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 Malati di Alzheimer

Laureanda

Diletta Maniglia

Relatore

Prof. Riccardo Martoglia

Anno Accademico 2019/2020

PAROLE CHIAVE

Android studio

Alzheimer

Java

SQLite

Applicazione mobile

Sommario

Introduzione.....1

Parte I

Il caso di studio

Capitolo 1 - Analisi degli obiettivi

1.1 - Scopo del progetto.....4

1.2 - Applicazioni simili.....5

Capitolo 2 - Tecnologie utilizzate

2.1 - Perché Android.....7

2.2 - Introduzione ad Android Studio.....8

2.2.1 - Come nasce e si sviluppa Android Studio.....8

2.2.2 - Ultime innovazioni.....9
2.3 - Android SDK.....9
2.4 - Realizzazione applicazione su Android Studio.....10
2.5 - Com'è strutturato un progetto su Android Studio.....12
2.5.1 - Cartella manifests.....12
2.5.2 - Cartella res.....14
2.5.3 - Gradle.....14
2.6 - Le fondamenta di Android.....16
2.6.1 - Activity.....16
2.6.2 - Service, Content Provider, Broadcast Receiver.....18
2.6.3 - Intent.....19
2.6.3.1 - Gli extras.....20
2.7 - SQLite database.....20
2.8 - Come creare SQLite database.....21
2.8.1 - Classe DatabaseHelper.....21
2.8.2 - Classe DBAdapter.....22

Parte II

Progetto e sviluppo

Capitolo 3 - Progettazione

3.1 - Analisi dei requisiti.....26

3.2 - Requisiti funzionali.....27

3.2.1 - Caso d'uso.....28

3.2.2 - Diagramma delle attività.....30

3.3 - Progettazione database.....32

3.3.1 - Modello ER Ricordella.....32

3.3.2 - Entità tbl_reminder.....33

3.3.3 - Entità memories.....34

3.3.4 - Diagramma UML delle classi.....34

3.4 - SharedPreferences.....35

Capitolo 4 - Implementazione

4.1 - Introduzione all'implementazione.....37

4.2 - Struttura del progetto.....38

4.2.1 - Cartella main/java/com.example.ricordella....39

4.2.2 - Cartella main/res.....42

4.3 - Implementazione.....43

4.3.1 - AndroidManifest.....43

4.3.2 - SplashScreenActivity.....47

4.3.3 - MainActivity.....48

4.3.4 - info/MainActivityInfoMe.....54

4.3.5 - info/NewInfoMe.....56

4.3.6 - note/NoteEditorActivity.....58

4.3.7 - note/MainActivityNote.....59

4.3.8 - game/MainActivityMemory.....,,.....61

4.3.9 - polaroid/MainActivityPolaroid.....64

4.3.10 - polaroid/NewPolaoirdActivity.....66

4.3.11 - Altre classi cartella polaroid/.....67

4.3.12 - reminder/.....70

4.3.13 - reminder/AddActivityReminder.....72

Capitolo 5 - Conclusioni e sviluppi futuri

5.1 - Obiettivo raggiunto?.....76

Bibliografia.....78

Indice figure

Figura 1.1 : Applicazione ChatYourself

Figura 2.1 : Finestra Android Studio per selezionare un template per il progetto

Figura 2.2 : Finestra Android Studio per configurare il nuovo progetto

Figura 2.3 : Struttura di un progetto su Android Studio

Figura 2.4 : Struttura del file AndroidManifest.xml

Figura 2.5 : Struttura della cartella res

Figura 2.6 : Come lavora un build process

Figura 2.7 : Ciclo di vita di un'Activity in un'applicazione Android

Figura 2.8 : Codice per avviare una nuova Activity

Figura 2.9 : Codice per inserire una String nella componente Intent

Figura 2.10 : Codice per estrarre una String dalla componente Intent

Figura 2.11 : Sintassi classe DatabaseHelper

Figura 2.12 : Sintassi classe DBAdapter

Figura 2.13 : Continuazione classe DBAdapter

Figura 3.1 : Diagramma dei casi d'uso dell'applicazione Ricordella

Figura 3.2 : Leggenda diagramma delle attività

Figura 3.3 : Diagramma delle attività completo applicazione Ricordella

Figura 3.4 : Diagramma delle attività zoom prima parte

Figura 3.5 : Diagramma delle attività zoom seconda parte

Figura 3.6 : Modello ER Ricordella

Figura 3.7 : Entità promemoria

Figura 3.8 : Entità promemoria attributi

Figura 3.9 : Entità foto

Figura 3.10 : Entità foto attributi

Figura 3.11 : Diagramma delle classi Ricordella

Figura 4.1 : Cartella main, res e file XML AndroidManifest.xml

Figura 4.2 : Cartella main

Figura 4.3 : Cartella game

Figura 4.4 : Cartella info

Figura 4.5 : Cartella note

Figura 4.6 : Cartella polaroid

Figura 4.7 : Cartella reminder

Figura 4.8 : Cartella splashscreen

Figura 4.9 : Activity MainActivity

Figura 4.10 : Cartella res

Figura 4.11 : Schermata iniziale applicazione

Figura 4.12 : Codice SplashScreenActivity

Figura 4.13 : Layout MainActivity

Figura 4.14 : Parte di codice MainActivity

Figura 4.15 : Parte di codice MainActivity

Figura 4.16 : Funzione callNumber() MainActivity

Figura 4.17 : Funzione sendSms() MainActivity

Figura 4.18 : Esempio di chiamata al contatto di emergenza e invio del messaggio con posizione corrente

Figura 4.19 : Funzione getLastLocation() MainActivity

Figura 4.20 : Layout MainActivityInfoMe

Figura 4.21 : Pezzo di codice MainActivityInfoMe

Figura 4.22 : Layout NewInfoMe

Figura 4.23 : saveData() NewInfoMe

Figura 4.24 : Layout NoteEditotActivity

Figura 4.25 : onTextChanged() NoteEditorActivity

Figura 4.26 : Layout MainActivityNote

Figura 4.27 : Pezzo di codice MainActivityNote

Figura 4.28 : Layout MainActivityMemory

Figura 4.29 : Layout MainActivityMemory

Figura 4.30 : Pezzo di codice MainActivityMemory

Figura 4.31 : Pezzo di codice MainActivityMemory

Figura 4.32 : Layout MainActivityPolaroid

Figura 4.33 : Layout NewPolaroidActivity

Figura 4.34 : Funzioni NewPolaroidActivity

Figura 4.35 : Costruttore Polaroid

Figura 4.36 : PolaroidContract

Figura 4.37 : Funzioni PolaroidAdapter

Figura 4.38 : Definizione database PolaroidDbHelper

Figura 4.39 : Funzioni PolaroidDbHelper

Figura 4.40 : Layout MainActivityReminder

Figura 4.41 : Layout AddActivityReminder

Figura 4.42 : Funzione onSave() AddActivityReminder

Figura 4.43 : Funzioni AddActivityReminder

Figura 4.44 : AlarmReceiverReminder

Introduzione

L’immensa crescita inerente all’ambito informatico degli ultimi anni ha implicato lo sviluppo dal punto di vista tecnologico anche di altre discipline, una di queste riguarda la medicina in particolare la malattia dell’alzheimer. L'alzheimer presenta vari sintomi come il declino cognitivo, la graduale perdita di memoria, il senso di smarrimento e disorientamento, la tecnologia può assistere il paziente nel mantenimento della salute cognitiva, grazie all’apprendimento di cose nuove e connessioni sociali che potrebbero rallentare o impedire lo sviluppo della malattia.

La possibilità di poter rallentare questa patologia ha fatto nascere in me la voglia di ideare e sviluppare un’applicazione per aiutare i malati di alzheimer nelle prime fasi della malattia.

Il progetto Ricordella consiste nella creazione di un software con varie funzionalità, mirate ad aiutare l’utente a ricordare, esercitare la memoria e cercare aiuto in caso di emergenza.

Lo sviluppo dell'applicazione è stato implementato tramite l’utilizzo di Android Studio come IDE e SQLite come gestore dei dati utilizzati all'interno del software.

La tesi è strutturata in cinque capitoli nei quali vengono analizzate le fasi di studio delle tecnologie, l'analisi dei dei requisiti del sistema, la progettazione e l'implementazione dell'applicazione.

Il primo capitolo introduce la nascita del progetto, la sua struttura generale ed uno studio iniziale dei suoi obiettivi.

Il secondo capitolo illustra le tecnologie utilizzate per la creazione dell'applicazione Ricordella e le motivazioni di tali scelte.

Il terzo e quarto capitolo analizzano le fasi di progettazione e di implementazione del software, approfondendo l'analisi dei casi d'uso e delle funzionalità di base dell'applicazione, e descrivendo la realizzazione effettiva del progetto studiandone i vari elementi che lo compongono.

Infine nel quinto capitolo è presente una breve riflessione sul lavoro svolto e sull'utilità del software realizzato, concludendo con un appunto su possibili miglioramenti e sviluppi futuri che potranno essere conseguiti.

Parte I

Il caso di studio

Capitolo 1

Analisi degli obiettivi

1.1 - Scopo del progetto

Un software può avere varie peculiarità, per la realizzazione di un'applicazione per aiutare a ricordare ed esercitare la memoria e da utilizzare in situazioni di emergenza, destinata in particolare agli utenti affetti dalla malattia dell'alzheimer, la facilità d'uso e l'intuitività sono caratteristiche fondamentali, realizzando pagine pulite, semplici, facili da leggere e da utilizzare. Quindi uno degli obiettivi principali è stato la creazione di un'interfaccia user-friendly.

Una scelta importante consiste nella creazione di un software con varie funzionalità, per venire incontro alle possibili necessità del paziente, quindi creare una "memoria di riserva" dove poter visualizzare le informazioni personali dell'utente, la sua patologia, il suo medico di base e il numero per poterlo contattare, possibilità di scrivere pensieri o annotazioni, impostare dei promemoria con annessa notifica alla data e l'ora desiderata, caricare e visualizzare foto ricordo con didascalia, giocare a memory per allenare la memoria, poter effettuare una chiamata automatica al numero di emergenza o al

contatto di emergenza preimpostato con invio automatico del messaggio con annessa posizione al contatto stesso.

1.2 - Applicazioni simili

Facendo una ricerca su internet, non ci sono sono molte applicazioni simili. Le poche applicazioni dedicate a questa malattia, sono principalmente per gli utenti che devono prendersi cura della persona affetta da demenza. Una sola applicazione è dedicata all'utente paziente, Chat Yourself [1], che è in grado di memorizzare le informazioni utili relative alla vita di una persona, restituendole su richiesta all'utente. Inizialmente questa applicazione chiede all'utente di creare un profilo e di inserire i suoi dati personali, ad esempio: il nome dei familiari, delle persone di fiducia, l'indirizzo dei luoghi che frequenta, i recapiti telefonici, le abitudini alimentari, le ricorrenze e gli appuntamenti settimanali, gli orari in cui deve prendere le medicine. Successivamente quando l'utente ha bisogno di un'informazione, ciò che deve fare è aprire la chat e fare una domanda con ciò che in quel momento gli serve, ad esempio "Mi sono perso, dove abito?" e l'applicazione recupera le informazioni immesse precedentemente dall'utente in fase di registrazione, rispondendo così in modo immediato con indirizzo e mappa per tornare a casa.

Figura 1.1 : Applicazione ChatYourself

Capitolo 2

Tecnologie utilizzate

2.1 - Perché Android

Android è il principale sistema operativo per il mobile al mondo. Stando alle statistiche, anima circa tre dispositivi su quattro lasciando lo spazio restante principalmente ad iOS di Apple [2]. Esso ha avuto un forte riscontro con il pubblico per le fasce di prezzo dei dispositivi android che vanno da poche decine di euro a cifre gradualmente più alte. La diffusione di Android tra i programmatori invece è dovuta alla sua architettura di sistema e alle sue radice salde nel mondo open source. Inoltre esso mette a disposizione molto materiale e documentazione online, offre numerosi strumenti del tutto gratuiti ed è semplice da utilizzare per lo sviluppo dell'interfaccia grafica dell'applicazione. Queste

sono le motivazioni prioritarie che mi hanno portato alla scelta di sviluppare un'applicazione per questo sistema operativo.

2.2 - Introduzione ad Android Studio

Il modo migliore per sviluppare un'applicazione Android è utilizzare strumenti adeguati. Un ambiente di lavoro efficiente, intuitivo e multipiattaforma è Android Studio. In questo modo si può accedere all'ambiente Android SDK e lavorare al codice utilizzando tutte funzionalità dell'SDK, compreso creare interfacce utente e testare le applicazioni, quindi un ambiente contrassegnato dalla flessibilità e dalla praticità.

2.2.1 - Come nasce e si sviluppa Android Studio

Android Studio basato sul software di JetBrains IntelliJ IDEA, un IDE chiaro e facilmente comprensibile. Esso è stato progettato specificamente per lo sviluppo di applicazioni Android, questo ha fatto in modo che sostituisse gli Android Development Tools (ADT) di Eclipse, diventando l'IDE primario di Google per lo sviluppo nativo di applicazioni Android. Inoltre è un ambiente di sviluppo integrato (IDE) molto portabile quindi disponibile su Windows, Mac OS X e Linux. Il progetto di Android Studio è costantemente aggiornato, affinché sia sempre al passo con lo sviluppo globale e le innovazioni della piattaforma Android.

2.2.2 - Ultime innovazioni

Nella versione 1.3 del software attraverso un ricongiungimento tra SDK (per lo sviluppo in java) e NDK (Native Development Kit, C/C++) è stata aggiunta la possibilità di editing e debugging.
Successivamente ci sono stati miglioramenti che basati sull'aumento della produttività e la riduzione dei tempi di sviluppo, questo grazie alla sviluppo dell'Instant Run, utile per visualizzare velocemente, senza aspettare lunghi rebuild del progetto, le modifiche apportate al codice, e anche per potenziare gli emulatori che in particolare su macchine meno dotate provocano spesso un forte rallentamento del sistema, quindi rendendo testing del codice e la sua esecuzione più fluide.

2.3 - Android SDK

I l Software Development Kit (SDK) di Android include una serie completa di strumenti di sviluppo. Tra questi sono presenti: un debugger, librerie, un emulatore portatile basato su QEMU, documentazione, codici esemplificativi e tutorial. Questo pacchetto permette all'utente di accedere al codice sorgente di qualsiasi versione di Android e la possibilità di modificarlo in modo autonomo.
La composizione degli strumenti di sviluppo viene gestita tramite il programma Android SDK Manager, disponibile sia su Eclipse sia su Android Studio tramite il menù Tools. Questo programma permette quindi al programmatore di utilizzare le piattaforme e gli strumenti contenuti nel SDK nel modo conforme al suo lavoro.

2.4 - Realizzazione applicazione su Android Studio

Android Studio guida l'utente nei vari passaggi necessari per la realizzazione della nuova applicazione. Il primo passaggio consiste nel selezionare la tipologia di applicazione e il contesto applicativo, effettuando una scelta tra Phone, Tablet, Wear OS, etc... Successivamente è possibile scegliere un template di Activity per lo sviluppo del progetto, in base al dispositivo scelto.

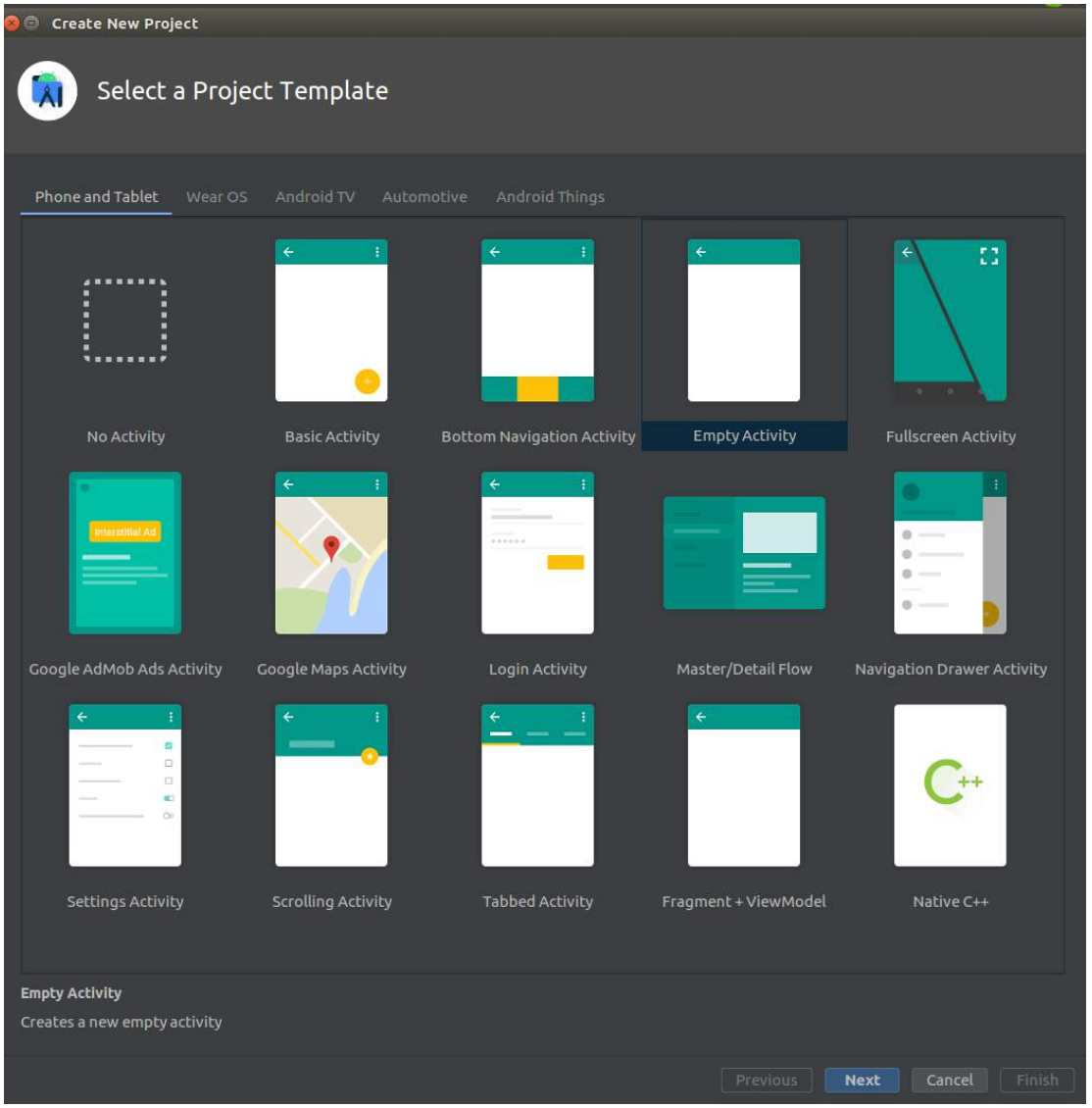

Figura 2.1 : Finestra Android Studio per selezionare un template per il progetto

Il terzo passaggio, invece, consiste nel scegliere il nome, il Package name, ovvero il nome viene utilizzato per identificare in modo univoco la tua app, la location in cui salvare il progetto, e il linguaggio da utilizzare, Kotlin o Java. La mia scelta del linguaggio è ricaduta su Java in quanto è un linguaggio che avevo imparato ad utilizzare in passato, inoltre essendo più utilizzato rispetto al Kotlin, si possono video tutorial, e risolvere bug e problemi nel codice, più facilmente grazie al supporto della comunità.

