

# Personal health monitor

Gian Marco Baroncini

Febbraio 23, 2021

## Introduzione

Personal Health Monitor è una piattaforma ideata per il monitoraggio giornaliero e costante di dati riguardanti la salute.

Offre all'utente un diario, o meglio una memoria digitale, nel quale inserire il proprio stato di salute e consultare, in caso di necessità, report passati dei quali non ci si ricordava con precisione.

Il logo (Figura 1) scelto per la piattaforma rappresenta un cuore con un elettrocardiogramma sopra, simboli immediatamente associati alla salute, per rendere più veloce il riconoscimento dell'applicazione e capirne il suo scopo.



Figura 1: Logo

Sono state scelte altre icone (Figura 2) con le quali identificare le sezioni dove vengono inseriti i dati, prive di bordo per rispettare la scelta di adottare material design, snello e con colori non invadenti, dove le pagine sono settate a FullScreen.

Rispettivamente troviamo: informazioni generali, temperatura corporea, valori del sangue, dolori e attività fisica.



Figura 2: Icone Sezioni

## UserFlow

All'apertura dell'applicazione viene aperta l'activity di benvenuto, mostrata in Figura 3, la quale richiede l'inserimento di qualche informazione. Essendo questi dati utili anche in altri contesti, essi vengono salvati nelle SharedPreferences, in modo da avere un canale rapido e globale di accesso. All'inserimento dei dati nel modo corretto sarà possibile cliccare su avanti, in tal caso diversi controlli vengono effettuati insieme ad altri fondamentali check per verificare che questa pagina sia visualizzata solamente la prima volta e sia salvata la buona riuscita solamente al click sul bottone. Questa scelta viene effettuata per fare in modo che la prematura chiusura dell'applicazione non disturbi la cattura di questi dati (senza cliccare avanti la pagina viene riproposta successivamente).

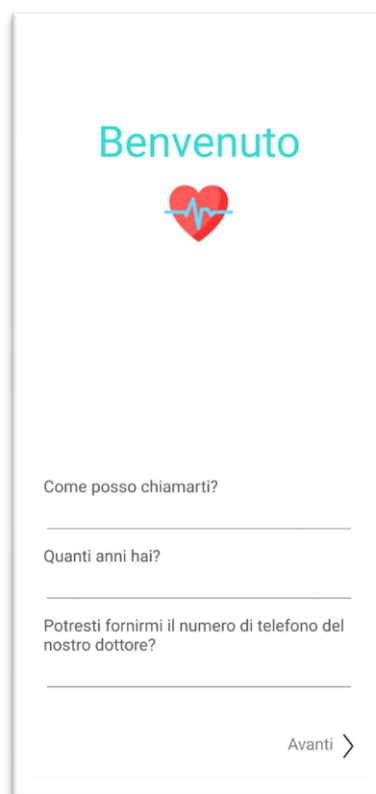


Figura 3: WelcomePage

Entrando nell'applicazione l'utente visualizzerà la main page che è il core della piattaforma (Figura 4.1).

Questa pagina è formata da una activity nella quale vengono inseriti due fragment. Dietro a questa scelta c'è l'idea di avere come primo riferimento il **report giornaliero** ("Salute") e in seconda battuta il **diario** (Figura 4.2) con tutti i report creati fino ad ora.

Questi due fragment non comprendono le ImageView cliccabili corrispondenti alle notifiche e alle impostazioni, e nemmeno la barra di navigazione situata in basso. Questi elementi rimangono pertanto fissi anche allo switch dei fragment.

Lo switch viene effettuato in diversi modi, tramite swipe, attraverso la barra di navigazione, oppure, se ci si trova sul diario, tramite il click del tasto "back" del telefono.



Figura 4.1: MainPage/Salute

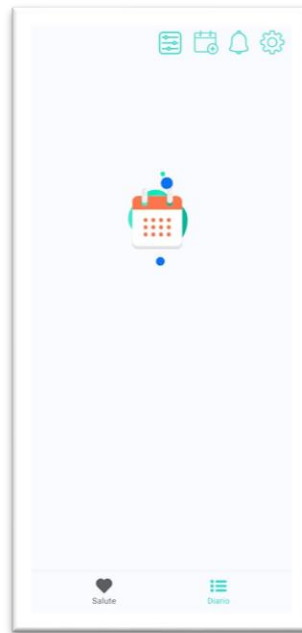


Figura 4.2: MainPage/Diario

Analizziamo ora le due pagine.

