Report Applicazione per il progetto di Applicazioni Mobili
anno accademico: 2021-2022

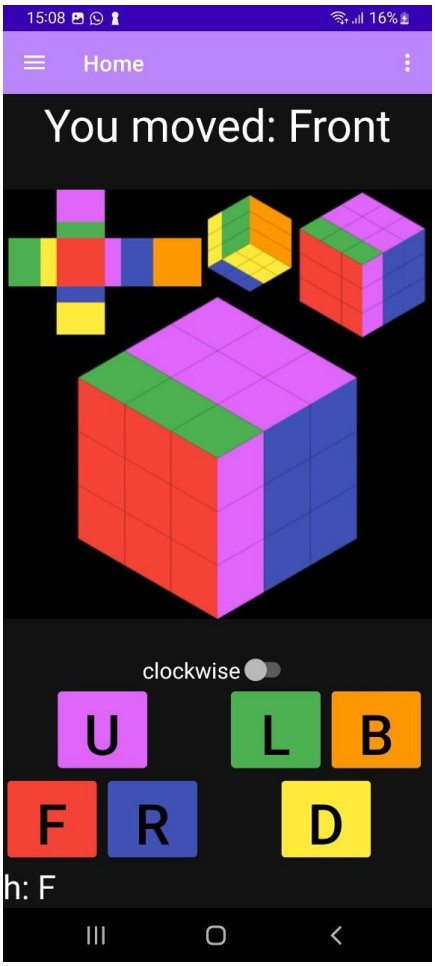
nome: Libera
cognome: Longo
email: libera.longo@studio.unibo.it
matricola: 0000839018

scopo dell'app:
imparare a risolvere un Cubo di Rubik

ambiente: Android Nativo (Java)

lingua di sviluppo: inglese
lingue in cui l'app viene tradotta: inglese, italiano, cinese semplificato.

Informazioni preliminari:
L'app si occupa di gestire un Cubo di Rubik.



Il cubo ha 6 facce chiamate:
Front (F), Right (R), Up (U), Back (B), Left (L), Down (D);
formata ciascuna da 9 tessere.

Su un Cubo di Rubik si possono fare queste operazioni:

- Una **mossa** di 90 gradi in senso orario ;
  F, R, U, B, L, D ; indicata con l'iniziale in inglese della faccia.
- Una **mossa** di 90 gradi in senso antiorario ;
  F', R', U', B', L', D' ; indicata con l'iniziale seguita dal carattere (').

Ruotare il cubo per vedere una determinata faccia in una certa posizione.
Ad esempio di solito si utilizza la faccia bianca sopra (bianca = Up)
nella mia app per questioni di estetica la faccia bianca è colorata di rosa.
Emulando la rotazione di un cubo reale si può ruotare a destra, sinistra o vedere la faccia gialla (opposta alla bianca, gialla = Down) sopra modificando la visualizzazione senza modificare la disposizione delle tessere.
Quindi gli algoritmi dopo aver chiesto di ruotare il cubo in un certo modo, da quel
momento in poi indicano la faccia sopra come Up, indipendentemente dal colore.

Scelta progettuale per gestire le rotazioni:

Tenere la visualizzazione più semplice possibile.
Lo storico è semplice ed è riferito sempre al "cubo non ruotato".

In questo modo è possibile imparare tutte le applicazioni che gestiscono il cubo di Rubik, anche quelle che non permettono di ruotare il cubo.

L'utente per fare la mossa è comunque facilitato perché vede le posizioni delle facce tramite le lettere e la faccia da muovere con il cubo "ruotato" in base al colore del centro della faccia.
Quindi se l'algoritmo dice "upside down, U" (ovvero "giallo è Up") su l'utente swappa opportunamente, vede che U è la faccia in alto dal nome dei bottoni, vede che il cubo ha la faccia in alto di colore giallo e quindi preme il bottone con il colore giallo, con la scritta D.
riassumendo: legge "upside down, U" → U è sopra → sopra è giallo → bottone giallo → D.

Per il progetto è stato utilizzato un **Navigation Drawer**:
Home: visualizza il Cubo “fermo” e lo SwapCube “swappabile”, e i comandi (bottoni, switch, menu di comandi).

Help: una breve spiegazione dell’app e cosa fare dopo aver letto una parte di un algoritmo del cubo di Rubik per poterlo eseguire.

Visualizza Algoritmo: per vedere lo SwapCube, i comandi e un algoritmo in formato txt o pdf.

Scrivi Algoritmo: l’utente può scrivere un algoritmo e salvarlo, oppure caricare l’algoritmo scritto la volta precedente.

Importa Algoritmo: dove si importa un pdf (o una pagina web, oppure ricerca un URI da visualizzare.

Invia e Ricevi: dove l’utente può scegliere di inviare un cubo salvato, un algoritmo txt, l’uri per l’algoritmo pdf o web.

Change Colors: dove l’utente può scegliere i colori del cubo per personalizzarlo a suo piacimento.

