Software Engineering Large Practical: Design document and Material Design

Stephen Gilmore
School of Informatics

October 27, 2017

Contents

Design document

Design document

- Four weeks ago you were given a specification of an Android-based mobile phone game to implement.
- This described the game in general terms of what was required, but left many design decisions for you to make about how things would be done.
- Part 2 of the practical requires you to submit a design document reporting on your decisions so far.
- Your design document should state abstract implementation decisions and show concrete user interface designs.

Design document recap [from the coursework specification]

- Your design document should record your design decisions for your app including a definitive list of the bonus features which are offered by your app.
- Your design document should represent the visual layout of your app by being illustrated with screenshots of the views which you have designed for the activities in your app.
- You should aim to show a typical play of the game including at least the user viewing a map, guessing which song this is, and being informed whether or not they are correct.
- The expected length of this design document is between 6 and 10 pages.

Typical questions for the design document to answer (1/2)

- How is the song that the player has to identify chosen?
- What does a player have to do to collect a word? Does anything happen to the placemark when the word is collected?
- After a player has collected several words can they review them? If so, how do they do that? What does the "review screen" look like?
- When a player thinks that they can guess the song how do they enter their guess?
  - What happens if their guess is correct?
  - What happens if their guess is incorrect?
- When a player thinks that they *can't* guess the song how do they indicate that they give up? What happens then?

Typical questions for the design document to answer (2/2)

- What determines which of the five versions of the map is shown?
- Can the player set the "level of difficulty" of the game? If so, how?
- After a player has identified a song, could that song be chosen again as the puzzle to solve?
- After a player has identified several songs, can they review their list of solved puzzles? What does that "review screen" look like?
- What does the game do if a data connection (4G) is not available? Can it be played at all?

Including screenshots

- The requirement to include some screenshots of views of your app in your design document means that, if you have not done so already, you should begin implementation of your app now and focus on designing your user interface.
- A set of design guidelines for Android apps have been developed with the aim that apps should have a consistent look-and-feel and use the same visual language.
- The design concepts and ideas which encourage this are expressed in the design guidance called *Material design* — `http://material.io`.
- You might choose not to follow these design guidelines, but it seems at least worthwhile to know of their existence.

