DEVCON

RWDevCon 2018 Vault

By the raywenderlich.com Tutorial Team

Notice of Rights

Notice of Liability

Trademarks

Table of Contents: Overview

Table of Contents: Extended

Prerequisites

Some tutorials and workshops at RWDevCon 2018 have prerequisites.

If you plan on attending any of these tutorials or workshops, **be sure to complete the steps below before attending the tutorial**.

If you don't, you might not be able to follow along with those tutorials.

> **Note**: All talks require **Xcode 9.2** installed, unless specified otherwise (such as in the ARKit tutorial and workshop).

7: Android for iOS Developers

This session is directed at iOS developers who want to get started with Android development.

Install Android Studio 3.1 by walking through this tutorial:

- https://www.raywenderlich.com/177533/beginning-android-development-kotlin-part-one-installing-android-studio

Before You Arrive

Wifi at a hotel full of developers trying to run multiple devices each is notoriously bad. Here are some things you should do before you arrive to save yourself some waiting time at the conference:

- Use the SDK manager to ensure you have the Android 5.0 Lollipop (API 21) SDK installed, as well as the latest Android SDK (Oreo 8.1, API 27 as of this writing).
- Use the Android Virtual Device manager to install an emulator which can run API 27.

- Open Android Studio and import the Demo 1 starter project, go to **File/New/Import Project...** and select **Demo1\starter\PropertyFinder**.
- Make sure **Instant Run** is configured with automatic activity restart disabled. Go to **Android Studio/Preferences** on Mac OSX or **File/Settings** on Windows. Navigate to **Build, Execution, Deployment/Instant Run**. Make sure **Enable Instant Run...** is checked and **Restart activity on code changes** is unchecked.
- Build the project and run it on your emulator.

This will ensure you have versions of all the appropriate dependencies cached on your computer.

7: Android for iOS Developers

Learn the fundamentals of Android development through this tutorial. You'll build an app from scratch that walks you through Android layout, resources, list views, navigation, networking and material design. Along the way, I'll compare and contrast the concepts with those found in building iOS apps.

19 Android for iOS Developers: Demo 1

By Christine Abernathy

In this demo, you will start building your app from scratch. You'll start off working with an Android `Activity` and learn how to design the UI using the layout editor and Android resources. You'll be introduced to the `Activity` lifecycle. Finally, you'll see an example of event handling via a `Button` click handler implementation.

The assumption is you followed the **Android for iOS** session pre-requisite steps. Therefore, you should have Android Studio 3.0.1 installed and the starter project running on an Android emulator.

The steps here will be explained in the demo, but here are the raw steps in case you miss a step or get stuck.

> **Note**: Begin work with the starter project in **Demo1\starter**.

1) Modify Welcome TextView

In **res/layout/activity_main.xml**, switch to the code editor.

In the `TextView`, modify the `android:text` string value to:

```
Search for houses to buy!
```

Remove the following attribute:

```
app:layout_constraintBottom_toBottomOf="parent"
```

Add the following attribute:

```
android:padding="10dp"
```

Run the app.

2) Add Instruction TextView

Add the following attribute to the welcome `TextView`:

```
android:id="@+id/welcomeLabel"
```

Add the following below the welcome `TextView`:

```
<TextView
  android:id="@+id/instructionLabel"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Search by place-name or zip code"
  android:padding="10dp"
  app:layout_constraintLeft_toLeftOf="parent"
  app:layout_constraintRight_toRightOf="parent"
  app:layout_constraintTop_toBottomOf="@+id/welcomeLabel" />
```

Click **Apply Changes**.

3) Remove Hardcoded Resources

In **res/values/strings.xml**, add the following:

```
<string name="welcome">Search for houses to buy!</string>
<string name="instruction">Search by place-name or zip code</string>
```

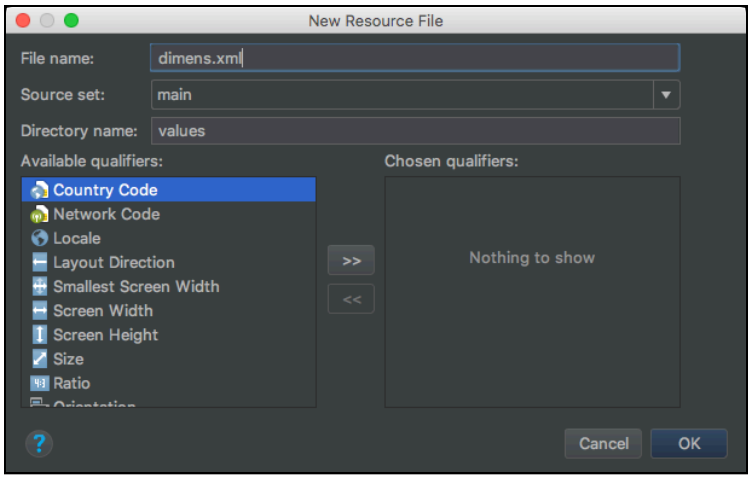
In **res/layout/activity_main.xml** change the `android:text` attribute for the welcome `TextView` to:

```
android:text="@string/welcome"
```

Then, change the `android:text` attribute for the instruction `TextView` to:

```
android:text="@string/instruction"
```

In **res/values** create a new `Values resource file` named **dimens.xml**:



Then, add the following resource to the new file:

```
<dimen name="default_padding">10dp</dimen>
```

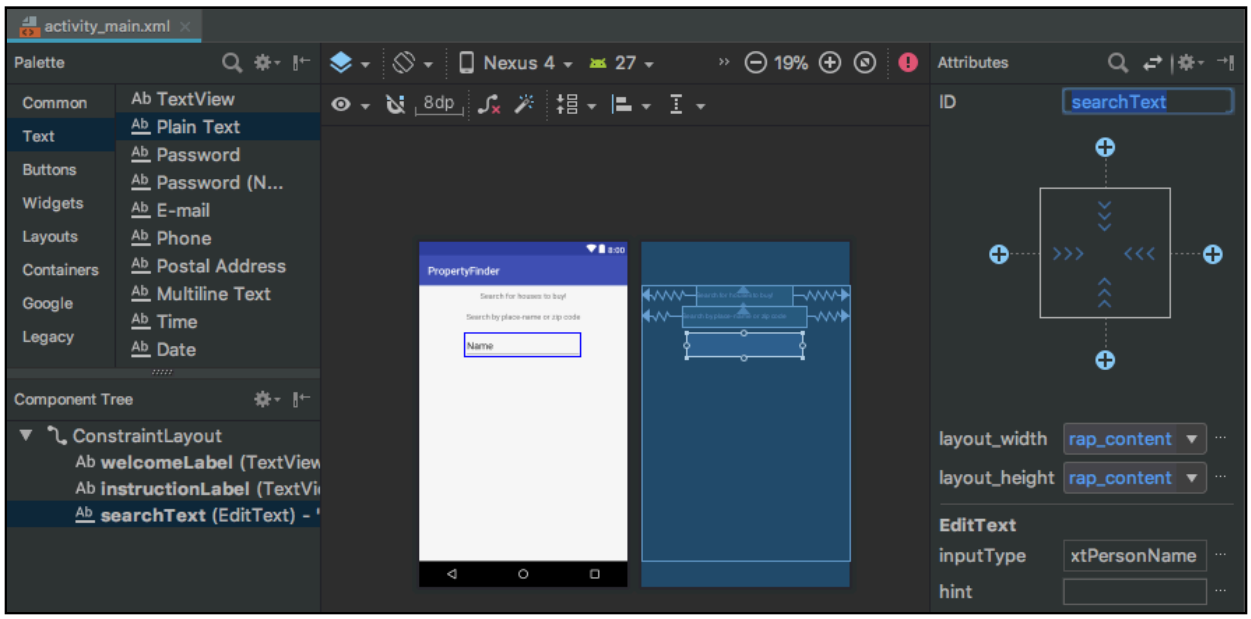
In **res/layout/activity_main.xml**, replace the value for the `android:padding` attribute in both `TextView` elements from `10dp` to `@dimen/default_padding`.

Click **Apply Changes**.

4) Add the Search Field

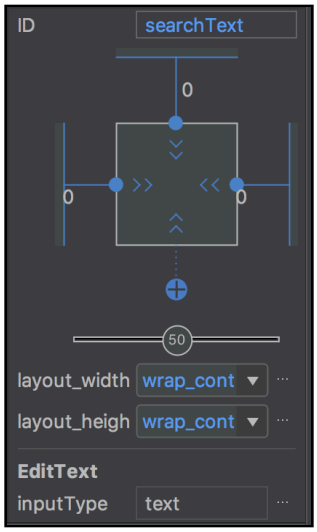
In **res/layout/activity_main.xml**, switch to the design editor.

Drag a `Plain Text` UI widget and place it under the second `TextView`. Then change the `ID` attribute to `searchText`:



In the Attributes pane, tap on **EditText\inputType**, then uncheck `textPersonName` and check `text`. Empty out the content under **TextView\text**.

Tap on the Constraints box and make connections from the left edge of the widget to the left edge of the parent and set the margin to 0. Make a connection from the right edge of the widget to the right end of the parent and set the margin to 0. Finally, make a connection from the top of the widget to the bottom of the nearest widget (the instruction text view) and set the margin to 0. Your connections should look like this:



Switch back to layout code editor. Add the following attributes to the `EditText` widget:

```
android:maxLines="1"
android:padding="@dimen/default_padding"
```

Click **Apply Changes**.

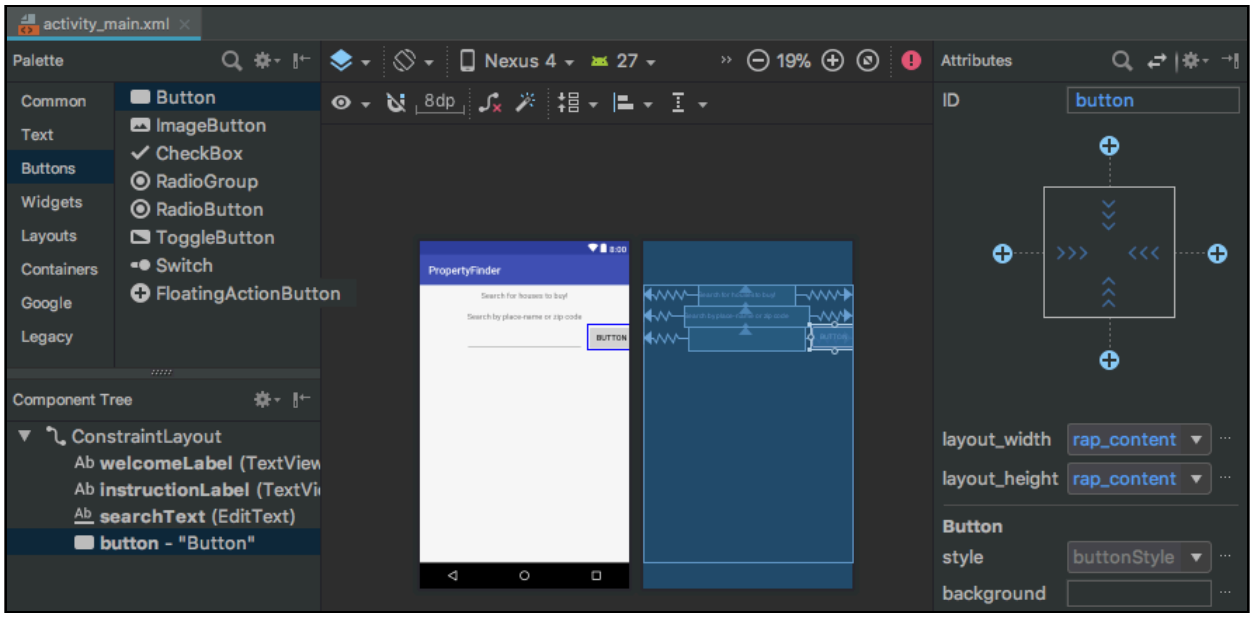
5) Add the Go button

In **res/values/strings.xml**, add the following:

```
<string name="button_go">Go</string>
```

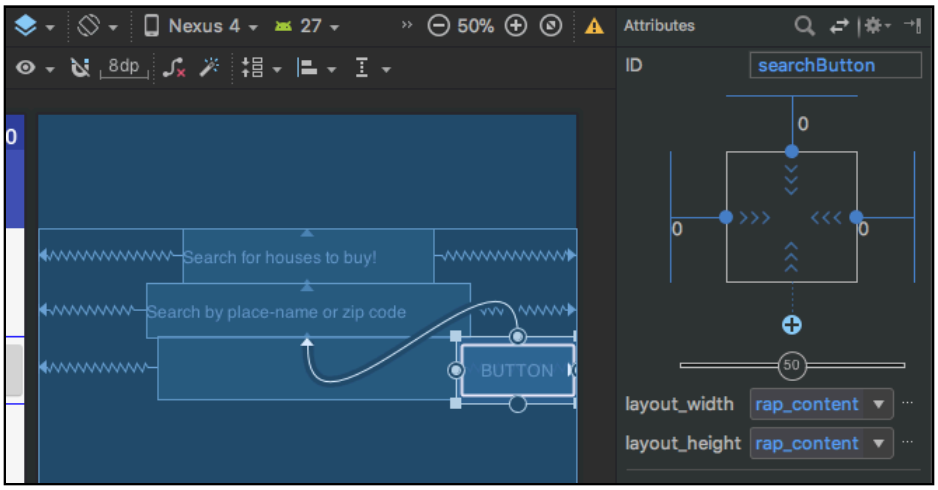
In **res/layout/activity_main.xml**, switch to the design editor.

Drag a `Button` Widget and place it to the right of the search text field:

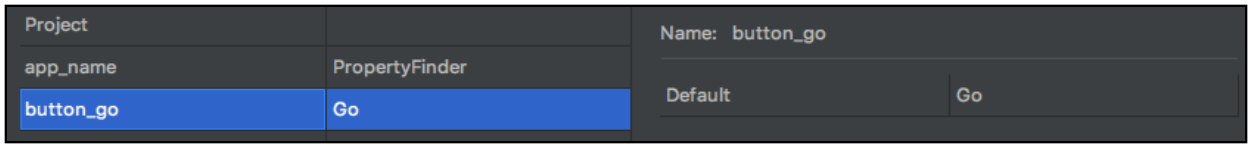


In the Attributes pane, change the `ID` attribute to `searchButton`.

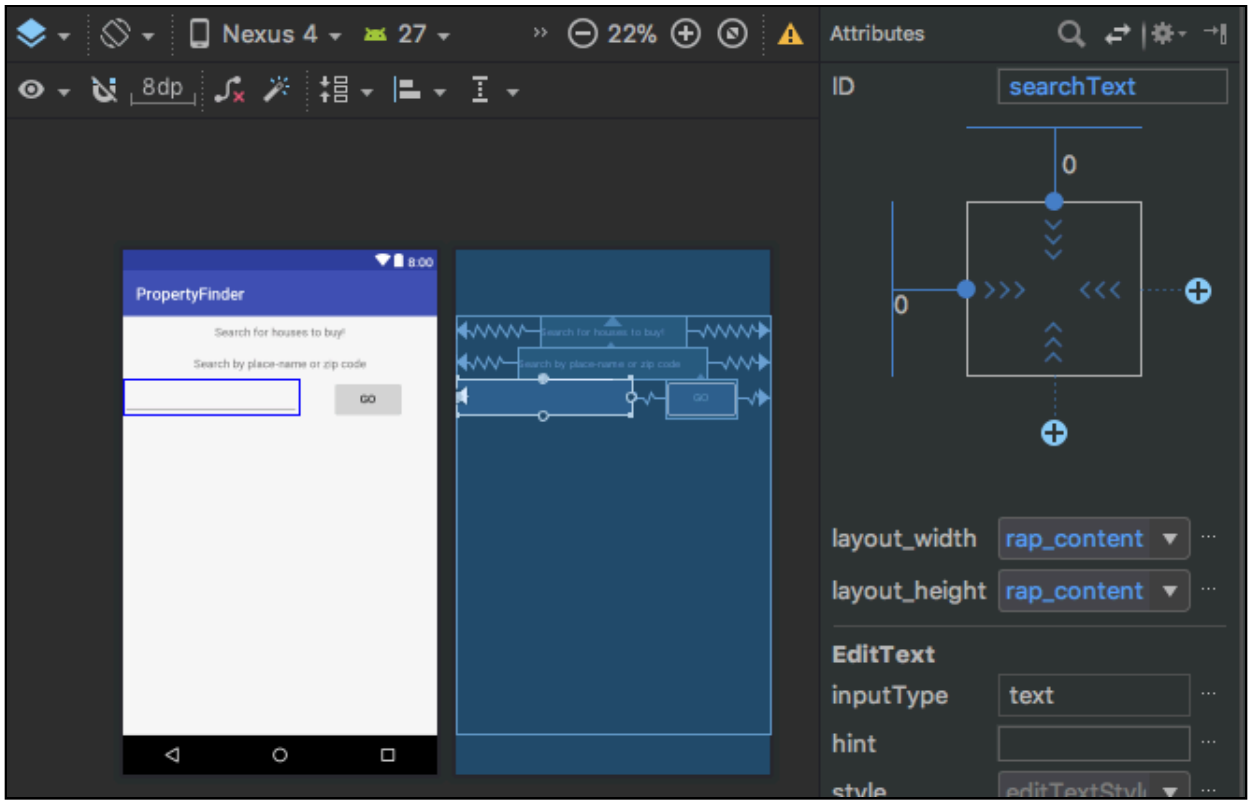
Tap on the Constraints box and make connections to the right, left, and top edges to the nearest neighbors with a margin of 0. Make sure the nearest top neighbor is `instructionLabel`. You can zoom in to see better:



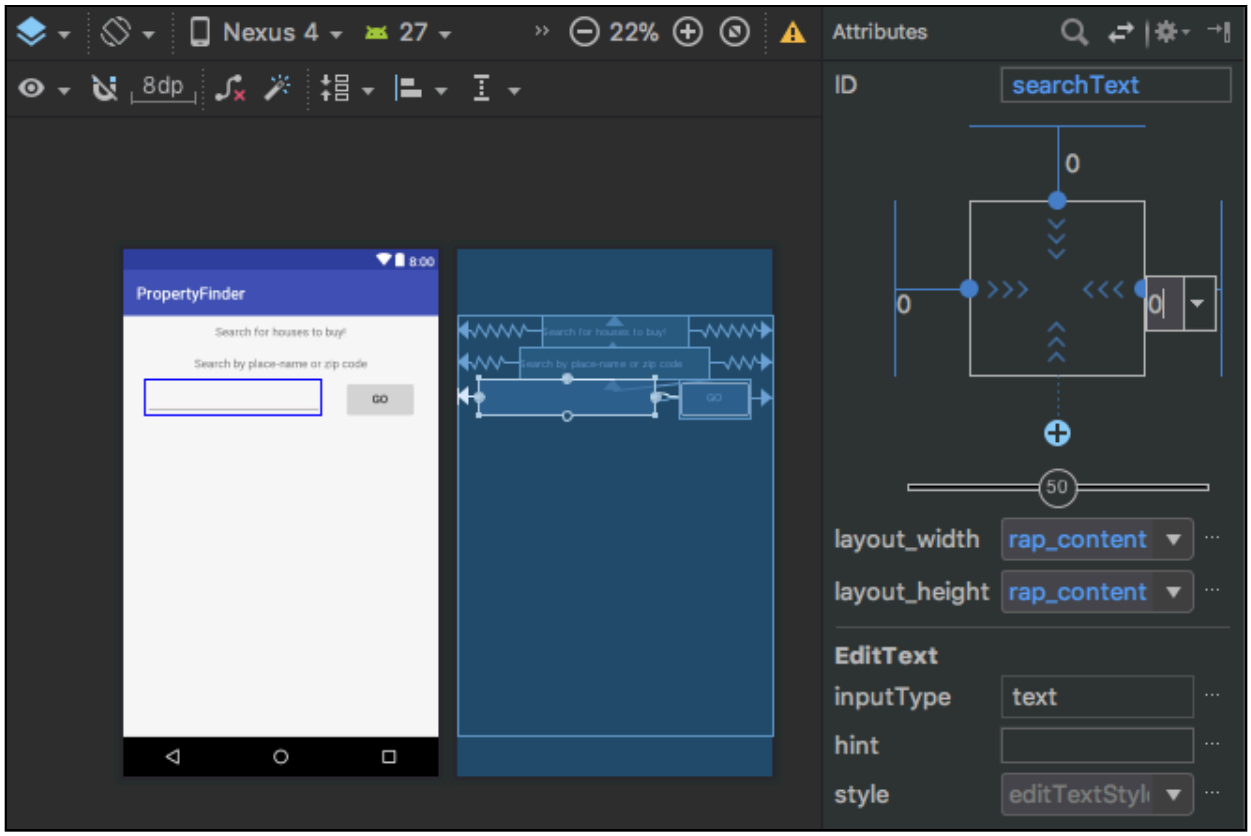
Tap on **TextView\text**. In the pop-up window select `button_go`:



Select `searchText` from the Component Tree pane. In the Constraints box, disconnect the right edge:



Then reconnect the right edge with a margin of 0:



Then change the `layout_width` attribute to `match_constraint` which is the same as setting it to 0dp.

Switch back to layout text editor. Replace the following attribute in `EditText`:

Add this attribute to the top-most element, `android.support.constraint.ConstraintLayout`:

```
android:padding="@dimen/default_padding"
```

In the instruction `TextView`, add the following attribute:

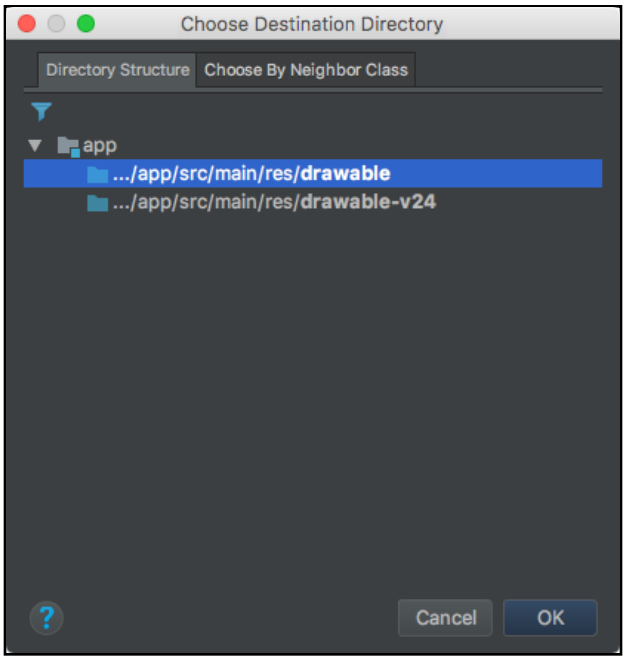
```
android:labelFor="@+id/searchText"
```

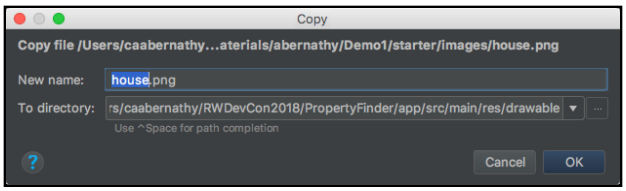
Click **Apply Changes**.

6) Add the Property image

Go to **Demo1\starter\images** and copy **house.png** to the clipboard (Cmd+C on Mac, Ctrl+C on Windows).

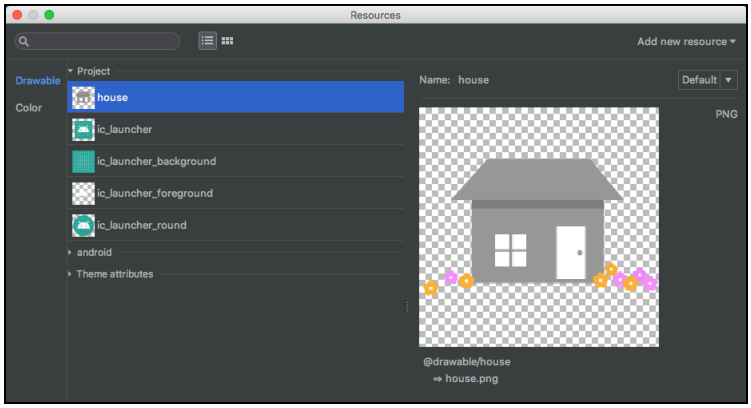
In Android Studio, navigate to **res/drawable**, highlight the folder and paste the image. If prompted to select a directory, select **../app/src/main/res/drawable**:



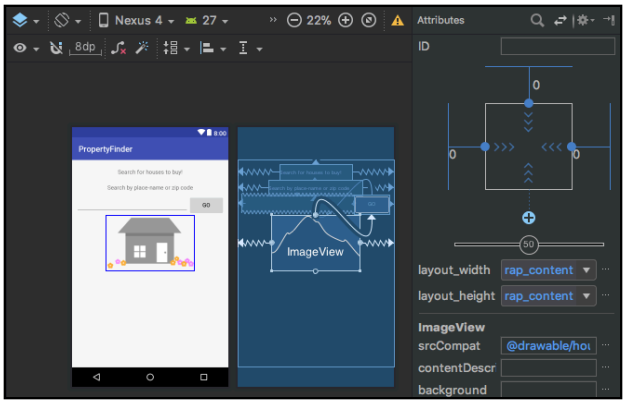


In **res/layout/activity_main.xml**, switch to the design editor.

Drag an `ImageView` widget below the search text and towards the middle. In the Resource popup, select the **Drawable\Project\house**:



In the Constraints box for the new image, pin the left, right, and top edges to the nearest neighbor with a margin of 0:



In **res/values/strings.xml**, add the following:

```
<string name="description_house">House</string>
```

In **res/layout/activity_main.xml**, switch to the layout code editor and add the following attribute to the `ImageView` widget:

```
android:contentDescription="@string/description_house"
android:padding="@dimen/default_padding"
```

Click **Apply Changes**.

7) Add the button click handler

In **MainActivity.kt**, add the following to the end of `onCreate()`:

```
val searchButton = findViewById<Button>(R.id.searchButton)
val searchText = findViewById<TextView>(R.id.searchText)
searchButton.setOnClickListener {
  Log.d("PropertyFinder", searchText.text.toString())
}
```

Click **Apply Changes**.

8) Set the default query

In **res/values/strings.xml**, add the following:

```
<string name="default_place">london</string>
```

In **MainActivity.kt**, add the following just after the `searchText` declaration:

```
searchText.setText(R.string.default_place)
```

Click **Apply Changes**.

9) That's it!

Congrats, at this time you should have a good understanding of the basics of Android and setting up the UI! It's time to build up **PropertyFinder** by adding navigation and list view features.

20 Android for iOS Developers: Demo 2

By Christine Abernathy

In this demo, you'll learn about navigation and working with lists. You'll modify your app from Demo 1 to display hard-coded results when the user initiates a search. You'll learn about using an `Intent` for handling navigation and `RecyclerView` and `ViewHolder` for displaying list data.

In Android Studio, go to **File/New/Import Project...** and select **Demo2\starter\PropertyFinder**. Then, build and run the app.

1) Navigate to the results view

In **MainActivity.kt**, add the following method to the end of the class:

```
undefined
```

Add the following just after the `Log.d()` statement in the button's click event handler:

```
undefined
```

Run the app.

2) Pass data to the results view

In **MainActivity.kt**, add the following variable at the top:

```
undefined
```

Add the following method to the end of the class:

```
undefined
```

In the button event handler, swap out the `displayResults()` call with the following:

```
undefined
```

Add the following just before the intent is started in `displayResults()`:

```
undefined
```

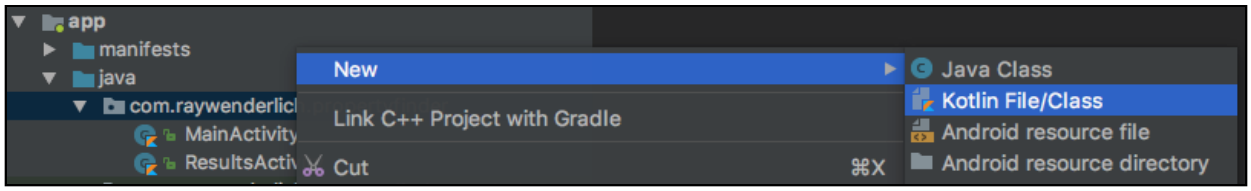
In **ResultsActivity.kt**, add the following to the end of `onCreate()`:

```
undefined
```

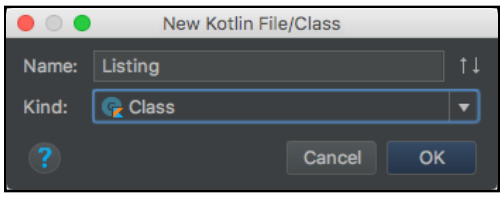
Click **Apply Changes**.

3) Parse JSON string into an object

Right-click on the folder containing your `Activity` files and tap **New\Kotlin File/Class**:



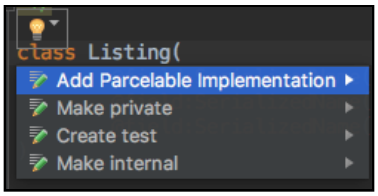
Name the file **Listing** and select **Class** for Kind, then tap **OK**:



In **Listing.kt**, replace the class implementation with the following:

```
undefined
```

Click on `Listing` inside the class file and in the suggestion icon that pops up, press **Add Parcelable Implementation**:



In **MainActivity.kt**, replace the `mResults` assignment inside `loadLocalResults()` with the following:

```
undefined
```

Run the app.

4) Add RecyclerView

In **ResultsActivity.kt**, remove the following lines:

```
undefined
```

In **res/layout/activity_results.xml**, switch to layout code editor. Remove the `TextView` widget and replace it with the following `RecyclerView` widget:

```
<android.support.v7.widget.RecyclerView
  android:id="@+id/listingsView"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:scrollbars="vertical" />
```

In **PropertyListAdapter.kt**, modify the class declaration to pass in the listings:

```
undefined
```

In `ListingHolder`, uncomment the member declarations lines:

```
undefined
```

In `onCreateViewHolder()`, uncomment the layout inflation code:

```
undefined
```

Add the following to the end of `onCreateViewHolder()`:

```
undefined
```

Implement `onBindViewHolder()` by adding the following:

```
undefined
```

Replace `getItemCount()` with the following:

```
undefined
```

In **MainActivity.kt**, inside `loadLocalResults()` delete the `mResults` assignment.

Replace the `mResults` variable declaration at the top with the following:

```
undefined
```

In `loadLocalResults()`, swap out the `var listings` assignment as follows:

```
undefined
```

In `displayResults()`, replace the `intent.putExtra` line with the following:

```
undefined
```

In **ResultsActivity.kt**, add the following to the end of `onCreate()`:

```
undefined
```

Click **Apply Changes**.

5) Display images

In **MyApplication.kt**, add the following method to the class:

```
undefined
```

In **res/layout/item_listing.xml**, add the following as the first child element in `LinearLayout`:

```
<com.facebook.drawee.view.SimpleDraweeView
  android:id="@+id/propertyImage"
  android:layout_width="match_parent"
  android:layout_height="200dp"/>
```

In **PropertyListAdapter.kt**, add the following to the end of `ListingHolder` initialization:

```
undefined
```

Add the following to the end of `onBindViewHolder()`:

```
undefined
```

Click **Apply Changes**.

6) That's it!

Congrats, at this time you should have a good understanding of the how to navigate in Android using intents and how to display lists using recycler views and adapters. You even got a taste of using Fresco to load your images.

It's time to up the ante for **PropertyFinder** by using real data and making the results pretty through Material Design.

21 Android for iOS Developers: Demo 3

By Christine Abernathy

In this demo, you'll learn you can add networking and make use of Material Design.

You'll use `CardView` widget to showcase Material Design in displaying list data. You'll make the contents appear like they're floating above the container and rearrange some of the views to make the view look prettier.

Instead of working with hard-coded data, you'll use the **Retrofit** open source library to query the property listing API and parse the data. You'll also add a progress bar create a better user experience while the API data is loading.

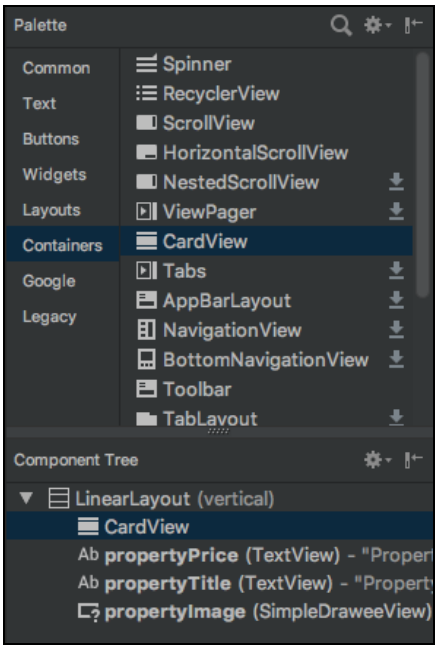
In Android Studio, go to **File/New/Import Project...** and select **Demo3\starter\PropertyFinder**. Then, build and run the app.

The steps here will be explained in the demo, but here are the raw steps in case you miss a step or get stuck.

1) Add CardView

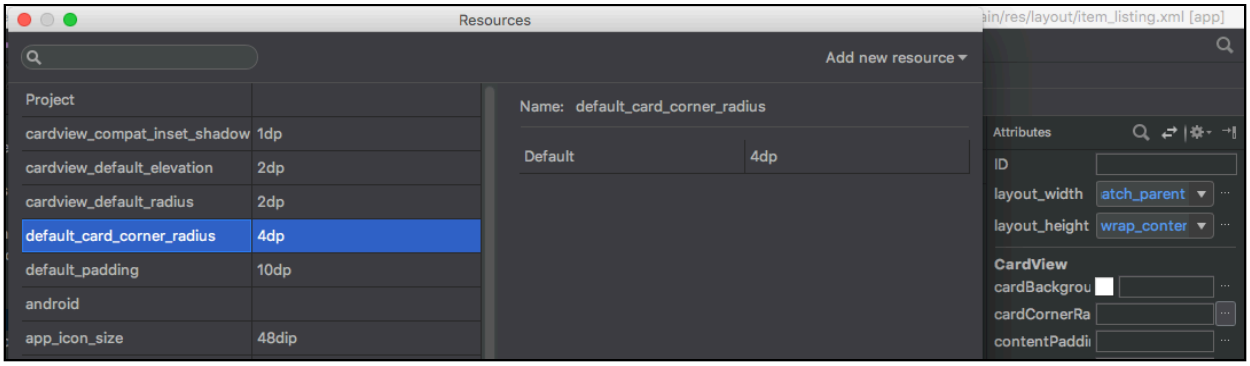
In **res/layout/item_listing.xml**, switch to the design editor.

From the Palette, drag a `CardView` and make it the first element of the layout:



In the Attributes pane, change `layout_height` to `wrap_content`.

In the Attributes pane, edit `cardCornerRadius` and select `default_card_corder_radius`:



Switch to the code editor.

Move the `CardView` so it becomes the parent element of `LinearLayout`. You'll need to change the closing `CardView` tag `/>` to `>`.

Add the following tag to the bottom:

```
</android.support.v7.widget.CardView>
```

Cut the following attributes from `LinearLayout`:

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

Then paste them to the new parent `CardView`.

Add the following attributes to `CardView`:

```
android:layout_gravity="center"
android:layout_margin="8dp"
```

Add the following attribute to the `TextView` representing the property price:

```
android:textStyle="bold"
```

Run the app.

2) Add the Nestoria API Service

In the module **build.gradle**, uncomment the following dependencies:

```
undefined
```

Right-click on the folder that contains your activities, select **New \ Kotlin File/ Class**, and name it **NestoriaResult** and select **Class** for Kind.

In **NestoriaResult.kt**, replace the existing `NestoriaResult` class implementations with the following:

```
undefined
```

Right-click on the folder that contains your activities, select **New \ Kotlin File/ Class**, and name it **NestoriaService** and select **Interface** for Kind.

In **NestoriaResult.kt**, add the following implementation to the interface:

```
undefined
```

If presented with an option of the `Call` library to use, select `retrofit2.Call`.

Add the following to the end of the interface:

```
undefined
```

In **MainActivity.kt**, add the following near the top of the class after the other variables are declared:

```
undefined
```

Add the following code to the end of `loadApiResults()`:

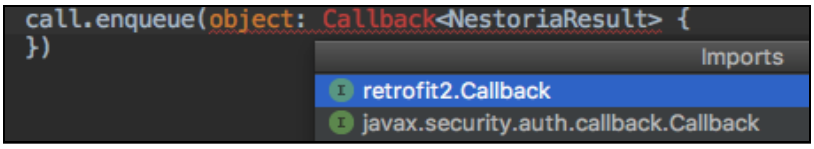
```
undefined
```

3) Display the API results

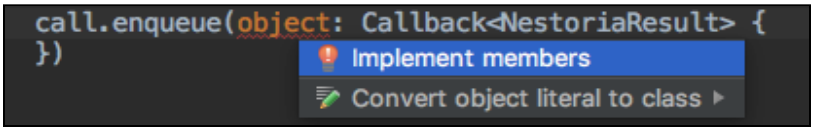
In **MainActivity.kt**, add the following to the end of `loadApiResults()`:

```
undefined
```

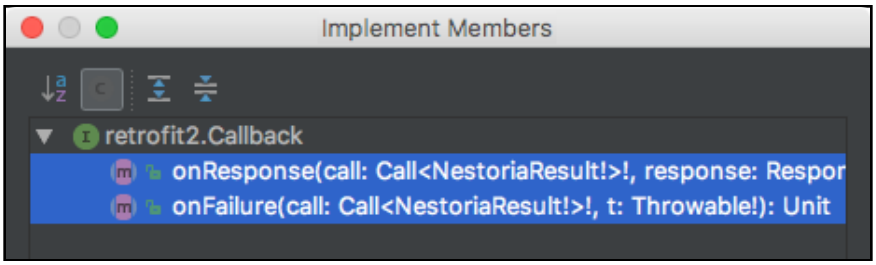
Import the `Callback` from the `retrofit2.Callback` namespace:



Click on the `object` parameter in `call.enque()` and press **Alt + Enter** on Windows/ Linus or **Option+Enter** on Mac:



Select **Implement members** and press Enter. In the resulting pop up, select all the members using the Shift key, then press **OK**:



Replace `TODO()` in `onResponse()` with the following:

```
undefined
```

Replace `TODO()` in `onFailure()` with the following:

```
undefined
```

In the button's click event handler, replace the call to `loadLocalResults()` with the following:

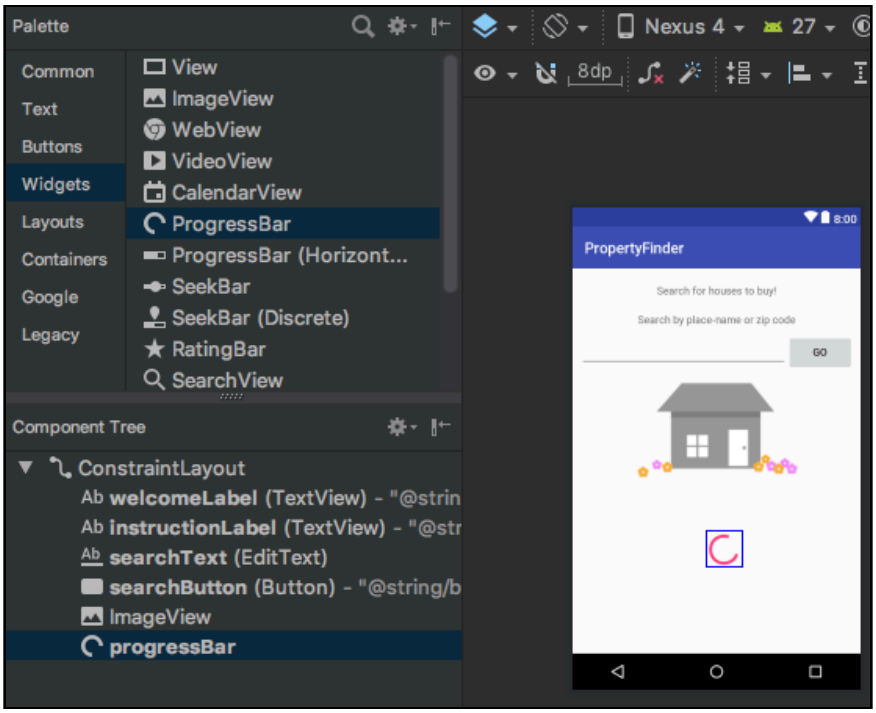
```
undefined
```

Click **Apply Changes**.

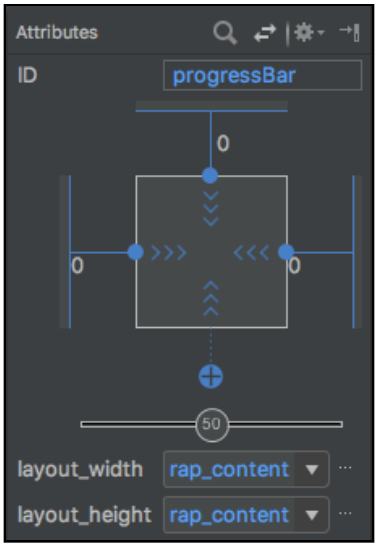
4) Add a Progress Bar

In **res/layout/activity_main.xml**, switch to the design editor.

From the Palette, drag a `ProgressBar` just below the house image:



Tap on the Constraints box and make connections from the left edge of the widget to the left edge of the parent and set the margin to 0. Make a connection from the right edge of the widget to the right end of the parent and set the margin to 0. Finally, make a connection from the top of the widget to the top of the nearest widget (the house image view) and set the margin to 0. Your connections should look like this:



In **MainActivity.kt**, add the following near the top of the class, below the other variables declarations:

```
undefined
```

In `onCreate()` add the following just after `searchText` is assigned:

```
undefined
```

If needed, import the `View` from the `android.view.View` namespace:

In `loadApiResults()`, add the following to the top of the method:

```
undefined
```

Add the following at the beginning of `onResponse()`:

```
undefined
```

Add the following at the beginning of `onFailure()`:

```
undefined
```

Click **Apply Changes**.

5) That's it!

Congrats, you've added the finishing touches to your app by adding Material Design and using Retrofit to query live data.

You should now have an understanding of many of the basic building blocks that go into building and Android app.