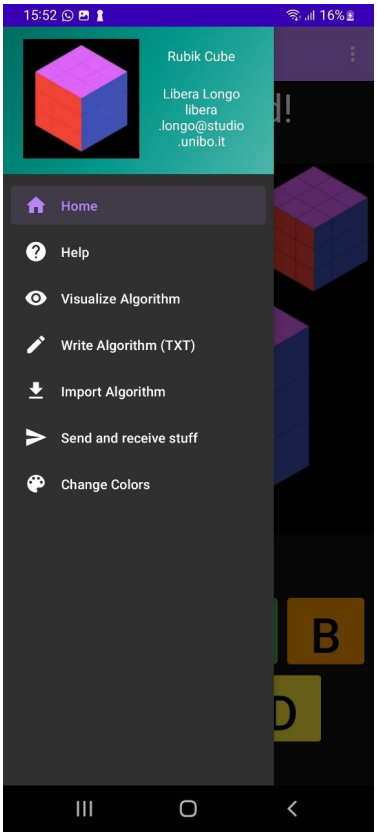
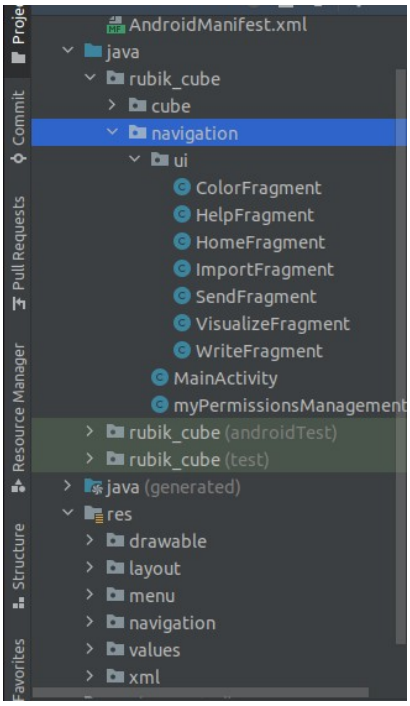
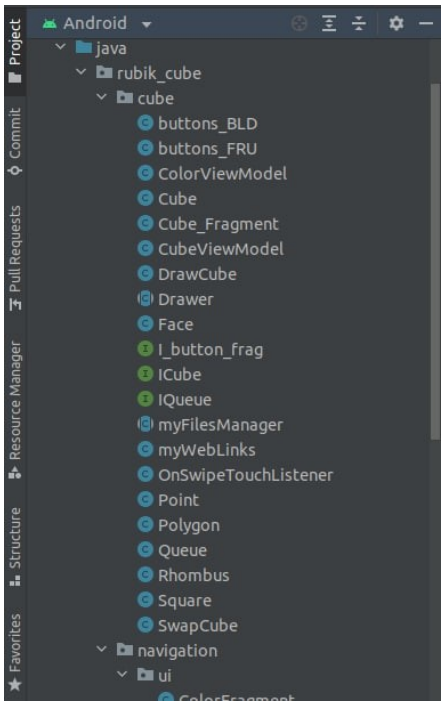


Fig: Navigation Drawer





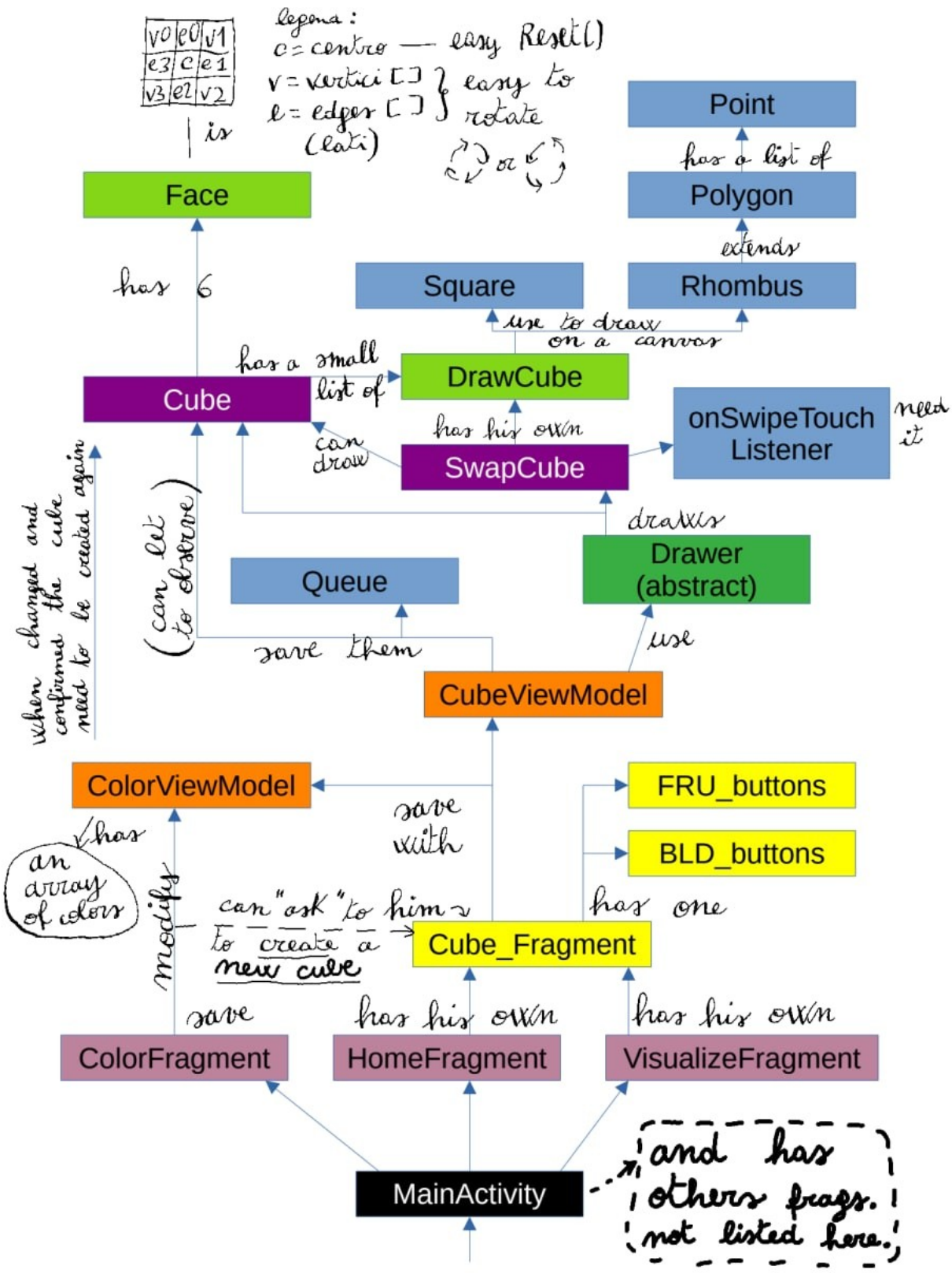
Organizzazione delle classi:
il package rubik_cube contiene tutto.
Il package navigation contiene MainActivity
e il package ui.
Ui contiene tutti i frammenti del Navigation Drawer
il package cube contiene tutte le classi ausiliarie, ovvero i viewModel (Cube, Color, myWebLinks),
il Cubo, i frammenti per visualizzare bottoni e per gestirlo, le classi per disegnarlo, la coda per la storia, le classi per gestire i file interni, lo SwapCube e il listener per lo swap, e le interfacce

Di seguito vengono riportate le scelte progettuali riferite ai singoli Fragment del Navigation Drawer.

