Name: Nguyễn Văn Cường – Student ID: 20215006

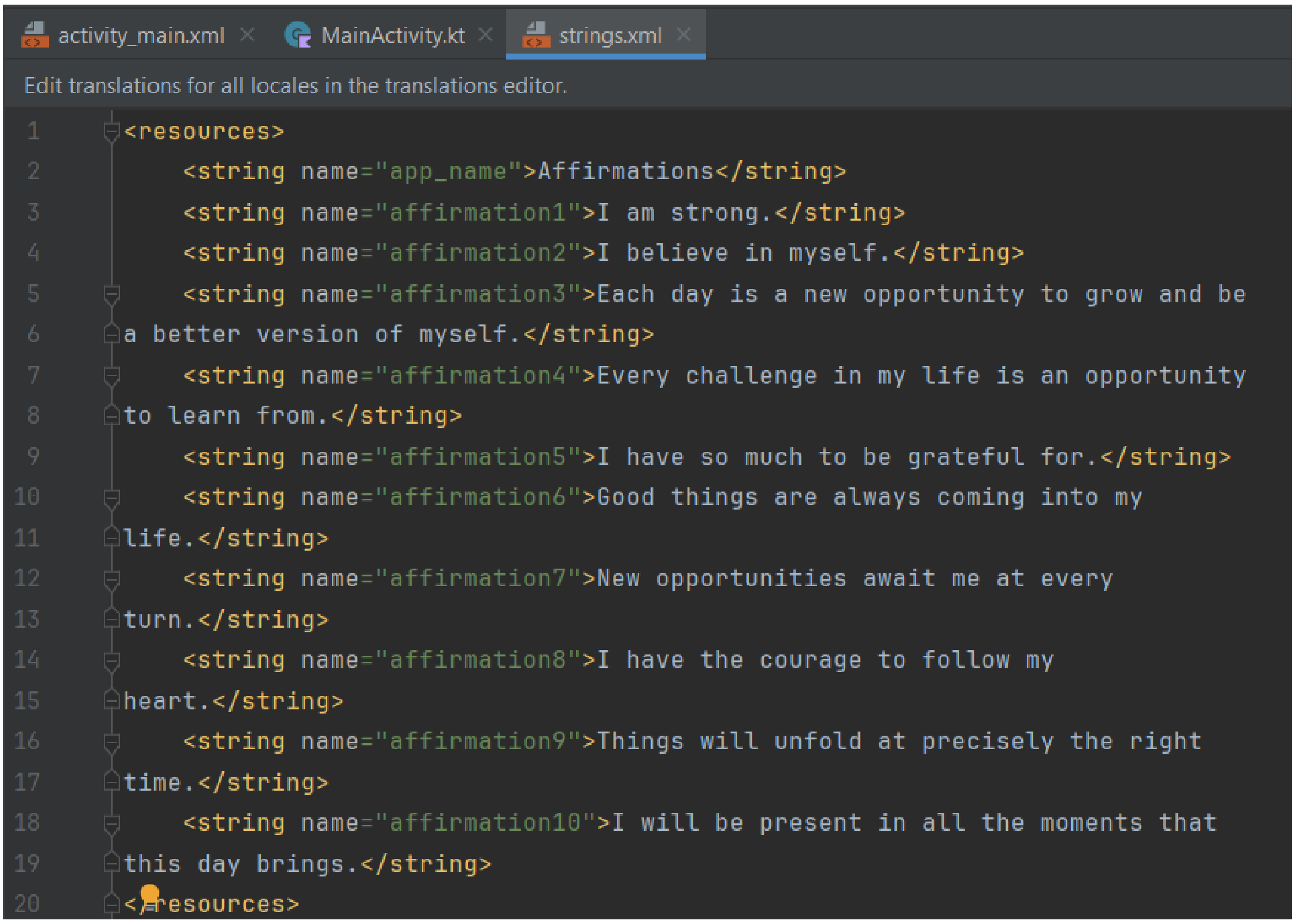
Report – Week 7

5.4. Use RecyclerView to display a scrollable list

Setting up the list of data

Add Affirmation strings

1. In the Project window, open app > res > values > strings.xml. This file currently has a single resource which is the name of the app.
2. In strings.xml, add the following affirmations as individual string resources. Name them affirmation1, affirmation2, and so on.

