



# Andhra Pradesh State Skill Development Corporation



Andhra Pradesh State Skill Development Corporation



# ANDROID APPLICATION DEVELOPMENT

## MATERIAL DESIGN

## Material Design

Material Design is a visual design philosophy that Google created in 2014. The aim of Material Design is a unified user experience across platforms and device sizes. Material Design includes a set of guidelines for style, layout, motion, and other aspects of app design. The complete guidelines are available in the [Material Design spec](#).

Material Design is for desktop web applications as well as for mobile apps. This chapter focuses only on Material Design for mobile apps on Android.

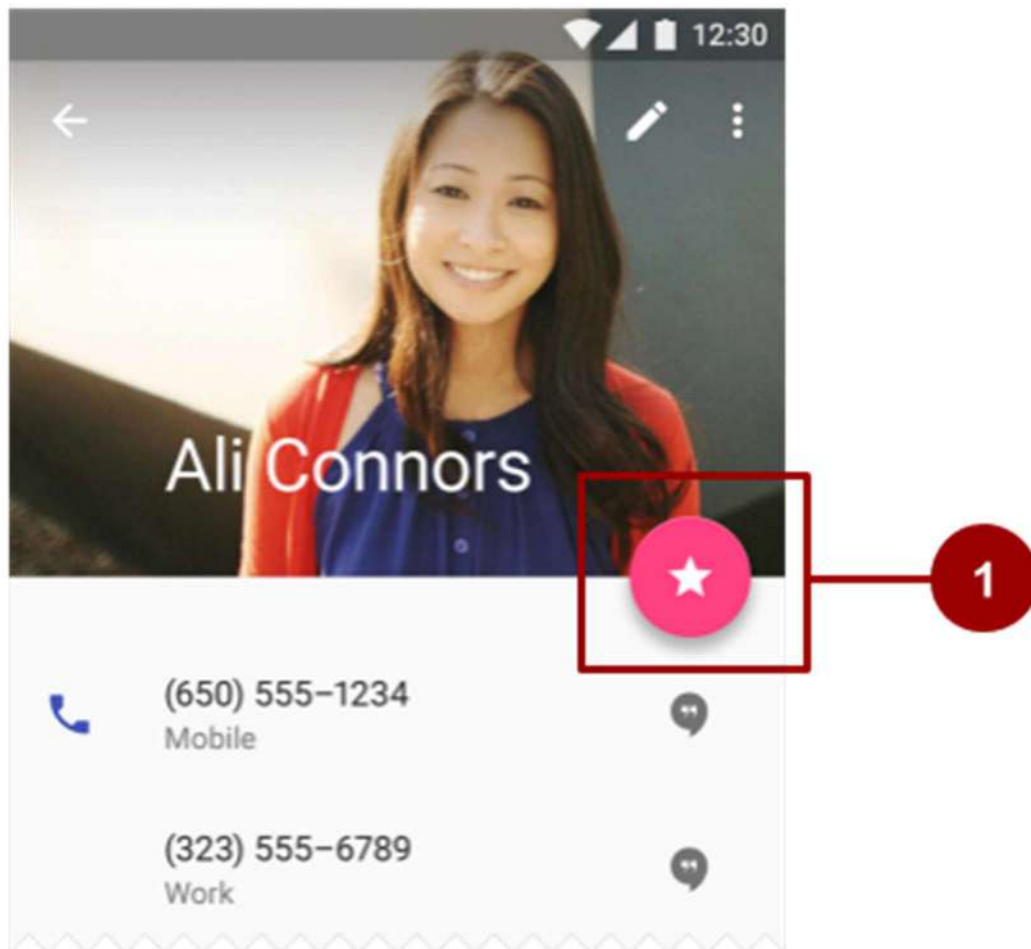
### Principles of Material Design

In Material Design, elements in your Android app behave like real world materials: they cast shadows, occupy space, and interact with each other.

#### **Bold, graphic, intentional**

Material Design involves deliberate color choices, edge-to-edge imagery, large-scale typography, and intentional white space that create a bold and graphic interface.

Emphasize user actions in your app so that the user knows right away what to do, and how to do it. For example, highlight things that users can interact with, such as buttons, EditText fields, and switches.



In the above figure, #1 is a [FloatingActionButton](#) with a pink accent color.

## Meaningful motion

Make animations and other motions in your app meaningful, so they don't happen at random. Use motions to reinforce the idea that the user is the app's primary mover. For example, design your app so that most motions are initiated by the user's actions, not by events outside the user's control. You can also use motion to focus the user's attention, give the user subtle feedback, or highlight an element of your app.

When your app presents an object to the user, make sure the motion doesn't break the continuity of the user's experience. For example, the user shouldn't have to wait for an animation or transition to complete.

The Motion section in this chapter goes into more detail about how to use motion in your app.

## Colors

Material Design principles include the use of bold color.

### Material Design color palette

The [Material Design color palette](#) contains colors to choose from, each with a primary color and shades labeled from 50 to 900:

- Choose a color labeled "500" as the primary color for your brand. Use that color and shades of that color in your app.
- Choose a contrasting color as your accent color and use it to create highlights in your app. Select any color that starts with "A".

When you create an Android project in Android Studio, a sample Material Design color scheme is selected for you and applied to your theme. In *colors.xml* in the *values* folder, three elements are defined, *colorPrimary*, *colorPrimaryDark*, and *colorAccent*:

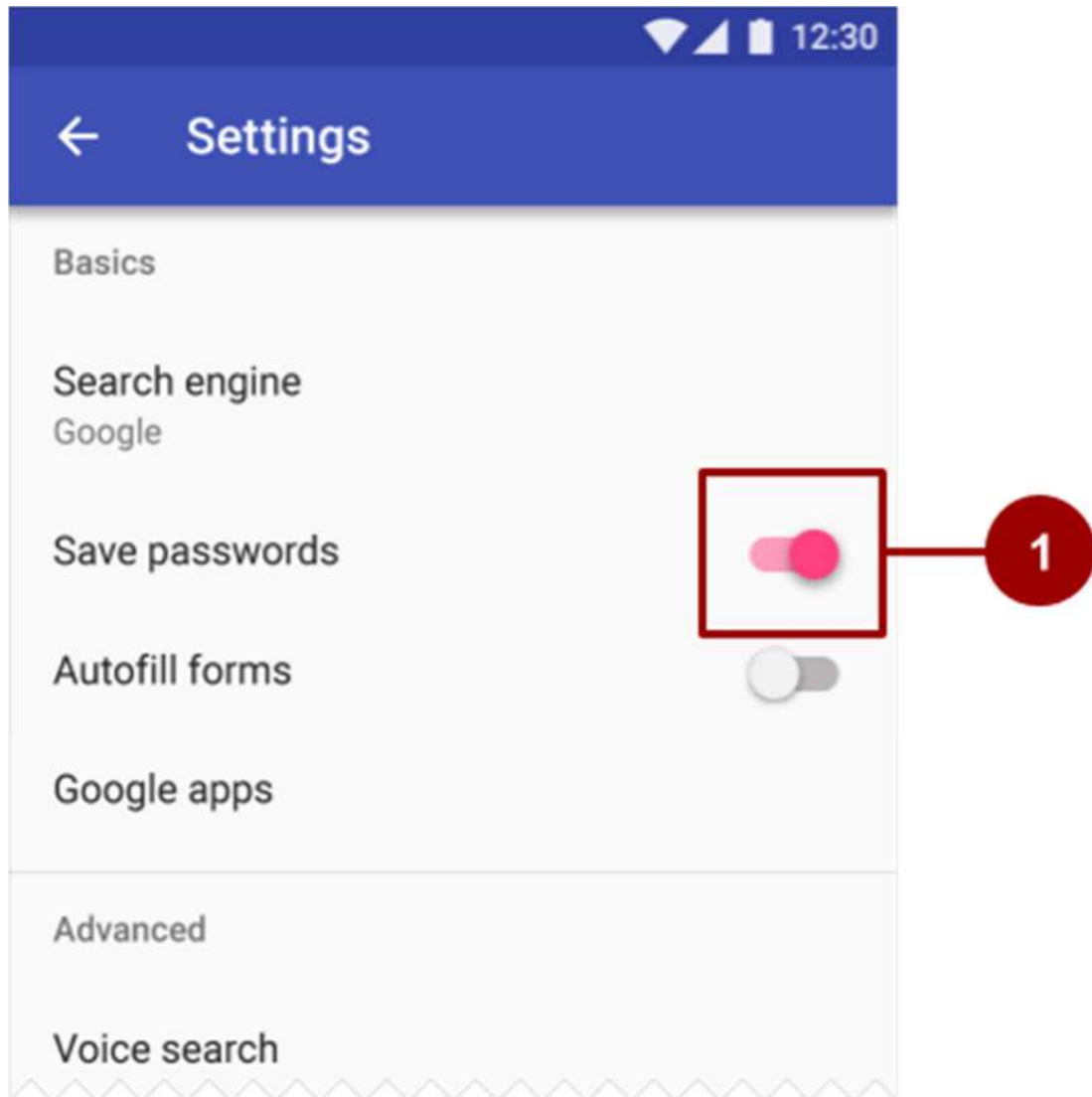
```
<resources>
  <color name="colorPrimary">#3F51B5</color>
  <!-- Indigo. -->
  <color name="colorPrimaryDark">#303F9F</color>
  <!-- A darker shade of indigo. -->
  <color name="colorAccent">#FF4081</color>
  <!-- A shade of pink. -->
</resources>
```

In *styles.xml* in the *values* folder, the three defined colors are applied to the default theme, which applies the colors to some app elements by default:

- *colorPrimary* is used by several *View* elements by default. For example, in the *AppTheme* theme, *colorPrimary* is used as the background color for the action bar. Change this value to the "500" color that you select as your brand's primary color.

