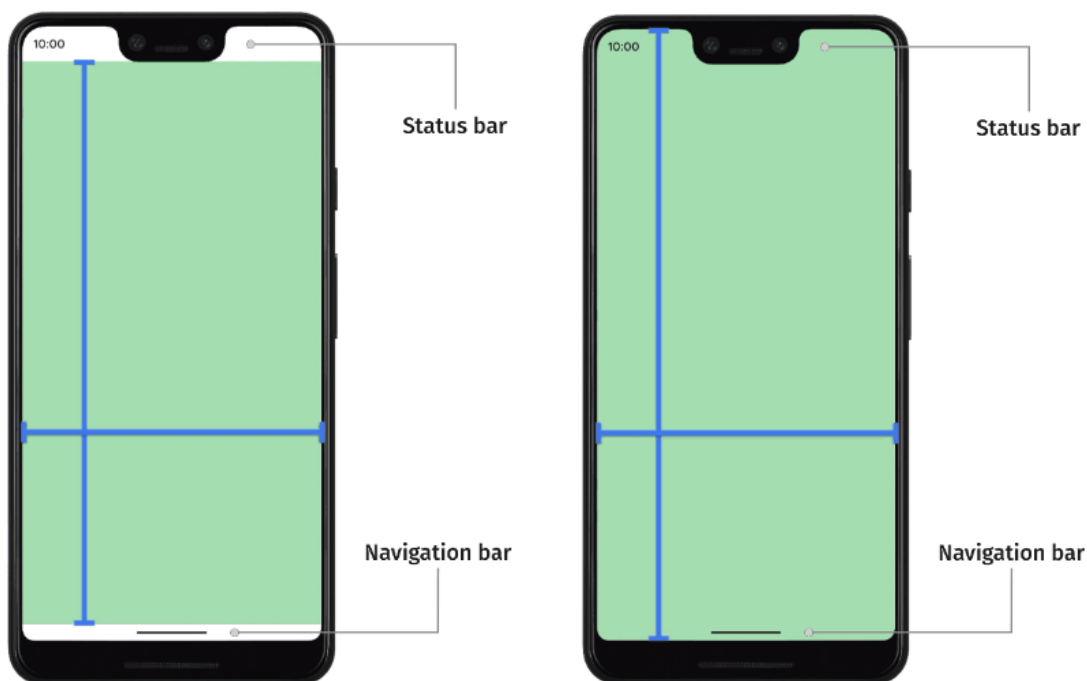


# Lay out your app within window insets

By default, apps are laid out below the status bar at the top and above the navigation bar at the bottom. Together, the status bar and the navigation bar are called the *system bars*. The system bars are areas that are generally dedicated to the display of notifications, communication of device status, and device navigation. However, you can configure your app to display content in these areas.

The overlaps between your app and the areas where the system UI is displayed is one example of *window insets*, which represent the parts of your screen where your app can intersect with the system UI. Intersecting with these parts of the UI can mean displaying above the content, but it can also inform your app about system gestures.



## Step 1:

If you are using custom themes for your app in `res/values/styles.xml`, then add these lines of code to the theme.

```
<item name="android:navigationBarColor">@android:color/transparent</item>
<item name="android:windowTranslucentNavigation">true</item>
<item name="android:windowTranslucentStatus">true</item>
```

Example of this code in a custom theme:

```
<style name="Theme.AppCompat.DayNight.NoActionBar.FullScreen" parent="Theme.AppCompat.DayNight.NoActionBar">
  <item name="windowNoTitle">true</item>
  <item name="windowActionBar">false</item>
  <item name="android:windowFullscreen">true</item>
  <item name="android:windowContentOverlay">@null</item>
  <!-- for translucent navigation bar and status bar -->
  <item name="android:navigationBarColor">@android:color/transparent</item>
  <item name="android:windowTranslucentNavigation">true</item>
  <item name="android:windowTranslucentStatus">true</item>
  <!-- end -->
  <item name="android:textColor">@android:color/black</item>
  <item name="android:editTextStyle">@style/ResetEditText</item>
  <item name="android:editTextBackground">@drawable/rn_edit_text_material</item>
  <!-- <item name="android:statusBarColor">@android:color/transparent</item> -->
</style>
```

This code allows the app to draw behind the status and navigation bar.