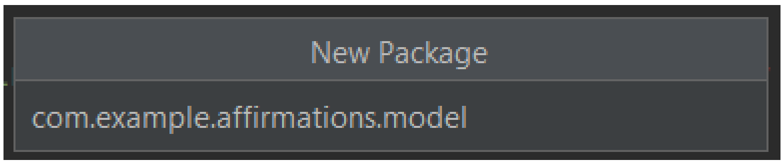


activity_main.xml
MainActivity.kt
strings.xml
Edit translations for all locales in the translations editor.

```xml
<resources>
    <string name="app_name">Affirmations</string>
    <string name="affirmation1">I am strong.</string>
    <string name="affirmation2">I believe in myself.</string>
    <string name="affirmation3">Each day is a new opportunity to grow and be
a better version of myself.</string>
    <string name="affirmation4">Every challenge in my life is an opportunity
to learn from.</string>
    <string name="affirmation5">I have so much to be grateful for.</string>
    <string name="affirmation6">Good things are always coming into my
life.</string>
    <string name="affirmation7">New opportunities await me at every
turn.</string>
    <string name="affirmation8">I have the courage to follow my
heart.</string>
    <string name="affirmation9">Things will unfold at precisely the right
time.</string>
    <string name="affirmation10">I will be present in all the moments that
this day brings.</string>
</resources>
```

Create new package

1. In Android Studio, in the Project pane, right-click app > java > com.example.affirmations and select New > Package.
2. . In the New Package popup, notice the suggested package name prefix. The suggested first part of the package name is the name of the package you right-clicked. While package names do not create a hierarchy of packages, reusing parts of the name is used to indicate a relationship and organization of the content!
3. In the popup, append model to the end of the suggested package name. Developers often use model as the package name for classes that model (or represent) the data.

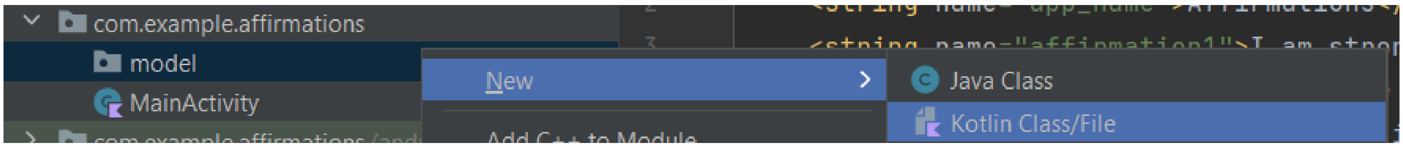


```
com.example.affirmations.model
```

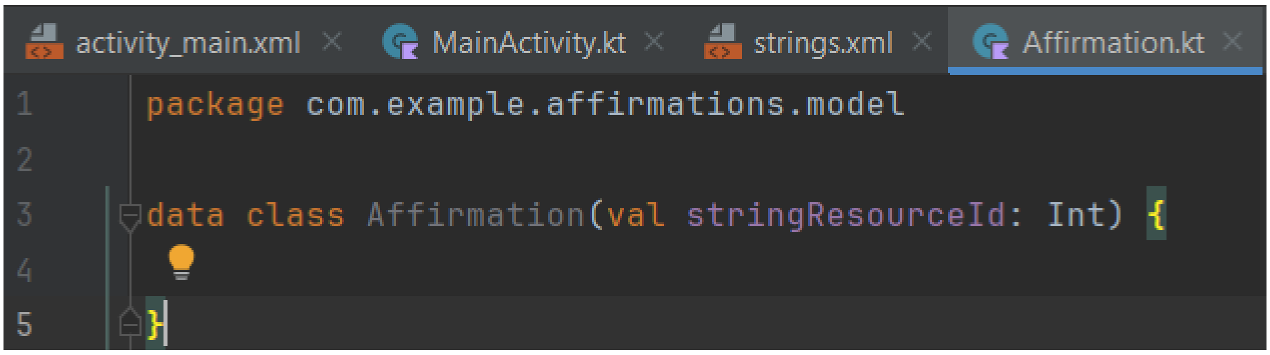
4. Press Enter. This creates a new package under the com.example.affirmations (root) package. This new package will contain any data-related classes defined in your app.