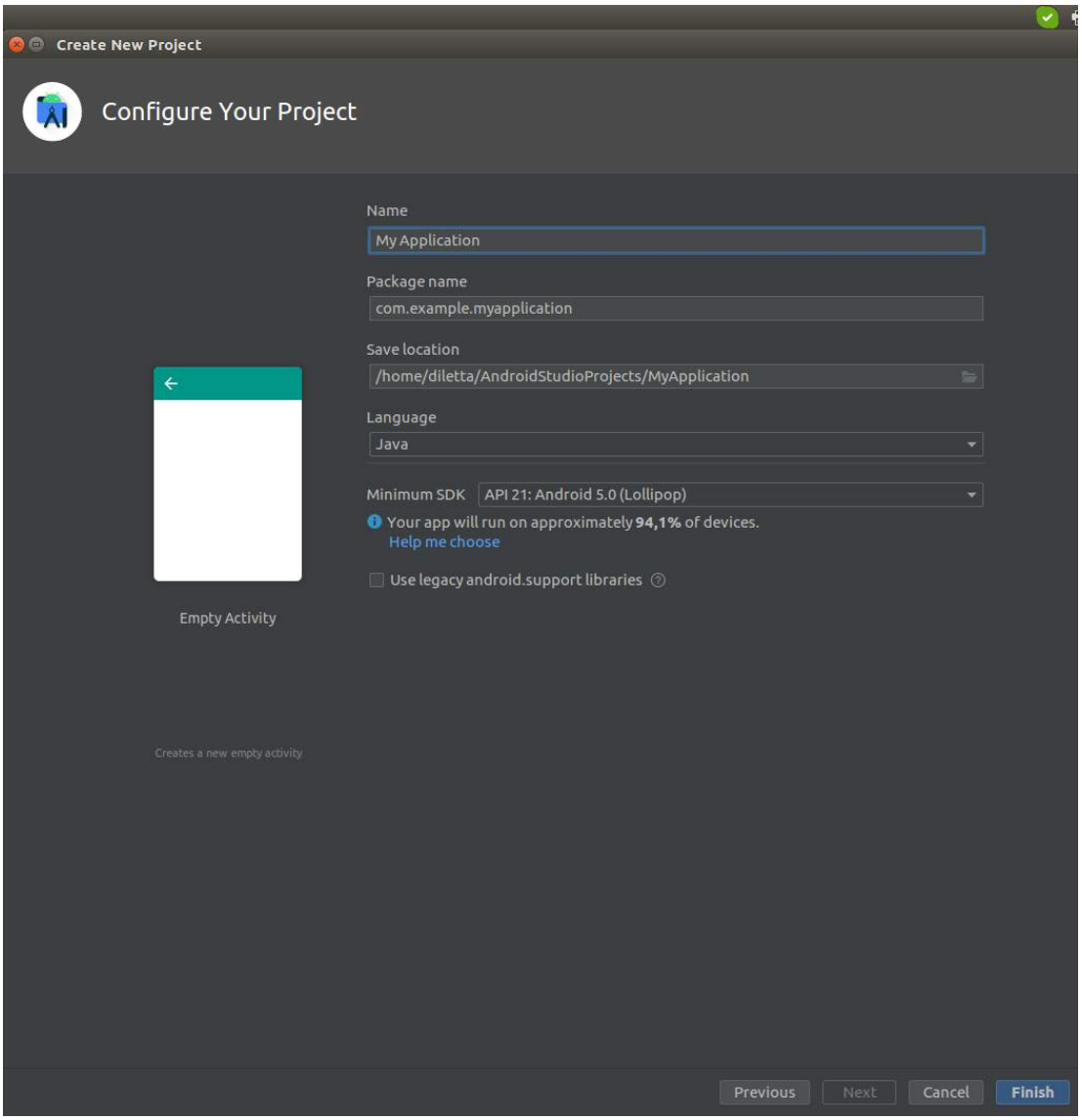

Figura 2.2 : Finestra Android Studio per configurare il nuovo progetto

2.5 - Com'è strutturato un progetto su Android Studio

Il progetto è formato principalmente da tre elementi :

- la cartella manifests, contenente il file *AndroidManifest.xml* che è un file XML che contiene informazioni dettagliate riguardo all'applicazione
- la cartella java, contenente il codice java
- la cartella res

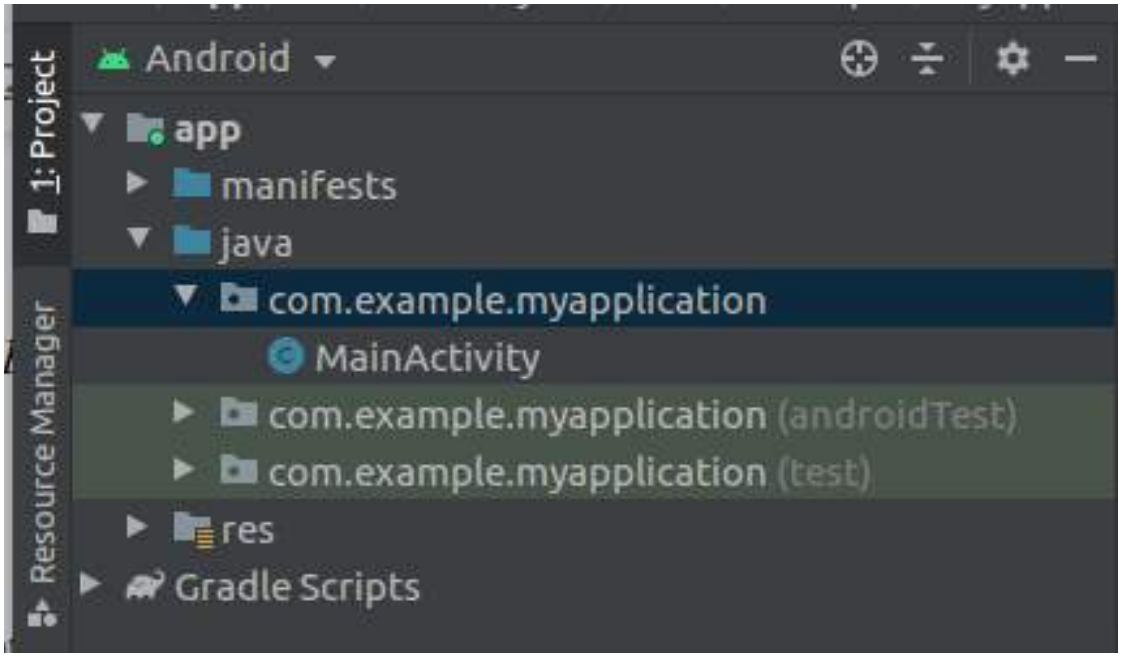

Figura 2.3 : Struttura di un progetto su Android Studio

2.5.1 - Cartella manifests

La cartella manifests include il file *AndroidManifest.xml* raccoglie informazioni basilari sull'app, informazioni necessarie al sistema per far girare qualsiasi porzione di codice

della stessa. Esso presenta varie funzionalità come dare un nome al package Java dell'applicazione, descrivere le componenti dell'applicazione (Activity, Service, Receiver, Provider, etc...), nominando la classi e constatando le loro competenze. Esso dichiara anche i permessi dell'applicazione e i permessi necessari alle altre applicazioni per interagire con la stessa. Inoltre asserisce il livello minimo di API Android richiesto dall'applicazione ed elenca le librerie necessarie ad essa.

Importante è l'elemento intent-filter il quale viene associato ad una sola Activity che sarà avviata nel momento in cui l'applicazione sarà aperta utilizzando il launcher del dispositivo.

AndroidManifest.xml

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Figura 2.4 : Struttura del file AndroidManifest.xml

2.5.2 - Cartella res

La cartella res contiene i dati esterni all'applicazione Android e i file relativi alle interfacce grafiche delle Activity. Le cartelle principali nelle quali vengono organizzati risorse e file sono :

- layout, raggruppa i file XML utilizzati per costruire un'interfaccia utente, quindi l'architettura ed il design dell'applicazione
- values, raggruppa i dati come stringhe, interi, array, colori, etc... che il programmatore definisce tramite specifici tag XML e che saranno utilizzati direttamente nel programma
- drawable, contiene le immagini utilizzate dal programma, in vari formati

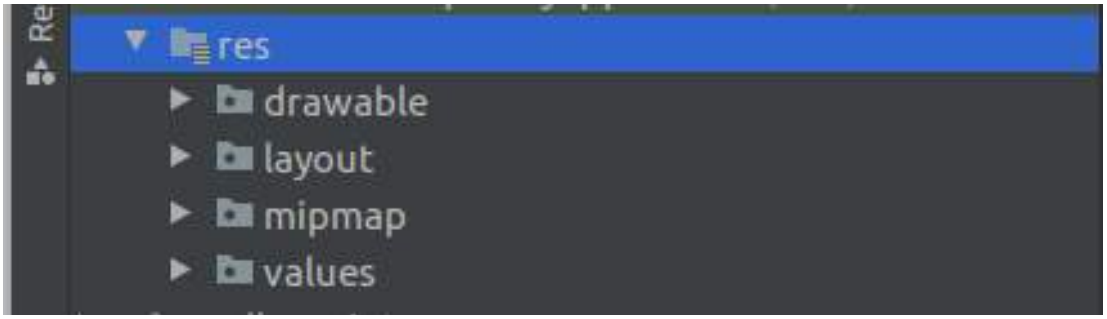

Figura 2.5 : Struttura della cartella res

2.5.2 - Gradle

Gradle è un build system, controlla e supervisiona l'operazione di compilazione, collegamento di file, esecuzione di casi di test e infine raggruppamento del codice in un file apk per l'applicazione Android.

Ci sono due file build.gradle per ogni progetto di studio Android di cui, uno è per l'applicazione, in cui il programmatore può definire configurazioni personalizzate per l'implementazione dell'applicazione, e l'altro è per i file di build a livello di progetto (livello modulo). Una sezione importante all'interno rispettivamente del primo file build gradle è quella delle *dependencies* per espandere le funzionalità del progetto, includendo librerie o file binari.

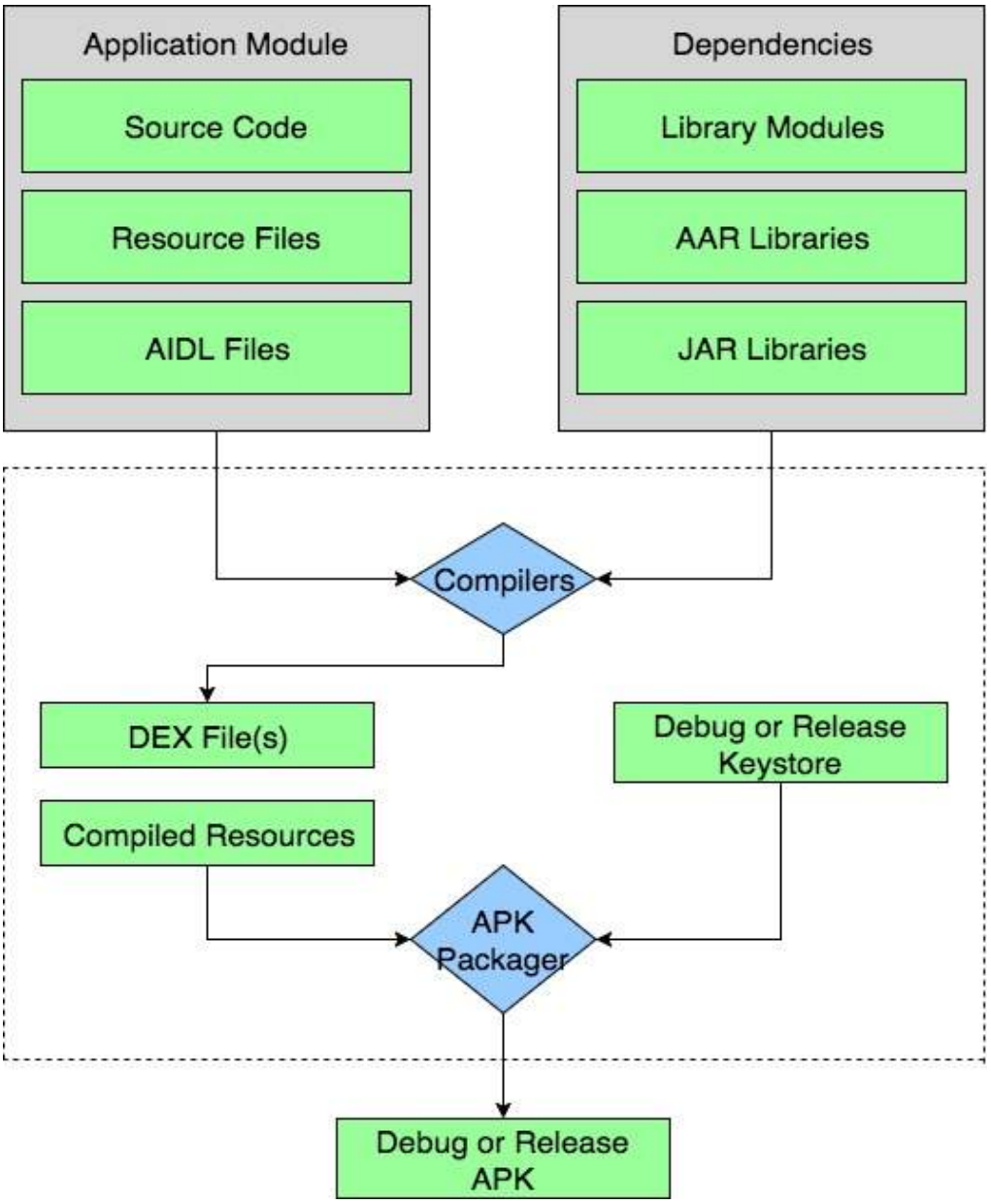

Figura 2.6 : Come lavora un build process

Nel processo di compilazione, il compilatore prende il codice sorgente, le risorse, i file JAR delle librerie esterne e AndroidManifest.xml e li converte in file .dex (Dalvik Executable files), che include bytecode. Il bytecode è supportato da tutti i dispositivi Android per eseguire l'applicazione. Quindi APK Manager combina i file .dex e tutte le altre risorse in un unico file apk.

2.6 - Le fondamenta di Android

Le applicazioni Android, a prescindere dalla loro finalità, affidano le loro funzionalità a quattro elementi principali, ovvero Activity, Service, Intent, Content Provider e Broadcast Receiver, essi permettono alle applicazioni di integrarsi alla perfezione nell'ecosistema Android.

2.6.1 - Activity

Un'Activity è un'interfaccia utente. Essa la possiamo definire come una "pagina" all'interno dell'applicazione con la quale l'utente interagisce. Le applicazioni sono generalmente formate da più pagine, quindi nel loro insieme le Activity costituiscono il flusso in cui l'utente entra per poter sfruttare al meglio le funzionalità messe a disposizione.
Di conseguenza è importante realizzare una buona interfaccia quanto una corretta progettazione della navigazione tra di esse, in modo tale da scorrere i contenuti in maniera coerente.

