Reasons to Choose Android OS Platform For Mobile Application Development

1. Wider Pool of Customers

2. Greater Innovative Potential

3. Easier Installation

4. Easier Entry to Market

5. Custom ROMs

6. Compatible with More Devices

7. Social Media Optimization

8. Easier to Learn (For Developers)

9. Cheap Apps and Free Apps

10. Lower Development Cost (OS Perspective)

Modern Android App Architecture

Architecture Overview

Understand why designing a good app Architecture matters. Learn about common architectural principles, the recommended app architecture, how to manage dependencies between components, and general best practices.

UI Layer

Learn about the role of the UI layer, how to define, manage, expose, and consume UI state using the principles of Unidirectional Data Flow, and how to show in-progress operations and errors on the screen.

Data Layer

Learn about the role of the Data layer, the different entities involved, the APIs to expose, threading, how to define a source of truth, the different types of data operations, how to test this layer, and examples of common tasks this layer usually performs.

Domain Layer

Learn about the role of the Domain layer, why it's optional and when you should need it, how to better invoke use cases, and examples of common tasks this layer could perform.

Handling UI events

Learn how to handle different types of UI events in the UI layer for events that are triggered both by the user or parts of your code. Also, learn what to do if your use case seems to be different from the ones covered in this page.

Android Layout and Views – Types and Examples

What is Android View?

A View is a simple building block of a user interface. It is a small rectangular box that can be TextView, EditText, or even a button. It occupies the area on the screen in a rectangular area and is responsible for drawing and event handling. View is a superclass of all the graphical user interface components.

Why and How to use the View in Android?

Now you might be thinking what is the use of a View. So, the use of a view is to draw content on the screen of the user's Android device. A view can be easily implemented in an Application using the java code. Its creation is more easy in the XML layout file of the project. Like, the project for hello world that we had made initially.

Types of Android Views:

Another thing that might now come to your mind must be, “what are the available types of view in Android that we can use?”
For that, we’ll see all these types one by one as follows:

- Text View
- Edit Text
- Button
- Image Button
- Date Picker
- Radio Button
- Check Box buttons
- Image View

And there are some more components.

What is Android View Group?

A View Group is a subclass of the View Class and can be considered as a superclass of Layouts. It provides an invisible container to hold the views or layouts. View Group instances and views work together as a container for Layouts. To understand in simpler words it can be understood as a special view that can hold other views that are often known as a child view.

Following are certain commonly used subclasses for View Group:

- Linear Layout
- Relative Layout
- Frame Layout
- Grid View
- List View

What is Android Layout?

Layout basically refers to the arrangement of elements on a page these elements are likely to be images, texts or styles. These are a part of **Android Jetpack**.

```
<?xml version="1.0" encoding="utf-8"?>

 <LinearLayout
     android:id="@+id/layout2"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:layout_weight="1"
     android:background="#8ED3EB"
     android:gravity="center"
     android:orientation="vertical" >

     <TextView
        android:id="@+id/textView4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:layout_marginTop="-40dp"
        android:fontFamily="@font/almendra_bold"
        android:text="This is a TextView" />

 </LinearLayout>
```

Types of Layouts in Android

Now that we’ve learned about the view and view groups and also somewhat about the layouts. Subsequently let us see the types of Layouts in Android, that are as follows:

- Linear Layout
- Relative Layout
- Constraint Layout
- Table Layout
- Frame Layout
- List View
- Grid View
- Absolute Layout
- WebView
- Scroll View

1. Linear Layout

We use this layout to place the elements in a linear manner. A Linear manner means one element per line. This layout creates various kinds of forms on Android. In this, arrangement of the elements is in a top to bottom manner.

This can have two orientations:
a. Vertical Orientation – It is shown above where the widgets such as TextViews, and all in a vertical manner.
b. Horizontal Orientation – It is shown above where the widgets such as TextViews, and all in a horizontal manner.

2. Relative Layout

This layout is for specifying the position of the elements in relation to the other elements that are present there.
In the relative layout, alignment of the position of the elements to the parent container is possible. To define it in such a way, we write the following:

- android:layout_alignParentTop= “true”
- android:layout_alignParentLeft= “true”

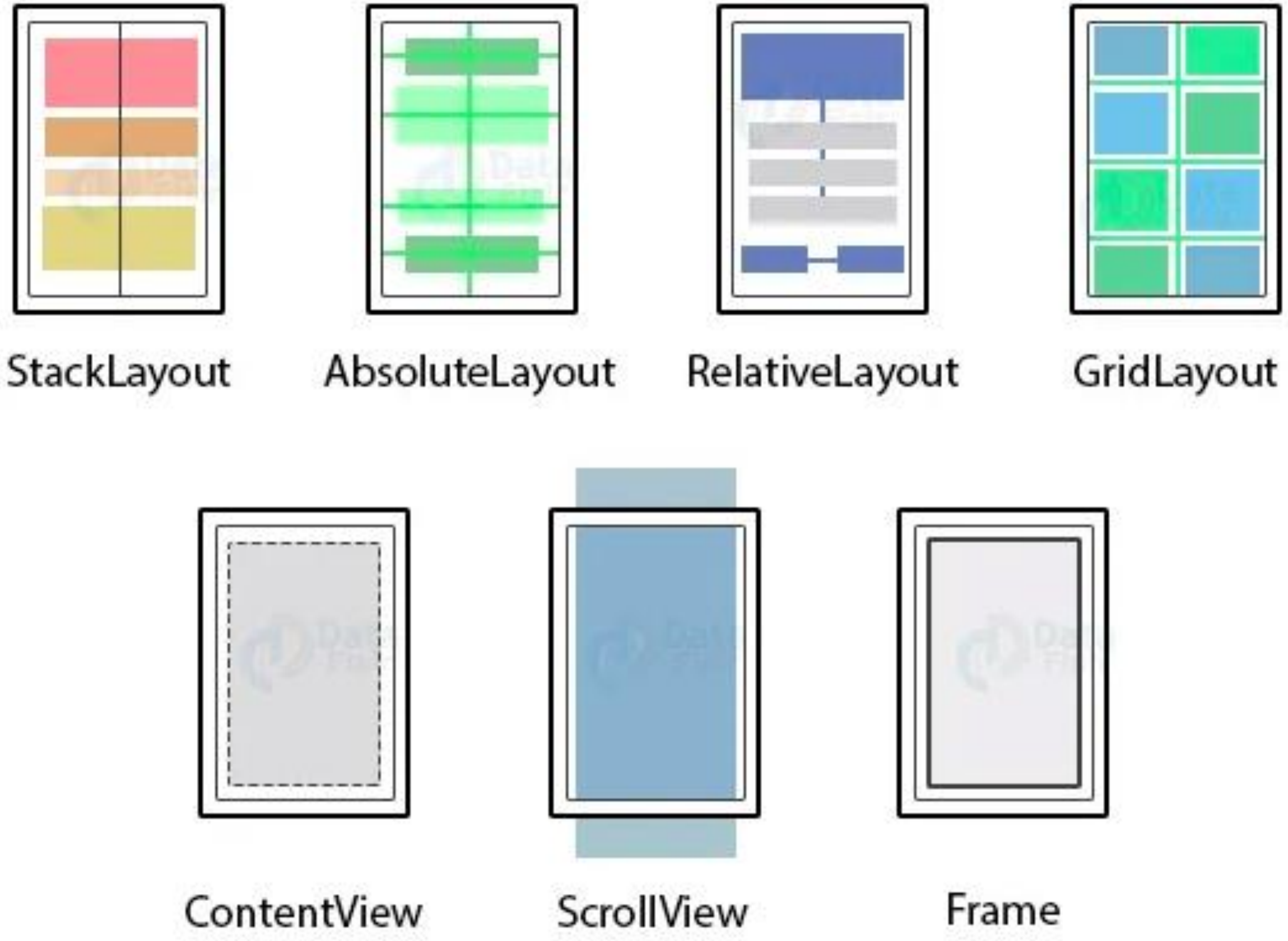
If we write the above code, the element will get aligned on the top left of the parent container.

If we want to align it with some other element in the same container, it can be defined is as follows:

- android:layout_alignLeft= “@+id/element_name”
- android:layout_below= “@+id/element_name”

This will align the element below the other element to its left.

Types of Android Layouts



The following are the attributes for customizing a Layout while defining it:

- **android:id:** It uniquely identifies the Android Layout.
- **android:hint:** It shows the hint of what to fill inside the EditText.
- **android:layout_height:** It sets the height of the layout.
- **android:layout_width:** It sets the width of the layout.
- **android:layout_gravity:** It sets the position of the child view.
- **android:layout_marginTop:** It sets the margin of the from the top of the layout.
- **android:layout_marginBottom:** It sets the margin of the from the bottom of the layout.
- **android:layout_marginLeft:** It sets the margin of the from the left of the layout.
- **android:layout_marginRight:** It sets the margin of the from the right of the layout.
- **android:layout_x:** It specifies the x coordinates of the layout.
- **android:layout_y:** It specifies the y coordinates of the layout.

Android Linear Layout

- android:id
- android:baselineAligned
- android:baselineAlignedCh ildIndex
- android:divider
- android:gravity
- android:orientation android:weightSum

Android Relative Layout

- android:id
- android:gravity
- android:ignoreGravity
- android:layout_above
- android:layout_alignBottom
- android:layout_alignLeft
- android:layout_alignParentBottom
- android:layout_alignParentEnd
- android:layout_alignParentLeft
- ndroid:layout_alignParentRight
- android:layout_alignParentStart
- android:layout_alignParentTop
- android:layout_alignRight
- android:layout_alignStart
    - android:layout_alignTop

- android:layout_alignTop
- android:layout_below
- android:layout_centerHorizontal
- android:layout_centerInParent
- android:layout_centerVertical
- android:layout_toEndOf
- android:layout_toLeftOf
- android:layout_toRightOf
- android:layout_toStartOf

Android Table Layout

- android:id
- android:collapseColumns
- android:shrinkColumns
- android:stretchColumns

Android Absolute Layout

- **android:id -**This is the ID which uniquely identifies the layout.
- **android:layout_x -** This specifies the x-coordinate of the view.
- **android:layout_y -** This specifies the y-coordinate of the view.

Frame Layout Attributes

•**android:id - th**is is the ID which uniquely identifies the layout.

•**android:foreground -** this defines the drawable to draw over the content and possible values may be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".

•**android:foregroundGravity -** Defines the gravity to apply to the foreground drawable. The gravity defaults to fill. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.

•**android:measureAllChildren -** Determines whether to measure all children or just those in the VISIBLE or INVISIBLE state when measuring. Defaults to false.

ListView Attributes

- **android:id -** This is the ID which uniquely identifies the layout.
- **android:divider- t**his is drawable or color to draw between list items
- **android:entries -** Specifies the reference to an array resource that will populate the ListView
- **android:dividerHeight -** This specifies height of the divider. This could be in px, dp, sp, in, or mm
- **android:headerDividersEnabled -** When set to false, the ListView will not draw the divider after each header view. The default value is true
- **android:footerDividersEnabled -** When set to false, the ListView will not draw the divider before each footer view. The default value is true

GridView Attributes

- **android:id -** This is the ID which uniquely identifies the layout.
- **android:columnWidth -** his specifies the fixed width for each column. This could be in px, dp, sp, in, or mm.
- **android:gravity -** Specifies the gravity within each cell. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
- **android:horizontalSpacing -** Defines the default horizontal spacing between columns. This could be in px, dp, sp, in, or mm.
- **android:numColumns -** Defines how many columns to show. May be an integer value, such as "100" or auto_fit which means display as many columns as possible to fill the available space.
- **android:stretchMode -** Defines how columns should stretch to fill the available empty space, if any.
- **android:verticalSpacing -** Defines the default vertical spacing between rows. This could be in px, dp, sp, in, or mm.

Android Activity Lifecycle with example in Kotlin

Android Activity Lifecycle: is managing the state of Activity like when its start, stop, user, using, not in front of the user, no longer. So that all states are managing by call back methods in action. You can override those methods and can do a particular operation to do the best output of your application.

Like your application is going in the background and you want to save some data, in this case, you have to know about **Android Activity Lifecycle.**

Why Use Activity Lifecycle callback methods?

If you do a good implementation of Lifecycle call-backs methods in your android app it will avoid many problems, Like :

- If the user files the form and suddenly rotates the screen so all filled data will be lost, It’s not good.
- Android limited resources can crash your app if you didn’t release it on time
- Losing the user’s progress if they leave your app and return to it at a later time.