Create the Affirmation data class

1. Right-click on the com.example.affirmations.model package and select New > Kotlin File/Class.



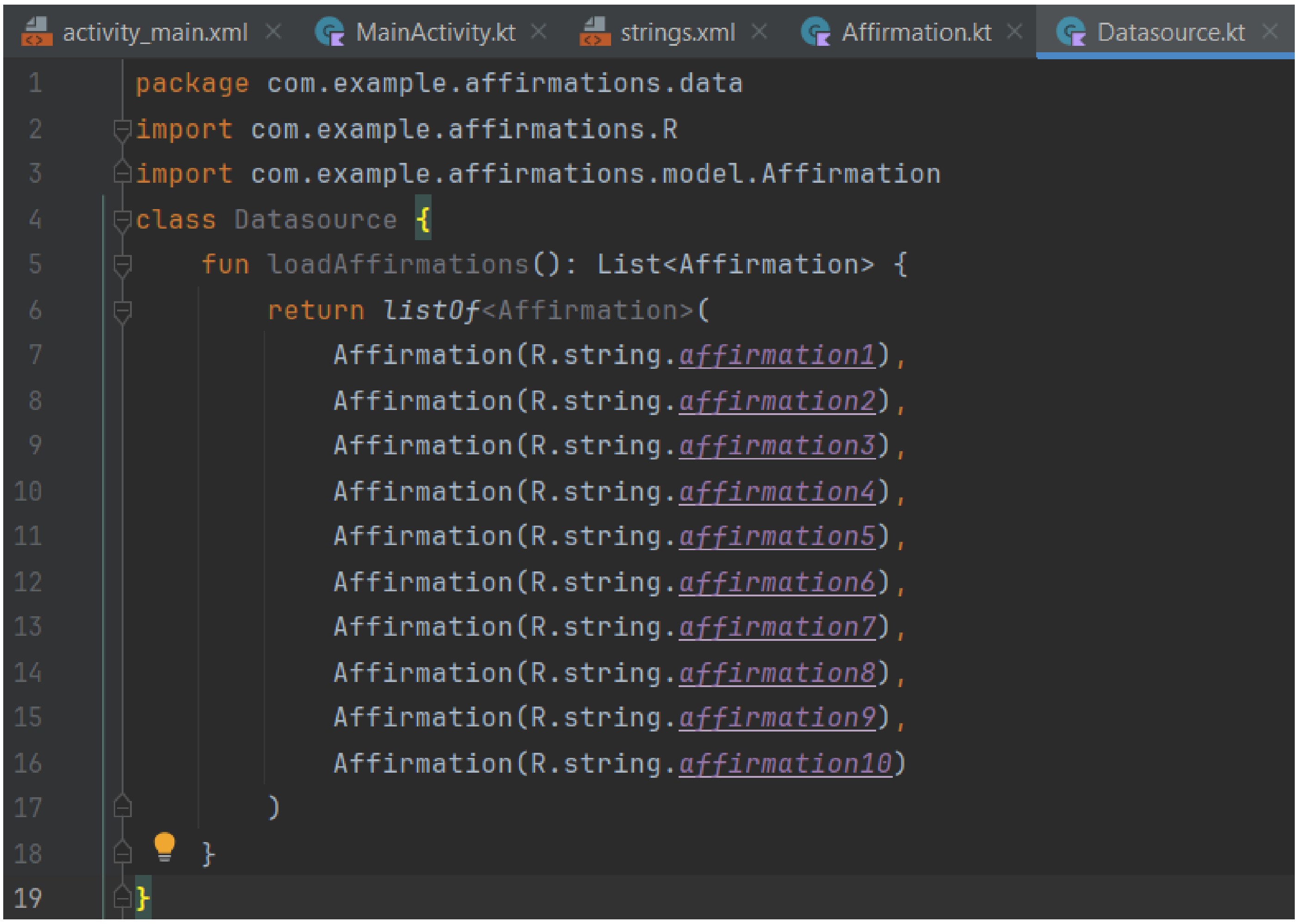
2. . In the popup, select Class and enter Affirmation as the name of the class. This creates a new file called Affirmation.kt in the model package.
3. Make Affirmation a data class by adding the data keyword before the class definition. This leaves you with an error, because data classes must have at least one property defined.
4. Add a val integer parameter stringResourceId to the constructor of the Affirmation class. This gets rid of your error.



```kotlin
package com.example.affirmations.model

data class Affirmation(val stringResourceId: Int) {

}
```

Create a class to be a data source

1. In Android Studio, in the Project window, right-click app > java > com.example.affirmations and select New > Package.
2. Enter data as the last part of the package name.
3. Right click on the data package and select new Kotlin File/Class.
4. Enter Datasource as the class name.
5. Inside the Datasource class, create a function called loadAffirmations().
6. Declare List as the return type of the method loadAffirmations().
7. In the body of loadAffirmations(), add a return statement.
8. After the return keyword, call listOf<>() to create a List.
9. Inside the angle brackets <>, specify the type of the list items as Affirmation. If necessary, import com.example.affirmations.model.Affirmation.
10. Inside the parentheses, create an Affirmation, passing in R.string.affirmation1 as the resource ID as shown below.
11. Add the remaining Affirmation objects to the list of all affirmations, separated by commas. The finished code should look like the following.



```kotlin
package com.example.affirmations.data
import com.example.affirmations.R
import com.example.affirmations.model.Affirmation
class Datasource {
    fun loadAffirmations(): List<Affirmation> {
        return listOf<Affirmation>(
            Affirmation(R.string.affirmation1),
            Affirmation(R.string.affirmation2),
            Affirmation(R.string.affirmation3),
            Affirmation(R.string.affirmation4),
            Affirmation(R.string.affirmation5),
            Affirmation(R.string.affirmation6),
            Affirmation(R.string.affirmation7),
            Affirmation(R.string.affirmation8),
            Affirmation(R.string.affirmation9),
            Affirmation(R.string.affirmation10)
        )
    }
}
```

