

# DataVirus.it

Federico Rachelli

18 maggio 2020

L'app **DataVirus.it** é ispirata dal sito web omonimo. Fornisce all'utente una visualizzazione grafica dei dati del Dipartimento di Protezione Civile sull'andamento dell'epidemia di COVID-19.

I dati sono accedibili dal repository GitHub ufficiale della Protezione Civile. Tali dati vengono aggiornati a cadenza giornaliera (a partire dalle ore 18:00) fino alla fine dello stato di emergenza dichiarato dal Consiglio dei Ministri in data 31 Gennaio 2020.

Per riferimenti precisi sul codice si prega di controllare il Javadoc fornito con il codice dell'app.

## 1 Visualizzazione dati

L'applicazione all'avvio visualizza una schermata di caricamento dei dati dal Dipartimento di Protezione Civile. Tali dati, una volta ottenuti, sono elaborati e vengono mostrati all'utente gli andamenti a livello nazionale dell'epidemia (Figura 1a).

Come si può notare nella figura 1b, questa é la schermata principale dell'applicazione. Viene mostrato in alto la denominazione territoriale dei dati (nella schermata principale verrà mostrato l'andamento nazionale). Sotto i pulsanti, viene visualizzata la data dell'ultimo aggiornamento disponibile dal Dipartimento di Protezione Civile.

Il pulsante "Aggiorna" ricarica i dati dal repository.

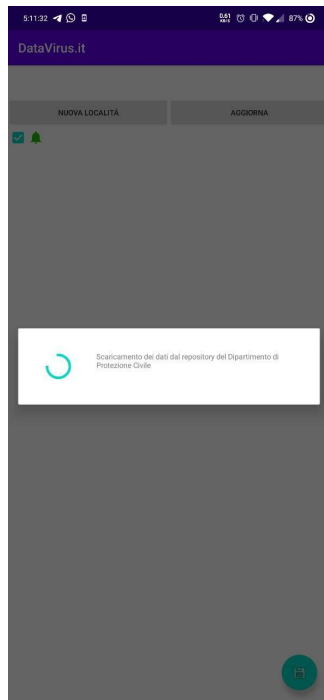
Vengono poi visualizzate le tile contenenti il dato odierno e la relativa variazione (delta) rispetto alla giornata precedente.

### 1.1 Implementazione

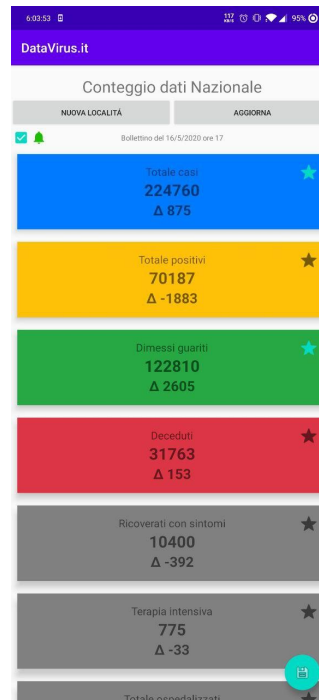
La MainActivity, alla sua creazione, istanzia un oggetto di tipo DataParser. La classe DataParser si occupa di eseguire il download dei file JSON dal Dipartimento di Protezione Civile (**DPC**) in modalità grafica (verrà visualizzato il dialog che segnala il download in esecuzione. Il dialog utilizzato é *LoadingDialog*).

Una volta concluso lo scaricamento, i dati verranno memorizzati in un oggetto di tipo *DPCData*, accessibile dalla callback *onDPCDataReady()* nell'interfaccia *OnDPCDataReady*, o alternativamente dal metodo statico *DataParser.getDPCDataInstance()*.

La MainActivity contiene un'istanza di *DataTilesFragment*, che si occupa di visualizzare le tile con i dati (ed incrementi), colorandole e posizionandole in



(a) Caricamento dei dati



(b) Andamento nazionale

ordine a seconda del parametro (es. la tile *Dimessi guariti* viene sempre colorata di verde e posizionata al terzo posto). Non per tutte le tile è previsto un ordinamento oppure un colore (verrà applicato il grigio come colore di default). Per la lista delle tile scrollabili è stata prevista una *RecyclerView* (integrata in *DataTilesFragment*). Le tile sono realizzate con un layout *CardView*. Al click su una tile seguirà una chiamata al metodo *onTileClick* presente in una classe che implementa l'interfaccia *OnTileClick*. Nella fattispecie tale classe sarà una istanza di *ChartActivity*.

## 2 Cambio zona geografica

La denominazione geografica può essere cambiata premendo sul pulsante “Nuova località” dalla schermata principale (vedi 1b).

Grazie al picker si può scegliere la zona geografica di interesse. Può essere una provincia, una regione oppure si può selezionare l'andamento nazionale. Per comodità si può ulteriormente cercare la zona d'interesse premendo sulla *lente d'ingrandimento*. Come risultato verranno visualizzati i dati relativi alla zona selezionata.

