Android
FPR-IOL2018

WALLPAPER

Università di Roma “Tor Vergata”
Ingegneria informatica

Ferraro Daniele, Geremia Petricca, Alessio Ruffini

Wallpapers

Wallpapers … di che cosa si parla?

Sono immagini che a scopo decorativo possono essere poste su un qualsiasi dispositivo mobile come sfondo:

- per la schermata iniziale
- e/o per la schermata di blocco

Permettono all'utente di personalizzare il proprio dispositivo.

Ogni telefono o tablet, viene fornito con una selezione di sfondi incorporati.

Wallpapers

Le **AZIONI** che si possono intraprendere sui wallpapers sono le seguenti:

- Settaggio wallpaper di default
- Importazione wallpaper da fonti esterne(es. Web)
- Creazione wallpaper personalizzati
- Ottenere informazioni sui wallpapers(es. colori dominanti)

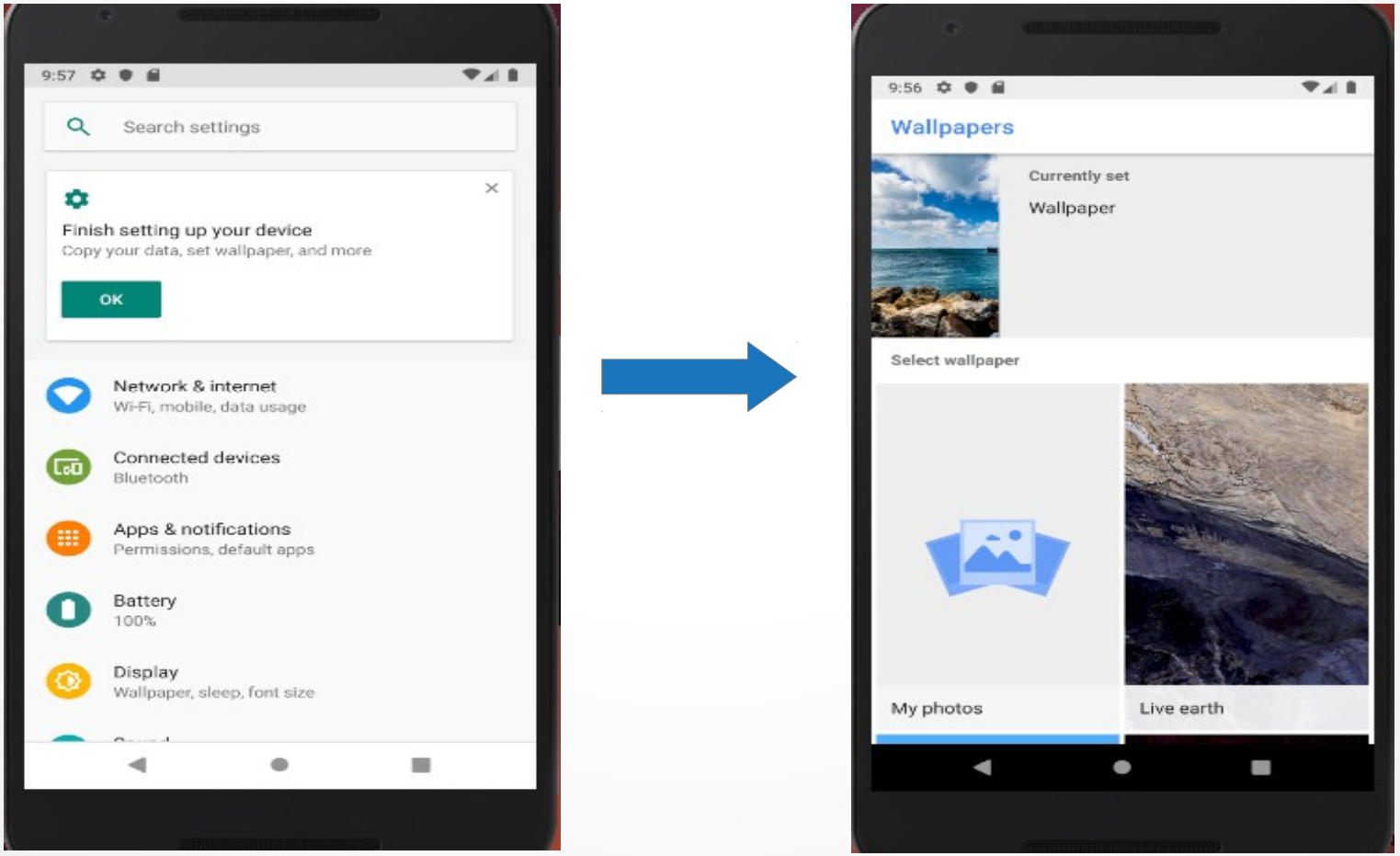
Fondamentalmente esistono 2 MODI per compiere queste azioni:

- Attraverso le **app di sistema**
- Utilizzando le **classi fornite dalla libreria Android**:
- public final class WallpaperManager added in API level 5
- public final class WallpaperColors added in API level 27

Wallpapers

APP DI SISTEMA

Navigando tra le impostazioni di sistema è possibile arrivare all'App di default che ci permette di selezionare, configurare ed impostare il wallpaper per le schermate Home e di blocco del dispositivo.



Wallpapers

CLASSI LIBRERIA ANDROID

E’ anche possibile interagire con i wallpaper implementando un’attività che sfrutta le classi fornite dalla libreria android.app. :

WallpaperManager

Fornisce accesso allo sfondo del sistema : con essa è possibile ottenere lo sfondo corrente, le dimensioni desiderate dello sfondo, impostare lo sfondo corrente partendo da diversi tipi di risorse (.jpg, .png, bitmap), cancellare lo sfondo ed altro.

Alcuni dei metodi più comunemente utilizzati sono :

- *getIstance(Context context)*: recupera un WallpaperManager associato al contesto
- *getDrawable()*: recupera lo sfondo del sistema corrente

Wallpapers

- *setResource(int resid)* : cambia lo sfondo corrente del sistema con la risorsa fornita

Il S.O. non ottimizza da se i wallpapers: la responsabilità della loro funzionalità è delegata al programmatore

WallpaperColors

Introdotta con android Oreo 8.1, permette alle app esterne di ottenere informazioni sui colori di un Wallpaper, attraverso i seguenti metodi:

- *getPrimaryColor()*: restituisce il più rappresentativo colore del wallpaper
- *getSecondaryColor()*: restituisce il 2° colore più dominante del wallpaper
- *getTertiarycolor()*: restituisce il 3° colore più dominante del wallpaper

Per esempio può essere utile per impostare il colore dei widgets della propria app uguale al colore dominante dello sfondo.

Wallpapers

PERMISSION

Per usufruire dei servizi offerti dalle classi sopra citate, l'app deve richiede alcuni permessi nel file Manifest dell'app.
A seconda dei metodi utilizzati, possono essere necessari più o meno permessi rispetto a quelli indicati nell'esempio seguente.

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

      <uses-permission android:name =
                  "android.permission.SET_WALLPAPER">
      </uses-permission>
      <uses-permission android:name =
                  "android.permission.WRITE_EXTERNAL_STORAGE">
      </uses-permission>
.....
```

Fino ad Android Mashmallow 6.0 (API level 23) tali permessi dovevano essere dichiarati solo nel Manifest.

Wallpapers

Da Mashmallow in poi invece, oltre a dichiararle nel Manifest, l'app dovrà richiedere le **autorizzazioni** necessarie all'utente in fase di **runtime**.

Inizialmente si controlla se l'app possiede i permessi :