Nota che Casa e Visualizza Algoritmo usano entrambe lo stesso Fragment per la gestione del cubo.

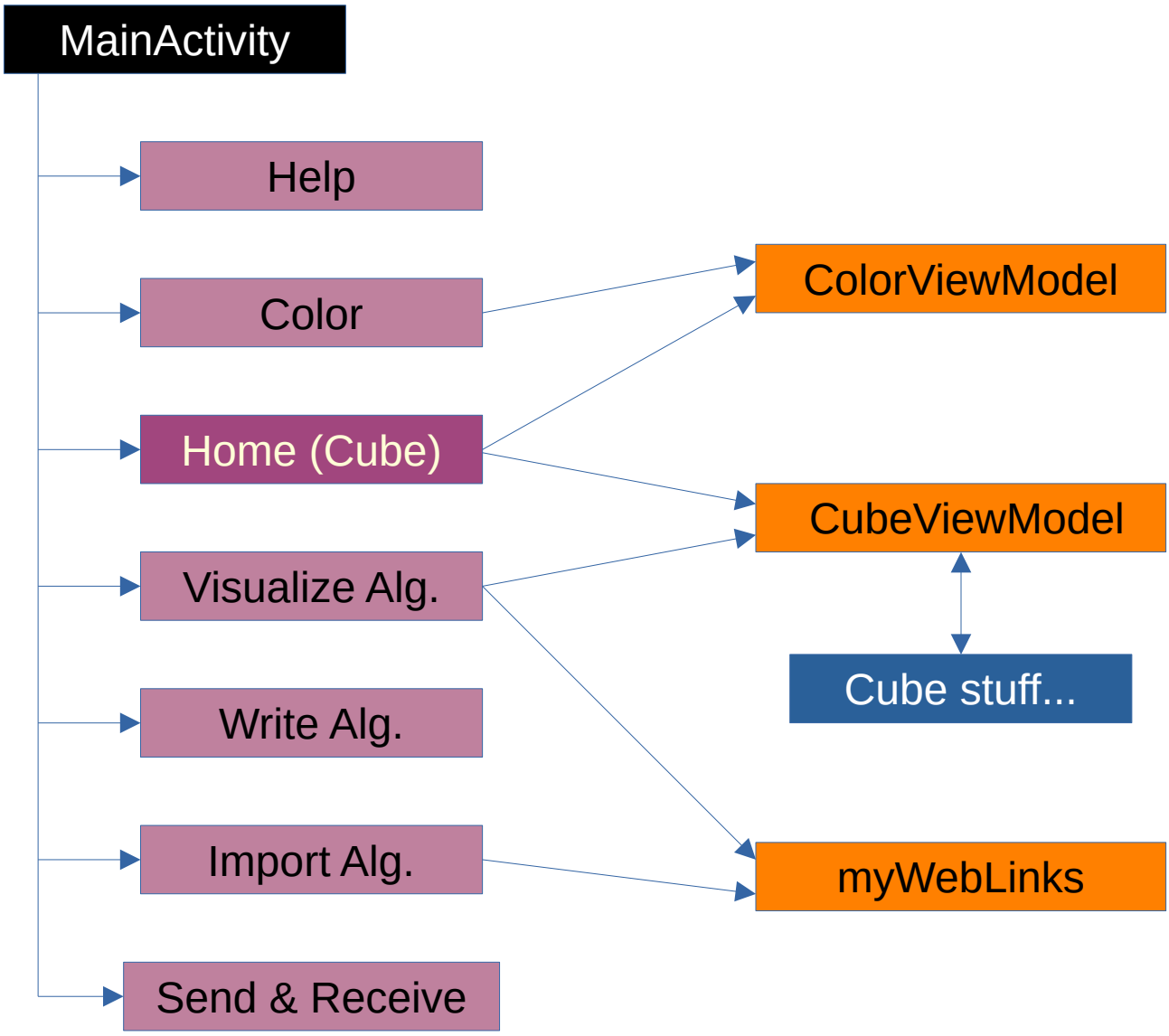
Help: una breve descrizione di cosa potrebbe fare l’utente, tradotta in tre lingue (inglese, italiano, cinese). Questo fragment è molto semplice, ho utilizzato una ScrollView e una TextView non avendo alcun evento da gestire.

Struttura delle classi per la gestione del Cubo di Rubik :