Display the size of the Affirmations list in a TextView

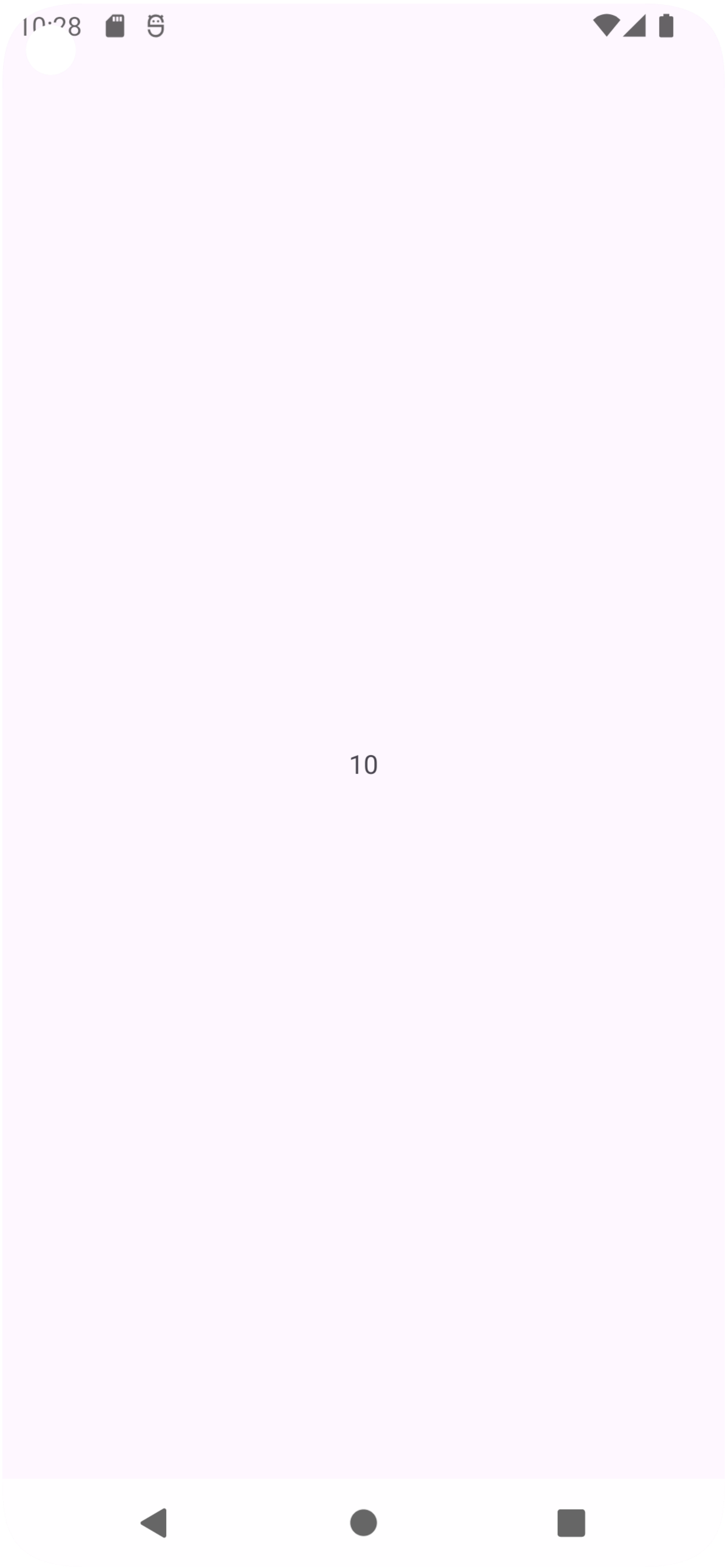
1. In layouts/activity_main.xml, give the TextView that comes with your template an id of textview.
2. In MainActivity in the onCreate() method after the existing code, get a reference to textview.

val textView: TextView = findViewById(R.id.textview)

3. Then add code to create and display the size of the affirmations list. Create a Datasource, call loadAffirmations(), get the size of the returned list, convert it to a string, and assign it as the text of textView.

textView.text = Datasource().loadAffirmations().size.toString()

4. Run your app. The screen should look as shown below



5. Delete the code you just added in MainActivity

Adding a RecyclerView to your app

Add a RecyclerView to the layout

1. Open activity_main.xml (app > res > layout > activity_main.xml)
2. If you are not already using it, switch to Split view
3. Delete the TextView
4. In the XML, replace ConstraintLayout with FrameLayout. The completed layout should look as shown below.

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

<FrameLayout 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"

tools:context=".MainActivity">

</FrameLayout>
```

5. Switch to Design view.
6. In the Palette, select Containers, and find the RecyclerView.
7. Drag a RecyclerView into the layout.
8. If it appears, read the Add Project Dependency popup and click OK. (If the popup doesn't appear, no action is needed.)
9. Wait for Android Studio to finish and the RecyclerView to appear in your layout.
10. If necessary, change the layout_width and layout_height attributes of the RecyclerView to match_parent so that the RecyclerView can fill the whole screen.
11. Set the resource ID of the RecyclerView to recycler_view.
12. Switch back to Code view. In the XML code, inside the RecyclerView element, add LinearLayoutManager as the layout manager attribute of the RecyclerView, as shown below.

