La notifica che riceveremo è come quella mostrata nella figura (fig. 4.15).

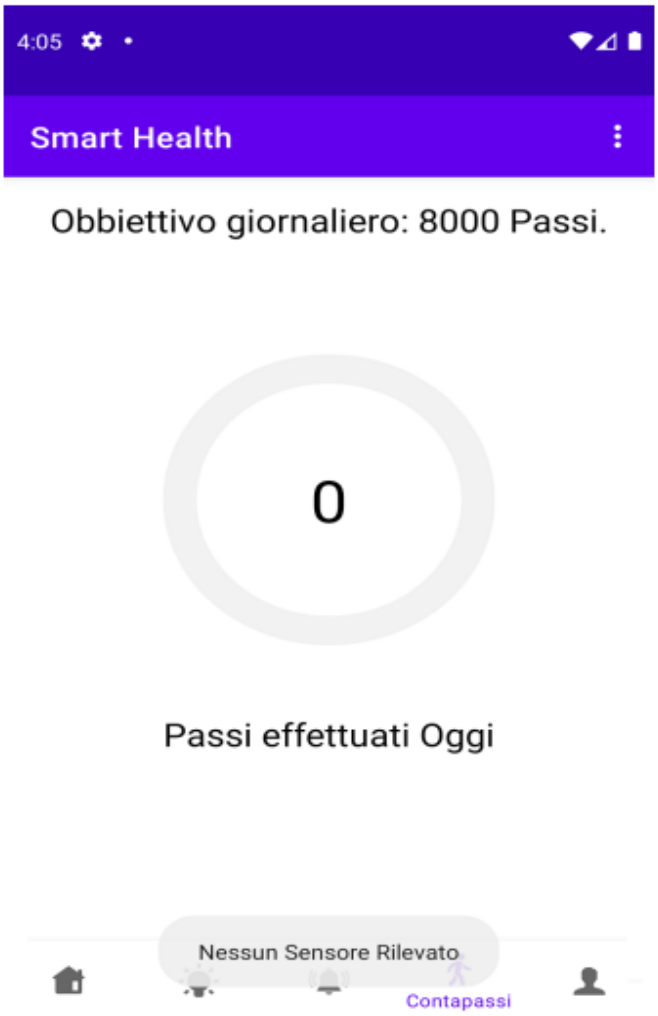
(fig. 4.14 imposta notifica)



(fig. 4.15 notifica)

4.6.15 – Contapassi

Nella sezione Contapassi possiamo trovare un piccolo contapassi che funziona utilizzando il sensore di movimento del nostro dispositivo. Il contapassi è stato implementato con l'obiettivo d'incentivare i pazienti che utilizzano l'applicazione a camminare di più per completare l'obiettivo giornaliero. Camminare ogni giorno aiuta a mantenere il diabete sotto controllo per questo motivo risulta una funzionalità utile per gli utenti. Nel caso in cui l'applicazione non trovi un sensore di movimento nel dispositivo utilizzato ci verrà segnalato con un errore come nell'immagine sottostante.



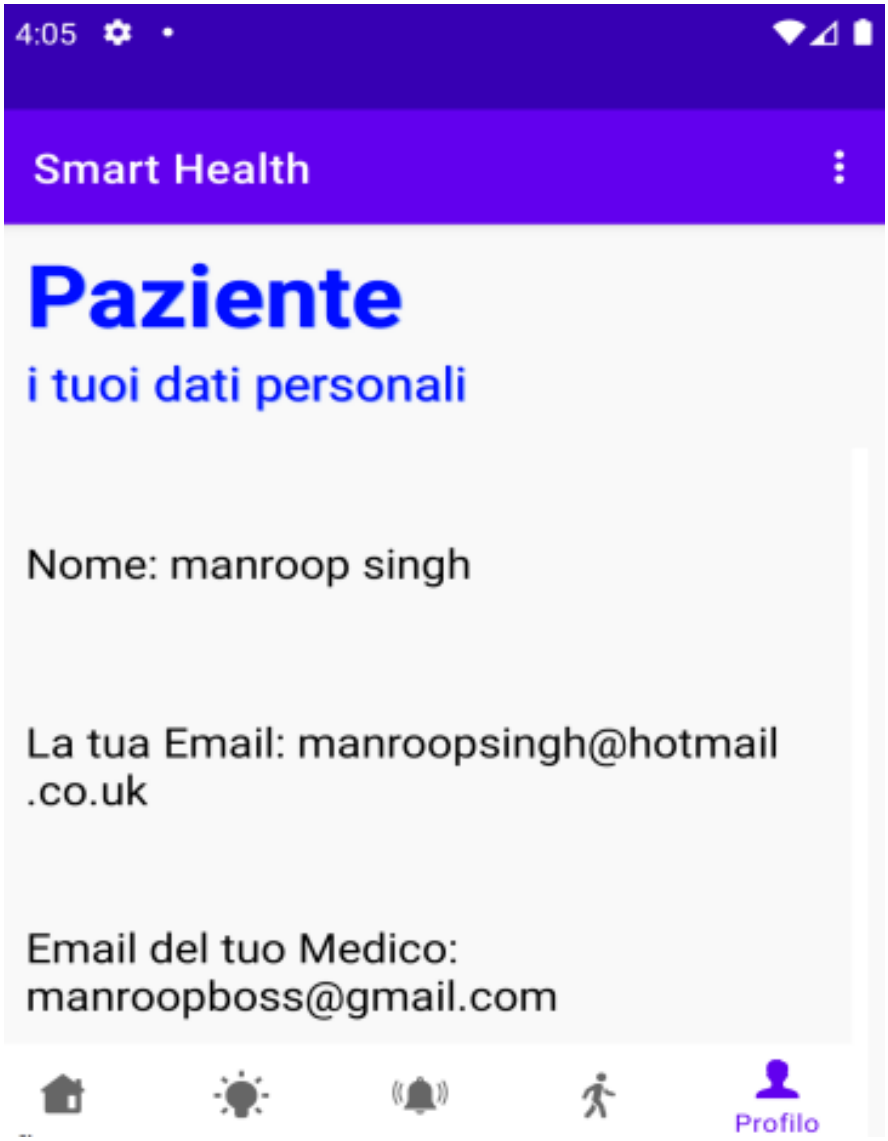
(fig. 4.16 Contapassi)