- **colorPrimaryDark** is used in areas that need to slightly contrast with your primary color, for example the status bar above the app bar. Set this value to a slightly darker version of your primary color.
- **colorAccent** is used as the highlight color for several *View* elements. It's also used for switches in the "on" position, [FloatingActionButton](#), and more.

In the screenshot below, the background of the action bar uses colorPrimary (indigo), the status bar uses colorPrimaryDark (a darker shade of indigo), and the switch in the "on" position (#1 in the figure below) uses colorAccent (a shade of pink).



In summary, here's how to use the Material Design color palette in your Android app:

1. Pick a primary color for your app from [Material Design color palette](#) and copy its hex value into the *colorPrimary* item in *colors.xml*.
2. Pick a darker shade of this color and copy its hex value into the *colorPrimaryDark* item.
3. Pick an accent color from the shades starting with an "A" and copy its hex value into the *colorAccent* item.





- If you need more colors, create additional elements in the *colors.xml* file. For example, you could pick a lighter version of indigo and create an additional element named *colorPrimaryLight*. (The name is up to you.)

```
<color name="colorPrimaryLight">#9FA8DA</color>
<!-- A lighter shade of indigo. -->
```

To use this color, reference it as `_@color/colorPrimaryLight_`.

Changing the values in *colors.xml* automatically changes the colors of the *View* elements in your app, because the colors are applied to the theme in *styles.xml*.

## Contrast

Make sure all the text in your app's UI contrasts with its background. Where you have a dark background, make the text on top of it a light color, and vice versa. This kind of contrast is important for readability and [accessibility](#), because not all people see colors the same way.

If you use a platform theme such as [Theme.AppCompat](#), contrast between text and its background is handled for you. For example:

- If your theme inherits from *Theme.AppCompat*, the system assumes you are using a dark background. Therefore all of the text is near white by default.
- If your theme inherits from *Theme.AppCompat.Light*, the text is near black, because the theme has a light background.
- If you use the *Theme.AppCompat.Light.DarkActionBar* theme, the text in the action bar is near white, to contrast with the action bar's dark background. The rest of the text in the app is near black, to contrast with the light background.

Use color contrast to create visual separation among the elements in your app. Use your *colorAccent* color to call attention to key UI elements such as *FloatingActionButton* and switches in the "on" position.

## Opacity

Your app can display text with different degrees of opacity to convey the relative importance of information. For example, text that's less important might be nearly transparent (low opacity).

Set the *android:textColor* attribute using any of these formats: *"#rgb"*, *"#rrggbb"*, *"#argb"*, or *"#aarrggbb"*. To set the opacity of text, use the *"#argb"* or *"#aarrggbb"* format and include a value for the alpha channel. The alpha channel is the *a* or the *aa* at the start of the *textColor* value.

The maximum opacity value, *FF* in hex, makes the color completely opaque. The minimum value, *00* in hex, makes the color complete transparent.

To determine what hex number to use in the alpha channel:

- Decide what level of opacity you want to use, as a percentage. The level of opacity used for text depends on whether your background is dark or light. To find out what level of opacity to use in different situations, see the [Text color portion](#) of the Material Design guide.

2. Multiply that percentage, as a decimal value, by 255. For example, if you need primary text that's 87% opaque, multiply  $0.87 \times 255$ . The result is 221.85.
3. Round the result to the nearest whole number: 222.
4. Use a hex converter to convert the result to hex: DE. If the result is a single value, prefix it with 0.

In the following XML code, the background of the text is dark, and the color of the primary text is 87% white (*deffffff*). The first two numbers of the color code (de) indicate the opacity.

## <TextView

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Hello World!"
android:textSize="45dp"
android:background="@color/colorPrimaryDark"
android:textColor="#deffffff"/>
```

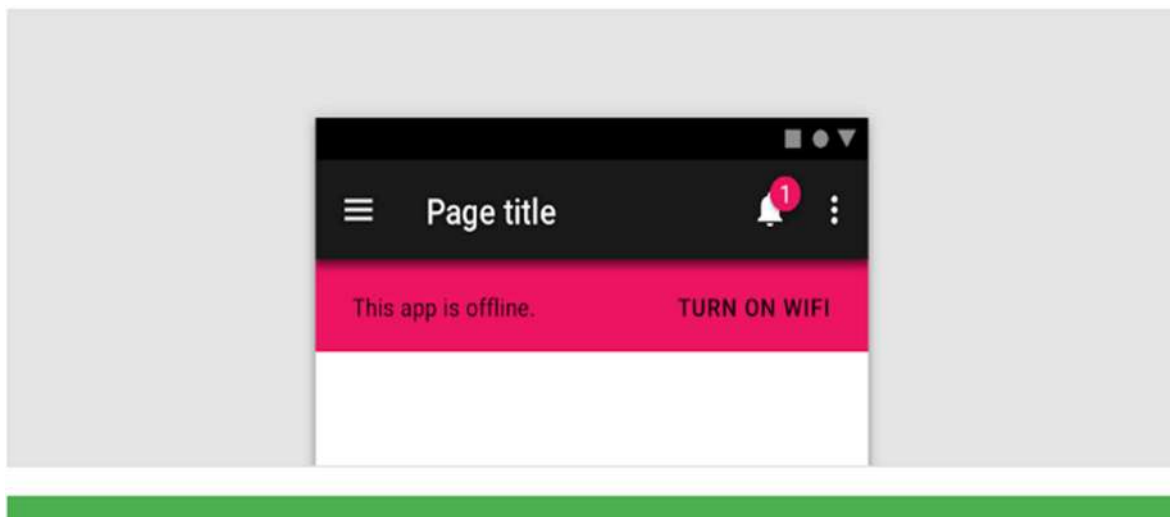
Instead of using gray text and icons on top of colored backgrounds, create better contrast by displaying white or black text with reduced opacity.

For example, black text displayed at 75% opacity on a green background gives the text an appearance of black, with a hint of green.

Alternatively, you can calculate the color of text by doing the following:

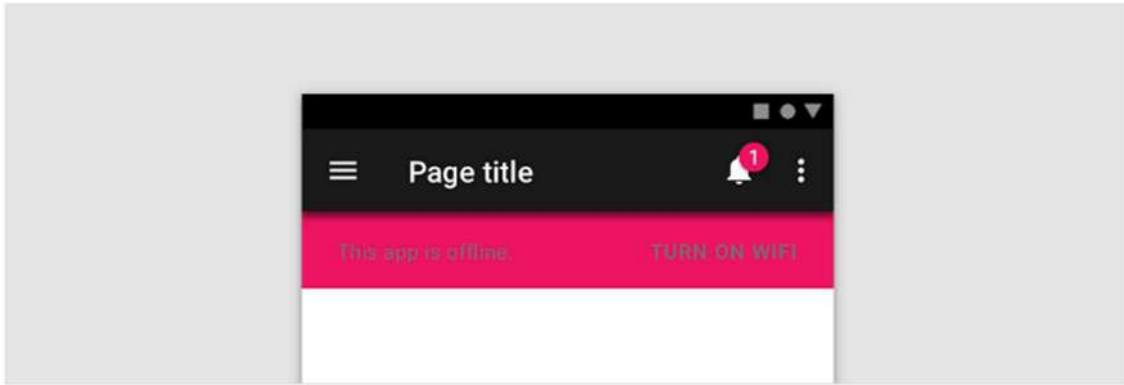
- Place the color black at reduced opacity in front of a green background
- Identify the hex value of the resulting darkened green color
- Use that hex value of that color for your text

In this case, if the surface behind the text changes color, you must update the hex color as well.



Do

Use a transparent version of black on a colored surface to preserve legibility.



Don't

Avoid using opaque gray text that isn't legible on colored surfaces.

## Typography

The Android design language relies on traditional typographic tools such as scale, space, rhythm, and alignment with an underlying grid. Successful deployment of these tools is essential to help users quickly understand a screen of information. To support such use of typography, Android supplies a type family named **Roboto**, created specifically for the requirements of UI and high-resolution screens.

With Android 8.0 (API level 26), you can also choose to provide a font as a resource in XML that is bundled with the app package (APK), or download a font from a font provider app. These features are available on devices running Android API versions 14 and higher through the Support Library 26.

## Font styles

The Android platform provides predefined font styles and sizes that you can use in your app. These styles and sizes were developed to balance content density and reading comfort under typical conditions. Type sizes are specified with sp (scaleable pixels) to enable large type modes for **accessibility**.

Be careful not to use too many different type sizes and styles together in your layout.



Display 4

Light 112sp

Display 3

Regular 56sp

Display 2

Regular 45sp

Display 1

Regular 34sp

Headline

Regular 24sp

Title

Medium 20sp

Subheading

Regular 16sp (Device), Regular 15sp (Desktop)

Body 2

Medium 14sp (Device), Medium 13sp (Desktop)

Body 1

Regular 14sp (Device), Regular 13sp (Desktop)

Caption

Regular 12sp

Button

MEDIUM (ALL CAPS) 14sp

To use one of these predefined styles in a View, set the `android:textAppearance` attribute. This attribute defines the default appearance of the text: its color, typeface, size, and style. Use the backward-compatible [TextAppearance.AppCompat](#) style.

For example, to make a TextView appear in the Display 3 style, add the following attribute



to the TextView in XML:

```
android:textAppearance="@style/TextAppearance.AppCompat.Display3"
```

For more information on styling text, view the [Typography](#) Material Design guidelines.

## Fonts as resources

Android 8.0 (API level 26) introduces Fonts in XML, which lets you bundle fonts as resources in your app package (APK). You can create a font folder within the res folder as a resource directory using Android Studio, and then add a font XML file to the font folder. The fonts are compiled in your R file and are automatically available in Android Studio. To access a font resource, use *@font/myfont*, or *R.font.myfont*.

To use the Fonts in XML feature, the device that runs your app must run Android 8.0 (API level 26). To use the feature on devices running Android 4.1 (API level 16) and higher, use the Support Library 26. For more information on using the support library, refer to the [Using the support library section](#).

To learn how to add fonts as XML resources, see [Fonts in XML](#).

## Downloadable fonts

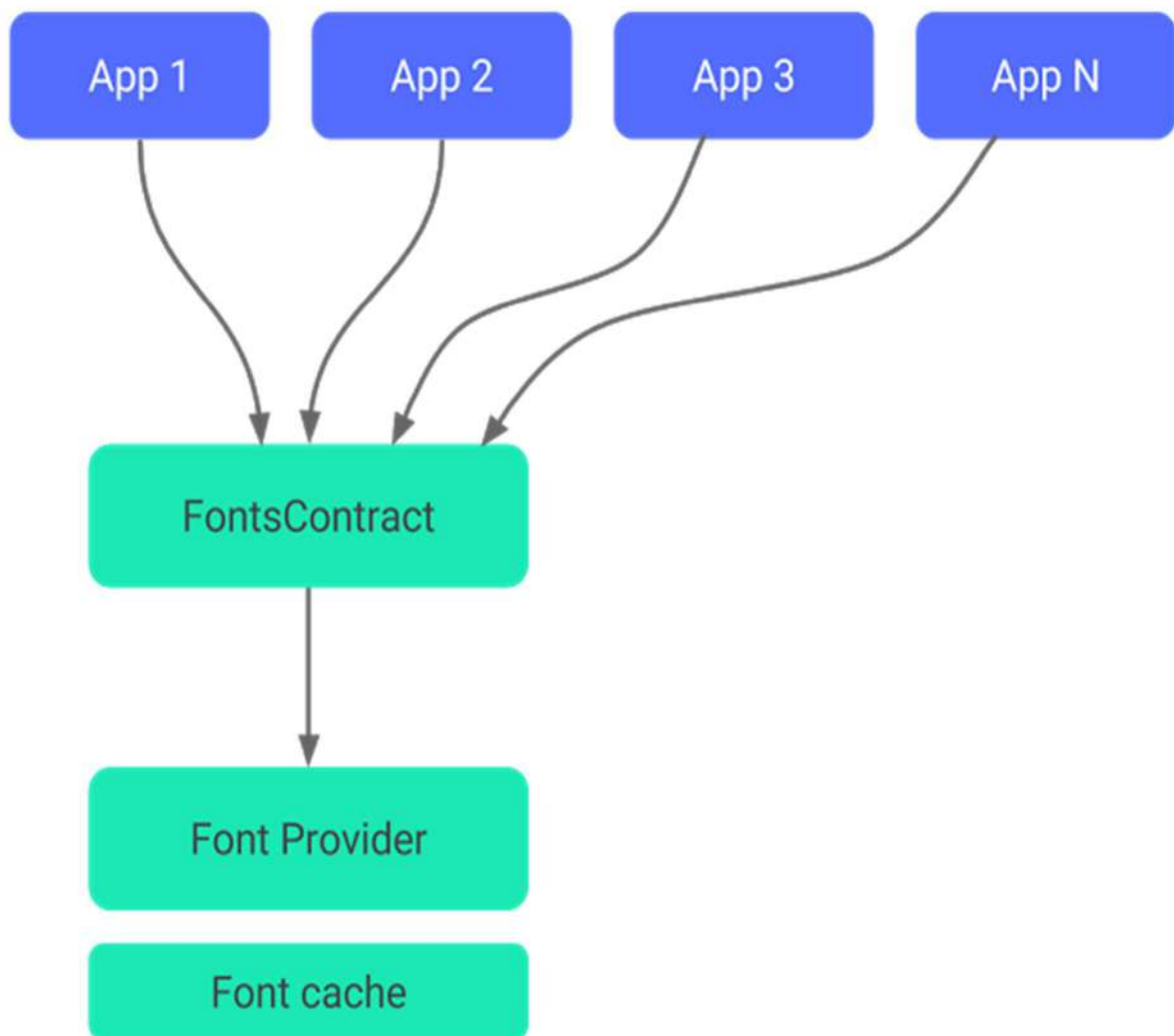
An alternative to bundling fonts with the app package (APK) is to download the fonts from a provider app. Android 8.0 (API level 26) enables APIs to request fonts from a provider app, and the feature is available on devices running Android API versions 14 and higher through the Support Library 26. A font provider app is an app that retrieves fonts and caches them locally so that other apps can request and share fonts.

Downloadable fonts offer the following benefits:

Reduces the APK size. Increases the app installation success rate. Improves the overall system health, as multiple APKs can share the same font through a provider. This saves users cellular data, phone memory, and disk space. In this model, the font is fetched over the network when needed. You can set your app to download fonts by using the layout editor in Android Studio 3.0. For detailed instructions, see [Downloadable Fonts](#).

## How does Downloadable Fonts work?

A font provider is an application that retrieves fonts and caches them locally so other apps can request and share fonts. Figure 1 illustrates the process.



## Layout

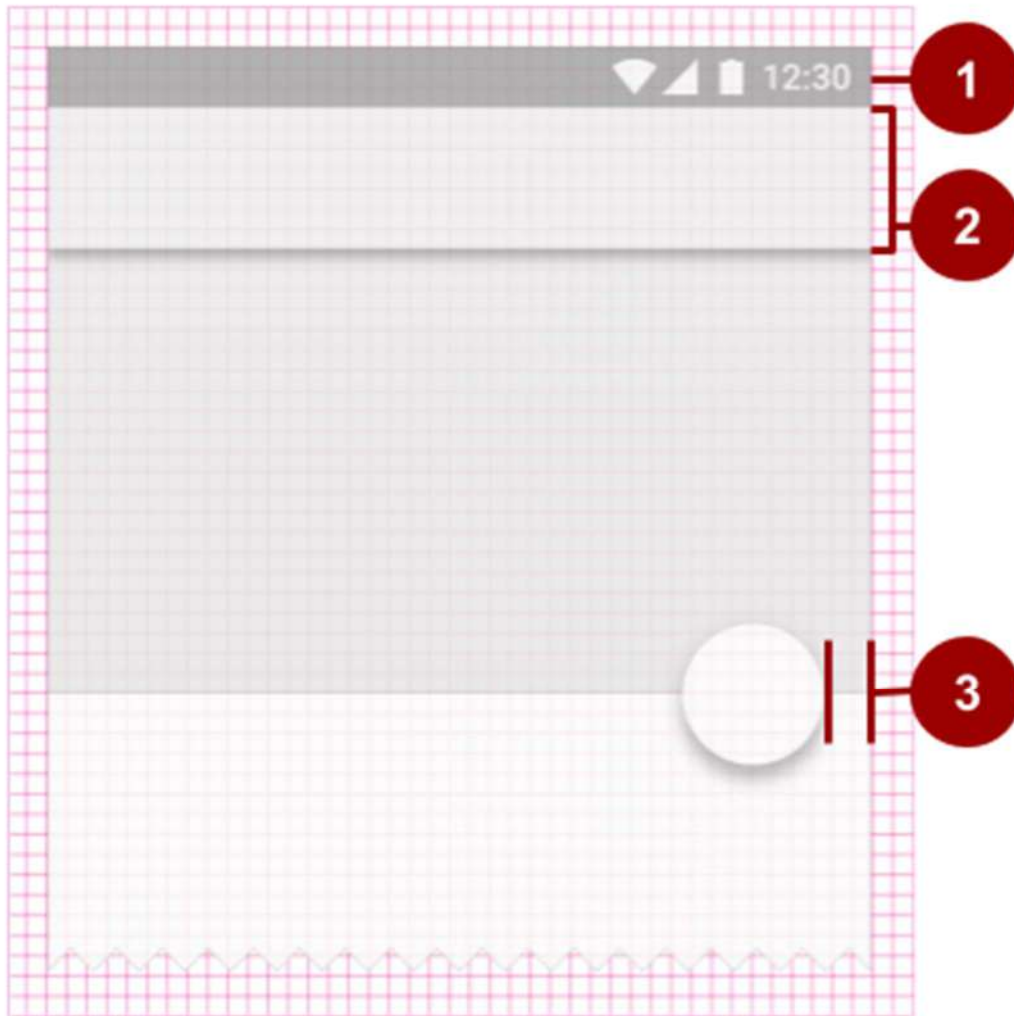
You specify View elements for your app's UI in layout resource files. Layout resources are written in XML and listed within the layout folder in the res folder in the Project > Android pane. The following guide explains some of the best practices for designing a layout.

## Metrics

Components in the Material Design templates that are meant for mobile, tablet, and desktop devices align to an 8dp square grid. A dp is a **density-independent pixel**, an abstract unit based on screen density. A dp is similar to an sp, but sp is also scaled by the user's font size preference. That's why sp is preferred for accessibility.

The 8dp square grid guides the placement of elements in your layout. Every square in the

grid is 8dp x 8dp, so the height and width of every element in the layout is a multiple of 8dp.



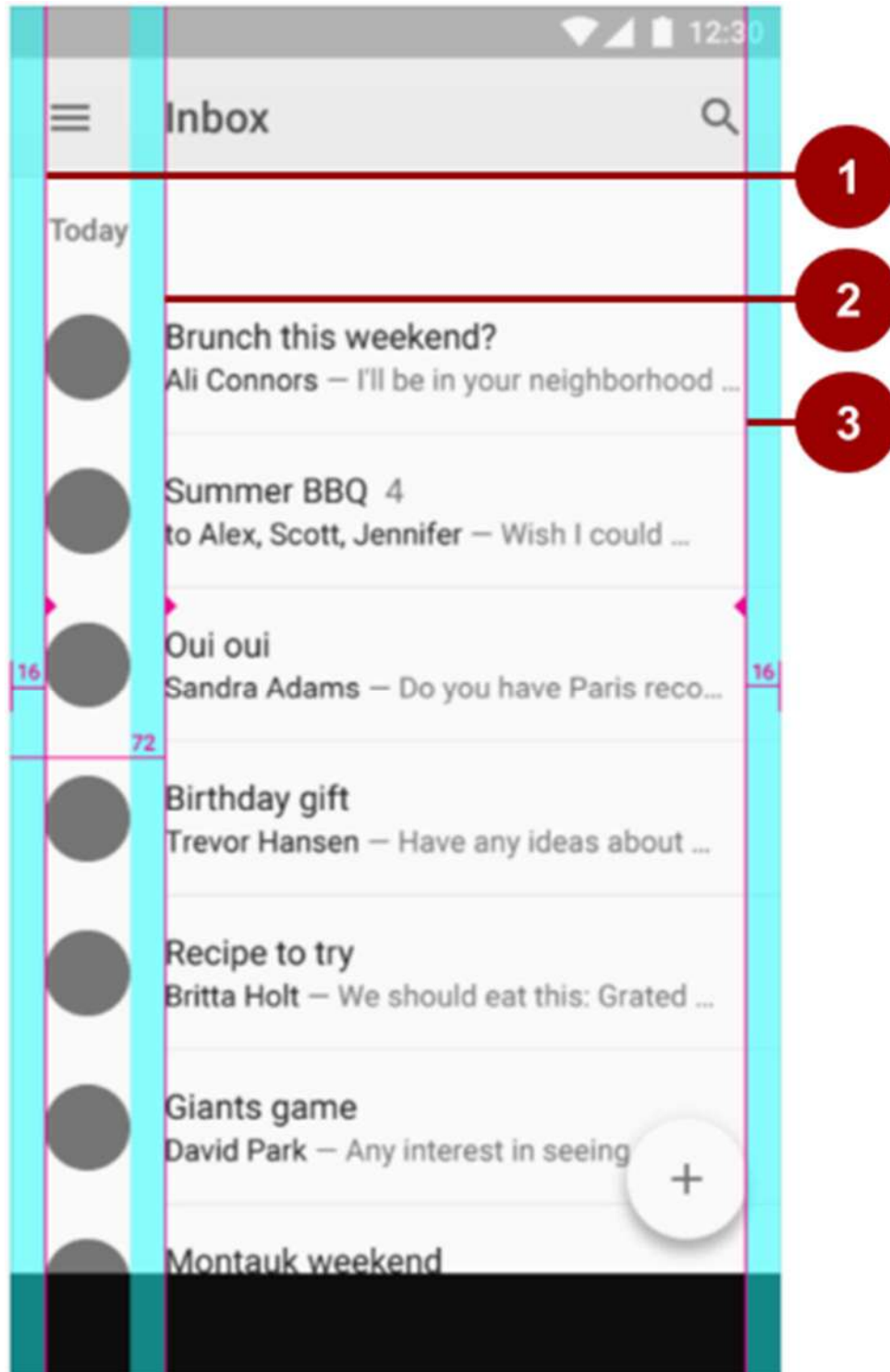
In the above figure:

1. The status bar in this layout is 24dp tall, the height of three grid squares.
2. The toolbar is 56dp tall, the height of seven grid squares.
3. One of the right-hand content margins is 16dp from the edge of the screen, the width of two grid squares.

Iconography in toolbars aligns to a 4dp square grid instead of an 8dp square grid, so the dimensions of icons in the toolbar are multiples of 4dp.

### Keylines

Keylines are outlines in a layout grid that determine the placement of text and icons. For example, keylines mark the edges of the margins in a layout.



In the above figure:

1. Keyline showing the left margin for the screen edge, which in this case is 16dp.
2. Keyline showing the left margin for content associated with an icon or avatar, 72dp.
3. Keyline showing the right margin for the screen edge, 16dp.

Material Design typography aligns to a 4dp baseline grid, which is a grid made up only of horizontal lines.



To learn more about metrics and keylines in Material Design, visit the [metrics and keylines guide](#).

## Components and patterns

Button elements and many other View elements used in Android conform by default to Material Design principles. The Material Design guide includes components and patterns that you can build on to help your users intuit how the elements in your UI work, even if users are new to your app.

Use Material Design components to guide the specs and behavior of buttons, chips, cards, and many other UI elements. Use Material Design [patterns](#) to guide how you format dates and times, gestures, the navigation drawer, and many other aspects of your UI.

This section teaches you about the Design Support Library and some of the components and patterns that are available to you. For complete documentation about all the components and patterns that you can use, see the [Material Design guide](#).

## Design Support Library

The [Design](#) package provides APIs to support adding Material Design components and patterns to your apps. The [Design Support Library](#) adds support for various Material Design components and patterns for you to build on. To use the library, include the following dependency in your build.gradle (Module: app) file:

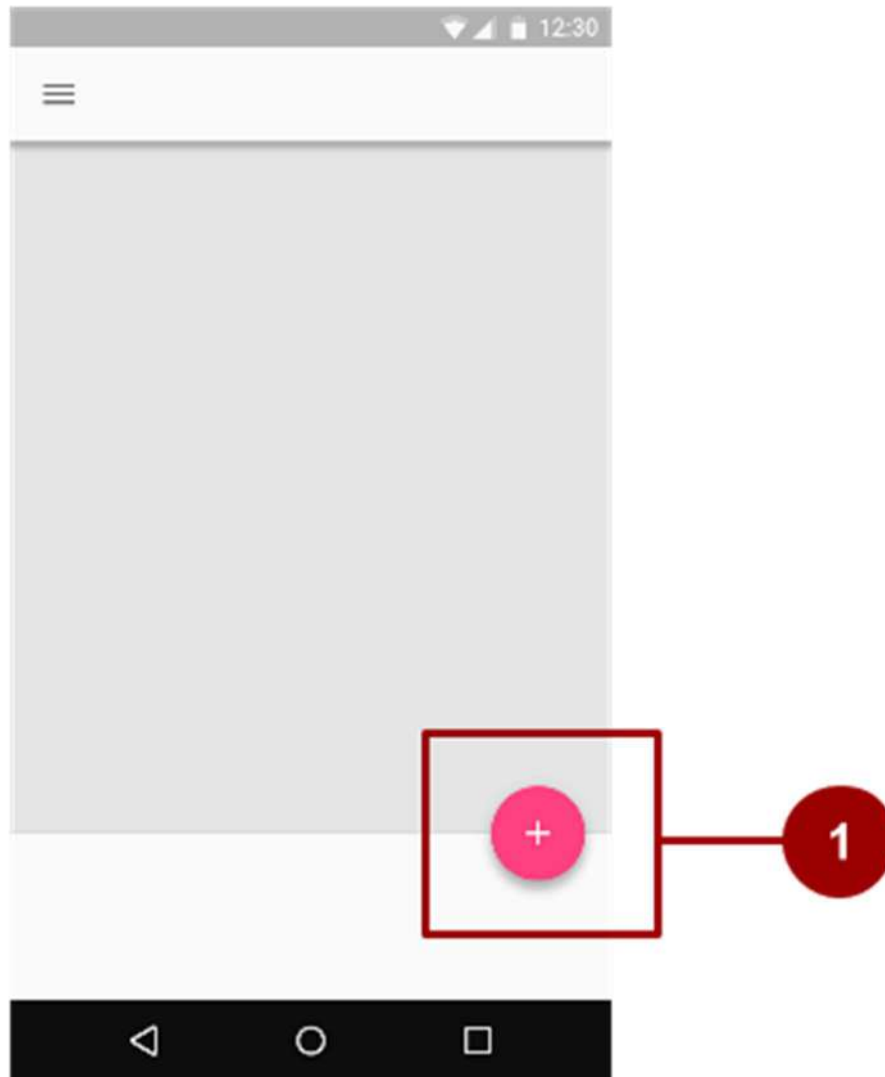
```
implementation 'com.google.android.material:material:1.1.0'
```

To make sure you have the most recent version number for the Design Support Library, check the Support Library page.

## Floating action buttons (FABs)

Use a floating action button (FAB) for actions you want to encourage users to take. A FAB is a circled icon that floats "above" the UI. On focus it changes color slightly, and it appears to lift up when selected. When tapped, it can contain related actions.





In this figure:

A normal-sized FAB To implement a FAB, use [FloatingActionButton](#) and set the FAB's attributes in your layout XML. For example:

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="end|bottom"
    android:src="@drawable/ic_my_icon"
    android:contentDescription="@string/submit"
    android:layout_margin="16dp"
    app:fabSize="normal"
    app:elevation="10dp"/>
```