```
public class MainActivity extends AppCompatActivity implements
                        AdapterView.OnItemClickListener {
  ……….
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        gridView = findViewById(R.id.gridView);
        imageAdapter = new ImageAdapter(this);
        gridView.setAdapter(imageAdapter);
        gridView.setOnItemClickListener(this);
       ……...
        /*Controllo se l'app ha il permesso: se non lo ha*/
        if (ContextCompat.checkSelfPermission(this,
                  Manifest.permission.WRITE_EXTERNAL_STORAGE) !=
                                        PackageManager.PERMISSION_GRANTED) {
            /*se non si possiede la relativa permission,bisogna chiedere
            all'utente di concederla esplicitamente, mostrando una finestra di
            dialogo*/
            ActivityCompat.requestPermissions(this,
                  new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
                                              REQUEST_ID);
            }
        }
        else { /*permesso già concesso, prosegui con il lavoro*/}
     }
……...
```

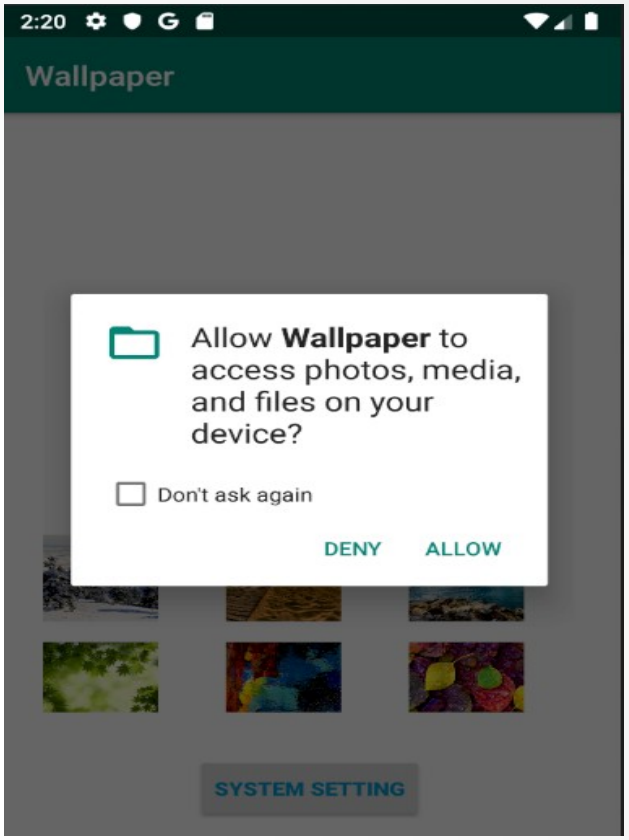
Wallpapers

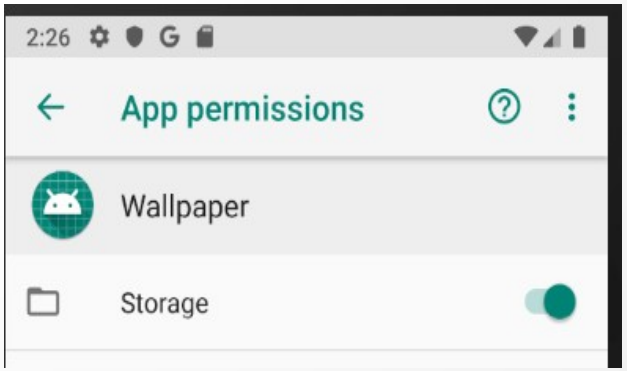
Se l'app non è in possesso dei permessi necessari, con il metodo **ContextCompat.requestPermission()** attraverso una finestra di dialogo viene richiesto all'utente di concederli esplicitamente.

Una volta concesse, le autorizzazioni non verranno più richieste, perché il S.O. memorizza le impostazioni.

Tuttavia l'utente potrà revocarle in qualsiasi momento accedendo alle impostazioni di sistema.

Se invece l'utente rifiuta di concedere le autorizzazione richieste, l'app dovrà gestire tale scelta.





Wallpapers

La scelta effettuata dall'utente (attraverso la finestra di dialogo) di concedere o meno le autorizzazioni all'app richiedente, sarà gestita nella **callback** *onRequestPermissionsResult()*:

```
@Override
public void onRequestPermissionsResult(int requestCode, String permissions[],
                                       int[] grantResults) {
    switch (requestCode) {
        case REQUEST_ID: {
            if (grantResults.length > 0
                    && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                /*permission accettata: possiamo attivare il codice*/

                updateWallpaper();

            }
            /*permission negata: disattiviamo i servizi che ne hanno bisogno*/
            else {
                STATO_PERMISSION=-1;