### **Salute:**

Troviamo diversi elementi all'avvio dell'applicazione: tre card un bottone "+".

Per quanto riguarda le card, queste sono ausiliare per facilitare l'utente nell'esecuzione di alcuni task che sono stati ritenuti utili.

Al tap della card raffigurante un dottore l'utente viene reindirizzato ad una activity (Figura 5) dalla quale può effettuare una chiamata al numero inserito nella pagina di benvenuto.

Se invece viene cliccata la card raffigurante il virus si apre una activity (Figura 5.1) contenente informazioni utili sul covid e un campo per scriversi un promemoria (aggiunto a notifiche, Figura 10) .



Figura 5: CallDoctor



Figura 5.1: Virus

Infine al click sulla terza card l'utente viene reindirizzato a una pagina YouTube (oppure all'applicazione di YouTube se installata) nel quale sono stati ricercati alcuni video dai quali ispirarsi per un allenamento casalingo.

Con il bottone "+" esce un menù dal quale è possibile selezionare una delle sezioni (citate inizialmente) per l'inserimento dei dati.

Questa scelta è stata effettuata in modo da non costringere l'utente ad inserire tutte le informazioni oppure avere tanti dati vuoti.

In questo modo verranno visualizzate solamente le informazioni necessarie, rendendo immediata la consultazione di report passati.

Un esempio in Figura 6.

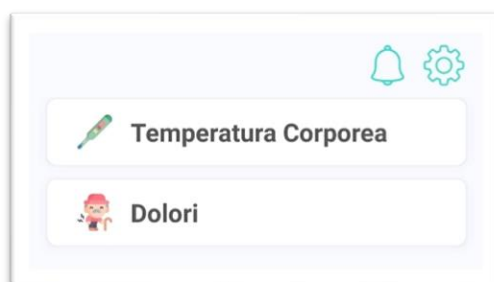


Figura 6: Sezioni Create

Le sezioni sono **modificabili** ed **eliminabili** in qualsiasi momento con un semplice swipe da sinistra verso destra.

All'apertura di una sezione ci si trova davanti una pagina dedicata e personalizzata riguardante le informazioni in questione, come mostra la figura 6.1 dove la tipologia è "**temperatura corporea**". In questo caso abbiamo tre possibili inserimenti (nel box bianco) e un grafico che ci mostra fino a quattro giorni di informazioni. Il **dataset** è formato dalla data corrente e tre date precedenti alla prima.



Figura 6.1: Pagina Sezione

### Diario:

Quando sono presenti dei **report** il diario si presenta come in Figura 7, dove le card sono ordinate in base alla data, dal più recente al meno recente.

Ogni card corrisponde a una giornata diversa e cliccandola si apre, in una nuova activity, una visualizzazione (Figura 7.1) simile a quella del fragment “Salute”.

In questa sezione è possibile modificare tutto quello che si vuole: le sezioni, eliminandole e aggiungendole, o il report stesso, che è possibile rimuovere tramite il bottone rosso presente nello screen.

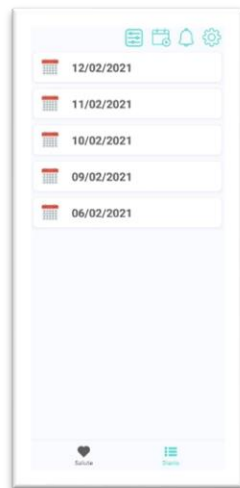


Figura 7: Diario

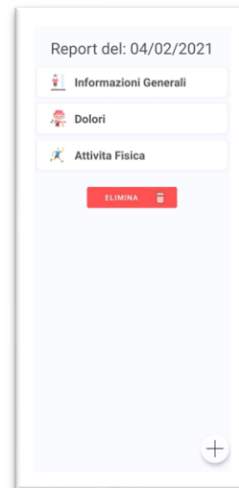


Figura 7.1: Visualizzazione Report

Come si nota dalla Figura 7 nel diario compaiono due ulteriori immagini cliccabili in alto a destra. Tramite il click del calendario viene aperto un dialog contenente un **DatePicker** (Figura 8), dove, selezionando una data, è possibile aggiungere un report con le seguenti caratteristiche: la data deve essere precedente alla data in cui si inserisce il nuovo report e non deve essere già presente nel database.