Many more problems you can save by just using the proper guideline of **Activity Lifecycle.**

Activity has six states

- *Created*
- *Started*
- *Resumed*
- *Paused*
- *Stopped*
- *Destroyed*

Activity lifecycle has seven methods

- onCreate()
- onStart()
- onResume()
- onPause()
- onStop()
- onRestart()
- onDestroy()

Practical Situations :

When open the app
onCreate() –> onStart() –> onResume()

When back button pressed and exit the app
onPaused() — > onStop() –> onDestory()

When home button pressed
onPaused() –> onStop()

After pressed home button when again open app from recent task list or clicked on app icon
onRestart() –> onStart() –> onResume()

When open app another app from notification bar or open settings
onPaused() –> onStop()

Back button pressed from another app or settings then used can see our app
onRestart() –> onStart() –> onResume()

Practical Situations :

When any dialog open on screen
onPause()

After dismiss the dialog or back button from dialog
onResume()

Any phone is ringing and user in the app
onPause() –> onResume()

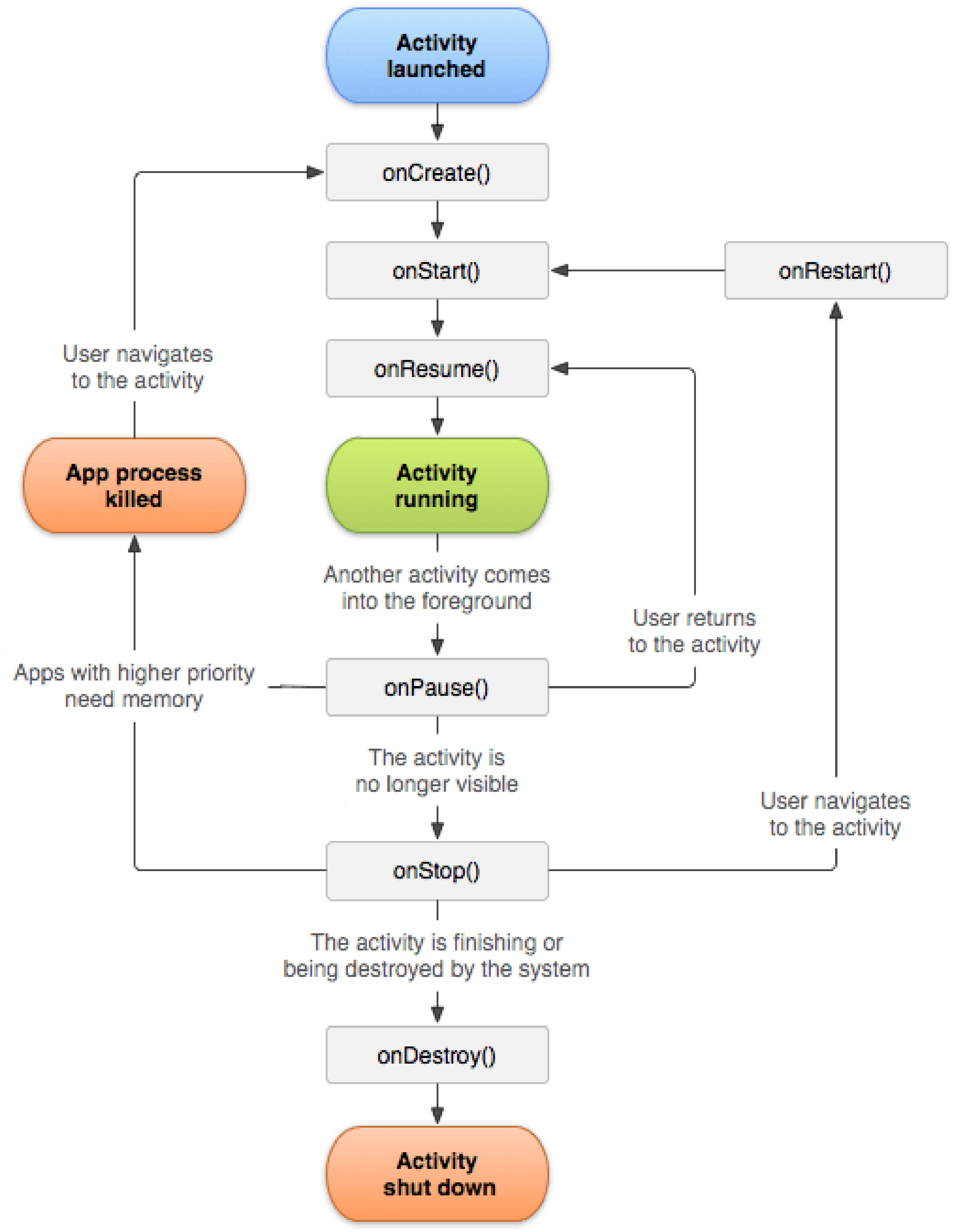
When user pressed phone’s answer button
onPause()

After call end
onResume()

When phone screen off
onPaused() –> onStop()

When screen is turned back on
onRestart() –> onStart() –> onResume()

Let's come into deep: **Lifecycle callbacks**


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

onCreate()

When the android system first creates the activity, it will call onCreate(). Used to initialize the activity, for example, create the user interface.

onStart()

When the activity enters the Started state, the system invokes this callback. The onStart() call makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive.

onResume()

When the activity enters the Resumed state, it comes to the foreground, and then the system invokes the onResume()callback. This is the state in which the app interacts with the user.

onPause()

The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed); it indicates that the activity is no longer in the foreground (though it may still be visible if the user is in multi-window mode).

onStop()

When your activity is no longer visible to the user, it has entered the Stopped state, and the system invokes theonStop() callback. This may occur, for example, when a newly launched activity covers the entire screen.

onRestart()

From the Stopped state, the activity either comes back to interact with the user, or the activity is finished running and goes away. If the activity comes back, the system invokes onRestart()

onDestroy()

Android system invokes onDestroy() method before the activity is destroyed. Activity is destroyed because of :

1. the activity is finishing (due to the user completely dismissing the activity or due to finish() being called on the activity), or
2. the system is temporarily destroying the activity due to a configuration change (such as device rotation or multi-window mode)
3. or a memory issue in the app

Some more callbacks methods added in activity lifecycle for saving and retrieving data :

onSaveInstanceState(Bundle outState)

As your activity begins to stop, the system calls the methodonSaveInstanceState() so your activity can save state information to an instance state bundle.

onRestoreInstanceState(Bundle savedInstanceState)

When your activity is recreated after it was previously destroyed, you can recover your saved instance state from the Bundle that the system passes to your activity. Both the onCreate() and onRestoreInstanceState() callback methods receive the same Bundle that contains the instance state information.

Check the code :

Step 1. Create new project “Build Your First Android App in Kotlin“
Step 2. Open MainActivity.kt kotlin class and override callbacks methods
add a print method with a log.

Check the code :

```
package `in`.eyehunt.androidactivitylifecyclekotlin

import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.util.Log

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        print("onCreate")
    }

    override fun onStart() {
        super.onStart()
        print("onStart")
    }
```

Check the code :

```
override fun onResume() {
    super.onResume()
    print("onResume")
}

override fun onPause() {
    super.onPause()
    print("onPause")
}

override fun onStop() {
    super.onStop()
    print("onStop")
}
```

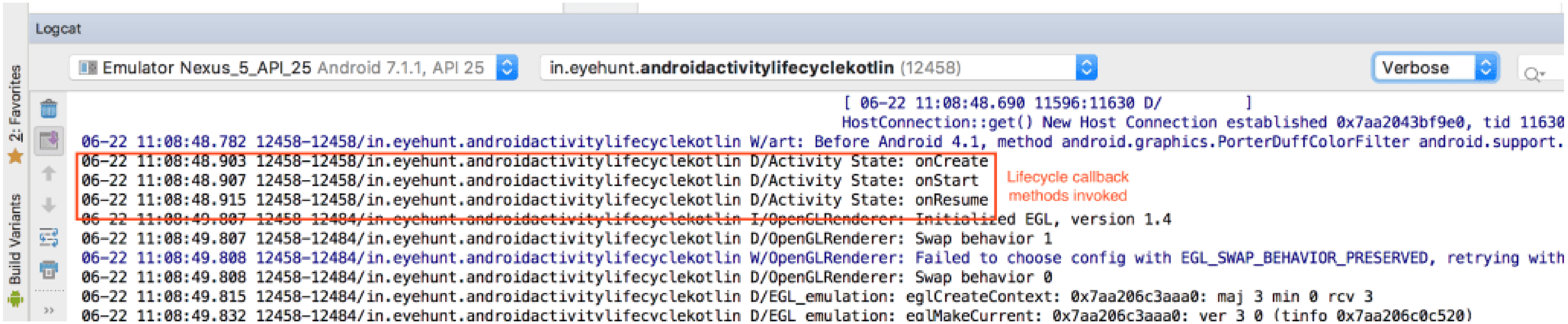
```
    override fun onRestart()
{
        super.onRestart()
        print("onRestart")
    }

    override fun onDestroy()
{
        super.onDestroy()
        print("onDestroy")
    }

    fun print(msg: String){
        Log.d("Activity State
",msg)
    }
}
```

Step 3. Run the application, in the emulator or On your Android device

Check the output in **Logcat**.


Logcat
Emulator Nexus_5_API_25 Android 7.1.1, API 25
in.eyehunt.androidactivitylifecyclekotlin (12458)
Verbose
2: Favorites
Build Variants
Lifecycle callback methods invoked

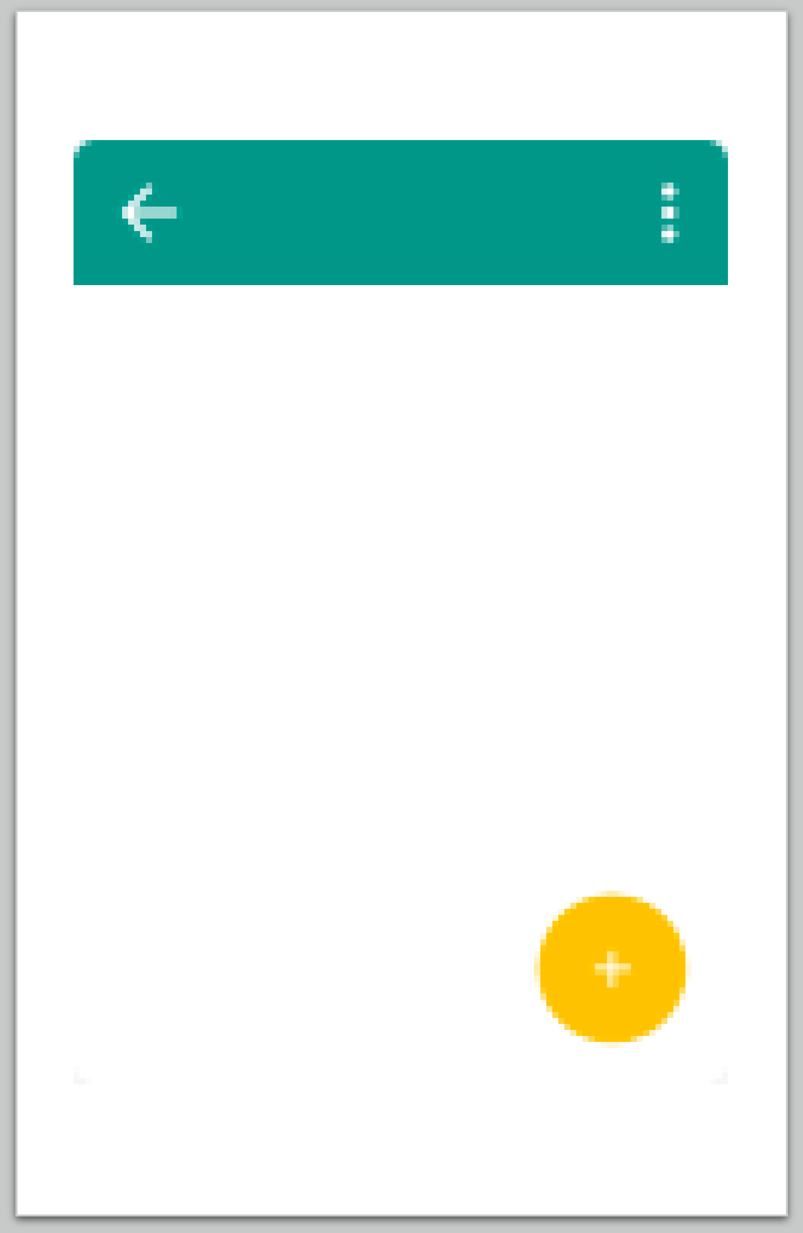
```
                                                                                   [ 06-22 11:08:48.690 11596:11630 D/         ]
                                                                                   HostConnection::get() New Host Connection established 0x7aa2043bf9e0, tid 11630
06-22 11:08:48.782 12458-12458/in.eyehunt.androidactivitylifecyclekotlin W/art: Before Android 4.1, method android.graphics.PorterDuffColorFilter android.support.
06-22 11:08:48.903 12458-12458/in.eyehunt.androidactivitylifecyclekotlin D/Activity State: onCreate
06-22 11:08:48.907 12458-12458/in.eyehunt.androidactivitylifecyclekotlin D/Activity State: onStart
06-22 11:08:48.915 12458-12458/in.eyehunt.androidactivitylifecyclekotlin D/Activity State: onResume
06-22 11:08:49.807 12458-12484/in.eyehunt.androidactivitylifecyclekotlin I/OpenGLRenderer: Initialized EGL, version 1.4
06-22 11:08:49.807 12458-12484/in.eyehunt.androidactivitylifecyclekotlin D/OpenGLRenderer: Swap behavior 1
06-22 11:08:49.808 12458-12484/in.eyehunt.androidactivitylifecyclekotlin W/OpenGLRenderer: Failed to choose config with EGL_SWAP_BEHAVIOR_PRESERVED, retrying with
06-22 11:08:49.808 12458-12484/in.eyehunt.androidactivitylifecyclekotlin D/OpenGLRenderer: Swap behavior 0
06-22 11:08:49.815 12458-12484/in.eyehunt.androidactivitylifecyclekotlin D/EGL_emulation: eglCreateContext: 0x7aa206c3aaa0: maj 3 min 0 rcv 3
06-22 11:08:49.832 12458-12484/in.eyehunt.androidactivitylifecyclekotlin D/EGL_emulation: eglMakeCurrent: 0x7aa206c3aaa0: ver 3 0 (tinfo 0x7aa206c0c520)
```

Activity template

One of the most common uses of templates is adding new activities to an existing app module. For example, to create a login screen for your app's users, add an activity with the Login Activity template.

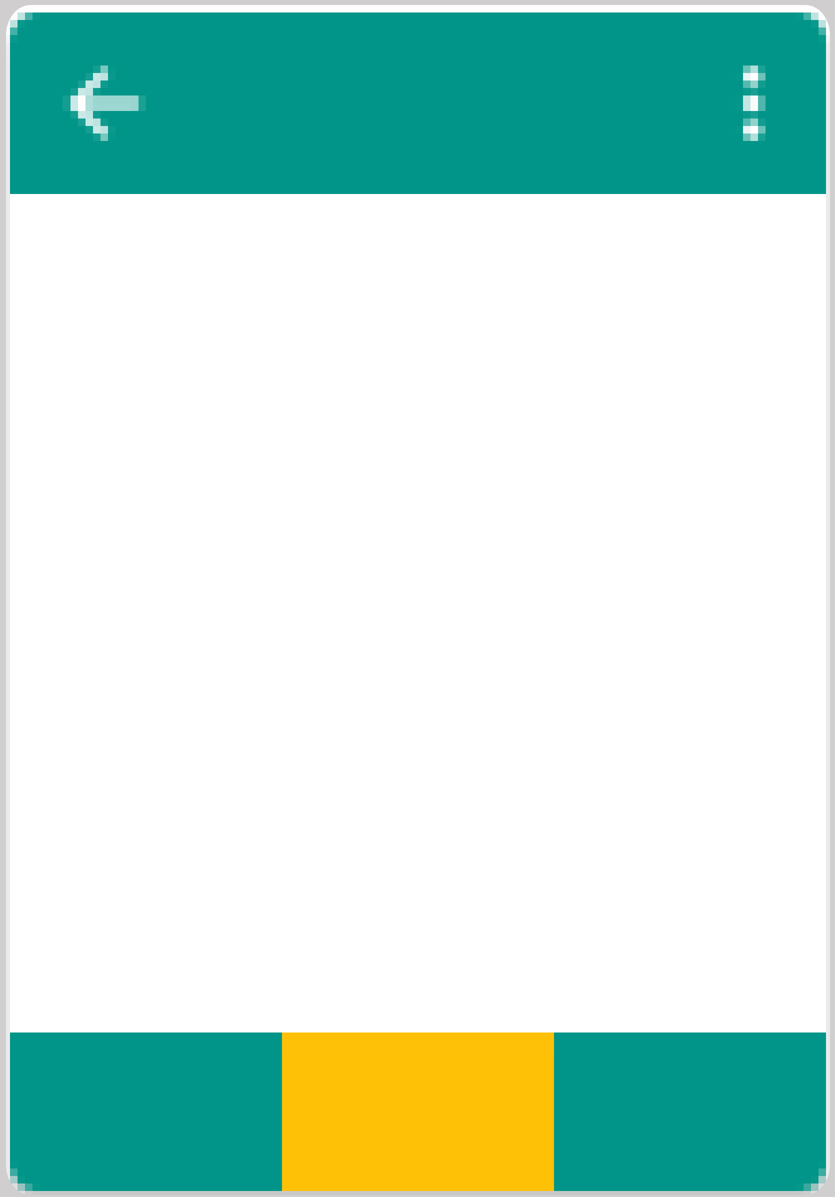
This section covers the commonly used activity templates for phone and tablet apps. Android Studio also provides templates for a variety of different app module types, including Wear OS, Android TV, and Cloud App Engine. You can view templates for these different module types when creating an app module. Templates also exist for more API-specific modules and activities, such as Google AdMobs Ads and Google Maps.

The following phone and tablet templates provide you with the code components for specific usage contexts, such as logging into an account, presenting a list of items with details, or scrolling through a long block of text. Each can serve as either an entire app module or an individual activity.



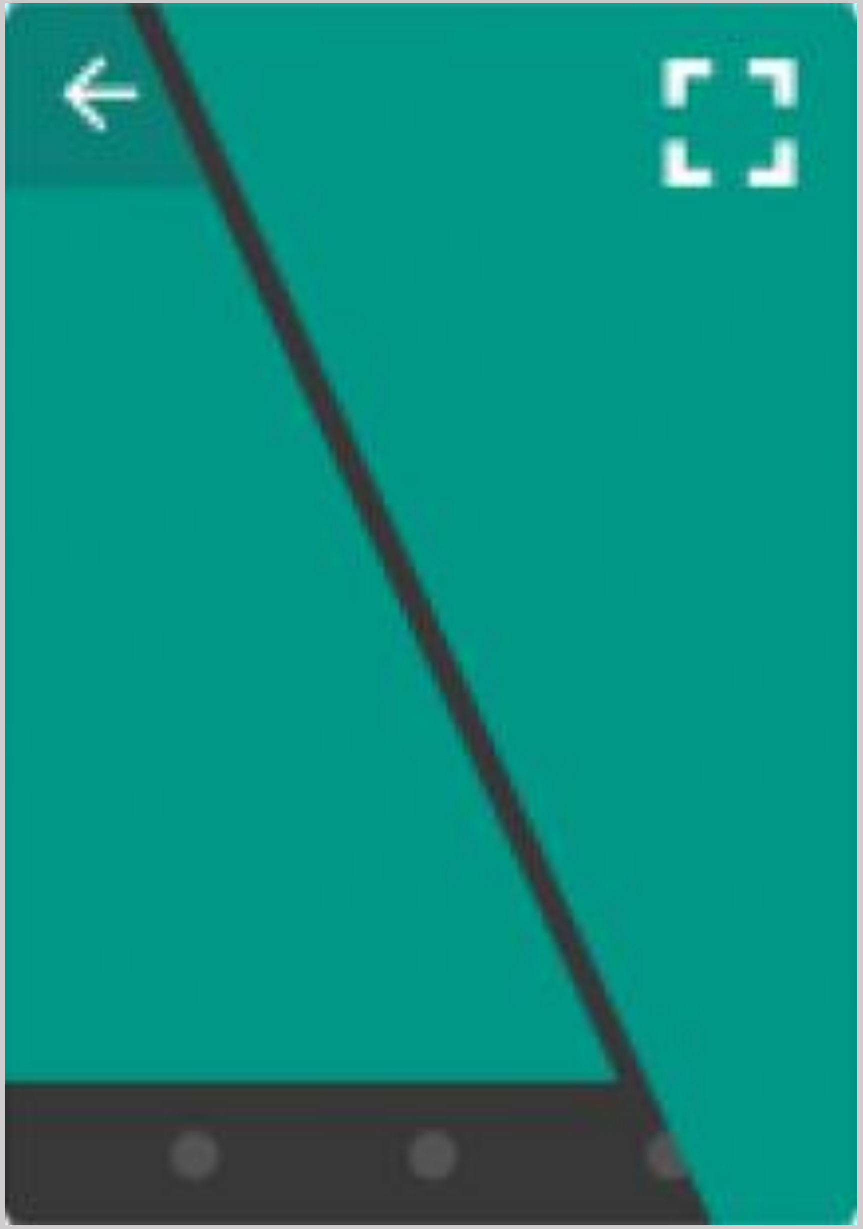
Basic Activity

- This template creates a simple app with an app bar and a floating action button. It acts as a starting point for your project by providing commonly used UI components.
- This template includes:
- AppBar
- FloatingActionButton
- Two layout files: one for the activity and one to separate out text content



Bottom Navigation Activity

- This template provides a standard bottom navigation bar for an activity to make it easy for users to explore and switch between top-level views in a single tap. Use this template when your application has three to five top-level destinations. For more information, see the Bottom Navigation Component design guidelines.
- This template includes:
- AppBar
- Single layout file with sample layout for bottom navigation

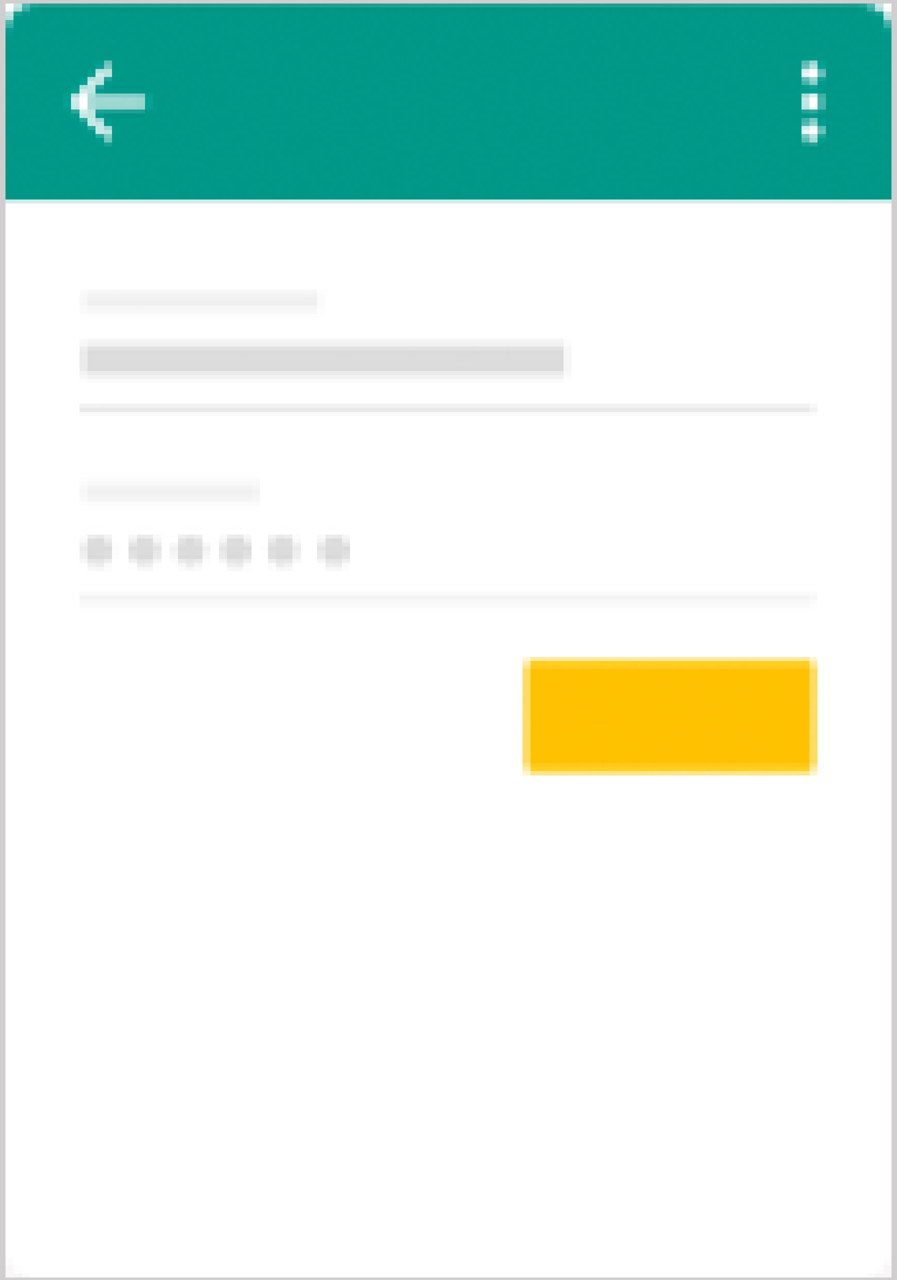


Fullscreen Activity

This template creates an app that alternates between a primary fullscreen view and a view with standard user interface (UI) controls. The fullscreen view is the default and a user can activate the standard view by touching the device screen.

This template includes:

- Touch listener implementation for hiding the standard view elements
- Button that appears in the standard view but does not do anything
- AppBar for the standard view
- Single layout file with both the fullscreen view and a frame layout for standard view elements



Login Activity

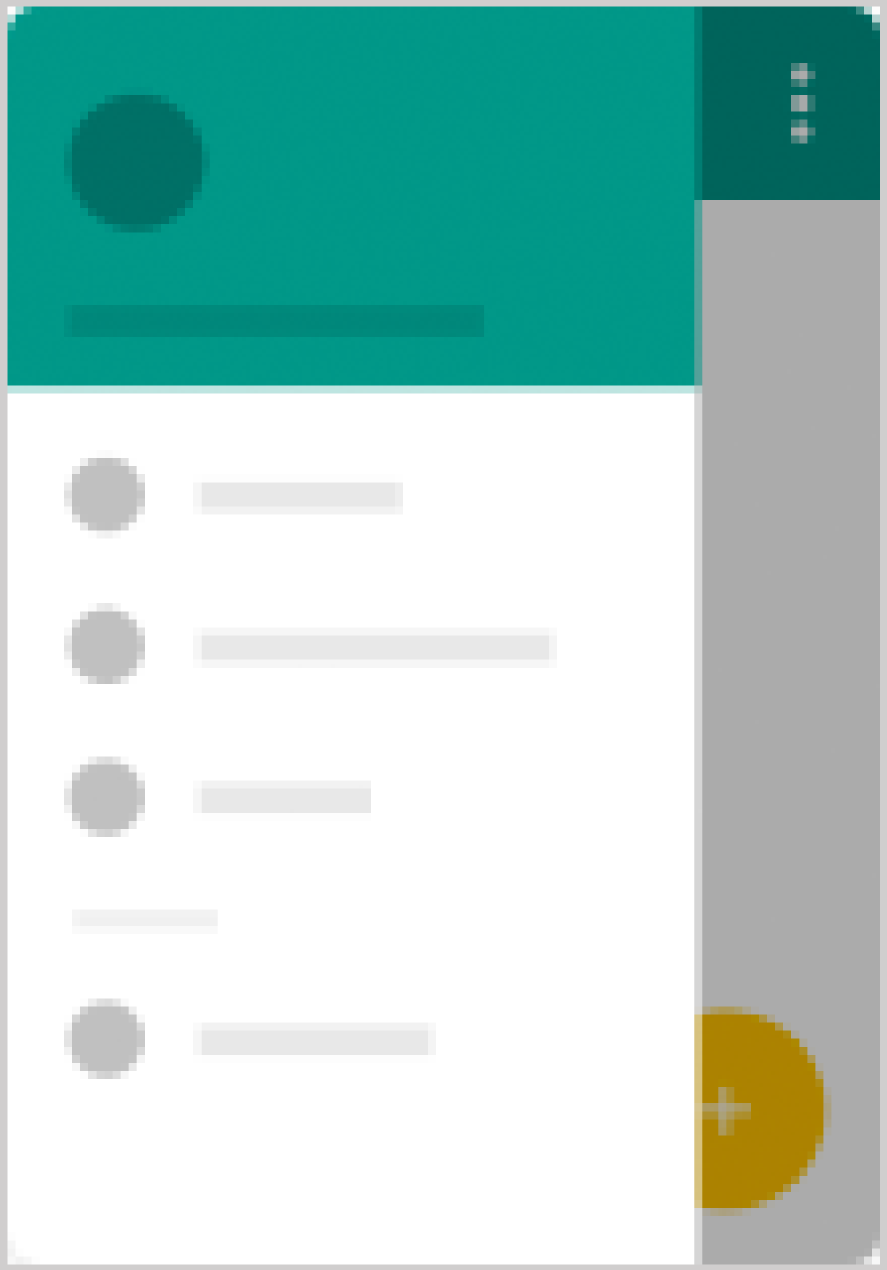
This template creates a standard login screen. The user interface includes email and password fields and a sign-in button. It is more commonly used as an activity template than as an app module template.
This template includes:

- AsyncTask implementation for handling network operations separately from the main user interface thread
- Progress indicator during network operations
- Single layout file with the recommended login UI:
  - Email and password input fields
  - Sign-in button

Primary/Detail Flow (Renamed and updated in 4.2 Canary 8)

This template creates an app that has both an item list display and a display for the details of an individual item. Clicking on an item on the list screen opens a screen with the item's details. The layout of the two displays depends on the device that is running the app.
This template also provides API stubs for handling certain mouse and keyboard inputs such as right-click actions on the list items as well as common keyboard shortcuts.
This template includes:

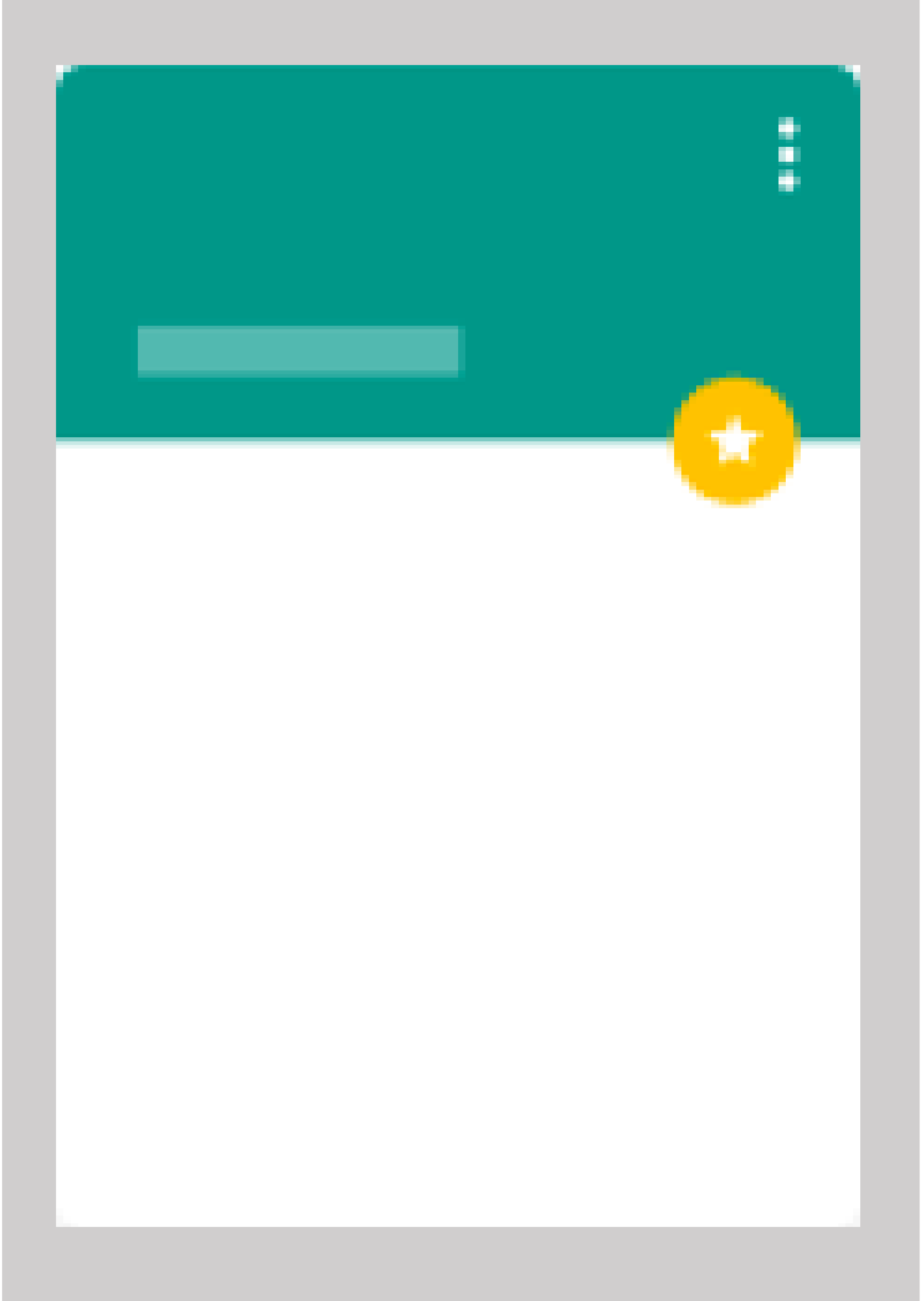
- Fragment representing the list of items
- Fragment displaying an individual item's details
- FloatingActionButton on each screen
- Collapsing toolbar for the item detail screen
- Alternative resource layout files for different device configurations
- ContextClickListener on the list items to handle right-click actions
- OnUnhandledKeyEventListener to detect keyboard shortcuts on the item list fragment



Navigation Drawer Activity

This template creates a **Basic Activity** with a navigation drawer menu. The navigation bar expands from the left or right side of your app and appears in addition to the regular app bar.
This template includes:

- Navigation drawer implementation with a DrawerLayout, corresponding event handlers, and example menu options
- AppBar
- FloatingActionButton
- Layout files for the navigation drawer and the navigation drawer header, in addition to those from the **Basic Activity** template

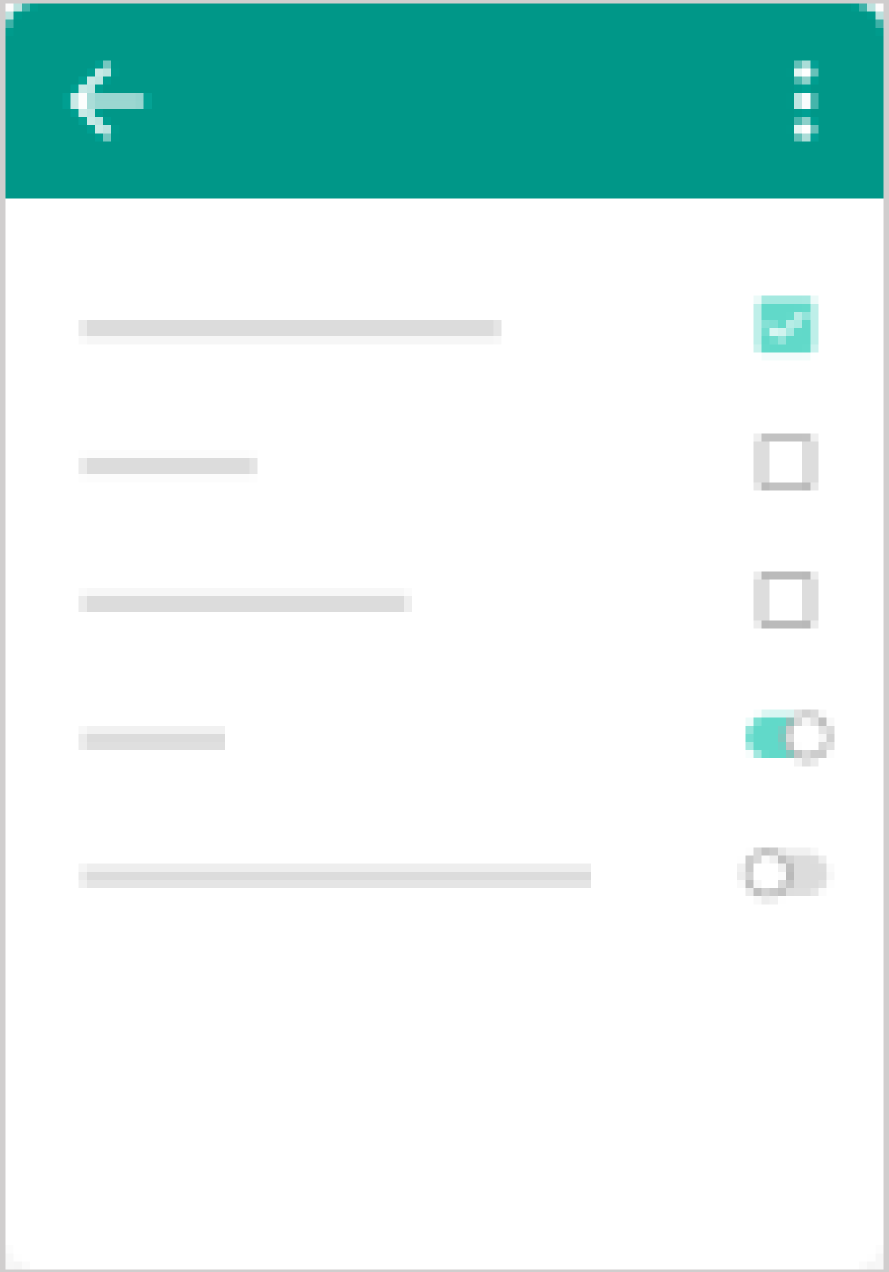


Scrolling Activity

This template creates an app with a collapsing toolbar and a scrolling view for long text content. As you scroll down the page, the toolbar, which can serve as a header, automatically condenses, and the floating action button disappears.

This template includes:

- Collapsing toolbar in place of the regular AppBar
- FloatingActionButton
- Two layout files: one for the activity and one to separate out the text content into a NestedScrollView

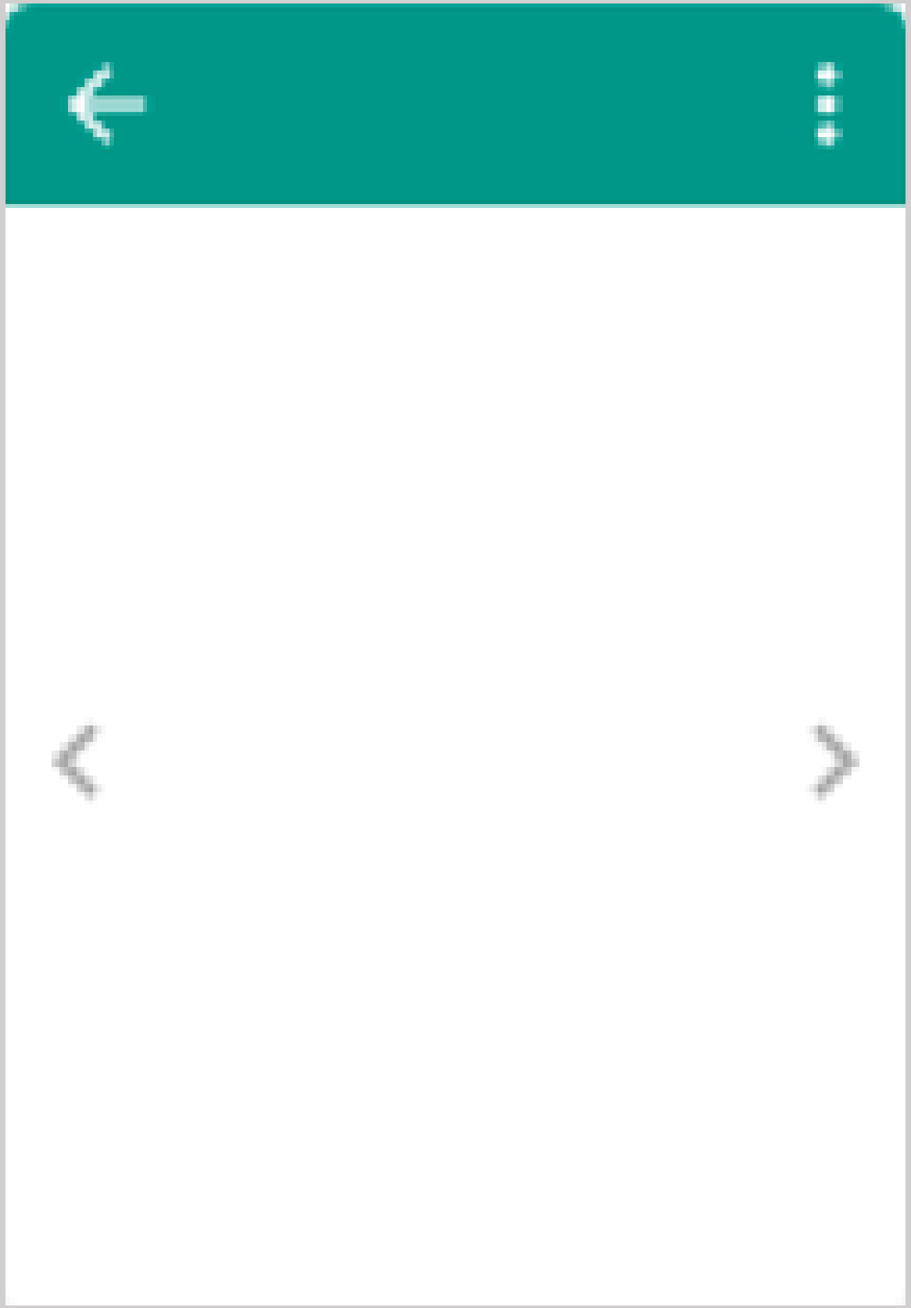


Settings Activity

This template creates an activity that displays user preferences or settings for an app. It extends the PreferenceActivity class and is more commonly used as an activity template than as an app module template.

This template includes:

- Activity that extends PreferenceActivity
- XML files (in the res/xml/ directory of your project) to define the displayed settings



Tabbed Activity

This template creates an app with multiple sections, swiping navigation, and an app bar. The sections are defined as fragments between which you can swipe left and right to navigate.
This template includes:

- AppBar
- Adapter that extends FragmentPagerAdapter and creates a fragment for each section
- ViewPager instance, a layout manager for swiping between sections
- Two layout files: one for the activity and one for individual fragments

Font sizes:

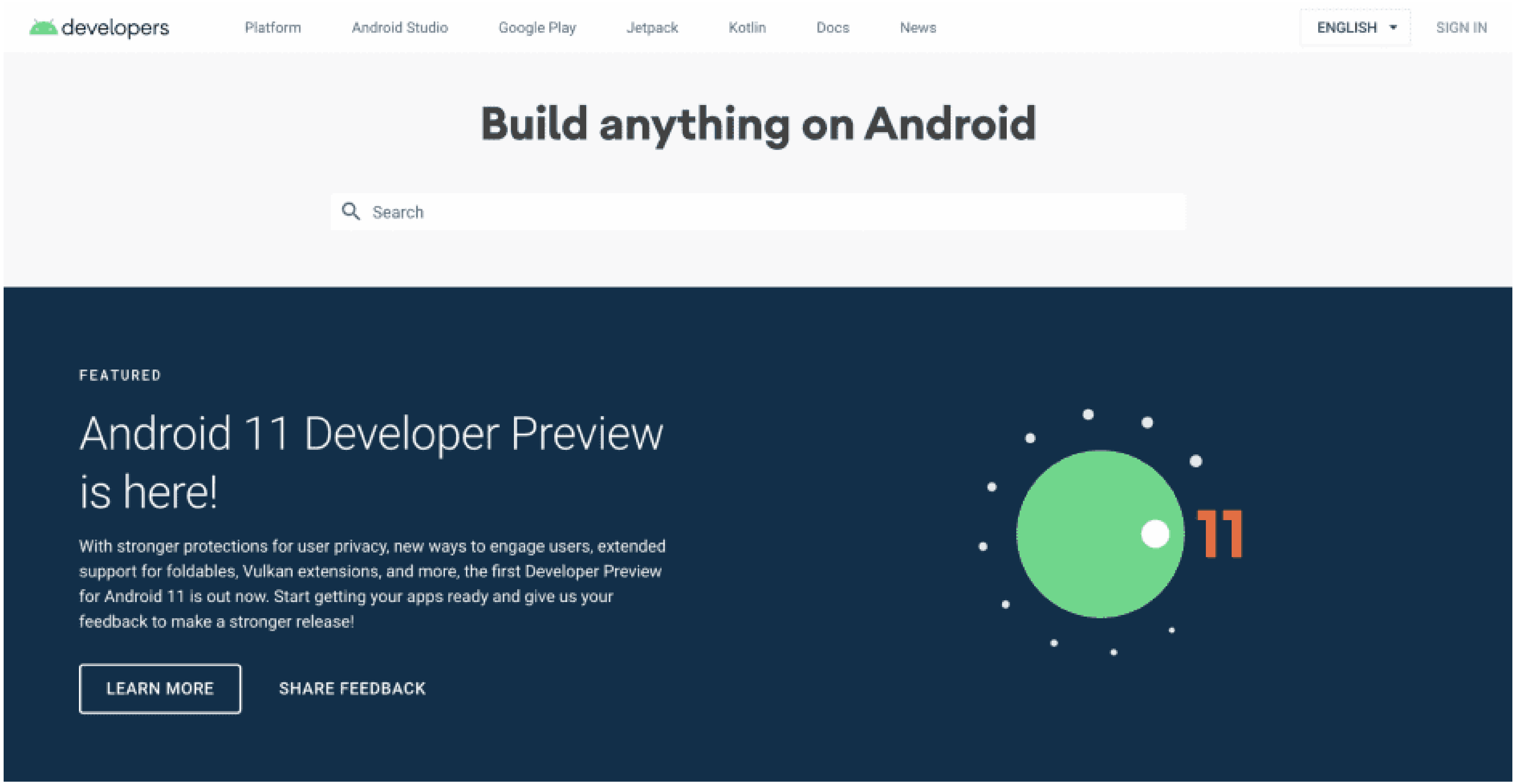
sp for font sizes **dp** for everything else

dp
Density-independent Pixels - an abstract unit that is based on the physical density of the screen. These units are relative to a 160 dpi screen, so one dp is one pixel on a 160 dpi screen. The ratio of dp-to-pixel will change with the screen density, but not necessarily in direct proportion. Note: The compiler accepts both "dip" and "dp", though "dp" is more consistent with "sp".

sp
Scale-independent Pixels - this is like the dp unit, but it is also scaled by the user's font size preference. It is recommend you use this unit when specifying font sizes, so they will be adjusted for both the screen density and user's preference.

How to learn Android development – 6 key steps for beginners

1. Take a look at the official Android website



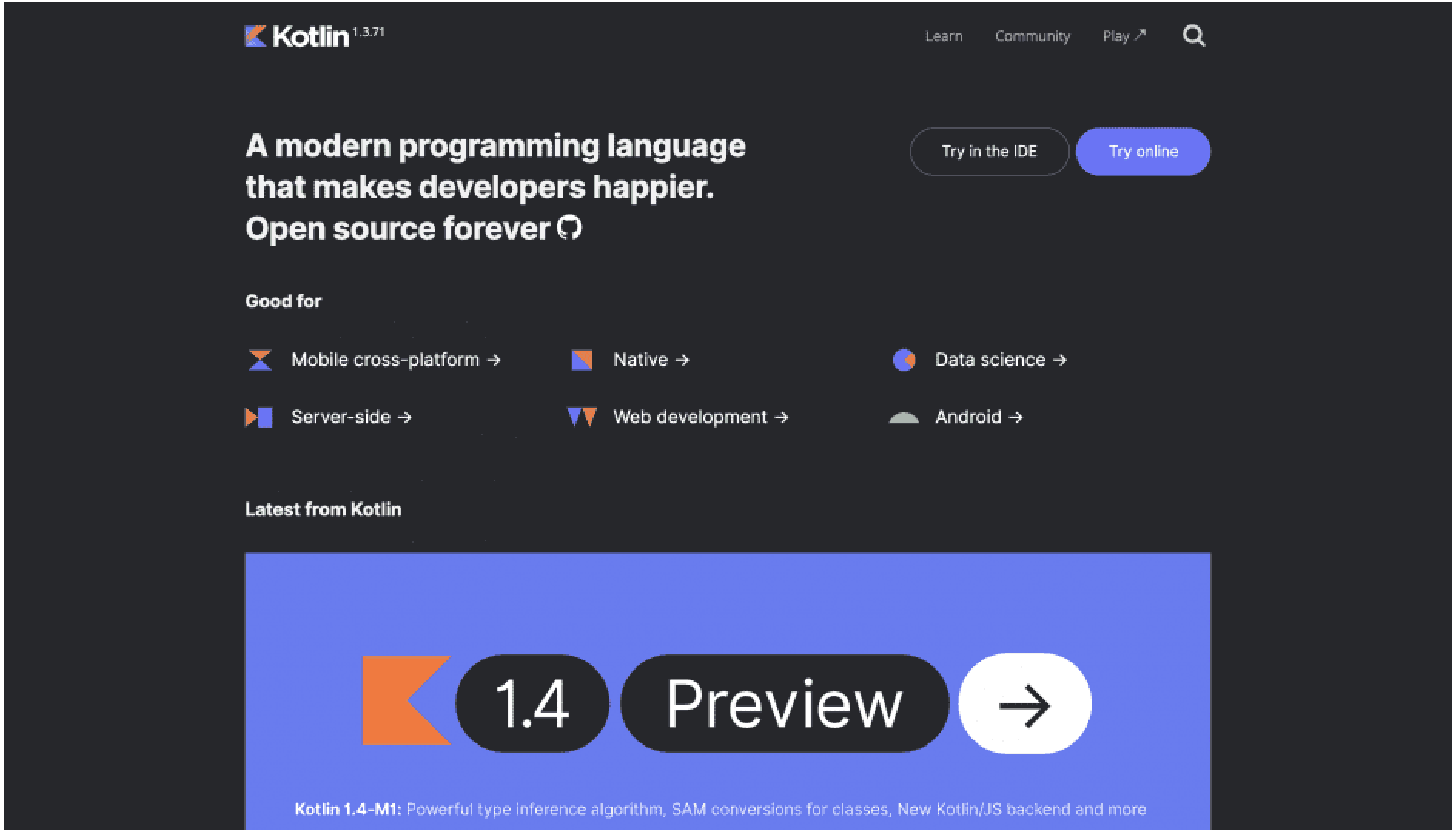
Build anything on Android

FEATURED

Android 11 Developer Preview is here!

With stronger protections for user privacy, new ways to engage users, extended support for foldables, Vulkan extensions, and more, the first Developer Preview for Android 11 is out now. Start getting your apps ready and give us your feedback to make a stronger release!

2. Check out Kotlin



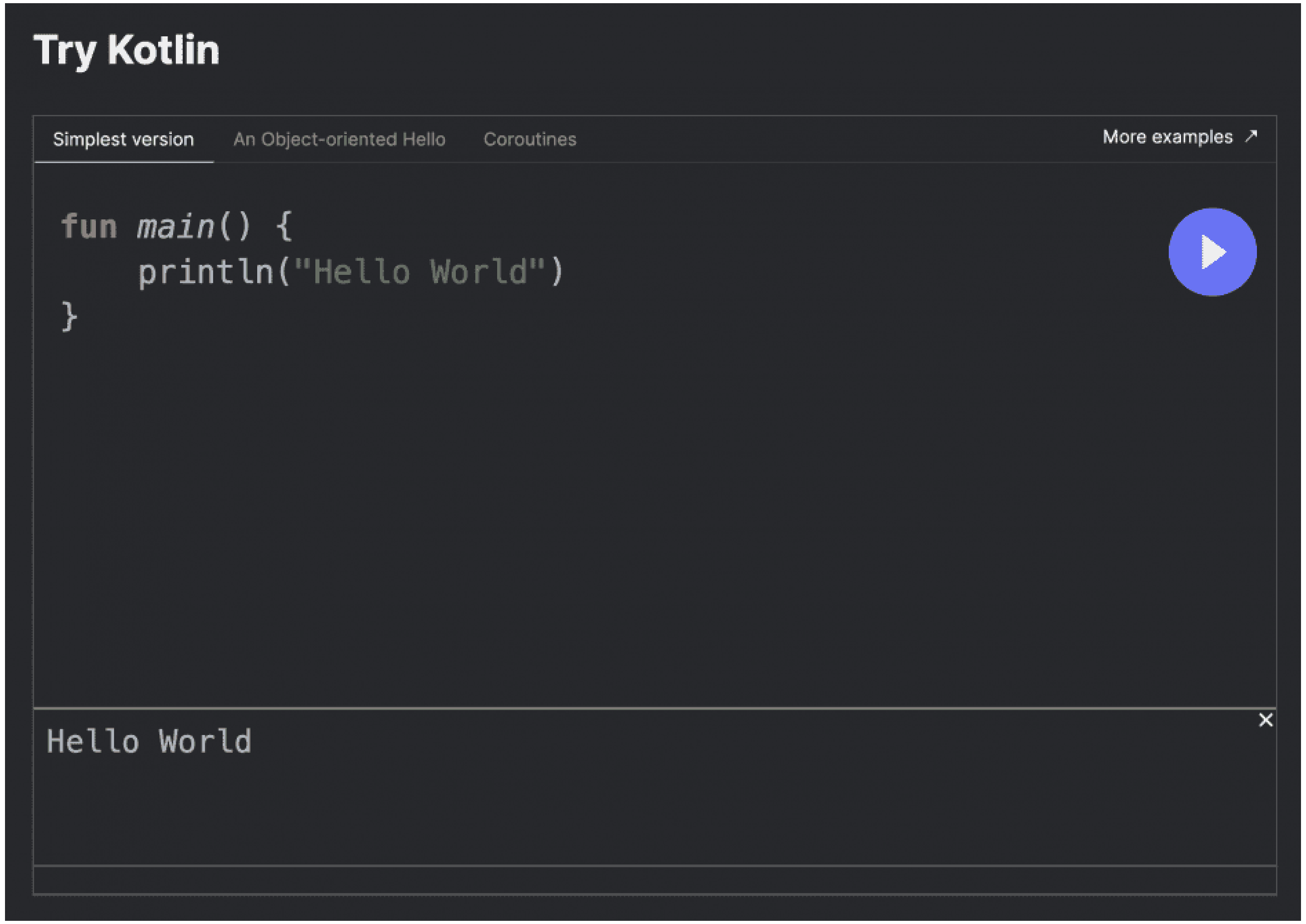
A modern programming language that makes developers happier. Open source forever

Good for

Latest from Kotlin

Kotlin 1.4-M1: Powerful type inference algorithm, SAM conversions for classes, New Kotlin/JS backend and more

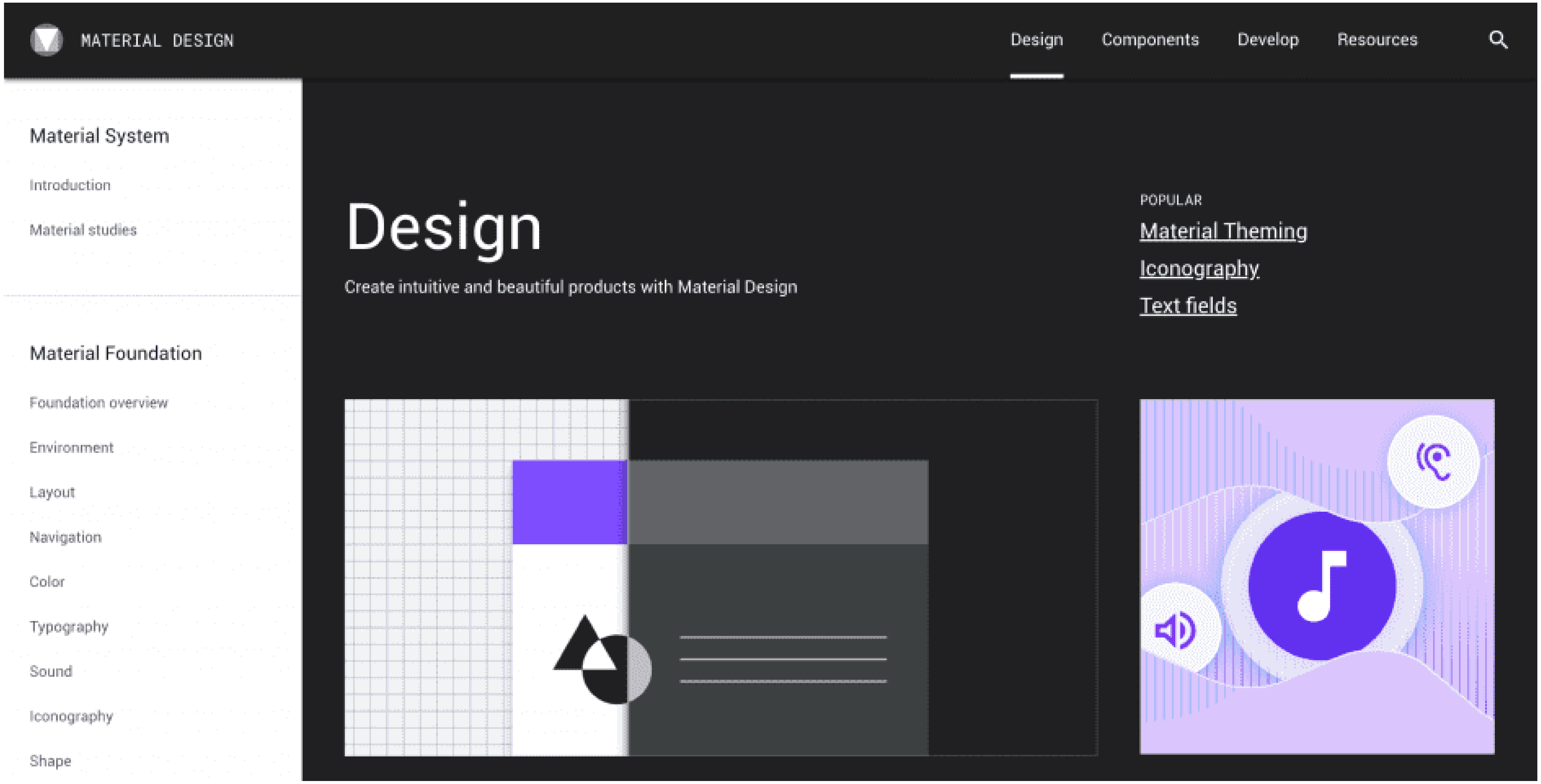
Google officially supports Kotlin on Android as a “first-class” language since May 2017. Fewer companies are developing commercial applications using only Java, so studying Kotlin is a necessary step to become a successful Android Developer.



```kotlin
fun main() {
    println("Hello World")
}
```

```
Hello World
```

3. Get to know Material Design



Material System

Introduction

Material studies

Material Foundation

Foundation overview

Environment

Layout

Navigation

Color

Typography

Sound

Iconography

Shape

Design

Create intuitive and beautiful products with Material Design

POPULAR

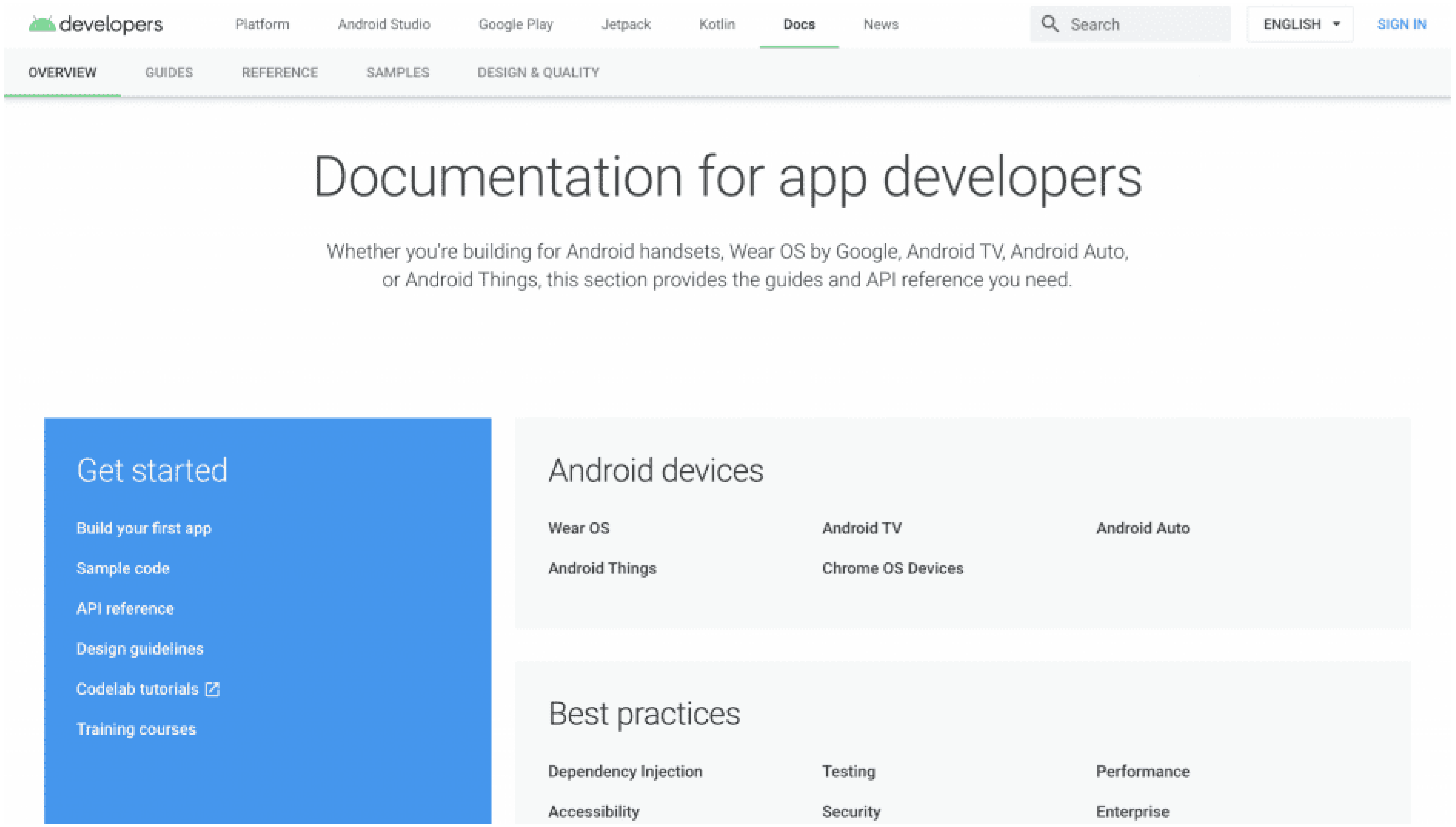
Material Theming

Iconography

Text fields

4. Download Android Studio IDE

5. Write some code



Documentation for app developers

Whether you're building for Android handsets, Wear OS by Google, Android TV, Android Auto, or Android Things, this section provides the guides and API reference you need.

Get started

- Build your first app
- Sample code
- API reference
- Design guidelines
- Codelab tutorials
- Training courses

Android devices

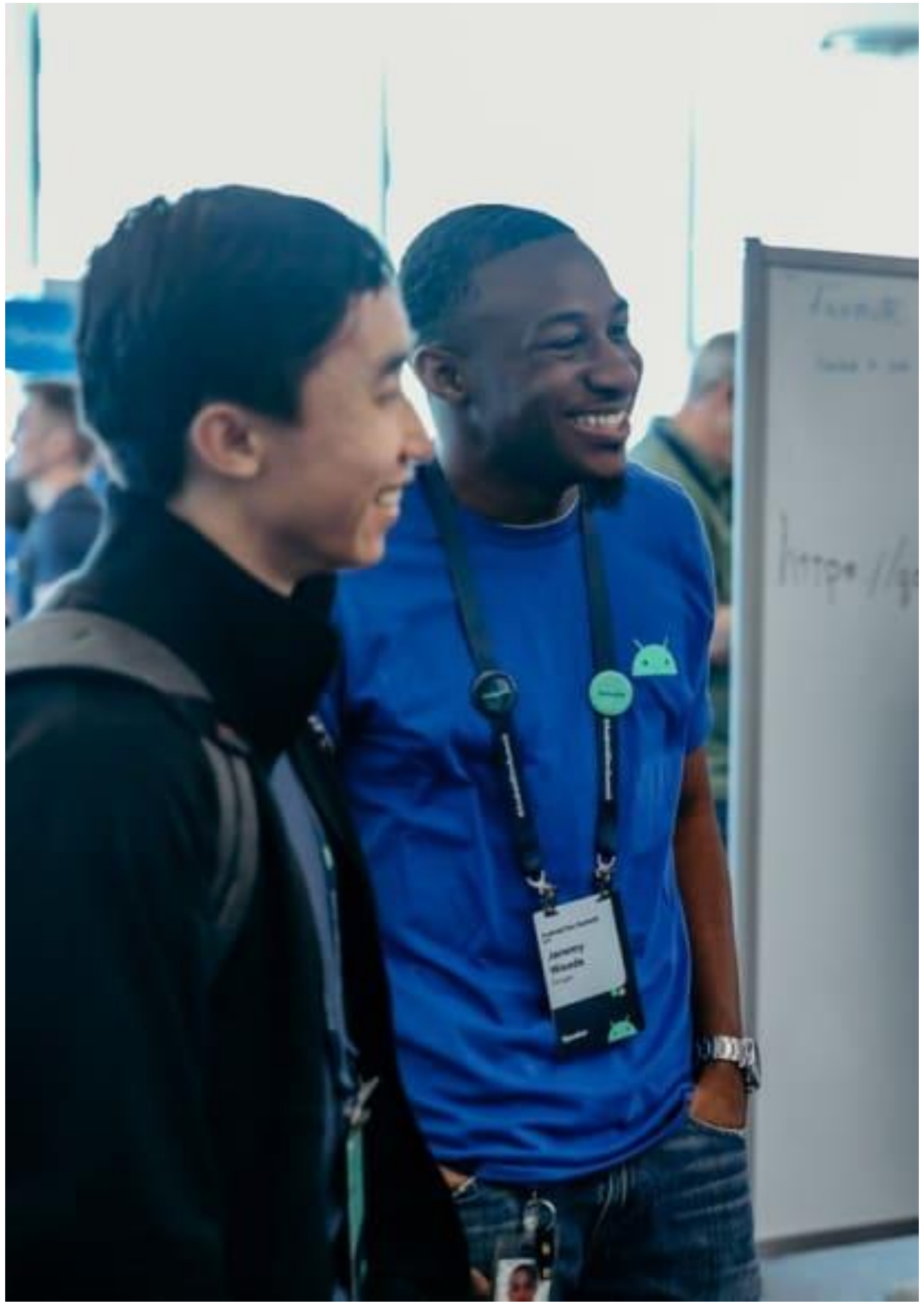
- Wear OS
- Android TV
- Android Auto
- Android Things
- Chrome OS Devices

Best practices

- Dependency Injection
- Testing
- Performance
- Accessibility
- Security
- Enterprise

6. Stay up to date

10 Twitter Follows for Android Developers

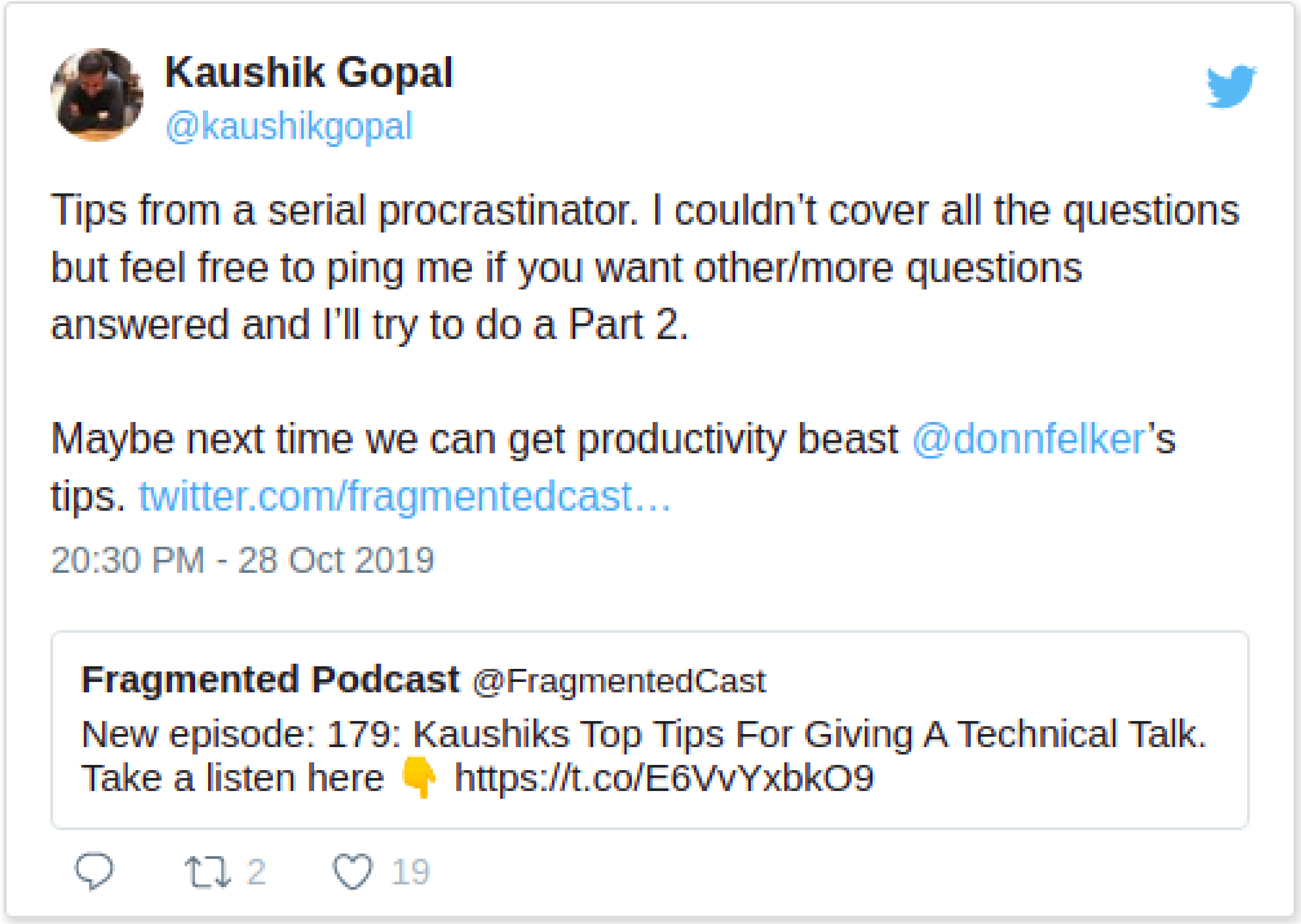


Android Developers (@AndroidDev)

•The official Twitter account of the Android Developer Team!
•Follow this account for news and announcements as well as general Android team goodness!

Adam McNeilly
(@AdamMc331)

- Senior Android Developer at @okcupid
- adammcneilly.com


Kaushik Gopal
@kaushikgopal

Tips from a serial procrastinator. I couldn't cover all the questions but feel free to ping me if you want other/more questions answered and I'll try to do a Part 2.

Maybe next time we can get productivity beast @donnfelker's tips. twitter.com/fragmentedcast…

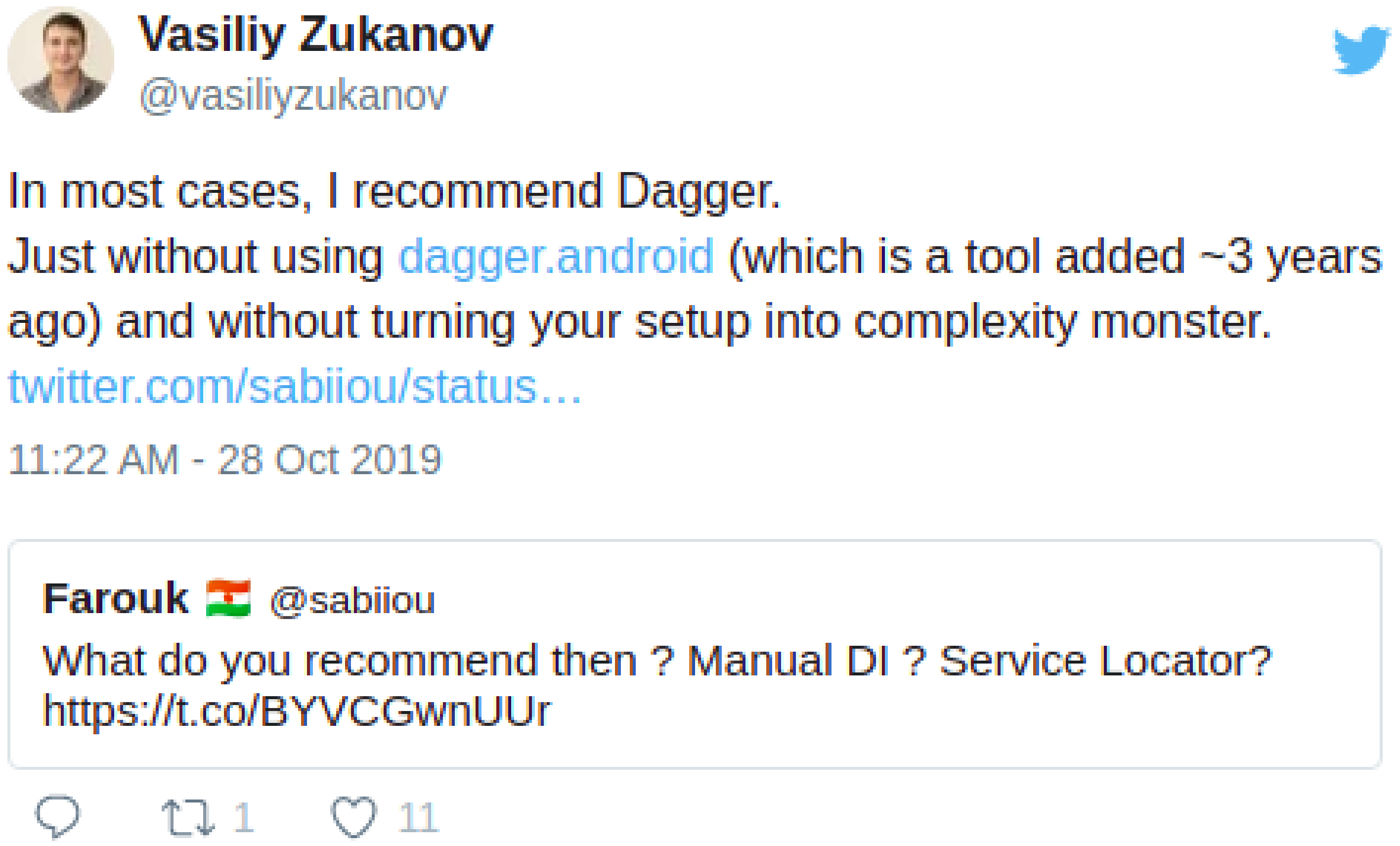
20:30 PM - 28 Oct 2019

Fragmented Podcast @FragmentedCast
New episode: 179: Kaushiks Top Tips For Giving A Technical Talk. Take a listen here 👇 https://t.co/E6VvYxbkO9