            }
        }

    }
}
```

Wallpapers

APP WALLPAPER

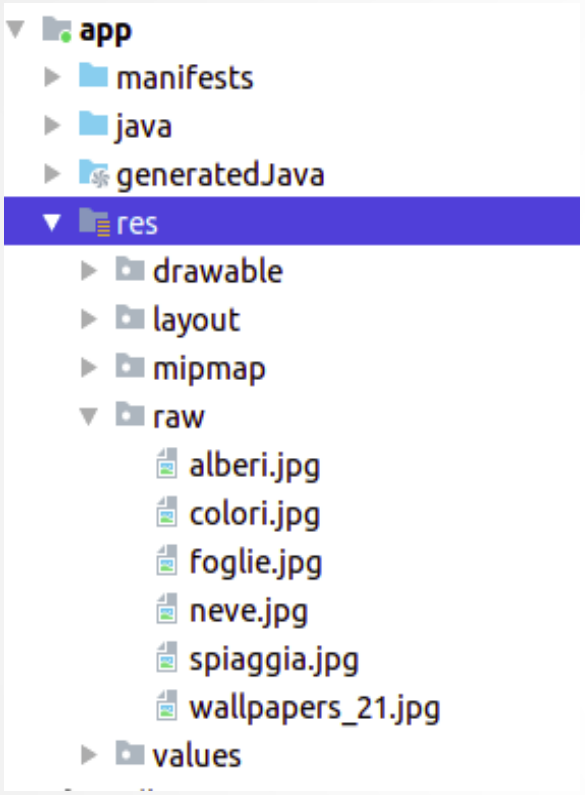
Per mostrare un esempio di utilizzo dei servizi citati, forniamo a corredo di tale presentazione l'app Wallpaper da noi sviluppata.

Tale app è stata progettata e testata per essere utilizzata con "minSdkVersion 19" e "targetSdkVersion 28": in quanto sono state implementate entrambe le richieste di permessi (manifest e runtime).

Una volta lanciata l'app, l'attività principale ha il compito di:

- visualizzare una griglia con tutte le risorse fornite. Viene utilizzata una gridView per mostrare le immagini che sono fornite da ImageAdaptor (il quale estende la classe BaseAdapter).
- L'adattatore preleva le immagini dalla cartella res/raw, nella quale sono state precedentemente salvate.

Wallpapers




```
public class ImageAdapter extends BaseAdapter {

    private Context context;

    private Integer[] array_idSfondi =
            {R.raw.neve, R.raw.spiaggia, R.raw.wallpapers_21,
            R.raw.alberi, R.raw.colori, R.raw.foglie, };

.....

    @Override
    public long getItemId(int position) {
        return array_idSfondi[position];
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;
        if(convertView == null){
            imageView = new ImageView(context);
            imageView.setLayoutParams(new ViewGroup.LayoutParams(200, 200));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8,8,8,8);