Material design

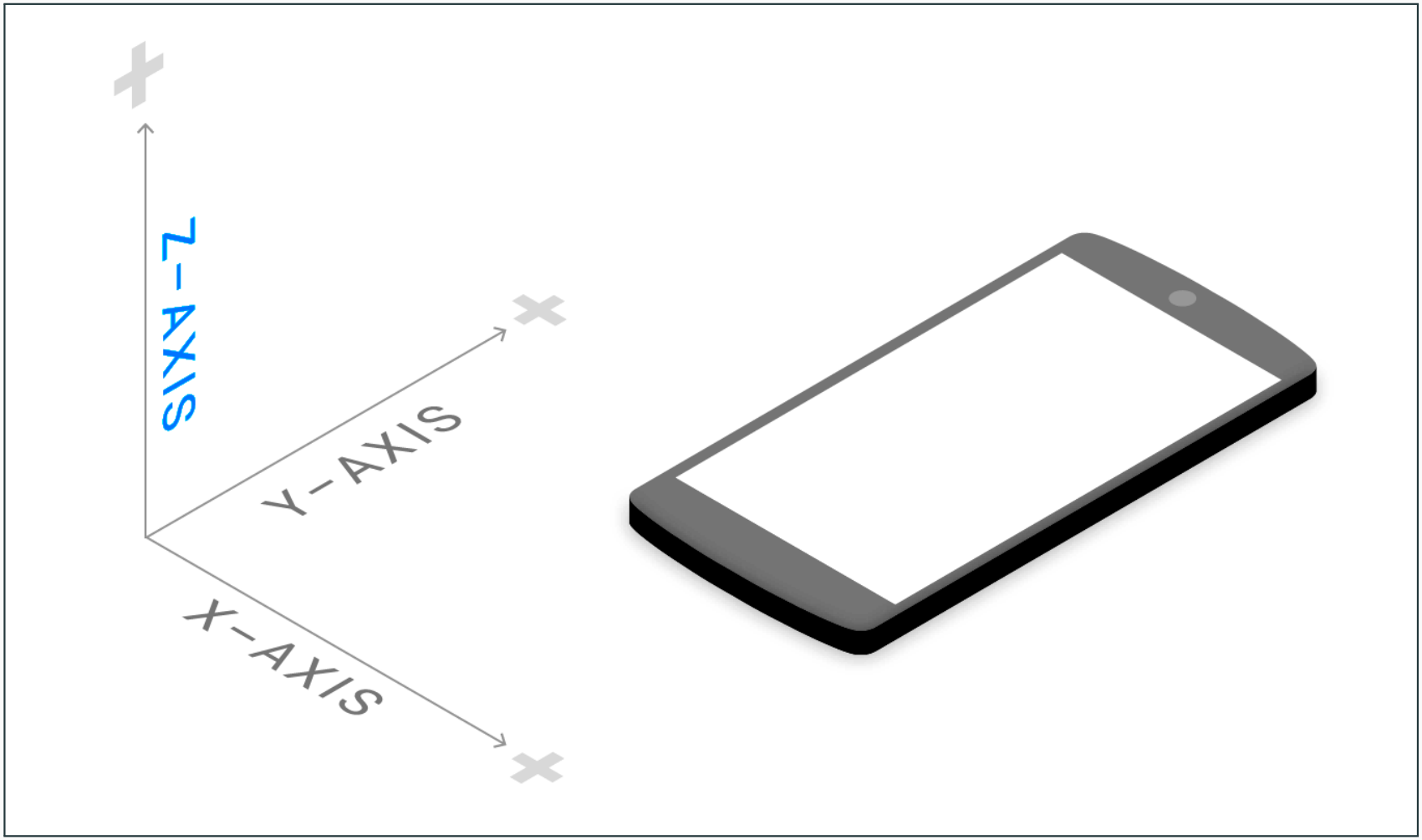
Key features of material design

- Material design is a three-dimensional environment containing light, material, and shadows. All material objects have x, y, and z dimensions.
- Material objects have varying x and y dimensions (measured in dp — "dp" is *device-independent pixel*, pronounced "dip")

---

https://material.io/guidelines/material-design/environment.html

3D space with x, y, and z axes



https://material.io/guidelines/material-design/environment.html

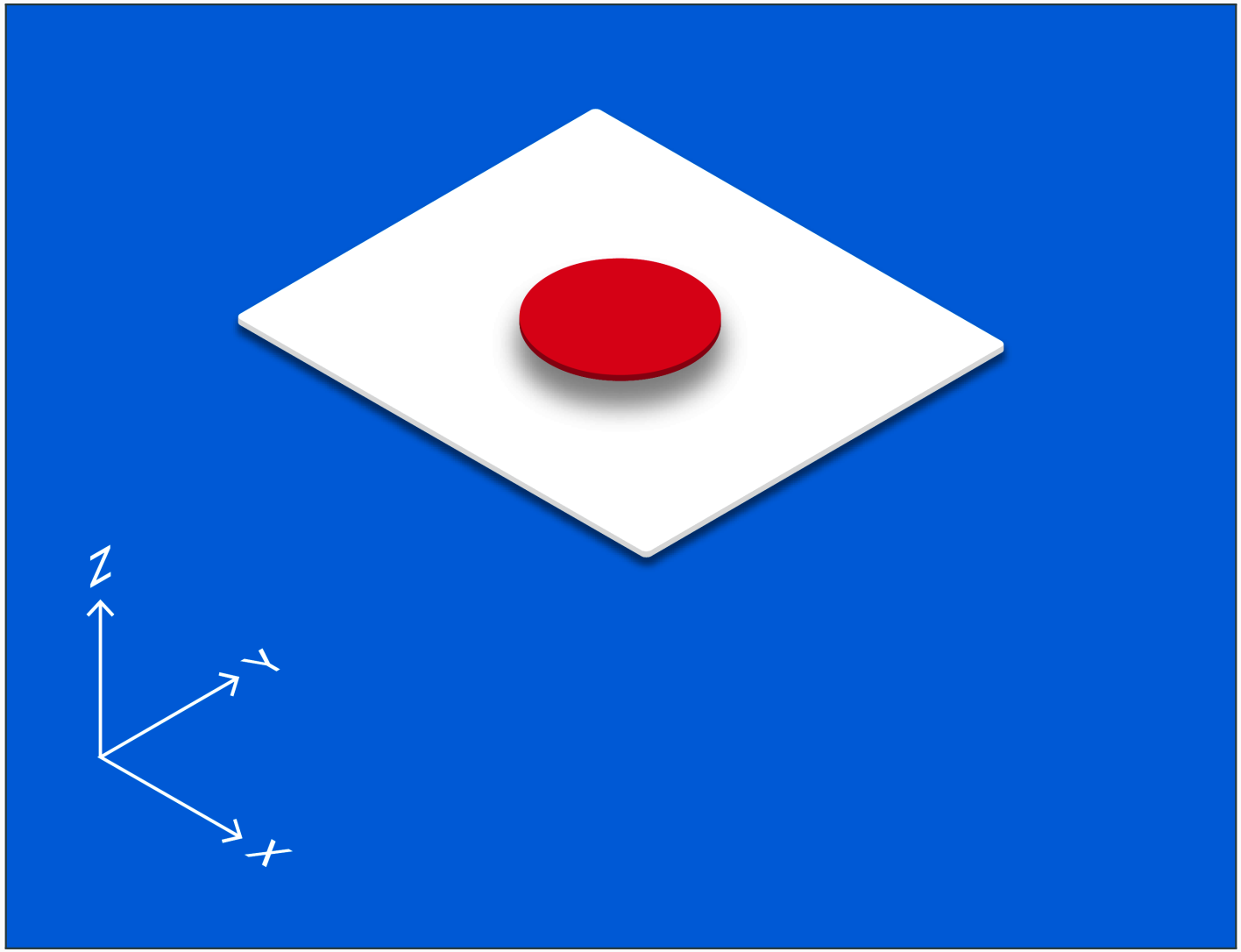
Key features of material design

- All material objects are 1dp thick and have a single z-axis position.
- Key lights create directional shadows, and ambient light creates soft shadows.
- Shadows are created by the elevation difference between overlapping material.

---

https://material.io/guidelines/material-design/environment.html

Co-ordinates and shadows



https://material.io/guidelines/material-design/environment.html

Material properties

- Material is solid.
- Input events cannot pass through material.
  - Input events only affect the foreground material.
- Multiple material elements cannot occupy the same point in space simultaneously.
- Material cannot pass through other material.
  - For example, one sheet of material cannot pass through another sheet of material when changing elevation.

---

https://material.io/guidelines/material-design/material-properties.html

Object elevation — resting elevation

- All material objects, regardless of size, have a *resting elevation*, or default elevation that does not change.
- Components maintain consistent resting elevations across apps.
  - For example, the floating action button's elevation does not vary from one app to another.
- Components may have different resting elevations across platforms and devices, depending on the depth of the environment.

---

https://material.io/guidelines/material-design/elevation-shadows.html

Object elevation — responsive elevation

- Some component types have *responsive elevation*, meaning they change elevation in response to user input (e.g., normal, focused, and pressed) or system events.
  - If an object changes elevation, it should return to its resting elevation as soon as possible.
- These elevation changes are consistently implemented using dynamic elevation offsets.

---

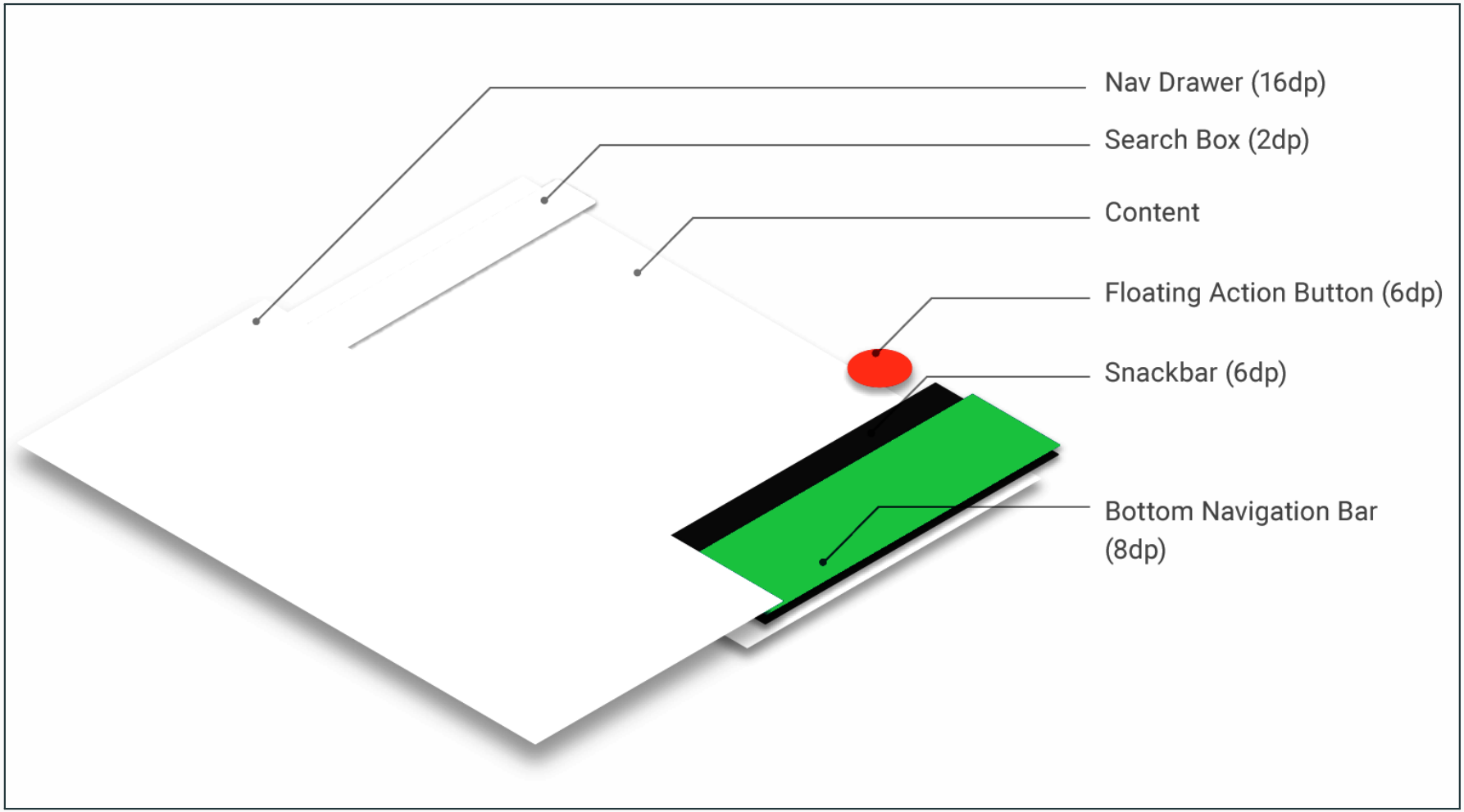
https://material.io/guidelines/material-design/elevation-shadows.html