```
app:layoutManager="LinearLayoutManager"
```

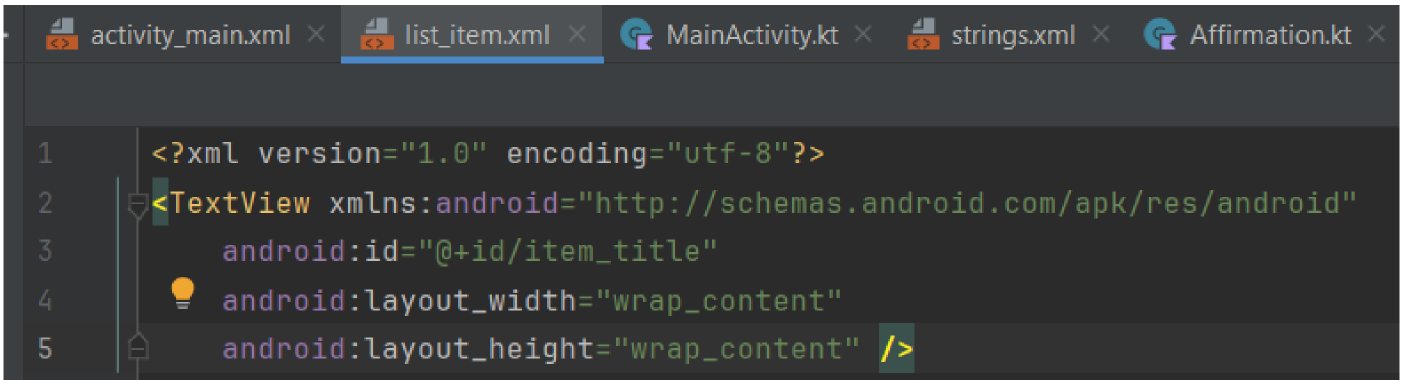
13. Inside RecyclerView, add an android:scrollbars attribute set to vertical.

```
android:scrollbars="vertical"
```

Implement an Adapter for the RecyclerView

Create a layout for items

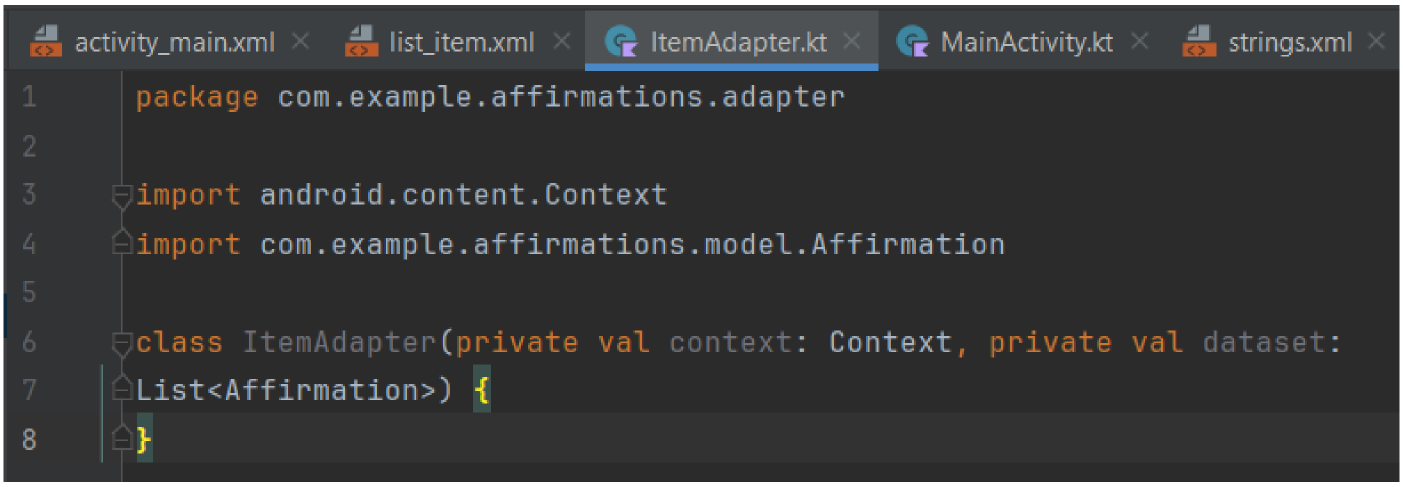
1. In res > layout, create a new empty File called list_item.xml.
2. Open list_item.xml in Code view.
3. Add a TextView with id item_title.
4. Add wrap_content for the layout_width and layout_height, as shown in the code below.



```xml
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/item_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Create an ItemAdapter class