Nel caso di click sull'altra icona, esce un popup menù dal quale è possibile filtrare i report sulla base delle sezioni che questi contengono al loro interno.



Figura 8: Dialog DatePicker

Sono rimasti in sospeso solamente le notifiche e le impostazioni, analizziamole una per volta.

### **Impostazioni:**

Quando l'utente clicca sull'icona delle impostazioni gli viene mostrata una activity (Figura 9) in cui poter settare: l'orario per la ricezione del promemoria giornaliero, quali parametri controllare e quali valori soglia inserire per la pressione e per la febbre.

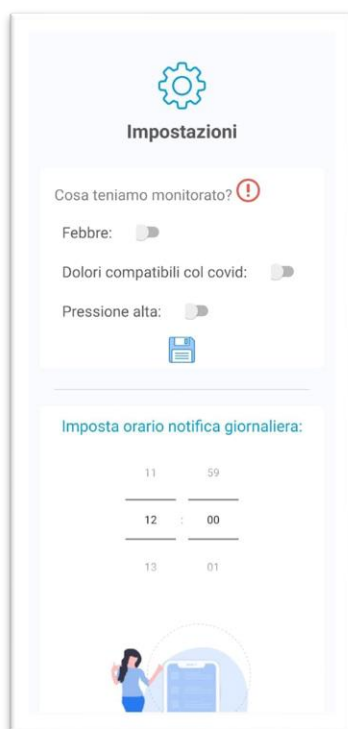


Figura 9: Impostazioni

All'orario salvato arriva una notifica come la seguente in Figura 9.1, dove è possibile tramite il bottone "POSTICIPA" (cliccabile una sola volta) selezionare un secondo orario per la ricezione di un'ulteriore promemoria su un'apposita activity simile a quella in Figura 9. Cliccando, invece, sulla parte restante della notifica si accede all'applicazione.

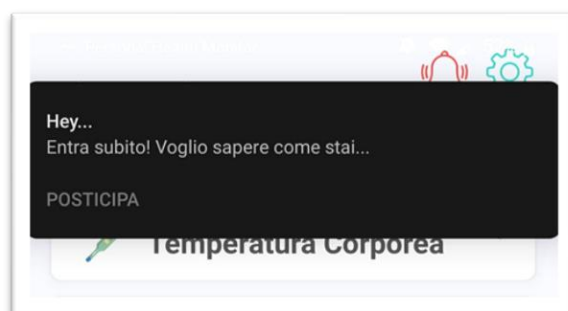


Figura 9.1: Notifica Giornaliera

## **Notifiche:**

Quando l'utente clicca sull'icona delle notifiche gli viene mostrata una activity (Figura 10) in cui poter visualizzare le notifiche ricevute dall'applicazione e alcune notifiche di avvertimento quando vengono rilevati valori non conformi a uno stato di salute sano.

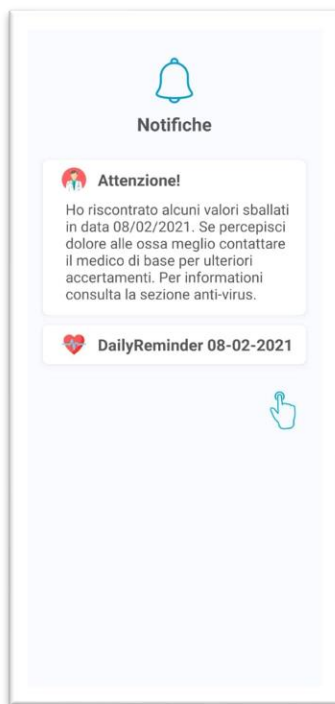


Figura 10: Sezione Notifiche

Questa pagina è stata pensata come supporto all'utente per fornirgli consigli e visualizzare la cronologia delle notifiche.

In questo modo si utilizza questa sezione per visualizzare, per esempio, le notifiche che avvertono dell'inserimento di valori critici, così diventa quasi impossibile dimenticarsi di chiamare il dottore per avere un parere chiaro sullo stato di salute.

Le card sono espandibili ed eliminabili tramite swipe, rendendo personalizzabile la bacheca.

## Scelte Implementative

### Salvataggio Dati

Per il salvataggio dei dati è stata scelta un'estensione di SQLite, **Room** persistence library, per garantire persistenza dei report inseriti e velocità nel reperirli.

Come entità principale è stata creata **EntityReports** (Figura 11) che contiene la chiave primaria identificata con la data, salvata come stringa, e dei booleani per tenere traccia delle sezioni create in quello specifico giorno.

Sono poi state create delle entità relative ad ogni sezione, sempre con chiave primaria identificata dalla data, seguita dai valori numerici o booleani appositi.

Questa scelta, anche se rindondante, è stata fatta per facilitare la creazione delle recycleView, infatti accedendo alle EntityReports si evita di guardare se per ogni data esiste la tabella corrispondente alla sezione.

```
@Entity
public class EntityReports {

    @PrimaryKey
    @NonNull
    @ColumnInfo(name = "Report Date")
    public String date;

    @ColumnInfo(name = "info")
    public boolean infoClickable = false;

    @ColumnInfo(name = "temperature")
    public boolean temperatureClickable = false;
    |
    @ColumnInfo(name = "pains")
    public boolean painsClickable = false;

    @ColumnInfo(name = "physicalActivity")
    public boolean physicalActivityClickable = false;
}
```

Figura 11: EntityReports

Per ogni entità è associata un'interfaccia **DAO** per permettere tutte le operazioni CRUD e alcune Query per prelevare le tabelle tramite la data associata.

Un esempio per la DAO riferita alla tabella delle temperature corporee.

```
@Query("SELECT * FROM EntityTemperature WHERE `date` LIKE :date")
EntityTemperature findEntityTemperatureByDate(String date);
```

Tutti i dati vengono inseriti tramite **EditText** o **CheckBox** e salvati al click del bottone "Salva" presente in ogni sezione.

Prima di aggiornare il database vengono fatti diversi controlli sull'esistenza o meno delle tabelle e sulla correttezza dei dati inseriti.



Oltre al database vengono sfruttate anche le SharedPreferences, come citato in precedenza. Se ne fa uso nel caso di dati singolari dove non serve una ripetizione di istanze con lo stesso modello; sono pertanto utilizzate per alcuni valori delle impostazioni, i dati presi nella pagina di benvenuto e poco altro.

## Settaggio Dati e App Design

Tutti i dati vengono presi e aggiornati dinamicamente tramite **RecyclerView**, utilizzate nei due fragment principali e nell'activity delle notifiche.

Risulta così semplice prendere i dati e settare il modello standard delle card che vengono presentate nell'interfaccia tramite degli **adapter**, i quali forniscono degli strumenti per facilitare l'aggiornamento quando dei dati vengono eliminati o inseriti tramite dialog.

L'unica difficoltà riscontrata è stata il passare informazioni fondamentali come la data che andava sia settata nelle card per mostrarla all'utente, sia utilizzata per fare una richiesta; questo è stato possibile spesso tramite **Intent** che permettono di passare dati aggiuntivi tra le activity con la funzione **putExtra()**, come mostra questo frammento di codice situato all'interno dell'adapter responsabile della creazione delle card nel fragment **ReportCollection**.

```
Intent newPage = new Intent(c, ReportVisualization.class);
newPage.putExtra( name: "date", holder.title.getText());
c.startActivity(newPage);
```

Si è data molta importanza alla grafica e alla scelta dei colori perchè ritenuto che un'applicazione, anche se funzionante al meglio, senza una grafica accattivante e un'esperienza utente piacevole non convince completamente.

La grafica e il flow sono infatti le uniche cose visibili con le quali indirizzare e comunicare con chi utilizza l'applicazione (tutto il codice, anche se ben costruito, rimane nascosto).

Il layout presenta elementi poco invadenti con una predominanza del GhostWhite (#F8F8FF) e come colore principale un verde acqua.

Vengono molto utilizzate icone con stile piatto e gif animate per rendere l'esperienza utente più avvolgente ed intuitiva.

Sempre per lo stesso motivo, ad eccezione dei due fragment che si alternano sulla MainActivity, le altre pagine sono activity che vanno ad occupare tutto lo schermo incentrando l'attenzione su delle specifiche funzionalità e che alla chiusura tramite tasto "back" del telefono rimandano l'utente sulla parent activity, dove le feature sono nuovamente accessibili.

Inoltre sono state inserite animazioni alla creazione delle card e il cambio di colore per la campanellina che segnala una nuova notifica (questo è stato possibile attraverso un'interfaccia con schema di tipo observer che ascolta il cambio di stato di un booleano).

## Notifiche

Le notifiche sono state implementate in una classe apposita **NotificationHelper** in modo che venissero utilizzate in tutto il progetto senza bisogno di duplicazione di codice.

Viene chiamata la funzione **CreateChannel** quando l'oggetto viene istanziato solo se la versione ne richiede l'utilizzo.

Vengono poi messe a disposizione due funzioni simili, una per le notifiche giornaliere che richiedono il settaggio del bottone "POSTICIPA", e una che invece invia una notifica standard.

Entrambe prendono in input i vari testi da settare in modo da renderle versatili a tutti gli utilizzi.

Viene allegata la funzione (Figura 12) per la creazione della notifica giornaliera richiamata dal **MyReceiver** che estende **BroadcastReceiver** e si occupa di mandare la notifica giornaliera allo scattare del timer inizializzato nelle impostazioni.

```
public void sendDailyNotification(String title, String text) {
    Intent intent = new Intent( packageContext: this, MainActivity.class);
    PendingIntent pendingIntent = PendingIntent.getActivity( context: this, notificationCode, intent, PendingIntent.FLAG_UPDATE_CURRENT);
    Intent postponeIntent = new Intent( packageContext: this, Postpone.class);
    postponeIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    PendingIntent postponePendingIntent = PendingIntent.getActivity( context: this, requestCode: 1200, postponeIntent, PendingIntent.FLAG_ONE_SHOT);
    Notification notification = new NotificationCompat.Builder( context: this, channelId)
        .setContentTitle(title)
        .setContentText(text)
        .setStyle(new NotificationCompat.BigTextStyle().bigText("Entra subito! Voglio sapere come stai..."))
        .setAutoCancel(true)
        .setPriority(NotificationCompat.PRIORITY_HIGH)
        .setContentIntent(pendingIntent)
        .setSmallIcon(R.drawable.app_icon_notification)
        .addAction(R.drawable.alarm_clock, title: "posticipa", postponePendingIntent)
        .build();
    NotificationManagerCompat.from(this).notify(new Random().nextInt(), notification);

    Calendar calendar = Calendar.getInstance();
    SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "dd-MM-yyyy");
    Utils.addNotificationNote( title: "DailyReminder " + dateFormat.format(calendar.getTime()) , text, type: "sendNotification", getApplicationContext());
}
```

Figura 12: SendDailyNotification

Per quanto riguarda la **sezione notifiche**, l'implementazione è basata, come spiegato in precedenza, su **recyclerView**.

Quest'ultima viene inizializzata prendendo i dati dal database e passandoli all'adapter, creando delle card espandibili al tap che mostrano titolo e testo delle notifiche.

A parte l'inizializzazione del layout e la possibilità di eliminare gli elementi tramite swipe attraverso **ItemTouchHelper**, non si trova molto altro all'interno di questa classe, perchè la creazione e aggiunta al DB delle notifiche è lasciata alle **Utils** (Figura 13), rendendole disponibili in qualsiasi momento a tutto il progetto.

```
public static void addNotificationNote(String title, String text, String type, Context context){  
    EntityNotificationNote entityNotificationNote = new EntityNotificationNote();  
    entityNotificationNote.title = title;  
    entityNotificationNote.text = text;  
    entityNotificationNote.type = type;  
    DB.getInstance(context).entityNotificationNoteDao().insert(entityNotificationNote);  
    Connect.setMyBoolean(true);  
}
```

Figura 13: Creazione Card Notifiche

**Connect** è la classe che funge da listener sul cambiamento di stato di un booleano, consentendo esternamente di settare la campanellina nella modalità “nuova notifica” una volta tornati alla MainActivity.

## Grafici

Per permettere la visualizzazione di grafici è stata utilizzata la libreria **MPAndroidChart** attraverso la quale sono stati realizzati **barChart**, **pieChart** e **lineChart**.

Rispettivamente è stato usato il primo tipo per mostrare l'andamento delle temperature corporee e visualizzarne gli sbalzi, mentre il secondo per avere un'idea della percentuale con cui si pratica uno sport o dei dolori percepiti maggiormente.

Negli altri casi si è optato per un grafico a linea dove i dati da vedere sono più numerosi ed è interessante vedere i piccoli cambiamenti.