## Step 2: <SafeAreaView> from react-native-safe-area-context

Allowing the app to draw behind system bars may cause important UI elements like text to get cut out behind notches. To avoid this, we use component < SafeAreaView>.

SafeAreaView renders nested content and automatically applies padding to reflect the portion of the view that is not covered by navigation bars, tab bars, toolbars, and other ancestor views. Moreover, and most importantly, Safe Area's paddings reflect the physical limitation of the screen, such as rounded corners or camera notches (i.e., the sensor housing area on iPhone 13).

```
import { SafeAreaView } from "react-native-safe-area-context";
```

NOT to be confused with SafeAreaView from react-native library which only works on ios devices.

Wrap the top-level view with SafeAreaView. Ex:

```
return (
  <>
    <StatusBar barStyle={"dark-content"} translucent={true} />
    <SafeAreaView style={styles.container}>
      <Text>Open up App.js to start working on your app!</Text>
      <Text>
        {height} {width}
      </Text>
    </SafeAreaView>
  </>
);
```

Here adding the values of window-height with top and bottom insets gives the screen height.

---

```
insets dimensions top:28.44444465637207
bottom:16 left:0 right:0
```

```
Screen dimensions: h:832 w:384
```

---

```
Window dimensions: h:787.5555555555555 w:384
```

## StatusBar Component:

### Props

animated

backgroundColor ●

barStyle

hidden

networkActivityIndicatorVisible ●

showHideTransition ●

translucent ●

### Step 3:

Sometimes, due to the user's settings for dark and light mode on their device, the status bar content won't be visible as it will be white text on a white background or dark text on a dark background. To avoid this, we use react hook `useColorScheme()`. Ex:

```
function App(): JSX.Element {
  const isDarkMode = useColorScheme() === 'dark';

  const backgroundStyle = {
    backgroundColor: isDarkMode ? Colors.darker : Colors.lighter,
  };

  return (
    <SafeAreaView style={backgroundStyle}>
      <StatusBar
        barStyle={isDarkMode ? 'light-content' : 'dark-content'}
        backgroundColor={backgroundStyle.backgroundColor}
      />
    </SafeAreaView>
  );
}
```

`barStyle` prop is important here as visibility of content in the status bar is dependent on this prop. It is preferred to use `dark-content` on lighter background colours and `light-content`

## Step 4:

Because of the use of gestures for navigation, padding should be added on the sides to allow for gesture insets.



SafeAreaView accounts for padding on the top and bottom, but padding has to be added to interactive UI elements on the sides.

## Adaptive Icons









Step 1: Make a logo with a transparent background. Ex:



## Step 2:

Go to <https://easyappicon.com/>. The way that an adaptive icon works is that there is a foreground image (the logo), the background image (which can be set to any colour). Then according to the devices default settings, the foreground + background image is masked.

After downloading the zip file from this website, these will be the contents.

 mipmap-anydpi-v26	04-07-2023 16:18	File folder
 mipmap-hdpi	04-07-2023 16:18	File folder
 mipmap-ldpi	04-07-2023 16:18	File folder
 mipmap-mdpi	04-07-2023 16:18	File folder
 mipmap-xhdpi	04-07-2023 16:18	File folder
 mipmap-xxhdpi	04-07-2023 16:18	File folder
 mipmap-xxxhdpi	04-07-2023 16:18	File folder
 values	04-07-2023 16:18	File folder

Add these files to your existing project in the android res folder.

## Step 3:

Add these lines of code to your AndroidManifest.xml file if not already present.

```
<application
...
    android:icon="@mipmap/ic_launcher"
    android:roundIcon="@mipmap/ic_launcher_round"
...>
</application>
```