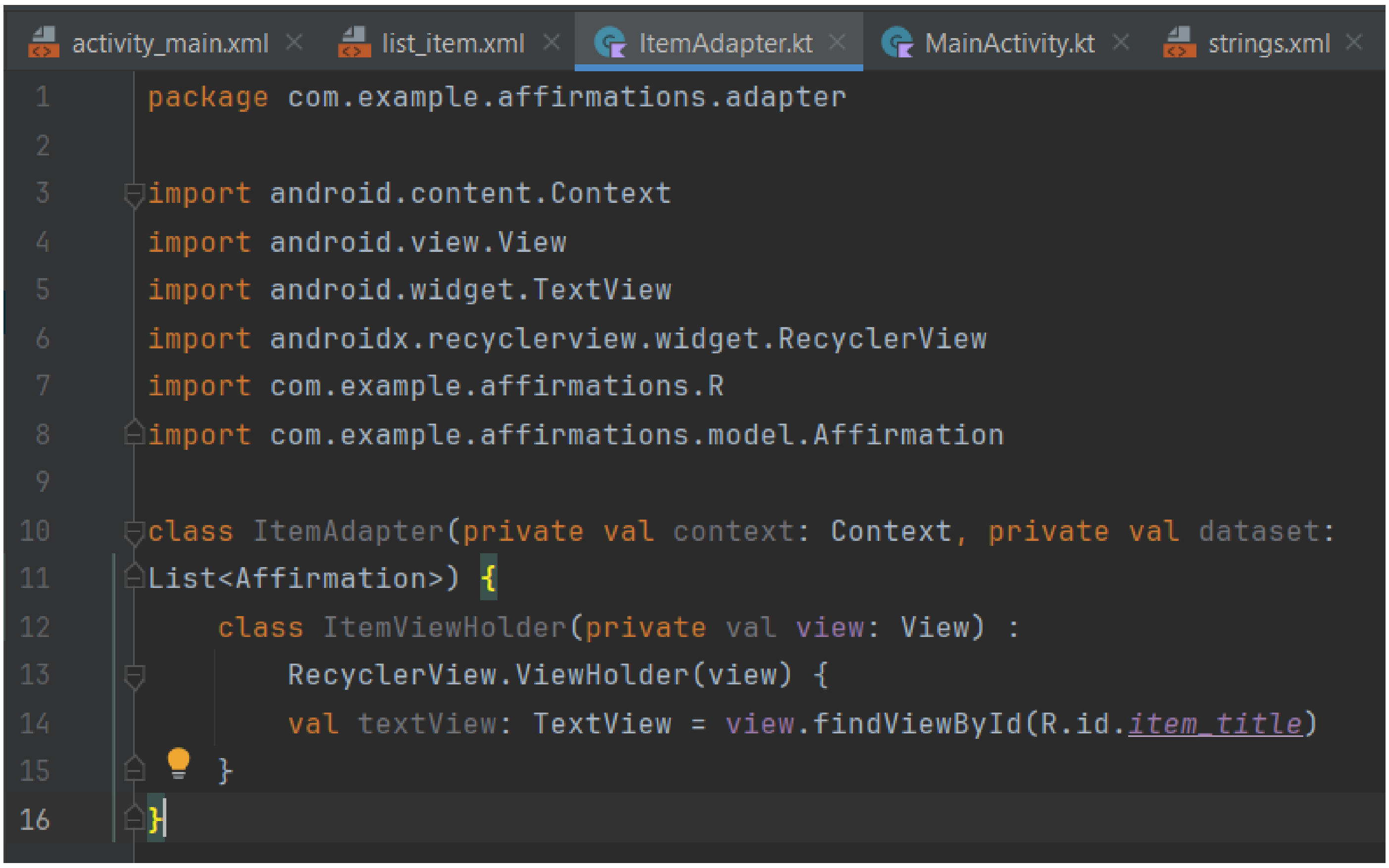
1. In Android Studio in the Project pane, right-click app > java > com.example.affirmations and select New > Package.
2. Enter adapter as the last part of the package name.
3. Right-click on the adapter package and select New > Kotlin File/Class.
4. Enter ItemAdapter as the class name, finish, and the ItemAdapter.kt file opens.
5. Add a parameter to the ItemAdapter constructor that is a val called dataset of type List. Import Affirmation, if necessary.
6. Since the dataset will be only used within this class, make it private.
7. Add a parameter to the ItemAdapter constructor that is a val called context of type Context. Position it as the first parameter in the constructor



```kotlin
package com.example.affirmations.adapter

import android.content.Context
import com.example.affirmations.model.Affirmation

class ItemAdapter(private val context: Context, private val dataset:
List<Affirmation>) {
}
```

Create a ViewHolder

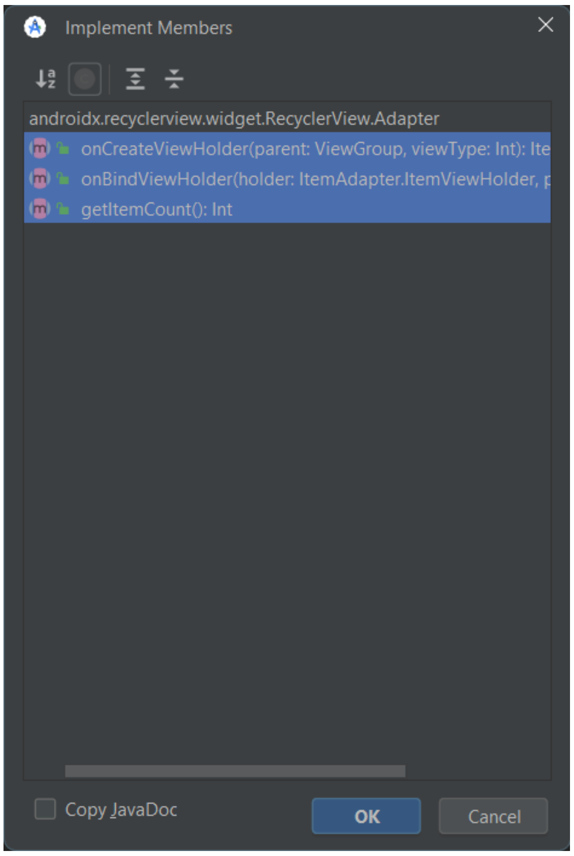
1. Inside the ItemAdapter class, before the closing curly brace for ItemAdapter, create an ItemViewHolder class.
2. Add a private val view of type View as a parameter to the ItemViewHolder class constructor.
3. Make ItemViewHolder a subclass of RecyclerView.ViewHolder and pass the view parameter into the superclass constructor.
4. Inside ItemViewHolder, define a val property textView that is of type TextView. Assign it the view with the ID item_title that you defined in list_item.xml.



```kotlin
package com.example.affirmations.adapter

import android.content.Context
import android.view.View
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.affirmations.R
import com.example.affirmations.model.Affirmation

class ItemAdapter(private val context: Context, private val dataset:
List<Affirmation>) {
    class ItemViewHolder(private val view: View) :
        RecyclerView.ViewHolder(view) {
        val textView: TextView = view.findViewById(R.id.item_title)
    }
}
```

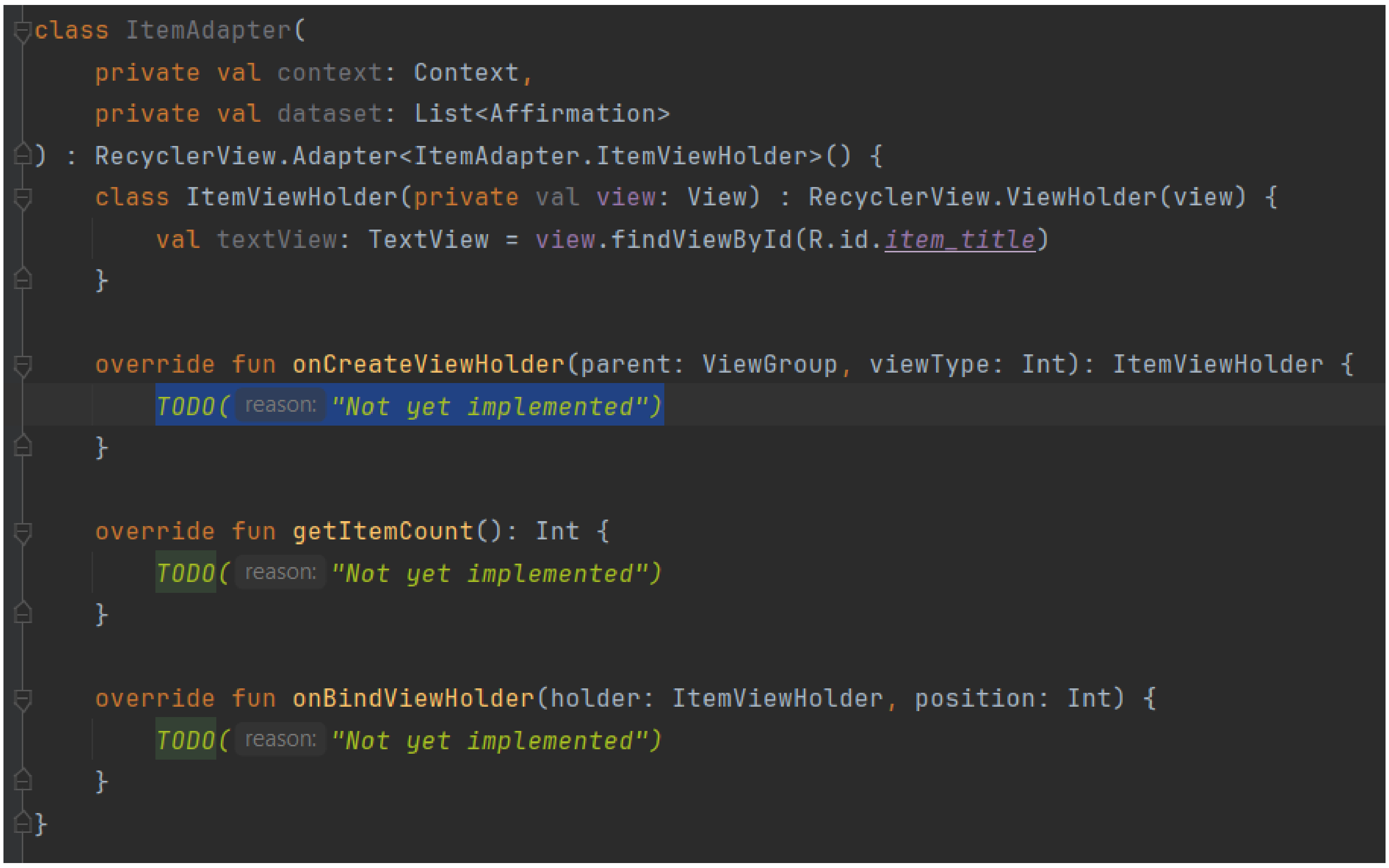
Override adapter methods

1. Add the code to extend your ItemAdapter from the abstract class RecyclerView.Adapter. Specify ItemAdapter.ItemViewHolder as the view holder type in angle brackets.
2. Put your cursor on ItemAdapter and press Command+I (Control+I on Windows). This shows you the list of methods you need to implement: getItemCount(), onCreateViewHolder(), and onBindViewHolder().