Components and elevations

Elevation	Component(s)
24 dp	Dialog (a pop-up), Picker (e.g. date picker, time picker)
16 dp	Nav drawer, Right drawer, Modal bottom sheet
12 dp	Floating action button (FAB — pressed)
9 dp	Sub menu (+1dp for each sub menu)
8 dp	Bottom navigation bar, Menu, Card (when picked up), Raised button (pressed state)
6 dp	Floating action button (FAB — resting elevation), Snackbar
4 dp	App Bar
3 dp	Refresh indicator, Quick entry / Search bar (scrolled state)
2 dp	Card (resting elevation), Raised button (resting elevation), Quick entry / Search bar (resting elevation)
1 dp	Switch

https://material.io/components/android/

Components and elevations examples



material.google.com/material-design/elevation-shadows.html

Adding material design to Android applications

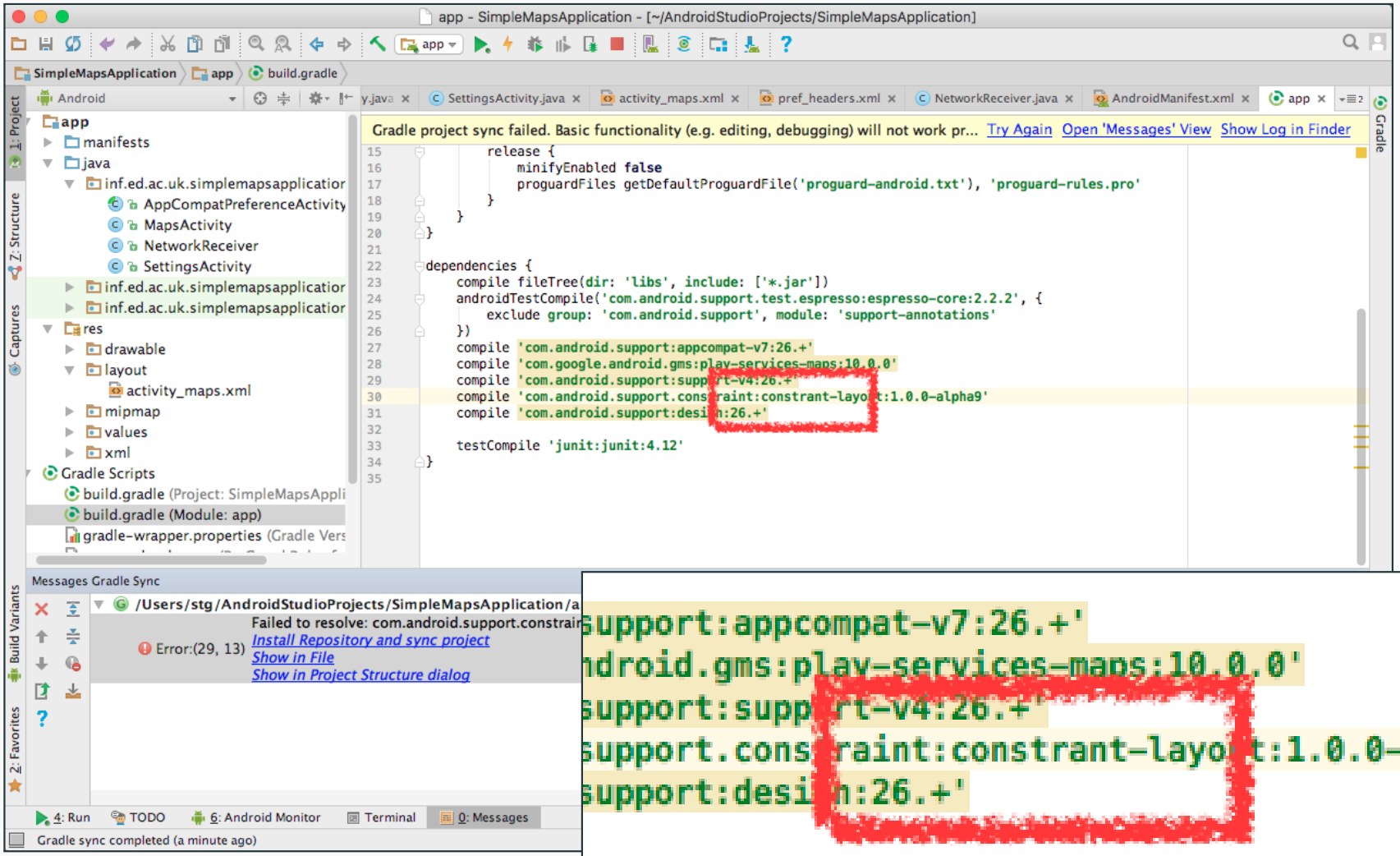
Your build.gradle file should list the dependency on the Material Design support library.

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2',
        {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:26.+'
    compile 'com.android.support.constraint:constraint-layout:1.0.0-alpha9'
    compile 'com.android.support:design:26.+'
    testCompile 'junit:junit:4.12'
}
```

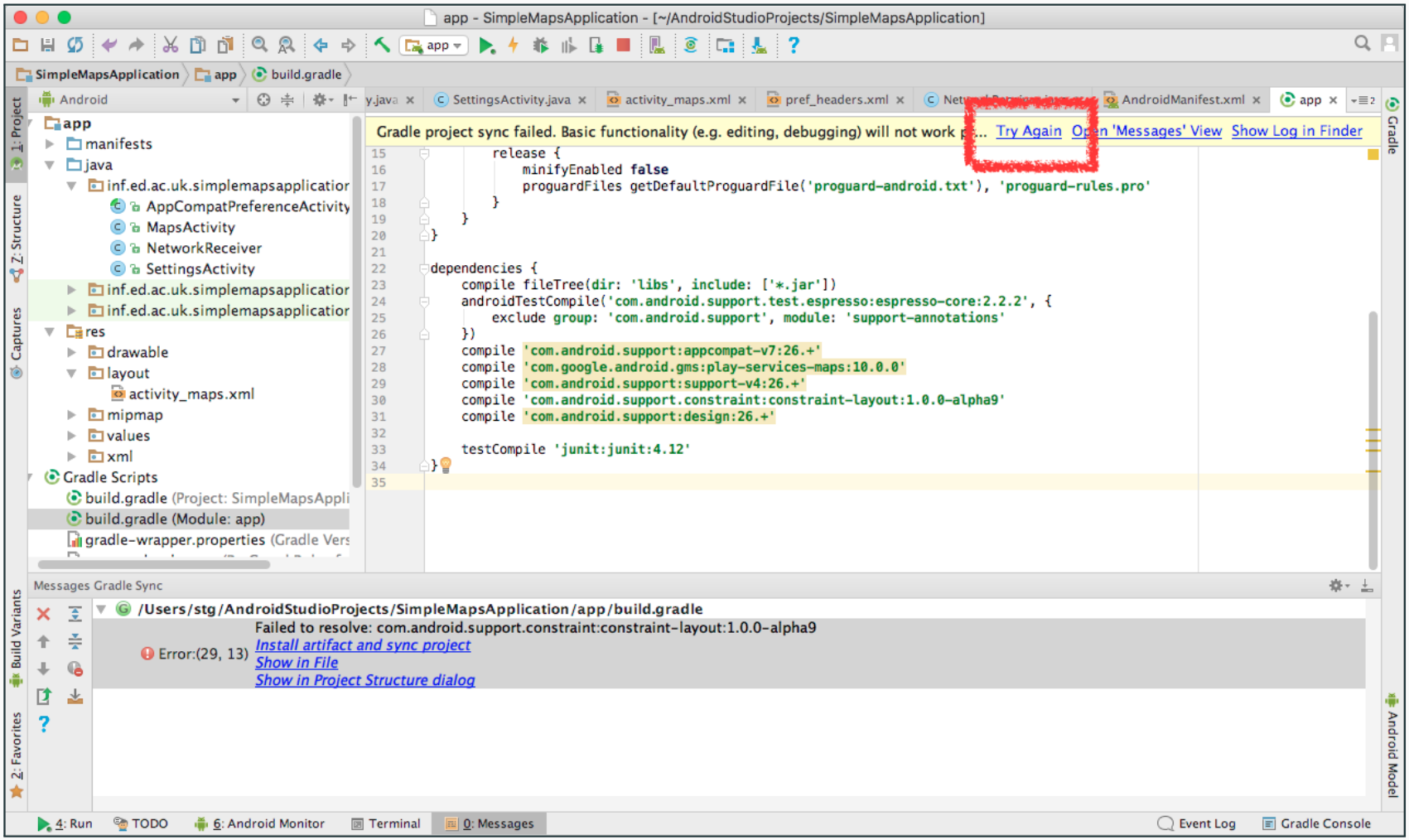
---

https://material.io/components/android/docs/

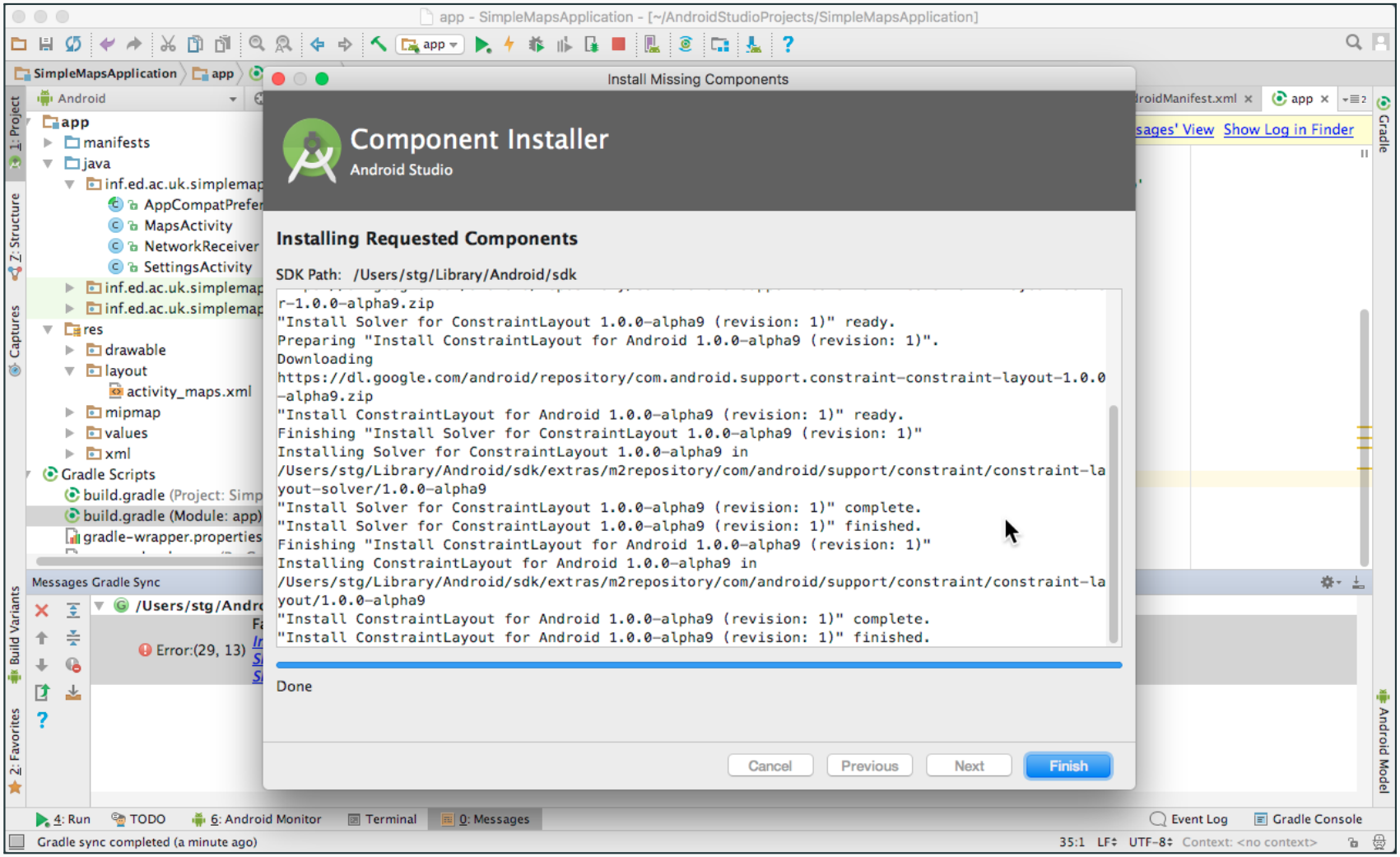
Errors in gradle.build files — look for typos



Typos fixed — try again



Missing components should be installed



Adding material design to Android applications

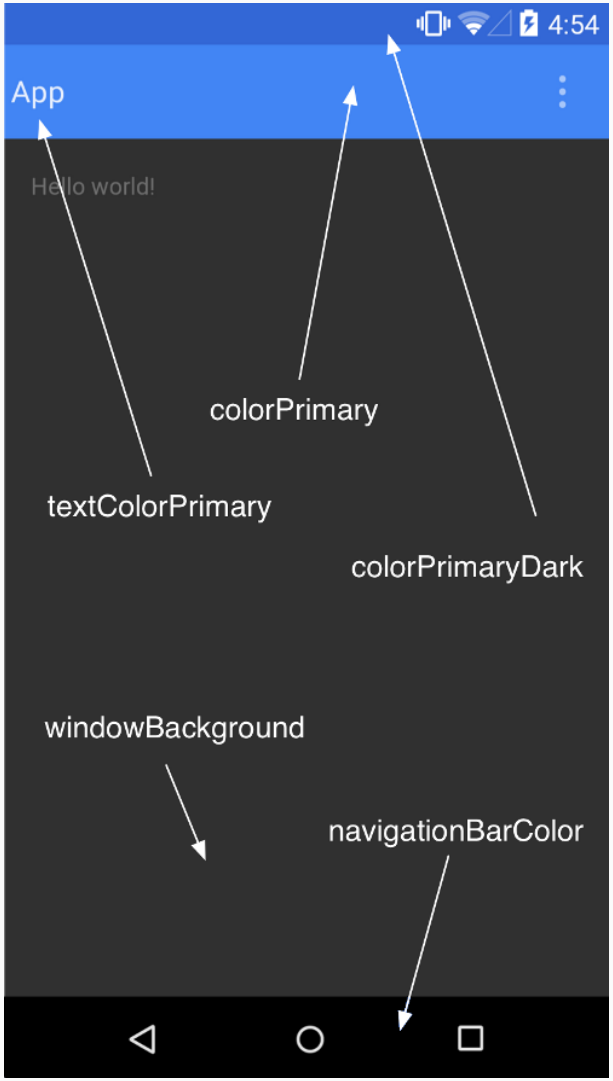
In order to add material design to our Android app we first need to specify that we are using the Material style in our app's styles definition file and customise the theme as necessary.

```
<!-- res/values/styles.xml -->
<resources>
  <!-- your theme inherits from the material theme -->
  <style name="AppTheme" parent="android:Theme.Material">
    <!-- theme customizations -->
  </style>
</resources>
```

---

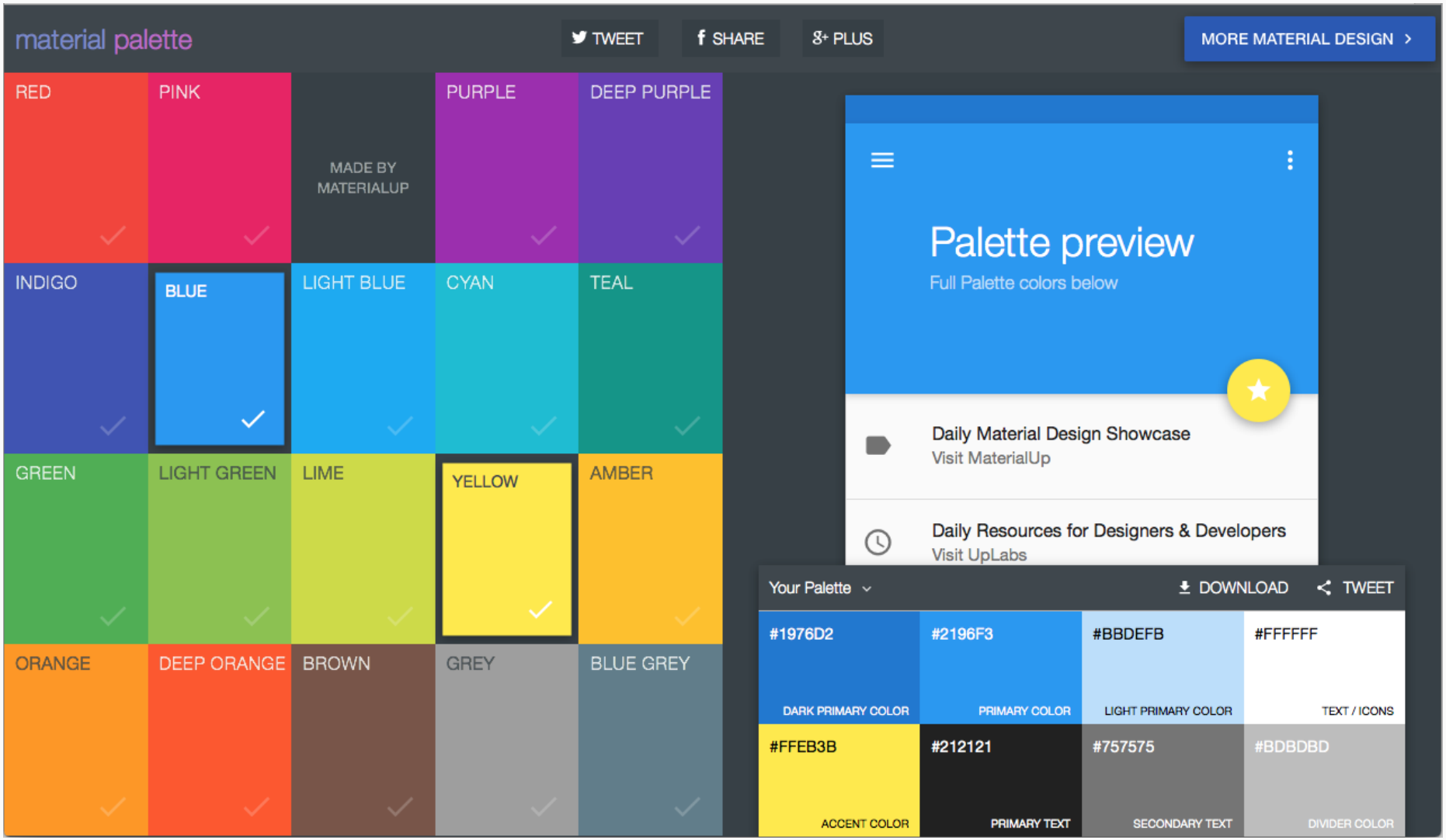
https://developer.android.com/training/material/get-started.html

Customisation options for the material theme



https://developer.android.com/training/material/theme.html

Material design colour palette generator



https://www.materialpalette.com/

XML download offered

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Palette generated by Material Palette -
     materialpalette.com/blue/yellow -->
<resources>
  <color name="primary">#2196F3</color>
  <color name="primary_dark">#1976D2</color>
  <color name="primary_light">#BBDEFB</color>
  <color name="accent">#FFEB3B</color>
  <color name="primary_text">#212121</color>
  <color name="secondary_text">#757575</color>
  <color name="icons">#FFFFFF</color>
  <color name="divider">#BDBDBD</color>
</resources>
```

---

https://www.materialpalette.com/

Setting elevations of views

The elevation of a particular view in our app is set by specifying its android:evelation attribute.

```
<TextView
    android:id="@+id/my_textview"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/next"
    android:background="@color/primary_light"
    android:elevation="5dp" />
```

---

developer.android.com/training/material/get-started.html

Setting up the App Bar at the top of the screen

```
public class MyActivity extends AppCompatActivity {
  // ...
}
```

```
<!-- In AndroidManifest.xml -->
<application
    android:theme="@style/Theme.AppCompat.Light.NoActionBar" />
```

```
<!-- In res/layout/activity_my.xml -->
<android.support.v7.widget.Toolbar
   android:id="@+id/my_toolbar"
   android:layout_width="match_parent"
   android:layout_height="?attr/actionBarSize"
   android:background="@color/primary"
   android:elevation="4dp"
   android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
   app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
```

---

http://developer.android.com/training/appbar/setting-up.html

Methods onCreate and onCreateOptionsMenu

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_my);
    Toolbar myToolbar = (Toolbar) findViewById(R.id.my_toolbar);
    setSupportActionBar(myToolbar);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}
```

---

http://developer.android.com/training/appbar/setting-up.html

Adding action buttons to the App Bar

```
<!-- res/menu/menu_main.xml -->
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <!-- "Mark Favourite", as action button if possible. Download
         icon from https://material.io/icons/ and add to project -->
    <item android:id="@+id/action_favourite"
          android:icon="@drawable/ic_favourite_black_48dp"
          android:title="@string/action_favourite"
          app:showAsAction="ifRoom" />

    <!-- Settings, should always be in the overflow -->
    <item android:id="@+id/action_settings"
          android:title="@string/action_settings"
          app:showAsAction="never" />
</menu>
```

https://developer.android.com/training/appbar/actions.html

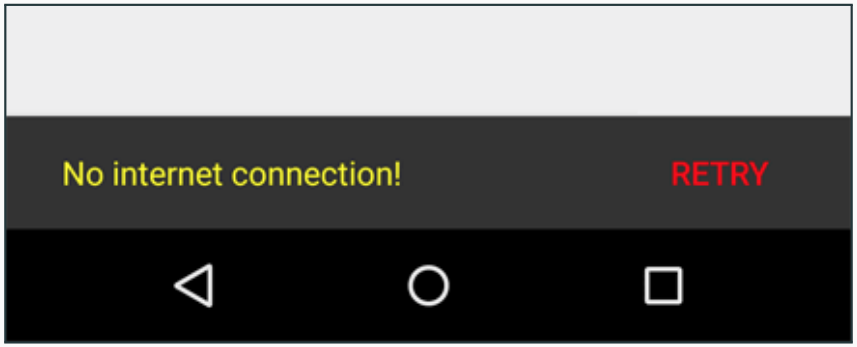
Responding to actions

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_settings:
            // User chose the "Settings" item, show the app settings UI
            return true;
        case R.id.action_favourite:
            // User chose the "Favourite" action, mark the current item
            // as a favourite...
            return true;
        default:
            // If we got here, the user's action was not recognised.
            // Invoke the superclass to handle it.
            return super.onOptionsItemSelected(item);
    }
}
```

https://developer.android.com/training/appbar/actions.html

Building and displaying a message

- You can use a Snackbar to display a brief message to the user, shown for a time LENGTH_SHORT, LENGTH_LONG, or LENGTH_INDEFINITE.
- A snackbar is ideal for brief messages that the user doesn't necessarily need to act on.
- For example, an email app could use a snackbar to tell the user that the app successfully archived an email message, or that there is no internet connection.



Attaching a SnackBar to a CoordinatorLayout

```
<android.support.design.widget.CoordinatorLayout
    android:id="@+id/myCoordinatorLayout"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- Here are the existing layout elements, now wrapped in
                a CoordinatorLayout -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <!--...Toolbar, other layouts, other elements...-->
    </LinearLayout>

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

https://developer.android.com/training/snackbar/showing.html

Showing an “email archived” message to the user

```
Snackbar.make(findViewById(R.id.myCoordinatorLayout),
                          R.string.email_archived,
                          Snackbar.LENGTH_SHORT)
        .show();
```

---

https://developer.android.com/training/snackbar/showing.html

Adding an action to a message

```
public class MyUndoListener implements View.OnClickListener{

    @Override
    public void onClick(View v) {
        // Code to undo the user's last action
    }

}
```

```
Snackbar mySnackbar =
            Snackbar.make(findViewById(R.id.myCoordinatorLayout),
                                R.string.email_archived,
                                Snackbar.LENGTH_SHORT);
mySnackbar.setAction(R.string.undo_string, new MyUndoListener());
mySnackbar.show();
```

---

https://developer.android.com/training/snackbar/action.html

Concluding remarks

- Material Design attempts to standardise user interface elements across apps and across platforms.
- There are many aspects which we have not covered here: fonts, text layout, animations, transitions, and others.

Links

- https://material.io/components/android/
- https://material.io/icons/
- https://material.io/guidelines/
- https://developer.android.com/training/best-ui.html