The [fabSize](#) attribute sets the FAB's size. It can be *"normal"* (56dp), *"mini"* (40dp), or *"auto"*, which changes based on the window size.

The FAB's elevation is the distance between its surface and the depth of its shadow. You

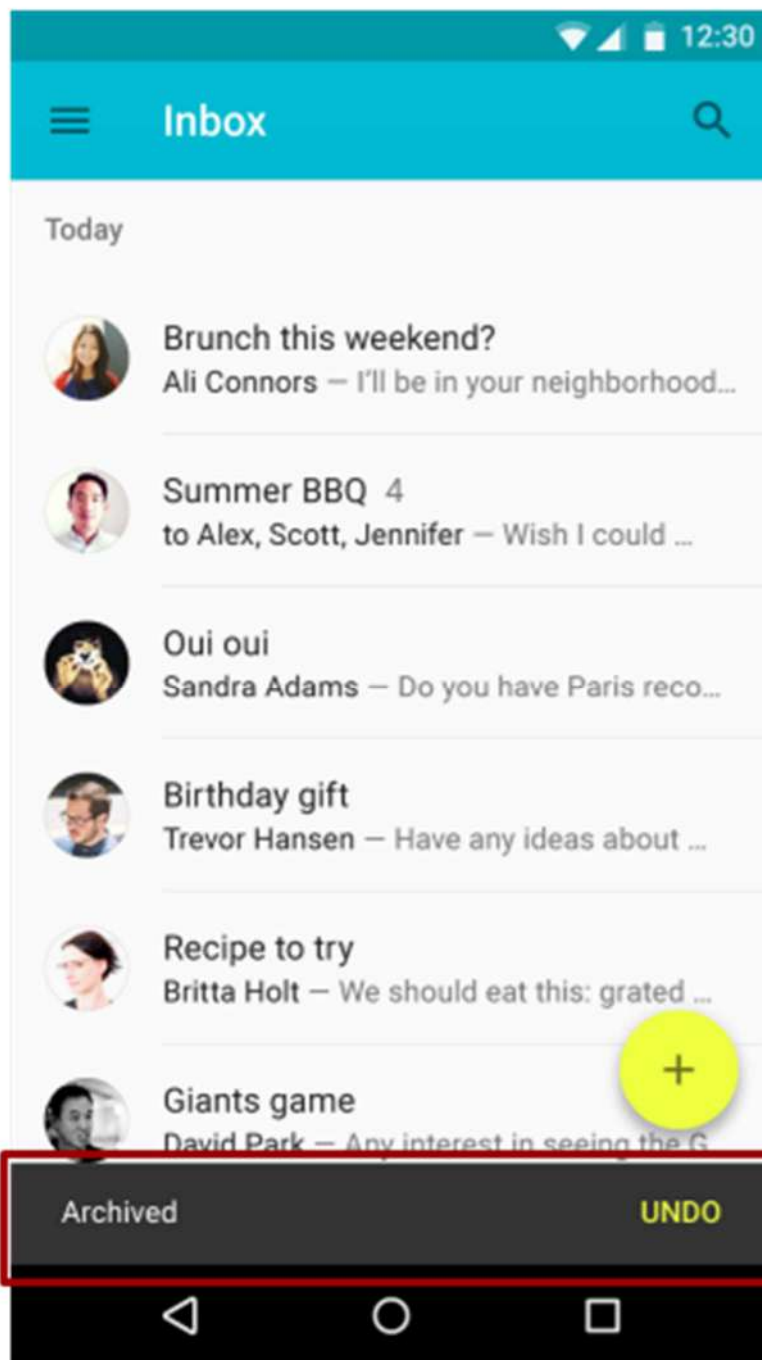


can set the elevation attribute as a reference to another resource, a string, a boolean, or several other ways.

To learn about all the attributes you can set for a FAB including clickable, rippleColor, and backgroundTint, see FloatingActionButton. To make sure you're using FABs as intended, check the extensive [FAB usage information in the Material Design guide](#).

## Snackbars

A snackbar provides brief feedback about an operation through a message in a horizontal bar on the screen. It contains a single line of text directly related to the operation performed. A snackbar (shown as #1 in the figure below) can contain a text action, but no icons.



Snackbars automatically disappear after a timeout or after a user interaction elsewhere on the screen. You can associate a snackbar with any kind of view (any object derived from the View class). However, if you associate the snackbar with a [CoordinatorLayout](#), the snackbar gains additional features:

- The user can dismiss the snackbar by swiping it away.
- The layout moves some other UI elements when the snackbar appears. For example, if the layout has a FAB, the layout moves the FAB up when it shows the snackbar, instead of drawing the snackbar on top of the FAB.

To create a *Snackbar* object, use the `Snackbar.make()` method. Specify the ID of the `CoordinatorLayout` to use for the snackbar, the message that the snackbar displays, and the length of time to show the message. For example, this statement creates the snackbar and calls `show()` to show the snackbar to the user:

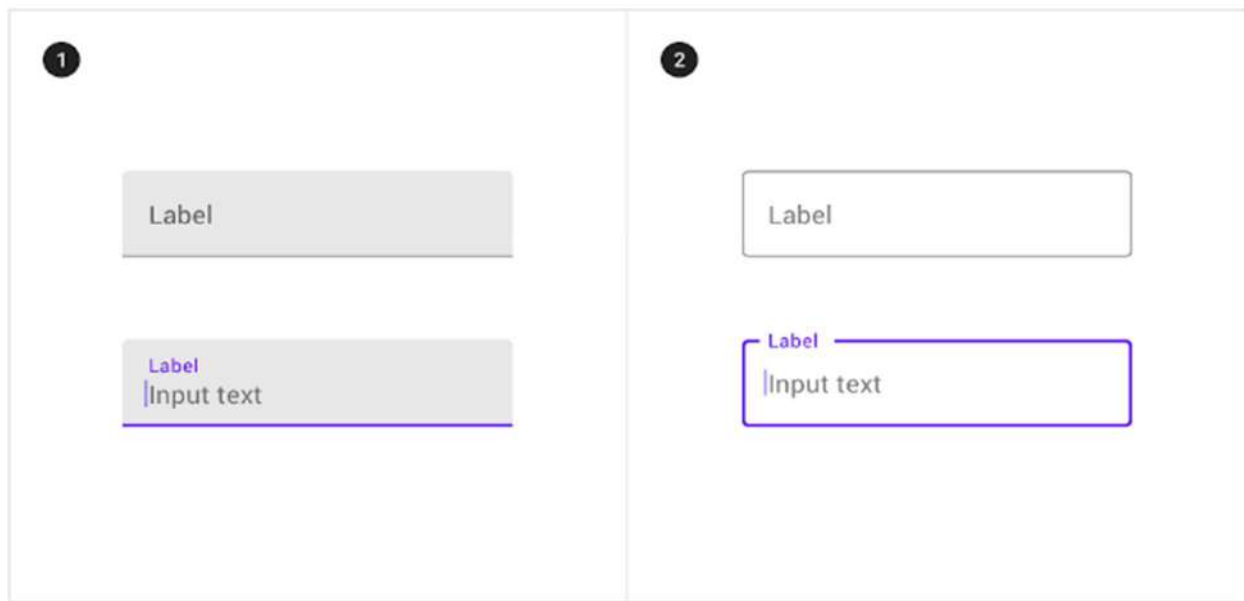
```
Snackbar.make(findViewById(R.id.myCoordinatorLayout),R.string.email_sent,Snackbar.LENGTH_SHORT).show;
```

For more information, see [Building and Displaying a Pop-Up Message](#) and the Snackbar reference. To make sure you're using snackbars as intended, see the [snackbar usage information in the Material Design guide](#).

**Tip:** A Toast is similar to a Snackbar, except that a Toast is usually used for a system message, and a Toast can't be swiped off the screen.

## Text fields

Text fields let users enter and edit text. Text fields allow users to enter text into a UI. They typically appear in forms and dialogs. ##### Types Text fields come in two types: - Filled text fields - Outlined text fields Both types of text fields use a container to provide a clear affordance for interaction, making the fields discoverable in layouts.

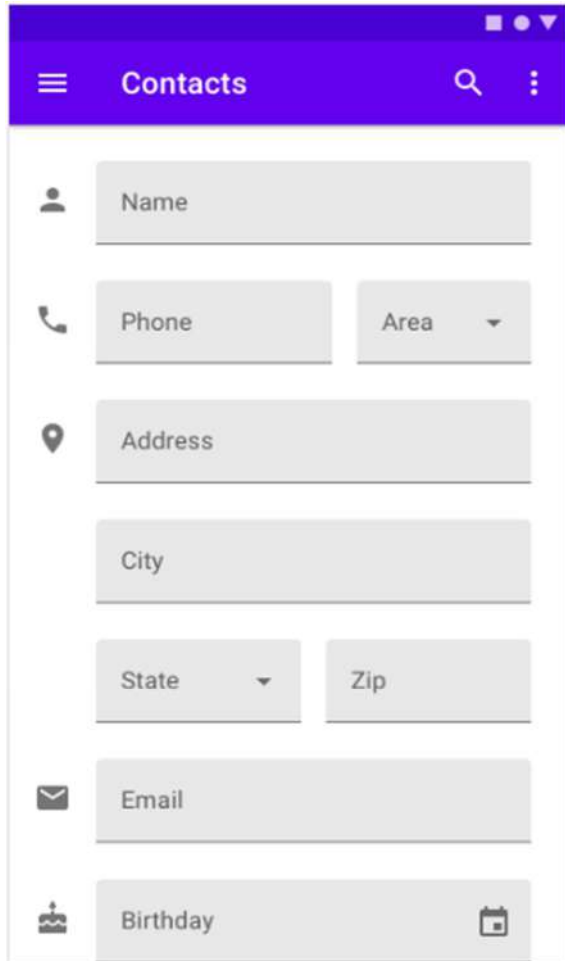


1. Filled text fields
2. Outlined text fields

## Choosing the right text field

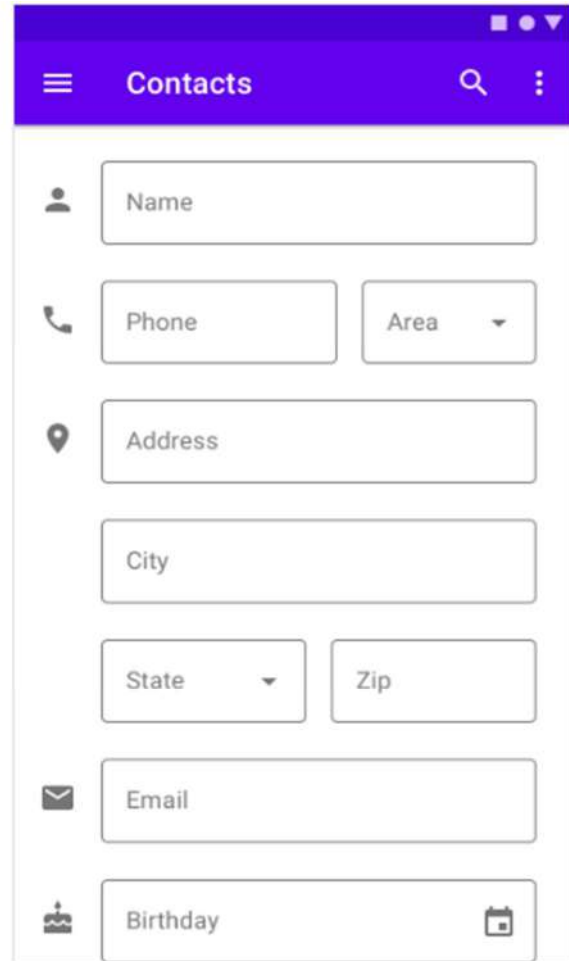
Both types of text fields provide the same functionality, so the type of text field you use can depend on style alone.