4.6.16 – Profilo utente

Nell'ultimo Fragment abbiamo il profilo dell'utente, questa sezione sarà uguale sia per il paziente che per il medico con l'unica differenza che il paziente potrà vedere l'indirizzo mail del medico ove disponibile, mentre il medico vedrà soltanto le sue informazioni in questa sezione. Per poter mostrare le informazioni dell'utente corrente l'applicazione effettua una richiesta a Firebase, sotto possiamo leggere il codice che ci permette di avere l'info dell'user.

```
firebaseAuth= FirebaseAuth.getInstance();
```

```
firebaseFirestore= FirebaseFirestore.getInstance();


FirebaseUser firebaseUser=firebaseAuth.getCurrentUser();



String userid;
userid=firebaseAuth.getCurrentUser().getUid();
```



(fig. 4.17 Profilo utente)

Capitolo 5

Conclusione

Questo percorso di autoapprendimento ha predisposto le fasi di sviluppo e di progettazione che mi hanno permesso di realizzare un'applicazione non solo utile ma anche unica nel suo genere. Uno degli obiettivi principali nel processo di sviluppo era quello di realizzare un'applicazione semplice ed intuitiva anche per chi non è molto pratico con la tecnologia. SmartHealth era originariamente un'applicazione destinata alle persone diabetiche ma alla fine è diventata un'applicazione utile anche per gli utenti non diabetici, infatti, SmartHealth ha varie funzionalità utilizzabili da chiunque ad esempio: impostare un promemoria per la medicina anche senza dover inviare una conferma al proprio medico, avere indicazioni utili sui medicinali e su uno stile di vita sano, oppure utilizzare il contapassi integrato all'interno dell'applicazione, quindi anche se l'applicazione era stata pensata per una determinata categoria di utenti, qualsiasi persona potrà impostare un promemoria per la medicina e inviare in tempo reale una conferma al proprio medico di base. In conclusione, possiamo dire che gli obiettivi fissati in fase di progettazione sono stati raggiunti, ma abbiamo ancora alcune funzionalità che possono essere migliorate, come la possibilità di cambiare l'obiettivo del contapassi oppure migliorare la praticità del promemoria in modo che se l'utente non ha assunto la medicina entro un'ora dal promemoria, venga inviata un'ulteriore notifica ricordandogli di assumerla. SmartHealth è un'applicazione con una struttura e un format adattabile a situazioni differenti ma con ampi margini di miglioramento in parecchi aspetti; infatti, in un aggiornamento futuro si potrebbero aggiungere funzionalità che erano state scartate per il momento come la possibilità di chattare con il proprio medico di base o l'implementazione di un calendario tenente traccia di tutti i giorni nei quali è stata presa la medicina e in quali no, questo aggiornamento potrebbe essere anche utile per rendere

l’applicazione accessibile a un numero maggiore di utenti rendendola ancora più pratica e funzionale.

Bibliografia

[1] “L’SDK e l’ambiente di sviluppo” :
https://www.html.it/pag/19497/i-software-necessari/

[2] “Introduzione: perché Android” :
https://www.html.it/pag/19496/introduzione-perch-android/

[3] “FireBase Documentation”:

https://firebase.google.com/docs/

[4] “Articolo Sql e NoSql a confronto”:
https://www.html.it/articoli/sql-e-nosql-adocumenti-il-confronto/