2 19

Kaushik Gopal (@kaushikgopal)

- Google Developer Expert
- 1/2 of the Fragmented Podcast



Vasiliy Zukanov
@vasiliyzukanov

In most cases, I recommend Dagger.
Just without using dagger.android (which is a tool added ~3 years ago) and without turning your setup into complexity monster.
twitter.com/sabiiou/status…

11:22 AM - 28 Oct 2019

Farouk 🇳🇪 @sabiiou
What do you recommend then ? Manual DI ? Service Locator?
https://t.co/BYVCGwnUUr

1 11

Vasiliy Zukanov
(@VasiliyZukanov)

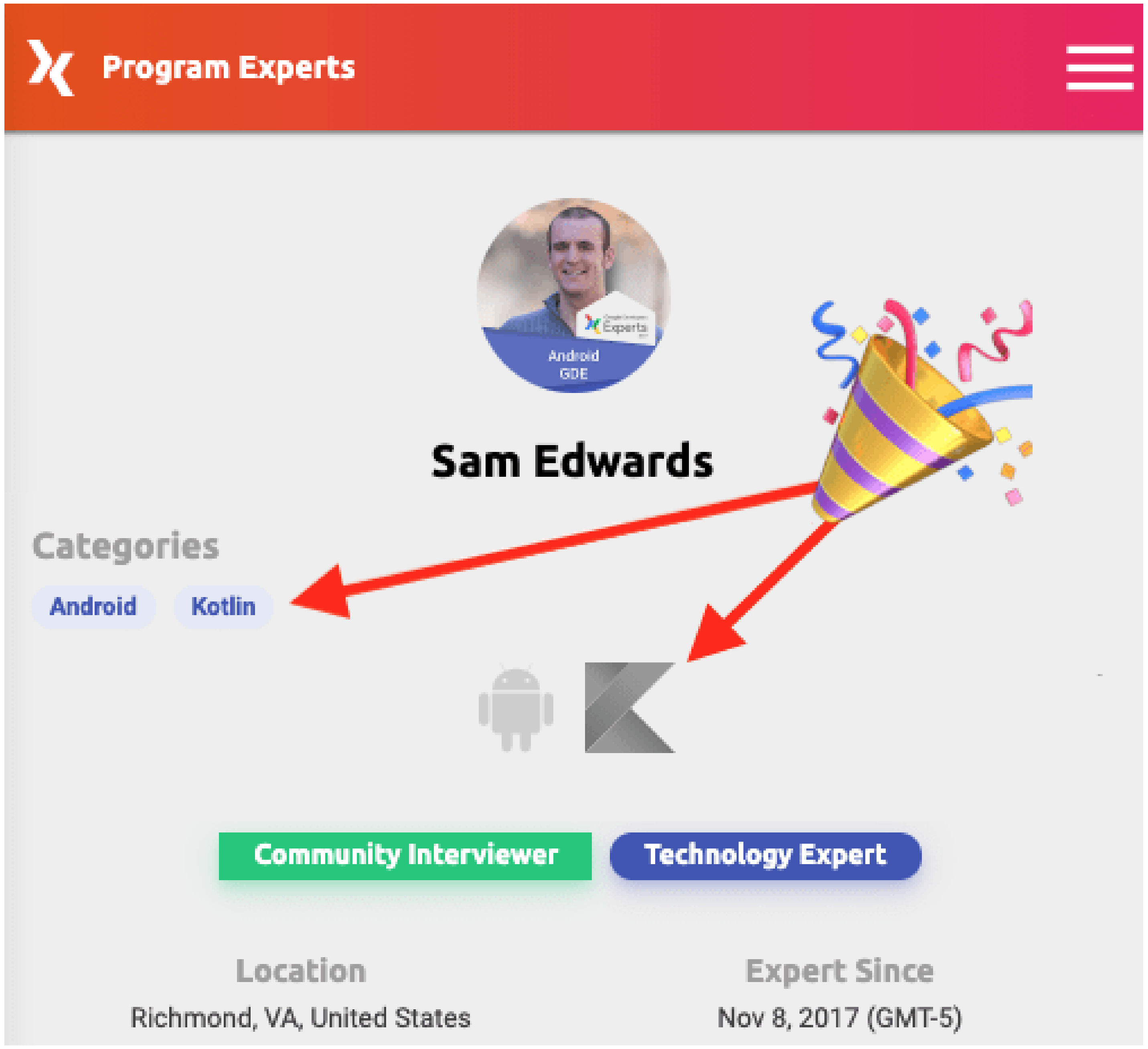
- Android Developer
- techyourchance.com

Dave Burke (@davey_burke)

- Android VP Engineering

Ben Weiss
(@keyboardsurfer)

Android Developer Relations Engineer


Program Experts

Android GDE

Sam Edwards

Categories

Android Kotlin

Community Interviewer Technology Expert

Location

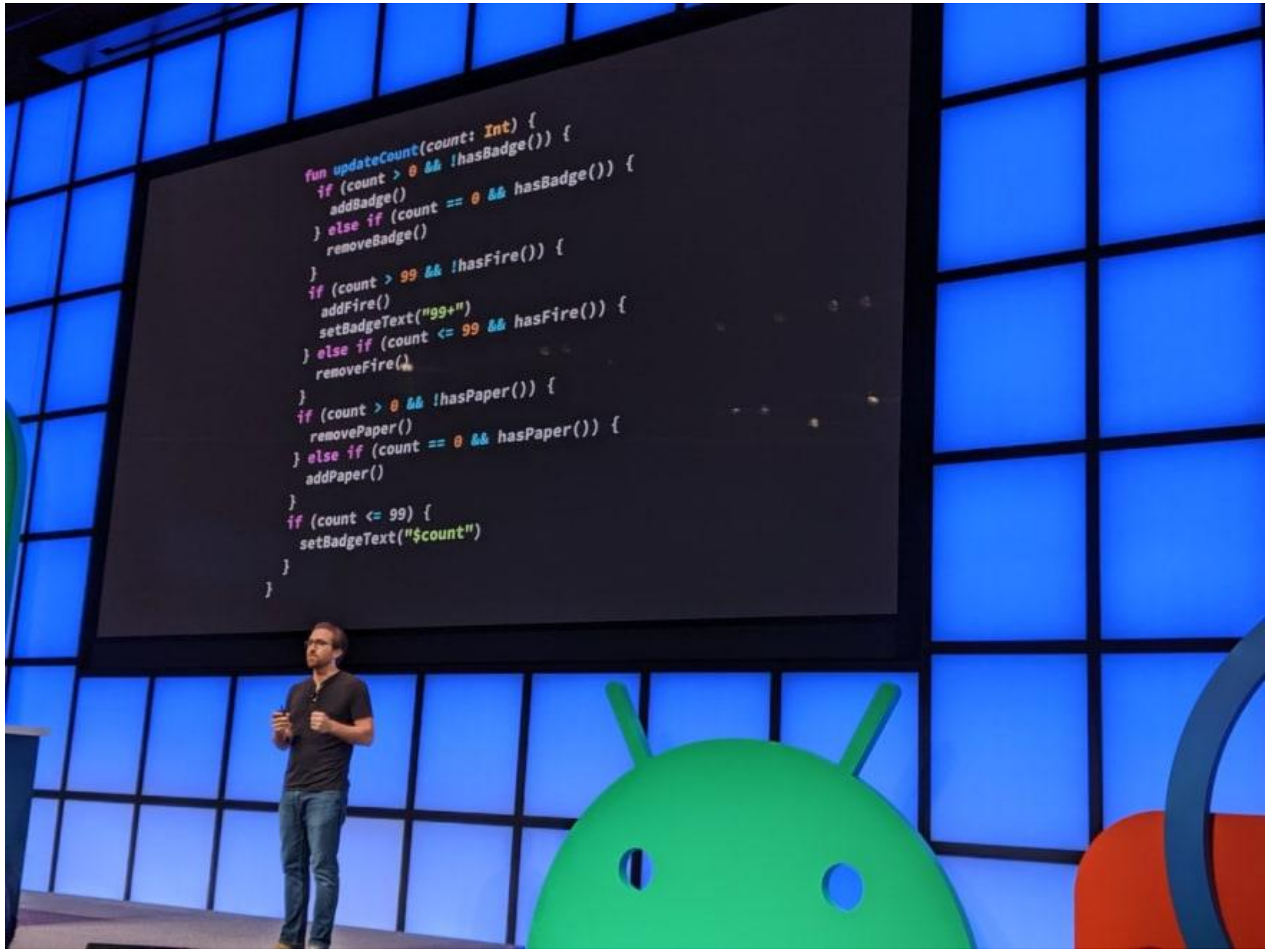
Richmond, VA, United States

Expert Since

Nov 8, 2017 (GMT-5)

Sam Edwards (@HandstandSam)

- Google Developer Expert
- Instructor at Caster.io



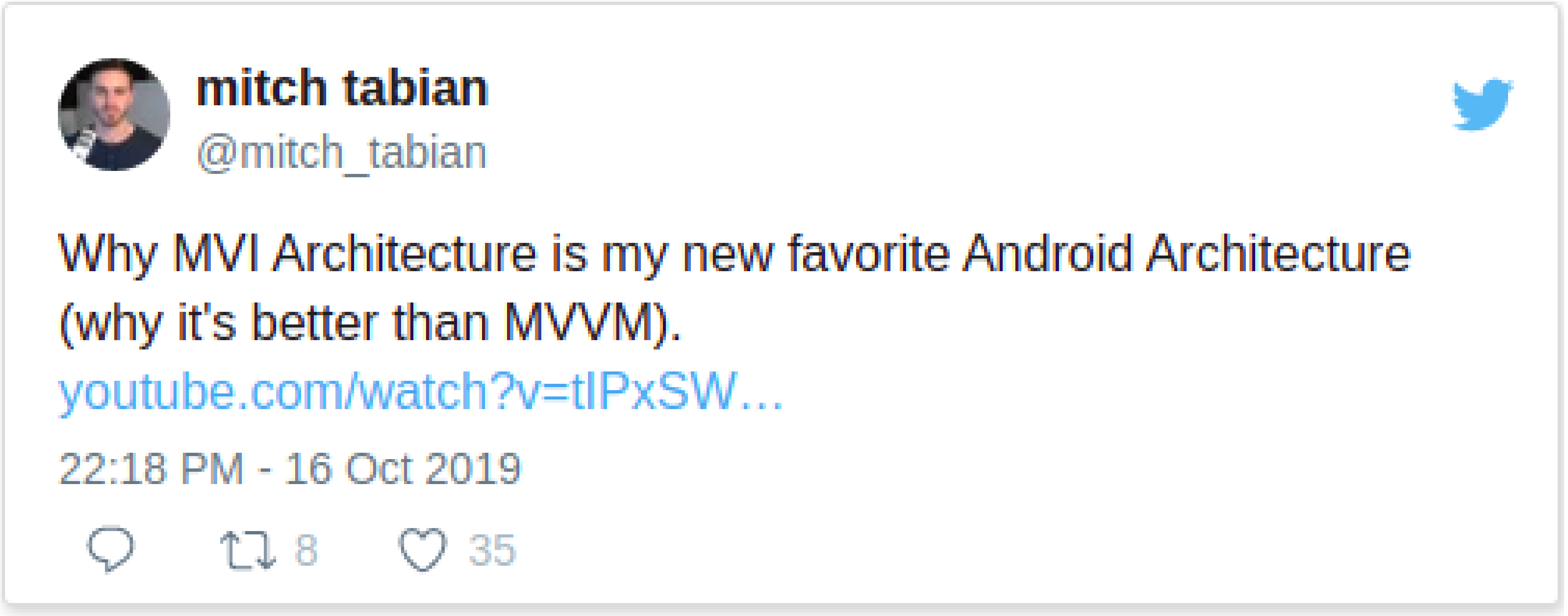
```
fun updateCount(count: Int) {
    if (count > 0 && !hasBadge()) {
        addBadge()
    } else if (count == 0 && hasBadge()) {
        removeBadge()
    }
    if (count > 99 && !hasFire()) {
        addFire()
        setBadgeText("99+")
    } else if (count <= 99 && hasFire()) {
        removeFire()
    }
    if (count > 0 && !hasPaper()) {
        removePaper()
    } else if (count == 0 && hasPaper()) {
        addPaper()
    }
    if (count <= 99) {
        setBadgeText("$count")
    }
}
```

Annyce Davis (@brwngrldev)

- Engineering Manager at @Meetup
- adavis.info

Mitch Tabian (@mitch_tabian)

- Android Developer
- YouTuber



mitch tabian

@mitch_tabian

Why MVI Architecture is my new favorite Android Architecture (why it's better than MVVM). youtube.com/watch?v=tIPxSW…

22:18 PM - 16 Oct 2019

8 35

Ryan Harter (@rharter)

- Google Developer Expert
- Android lead at @pixiteapps