Choose the type that: - Works best with your app's visual style - Best accommodates the goals of your UI - Is most distinct from other components (like buttons) and surrounding content



A mobile application interface for a 'Contacts' app. The header is a solid purple bar with a white hamburger menu icon, the word 'Contacts' in white, a white magnifying glass icon, and a white three-dot menu icon. The form fields are solid grey with rounded corners. The fields are: 'Name' (with a person icon), 'Phone' (with a phone icon) and 'Area' (with a dropdown arrow), 'Address' (with a location pin icon), 'City', 'State' (with a dropdown arrow) and 'Zip', 'Email' (with an envelope icon), and 'Birthday' (with a birthday cake icon and a calendar icon).

Mobile form using filled text fields



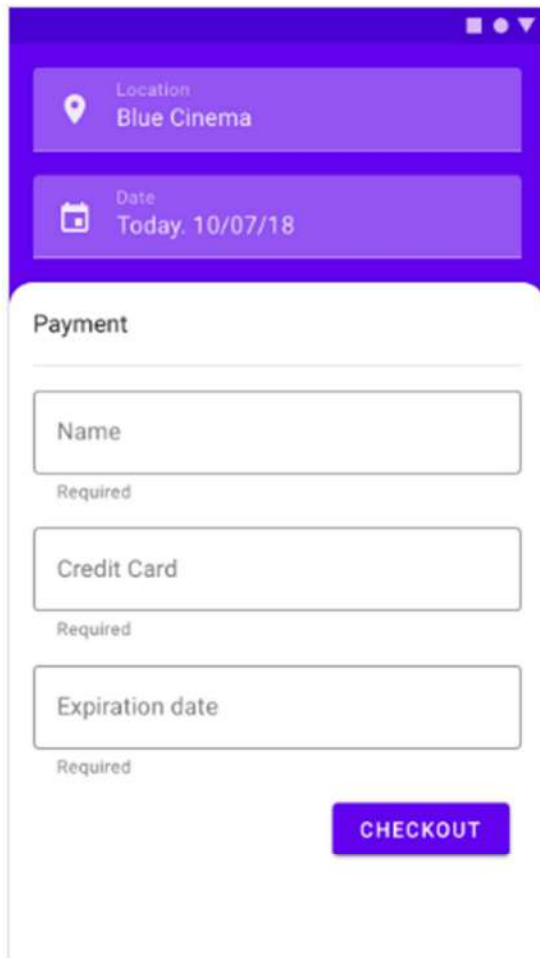
A mobile application interface for a 'Contacts' app, identical in layout and icons to the first one, but with outlined text fields. The header is a solid purple bar with a white hamburger menu icon, the word 'Contacts' in white, a white magnifying glass icon, and a white three-dot menu icon. The form fields are white with black outlines and rounded corners. The fields are: 'Name' (with a person icon), 'Phone' (with a phone icon) and 'Area' (with a dropdown arrow), 'Address' (with a location pin icon), 'City', 'State' (with a dropdown arrow) and 'Zip', 'Email' (with an envelope icon), and 'Birthday' (with a birthday cake icon and a calendar icon).

The same mobile form using outlined text fields

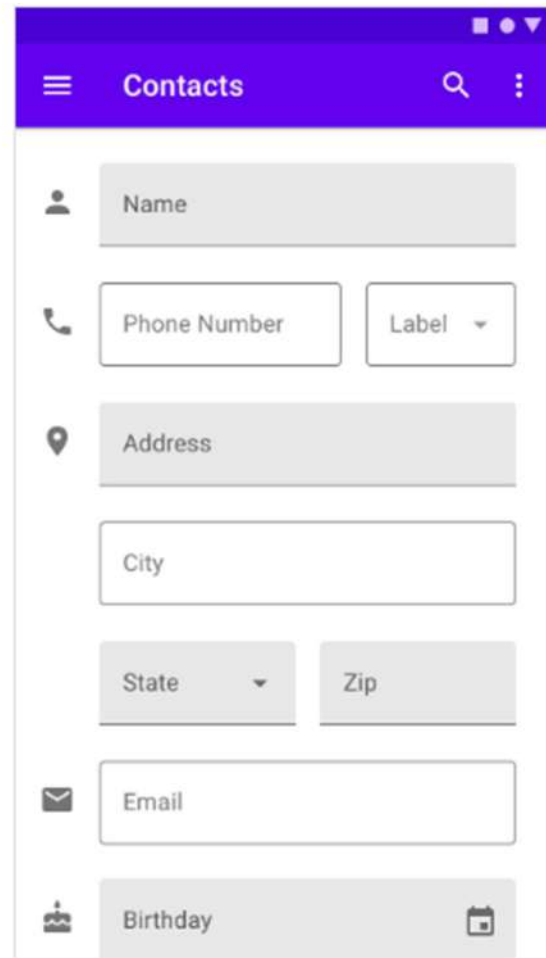
## Both types of text fields in one UI

If both types of text fields are used in a single UI, they should be used consistently within different sections, and not intermixed within the same region. For example, you could use outlined text fields in one section and filled text fields in another.





**Do**  
When using both types of text fields in a UI, separate them by region.



**Don't**  
When using a both types of text fields, don't use both next to each other, or within the same form.

for more info [CLICKHERE](#)

## Date pickers

Date pickers let users select a date, or a range of dates.

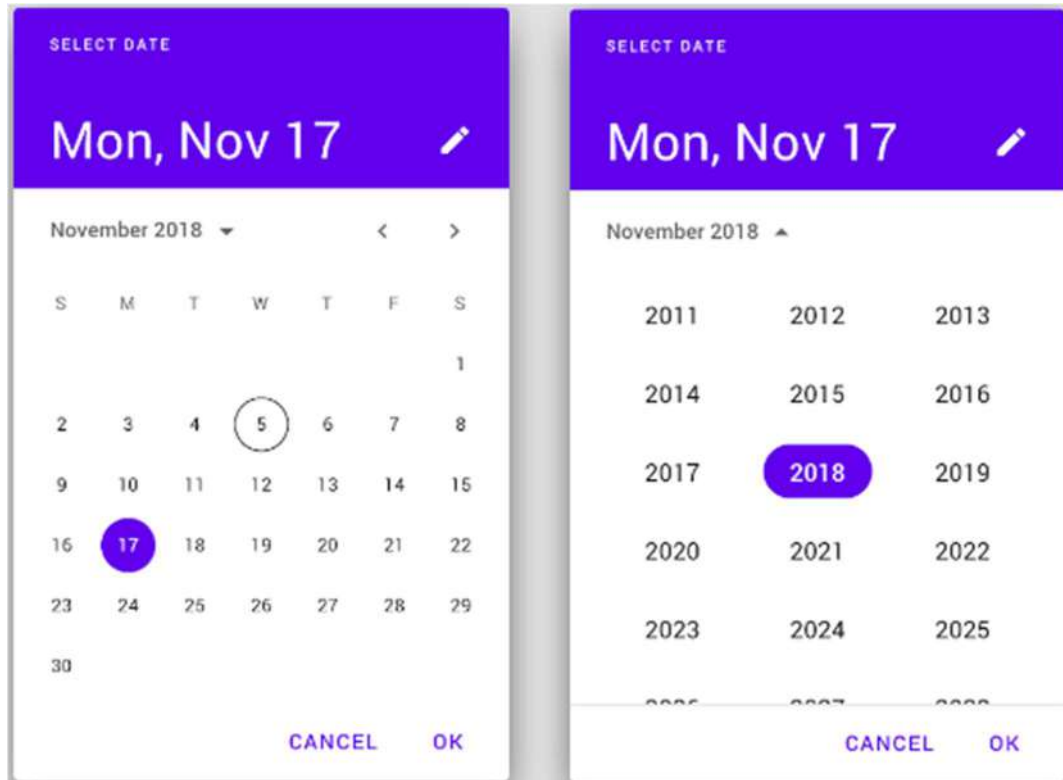
### Usage

Mobile calendar pickers can be used to select dates in the near future or past, when it's useful to see them in a calendar month format. They are displayed in a dialog. Common use cases include:

- Making a restaurant reservation
- Scheduling a meeting

They aren't ideal for selecting dates in the distant past or future that require more navigation, such as entering a birth date or expiration date.

Mobile calendar pickers allow selection of a single date and year.