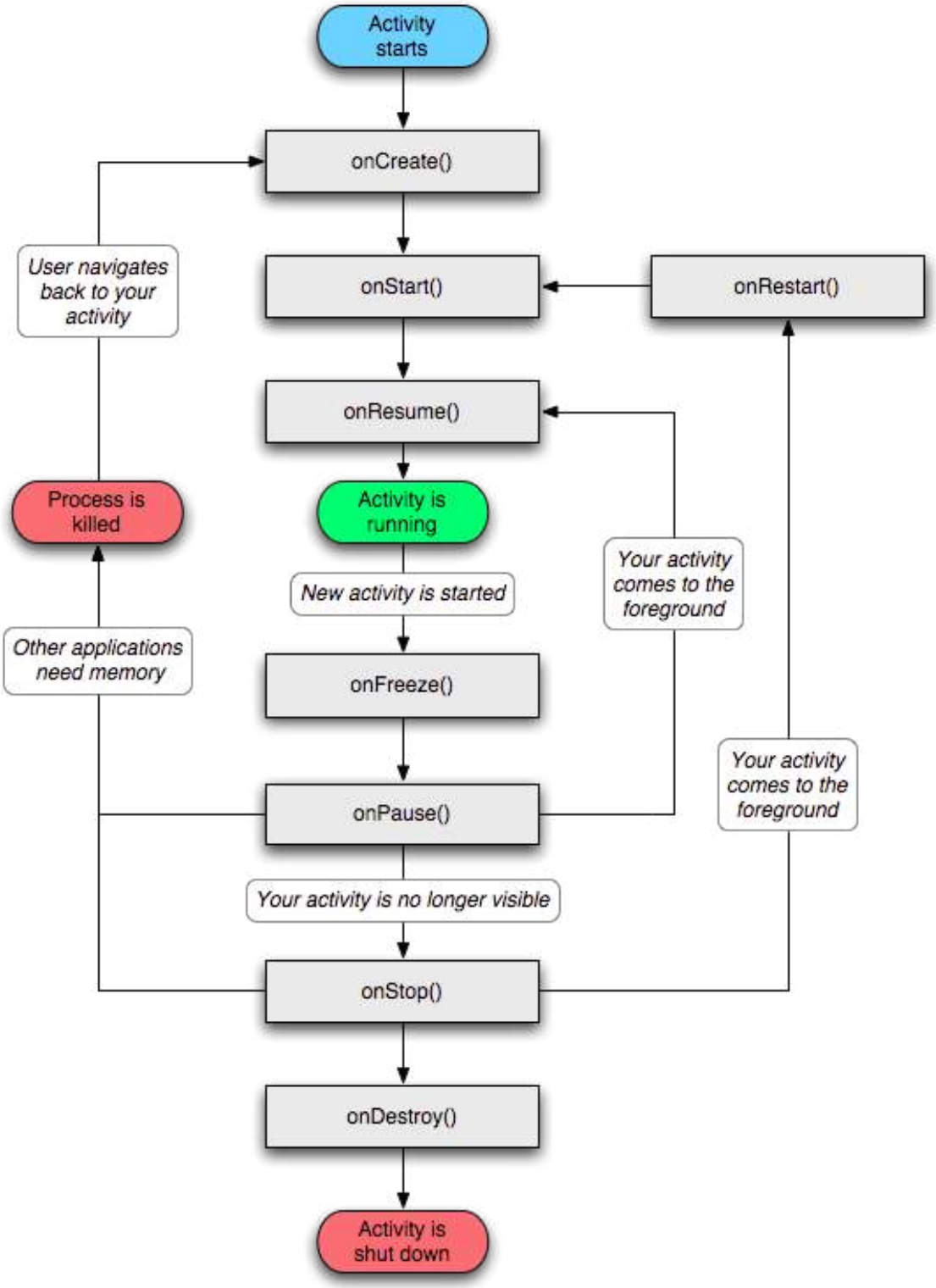

Figura 2.7 : Ciclo di vita di un'Activity in un'applicazione Android

Nel momento in cui un'Activity viene avviata,vengono invocati i metodi onCreate() per configurare l'interfaccia al primo avvio di un'Activity, onStart() attivare le funzionalità per rendere l'Activity visibile all'utente e
onResume() per avviare l'interazione con l'utente.
I metodo restanti come onPause() si verifica quando la corrente Activity viene messa in pausa ed una precedente activity viene riattivata, onStop() si verifica quando un'Activity non è più visibile all'utente, onDestroy() si verifica prima che l'Activity venga distrutta dal sistema, mentre il metodo onRestart() si verifica quando l'activity viene arrestata e riavviata.

2.6.2 - Service, Content Provider, Broadcast Receiver

Un Service svolge un lavoro in background senza bisogno di interagire in modi diretto con l'utente. Un esempio in cui viene utilizzato è quando un'applicazione permette di avviare un audio senza interromperlo anche se la sua interfaccia viene chiusa.
Un Content Provider è un componente che permette di condividere dati tra applicazioni nell'ambito del sistema, come nozioni presenti in un database, in un file o reperibili tramite accesso in rete. Questi contenuti potranno essere utilizzati dalle altre applicazioni senza rischiare di irrompere nello spazio di memoria.
Un Broadcast Receiver è un elemento che reagisce ad un invio di messaggi a livello di sistema, in broadcast, con i quali Android notifica l'accadimento di uno specifico evento, come l'arrivo di una chiamata o un messaggio.

2.6.3 - Intent

Un intent può essere definito come un valido strumento di comunicazione gestita dal sistema operativo Android, con cui una componente può richiedere l'esecuzione di un'azione da parte di un'altra componente.

I tre utilizzi principali di questa componente sono :

- avviare un'Activity

```
Intent newPage = new Intent( packageContext: this, NewActivity.class);
startActivity(newPage);
```

Figura 2.8 : Codice per avviare una nuova Activity

- avviare un Service
- inviare un messaggio in broadcast il quale può essere ricevuto dalle altre applicazioni

Gli intent possono essere di due tipi, espliciti o impliciti. In quelli espliciti viene dichiarata la componente da attivare, utilizzati soprattutto nelle aperture delle Activity, mentre quelli impliciti indicano quale componente attivare ma quale azione svolgere, essi vengono invocati principalmente quando avviene l'apertura di una finestra di dialogo che domanda all'utente quale applicazione bisogna aprire per completare l'azione.

2.6.3.1 - Gli extras

Una particolarità della componente Intent è che nel recapitare il messaggio possono custodire dei dati che poi possono essere estratti e letti dal destinatario.

```
Intent newPage = new Intent( packageContext: this, NewActivity.class);
newPage.putExtra( name: "username", username);
startActivity(newPage);
```

Figura 2.9 : Codice per inserire una String nella componente Intent

```
Intent intent = getIntent();
String username = intent.getStringExtra( name: "username");
```

Figura 2.10 : Codice per estrarre una String dalla componente Intent

2.7 - SQLite database

SQLite è una libreria software open source, utilizzato per eseguire operazioni di database su dispositivi Android come l'archiviazione, la manipolazione o il recupero di dati persistenti dal database. È incorporato in Android per impostazione predefinita. Pertanto, non è necessario eseguire alcuna configurazione del database o attività di amministrazione.

2.8 - Come creare SQLite database

Il primo passo per procedere alla creazione di un SQLite database consiste nel preparare uno script SQL in modo tale da creare la struttura interna del database. Successivamente è necessario creare una classe Java che estenda SQLiteOpenHelper, per gestire la creazione e l'aggiornamento del database sulla memoria fisica, e utilizzare un riferimento al database per accedere ai dati. Infine è necessario creare una classe per interagire con il database.

2.8.1 - Classe DatabaseHelper

Per la gestione di tutte le operazioni relative al database è stata assegnata una classe helper che estende SQLiteOpenHelper. Gestisce la creazione e l'aggiornamento del database, fornendo metodi intuitivi e flessibili per inserire, eliminare modificare i record del database. Nella classe DBHelper si possono memorizzare anche le costanti statiche del database, come il nome della tabella.

Nella figura 2.11 sottostante, il metodo onCreate() viene chiamato durante la creazione del database, esegue lo script per creare la base di dati SQL, mentre il metodo onUpgrade() viene chiamato durante l'upgrade del database, ad esempio in questo codice effettua il DROP, ovvero l'eliminazione, della tabella esistente e richiamando il metodo onCreate() la sostituisce con una nuova definizione della stessa.

```java
public class DatabaseHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "mydatabase.db";
    private static final int DATABASE_VERSION = 1;

    private static final String DATABASE_CREATE = "create table contact ( _id integer primary key autoincrement, " +
            "name text not null, surname text not null, sex text not null, birth_date text not null);";


    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, factory: null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase database) {
        database.execSQL(DATABASE_CREATE);
    }

    @Override
    public void onUpgrade( SQLiteDatabase database, int oldVersion, int newVersion ) {
        database.execSQL("DROP TABLE IF EXISTS contact");
        onCreate(database);
    }
}
```

Figura 2.11 : Sintassi classe DatabaseHelper

2.8.2 - Classe DBAdapter

La classe DBAdapter fornisce l'interfaccia diretta per maneggiare il database.

```java
    public DbAdapter(Context context) {
        this.context = context;
    }
    public DbAdapter open() throws SQLException {
        dbHelper = new DatabaseHelper(context);
        database = dbHelper.getWritableDatabase();
        return this;
    }
    public void close() {
        dbHelper.close();
    }
    private ContentValues createContentValues(String name, String surname, String sex, String birth_date ) {
        ContentValues values = new ContentValues();

        values.put( KEY_NAME, name );
        values.put( KEY_SURNAME, surname );
        values.put( KEY_SEX, sex );
        values.put( KEY_BIRTH_DATE, birth_date );
        return values;
    }
```

Figura 2.12 : Sintassi classe DBAdapter

Nel codice quando definiamo il costruttore è necessario configurare sempre il *context* (classe astratta che permette di accedere alla risorse specifiche dell'applicazione, come activity, intent), da passare nel momento in cui si vuole istanziare un nuovo adapter per eseguire delle query.

I metodi open() e close() servono per comunicare con il database e poterci lavorare sopra. Il primo fornisce l'interfaccia di creazione/aggiornamento/gestione del database, richiama il metodo getWritetableDatabase() di SQLiteOpenHelper che restituisce un oggetto in modalità lettura/scrittura attivo fino a quando non viene invocato il metodo close() che richiama il metodo close() della classe SQLiteOpenHelper, in modo tale da chiudere ogni oggetto database aperto.

Il metodo createContentValues() si occupa di memorizzare una serie di valori che possiamo utilizzare nelle nostre query e successivamente verranno processati dal ContentResolver, che serve per comunicare con il ContentProvider che gestisce l'accesso ad un insieme di dati strutturati.

```
//create a contact
public long createContact(String name, String surname, String sex, String birth_date ) {
    ContentValues initialValues = createContentValues(name, surname, sex, birth_date);
    return database.insertOrThrow(DATABASE_TABLE,  nullColumnHack: null, initialValues);
}
//update a contact
public boolean updateContact( long contactID, String name, String surname, String sex, String birth_date ) {
    ContentValues updateValues = createContentValues(name, surname, sex, birth_date);
    return database.update(DATABASE_TABLE, updateValues,  whereClause: KEY_CONTACTID + "=" + contactID,  whereArgs: null) > 0;
}
//delete a contact
public boolean deleteContact(long contactID) {
    return database.delete(DATABASE_TABLE,  whereClause: KEY_CONTACTID + "=" + contactID,  whereArgs: null) > 0;
}
```

Figura 2.13 : Continuazione sintassi classe DBAdapter

Per esempio per programmare un'applicazione Rubrica viene creato un contatto con attributi name, surname, sex, birth_date. Quindi quando viene creato un nuovo contatto nel metodo createContact(String name, String surname, String sex, String

birth_date), devono essere inseriti i quattro parametri necessari, ed una volta creato esso verrà associato ad un identificatore univoco, definito da una chiave primaria auto incrementale (_id) nello script per la creazione della tabella nella classe DatabaseHelper. Successivamente tramite la funzione insertOrThrow() il record viene inserito nel database, come parametri bisogna immettere il nome della tabella in cui inserire la riga, un valore che indica come comportarsi nel caso in cui i valori iniziali siano vuoti ed infine i valori da inserire.

Parte II

Progetto e sviluppo

Capitolo 3

Progettazione

3.1 - Analisi dei requisiti

La fase di analisi dei requisiti è necessaria e presente sostanzialmente in ogni modello di ciclo di vita del software. Rappresenta un'attività preliminare per procedere allo sviluppo di un sistema software. Lo scopo principale è definire le funzionalità che, in questo caso, la nuova applicazione deve offrire, cioè i requisiti che il software da sviluppare deve soddisfare. Quindi specificare cosa deve fare e quali sono gli obiettivi dell'applicazione, indipendentemente dal linguaggio di programmazione, l'architettura e la tecnologia utilizzata.

3.2 - Requisiti funzionali

Il primo passo per iniziare è stato pensare ed esaminare quali erano gli obiettivi che l'applicazione doveva raggiungere. Le operazioni fondamentali sono :

- **Inserimento, modifica informazioni personali:** l'applicazione permette all'utente di immettere le proprie informazioni personali da poter utilizzare in caso di necessità e di poterle in seguito modificare

- **Creazione, modifica, eliminazione note:** possibilità di creare una nota, poterla salvare per visualizzarla successivamente in un elenco, modificarla ed eliminarla

- **Memory:** possibilità di poter allenare la memoria, giocando al gioco memory, che richiede concentrazione e memoria, in cui i giocatori devono accoppiare le carte

- **Creazione, modifica, eliminazione promemoria:** possibilità di creare un promemoria e impostare il ricevimento di una notifica ad un determinato orario e una determinata data, anche di modificarlo ed eliminarlo

- **Caricamento, visualizzazione, eliminazione foto:** possibilità di caricare delle foto con annessa descrizione (opzionale), visualizzarle in stile polaroid, e eliminarle

- **Chiamata di emergenza:** possibilità di fare una chiamata automatica al numero di emergenza 118

- **SOS:** premendo su un unico ImageButton, possibilità di fare una chiamata di emergenza automatica al numero di emergenza immesso nelle informazioni personali, e invio automatico allo stesso numero di un messaggio di testo contenente un messaggio di aiuto con la posizione corrente dell'utente visualizzata in latitudine e longitudine

3.2.1 - Caso d'uso

Il diagramma dei casi d'uso [3] esprime un comportamento, desiderato o offerto. Individua chi o che cosa ha a che fare con il sistema (attore) e che cosa l'attore può fare (**caso d'uso**). Tipicamente è il primo tipo di diagramma ad essere creato in un processo o ciclo di sviluppo, nell'ambito dell'analisi dei requisiti. In questo modo è possibile ottenere una visione ad alto livello delle attività che l'applicazione è in grado di svolgere quindi si giunge ad una descrizione degli scenari di utilizzo dell'applicazione da parte degli attori che la utilizzano.

Nella mia applicazione Ricordella l'attore è rappresentato dall'utente, ovvero particolarmente ad un paziente affetto dalla malattia dell'Alzheimer, invece i casi d'uso coincidono con le operazioni elencate nel paragrafo precedente.

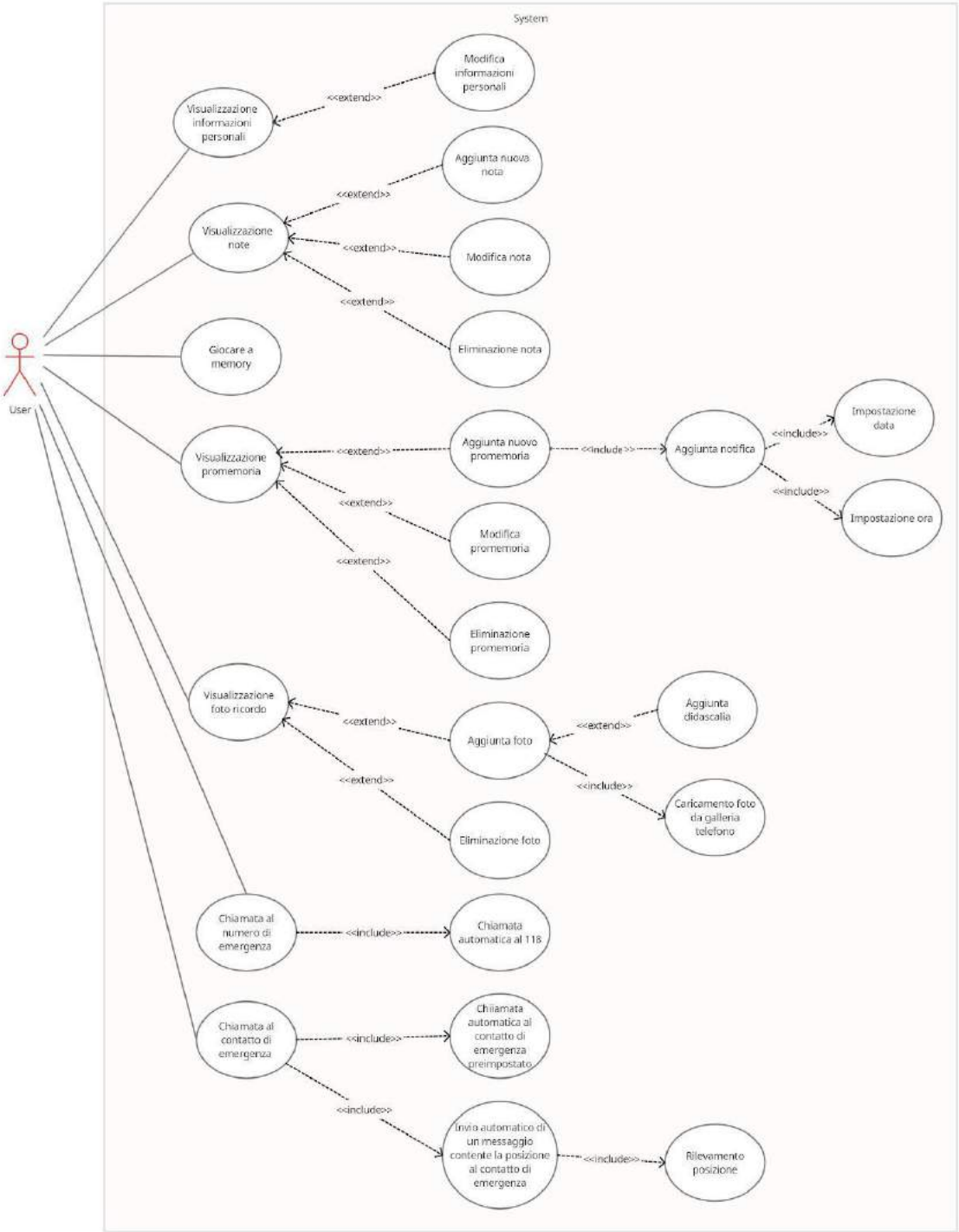

Figura 3.1 : Diagramma dei casi d'uso dell'applicazione Ricordella

3.2.2 - Diagramma delle attività

Un diagramma delle attività UML aiuta a visualizzare un certo caso d'uso ad un livello più dettagliato. È un diagramma comportamentale che illustra il flusso delle attività attraverso un sistema.

UML ha specificato un insieme di simboli e regole per i diagrammi di attività di disegno.

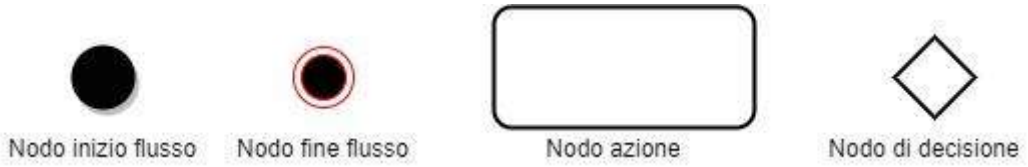

Figura 3.2 : Leggenda diagramma delle attività

Utilizzando questi componenti è possibile rappresentare le varie transizioni del flusso di un caso d'uso, in modo tale da descrivere il comportamento dinamico del sistema considerando i flussi alternativi.

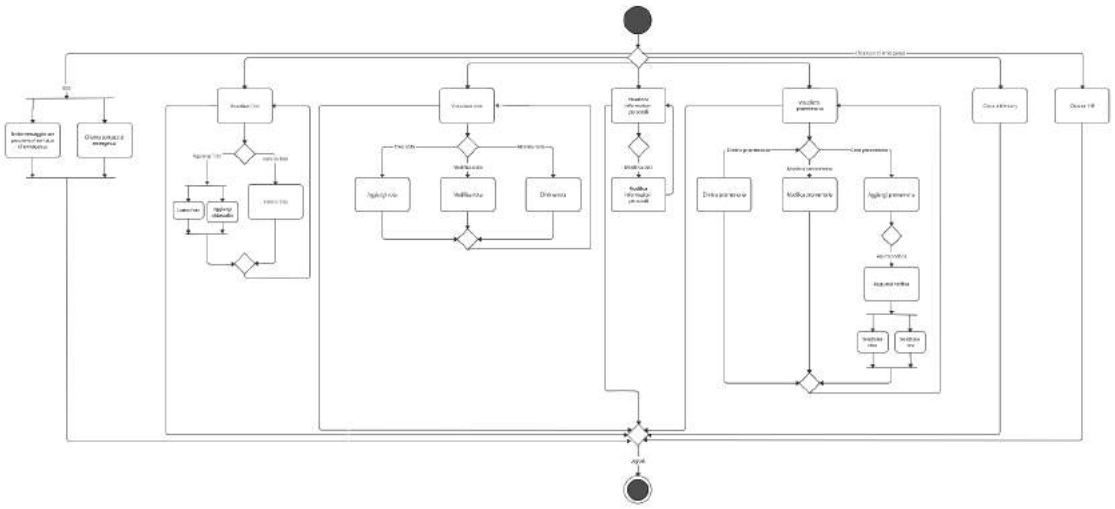

Figura 3.3 : Diagramma delle attività completo applicazione Ricordella

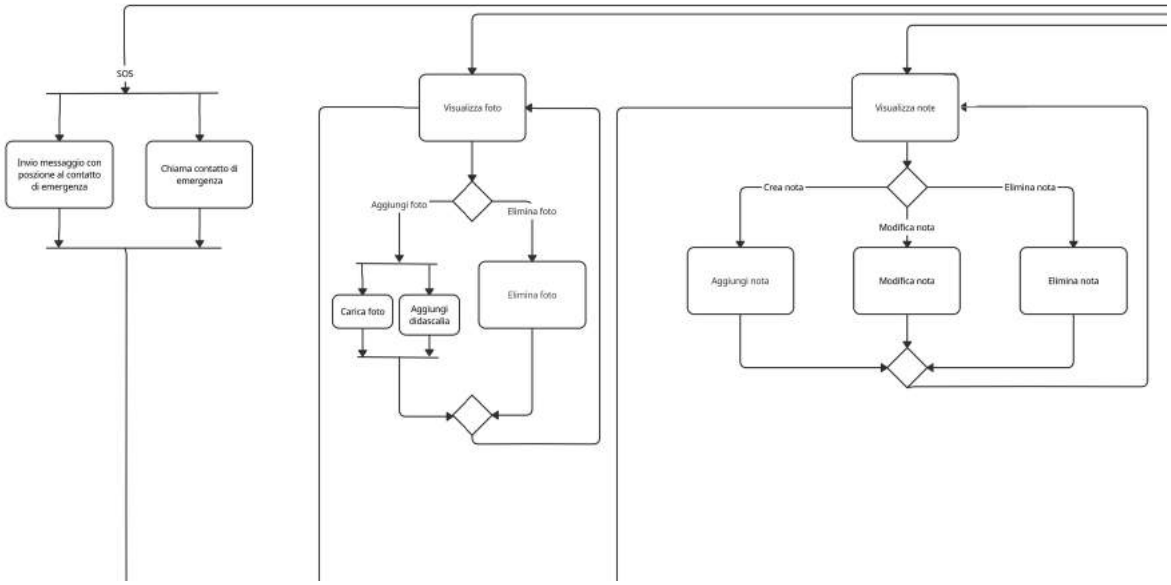

Figura 3.4 : Diagramma delle attività zoom prima parte

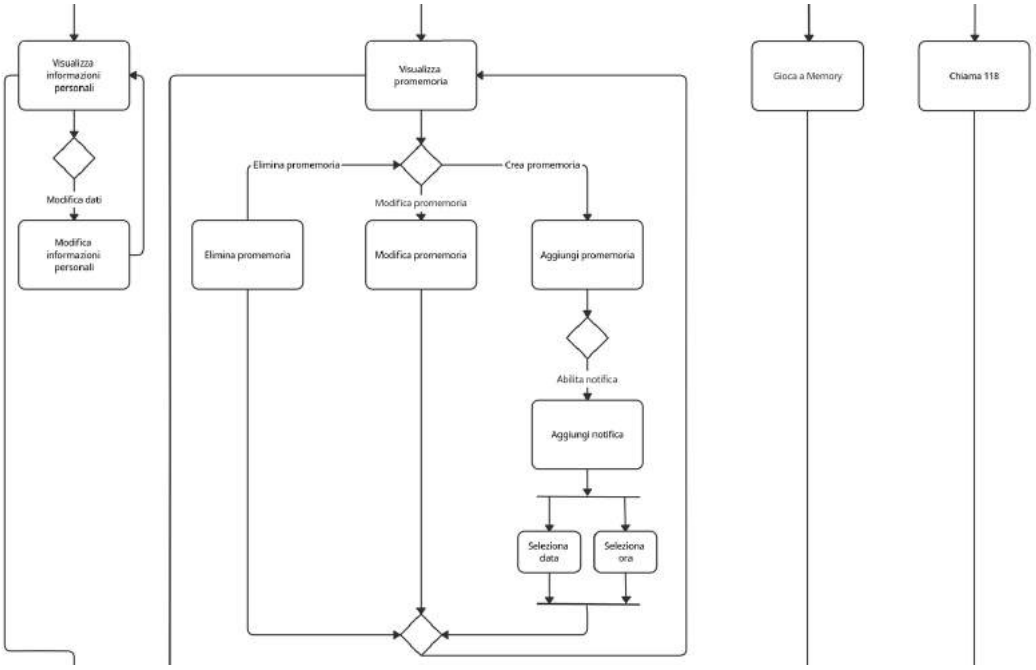

Figura 3.5 : Diagramma delle attività zoom seconda parte

3.3 - Progettazione database

La struttura dati scelta per il salvataggio di dati è stata SQLite. Prima di procedere alla realizzazione di un sistema per contenere i dati in modo corretto, una fase importante è stata quella di progettazione procedendo con la creazione di uno schema ER, ovvero modello entity-relationship. Questo modello è un modello concettuale di dati, e come tale fornisce una serie di strutture (costrutti), atte a descrivere la realtà in una maniera facile da comprendere e che prescinde dai criteri di organizzazione dei dati nei calcolatori.

3.3.1 - Modello ER Ricordella

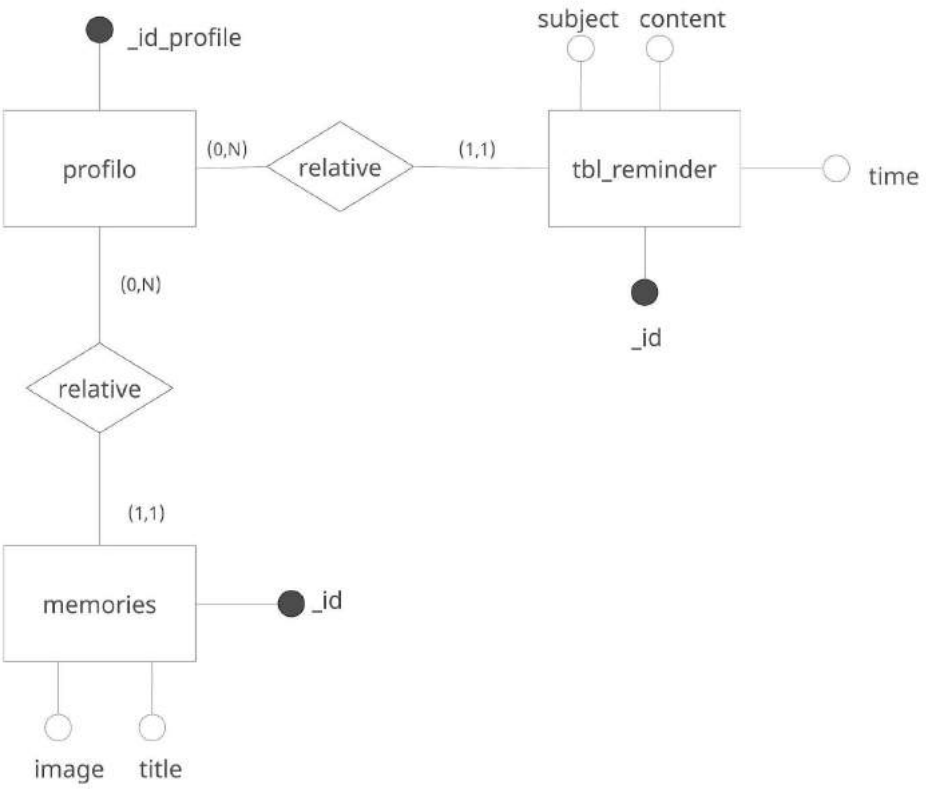

Figura 3.6 : Modello ER Ricordella

In questo schema ER, figura 3.6, ci sono due diverse entità *memories* ovvero dove vengono salvate le foto con relativa didascalia e *tbl_reminder*, dove vengono salvati invece i promemoria.

3.3.2 - Entità tbl_reminder

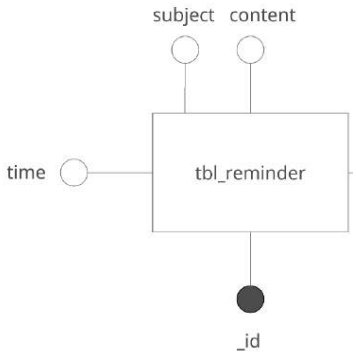

Figura 3.7 : Entità promemoria

L'entità tbl_reminder serve per tenere traccia dei promemoria immessi nell'applicazione con relativa data e ora per consentire l'invio della notifica in un determinato momento scelto. Ogni promemoria ha i seguenti attributi :

- _id : codice univoco che identifica un promemoria
- subject : titolo attribuito al promemoria
- content : contenuto del promemoria per aggiungere informazioni aggiuntive
- time : data e ora in cui ricevere la notifica del promemoria

	_id	subject	content	time
	Filter	Filter	Filter	Filter
1	1	appuntame...	chiedere ri...	2021-04-15...

tbl_reminder
_id
subject
content
time
alarm

Figura 3.8 : Entità promemoria attributi

3.3.3 - Entità memories

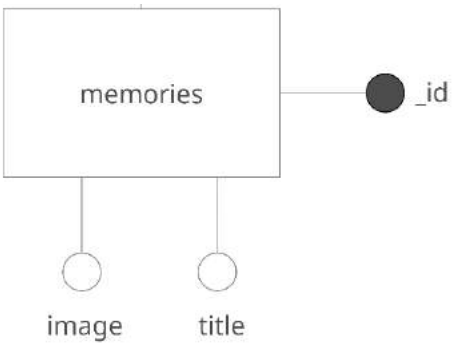

Figura 3.9 : Entità foto

L'entità memories serve per memorizzare le varie foto caricate con la relativa didascalia. Ogni foto ha i seguenti attributi :

- _id : codice univoco che identifica una foto
- title : didascalia o titolo attribuiti alla foto
- image : identificatore immagine caricata

Figura 3.10 : Entità foto attributi

3.3.4 - Diagramma UML delle classi

La seconda fase di progettazione consiste nel progettare il diagramma delle classi, il quale è un'estensione dei diagrammi entità/relazioni. Introduce classificazione, istanziazione e

aggregazione e definisce non solo gli attributi, ma anche le operazioni. Gli oggetti (istanze) di una stessa classe differiscono tra loro per i valori degli attributi e per le relazioni che li legano ad altri oggetti.

Le classi sono composte da tre campi :

- Il nome della classe, nel riquadro iniziale
- Gli attributi della classe, subito sotto il nome
- I metodi della classe, per ultimi

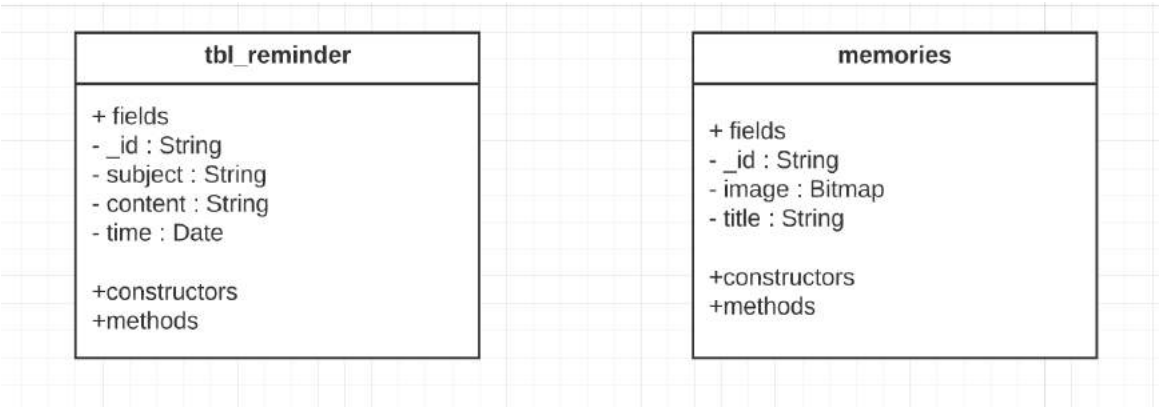

Figura 3.11 : Diagramma delle classi Ricordella

3.4 - SharedPreferences

Il salvataggio dei dati non è avvenuto solo tramite l'utilizzo di database, ma anche sfruttando le SharedPreferences. I dati collocati nelle SharedPreferences vengono salvati in un file XML contenuto nello storage interno, precisamente nella cartella *shared_prefs*. In particolare sono stati utilizzati per salvare i dati riguardanti le informazioni personali e quelli riguardanti le note. Le SharedPreferences rappresentano uno dei metodi più veloci ed efficienti per poter salvare dei dati su Android. Attraverso una coppia

chiave-valore è possibile salvare qualsiasi tipo di dato: boolean, float, int, long o String. È possibile avere un unico file con tutti i dati tramite il metodo *getPreferences()*, oppure utilizzare più file con il metodo *getSharedPreferences()* specificando il file su cui effettuare le operazioni.

Quindi per esempio per memorizzare "Nome e cognome" nelle informazioni personali, una volta inserito il valore, quel valore sarà salvato nel file XML associato ad una chiave (es "name_surname"-Diletta Maniglia), per recuperarlo e modificarlo, bisogna identificarlo tramite tag e poi eseguire le varie operazioni.

Capitolo 4

Implementazione

4.1 - Introduzione all'implementazione

Questo capitolo ripercorre le varie fasi che hanno caratterizzato lo sviluppo dell'applicazione del punto di vista implementativo. Quindi verranno analizzati tutti i file utilizzati per la realizzazione del progetto, studiate e spiegate le molteplici componenti, le Activity e le loro interfacce. Per realizzare l'applicazione Ricordella, come già detto, è stato usato l'IDE Android Studio, il linguaggio di programmazione Java e SQLite per gestire i dati.

Inizialmente per effettuare le operazioni di controllo è stato usato l'emulatore virtuale fornito da Android Studio ma questa tecnica ha causato spesso rallentamento del

sistema operativo e time consuming, quindi si è optato successivamente sulla scelta di collegare un dispositivo con sistema operativo Android al pc per testare l'applicazione direttamente su cellulare, in modo tale da velocizzare lo sviluppo del software.

4.2 - Struttura del progetto

L'applicazione Ricordella è composta da più file .java, contenuti nella cartella main/java/com.example.ricordell su Android Studio e raggruppati poi in varie sottocartelle in base alle funzionalità, che verranno di seguito illustrati e analizzati.

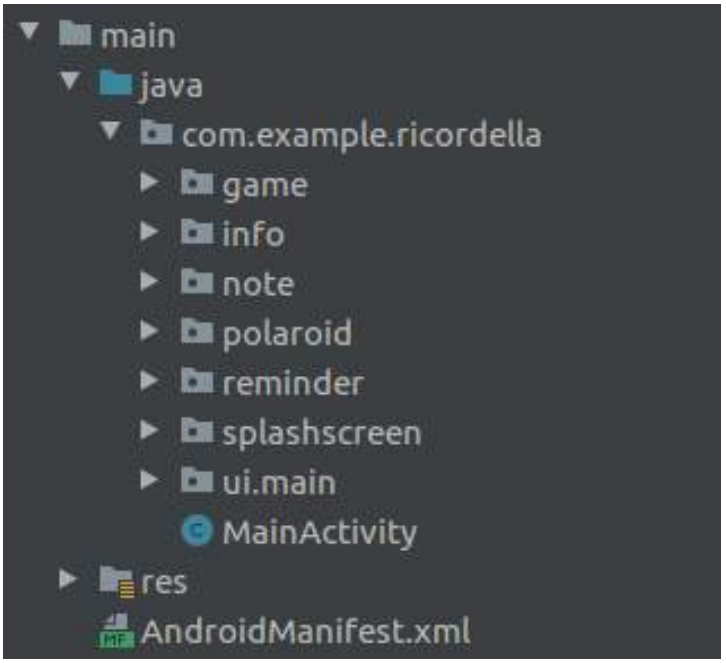

Figura 4.1 : Cartella main, res e file XML AndroidManifest.xml

4.2.1 - Cartella main/java/com.example.ricordella

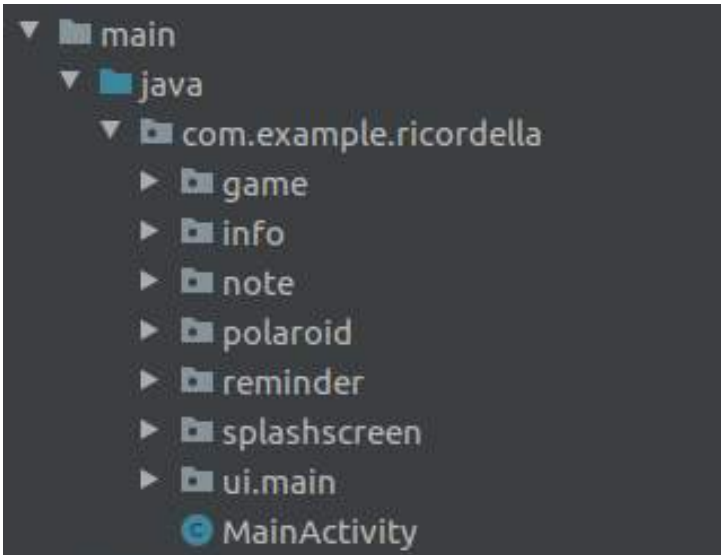

Figura 4.2 : Cartella main

- **game:** la cartella game contiene l'activity che implementa il gioco Memory

Figura 4.3 : Cartella game

- **info:** la cartella info le activity che si occupano del contesto riguardante le informazioni personali, quindi della visualizzazione, modifica e caricamento della facciata aggiornata

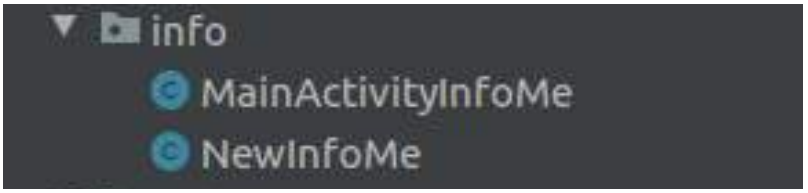

Figura 4.4 : Cartella info

- **note:** la cartella note possiede l'activity per caricare le note correttamente e la classe NoteEditorActivity che si occupa della funzionalità di creare e poter modificare una nota

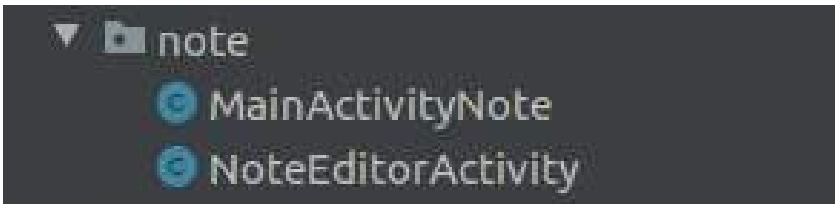

Figura 4.5 : Cartella note

- **polaroid:** la cartella polaroid contiene i file che hanno il compito di occuparsi delle funzionalità relative al contesto delle foto, l'activity MainActivityPolaroid carica la schermata principale in cui vengono visualizzate le foto caricate, mentre l'activity NewPolaroidActivity mostra la schermata che permette di effettuare il caricamento delle foto; le altre classi gestiscono l'oggetto foto, il salvataggio nel database e le eventuali altre operazioni possibili

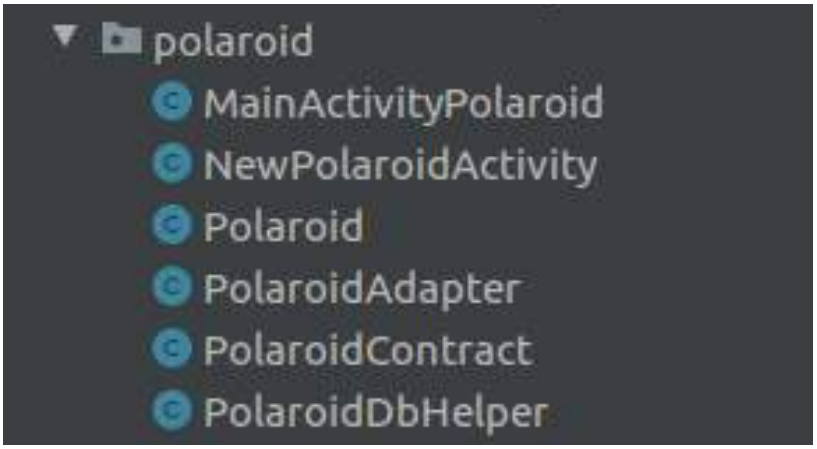

Figura 4.6 : Cartella polaroid

- **reminder:** la cartella reminder contiene le activity che caricano la lista aggiornata dei promemoria e permettono di aggiungere un nuovo promemoria grazie alle delle altre classi implementate, le quali servono a salvare i dati e svolgere le altre funzionalità sui dati contenuti nel database, e implementare l funzione di notifica

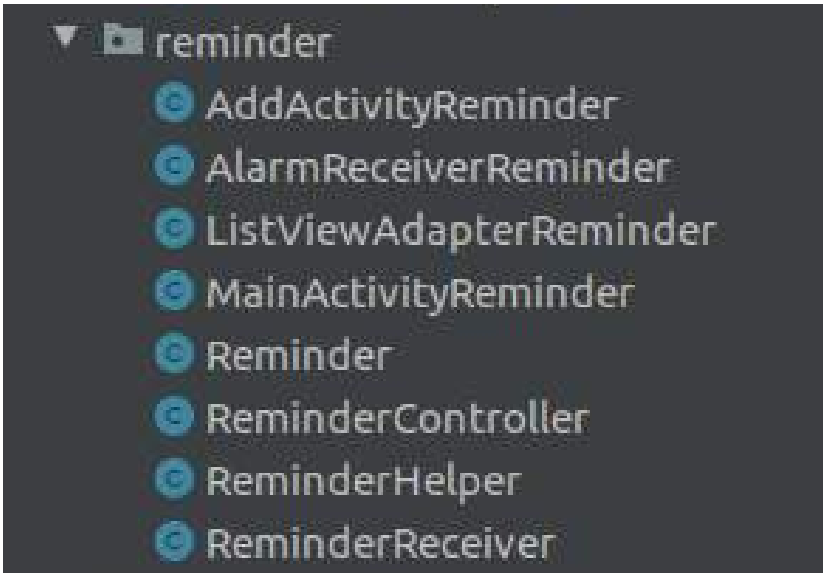

Figura 4.7 : Cartella reminder

- **splashscreen:** nella cartella splashscreen è presente l'activity che si occupa di caricare la schermata iniziale con il logo dell'applicazione, visibile per pochi secondi

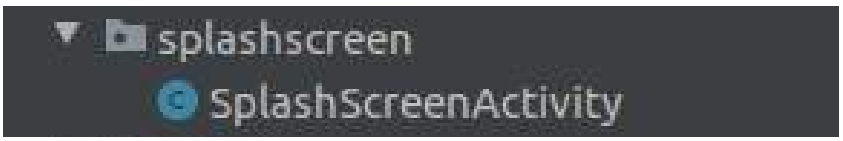

Figura 4.8 : Cartella splashscreen

- **MainActivity:** l'activity MainActivity è l'activity principale attraverso la quale si può accedere alle varie activity secondarie dell'applicazione Ricordella

Figura 4.9 : Activity MainActivity

4.2.2 - Cartella main/res

Questa directory ospita i file XML che descrivono delle animazioni. drawable. Questa directory deve essere usata per le immagini necessarie al software. Sono supportati i formati PNG e JPEG. In particolare, per esempio, la cartella drawable contiene i file .xml che contengono i componenti estetici utilizzati nell'applicazione come le immagini utilizzate nel gioco Memory. Nella cartella values invece sono immagazzinati i valori creati usati nell'applicazione come nuovi colori o font di testo, in modo tale da definirli una sola volta. Nella cartella layout invece ci sono i layout grafici associati alle rispettive activity.

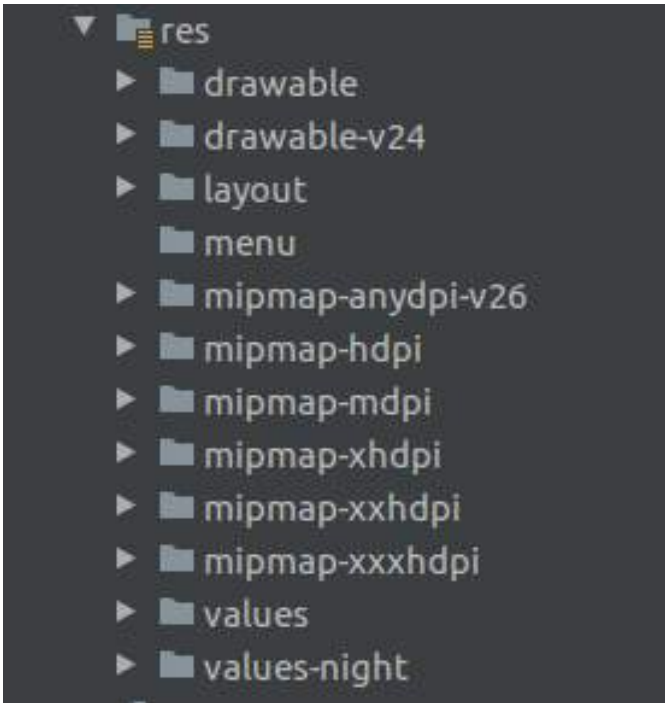

Figura 4.10 : Cartella res

4.3 - Implementazione

In questo paragrafo verranno illustrati i file principali che compongono il progetto, soffermandosi in particolare su pezzi di codice fondamentali.

4.3.1 - AndroidManifest

Il primo file da analizzare è il file *AndroidManifest.xml* che contiene le informazioni di base del progetto. Inizialmente vengono definiti i permessi che l'applicazione deve possedere, per poi poter utilizzare sul dispositivo le proprie funzionalità implementate.

Alcuni permessi sono :

- CALL_PHONE consente a un'applicazione di avviare una telefonata senza passare attraverso l'interfaccia utente di Dialer per consentire all'utente di confermare la chiamata.
- ACCESS_FINE_LOCATION consente a un'app di accedere alla posizione precisa.
- ACCESS_COARSE_LOCATION consente a un'app di accedere alla posizione approssimativa.
- ACCESS_NETWORK_STATE consente alle applicazioni di accedere alle informazioni sulle reti
- INTERNET consente alle applicazioni di aprire socket di rete
- SEND_SMS consente a un'applicazione di inviare messaggi SMS
- READ_PHONE_STATE consente l'accesso in sola lettura allo stato del telefono, comprese le informazioni sulla rete cellulare corrente, lo stato delle chiamate in corso e un elenco di tutti gli account telefonici registrati sul dispositivo
- WAKE_LOCK è un meccanismo all'interno del sistema operativo Android. I programmi che richiedono risorse di sistema utilizzano la funzione wakelock per mantenere il telefono sveglio. Quando il processo termina il suo compito, rilascia la funzione wakelock e il telefono torna in stato di stop

In particolare ACCESS_FINE_LOCATION e ACCESS_COARSE_LOCATION si usano entrambe in modo tale da memorizzare l'ultima posizione conosciuta quindi anche per risparmiare batteria sui dispositivi mobili.
Nella sezione *application* ci sono i dettagli dell'applicazione come il nome dell'applicazione, il logo visualizzato sul dispositivo e infine le varie *activity, receiver*

distinti tramite etichetta in base a quale funzionamento hanno e all'interno dichiarate ulteriori particolari funzionalità che necessitano offrire, come ad esempio l'activity che si aprirà al lancio dell'applicazione è *SplashScreenActivity*.

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

        <uses-permission android:name="android.permission.CALL_PHONE" />
        <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
        <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
        <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
        <uses-permission android:name="android.permission.INTERNET" />
        <uses-permission android:name="android.permission.SEND_SMS" />
        <uses-permission android:name="android.permission.READ_CONTACTS"
/>
        <uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
        <uses-permission
android:name="android.permission.READ_PHONE_STATE" />
        <uses-permission android:name="android.permission.WAKE_LOCK" />

        <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Ricordella"
        tools:targetApi="m">
        <activity android:name=".MainActivity"></activity>
        <activity android:name=".reminder.AddActivityReminder" />
        <activity android:name=".polaroid.NewPolaroidActivity" />
```

```
        <activity android:name=".polaroid.MainActivityPolaroid" />
        <activity android:name=".game.MainActivityMemory" />
        <activity
            android:name=".reminder.MainActivityReminder"
            android:label="@string/app_name">
            <intent-filter>
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".info.NewInfoMe" />
        <activity android:name=".info.MainActivityInfoMe" />
        <activity android:name=".note.NoteEditorActivity" />
        <activity android:name=".note.MainActivityNote" />
        <activity
           android:name=".splashscreen.SplashScreenActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

        <receiver
            android:name=".reminder.AlarmReceiverReminder"
            android:enabled="true"
            android:exported="false">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"
/>
                <action
android:name="android.intent.action.INPUT_METHOD_CHANGED" />
                <action
android:name="com.example.ricordella.AlarmReceiverReminder" />
            </intent-filter>
        </receiver>
        </application>

</manifest>
```

4.3.2 - SplashScreenActivity

Analizzando le diverse activity che fanno parte dell'applicazione Ricordella, la prima visualizzata è la SplashScreenActivity. Essa rappresenta la schermata iniziale la quale appare per pochi secondi quando viene aperta l'applicazione, mostra un layout contenente il nome e il logo dell'applicazione.

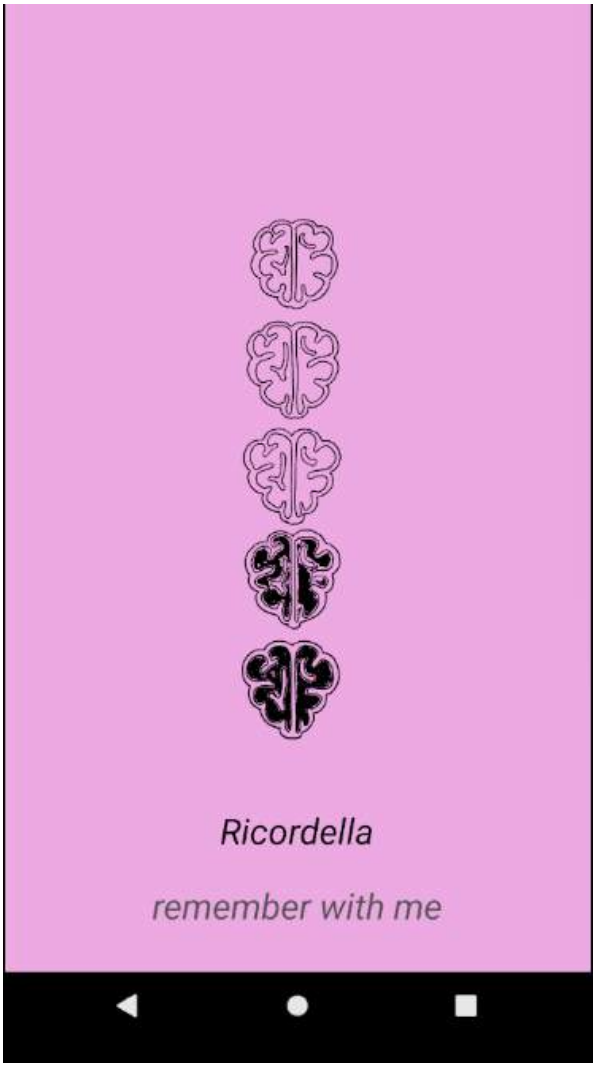

Figura 4.11 : Schermata iniziale applicazione

Quindi nella fase di implementazione del codice, inizialmente viene creata una nuova activity, poi nella onCreate dopo un'attesa di 4 secondi, viene lanciata l'activity principale MainActivity e terminata quella di launch.

```
public class SplashScreenActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash_screen);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);

        new Handler().postDelayed(new Runnable() {
            @Override
            public void run() {

                Intent intent = new Intent( packageContext: SplashScreenActivity.this, MainActivity.class);
                startActivity(intent);
                finish();
            }
        }, delayMillis: 4000);
    }
}
```

Figura 4.12 : Codice SplashScreenActivity

4.3.3 - MainActivity

MainActivity è l'activity principale, attraverso la qual si può accedere alle varie funzionalità dell'applicazione tramite tocco sui vari ImageButton. In questo modo vengono aperte activity secondarie che rappresentano una determinata funzione che è in grado di offrire l'applicazione.

Figura 4.13 : Layout MainActivity

Tramite l'utilizzo di *findViewById* viene associata l'immagine al corrispondente ImageButton trovando l'ID specificato corrispondente all'immagine stessa, viene restituito l'ID corrisponde a getId () o null se l'ID non è valido (<0). Una volta fatta questa associazione ad esempio nel caso di *noteButton* utilizzando *noteButton.setOnClickListener* impostiamo l'azione che si deve compiere al tocco dell'immagine, in questo caso verrà aperta una nuova activity MainActivityNote.

```
//to write note
noteButton = findViewById(R.id.imageButton2);

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

        Intent intent = new Intent( packageContext: MainActivity.this, MainActivityNote.class);
        startActivity(intent);            }
});
```

Figura 4.14 : Parte di codice MainActivity

Premendo invece l'immagine dell'ambulanza o di SOS, in particolare, non vengono aperte altre activity ma tramite il tocco sull'ambulanza viene fatta una chiamata di emergenza al *118* richiamando la funzione *callNumber()* mostrata nell'immagine 4.16, mentre con SOS utilizzando *SharedPreferences*, che permette di salvare e recuperare dati nella forma key-value, tramite tag viene recuperato il valore del contatto di emergenza immesso durante la fase di compilazione delle informazioni personali, e viene effettuata una chiamata automatica a questo contatto passandolo come parametro alla funzione *callNumber()*, inoltre viene inviato un messaggio automatico con la posizione corrente o l'ultima posizione disponibile con una richiesta di aiuto, allo stesso numero, richiamando le funzioni *sensSms()* figura 4.17 e *getLastLocation()* figura 4.19.

```
sosButton = findViewById(R.id.imageButton9);

sosButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // to send sms with location and call emergency contact
        SharedPreferences prefs = getSharedPreferences( name: "MY_DATA", MODE_PRIVATE);

        int numberEC = prefs.getInt( key: "EMERGENCY_CONTACT_NUMBER",  defValue: 0);

        String emergencyContact = String.valueOf(numberEC);
        sendSms(emergencyContact,  message: "Hi! I'm in danger, this is my position : " +getLastLocation());

        callNumber(emergencyContact);
    }

});
```

Figura 4.15 : Parte di codice MainActivity

Nella funzione *callNumber()* viene implementato il codice per effettuare una telefonata automatica. Inizialmente viene creato l'oggetto *intent* con azione definita *ACTION_CALL*, poi aggiunto il data del contatto di emergenza e prima di far partire questa activity, avviene un controllo per verificare se l'applicazione ha il permesso di effettuare una telefonata, se non lo possiede si apre una finestra di richiesta per consentire questa azione.

```
//to call phone number by clicking imageButton
void callNumber(String number){
    try{
        Intent intent = new Intent (Intent.ACTION_CALL);
        intent.setData(Uri.parse("tel:"+number));

        if (ActivityCompat.checkSelfPermission( context: MainActivity.this,
            Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED)
        {
            ActivityCompat.requestPermissions( activity: MainActivity.this, new String[]{Manifest.permission.CALL_PHONE},  requestCode: 1);
        }
        startActivity(intent);
    }
    catch (Exception e) {
        Toast.makeText( context: MainActivity.this,  text: "Call failed, please try again", Toast.LENGTH_SHORT).show();
    }
}
```

Figura 4.16 : Funzione callNumber() MainActivity

La funzione *sendSms()* funzione in modo quasi analogo alla funzione precedente, ma in questa funzione per procedere all'invio del messaggio viene utilizzata la classe SmsManager.

```
void sendSms(String number, String message)
{
    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.SEND_SMS) == PackageManager.PERMISSION_GRANTED)
    {
        try
        {
            SmsManager smsMgrVar = SmsManager.getDefault();
            smsMgrVar.sendTextMessage(number,  scAddress: null, message,  sentIntent: null,  deliveryIntent: null);
            Toast.makeText(getApplicationContext(),  text: "Message Sent",
                    Toast.LENGTH_LONG).show();
        }
        catch (Exception ErrVar)
        {
            Toast.makeText(getApplicationContext(),ErrVar.getMessage().toString(),
                    Toast.LENGTH_LONG).show();
            ErrVar.printStackTrace();
        }
    }
    else
    {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M)
        {
            requestPermissions(new String[]{Manifest.permission.SEND_SMS},  requestCode: 10);
        }
    }
}
```

Figura 4.17 : Funzione sendSms() MainActivity

Di seguito è mostrato un esempio delle azioni svolte all'avvio del SOS.

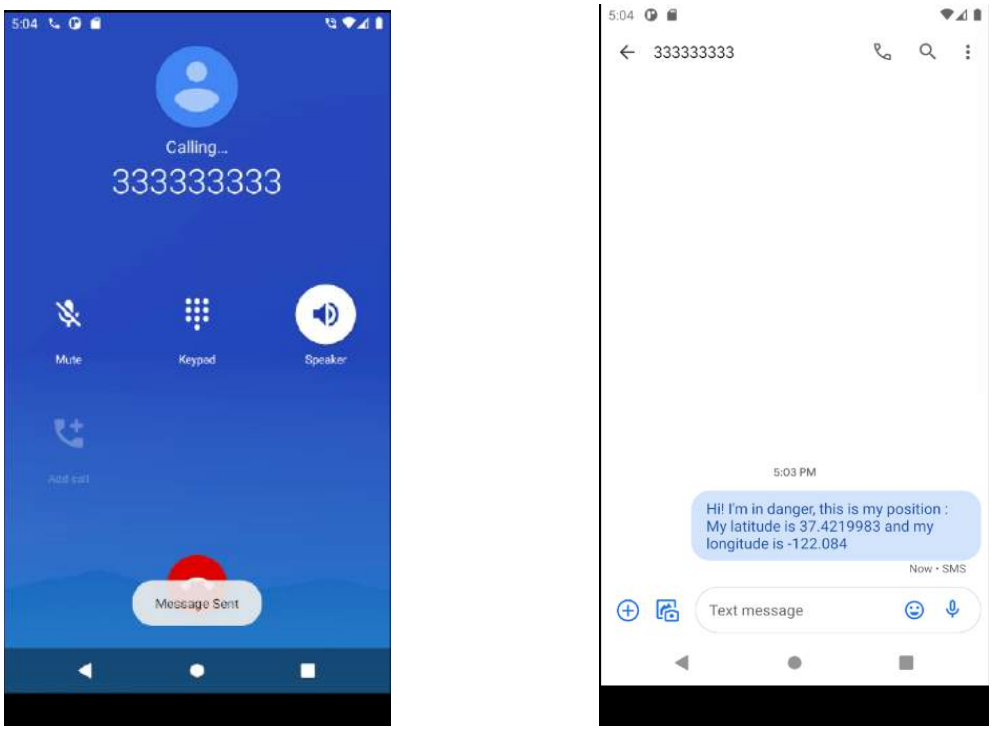

Figura 4.18 : Esempio di chiamata al contatto di emergenza e invio del messaggio con posizione corrente

La funzione *getLastLocation()* invece serve per recuperare la posizione corrente del dispositivo o l'ultima posizione rilevata. Per recuperare l'ultima posizione recente conosciuta viene utilizzato il nuovo Google Play Services SDK (v11+) attraverso la classe *FusedLocationProviderClient*. Quindi viene creato un oggetto di quel tipo con i metodi *.getLatitude()* e *.getLongitude()* recuperati i valori di latitudine e longitudine da far visualizzare poi nel messaggio inviato al contatto di emergenza, in modo tale da risalire alla posizione del dispositivo.

```
// get location
@SuppressLint("MissingPermission")
public String getLastLocation() {
    // Get last known recent location using new Google Play Services SDK (v11+)
    FusedLocationProviderClient locationClient = getFusedLocationProviderClient( activity: this);

    locationClient.getLastLocation()
            .addOnSuccessListener(new OnSuccessListener<Location>() {
                @Override
                public void onSuccess(Location location) {
                    // GPS location can be null if GPS is switched off
                    if (location != null) {
                        onLocationChanged(location);
                        double lat = location.getLatitude();
                        double lon = location.getLongitude();
                        latitude = String.valueOf(lat);
                        longitude = String.valueOf(lon);
                    }
                }
            })
            .addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                    Log.d( tag: "MapDemoActivity",  msg: "Error trying to get last GPS location");
                    e.printStackTrace();
                }
            });
    return "My latitude is "+ latitude +" and my longitude is "+ longitude;

}
```

Figura 4.19 : Funzione getLastLocation() MainActivity

4.3.4 - info/MainActivityInfoMe

La schermata iniziale premendo sull'*ImageButton* delle info, che fa avviare l'activity *MainActivityInfoMe*, è la seguente nell'immagine 4.20, dove a sinistra c'è la schermata con i valori di default mentre a destra una schermata ipotetica compilata dall'utente.

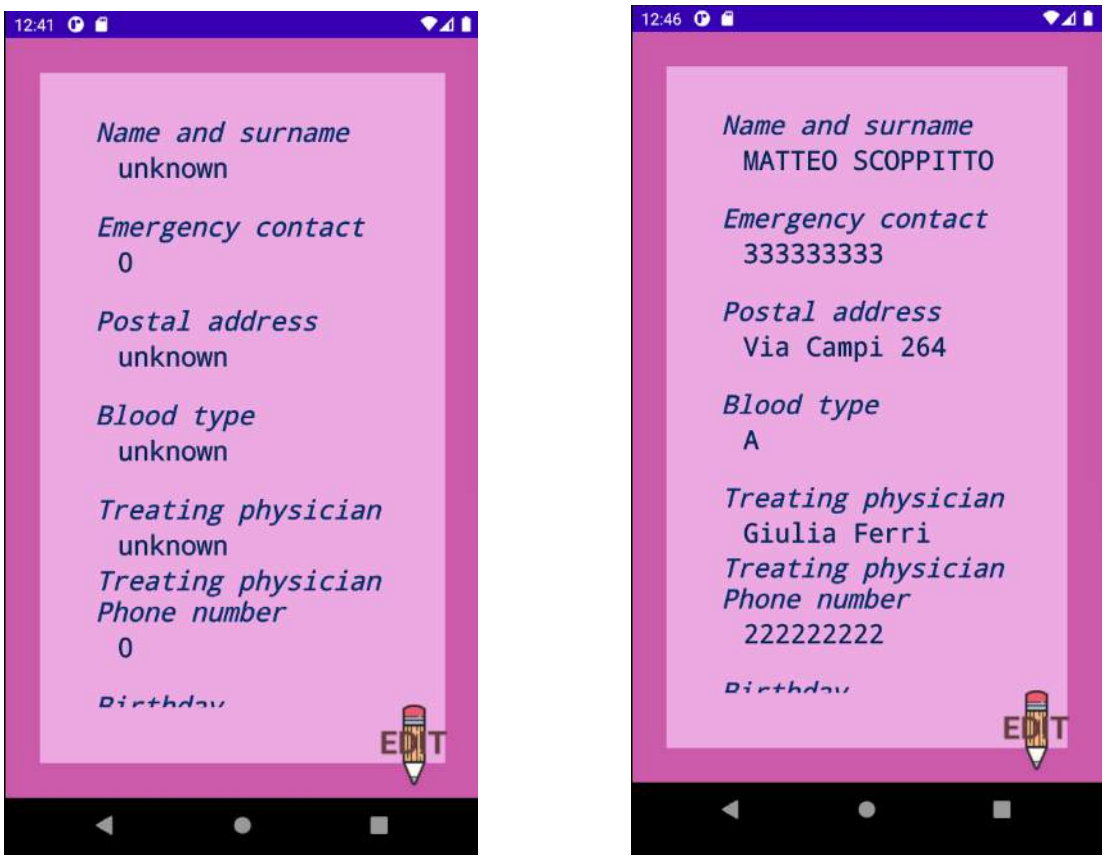

Figura 4.20 : Layout MainActivityInfoMe

Nella classe *MainActivityInfoMe* attraverso l'utilizzo di *SharedPreferences*, vengono recuperati tramite specifico tag i valori immessi nelle informazioni personali per essere visualizzati, se non ancora immessi viene visualizzato il valore di default. In questa

activity tramite *ImageButton edit* è possibile avviare l'activity per modificare le informazioni personali *NewInfoMe*.

```
SharedPreferences prefs = getSharedPreferences( name: "MY_DATA", MODE_PRIVATE);
// prefs.edit().clear().commit();


//get(key, default value)
personame = prefs.getString( key: "MY_NAME",  defValue: "unknown");
numberEC = prefs.getInt( key: "EMERGENCY_CONTACT_NUMBER",  defValue: 0);
postalAddress = prefs.getString( key: "POSTAL_ADDRESS",  defValue: "unknown");
bloodtype = prefs.getString( key: "BLOOD_TYPE",  defValue: "unknown");
treatingPhysician = prefs.getString( key: "TREATING_PHYSICIAN",  defValue: "unknown");
numberT = prefs.getInt( key: "TREATING_PHYSICIAN_PHONE_NUMBER",  defValue: 0);
birthday = prefs.getString( key: "BIRTHDAY",  defValue: "unknown");
pathology = prefs.getString( key: "PATHOLOGY",  defValue: "unknown");

//set values
((TextView)findViewById(R.id.personName)).setText(personame);
((TextView)findViewById(R.id.numberEmC)).setText(numberEC+"");
((TextView)findViewById(R.id.postalAddress)).setText(postalAddress);
((TextView)findViewById(R.id.bloodtype)).setText(bloodtype);
((TextView)findViewById(R.id.treatingPhysician)).setText(treatingPhysician);
((TextView)findViewById(R.id.phoneNumberT)).setText(numberT+"");
((TextView)findViewById(R.id.birthday)).setText(birthday);
((TextView)findViewById(R.id.pathology)).setText(pathology);

prefs.edit().putInt("EMERGENCY_CONTACT_NUMBER", numberEC).apply();
edit = findViewById(R.id.editButton);
```

Figura 4.21 : Pezzo di codice MainActivityInfoMe

4.3.5 - info/NewInfoMe

Figura 4.22 : Layout NewInfoMe

Nell'activity NewInfoMe è possibile modificare e salvare i dati delle informazioni personali, sempre grazie all'utilizzo di SharedPreferences, che utilizzando specifici tag associa essi alle informazioni immesse, in modo tale da poterli recuperare facilmente. Di seguito viene illustrata la funzione *saveData()* che tramite il metodo *.putString* o *.putInt*, in base al tipo dell'input, della classe *SharedPreferences.Editor*, immette gli input scritti dall'utente nell'oggetto *editor* associandolo ad uno specifico tag. Inoltre questa funzione fa tornare automaticamente l'applicazione all'activity precedente *MainActivityInfoMe*.

```
public void saveData(View view){

    int ec_ = 0, tppn_ = 0;

    //get input text
    String name = nameInput.getText().toString();
    String ec = emergencyContactInput.getText().toString();
    String pa = postalAddressInput.getText().toString();
    String bt = bloodTypeInput.getText().toString();
    String tp = treatingPhysicianInput.getText().toString();
    String tppn = treatingPhysicianPhoneNumberInput.getText().toString();
    String b = birthdayInput.getText().toString();
    String p = pathologyInput.getText().toString();

    try{
        ec_ = parseInt(ec);
        tppn_ = parseInt(tppn);
    } catch(NumberFormatException ex){ // handle your exception

    }

    //save data
    SharedPreferences.Editor editor = prefs.edit();

    editor.putString("MY_NAME", name);
    editor.putInt("EMERGENCY_CONTACT_NUMBER", ec_);
    editor.putString("POSTAL_ADDRESS", pa);
    editor.putString("BLOOD_TYPE", bt);
    editor.putString("TREATING_PHYSICIAN", tp);
    editor.putInt("TREATING_PHYSICIAN_PHONE_NUMBER", tppn_);
    editor.putString("BIRTHDAY", b);
    editor.putString("PATHOLOGY", p);

    editor.apply();

    Intent intent = new Intent( packageContext: NewInfoMe.this, MainActivityInfoMe.class);
    startActivity(intent);

    //return previous activity
    finish();
}
```

Figura 4.23 : saveData() NewInfoMe

4.3.6 - note/NoteEditorActivity

L'activity *NoteEditorActivity* implementa la funzionalità di creazione di una nuova nota.

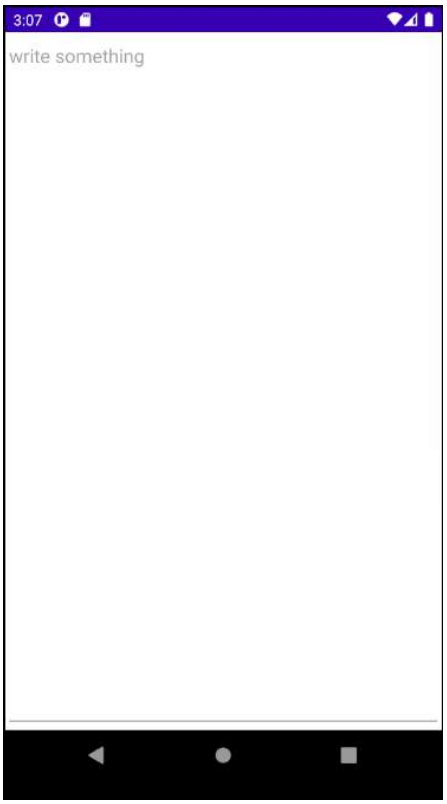

Figura 4.24 : Layout NoteEditotActivity

Di seguito il codice in cui viene messo nell'*ArrayList notes* della classe *MainActivityNote* il testo immesso dall'utente nel blocco delle note, associato ad uno specifico id, successivamente viene creato l'oggetto *sharedPreferences* in cui caricare i dati. Quindi l'oggetto *notes* viene caricato in un oggetto di tipo *HashSet<String>* il quale viene infine associato ad un tag e salvato nell'oggetto *sharedPreferences* in modo tale da poterlo recuperare all'avvio dell'applicazione in modo automatico.

```java
@Override
public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {
    MainActivityNote.notes.set(noteId, String.valueOf(charSequence));
    MainActivityNote.arrayAdapter.notifyDataSetChanged();
    // Creating Object of SharedPreferences to store data in the phone
    SharedPreferences sharedPreferences = getApplicationContext().getSharedPreferences( name: "com.example.notes", Context.MODE_PRIVATE);
    HashSet<String> set = new HashSet(MainActivityNote.notes);
    sharedPreferences.edit().putStringSet( key: "notes", set).apply();
}
```

Figura 4.25 : onTextChanged() NoteEditorActivity

4.3.7 - note/MainActivityNote

L'activity MainActivityNote permette di visualizzare una lista di note, modificarle e far partire l'activity NoteEditorActivity in modo tale da poterne aggiungere altre.

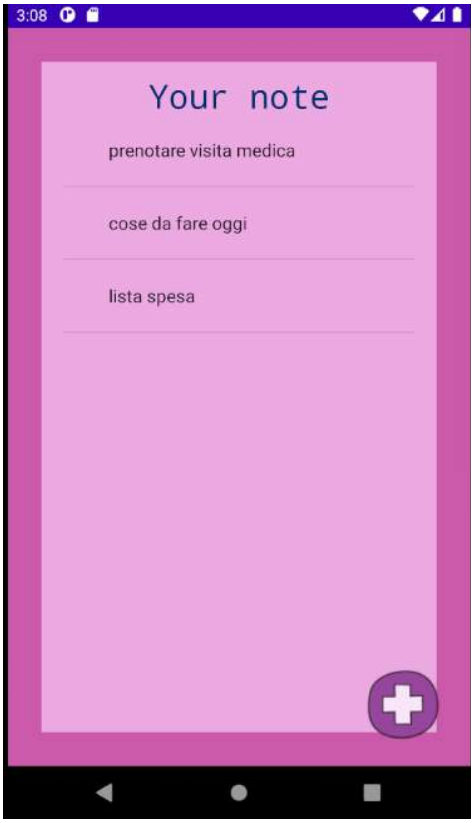

Figura 4.26 : Layout MainActivityNote

Nel codice sottostante possiamo vedere che viene creata una *ListView* per poter visualizzare le varie note in stile lista, gli elementi della lista vengono identificati tramite un id in modo tale che quando vengono premute per essere aperte, è possibile aprire la nota cliccata utilizzando la classe *NoteEditorActivity*. Inoltre premendo in modo prolungato con *.onItemLongClick* è stata implementata la funzione per poter eliminare la nota cliccata, in un primo momento dall'*ArrayList* notes, in cui vengono memorizzate durante la creazione, e successivamente viene creato un oggetto *HashSet* in cui viene memorizzato *notes* modificato precedentemente e infine vengono applicate le modifiche all'oggetto *SharedPreferences* al tag corrispondente *"notes"* a cui sono associate le note salvate.

```
listView.setAdapter(arrayAdapter);

listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {

        // Going from Note to NotesEditorActivity
        Intent intent = new Intent(getApplicationContext(), NoteEditorActivity.class);
        intent.putExtra( name: "noteId", i);
        startActivity(intent);
    }
});

listView.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
    @Override
    public boolean onItemLongClick(AdapterView<?> adapterView, View view, int i, long l) {

        final int itemToDelete = i;
        // To delete the data from the App
        new AlertDialog.Builder( context: MainActivityNote.this)
                .setIcon(android.R.drawable.ic_dialog_alert)
                .setTitle("Are you sure?")
                .setMessage("Do you want to delete this note?")
                .setPositiveButton( text: "Yes", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialogInterface, int i) {
                        notes.remove(itemToDelete);
                        arrayAdapter.notifyDataSetChanged();
                        SharedPreferences sharedPreferences = getApplicationContext().getSharedPreferences( name: "com.example.notes", Context.MODE_PRIVATE);
                        HashSet<String> set = new HashSet(MainActivityNote.notes);
                        sharedPreferences.edit().putStringSet( key: "notes", set).apply();
                    }
                }).setNegativeButton( text: "No", listener: null).show();
        return true;
    }
});
```

Figura 4.27 : Pezzo di codice MainActivityNote

4.3.8 - game/MainActivityMemory

L'activity *MainActivityMemory* implementa il gioco memory che consiste nel trovare tutte le coppie di immagine uguali. Nell'immagine 4.28 possiamo notare la schermata iniziale del gioco a sinistra e a destra invece un'immagine nel pieno del gioco, in cui le coppie di immagini trovate diventano invisibile

.

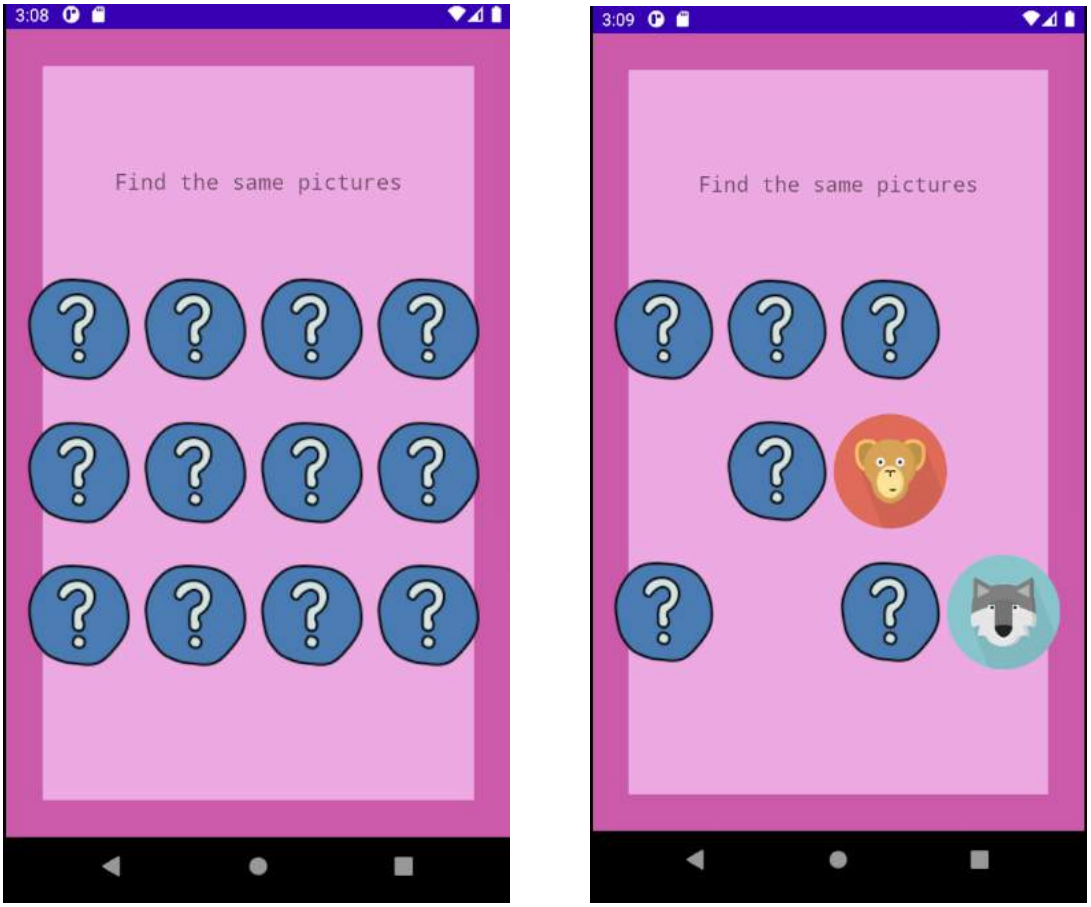

Figura 4.28 : Layout MainActivityMemory

A fine partita si aprirà una *dialog* che chiederà al giocatore se iniziare una nuova partita o uscire e tornare all'activity principale.

Figura 4.29 : Layout MainActivityMemory

Le immagini a inizio gioco vengono caricate in modo casuale. Vengono caricate 12 immagini (6 immagini diverse tra loro e 6 doppione di esse), ad ogni immagine viene associato un id, da 101 a 106 e da 201 a 206 (ai doppioni), al momento del controllo per verificare se due immagini sono uguali, viene inizialmente controllato quali immagini sono state selezionate e vengono salvate in una variabile temporanea. Poi se il tag dell'immagine selezionata è maggiore di 200, ad esso viene sottratto 100, in modo tale da avere tutti i tag delle immagini selezionate che vanno da 101 a 106. In questo modo poi tramite la funzione *calculate()* è possibile verificare se i due tag delle immagini coincidono, se essi coincidono vengono rese invisibili le *ImageView* e viene fatto un

controllo generale su tutte le immagini richiamando la funzione *checkEnd()*, figura 4.31, in cui se tutte le immagini risultano con la visibilità uguale a *View.INVISIBLE*, il gioco termina e viene aperta automaticamente una *dialog* in cui si può decidere di giocare nuovamente o no.

```
//check which image is selected and save it in a temporary variable
if(cardNumber == 1){

    firstCard = cardsArray[card];
    if(firstCard > 200){
        firstCard = firstCard - 100;
    }
    cardNumber = 2;
    clickedFirst = card;
    iv.setEnabled(false);

} else if (cardNumber == 2) {

    secondCard = cardsArray[card];
    if( secondCard > 200) {
        secondCard = secondCard - 100;
    }
    cardNumber = 1;
    clickedSecond = card;

    iv_11.setEnabled(false);
    iv_12.setEnabled(false);
    iv_13.setEnabled(false);
    iv_14.setEnabled(false);
    iv_21.setEnabled(false);
    iv_22.setEnabled(false);
    iv_23.setEnabled(false);
    iv_24.setEnabled(false);
    iv_31.setEnabled(false);
    iv_32.setEnabled(false);
    iv_33.setEnabled(false);
    iv_34.setEnabled(false);

    Handler handler = new Handler();
    handler.postDelayed(new Runnable(){
        @Override
        public void run(){
            //check if the selected image are equal
            calculate();
        }
    }, delayMillis: 1000);
}
```

Figura 4.30 : Pezzo di codice MainActivityMemory

```java
private void checkEnd(){

    if(iv_11.getVisibility() == View.INVISIBLE &&
            iv_12.getVisibility() == View.INVISIBLE &&
            iv_13.getVisibility() == View.INVISIBLE &&
            iv_14.getVisibility() == View.INVISIBLE &&
            iv_21.getVisibility() == View.INVISIBLE &&
            iv_22.getVisibility() == View.INVISIBLE &&
            iv_23.getVisibility() == View.INVISIBLE &&
            iv_24.getVisibility() == View.INVISIBLE &&
            iv_31.getVisibility() == View.INVISIBLE &&
            iv_32.getVisibility() == View.INVISIBLE &&
            iv_33.getVisibility() == View.INVISIBLE &&
            iv_34.getVisibility() == View.INVISIBLE){

        AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder( context: MainActivityMemory.this);
        alertDialogBuilder
                .setMessage("GOOD JOB")
                .setCancelable(false)
                .setPositiveButton( text: "NEW", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        Intent intent = new Intent (getApplicationContext(), MainActivityMemory.class);
                        startActivity(intent);
                        finish();
                    }
                })
                .setNegativeButton( text: "EXIT", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) { finish(); }
                });
        AlertDialog alertDialog = alertDialogBuilder.create();
        alertDialog.show();
    }
}
```

Figura 4.31 : Pezzo di codice MainActivityMemory

4.3.9 - polaroid/MainActivityPolaroid

L'activity *MainActivityPolaroid* permette di visualizzare le foto caricate dall'utente in stile polaroid. Quando una foto viene caricata, viene inserita in un database *SQLite* insieme alla sua didascalia. Per essere visualizzate, l'immagine viene caricata in un oggetto *ImageView* mentre la didascalia in un oggetto *TextView* , infine inserite in un oggetto *GridView* in modo tale da poter visualizzare tutte le foto caricate. Se la foto viene caricata correttamente, viene visualizzata come la foto nell'immagine 4.32 con didascalia *"la mia casa"*, altrimenti viene caricata e visualizzata un'immagine di default. Tenendo premuto sulla foto, si apre inoltre una dialog che volendo ti permette di eliminare la foto premuta sia dalla *GridView* che dal database. Infine è possibile aggiungere una nuova foto tramite l'*ImageButton* con il simbolo "+" il quale apre l'activity *NewPolaroidActivity*.

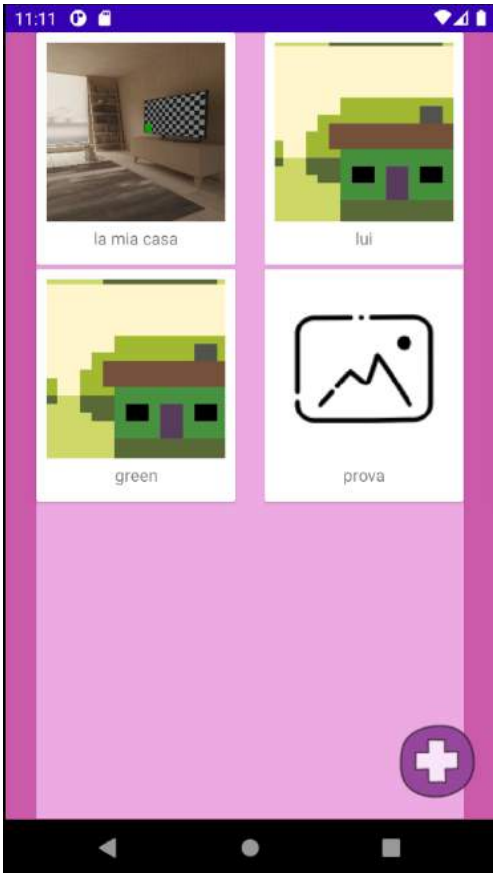

Figura 4.32 : Layout MainActivityPolaroid

4.3.10 - polaroid/NewPolaroidActivity

Figura 4.33 : Layout NewPolaroidActivity

Questa activity implementa il codice poter caricare una foto dalla galleria, salvarla e visualizzarla. Nell'immagine 4.34 nella funzione *onActivityResult()* è stato fatto l'override della funzione in modo tale da poter prendere in input la foto dalla galleria, decodificarla dall'oggetto *InputStream* in *Bitmap* e caricarla nell'oggetto *selectedImageView*. Questa funzione viene poi chiamata nelle funzione *openGallery()* che utilizzando l'oggetto intent e selezionando l'azione desiderata in questo caso per

aprire la galleria *Intent.ACTION_GET_CONTENT*, richiama infine *onActivityResult()* in modo tale da identificare l'immagine selezionata.

La funzione *save()* invece crea un oggetto *Polaroid* in cui viene salvata la foto caricata e la inerente didascalia immensa, e l'oggetto *Polaroid* viene aggiunto nel database della classe *PolaroidDbHelper*.

```
public void openGallery(View view) {
    Intent intent = new Intent();
    intent.setType("image/*");
    intent.setAction(Intent.ACTION_GET_CONTENT);
    startActivityForResult(Intent.createChooser(intent, title: "Select Picture"), GALLERY_REQUEST_CODE);
}

public void save(View view) {
    Bitmap image = ((BitmapDrawable)selectedImageView.getDrawable()).getBitmap();
    new PolaroidDbHelper( context: this).addMemory(new Polaroid(titleEditText.getText().toString(), image));
    finish();
}

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

    if (resultCode == RESULT_OK && requestCode == GALLERY_REQUEST_CODE) {
        try {
            Uri selectedImage = data.getData();
            InputStream imageStream = getContentResolver().openInputStream(selectedImage);
            selectedImageView.setImageBitmap(BitmapFactory.decodeStream(imageStream));
        } catch (IOException exception) {
            exception.printStackTrace();
        }
    }
}
```

Figura 4.34 : Funzioni NewPolaroidActivity

4.3.11 - Altre classi cartella polaroid/

Le classi rimanenti utilizzate per implementare la funzionalità dell'applicazione riguardante questo contesto di visualizzazione delle foto ricordo stile polaroid sono :

- **Polaroid** : il compito principale di questa classe è la creazione di un oggetto di tipo *Polaroid*, figura 4.35, contenente title e image entrambi di tipo *String*. La

foto di tipo *Bitmap* viene quindi prima ridimensionata con la funzione *resizeBitmap()* e successivamente convertita in *String* con la funzione *bitmapToString()*.

```
public Polaroid(String title, Bitmap image) {
    this.title = title;
    this.image = bitmapToString(resizeBitmap(image));
}
```

Figura 4.35 : Costruttore Polaroid

- **PolaroidContract** : estende la classe BaseColumns fornisce i nomi per le colonne _ID e _COUNT molto comuni. L'utilizzo di nomi comuni consente alla piattaforma Android (e anche agli sviluppatori) di indirizzare qualsiasi elemento di dati, indipendentemente dalla sua struttura complessiva (ovvero altre colonne non ID) in modo unificato. La definizione di costanti per le stringhe di uso comune in una classe evita ripetizioni e errori di battitura in tutto il codice. L'utilizzo di una colonna denominata _ID (il valore costante di PolaroidContract._ID) è richiesto da per esempio dalla classe CursorAdapter che estende la classe PolaroidAdapter.

```
public class PolaroidContract {

    //An empty private constructor makes sure that the class is not going to be initialised.
    private PolaroidContract() {}

    public static final class MemoryEntry implements BaseColumns {
        public static final String TABLE_NAME = "memories";
        public static final String COLUMN_TITLE = "title";
        public static final String COLUMN_IMAGE = "image";
    }
}
```

Figura 4.36 : PolaroidContract

- **PolaroidAdapter** : estende la classe CursorAdapter, il suo scopo è trasformare ogni riga del risultato della query in una View. il Cursors deve includere una colonna denominata "_ID" o questa classe non funzionerà. Il CursorAdapter opererà mediante override di due metodi in particolare:
    - newView : crea la View cui associare i dati prelevati dal database.
    - bindView: riceve in input una View, da completare con i dati di un singolo record del cursore, e il Cursor già posizionato sulla riga giusta. Grazie all'implementazione di newView, questo metodo riceverà sempre una View pronta, creata per l'occasione o "riciclata" da quelle già esistenti. Le operazioni dominanti in bindView riguarderanno il recupero di controlli presenti nella View ed il loro completamento con i dati presi dal Cursor.

```
@Override
public View newView(Context context, Cursor cursor, ViewGroup viewGroup) {
    View view = LayoutInflater.from(context).inflate(R.layout.memory_list_item, viewGroup,  attachToRoot: false);
    view.setTag(new ViewHolder(view));
    return view;
}

@Override
public void bindView(View view, Context context, Cursor cursor) {
    ViewHolder holder = (ViewHolder)view.getTag();

    Polaroid memory = new Polaroid(cursor);

    holder.titleTextView.setText(memory.getTitle());
    holder.imageView.setImageBitmap(memory.getImage());
}
```

Figura 4.37 : Funzioni PolaroidAdapter

- **PolaroidDbHelper** : estende la classe *SQLiteOpenHelper*, servirà a gestire la nascita e l'aggiornamento del database su memoria fisica e a recuperare un riferimento all'oggetto *SQLiteDatabase*, usato come accesso ai dati. Viene quindi il database in cui salvare gli oggetti *Polaroid*. I parametri immessi sono rispettivamente :
    - il nome del database : *PolaroidContract.MemoryEntry.TABLE_NAME*

- *PolaroidContract.MemoryEntry._ID* di tipo int impostata la constraint *"PRIMARY KEY"* che permette di distinguere univocamente ogni riga
- la didascalia della foto viene identificata con il parametro *PolaroidContract.MemoryEntry.COLUMN_TITLE*
- il path delle foto viene identificato con il parametro *PolaroidContract.MemoryEntry..COLUMN_IMAGE*

```
private static final String SQL_CREATE_ENTRIES =
        "CREATE TABLE " + PolaroidContract.MemoryEntry.TABLE_NAME + " (" +
                PolaroidContract.MemoryEntry._ID + INTEGER_TYPE + " PRIMARY KEY" + COMMA_SEP +
                PolaroidContract.MemoryEntry.COLUMN_TITLE + TEXT_TYPE + COMMA_SEP +
                PolaroidContract.MemoryEntry.COLUMN_IMAGE + TEXT_TYPE + " )";
```

Figura 4.38 : Definizione database PolaroidDbHelper

Questa classe inoltre implementa le funzioni *addMemory()* per aggiungere e memorizzare un nuovo oggetto di tipo *Polaroid* al database e *deleteMemory()* invece per implementare la funzione di eliminazione di un oggetto *Polaroid* dal database.

```
public boolean addMemory(Polaroid memory) {
    SQLiteDatabase db = getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(PolaroidContract.MemoryEntry.COLUMN_TITLE, memory.getTitle());
    values.put(PolaroidContract.MemoryEntry.COLUMN_IMAGE, memory.getImageAsString());

    return db.insert(PolaroidContract.MemoryEntry.TABLE_NAME,  nullColumnHack: null, values) != -1;
}

public void deleteMemory(int rowId){
     SQLiteDatabase db = getWritableDatabase();
    db.delete(PolaroidContract.MemoryEntry.TABLE_NAME,  whereClause: PolaroidContract.MemoryEntry._ID + "=" + rowId,  whereArgs: null);
}
```

Figura 4.39 : Funzioni PolaroidDbHelper

4.3.12 - reminder/

MainActivityReminder è un activity che implementa tramite l'utilizzo di una *ListView* e di una classe ***ListViewAdapter*** che estende *BaseAdapter* (classe base comune di implementazione per un adattatore che può essere utilizzato in *ListView*, implementando l'interfaccia *ListViewAdapter* specializzata), la funzione di visualizzazione dei promemoria salvati in un database creato con l'aiuto delle classe ***ReminderHelper*** che estende *SQLiteOpenHelper*, in cui i parametri del database sono *subject*, *content*, *time*, ovvero rispettivamente il titolo del promemoria, le informazioni aggiuntive del promemoria, la data e l'ora della notifica impostata.

Nella *ListView* di promemoria le informazioni visibili messe in primo piano sono *subject* e *time*.

Viene utilizzato quindi un procedimento analogo a quello utilizzato nella cartella *polaroid/*.

Anche qui toccando un promemoria è possibile eliminarlo o modificarlo, scegliendo un'opzione nella finestra di dialog che si aprirà al tocco. Premendo sull'*ImageButton* con il simbolo "+" invece si apre una nuova activity *AddActivityReminder* che permette di creare un nuovo promemoria.

Figura 4.40 : Layout MainActivityReminder

4.3.13 - reminder/AddActivityReminder

Nell'activity *AddActivityReminder* si aggiunge un nuovo promemoria, immettendo nelle apposite *EditTextView* le informazioni richieste ovvero *subject*, *content* e successivamente impostando data e ora della notifica premendo sull'apposito *ImageButton "SET TIME"*.

Figura 4.41 : Layout AddActivityReminder

Terminata la fase di compilazione, è necessario salvare il promemoria premendo su *"SAVE"*. Come possiamo vedere nella figura 4.41, viene creato quindi un oggetto ***Reminder*** con gli appositi parametri e se è un nuovo promemoria viene aggiunto all'oggetto database *reminderHelper*, altrimenti se è un promemoria già creato che si sta cercando di modificare, il promemoria precedente viene sostituito con quello modificato.

```
private void onSave() {
    subject = et_subject.getText().toString();
    content = et_content.getText().toString();
    Reminder _reminder = new Reminder(subject, content, time, alarm: 0);
    ReminderHelper reminderHelper = new ReminderHelper( context: this);
    if (isNew) {
        long id = reminderHelper.insert(_reminder);
        _reminder.setId(id);
    } else {
        reminderHelper.update(reminder.getId(), _reminder);
    }
```

Figura 4.42 : Funzione onSave() AddActivityReminder

Nella funzione *onSave()* viene chiamata alla fine la funzione *setTimeNotification()* per abilitare la notifica. Viene calcolato la variabile *tot* ovvero l'intervallo di tempo allo scadere del quale deve arrivare la notifica (alla data e ora impostate dall'utente), e immesso come parametro nella chiamata della funzione *scheduleNotification()*. Il primo parametro di quest'ultima funzione invece è un oggetto *Notification*, in questo caso la funzione *getNotification()* in cui viene creata la notifica. L'*AlarmManager* nella funzione *scheduleNotificatio()* permette di attivare del codice al momento richiesto. Avrà bisogno di ricevere nostre istruzioni riguardanti le tempistiche e le modalità di attivazione ed un Intent che descrive l'azione da intraprendere. l riferimento *pendingIntent* invece rappresenta l'azione che deve essere svolta ad ogni allarme ed appartiene alla classe *PendingIntent*. Possiamo considerare questi oggetti dei contenitori di *Intent* che li conservano lasciandoli in sospeso fino al momento dell'attivazione.

Quando all'inizio di questa funzione viene creato l'oggetto *notificationIntent* con un componente *BroadcastReceiver [4]* ovvero ***AlarmReceiverReminder***, che ha la capacità di restare dormiente finché non viene attivato da un messaggio (in questo caso lanciato dall'*AlarmManager*).

```java
private void scheduleNotification(Notification notification, long delay) {
    Intent notificationIntent = new Intent( packageContext: this, AlarmReceiverReminder.class);
    notificationIntent.putExtra(AlarmReceiverReminder.NOTIFICATION_ID,  value: 1);
    notificationIntent.putExtra(AlarmReceiverReminder.NOTIFICATION, notification);
    PendingIntent pendingIntent = PendingIntent.getBroadcast( context: this,  requestCode: 0, notificationIntent, PendingIntent.FLAG_UPDATE_CURRENT);
    long futureInMillis = SystemClock.elapsedRealtime() + delay;
    AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
    assert alarmManager != null;
    alarmManager.set(AlarmManager.ELAPSED_REALTIME_WAKEUP, futureInMillis, pendingIntent);
}

private Notification getNotification() {
    subject = et_subject.getText().toString();
    content = et_content.getText().toString();
    Reminder _reminder = new Reminder(subject, content, time,  alarm: 0);
    NotificationCompat.Builder builder = new NotificationCompat.Builder( context: this, default_notification_channel_id);
    builder.setContentTitle(_reminder.getSubject());
    builder.setContentText(_reminder.getContent());
    builder.setSmallIcon(R.drawable.ic_launcher_foreground);
    builder.setAutoCancel(true);
    builder.setChannelId(NOTIFICATION_CHANNEL_ID);
    return builder.build();
}

public void setTimeNotification() {
    calendar = Calendar.getInstance();
    long ct = calendar.getTimeInMillis();

    calendar.setTime(time);
    long timeinmillis;
    timeinmillis= (long) calendar.getTimeInMillis();
    long tot = timeinmillis-ct;

    scheduleNotification(getNotification(), tot);
}
```

Figura 4.43 : Funzioni AddActivityReminder

Il BroadcastReceiver citato sopra, lo possiamo visualizzare nella figura sottostante. Il BroadcastReceiver viene creato come una normale classe Java che vede al suo interno solo un metodo, *onReceive*, contenente il codice da attivare al momento della ricezione di un Intent. Questo *Intent* da inviare sarà consegnato all'*AlarmManager* incapsulato nel *PendingIntent* nella classe *AddActivityReminder*. Il metodo *onReceive* riceverà in input un riferimento al *Context* per poter avviare qualunque richiesta nel sistema ed un Intent, quello con cui è stato attivato il *Receiver*.

Per inviare la notifica è stato utilizzato *NotificationManager*, un servizio di sistema, specializzato nel far apparire notifiche nella barra di stato del dispositivo.

```java
public class AlarmReceiverReminder extends BroadcastReceiver {

    public AlarmReceiverReminder(){}
    public static String NOTIFICATION_ID = "notification-id" ;
    public static String NOTIFICATION = "notification" ;
    @Override
    public void onReceive(Context context, Intent intent) {
        NotificationManager notificationManager = (NotificationManager) context.getSystemService(Context. NOTIFICATION_SERVICE ) ;
        Notification notification = intent.getParcelableExtra( NOTIFICATION ) ;
        if (android.os.Build.VERSION. SDK_INT >= android.os.Build.VERSION_CODES. O ) {
            int importance = NotificationManager. IMPORTANCE_HIGH ;
            NotificationChannel notificationChannel = new NotificationChannel( NOTIFICATION_CHANNEL_ID ,  name: "NOTIFICATION_CHANNEL_NAME" , importance) ;
            assert notificationManager != null;
            notificationManager.createNotificationChannel(notificationChannel) ;
        }
        int id = intent.getIntExtra( NOTIFICATION_ID ,  defaultValue: 0 ) ;
        assert notificationManager != null;
        notificationManager.notify(id , notification) ;
    }
}
```

Figura 4.44 : AlarmReceiverReminder

Capitolo 5

Conclusioni e sviluppi futuri

5.1 - Obiettivo raggiunto?

La progettazione e le varie fasi di sviluppo dell'applicazione hanno portato alla realizzazione di Ricordella. In questo percorso di autoapprendimento è stata creata una nuova applicazione nata con lo scopo di aiutare le persone affette da alzheimer, ma che fondamentalmente può essere utile a tutti. Infatti l'applicazione ha varie funzionalità tra cui scrivere note, impostare promemoria per qualsiasi cosa, visualizzare foto ricordo con una piccola didascalia, poter allenare la memoria giocando, ma anche funzioni utili in caso di emergenza, come tenere sempre salvate in un'unica applicazione informazioni personali dell'utente da poter utilizzare in qualsiasi occasione e anche la possibilità di chiamare premendo un solo tasto il 118 o inviare una richiesta di aiuto ad una persona scelta come contatto di emergenza inviandole un messaggio automatico che contiene anche la propria posizione e inoltre avviando contemporaneamente una chiamata. Quindi, anche se è un'applicazione pensata principalmente per una determinata categoria di persone, potrebbe essere utile a tutti gli utenti, avendo la possibilità di cercare aiuto in qualsiasi eventualità, magari in modo più veloce.

Concludendo, gli obiettivi prefissati nella fase di progettazione dell'applicazione sono stati raggiunti, però ci sono alcune sezioni che possono essere migliorate.

Una delle prime migliorie sarebbe aggiungere nella sezione promemoria, non solo una notifica per ricordare in promemoria, ma un allarme. Inoltre sarebbe utile aggiungere dei "promemoria dei medicinali", che permettono di impostare che tipo di medicinale bisogna prendere (pillola, sciroppo, etc..), ogni quanto ripetere il promemoria, aggiungere una foto del medicinala...

Nella sezione giochi invece si potrebbero aggiungere altri giochi per la memoria come sudoku, e implementare invece il gioco presente Memory, aggiungendo più livelli di difficoltà.

Nella sezione foto si potrebbe aggiungere la funzionalità di poter scattare la foto al momento invece di poterla aggiungere unicamente dalla galleria.

Nella sezione di emergenza magari si potrebbe aggiungere la funzione di inviare il messaggio con la richiesta di aiuto e la posizione a più utenti.

Ricordella ha una struttura ed un format adattabile a molte situazioni differenti ma è possibile migliorarla ulteriormente in diversi aspetti.

Nelle prospettive future l'idea è quella di migliorare l'applicazione magari per poterla un giorno pubblicare e rendere utilizzabile dagli utenti Android che ne avranno necessità.

Bibliografia

[1] “Alzheimer, arriva Chat Yourself, un’app che aiuta i pazienti” :
https://www.osservatoriomalattierare.it/alzheimer/13947-alzheimer-arriva-chat-yourself-un-app-che-aiuta-i-pazienti

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

[3] “Il diagramma dei casi d’uso” :
https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjV7cmx0PPvAhUUO-wKHWaxCRwQFjABegQIBhAD&url=http%3A%2F%2Fwww.cs.unibo.it%2F~zuppirol%2Fcasiduso.pdf&usg=AOvVaw1MZuMTS9JDHvONKaGKrrqY

[4] “Gestire allarmi e notifiche nelle applicazioni Android” :
http://www.devapp.it/wordpress/gestire-allarmi-e-notifiche-nelle-applicazioni-android/