Material Design Codelab

Created by: onome@

Last update: onome@, 8/22/2015

|  |  |
| --- | --- |
| **Summary** | In this codelab, you'll learn how to build an Android app using Material Design principles. |
| **Material Design Spec** | <https://www.google.com/design/spec/material-design/introduction.html> |
| **Category** | Android |
| **Status** | Final Draft |
| **Feedback Link** | <https://docs.google.com/forms/d/16K77iRQIB3r0jJPzcEJqzYg9_JEDqRKpbjrsHctYwIY> |

[Overview](#h.30j0zll)

[What you’ll learn](#h.1fob9te)

[What you’ll need](#h.3znysh7)

[Setup](#h.2et92p0)

[Lessons](#h.tyjcwt)

[Lesson 1 - Color and Typography](#h.3dy6vkm)

[Customize the Theme and Color Palette](#h.1t3h5sf)

[Lesson 2 - Layout and Animation](#h.4d34og8)

[Add a Top and Bottom ToolBar](#h.2s8eyo1)

[Add a RecyclerView and Cards with Ripples](#h.17dp8vu)

[Add an Image to a CollapsingToolbar](#h.3rdcrjn)

[Lesson 3 - Page Elements](#h.lnxbz9)

[Add a NavigationDrawer](#h.35nkun2)

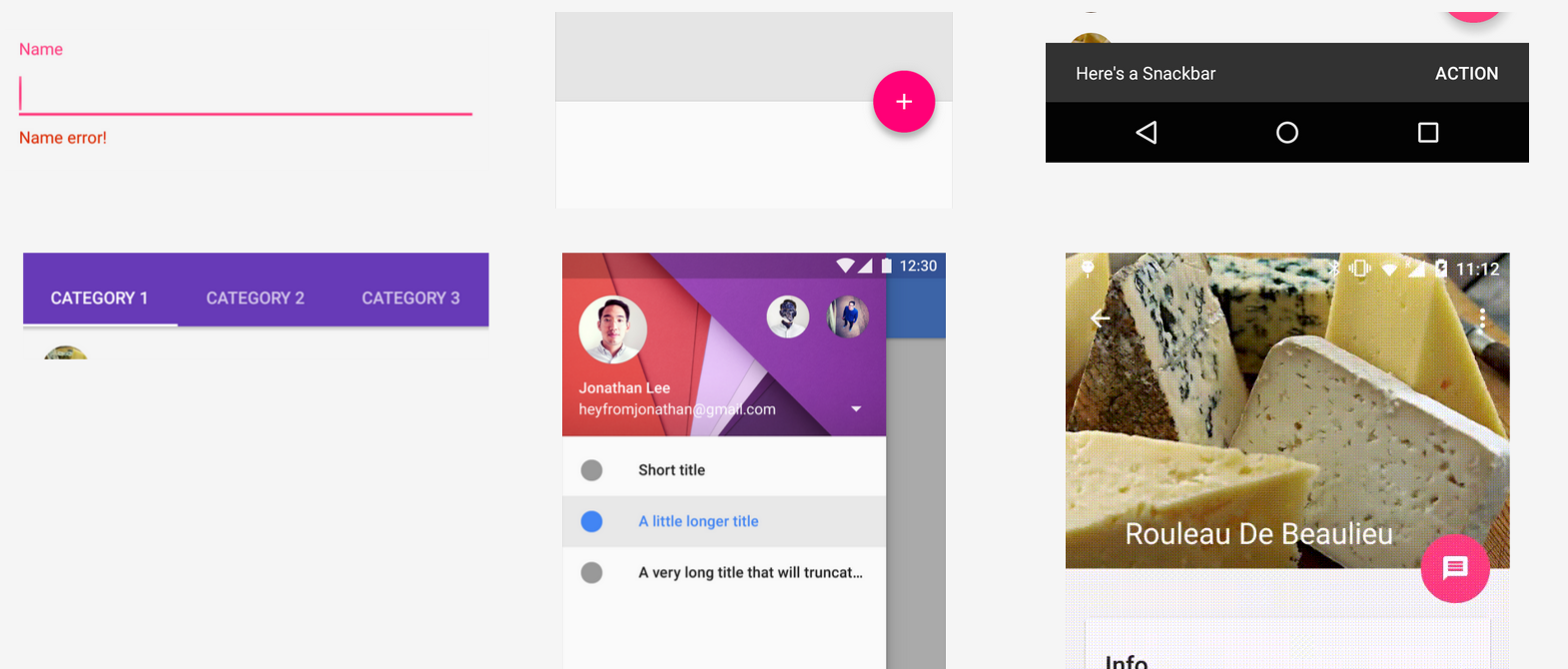
[Add a Floating Action Button (FAB) and trigger a SnackBar](#h.1ksv4uv)

Congratulations! You have finished the code lab!

[Optional Lessons](#h.44sinio)

# Overview

Duration: 0:15



Material Design is a visual language that synthesizes the classic principles of good design with the innovation and possibility of technology and science. In this codelab, you’ll learn the principles of this design language by building a sample Android app.

### What you’ll learn

* Material Design principles:
  + Pseudo-physical and tangible surfaces
  + Bold, graphic and intentional elements
  + Appropriate, authentic and meaningful motion
* Creating an Android app using the principles outlined above.

### What you’ll need

* Experience developing Android Apps
* A development machine with [Android Studio](https://developer.android.com/sdk/installing/studio.html) version 1.0+ and [JDK 7+](http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html)
* A test device with Android 5.0 (Lollipop, API Level 21)\*, or an Emulator with Lollipop

\* Android devices running Android 2.3.3 (Gingerbread, API Level 10) or higher may be used, but, some Material Design effects will not be visible.

# Setup

Duration: 0:45

Environment: Web

<https://docs.google.com/document/d/17EBeztg5iHHXtE9cX4Bp_dyB0YFJ_IovG-1Af0bbogA/edit>

# Lessons

Duration: 1:45

Environment: Android Studio

## Lesson 1 - Color and Typography

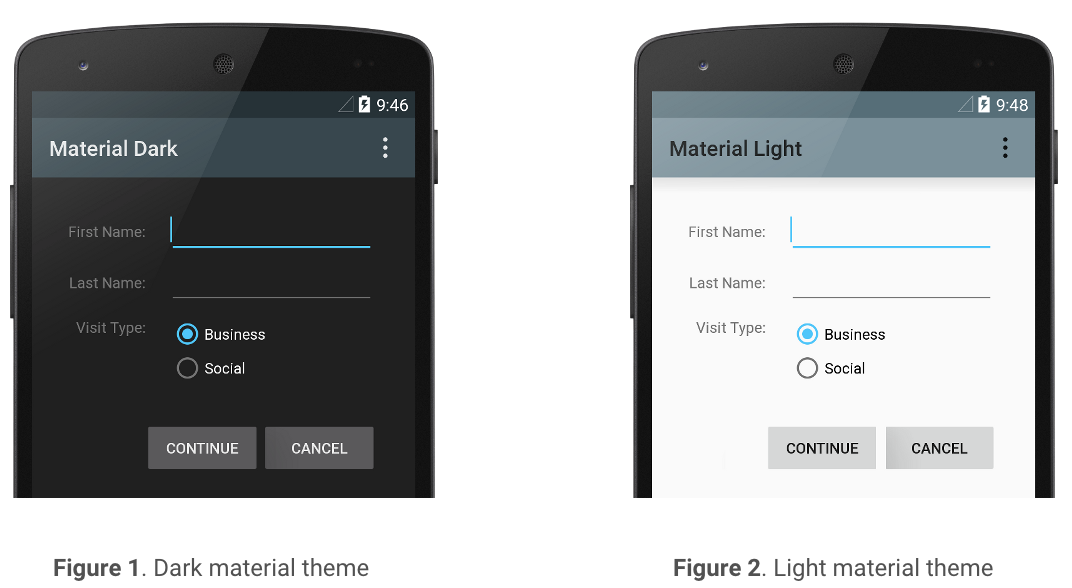
Duration: 0:15

|  |
| --- |
| Material Design Principles covered:   * Print-like Design |

Let’s jump right into one of the key features of Material Design: Themes and Color!

### [Customize the Theme and Color Palette](http://developer.android.com/training/material/theme.html#ColorPalette) -

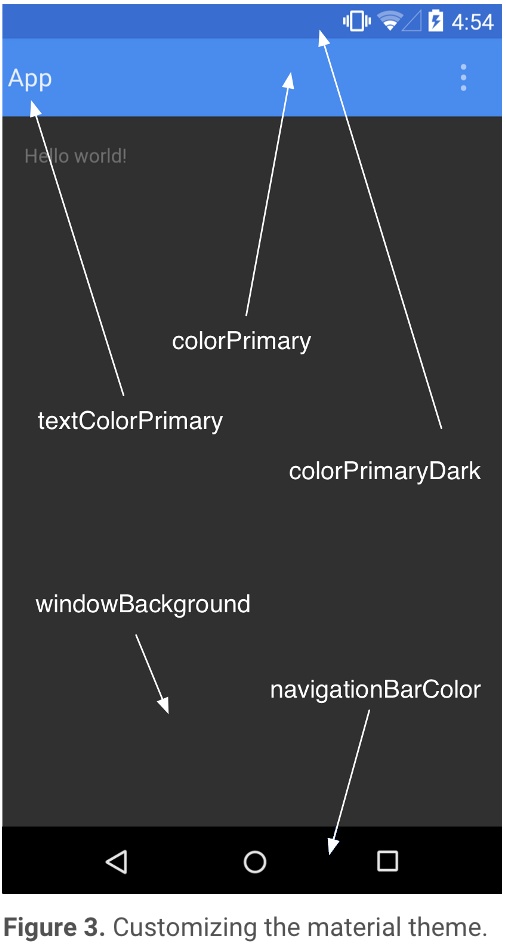
Themes let you apply a consistent tone to an app, and developers can choose between dark or light theme (see Figure 1 and Figure 2).



Custom colors can also be defined using theme attributes which are then

automatically used by the app for different components e.g colorPrimaryDark for the

Status Bar and colorPrimary for the App Bar (see Figure 3).



* 1. Add the Light theme to our app and customize some of the colors in res/values/styles.xml(v21)
  2. Set typeface to Roboto and change the density and orientation of our app text in activity\_main.xml

a. *res/values/styles.xml (v21)*

<**resources**>

<**style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar"**>

*<!-- Customize your theme here. -->*

<**item name="colorPrimary"**>#3F51B5</**item**> <!-- Light Indigo -->

<**item name="colorPrimaryDark"**>#303F9F</**item**> <!-- Dark Indigo -->

<**item name="colorAccent"**>#E91E63</**item**> <!-- Magenta -->

</**style**>

</**resources**>

b. *res/layout/activity\_main.xml*

<**TextView**

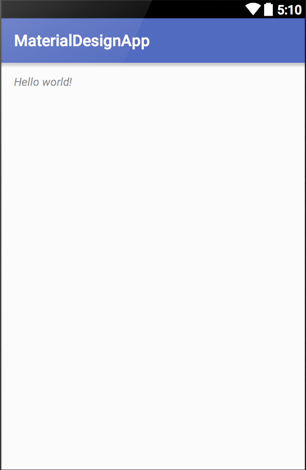
**android:text="@string/hello\_world"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:textStyle="bold|italic"**

**android:fontFamily="sans-serif-thin"**/> <!-- Roboto Typeface -->

App should now look like this:

Extra Credit : If you have some time left at the end, play around with the [typeface](http://www.google.com/design/spec/style/typography.html#typography-language-categorization) and [color](http://www.google.com/design/spec/style/color.html#color-color-palette) as you add more content to your app.

## Lesson 2 - Layout and Animation

Duration: 0:45

|  |
| --- |
| Material Design Principles covered:   * Tangible Surfaces * Bold Elements * Meaningful Motion |

### Add a Top and Bottom [ToolBar](http://www.google.com/design/spec/layout/structure.html#structure-toolbars)

* 1. Create a file called [menu\_bottom.xml](https://drive.google.com/a/google.com/file/d/0B8fmn9WHMkHYblZ1by0wZTBEZVk/view) containing 3 standard menu items (copy, cut and microphone), and place it under res/menu/ folder
  2. In activity\_main.xml, add a top and bottom ToolBar around the TextView
  3. In MainActivity.java, inflate the menu items from (a) into the bottom ToolBar

b. res/layout/activity\_main.xml

<**RelativeLayout**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**

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

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

<**android.support.v7.widget.Toolbar**

**android:id="@+id/toolbar"**

**android:layout\_width="match\_parent"**

**android:layout\_height="?attr/actionBarSize"**

**app:title="@string/app\_name"**

**android:background="?attr/colorPrimaryDark"**

**android:theme="@style/ThemeOverlay.AppCompat.Dark"**/>

<!-- TextView **-->**

<**android.support.v7.widget.Toolbar**

**android:id="@+id/bottom\_toolbar"**

**android:layout\_height="?attr/actionBarSize"**

**android:layout\_width="match\_parent"**

**android:theme="@style/Base.ThemeOverlay.AppCompat.Dark"**

**android:background="?attr/colorPrimaryDark"**

**android:layout\_alignParentBottom="true"/**>

</**RelativeLayout**>

c. java/.../MainActivity.java

**public class** MainActivity **extends** AppCompatActivity {

@Override

**protected void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.***activity\_main***);

/\* Top toolbar \*/

Toolbar toolbar = (Toolbar) findViewById(R.id.***toolbar***);  
 setSupportActionBar(toolbar);

*/\* Bottom toolbar. \*/*

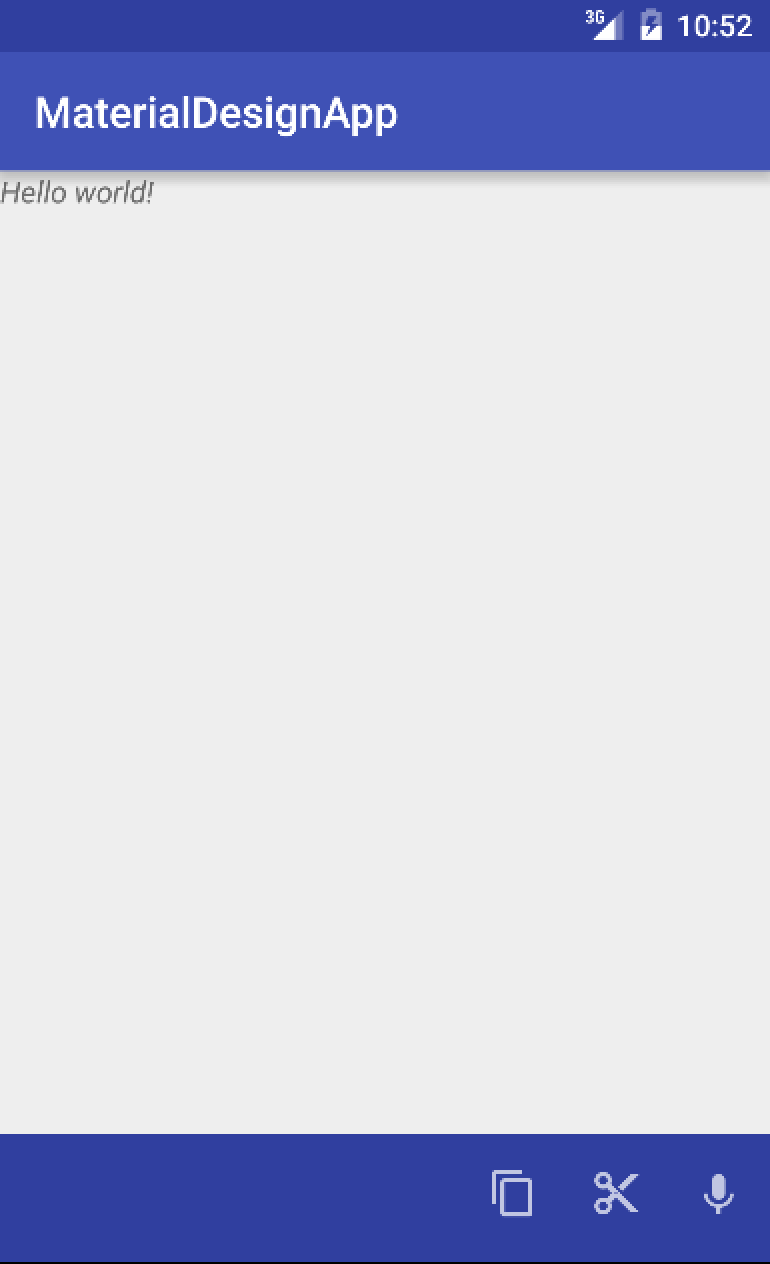
Toolbar bottomToolbar = (Toolbar) findViewById(R.id.***bottom\_toolbar***);

bottomToolbar.inflateMenu(R.menu.***menu\_bottom***);

}

...

}

App should now look like this: 

Extra credit:

### Add a [RecyclerView](https://developer.android.com/training/material/lists-cards.html) and [Cards](http://www.google.com/design/spec/components/cards.html) with [Ripples](https://www.google.com/design/spec/animation/responsive-interaction.html)

RecyclerView is a container for displaying large data sets that can be scrolled very

efficiently by maintaining a limited number of views.

Our data set for this code lab are empty cards which are pieces of paper that serve as an

entry point to more information.

Let’s add a RecyclerView and some cards to our app:

* 1. Create a file defining a card called [cardview.xml](https://docs.google.com/document/d/1fxf_xIoyrhiJgyZHh2jDBTzNz8fhpNZcu7DFEXb7Pw8/edit?usp=sharing) under res/layout/. Take a look at the individual attributes to see what customizations are available
  2. In activity\_main.xml, replace the Text View with a [CoordinatorLayout](https://developer.android.com/reference/android/support/design/widget/CoordinatorLayout.html) (enables components to trigger actions based on interactions) containing a RecyclerView
  3. In MainActivity.java, create and customize the RecyclerView by adding the cards using the view defined in (a) via a [RecyclerView.Adapter.](https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html)
     1. Add a scroll animation to the cards with the RecyclerView
     2. Add a ripple (press and release) animation to the cards

b. res/layout/activity\_main.xml

**<RelativeLayout**

**<!-- Top Toolbar -->**

<**android.support.design.widget.CoordinatorLayout**

**android:id="@+id/main\_content"**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**

**android:layout\_marginTop="?attr/actionBarSize"**

**android:layout\_marginBottom="?attr/actionBarSize"**

**android:layout\_alignParentTop="true"**>

<**android.support.v7.widget.RecyclerView**

**android:id="@+id/my\_recycler\_view"**

**android:scrollbars="vertical"**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**

**android:paddingTop="8dp"**

**android:paddingBottom="8dp"**

**android:clipToPadding="false"**

**app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior"**/>

</**android.support.design.widget.CoordinatorLayout**>

**<!-- Bottom Toolbar -->**

</**RelativeLayout**>

c. java/.../MainActivity.java

**public class** MainActivity **extends** AppCompatActivity {

@Override

**protected void** onCreate(Bundle savedInstanceState) {

*/\* Top toolbar. \*/*

*/\* Bottom toolbar. \*/*

*/\* Create and customize RecyclerView. \*/*

RecyclerView recyclerView = (RecyclerView) findViewById(R.id.***my\_recycler\_view***);

recyclerView.setHasFixedSize(**true**);

recyclerView.setLayoutManager(**new** LinearLayoutManager(**this**));

*// Add 8 cards*

MyAdapter adapter = **new** MyAdapter(**new** String[8]);

recyclerView.setAdapter(adapter);

}

*/\** Create RecylcerView Adapter*. \*/*

**public static class** MyAdapter **extends** RecyclerView.Adapter<MyAdapter.ViewHolder> {

**private** String[] **mDataset**;

**public static class** ViewHolder **extends** RecyclerView.ViewHolder {

**public** View **view**;

**public** TextView **title**;

**public** ViewHolder(View v) {

**super**(v);

**view** = v;

**title** = (TextView) v.findViewById(R.id.card\_title);

}

}

**public** MyAdapter(String[] myDataset) {

**mDataset** = myDataset;

}

@Override

**public** MyAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, **int** viewType) {

View cardview = LayoutInflater.*from*(parent.getContext())

.inflate(R.layout.***cardview***, parent, **false**);

**return** **new** ViewHolder(cardview);

}

@Override

**public void** onBindViewHolder(ViewHolder holder, **int** position) {

holder.title.setText("Card " + (position + 1));

}

@Override

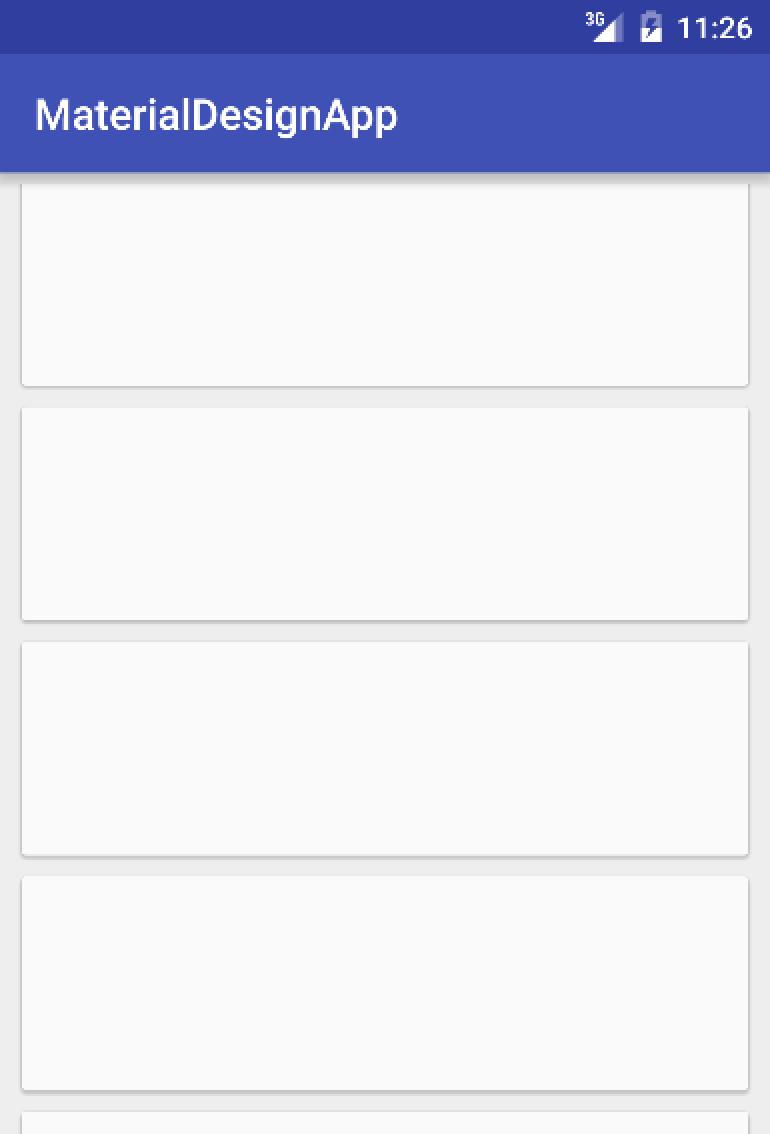
**public int** getItemCount() {

**return mDataset**.**length**;

}

}

}

App should now look like this: 

Extra Credit : Add [Lift on Touch](https://www.google.com/design/spec/animation/responsive-interaction.html#responsive-interaction-material-response) animation.

### Add an Image to a [CollapsingToolbar](https://developer.android.com/reference/android/support/design/widget/CollapsingToolbarLayout.html)

A Collapsing Toolbar provides visual transitions by collapsing a toolbar as the user

scrolls down the app.

Let’s go ahead and add a Collapsing Toolbar:

* 1. Download this [image](http://www.gettyimages.co.uk/gi-resources/images/Homepage/Category-Creative/UK/UK_Creative_462809583.jpg) or any image of your choice and add it under res/drawable folder as image.png
  2. In activity\_main.xml, add an [AppBarLayout](https://developer.android.com/reference/android/support/design/widget/AppBarLayout.html) (enables components react to scrolling) containing a CollapsingToolbar which consists of:
     1. An ImageView of the image from (a)
     2. A ToolBar that the image collapses into

These should be added within the CoordinatorLayout created in (2) above

* 1. In MainActivity.java, create a [Palette](https://developer.android.com/reference/android/support/v7/graphics/Palette.html) instance from the bitmap of the image referred to by the ImageView
     1. Using the Palette created, asynchronously set the collapsed toolbar color

b. res/layout/activity\_main.xml

<**RelativeLayout**

**<!-- Top Toolbar -->**

<**android.support.design.widget.CoordinatorLayout**

**android:id="@+id/main\_content"**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**

**android:layout\_marginBottom="?attr/actionBarSize"**

**android:layout\_alignParentTop="true"**>

<**android.support.design.widget.AppBarLayout**

**android:id="@+id/appbar"**

**android:layout\_width="match\_parent"**

**android:layout\_height="192dp"**>

<**android.support.design.widget.CollapsingToolbarLayout**

**android:id="@+id/collapsing\_toolbar"**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**

**android:theme="@style/ThemeOverlay.AppCompat.Dark"**

**app:layout\_scrollFlags="scroll|exitUntilCollapsed"**>

<**ImageView**

**android:id="@+id/image"**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**

**android:background="@drawable/image"**

**android:fitsSystemWindows="true"**

**android:scaleType="centerCrop"**

**app:layout\_collapseMode="parallax"** />

<**android.support.v7.widget.Toolbar**

**android:id="@+id/toolbar"**

**android:layout\_width="match\_parent"**

**android:layout\_height="?attr/actionBarSize"**

**app:popupTheme="@style/ThemeOverlay.AppCompat.Light"**

**app:layout\_collapseMode="pin"** />

</**android.support.design.widget.CollapsingToolbarLayout**>

</**android.support.design.widget.AppBarLayout**>

**<!-- RecyclerView -->**

</**android.support.design.widget.CoordinatorLayout**>

**<!-- Bottom Toolbar -->**

</**RelativeLayout**>

c. java/.../MainActivity.java

**public class** MainActivity **extends** AppCompatActivity {

**private** CollapsingToolbarLayout **ctb**;

**private int mutedColor**;

@Override

**protected void** onCreate(Bundle savedInstanceState) {

/\* Top toolbar \*/

/\* Bottom toolbar \*/

/\* RecyclerView \*/

/\* Cards \*/

/\* Collapsing toolbar \*/

ctb = (CollapsingToolbarLayout) findViewById(R.id.***collapsing\_toolbar***);

*/\* Define the image \*/*

ImageView image = (ImageView) findViewById(R.id.***image***);

*/\* Decode bitmap from the image \*/*

Bitmap bitmap = BitmapFactory.*decodeResource*(getResources(), R.drawable.***image***);

*/\* Generate palette from the image bitmap \*/*

Palette.*from*(bitmap).generate(**new** Palette.PaletteAsyncListener() {

@Override

**public void** onGenerated(Palette palette) {

**mutedColor** = palette.getMutedColor(R.attr.***colorPrimary***);

*/\* Set toolbar color from the palette \*/*

ctb.setContentScrimColor(**mutedColor**);

}

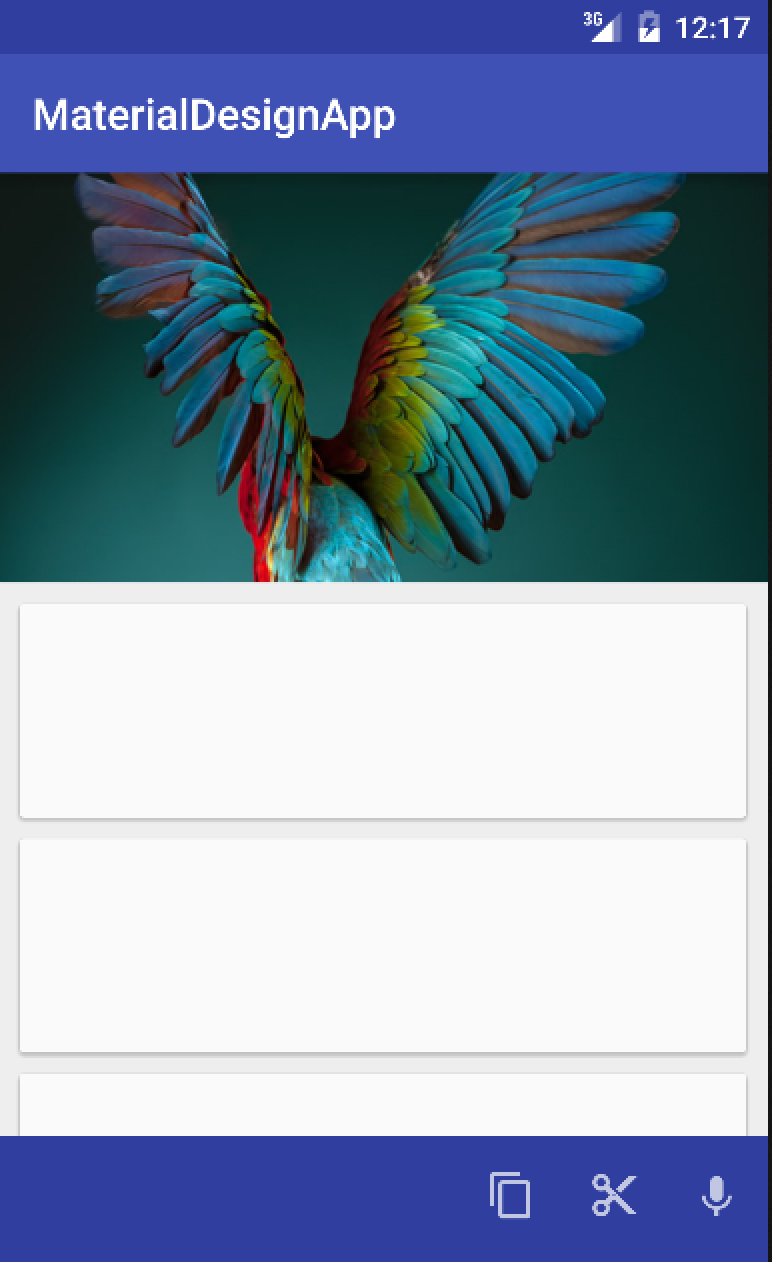
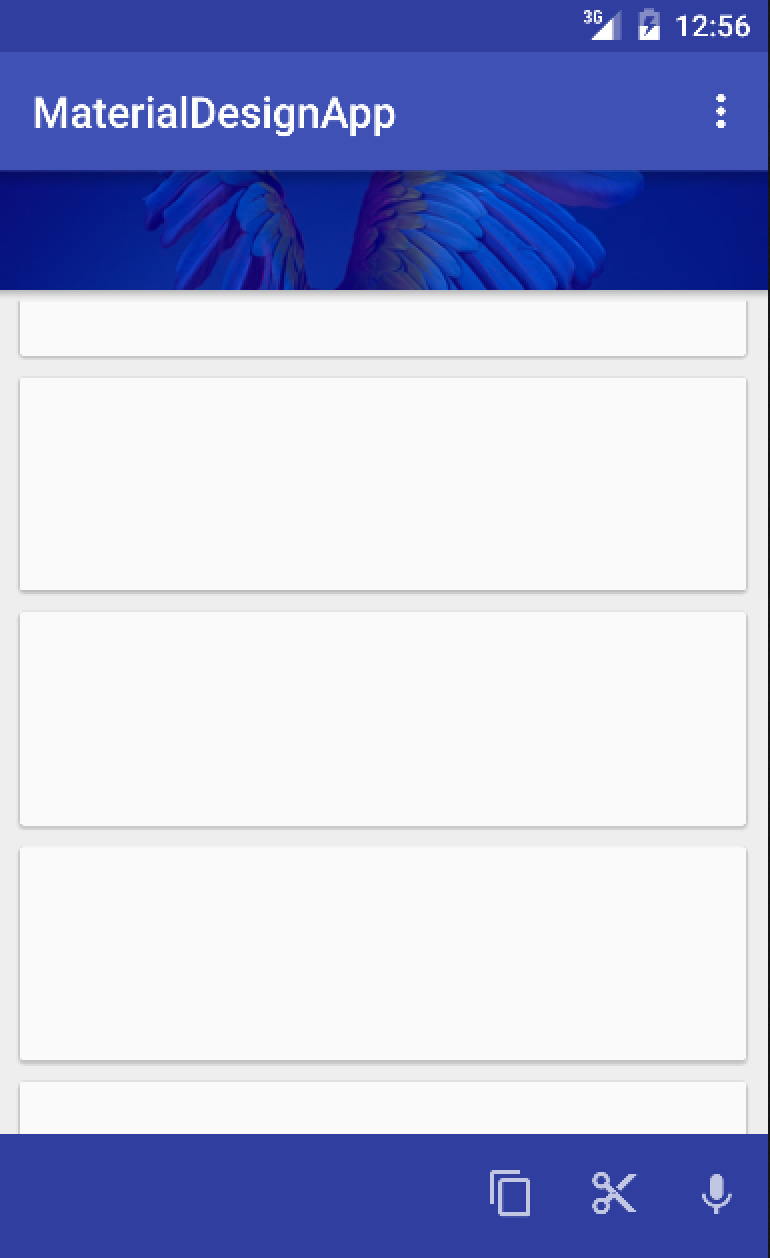
});

}

...

}

App should now look like this:

Before collapse: After collapse: 

# 

## Lesson 3 - Page Elements

Duration: 0:45

|  |
| --- |
| Material Design Principles covered:   * Tangible Surfaces * Bold Elements |

### Add a [NavigationDrawer](http://www.google.com/design/spec/patterns/navigation-drawer.html)

The navigation drawer slides in from the left. It is a common pattern found in Google apps and follows the keylines and metrics for lists.

Let’s add a NavigationDrawer and set it to open when selected:

* 1. Create a file [menu\_navigation.xml](https://drive.google.com/a/google.com/file/d/0B8fmn9WHMkHYU1gyS1psNVlxOEk/view) defining the navigation items under res/menu folder
  2. Create a file [navheader.xml](https://drive.google.com/a/google.com/file/d/0B8fmn9WHMkHYSkVFZTVqdjlBRmM/view) defining a Navigation Drawer material under res/layout/ folder
  3. Create a file [ic\_menu\_24dp.xml](https://drive.google.com/a/google.com/file/d/0B8fmn9WHMkHYRGVKSFlhNzRpM2c/view) defining the navigation menu under res/drawable folder (to support versions of Android before Lollipop, download the icon from [here](https://www.google.com/design/icons/#ic_menu))
  4. In activity\_main.xml:
     1. Encapsulate all the components within a [DrawerLayout](https://developer.android.com/reference/android/support/v4/widget/DrawerLayout.html) which enables interactive drawer views to be pulled out from the edge of the window.
     2. Add a NavigationView outside the RelativeLayout
  5. In MainActivity.java:
     1. Add navigation menu to the ActionBar
     2. Enable open and close of NavigationView

c. res/layout/activity\_main.xml

<**android.support.v4.widget.DrawerLayout**

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

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

**android:id="@+id/drawer"**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**

**android:fitsSystemWindows="true"**>

**<RelativeLayout**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**>

.

.

.

</**RelativeLayout**>

<**android.support.design.widget.NavigationView**

**android:id="@+id/nav\_view"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="match\_parent"**

**android:layout\_gravity="start"**

**android:background="#ffffff"**

**android:clickable="true"**

**app:headerLayout="@layout/navheader"**

**app:menu="@menu/menu\_navigation"**

**app:itemBackground="?attr/selectableItemBackground"**/>

</**android.support.v4.widget.DrawerLayout**>

c. java/.../MainActivity.java

**public class** MainActivity **extends** AppCompatActivity {

**private** CollapsingToolbarLayout **ctb**;

**private int mutedColor**;

**private** DrawerLayout **drawerLayout**;

**private** NavigationView **navigationView**;

@Override

**protected void** onCreate(Bundle savedInstanceState) {

/\* Top toolbar \*/

/\* Bottom toolbar \*/

/\* RecyclerView \*/

/\* Cards \*/

/\* Collapsing toolbar \*/

/\* NavigationView \*/

**navigationView** = (NavigationView) findViewById(R.id.***nav\_view***);

**drawerLayout** = (DrawerLayout) findViewById(R.id.***drawer***);

*// On click of menu icon on toolbar*

**toolbar**.setNavigationIcon(R.drawable.**ic\_menu\_24dp**);

**toolbar**.setNavigationOnClickListener(new View.OnClickListener() {

@Override

**public void** onClick(View v) {

**drawerLayout**.openDrawer(GravityCompat.**START**);

}

});

*// On click of the navigation menu*

**navigationView**.setNavigationItemSelectedListener(**new** NavigationView.OnNavigationItemSelectedListener() {

*// This method will trigger on item Click of navigation menu*

@Override

**public boolean** onNavigationItemSelected(MenuItem menuItem) {

*// Set item in checked state*

menuItem.setChecked(**true**);

*//TODO: handle navigation*

*//Closing drawer on item click*

**drawerLayout**.closeDrawers();

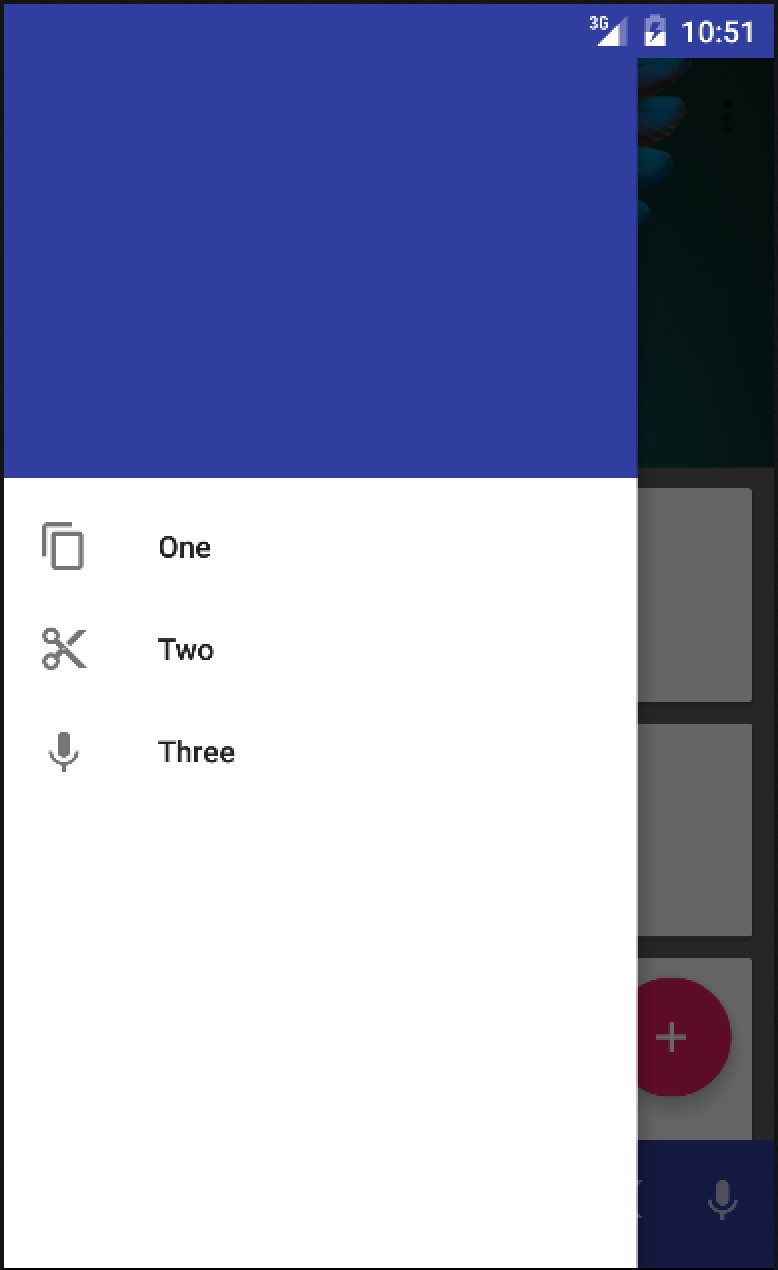
**return true**;

}

});

}

}

App should now look like this when menu is selected: 

### Add a [Floating Action Button (FAB)](http://www.google.com/design/spec/components/buttons-floating-action-button.html) and trigger a [SnackBar](http://www.google.com/design/spec/components/snackbars-toasts.html)

Floating action buttons are used for a promoted action and are distinguished by a

circled icon floating above the UI.

Let’s create a FAB that triggers a SnackBar which provides lightweight feedback

feedback about an operation by showing a brief message:

* 1. Download this [icon](https://developer.android.com/samples/FloatingActionButtonBasic/res/drawable-xxhdpi/ic_add.png) and add it under res/drawable/ as ic\_add.png
  2. In activity\_main.xml:
     1. Add a FloatingActionButton to the end of the CoordinatorLayout, after the RecyclerView.
     2. Set the source(src attribute) of the FAB to the icon from (a)
  3. In MainActivity.java, add a Listener to the FAB that creates a SnackBar onclick

c. res/layout/activity\_main.xml

<**android.support.v4.widget.DrawerLayout**

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

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

**android:id="@+id/drawer"**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**

**android:fitsSystemWindows="true"**>

**<RelativeLayout**

**<android.support.design.widget.CoordinatorLayout**

**.**

**.**

**<android.support.design.widget.FloatingActionButton**

**android:id="@+id/fab"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:layout\_marginBottom="16dp"**

**android:layout\_marginRight="16dp"**

**android:layout\_above="@+id/bottom\_toolbar"**

**android:layout\_gravity="right|bottom"**

**android:tint="#ffffff"**

**android:src="@drawable/ic\_add"/>**

**</android.support.design.widget.CoordinatorLayout>**

**.**

**.**

</**RelativeLayout**>

</**android.support.v4.widget.DrawerLayout**>

*MainActivity.java*

@Override

**protected void** onCreate(Bundle savedInstanceState) {

/\* Top toolbar \*/

/\* Bottom toolbar \*/

/\* RecyclerView \*/

/\* Cards \*/

/\* Collapsing toolbar \*/

/\* NavigationView \*/

*/\* Floating Action Button. \*/*

FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.***fab***);

fab.setOnClickListener(**new** View.OnClickListener() {

@Override

**public void** onClick(View v) {

Snackbar.*make*(v, **"Do something awesome?"**,

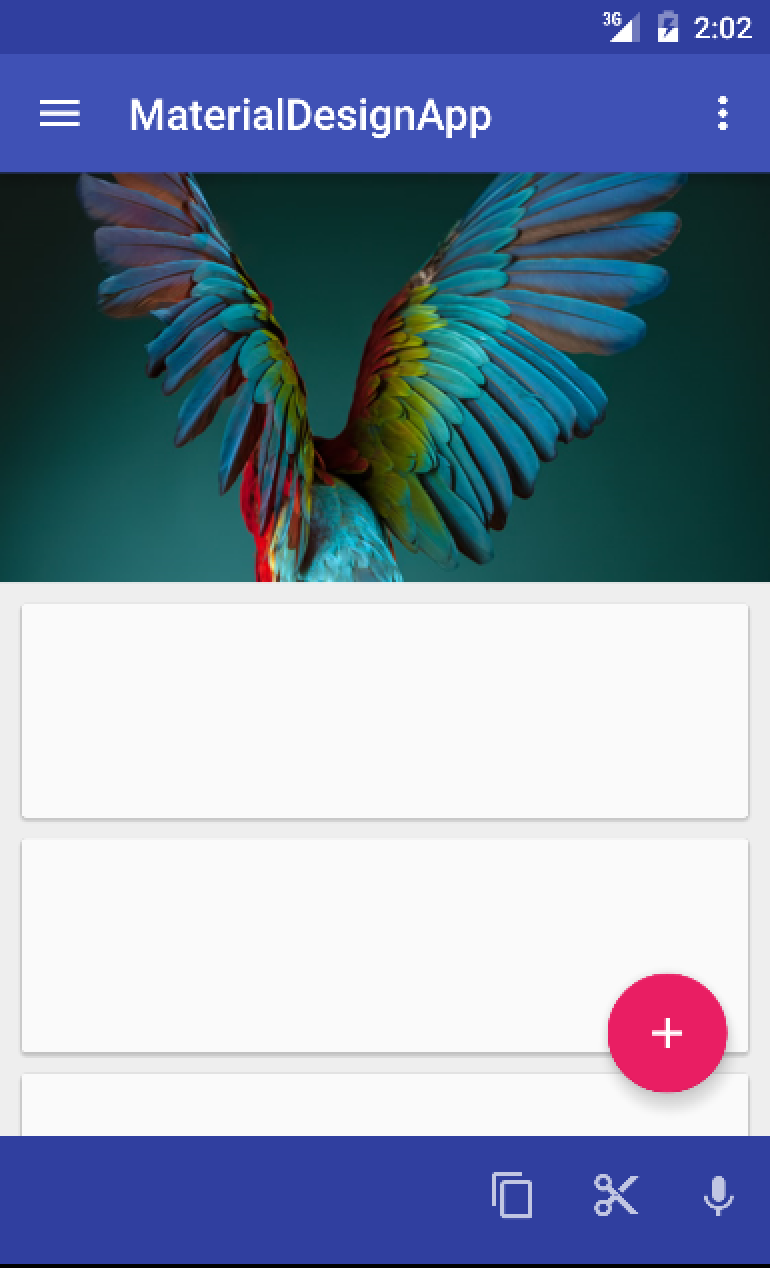
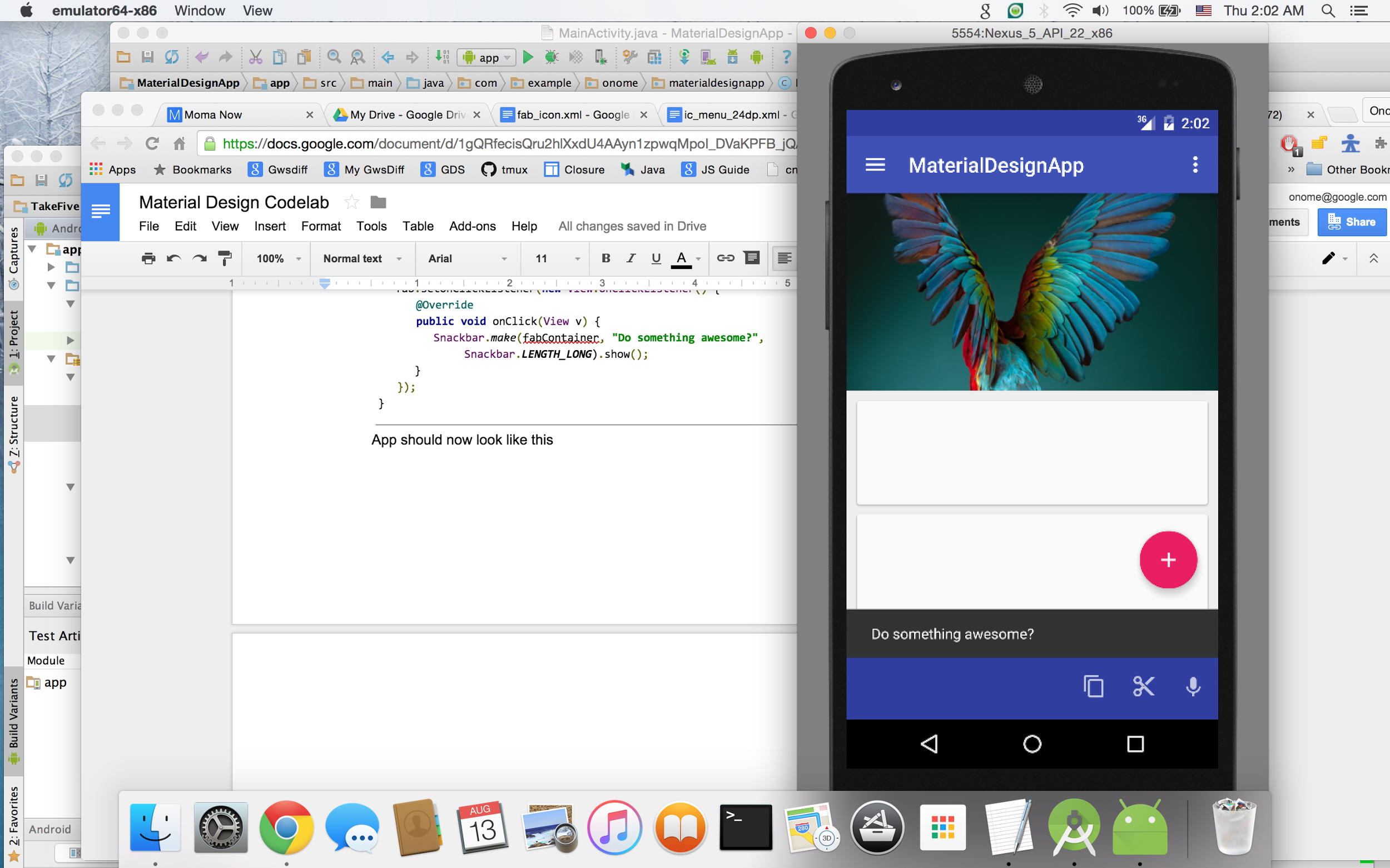
Snackbar.***LENGTH\_LONG***).show();

}

});

}

App should now look like this:

Before clicking FAB:After clicking FAB:

Congratulations! You have finished the code lab!

# Optional Lessons

Duration: 0:15 - ~

If you finish the codelab with time to spare, feel free to:

1. Trigger actions based on card selection in the RecyclerView
2. Trigger actions based on item selection in the NavigationView
3. Add more content to your app
4. Play around with the colors, themes and typography