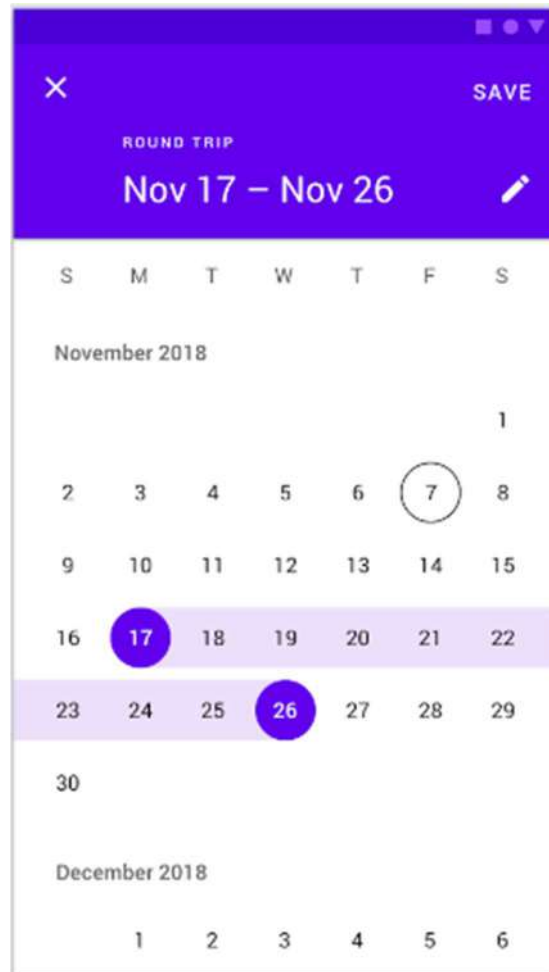


## Mobile date range pickers

### Usage

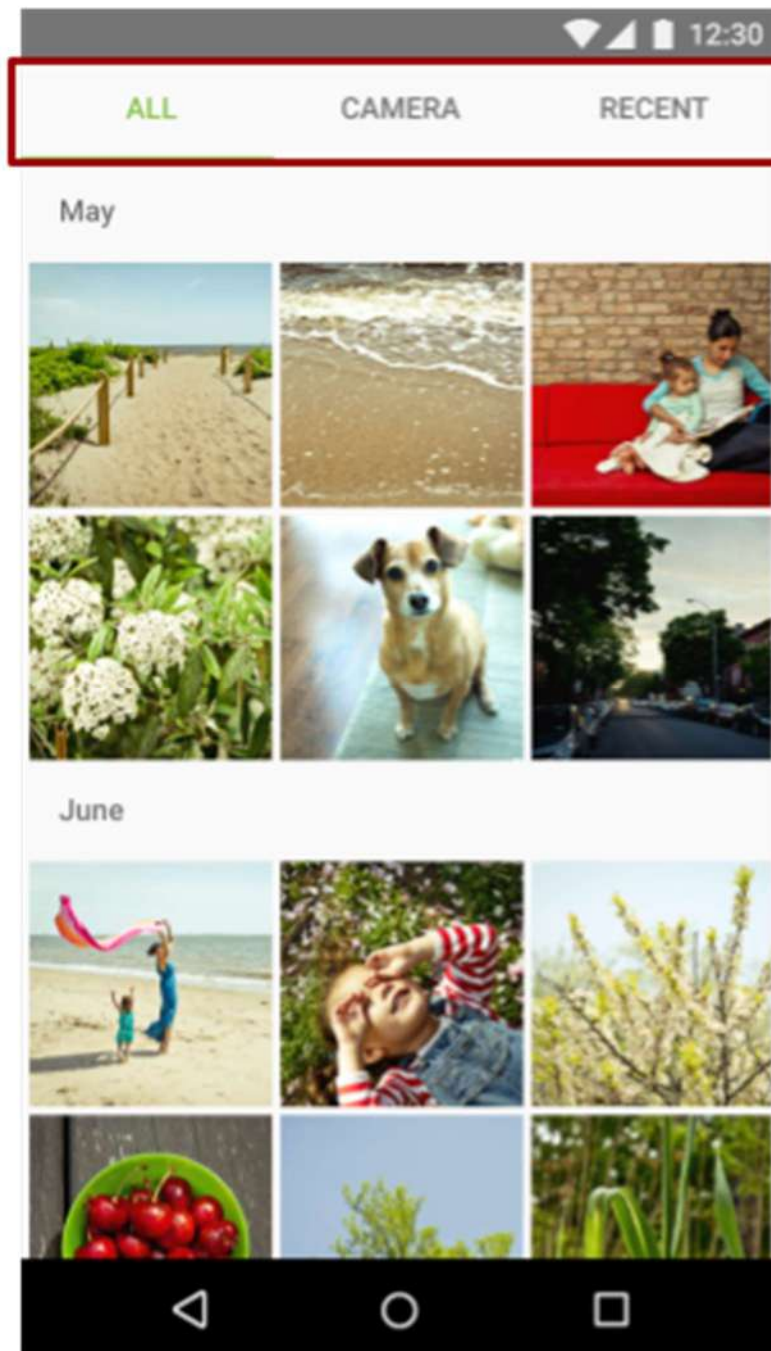
Mobile date range pickers allow selection of a range of dates. They cover the entire screen. Common use cases include:

- Booking a flight
- Reserving a hotel



## Tabs

Use tabs to organize content at a high level. For example, the user might use tabs to switch between View elements, data sets, or functional aspects of an app. Present tabs as a single row above their associated content. Make tab labels short and informative. For example, in the figure below, the app shows three tabs (marked by #1), with the **All** tab selected.



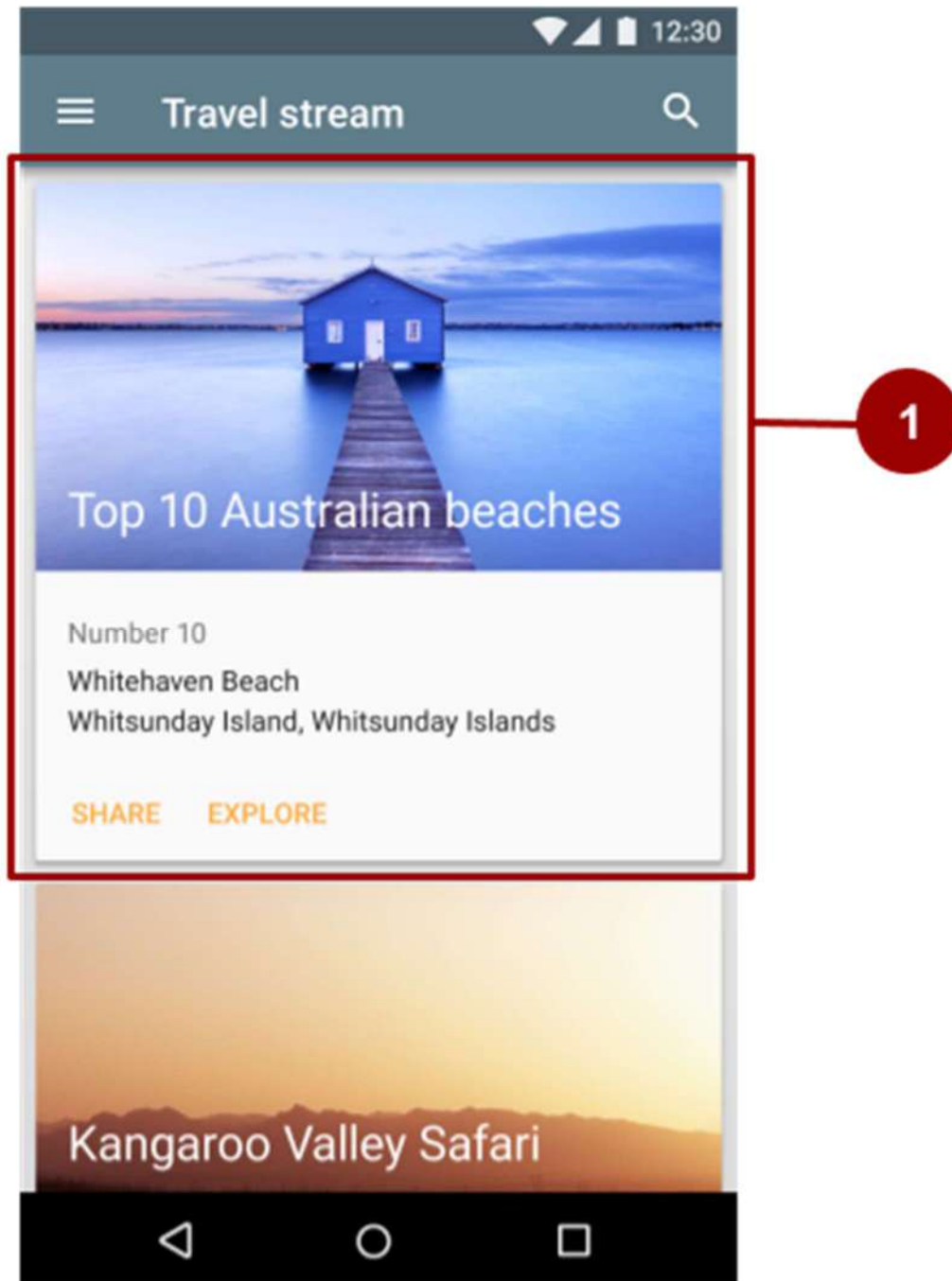
You can use tabs with swipe views in which users navigate between tabs with a horizontal finger gesture (horizontal paging). If your tabs use swipe views, don't pair the tabs with content that also supports swiping. For an example, see the lesson in this course on providing user navigation.

For information on implementing tabs, see [Creating Swipe Views with Tabs](#). To make sure you're using tabs as intended, see the extensive [tab usage information in the Material Design guide](#).

## Cards

A card is a sheet of material that serves as an entry point to more detailed information. Each card covers only one subject. A card may contain a photo, text, and a link. It can display content containing elements of varying size, such as photos with captions of variable length.

A card collection is a layout of cards on the same plane. The figure below shows one card in a card collection (marked by #1).