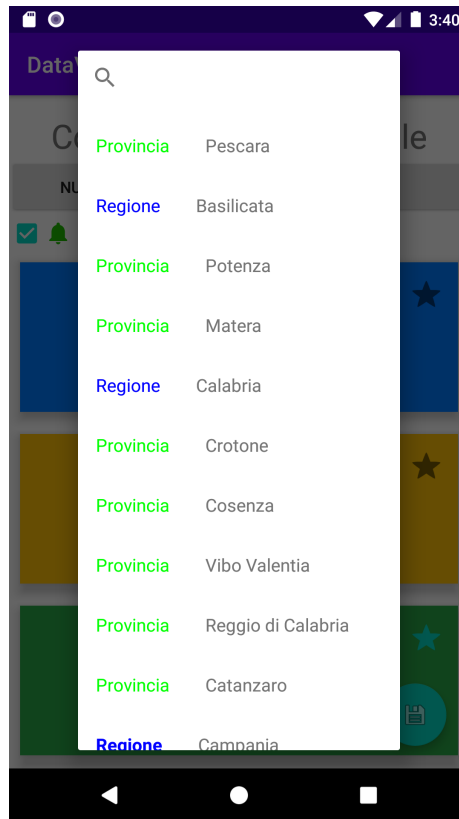


Figura 2: Picker zona geografica

Premendo il pulsante back *fisico* si scorreranno all'indietro tutte le selezioni geografiche finora cercate dall'utente.

## 2.1 Implementazione

La classe *DPCGeoAdapter* è un'estensione di un *DialogFragment* che, quando aperto, mostrerà una *RecyclerView* contenente la lista delle regioni e delle province. In prima posizione si trova la selezione per il dato nazionale.

La lista è strutturata in modo che, per ogni regione, ci sia la lista delle province che la compongono nelle posizioni sottostanti.

Alla pressione di una posizione geografica, il *GeoPicker* chiamerà la funzione *onDPCGeoClick* di una classe che implementa *OnDPCGeoListener* (nella fattispecie verrà utilizzata la *MainActivity*). L'oggetto passato è di tipo *GeographicElement*, il quale descrive il nome dell'ambito geografico e la sua tipologia (*NAZIONALE*, *REGIONALE*, *PROVINCIALE*).



Figura 3: Esempio visualizzazione regione Lombardia

### 3 Preferiti

Ogni tile ha in alto a destra una *stellina*: quando premuta, la tile relativa viene salvata tra i preferiti. Per recuperare la lista delle tile salvate é sufficiente premere sul floating button in basso a destra (visibile nella schermata principale dell'app, 1b).

Come si può notare dalla figura 4, le tile salvate sono visualizzate con la loro denominazione geografica a seguito. Premendo su una *stellina* precedentemente marcata verrà rimossa tale tile dalla lista dei preferiti.

#### 3.1 Implementazione

Per il salvataggio dei preferiti si fa uso della classe *ManageStarredTiles*, che offre i metodi per aggiungere o rimuovere un elemento dai preferiti. Per identificare una tile da aggiungere o rimuovere viene passato un oggetto di classe *FieldGeographicElement*.

I dati sulle tile marcate come starred vengono salvati in un file privato dell'app,

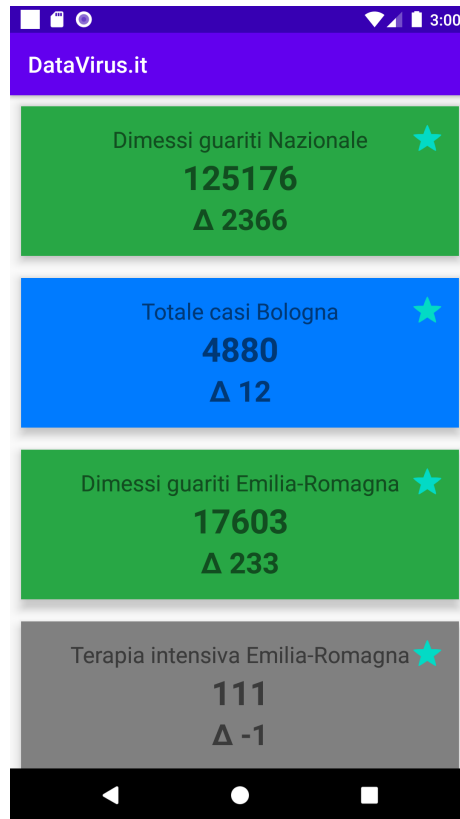


Figura 4: Lista dei preferiti

in formato json.

Per la visualizzazione della lista delle tile preferite viene utilizzata la *StarredActivity*, che si occupa di istanziare un *DataTilesFragment* che visualizzerà tutte le tile salvate.

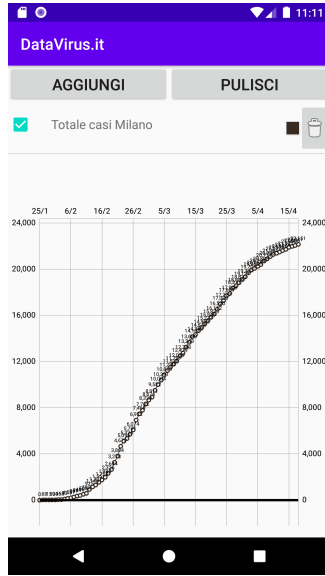
## 4 Grafici

Ogni qualvolta si premerà su una tile, verrà mostrato l'andamento nel tempo del dato selezionato in un grafico. Si presenterà la schermata come quella in figura 5a.

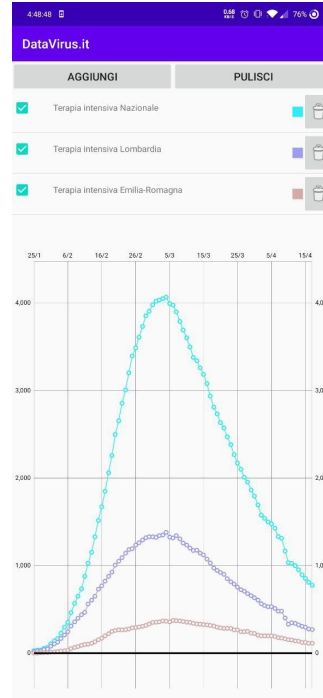
Il dato che ora viene mostrato può essere confrontato con altri dati. Per ottenere questo risultato sarà sufficiente premere sul pulsante “Aggiungi” e selezionare una nuova tile.

Ad ogni dato aggiunto viene assegnato un colore random per la sua rappresentazione e apparirà nella lista dei campi attualmente disegnati. Per ciascuno di questi si potrà decidere se nascondere o mostrarlo (con la checkbox sulla sini-

stra) oppure cancellare dalla lista premendo sull'icona del *cestino*.  
 Il grafico é ingrandibile a piacimento, semplicemente utilizzando il *pinch-to-zoom*.



(a) Esempio grafico Milano



(b) Confronto tra terapie intensive

Alla pressione del pulsante “Pulisci” si chiuderá la schermata del grafico e il grafico attualmente disegnato verrà scartato, insieme a tutti i dati precedentemente immessi. Il medesimo risultato si può raggiungere alla pressione del pulsante back *fisico*;

## 4.1 Implementazione

Per il grafico é stata utilizzata la libreria **MPAndroidChart**, che fornisce un’interfaccia completa per il plotting fornendo molteplici funzionalità. Questa viene utilizzata dentro alla *ChartActivity*. I dati sono ottenuti attraverso il singleton *ChartModel*, la quale istanza gestisce i dati da disegnare sul grafico. L’istanza del singleton *ChartModel* può essere modificata tramite il fragment *ChartElementsList*, che gestisce la RecyclerView contenente tutti gli elementi aggiunti al grafico e le funzionalità per nascondere il dato oppure rimuoverlo. Il pulsante “Aggiungi” termina l’activity corrente senza pulire la lista dei dati nel grafico, al contrario del pulsante “Pulisci” che elimina tutti gli elementi finora aggiunti.

Quando si preme su una tile si genera l'*intent* per eseguire *ChartActivity*, passando come extra valori gli elementi contenuti in una istanza di *GeographicFieldElement*.

## 5 Notifiche periodiche

Nell'activity principale, a sinistra della data di ultimo aggiornamento, é presente una checkbox con a fianco una campanellina (vedere fig. 1b).

Quando marcata, a partire dalle ore 18:00 di ogni giorno, ogni 10 minuti verrà performata una ricerca di nuovi dati aggiornati (orario nel quale il DPC li rende pubblici). Nel caso in cui i dati scaricati abbiano data odierna, l'utente verrà avvisato attraverso una notifica. Alla pressione di tale notifica si aprirà l'app.

### 5.1 Implementazione

Per gestire le notifiche si é fatto uso di una istanza *DataNotifyReceiver*, classe che estende "Broadcast Receiver" che alla sua chiamata istanzia un oggetto di tipo *DataParser* (senza interfaccia GUI) per ottenere dati aggiornati.

La chiamata a *DataNotifyReceiver* viene effettuata attraverso un'*intent* performato dall'*AlarmManager*, per impostare l'orario in cui cominciare la ricerca (ore 18:00).

Nel caso in cui i dati ottenuti dalla repository GitHub siano aggiornati alla data odierna, *DataNotifyReceiver* invia una notifica all'utente. Altrimenti viene impostata l'esecuzione del *DataNotifyReceiver* differita di 10 minuti per una nuova ricerca.