Si è scelto di avere grafici relativi ad ogni tipologia di informazione, inseriti all'interno delle sezioni apposite, rispecchiando l'idea iniziale di incentrare l'attenzione dell'utente su una cosa per volta, senza rischiare di perderne l'attenzione.

Per facilitare l'interazione con i dati raccolti è possibile salvare i grafici nella galleria cliccandoci sopra o attraverso l'apposito bottone.

Le creazioni dei dati da inserire nei grafici avvengono all'interno delle specifiche activity (Figura 14) delle sezioni dove grazie alla data si possono reperire le informazioni necessarie, per poi inserirle una lista di **Entries** che andranno a formare il **dataset**.

Il setting del layout è lasciato alla classe **Utils** in modo da poter riutilizzare il codice per diverse sezioni (Figura 15).

```
ArrayList<PieEntry> pieEntries = new ArrayList<>();
pieEntries.add(new PieEntry(arts, data: 0));
pieEntries.add(new PieEntry(neck, data: 1));
pieEntries.add(new PieEntry(need, data: 2));
pieEntries.add(new PieEntry(belly, data: 3));
pieEntries.add(new PieEntry(eyes, data: 4));
pieEntries.add(new PieEntry(back, data: 5));

PieDataSet pieDataSet = new PieDataSet(pieEntries, label: "");
pieDataSet.setColors(getColor(R.color.sageGreenRipple), Color.parseColor( "ff6767"), getColor(R.color.orangeYellow), Color.parseColor( "92b4f2"), getColor(R.color.backgroundB1gSection));
pieDataSet.setSliceSpace(1.5f);
pieDataSet.setValueTextColor(getColor(R.color.textColor));
pieDataSet.setValueTextSize(12f);
pieDataSet.setSliceSpace(18f);

PieData pieData = new PieData(pieDataSet);

Legend legend = pieChart.getLegend();
legend.setEntries();
legend.setEntrySpace(18f);
legend.setShadowEnabled(true);

LegendEntry l1 = new LegendEntry( label: "Arti", Legend.LegendForm.CIRCLE, formSize: 10f, formLineWidth: 2f, formLineDashEffect: null, getColor(R.color.sageGreenRipple));
LegendEntry l2 = new LegendEntry( label: "Collo", Legend.LegendForm.CIRCLE, formSize: 10f, formLineWidth: 2f, formLineDashEffect: null, Color.parseColor( "ff6767"));
LegendEntry l3 = new LegendEntry( label: "Testa", Legend.LegendForm.CIRCLE, formSize: 10f, formLineWidth: 2f, formLineDashEffect: null, getColor(R.color.orangeYellow));
LegendEntry l4 = new LegendEntry( label: "Pancia", Legend.LegendForm.CIRCLE, formSize: 10f, formLineWidth: 2f, formLineDashEffect: null, Color.parseColor( "92b4f2"));
LegendEntry l5 = new LegendEntry( label: "Occhi", Legend.LegendForm.CIRCLE, formSize: 10f, formLineWidth: 2f, formLineDashEffect: null, getColor(R.color.backgroundB1gSection));
LegendEntry l6 = new LegendEntry( label: "Schiena", Legend.LegendForm.CIRCLE, formSize: 10f, formLineWidth: 2f, formLineDashEffect: null, Color.parseColor( "99e699"));

legend.setCustom(new LegendEntry[] {l1, l2, l3, l4, l5, l6});

legend.setEnabled(true);
```

Figura 14: Creazione Dataset PieChart in PainsSection

```
public static void setBarChartLayout(BarChart barChart){
    barChart.setDrawBarShadow(false);
    barChart.setDrawValueAboveBar(true);
    barChart.setDrawGridBackground(false);
    barChart.getAxisRight().setEnabled(false);
    barChart.getLegend().setEnabled(false);
    LimitLine line = new LimitLine(37f);
    barChart.getAxisLeft().addLimitLine(line);
    barChart.animateXY( durationMillis: 2500, durationMillis: 2200);
    barChart.getDescription().setText("");
    barChart.getAxisLeft().setPosition(YAxis.AxisLabelPosition.OUTSIDE_CHART);
    barChart.getAxisLeft().setAxisMinimum(35);
    barChart.getAxisLeft().setAxisMaximum(40);
    barChart.getAxis().setEnabled(true);
}
```

Figura 15: Setting del Layout BarChart