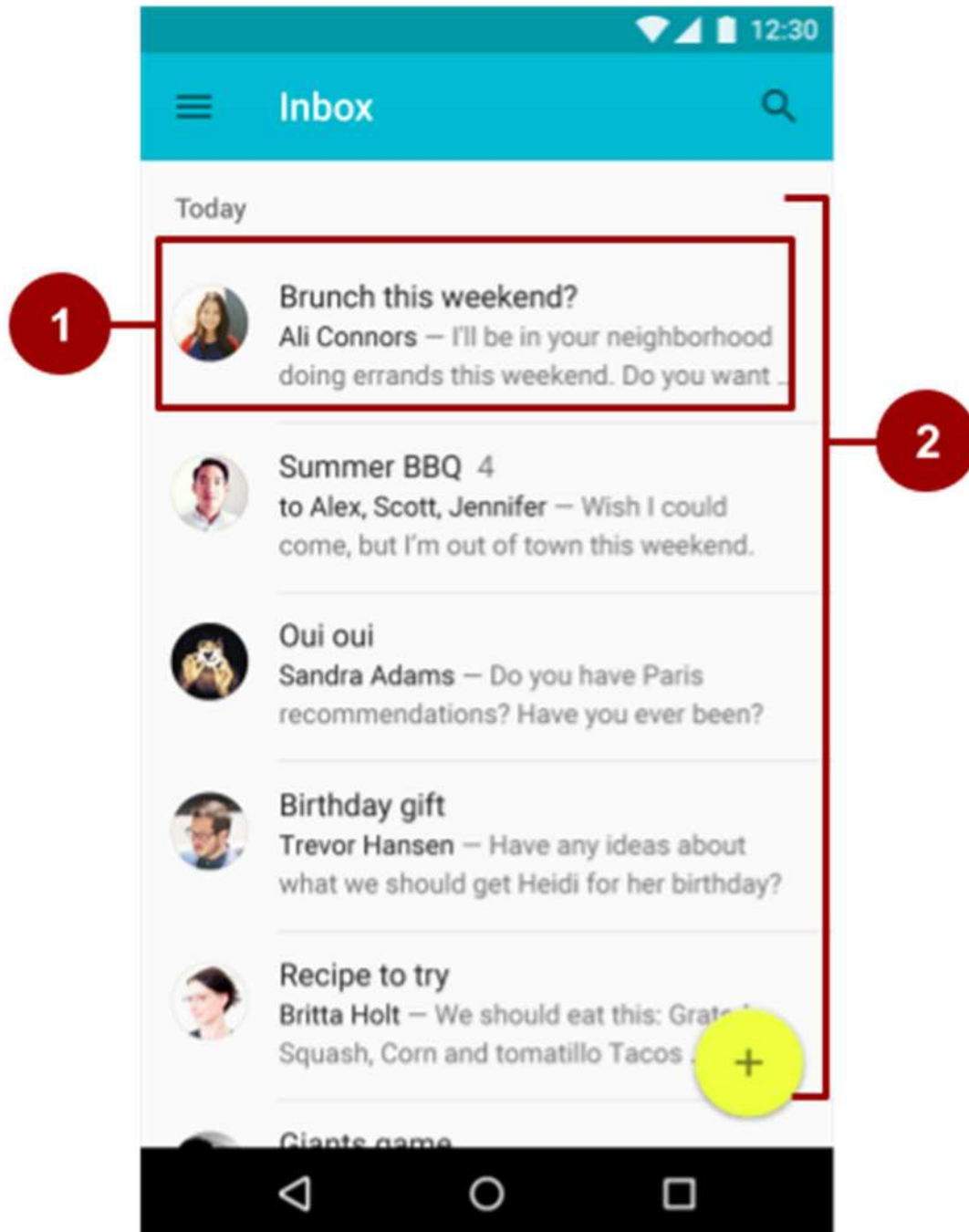
**CardView** is included as part of the v7 support library. To use the library, include the following dependency in your build.gradle (Module: app) file:



implementation 'androidx.cardview:cardview:1.0.0'

## Lists

A list is a single continuous column of rows of equal width. Each row functions as a container for a tile. Tiles hold content, and can vary in height within a list.



In the figure above:

1. A tile within the list
2. A list with rows of equal width, each containing a tile

To create a list, use the RecyclerView widget. include the following dependency in your build.gradle (Module: app) file:

implementation 'androidx.recyclerview:recyclerview:1.1.0'

For more information on creating lists in Android, see [Create a List with RecyclerView](#).

### Practicle Exmaple on MaterialDesign:

- Create new Android Studio Project
- Adding dependency

implementation 'com.google.android.material:material:1.1.0'

- Change Theme in style.xml

```
<style name="AppTheme" parent="Theme.MaterialComponents.Light">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/materialPrimary</item>
    <item name="colorPrimaryDark">@color/materialPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

- design xml file (activity\_main.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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:orientation="vertical"
    android:padding="10dp"
    tools:context=".MaterialDesign">
```

```
<Button
    style="@style/Widget.MaterialComponents.Button.OutlinedButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Material Button" />
```

```
<com.google.android.material.textfield.TextInputLayout
    style="@style/Widget.MaterialComponents.TextInputLayout.FilledBox"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:hint="Enter Name">
```

```
<com.google.android.material.textfield.TextInputEditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```



```
</com.google.android.material.textfield.TextInputLayout>
```

```
<com.google.android.material.textfield.TextInputLayout
```

```
    style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:hint="Enter Name">
```

```
<com.google.android.material.textfield.TextInputEditText
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

```
</com.google.android.material.textfield.TextInputLayout>
```

```
<com.google.android.material.textfield.TextInputLayout
```

```
    style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:hint="Enter Password"
    app:enablePasswordToggle="true"
    app:enablePasswordToggleTint="#B61004"
    app:startIconDrawable="@android:drawable/ic_lock_lock"
    app:startIconTint="#FF5722">
```

```
<com.google.android.material.textfield.TextInputEditText
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

```
</com.google.android.material.textfield.TextInputLayout>
```

```
<com.google.android.material.textfield.TextInputLayout
```

```
    style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:hint="Enter Name"
    app:counterEnabled="true"
    app:counterMaxLength="20"
    app:errorEnabled="true"
    app:helperText="Your Name Should Not More then 20">
```

```
<com.google.android.material.textfield.TextInputEditText
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

```
</com.google.android.material.textfield.TextInputLayout>
```

```
<Button
```

```
android:id="@+id/snackbar"  
android:layout_width="match_parent"  
android:layout_height="wrap_content"  
android:layout_marginTop="10dp"  
android:text="Snackbar" />
```

**<Button**

```
android:id="@+id/datePicker"  
android:layout_width="match_parent"  
android:layout_height="wrap_content"  
android:layout_marginTop="10dp"  
android:text="DatePicker" />
```

**<Button**

```
android:id="@+id/rangedatePicker"  
android:layout_width="match_parent"  
android:layout_height="wrap_content"  
android:layout_marginTop="10dp"  
android:text="Ranged DatePicker" />
```

**</LinearLayout>**

- Implement Java functionality in MainActivity.java file

```
import android.os.Bundle;
```

```
import android.view.View;
```

```
import android.widget.Toast;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
import androidx.core.util.Pair;
```

```
import com.google.android.material.datepicker.MaterialDatePicker;
```

```
import com.google.android.material.datepicker.MaterialPickerOnPositiveButtonClickListener;
```

```
import com.google.android.material.snackbar.Snackbar;
```

```
public class MaterialDesign extends AppCompatActivity {
```

```
    Snackbar snackbar;
```

```
    @Override
```

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

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_material_design);
```

```
        /*This is Snackbar Code*/
```

```
        findViewById(R.id.snackbar).setOnClickListener(new View.OnClickListener() {
```

```
            @Override
```

```
            public void onClick(View view) {
```

```
                snackbar = Snackbar.make(view, "This Is Snackbar", Snackbar.LENGTH_LONG)
```

```
            ;
```

```

        snackbar.setAction("Retry", new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(MaterialDesign.this, "Retry", Toast.LENGTH_SHORT).show();
            }
        });
        snackbar.setBackgroundTint(getResources().getColor(R.color.colorPrimaryDark));
        snackbar.setActionTextColor(getResources().getColor(android.R.color.white));
        snackbar.show();
    }
});

/*This is DatePicker*/

findViewById(R.id.datePicker).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(final View view) {

        MaterialDatePicker.Builder builderdate = MaterialDatePicker.Builder.datePicker();

        final MaterialDatePicker materialDatePicker = builderdate.build();
        materialDatePicker.show(getSupportFragmentManager(), "");

        materialDatePicker.addOnPositiveButtonClickListener(new MaterialPickerOnPositiveButtonClickListener() {
            @Override
            public void onPositiveButtonClick(Object selection) {
                snackbar = Snackbar.make(view, "" + materialDatePicker.getHeaderText(), Snackbar.LENGTH_LONG);
                snackbar.show();
            }
        });
    }
});

/*This is ranged DatePicker*/

findViewById(R.id.rangedatePicker).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(final View view) {

        MaterialDatePicker.Builder<Pair<Long, Long>> rangeDateBuilder = MaterialDatePicker.Builder.dateRangePicker();
        final MaterialDatePicker materialDatePicker = rangeDateBuilder.build();
        materialDatePicker.show(getSupportFragmentManager(), "");
        materialDatePicker.addOnPositiveButtonClickListener(new MaterialPickerOnPositiveButtonClickListener() {
            @Override
            public void onPositiveButtonClick(Object selection) {
                snackbar = Snackbar.make(view, "" + materialDatePicker.getHeaderText(), Snackbar.LENGTH_LONG);
                snackbar.show();
            }
        });
    }
});

```



```

tiveButtonClickListener() {
    @Override
    public void onPositiveButtonClick(Object selection) {
        snackbar = Snackbar.make(view, "" + materialDatePicker.getHeaderText(), S
nackbar.LENGTH_LONG);
        snackbar.show();
    }
}
});
});
}
}
}
}
}

```

## Output:

