Buse Çarık 14/08/18

INTERRA R&D Material Design

Material Design

Bottom Sheet

https://github.com/InterraMaterialDesign/BottomSheet

**INDEX**

1. **INTRODUCTION….….….…….….……….………….……………………….……….3**
   1. **Purpose of this document…………………………………………………………...3**
   2. **Overview………………………….……….…….…………………………………...3**
2. **WHAT IS BOTTOM SHEET………...…………………………………………………4**
   1. **Persistent Bottom Sheet……………………………………………………….…….4**
      1. **Areas of Usage………….…………….………………………………………5**
      2. **Implementation….……………….……….………………………………….8**
   2. **Modal Bottom Sheet………………………………………………………….……14**
      1. **Areas of Usage………………………………………………………………15**
      2. **Implementation…………………………………….……………………….17**
3. **BOTTOM SHEET PROJECT……………………….………….…………………….18**
   1. **Description…………………………………………………………………………18**
   2. **Purpose ………………………………………….…………………………………18**
   3. **Main Screen…….………………….……………………………………………….18**
      1. **Favorite…….…………….…………………….……………………………19**
      2. **Hits………………….……………………………………………………….20**
      3. **Mood…………………………………………………………………………20**
      4. **Song Progress……………………………………………………………….20**
   4. **Detail of Music That Is Played………………….…….………………...…………20**
4. **REQUIREMENTS FOR IMPLEMENTATION…….…………….…………………21**
5. **REFERENCES…………………………………….……………………………………22**

**INTRODUCTION**

**PURPOSE OF THIS DOCUMENT**

The aim of this project is analyzing bottom sheet and its types.

**OVERVIEW**

The project that is described in this document, includes bottom sheet’ description, types of the bottom sheet, where and when it is used and how it should be designed according to Material Design.

**Bottom Sheet**

Bottom Sheet is one of the android design support library components that is introduced with Material Design concepts and they are still in under development phase. Bottom sheets are placed in the bottom of the screen, in front of the other UI elements. They provide extra information about main content. They expand only with user interaction. Bottom sheets are primarily used in mobiles because their screens are limited, and bottom sheets can provide extra surface for the main content. In large screens like desktops, instead of bottom sheets, navigation bars or side sheets can be preferred. To anchor a view like FAB to the bottom sheet ***layout\_anchor*** attribute can be used. Reference the id of the view which has set the BottomSheetBehavior with the layout\_anchor attribute. The view should be the direct child of the Coordinator Layout. Also, to arrange its place, ***layout\_anchorGravity*** attribute is used.

<android.support.design.widget.FloatingActionButton

android:id="@+id/fab"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:src="@android:drawable/ic\_dialog\_email"

app:layout\_anchor="@id/app\_bar"

app:layout\_anchorGravity="bottom|end"

/>

According to usage, placement and appearance; there are two types of bottom sheets which are Persistent(Standard) Bottom Sheet and Modal Bottom Sheet.

**Persistent (Standard) Bottom Sheet**

A small part of this type of the bottom sheet is found together with main UI region on the screen. While the user is interacting with the main UI, this bottom sheet is visible on the screen. This means that the users can interact with both bottom sheet and main content. They usually hold contents that complements the main content. They can also contain an important feature or secondary content that are related with the main content. And, they can be used to keep that secondary content on the screen, while the main UI was scrolled down or up.

By default, the sheet has 8dp elevation which enables that as a part of the main content but at the same time, when the main content is scrolled or panned, it seems like they scroll behind the sheet. Also, when the sheet becomes full-screen mode, it allows them to come to front of the main content. An important point is that, when the sheet is expanded and covered the page, the main content can be still clickable. When the user clicks the screen, main content can do actions.

Initially, a small part of the content is visible on the screen. This part can be set by the developer in two ways. First option, it can be set in the XML file of the bottom sheet with ***app:behavior\_peekHeight*** attribute or can be set programmatically with ***setPeekHeight***(int peekHeight) method (implementation is explained below in detail). In mobile, this permanent part is in full-width; in tablets and desktops, there can be full-width, or some insets in sides. But these insets should not be in all sides because sheets can be confused with snackbars. To show more or less content and become full screen mode, the vertical drag movement is used. When they are in the full screen-mode; to return the sheet to the initial position, they are needed to an icon or the drag movement can be used.

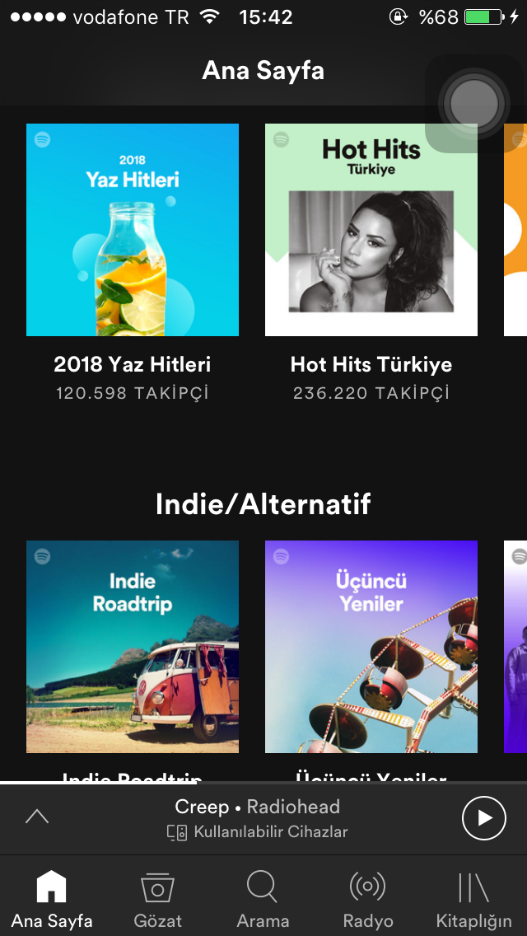
By default, Bottom Sheet mechanism does not prevent the overlap between the sheet and the main content. If an enough space is not allowed, small part of the main content may stay behind the sheet. To avoid this consequence, set the bottom margin according to sheet.

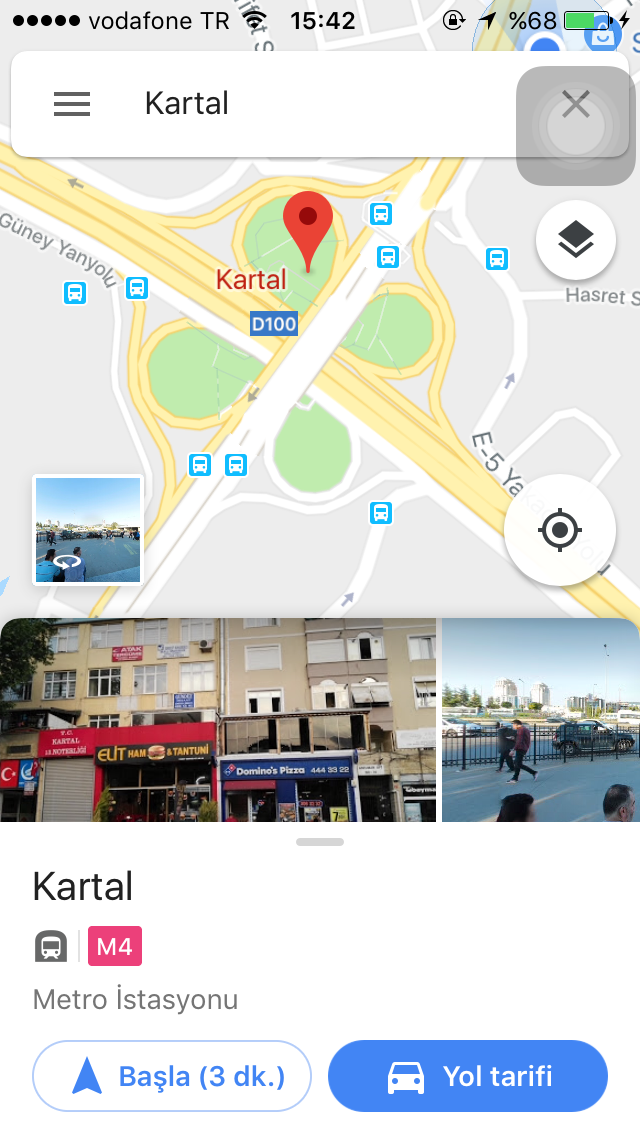
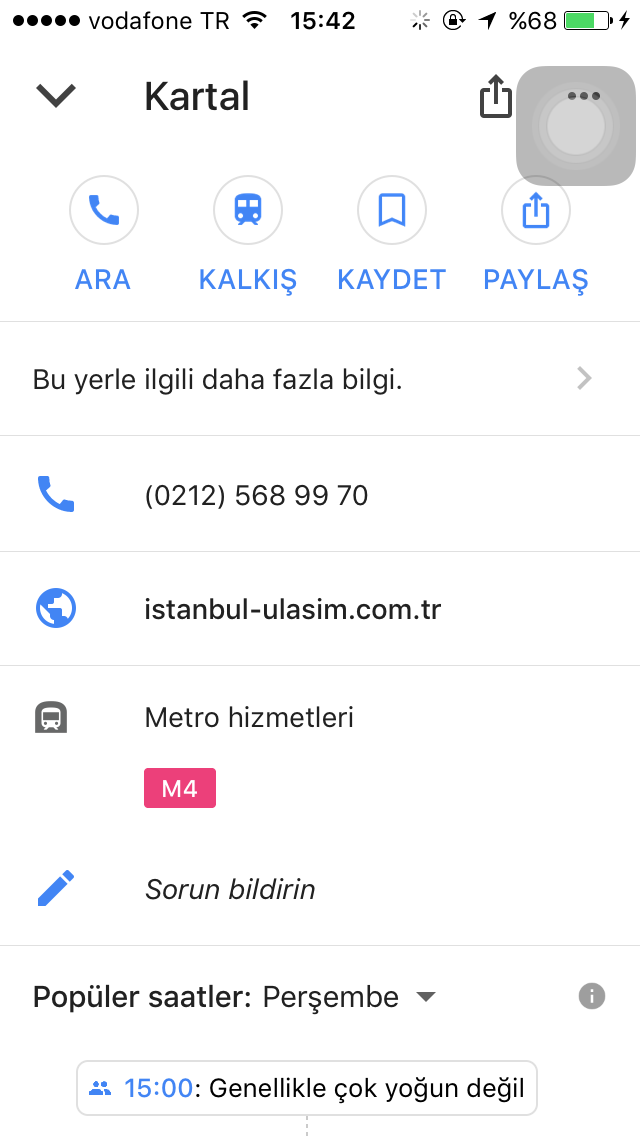
One the differences between the two types of sheet is that when the persistent sheet is expanded, they occupy the entire screen regardless of the size of the content in the sheet; they turn into the full screen mode directly. A note: if you set the ***height*** of the bottom sheet’s container as ***wrap\_content***, they cannot cover the all screen. To cover the all screen, you should set the ***layout\_height*** as ***match\_parent***. On the other hand, the modal sheet can be expanded depending on the size of the content.

In full screen mode, the bottom sheets can be scrolled within themselves, therefore they can be an any length and other material design’s components can be placed in there like RecyclerView and CardView. And when they are scrolled enough, they can display a top app bar. This bar can contain an icon to collapse the sheet.

**Areas of Usage**

Persistent bottom sheet can be used for observing an important feature while the user is interacting with other contents. The common example of this usage is music apps like Spotify. Users can track the songs from the bottom of the screen, when they search other songs or bands. The other main usage is the apps that give information about location and direction. In those apps, the sheets can provide complementary information about the place or routes. Google maps is the best example of this usage.



The sheet is collapsed(default mode). The sheet is expanded. Top app bar is also placed.

**Implementation**

These two types of bottom sheets’ implementations are also different. Persistent bottom sheets can be created with ***BottomSheetBehavior*** class. To anchor the persistent bottom sheet to the bottom of the screen, its parent view must be ***Coordinator*** ***Layout***. Because, ***BottomSheetBehavior*** class can be applied to the child view of the Coordinator Layout and this class allows this view act as a bottom sheet. The class provides gestures to the bottom sheet like drag, swipe and peek. Also, it allows the sheet to stay permanently at the bottom of the screen.

To create a Persistent Bottom Sheet; firstly, a view should be included to the coordinator layout. This will be your bottom sheet’s container. The view can be any view depending on your desire, it can be RecyclerView, LinearLayout, RelativeLayout or NestedScrollView… When you create the UI of your bottom sheet, you can create a separate XML file and then include it to the coordinator layout with the ***<include>*** tag. Or you can write your bottom sheet directly to the main XML file.

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

xmlns:app="http://schemas.android.com/apk/res-auto"

android:orientation="vertical"

android:id="@+id/linear\_bottom\_sheet"

app:layout\_behavior="@string/bottom\_sheet\_behavior"

app:behavior\_peekHeight="50dp" >

<TextView

android:id="@+id/song\_name"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

</LinearLayout>

<android.support.design.widget.CoordinatorLayout

……… >

<include layout=”@layout/bottom\_sheet” />

<android.support.design.widget.CoordinatorLayout

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:id="@+id/coordinator"

tools:context=".MainActivity">

……

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:id="@+id/linear\_bottom\_sheet"

app:layout\_behavior="@string/bottom\_sheet\_behavior"

app:behavior\_peekHeight="50dp">

<TextView

android:id="@+id/song\_name"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

</LinearLayout>

The most important attribute in above code is ***app:layout\_behavior***. The value that is taken by this attribute must be string resource which is ***bottom\_sheet\_behavior*** its value is ***android.support.design.widget.BottomSheetBehavior***. With this attribute, the layout is recognized as a bottom sheet container and the behavior is applied to the layout. Other attributes that are belong to the BottomSheetBehavior class are ***app:behavior\_peekHeight***, ***app:behavior\_hideable*** and ***app:behavior\_skipCollapsed***. As in mentioned above, peekHeight attribute sets the height of the visible part of the persistent bottom sheet when is collapsed and the height can be also set programmatically with ***setPeekHeight***(int peekHeight) method and the value can be reached with ***getPeekHeight***() method. By default, the persistent bottom sheet cannot be completely disappeared if the peekHeight is set. Because behavior\_hideable attribute is set false by default. If the value of this attribute is set as true, the sheet will be hidden when it drags down even if you set a peekHeight. The attribute can be also set programmatically with ***setHideable***() method and you can reach the value with ***isHideable***() method. The last attribute sets whether the sheet will be collapsed before it reaches the hidden state when it is dragged down after it is expanded. This attribute can be also control programmatically with ***getSkipCollapsed***() and ***setSkipCollapsed***() methods.

**Bottom Sheet Behavior Class**

This class is added in version 23.4.0 and is extended from ***CoordinatorLayout.Behavior*** <V extends View> class. It allows coordinator layout's child view to be considered as a bottom sheet. It works with Coordinator Layout and allows the sheet to perform animations and drag/swipe/dismiss movements. To display the bottom sheet, first reference of the bottom sheet’s container should be taken; then with this reference, get the BottomSheetBehavior instance with ***from***() method.

LinearLayout llBottomSheet = findViewById(R.id.bottom\_sheet);  
  
BottomSheetBehavior bottomSheetBehavior = BottomSheetBehavior.from(llBottomSheet);

Bottom sheets have 5 states:

***STATE\_COLLAPSED***: It is the default state. Sheet is visible only the height that is set by peekHeight.

***STATE\_DRAGGING***: The bottom sheet is dragging up or down.

***STATE\_EXPANDED***: The sheet is visible and reaches its max height.

***STATE\_HIDDEN***: The sheet is not visible on the screen. disabled by default.

***STATE\_SETTLING***: short time between when the view is released and settling its final position.

These states can be set dynamically with ***setState***() method, also this method provides animations between transitions. ***getState***() method returns the current state of the sheet.

To observe the states of a bottom sheet, there is a callback class that is called as ***BottomSheetCallback***. The callback class provides callbacks to listen to changes in the state of bottom sheets. It has two methods; ***onStateChanged(View bottomSheet, int newState)*** and ***onSlide(View bottomSheet, float slideOffset)***. onStateChanged method is called when the bottom sheet’s state changes. Like dragging up the sheet to show more content. onSlide method is called when the sheet is being dragged up or down. One note is that onSlide method takes slideOffset parameter which its value between -1 to 1. When the sheet goes from hidden state to collapsed state, its value increases to 0.0 from -1.0. when the sheet goes from collapsed state to expanded state, its value increases to 1.0 from 0.0. A callback is set to the bottom sheet with ***setBottomSheetCallback(BottomSheetBehavior.BottomSheetCallback callback)*** method.

BottomSheetBehavior.BottomSheetCallback bottomSheetCallback = new BottomSheetBehavior.BottomSheetCallback() {

@Override

public void onStateChanged(@NonNull View bottomSheet, int newState) {

}

@Override

public void onSlide(@NonNull View bottomSheet, float slideOffset) {

}

};

bottomSheetBehavior.setBottomSheetCallback(bottomSheetCallback);

Other methods that the BottomSheetBehavior class has:

***onInterceptTouchEvent***(CoordinatorLayout parent, V child, MotionEvent event)

***onLayoutChild***(CoordinatorLayout parent, V child, int layoutDirection)

***onNestedPreFling***(CoordinatorLayout coordinatorLayout, V child, View target, float velocityX, float velocityY)

***onNestedPreScroll***(CoordinatorLayout, View, View, int, int, int[], int)

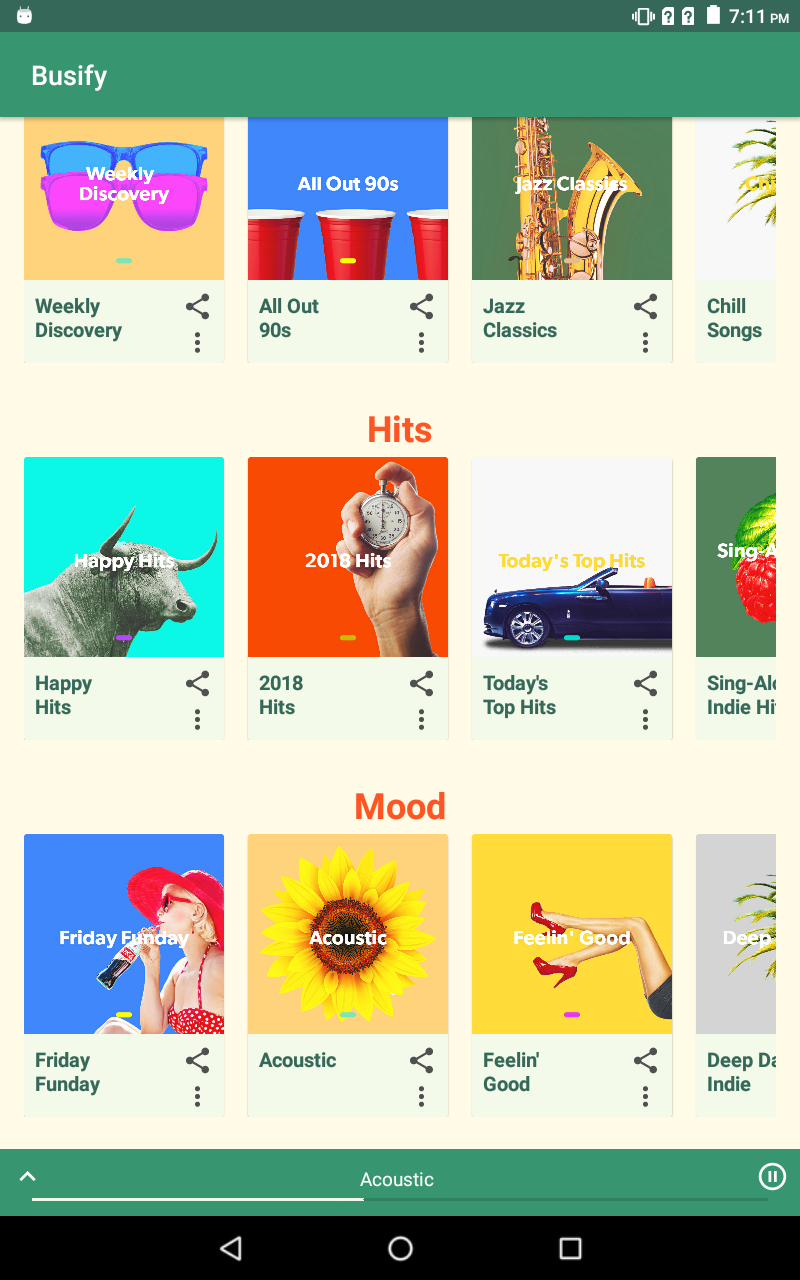
***onRestoreInstanceState***(CoordinatorLayout parent, V child, Parcelable state)

***onSaveInstanceState***(CoordinatorLayout parent, V child)

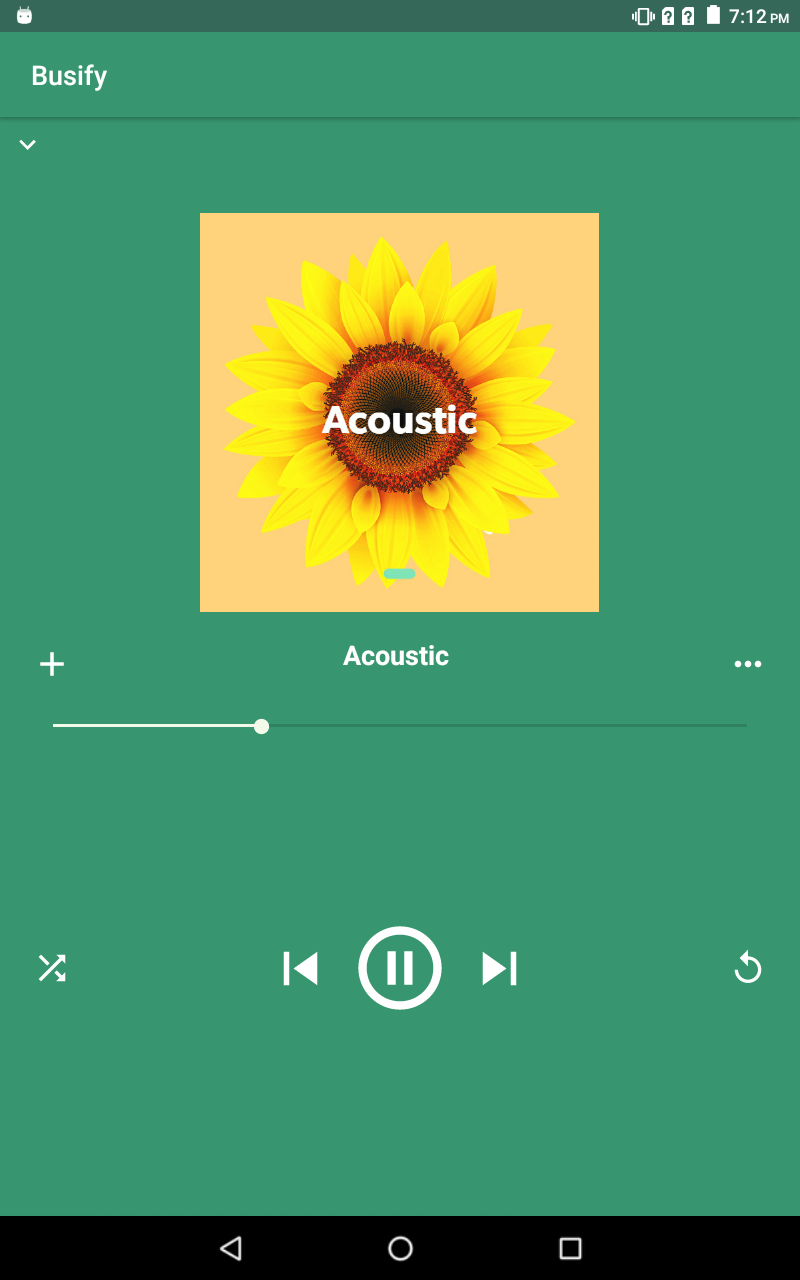
***onStartNestedScroll***(CoordinatorLayout, View, View, View, int, int)

***onStopNestedScroll***(CoordinatorLayout, View, View, int)

***onTouchEvent***(CoordinatorLayout parent, V child, MotionEvent event)



Persistent bottom sheet in collapsed state



Persistent bottom sheet in expanded state

**Modal Bottom Sheet**

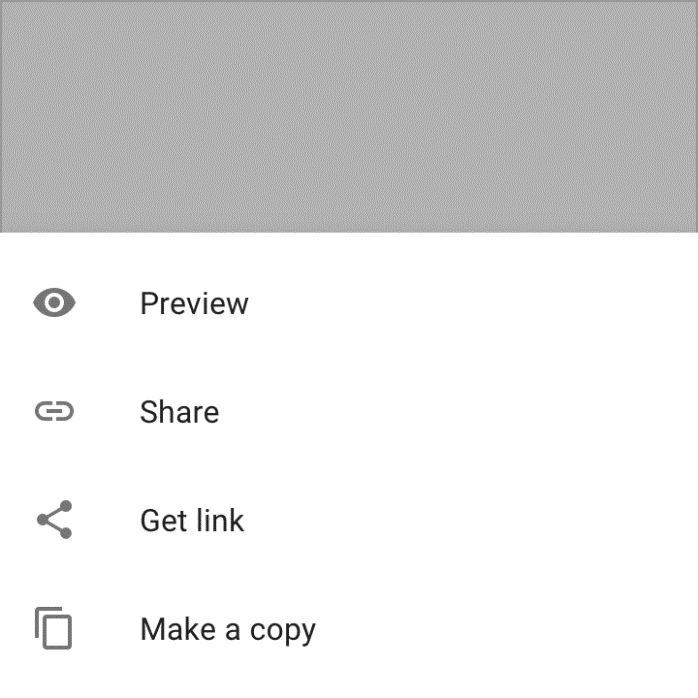
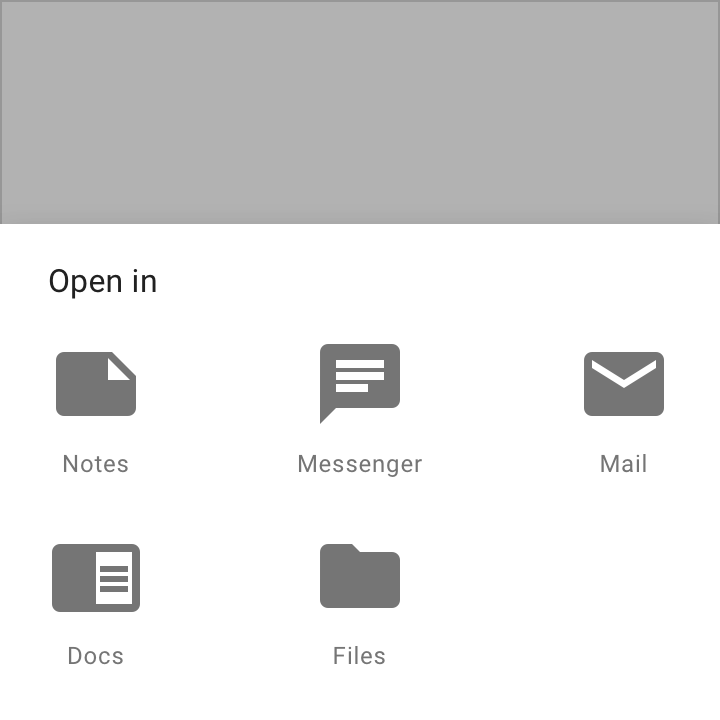
Modal sheets provide extra space for icons, actions and contents. They are an alternative for dialogs and menus on mobile. They are primarily for mobiles; in bigger screens like desktop, inline menus should be used instead of modal bottom sheets. Unlike the persistent bottom sheet; in this type, users cannot interact with the main UI. To interact, the sheet must dismiss. Icons, longer explanations, additional items and lists can be placed into modal sheet. They are not a part of the layout like persistent bottom sheets; instead their initial appearance can be an icon, or a button and they are shown dynamically. These icon or button should be placed in the Bottom App Bar or Top App Bar. When the icon or button is clicked, the sheet expands. To dismiss the sheet; by default, when the scrim area is clicked, the sheet disappears. Or it can be dragged down; or an icon can be used again.

The default elevation of the modal sheet is 16dp. This elevation allows it to the front of the other components. Also, they cover the action bar or toolbar. And for that effect, all the contents that are behind the modal sheet display the scrim. This effect causes the feeling that background content is not responded to the user interaction. When this scrim area is tapped, the sheet dismisses.

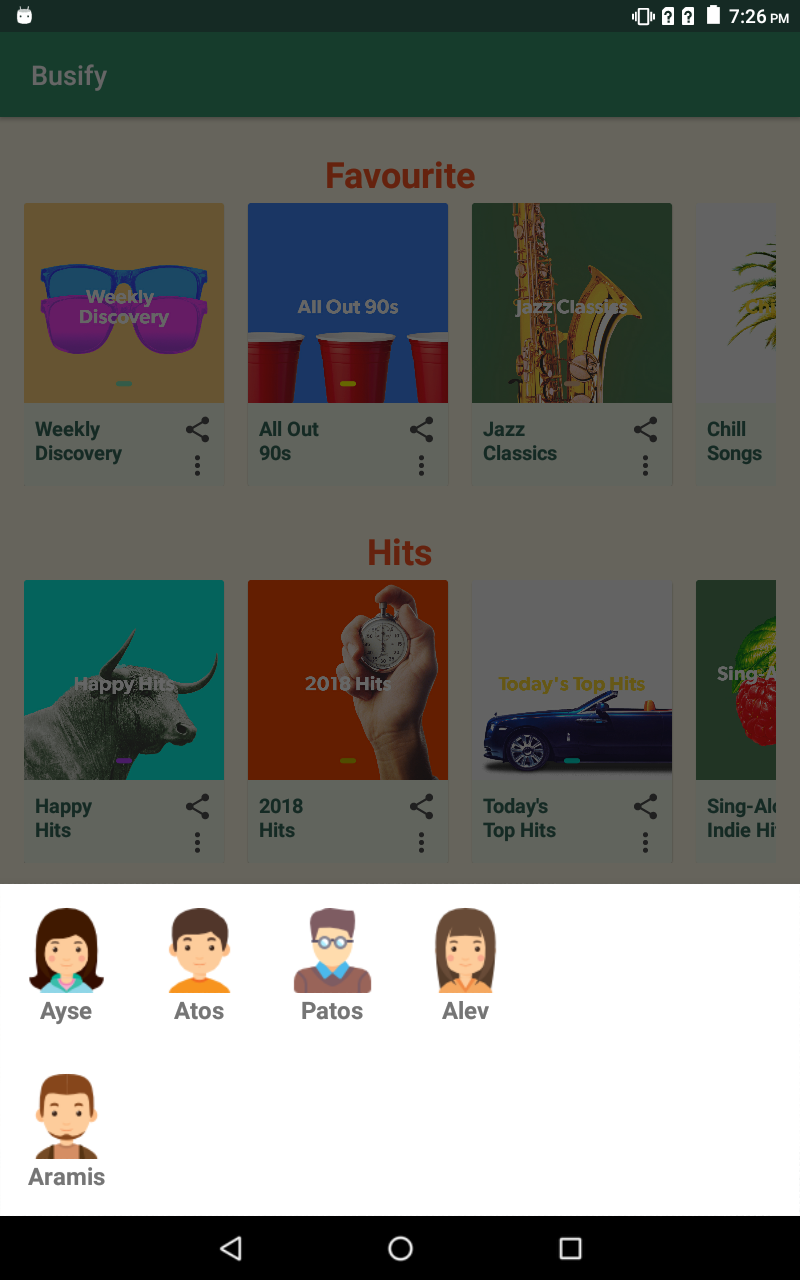
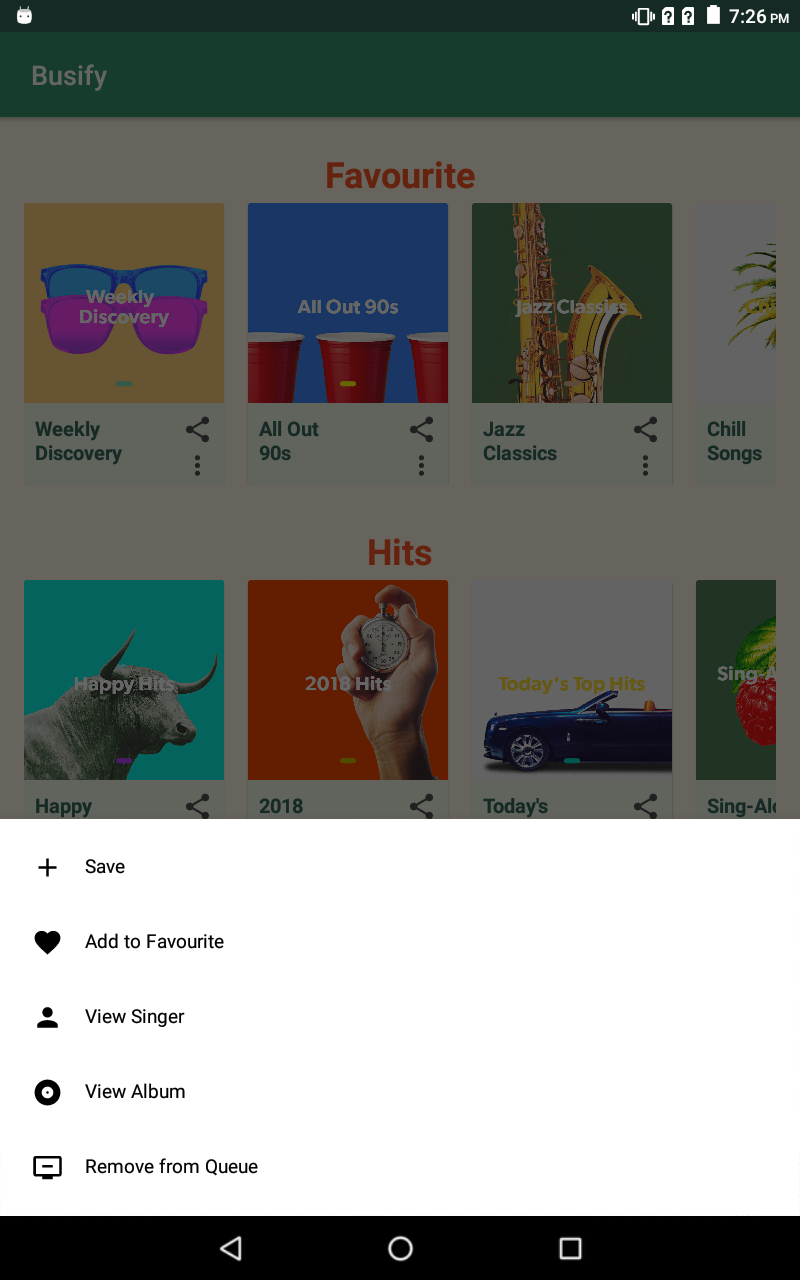
There is no need to calculate the peek height in the modal bottom sheet like persistent type, it handles automatically. Sheet’s visible area changes according the height of the sheet. If the height is below half of the screen, it will be at full height. If the height is between the 50 to 100%, its visible height will be half of the screen. To display the all sheet, it can scroll or tap to the surface. If the height is equal or greater than the 100%, its visible height will be half of the screen again. On scroll or tap, it scrolls internally. When it reaches the bottom of the sheet, it moves to the top of the screen and scrolls internally. To close the sheet, it can be dragged down or an action can be added.

**Areas of Usage**

Modal bottom sheets are used for providing extra area for additional items, icons, longer explanations. Or they can include deep-linked content from other apps. Also, they can contain extra screen actions like in music app. The contents can be displayed in list or grid view.

*List of actions Menu of apps (Material Design)*



**Implementation**

Modal bottom sheets are created with ***BottomSheetDialogFragment*** which is extended from ***AppCompatDialogFragment***. It basically behaves like a dialog but uses ***BottomSheetDialog*** to creates a bottom sheet instead of a floating dialog. BottomSheetDialog is subclass of the Dialog which is styled as bottom sheet.

Entrance animation of modal bottom sheet is better than the dialog. The other difference is that the modal bottom sheet is placed at the bottom of the screen.

A custom dialog can be created with ***onCreateDialog (Bundle savedInstanceState)*** method which is BottomSheetDialogFragment’s overridden method.

When implementing a modal bottom sheet, firstly you should create an XML file. This file is inflated to the BottomSheetDialogFragment and uses as content of the sheet. After that, you need a class which is extending BottomSheetDialogFragment. In that class, ***onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)*** should be overridden to inflate the layout.

public class PersonListDialogFragment extends BottomSheetDialogFragment {

public static PersonListDialogFragment newInstance() {

return new PersonListDialogFragment();

}

@Nullable

@Override

public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {

return inflater.inflate(R.layout.fragment\_person\_list\_dialog, container, false);

}

}

Then, to display the sheet, ***show(FragmentManager manager, String tag)*** method is used. First you should create a new instance of your modal bottom sheet and with show() method the sheet will be displayed. It needs ***Activity.getSupportFragmentManager()*** because BottomSheetDialogFragment is a subclass of ***AppCompatFragment***.

PersonListDialogFragment fragment = new PersonListDialogFragment();

fragment.show(getSupportFragmentManager(), “bottom sheet”);

Instead creating these files one by one, when you click ***file -> new -> fragment***, you see modal bottom sheet. This creates a modal bottom sheet containing a RecyclerView list can be multiple columns and creates the XML files and subclass of BottomSheetDialogFragment. It gives prepared Adapter and ViewHolder. Also, a Listener interface is existed. This is compatible back to API level 9.

Two notes:

You should prefer to override ***onCancel(DialogInterface)*** and ***onDismiss(DialogInterface)*** methods instead ***setOnClickListener()*** and ***setOnDismissListener()***.

And if you want to prevent to dismiss the modal sheet on touch outside, you can use getDialog().setCancelOnTouchOutside(false) method.

**Bottom Sheet Project**

**Description**

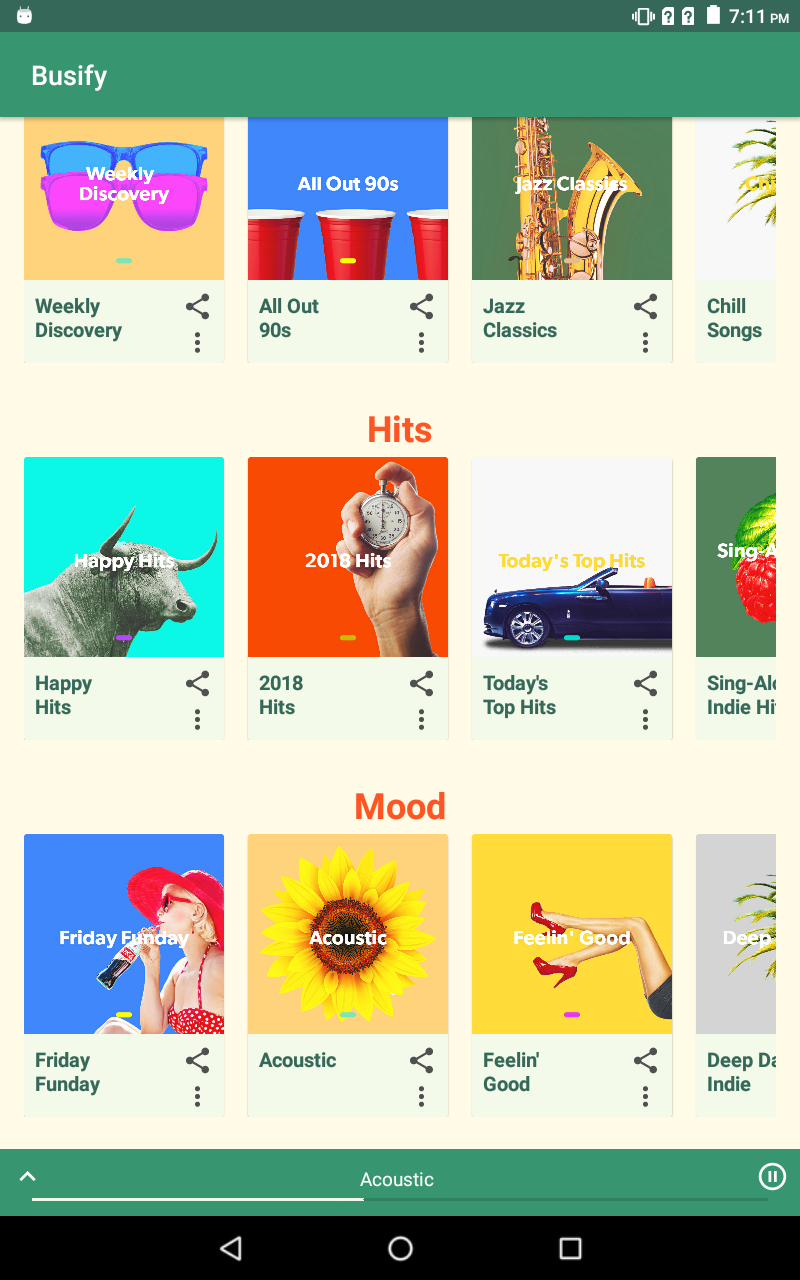
Busify is an app for listening to music, explore new bands and songs, and creating lists. Users can easily create lists according to their tastes and download them. And they can find similar bands and songs from their playlist’s radio. Besides the personal lists, the user can save the songs they like to the library. Also, they can listen lists according to their mood and discover new hits. The app also allows them to display the other songs of singers or that singer’s album. The users can observe the progress of the song in the main screen and they can play or pause. One of the important features of the app is that they can share songs with their friends.

**Purpose**

The main purpose of creating this application is that provide a nice platform for users to listen to music and make it easy for them to reach their favorite songs. Also, they can exchange songs with their friends.

**Main Screen**

On the main screen, there are three lists which are favorite, hits and mood. Users can play the lists by clicking on the cover. When they click to the cover, song progress part that is at the bottom of the screen, is changed according to the song that is played.



**Favorite**

In favorite part, there are group of lists that are created by the user. User can create as many lists as they want. These lists are scrolled horizontally. In each list, there are a cover of the list, its name below the cover and two icons. Clicking the cover starts to play the list. The top icon is for sharing the list with their friends. The below icon displays more options about the list. Users can save the list, add/remove to the favorite, display the singer or album of the current song and remove this list from the queue.

**Hits**

Hits is prepared by the application. It displays lists that are consist of the current hits. These lists arrange according to the type of the songs and concepts. These lists have the same appearance and functions with favorite. They have a cover, a name below the cover and two icons. These icons features are the same with favorite.

**Mood**

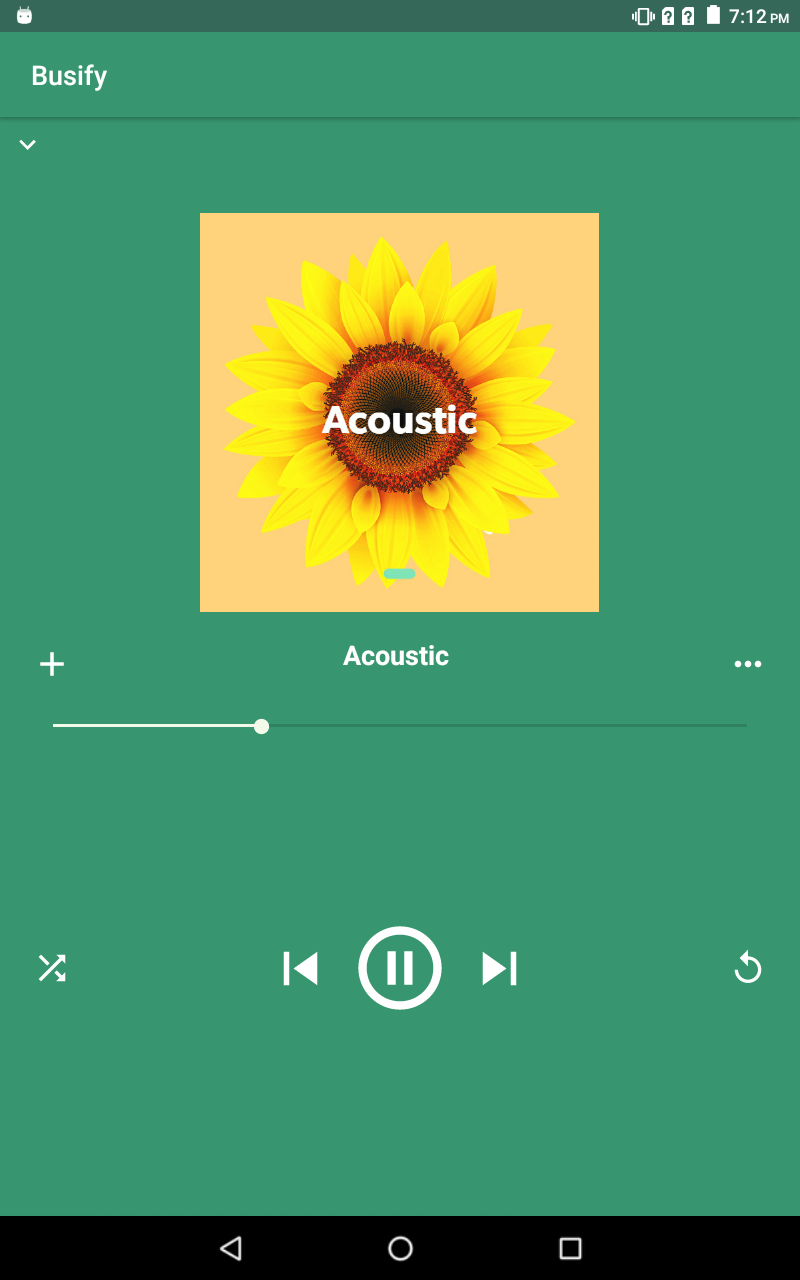
Mood is also prepared by the application. These lists are set according to mood, weather, seasons, special days and time of day. These lists have the same appearance and functions with favorite and hits. They have a cover, a name below the cover and two icons. These icons features are the same with favorite.

**Song Status**

In that part of the screen, users can see the song that is played and its progress. In here, there also two icons and the name of the list. The name is determined according to list that is played by the user. The icon on the left is for expanding this part. When this icon is clicked, a new screen is appeared. They can also drag up down the song progress part to expand or collapse the new screen. The icon on the right is for playing or pausing the song.

**Detail of Music That Is Played**

In here, the user can find details and more options about list and song. In the center, cover and the name of the list is displayed. The status of the song is under the name. At the top left of the screen, there is an icon which is collapsed this part. At left of the name, there is a plus icon which adds the list to the library. When it is clicked, a notification is appeared. The icon that is at the right of the name displays actions. In that part, user can save the list, add/remove the list to favorite or queue, display the singer or album of the song, play the radio of that list, download the list, take the screenshot of the screen, leave comments and get help about the application. This part of the app can be scrolled internally and cover the all screen. Below them, there are group of icons. When the leftmost is clicked, songs in the list is mixed. The icon that is at the right of the previous icon passes to the previous song. The icon that is at the center plays or pauses the song. The icon to the right of the center passes the next song. And finally, the rightmost icon plays the song again.



**Requirements for Implementation**

Bottom Sheets are part of android design library. They are added in version 23.2. To use bottom sheets for both types, you must add the following requirements to the build.gradle file in the project:

dependencies {  
    implementation ' com.android.support:appcompat-v7:28.0.0 -alpha1'  
 implementation 'com.android.support: design:28.0.0 -alpha1’ }

**References**

Android Developers. (n.d.). Retrieved June 27, 2018, from https://developer.android.com/

Bottom Sheet with Android Design Support Library. (2017, March 16). Retrieved August 16, 2018, from https://blog.iamsuleiman.com/bottom-sheet-android-design-support-library/

Homepage. (n.d.). Retrieved August 14, 2018, from https://material.io/