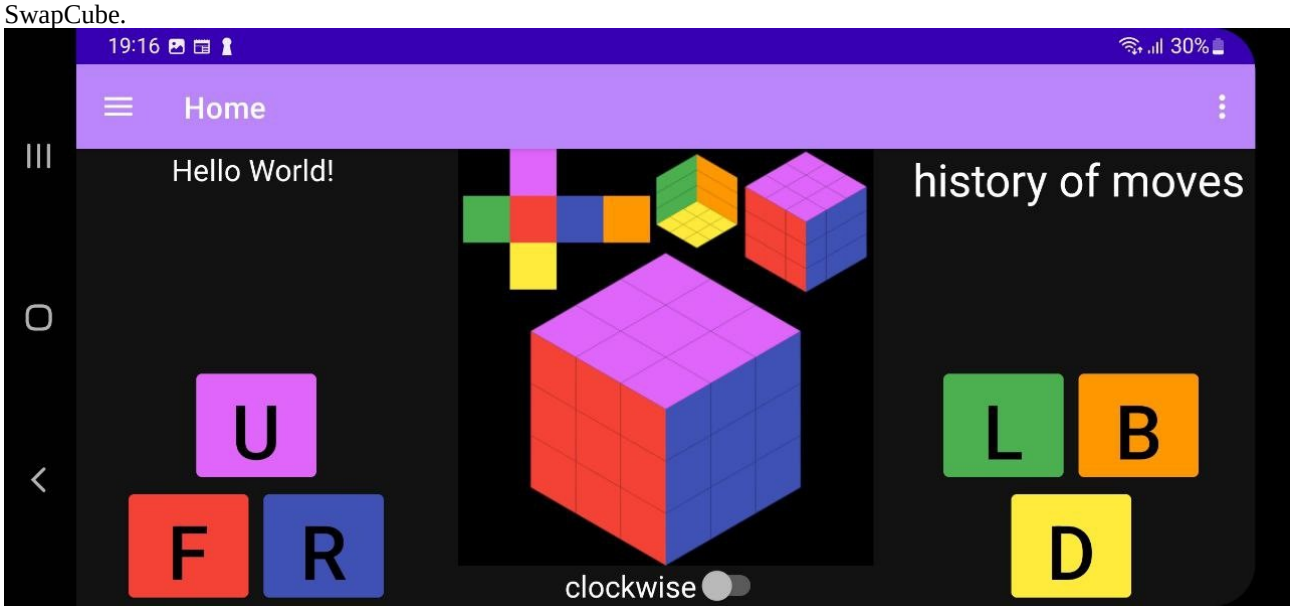


i colori significano: nero – Main Activity, che si preoccupa solo del Navigation Drawer.
rosa – è una frammento del NavigationDrawer, giallo - un frammento ausiliario,
arancione – è un ViewModel, viola è una classe molto importante,
verde chiaro – aiuta una classe importante, azzurro – è una classe ausiliaria,
verde scuro – è una classe astratta (con solo metodi statici, e nessun oggetto da salvare).

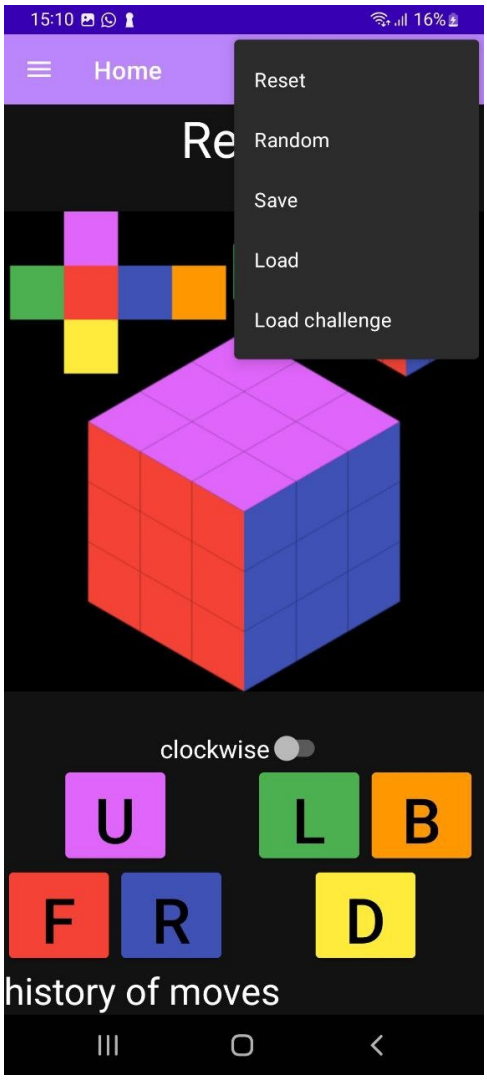
Schema di interazione tra i frammenti del Navigation Drawer e i ViewModels :



Home : utilizza il fragment “Cube_Fragment.java” che le consente di visualizzare sia il cubo fisso sia lo SwapCube.



Cube_Fragment.java



- utilizza CubeViewModel per poter osservare il Cubo ed altre classi ausiliarie come i drawers, lo swapCube e la coda delle ultime mosse.

- utilizza ColorViewModel per sapere quali sono i colori del cubo attualmente usati.
  Il cube Fragment per ricaricarli dai file salvati deve:
  - chiedere al CubeViewModel di caricare il cubo in memoria dai file
    - questo chiede la stringa del contenuto al myFilesManager e fa leggere la stringa alla funzione di lettura del Cubo e restituisce il vettore dei colori
  - salvare i colori nel Color View Model
  - chiedere ai frammenti che utilizzano i pulsanti di ricolorare i bottoni.

- utilizza due fragment per i bottoni:
  - il fragment “buttons_FRU” per i comandi Front, Right, Up che fanno le mosse F R U (visualizzati “davanti”), colorati come “colori di default” rosa, rosso e blu rispettivamente.
  - Il fragment “buttons_BLD” per i comandi Back, Left, Down che fanno le mosse B L D (visualizzati “dietro”), colorati come “colori di default” arancione, verde, giallo.

  Questi due fragment hanno codice molto simile e si occupano di gestire il comportamento e il colore dei bottoni, possono permettere al frammento del cubo di aggiornare i loro colori.

- Permette all’utente che preme il menu di:
  - Resettare il cubo alla configurazione “originale”;
  - Randomizzare facendo 10 mosse casuali;
  - Salvare il cubo nella memoria interna del telefono;
  - ricaricare il cubo salvato.
- Permette all’utente di caricare la sfida salvata da Invia e Ricevi.

Visualizza Algoritmo : utilizza il fragment “Cube_Fragment.java” che gli consente di visualizzare soltanto lo SwapCube, al centro della Canvas, più grande rispetto a quello di Home nella Canvas, che però ha meno spazio perché deve anche poter visualizzare l’algoritmo.

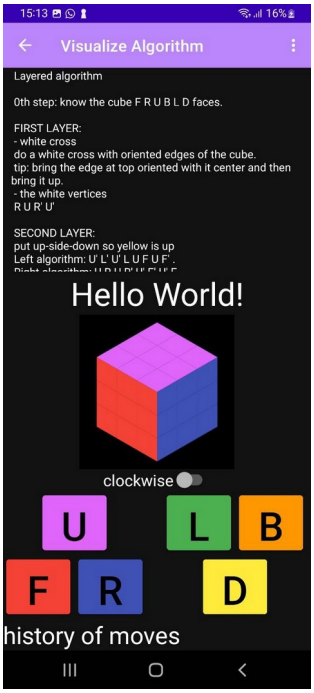


Fig. 1: default txt

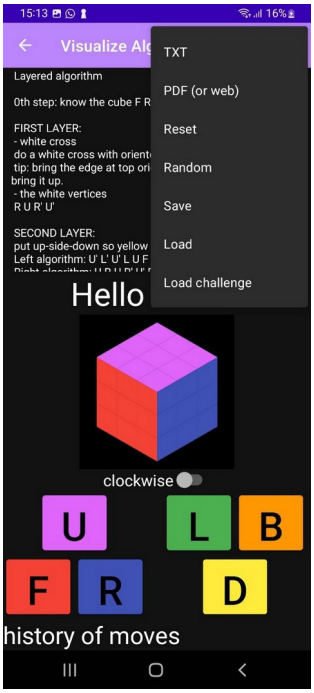


Fig. 2: menu

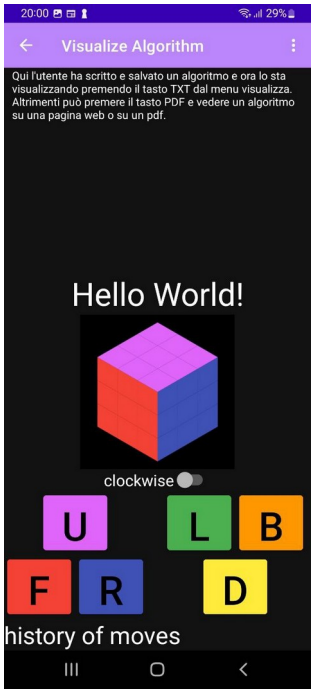


Fig. 3: TXT

Fig. 4: PDF (e web)

Scelte progettuali:

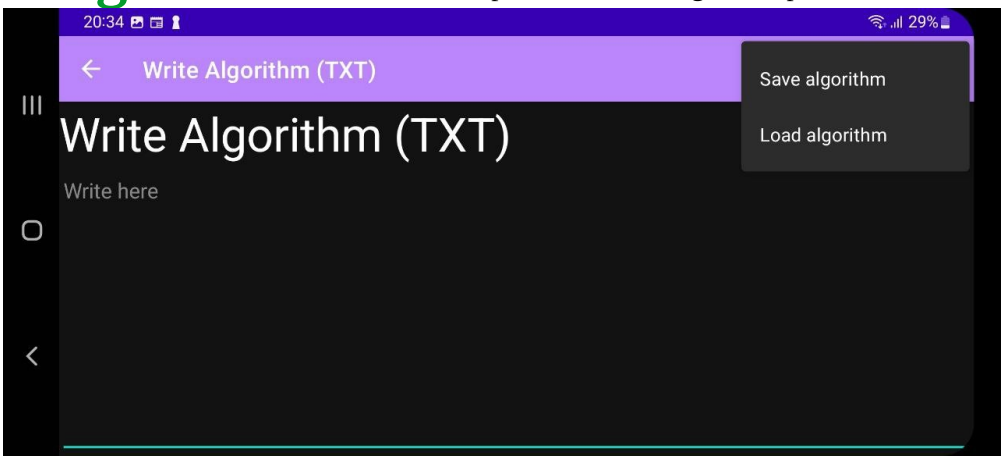
- Si è scelto di utilizzare una WebView per poter visualizzare algoritmi in formato PDF e web, questo perché le funzioni per le rispettive visualizzazioni sono molto simili.
  Infatti è necessario aggiungere solo una stringa davanti all’url per visualizzare correttamente il PDF.

```
@SuppressLint("SetJavaScriptEnabled")
public static void search(String uri, WebView webView) {
    String url = "";
    if(uri.endsWith(".pdf"))
        url += "https://drive.google.com/viewerng/viewer?embedded=true&url=";
    url += uri;
    webView.getSettings().setJavaScriptEnabled(true);
    webView.loadUrl(url);
    webView.setWebViewClient(new WebViewClient());
}
```

- I frammento viene sempre creato e ricreato con la visualizzazione dell’algoritmo, scritto in una stringa all’interno delle risorse (Fig.1).
  In questo modo viene sempre mostrato all’utente un algoritmo predefinito che funge da esempio, con lo scopo di suggerire all’utente di poter visualizzare un algoritmo, ovviando al rischio di dover leggere file inesistenti.
- Da qui l’utente può premere il menu e, oltre ai pulsanti gestiti dal “Fragment_Cube.java”, anche le opzioni “TXT” e “PDF (o web)” (Fig.2) .
- Utilizzo una WebView per visualizzare l’algoritmo in PDF e i link web.
- Utilizzo un TextView (all’interno di un anonimo ScrollView),
  per visualizzare l’algoritmo salvato in TXT.
- Quando viene scelto “TXT” viene resa visibile soltanto la TextView e si carica l’algoritmo salvato dall’utente nel file “write.txt” della memoria interna, dall’opzione Scrivi algoritmo (Fig. 3).
- Quando viene scelto “PDF (or web)” viene resa visibile solo la WebView e si carica l’URI del pdf salvato nel ViewModel “myWebLinks.java”, anche questo usato da più fragment, come “ColorViewModel”, senza la necessità di utilizzare un observer o un MutableLiveData.

Scrivi Algoritmo : dove l'utente può scrivere un algoritmo personalizzato in formato TXT.

Ha le opzioni salva algoritmo per salvarlo in memoria interna



chiamandolo "write.txt",
e ricarica algoritmo per richiamarlo dalla memoria interna, per poterlo modificare senza doverlo riscrivere.

Importa Algoritmo : Ha il compito di visualizzare e importare e scaricare un algoritmo in formato PDF, oppure visualizzare e importare un URI web.

Si fa uso del "myWebLinks" (un ViewModel senza observers e MutableLiveData), per memorizzare l'URI della pagina web da visualizzare nella WebView.
Ha queste opzioni nel menu:

- visualizzare la pagina Google per poter cercare un URI (Fig. 1).
- visualizzare una pagina di default ad esempio la pagina web di WikiHow (Fig. 2)
- visualizzare una pagina di default di un pdf ad esempio MasterCubo.com (Fig.3)
- importare un altro URI pdf o web scelto dall'utente (Scritto o Copiato).
- Visualizzare il PDF all'esterno della mia app chiamando un Intent ACTION_VIEW.

Può inoltre scaricare il file in formato PDF (solo pdf) nella cartella Download della memoria interna, ma questo serve per poter visualizzare il PDF all'esterno dell'applicazione aprendolo dalla cartella dei Download da tutte le altre app che possono visualizzarla. Non è necessario fare download per il funzionamento del resto dell'applicazione.

Fig 1: menu, Google

Fig 2: Wiki How

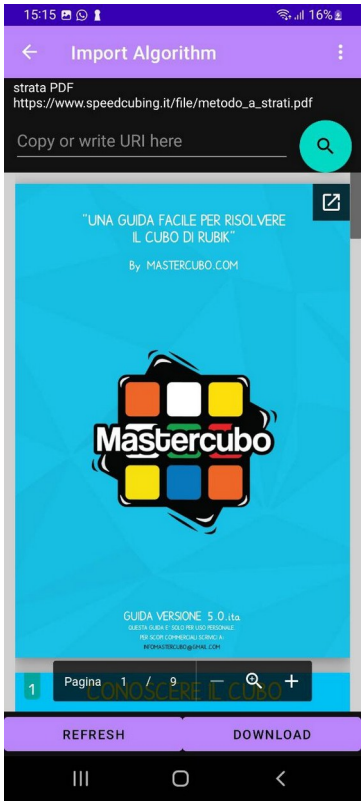


Fig 3: un pdf.

Invia e Ricevi : Ha il compito di inviare o ricevere da un’altra persona i file interni.

I file in questione sono:

- “cube.txt” : contenente una configurazione del cubo. Per inviare una configurazione “desiderata” è necessario: aprire l’app per la prima volta e inviarla, in questo caso il file interno di salvataggio non è ancora nella memoria interna quindi viene preso in automatico il cubo salvato all’interno del CubeViewModel, oppure salvare una configurazione e poi inviarla.
- “write.txt”: il file contenente l’algoritmo in formato TXT.

Inoltre può inviare l’URI del PDF o della web View memorizzata attualmente in “myWebLinks”.

Scelte progettuali:

invio un file .txt per cubo e algoritmo ed un URI per web e pdf.
Questo perché è sempre possibile copiare l’URI in Importa Algoritmo,
quindi è necessario inviarlo ma per riceverlo basta fare copia-incolla.

Per inviare salvo i due file nell’archivio, memoria interna del telefono (non la memoria SD) condivisa tra le applicazioni in Documents nella sottocartella rubik_cube creata dall’applicazione
path: /storage/emulated/0/Documents/rubik_cube/ ,
questo per fare in modo che se i file sono stati creati correttamente sia sempre possibile reinviarli da questo percorso.

Utilizzo un intent con ACTION_SEND (Fig. 2) e gli aggiungo i vari EXTRA necessari.
Come i file con MIME “text/plain” e le istruzioni per l’altro utente su come aprirli dalla sua applicazione.

Per ricevere mi aspetto che l’utente salvi i file esternamente nell’archivio,
memoria interna del telefono nella cartella Download, path: /storage/emulated/0/Download/ .
Poi apra l’app e prema sul pulsante ricevi corrispondente: ricevi Cubo se il file è “cube.txt” inviato dall’applicazione (o al massimo modificando i numeri che rappresentano i colori ma tenendo invariato tutto il resto), e ricevi TXT se si tratta di un algoritmo salvato in “write.txt”.

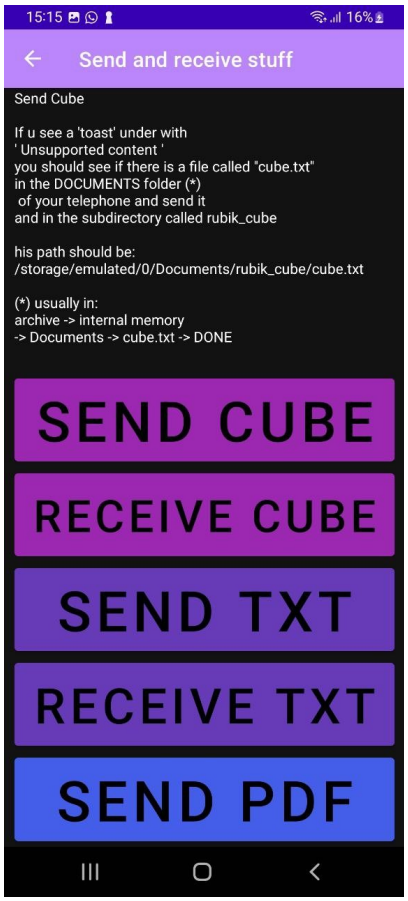


Fig. 1: send/receive buttons

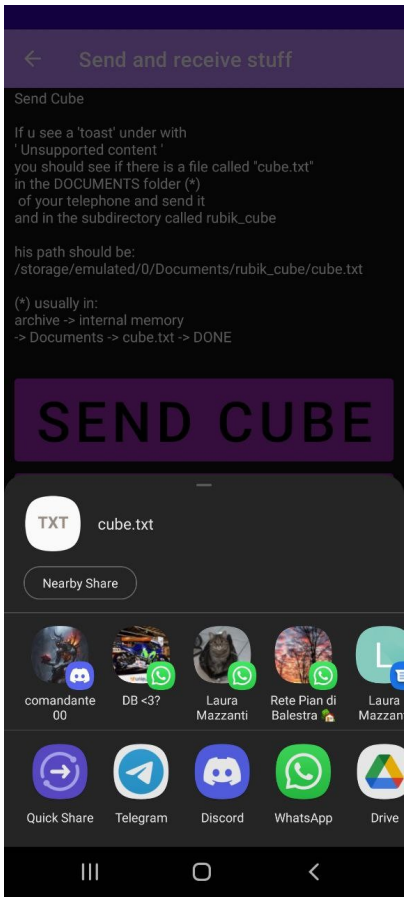


Fig. 2: send action

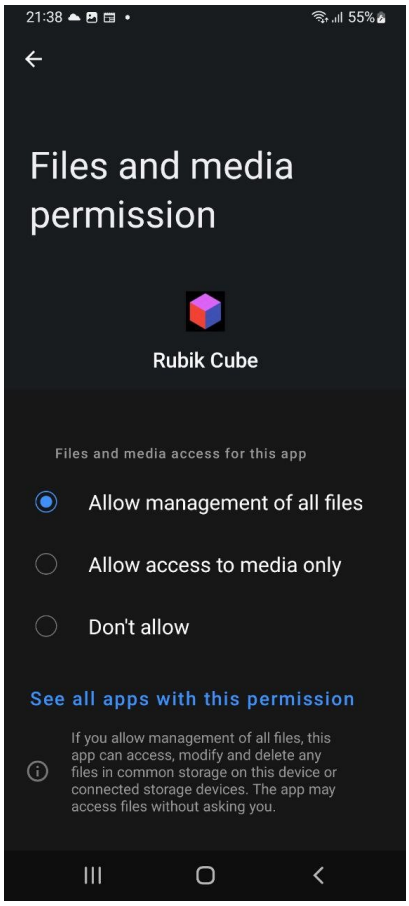


Fig. 3: MANAGE permissions for android 11+ (SDK 30+)

Cambia Colore : permette all'utente di cambiare i colori del cubo, per poterlo adattare ad un cubo che possiede in real life, personalizzarlo, o resettare i colori di default
(ma visto che il primo cubo al mondo era mono-colore
l'applicazione non si preoccupa di controllare quante facce abbiano lo stesso colore,
se si volesse un cubo con 1 <= N <= 6 facce colorate allo stesso modo
l'applicazione si limita a creare un cubo con le facce del colore richiesto).
Utilizza un array di bottoni, tutti con un Alert Dialog per scegliere un colore da un array di 6 colori. Questo permette di "mantenere" i colori usuali per i cubi di rubik in real life, che usano gli stessi sei colori senza mai cambiarli (ma rimane possibile "sceglierne altri" se si modifica il file "cube.txt" inviato dall'app e lo si carica con interi negativi differenti da quelli con cui è stato creato…).

```
//faces buttons behaviour
for (int i = 0; i < DIM; i++) {
    int face_choice = i;
    btnFaces[i].setOnClickListener(v -> {
        //FM12_navigation_architecture.pdf (slide 13)
        //Dialog: AlertDialog with a list
        final CharSequence[] items = getResources().getStringArray(R.array.colorName_array);
        AlertDialog.Builder builder = new AlertDialog.Builder(requireContext());
        builder.setTitle("Pick a color");
        builder.setSingleChoiceItems(items, checkedItem: -1, (dialog, color_index) -> {
            //for some reason when color_index = face_index and i already changed his value it not return back...
            default_colors[0] = getResources().getIntArray(R.array.color_array);
            //but why? it seem to be correct... isn't it?
            int color = default_colors[0][color_index];
            String msg = faces[face_choice] + " becomes " + color_name[color_index] ;
            tv.setText(msg);
            colour_model.set_Color_Index(face_choice, color);
            btnFaces[face_choice].setBackgroundColor(color);
        });
        AlertDialog alert = builder.create();
        alert.show();
    });
}
```

Dopo averli scelti, per applicare i colori modificati al cubo è necessario premere conferma.

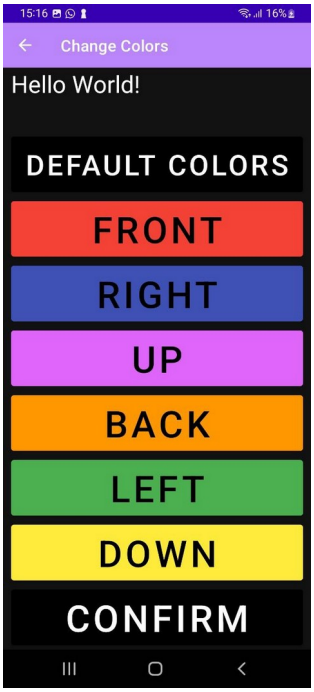


Fig.1: default colors

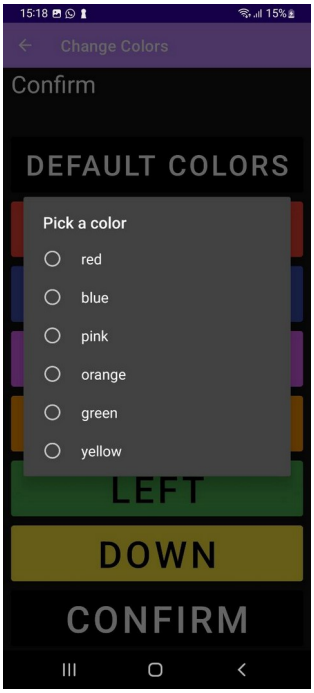


Fig. 2: dialog

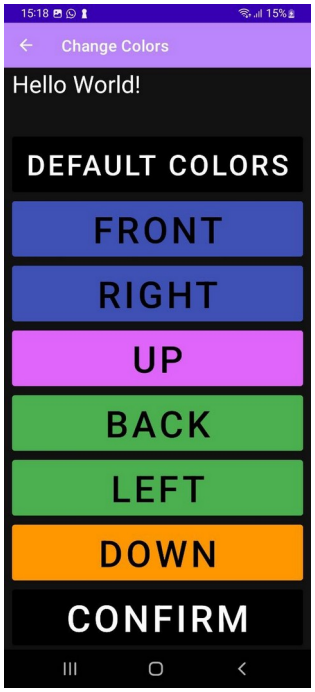


Fig. 3: una scelta

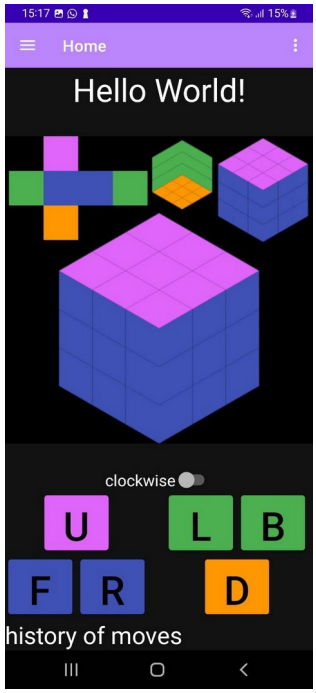


Fig. 4: il cubo nuovo

Difficoltà riscontrate:

per inviare e ricevere sono necessari i permessi READ, WRITE e un codice differente per MANAGE dei file per le versioni di Android 11+ con SDK 30+. Non capisco perché il programmatore è costretto a dover scrivere all'incirca 400 righe di codice di cui 100 solo per la richiesta dei permessi.

```
package rubik_cube.navigation;

import static
android.Manifest.permission.READ_EXTERNAL_STORAGE;
import static
android.Manifest.permission.WRITE_EXTERNAL_STORAGE;
import static android.os.Build.VERSION.SDK_INT;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.net.Uri;
import android.os.Environment;
import android.provider.Settings;
import android.util.Log;

import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

/** PERMISSIONS */
public class myPermissionsManagement {
        // look at
https://developer.android.com/training/permissions/requesting

        // In an educational UI, explain to the user why your app
requires this
        // permission for a specific feature to behave as
expected. In this UI,
        // include a "cancel" or "no thanks" button that allows
the user to
        // continue using your app without granting the
permission.
        private static boolean showInContextUI(Activity context,
String to_show) {
                final boolean[] permissionGranted = {false};
                AlertDialog.Builder builder = new
AlertDialog.Builder(context);

        builder.setMessage(to_show).setCancelable(false);
                builder.setPositiveButton("Yes, allow", (dialog,
id) -> {
                        //permissionGrantedNow
                        permissionGranted[0] = true;
                        //and exit dialog
                        dialog.cancel();
                });
                builder.setNegativeButton("No, tanks",
(dialog, id) -> {
                        //permissionStillDenied
                        permissionGranted[0] = false;
                        //and exit dialog
                        dialog.cancel();
                });
                AlertDialog alert = builder.create();
                alert.show();
                return permissionGranted[0];
        }

        public static boolean ask_permission(Activity activity,
String permission, int location, String why_app_need_it) {
                boolean permissionGranted = false;
                //i basically need only manage, read, write in
extern file for my app
                //then i check if permission is read or write
and handle the "manage" permission for android >= 11
                //if i need read and write i need also "manage"
                if
(permission.equals(READ_EXTERNAL_STORAGE) ||
permission.equals(WRITE_EXTERNAL_STORAGE)) {
                        Log.i("PERMISSIONS", "my sdk_int is
" + SDK_INT);
                        if (SDK_INT >= 30)
                                if (!
Environment.isExternalStorageManager()) {

        AlertDialog.Builder builder = new
AlertDialog.Builder(activity);

        builder.setMessage("Permission needed for managing
files!\n\n" + why_app_need_it).setCancelable(false);
                                                final boolean[]
manages_files = {false};

                builder.setPositiveButton("Settings", (dialog, id) -> {

                //permissionGrantedNow
                                                try {

                Uri uri = Uri.parse("package:" +
BuildConfig.APPLICATION_ID);

                Intent intent = new
Intent(Settings.ACTION_MANAGE_APP_ALL_FILES_ACCESS_PERMIS
SION, uri);

                activity.startActivity(intent);
                                                }
catch (Exception ex) {

                Intent intent = new Intent();

intent.setAction(Settings.ACTION_MANAGE_ALL_FILES_ACCESS_PE
RMISSION);
 ity.startActivity(intent);
 }
 ges_files[0] = true;
 og.cancel();
 });
 lder.setNegativeButton("No, tanks", (dialog, id) -> {
 rmissionStillDenied
 ges_files[0] = false;
 og.cancel();
 });
 AlertDialog alert = builder.create();
 alert.show();

 //now if i can't manage files exit, else continue to see others
permissions (read or write probably)
 if(!manages_files[0]) return false;
 }
 }
 //manage the remaining read or write permission
 if (ContextCompat.checkSelfPermission(activity, permission) ==
ckageManager.PERMISSION_GRANTED) {
 // You can use the API that requires the permission.
 permissionGranted = true;
 } else if
(ActivityCompat.shouldShowRequestPermissionRationale(activity,
permission)) {
 // In an educational UI, explain to the user why your app requires
this
 // permission for a specific feature to behave as expected. In this
UI,
 // include a "cancel" or "no thanks" button that allows the user to
 // continue using your app without granting the permission.
 permissionGranted = showInContextUI(activity,
why_app_need_it);
 } else {
 // You can directly ask for the permission.
 ctivityCompat.requestPermissions(activity, new String[] {
 permission }, location);
 }
 return permissionGranted;
 }
}
```

Tutto questo codice con il solo e unico scopo di ottenere un booleano:

- true se si posso leggere sui file perché l'utente me lo ha permesso oppure
- false perché l'utente non mi ha permesso uno dei permessi read write o manage files nella memoria interna del telefono.