        }
        else imageView = (ImageView) convertView;
        imageView.setImageResource(array_idSfondi[position]);
        return imageView;
    }
}
```

Wallpapers

- Viene inoltre impostata una ImageView che mostra inizialmente (autorizzazioni permettendo) l'anteprima dello sfondo corrente, e successivamente quello selezionato dall'utente. Ecco il layout dell'activity:

- Per gestire "velocemente" le autorizzazioni, un Button permette di accedere (attraverso un intento implicito) direttamente alle impostazioni di sistema.

Wallpapers

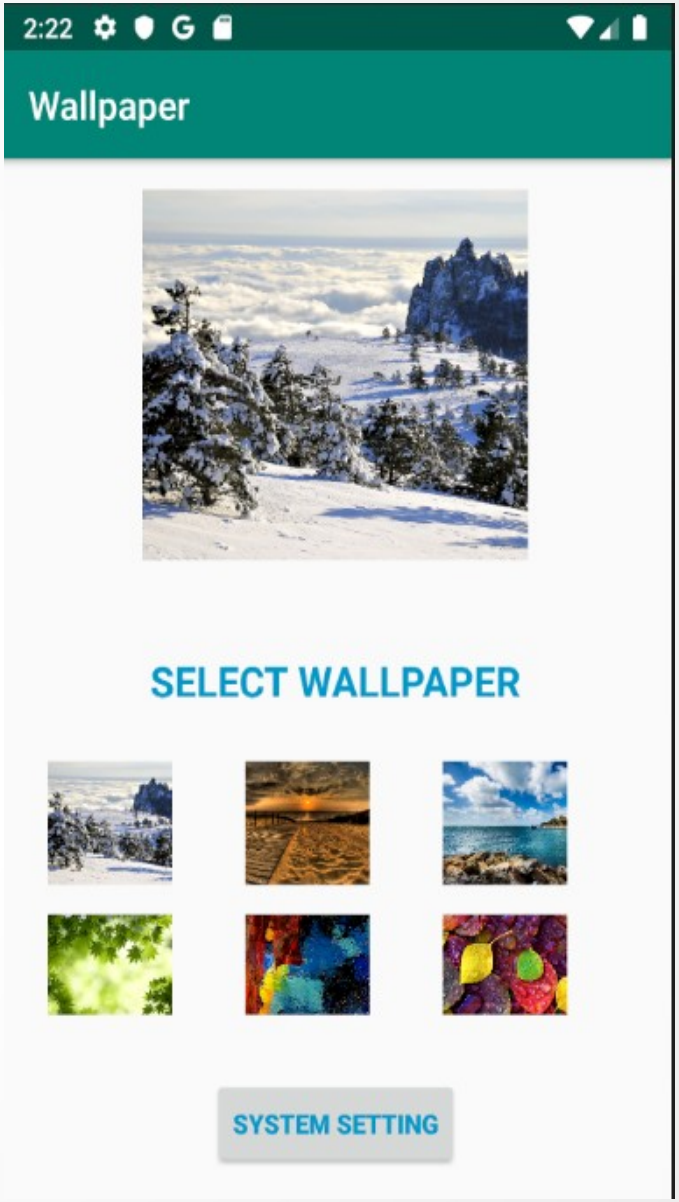
- Supponendo che l'app sia in possesso di tutte le autorizzazioni necessarie, la MainActivity chiama la procedura updateWallpaper(), la quale ottiene un'istanza di *WallpaperManager* attraverso il metodo statico ***WallpaperManager.getIstance()***.

- Sempre in updateWallpaper() viene poi chiamato il metodo ***getDrawable()***, dell'istanza sopra ottenuta, per recupere lo sfondo corrente del sistema. Questo viene restituito come oggetto *Drawable :* "astrazione generale per qualcosa che può essere disegnato".

- Infine, prima di lasciare la procedura updateWallpper(), con il metodo *ImageView.setImageDrawable()* l'activity imposta come contenuto dell'ImageView lo sfondo corrente appena recuperato.

Wallpapers

```
private void updateWallpaper() {

    /*getInstance() recupera 1 WallpaperManager associato con
    il dato contesto*/
    wallpaperManager =
            WallpaperManager.getInstance(getApplicationContext());

    drawable = wallpaperManager.getDrawable();

    if(drawable == null) {
        Log.d("Wallpaper : ", "Drawable = null");
    }
    /*imposta il wallpaper corrente come contenuto di questa
    ImageView*/
    imageView.setImageDrawable(drawable);
}
```



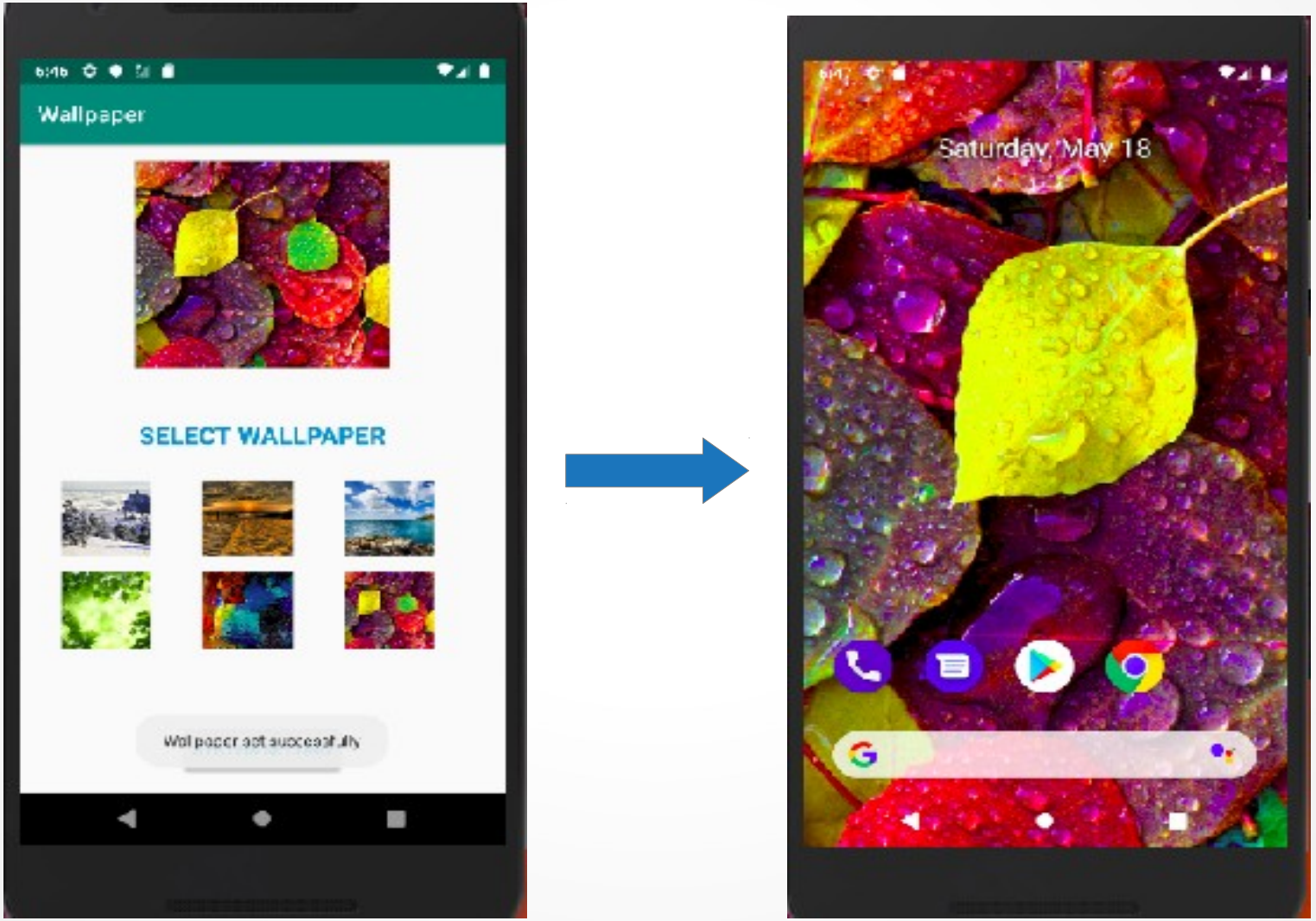
Wallpapers

- Infine, nella **callback** onItemClick() viene gestita la selezione dell'utente: nel momento in cui sceglie un determinato wallpaper, questo sarà impostato sia nella ImageView dell'app che come sfondo della schermata Home.

```
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {


    imageView.setImageResource((int)imageAdapter.getItemId(position));


    /* Se l'utente non si ricorda di aver negato l'autorizzazione e prova ad impostare uno degli sfondi
     * verrà avvisato che non lo può fare. Quindi continuerà a vedere solo e soltanto
     * l'ultimo sfondo impostato precedentemente */
    if (STATO_PERMISSION!=0){
        /* Allora informa l'utente che non può cambiare lo sfondo */
        DialogNoPermission mydialog = new DialogNoPermission();
        mydialog.show(getSupportFragmentManager(), tag: "mydialog");
    }else {
        /* Puoi impostare lo sfondo selezionato */
        try {
            wallpaperManager.setResource((int) imageAdapter.getItemId(position));
        } catch (IOException e) {
            e.printStackTrace();


        }
    }
}
```

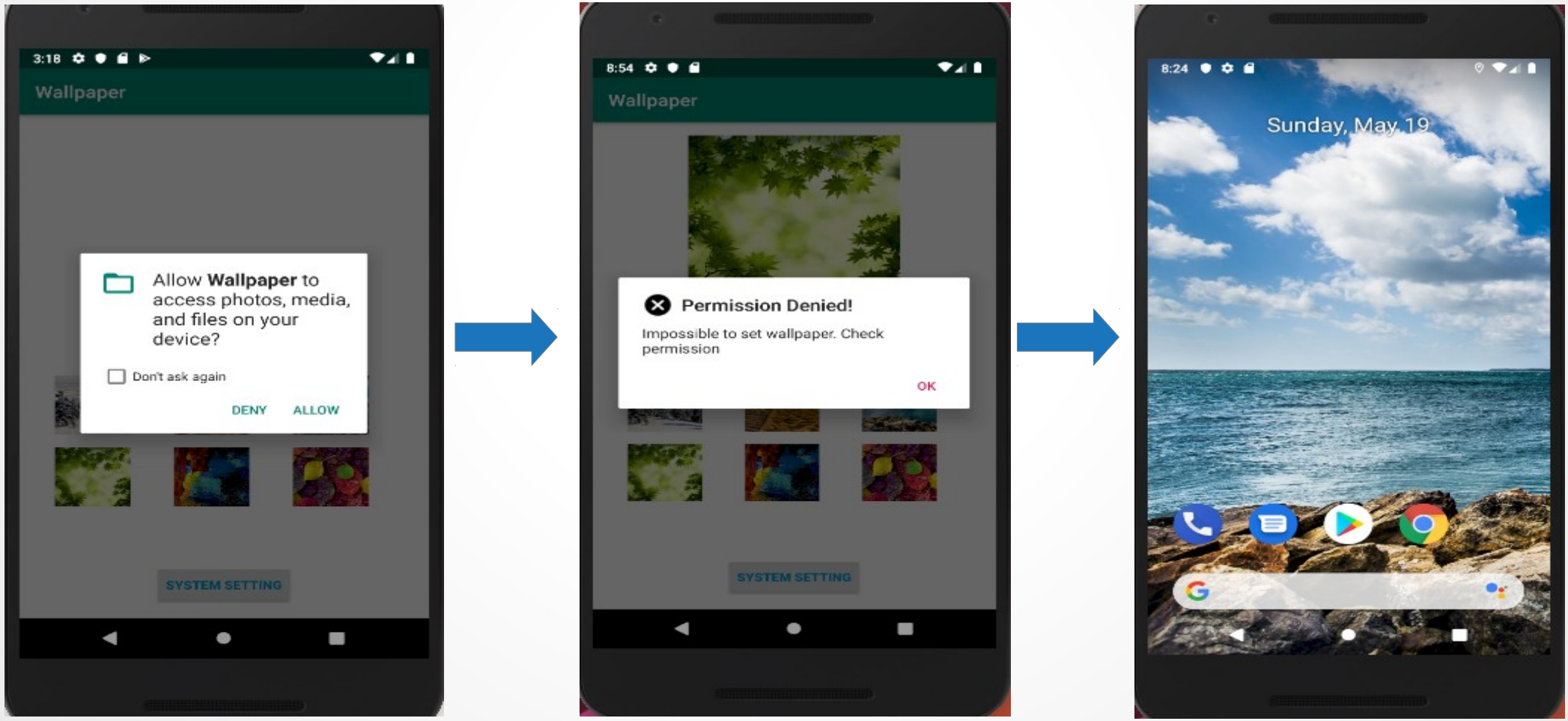
Wallpapers

- Attraverso il metodo ***wallpaperManager.setResource()*** viene effettivamente impostato lo sfondo corrente del sistema con l'immagine selezionata dall'utente.



Wallpapers

- Se l'App non è in possesso delle autorizzazioni necessarie (es. l'utente non le ha concesse a runtime) e si proverà a selezionare uno sfondo, apparirà una finestra di dialogo che specifica che l'app non detiene i permessi; di conseguenza lo sfondo corrente rimarrà l'ultimo impostato precedentemente.



Wallpapers

“Estensione” dei Wallpaper sono i **Live Wallpaper** (sfondo animato):
si differenziano da un wallpaper usuale per il **contenuto dinamico**.

Un Live Wallpaper è un servizio Android che può essere realizzato a partire da una gif animata (successivamente trasformata in un oggetto di tipo Movie) o tramite funzioni che “disegnano” ripetutamente figure geometriche in posizioni differenti dello schermo, dando un effetto di animazione.

La libreria *android.service.wallpaper* fornisce le seguenti classi per la gestione dei live Wallpaper aggiunte in API level 7 :

- *WallpaperService* : estende Service, ed è responsabile della visualizzazione del live wallpaper dietro le applicazioni
- *WallpaperService.Engine* : l’attuale implementazione del live wallpaper
- *WallpaperInfo* : Utilizzata per specificare le meta informazioni di un live wallpaper

Per approfondimenti consultare https://developer.android.com/

Wallpapers

RIFERIMENTI:

- https://developer.android.com/
- https://stackoverflow.com/
- https://www.html.it/
- http://www.massimoregoli.com