androidx.recyclerview.widget.RecyclerView.Adapter
onCreateViewHolder(parent: ViewGroup, viewType: Int): Ite
onBindViewHolder(holder: ItemAdapter.ItemViewHolder, p
getItemCount(): Int

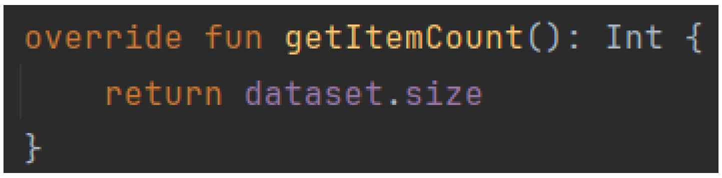
3. Select all three functions using Shift+click and click OK.



```
class ItemAdapter(
    private val context: Context,
    private val dataset: List<Affirmation>
) : RecyclerView.Adapter<ItemAdapter.ItemViewHolder>() {
    class ItemViewHolder(private val view: View) : RecyclerView.ViewHolder(view) {
        val textView: TextView = view.findViewById(R.id.item_title)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
        TODO( reason: "Not yet implemented")
    }

    override fun getItemCount(): Int {
        TODO( reason: "Not yet implemented")
    }

    override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
        TODO( reason: "Not yet implemented")
    }
}
```

Implement getItemCount()

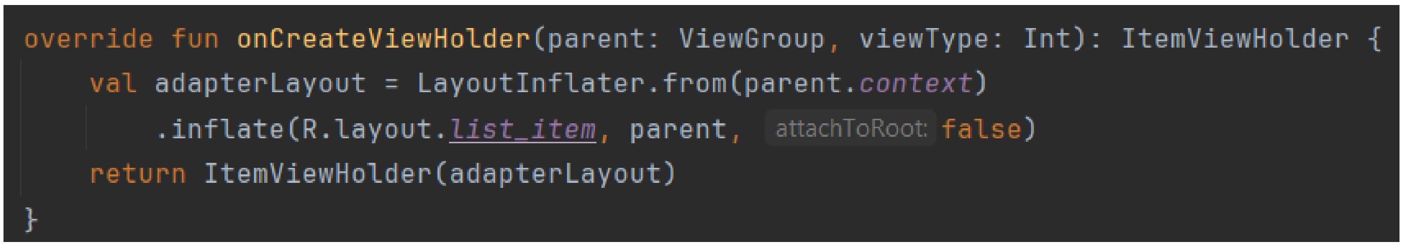
1. Replace getItemCount() with this:



```
override fun getItemCount(): Int {
    return dataset.size
}
```

Implement onCreateViewHolder()

1. In the onCreateViewHolder() method, obtain an instance of LayoutInflater from the provided context (context of the parent). The layout inflater knows how to inflate an XML layout into a hierarchy of view objects.
2. Once you have a LayoutInflater object instance, add a period followed by another method call to inflate the actual list item view. Pass in the XML layout resource ID R.layout.list_item and the parent view group. The third boolean argument is attachToRoot. This argument needs to be false, because RecyclerView adds this item to the view hierarchy for you when it's time.
3. In onCreateViewHolder(), return a new ItemViewHolder instance where the root view is adapterLayout.



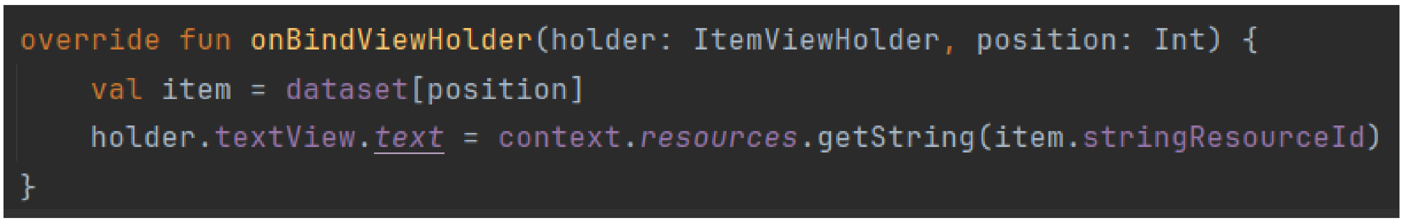
```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
    val adapterLayout = LayoutInflater.from(parent.context)
        .inflate(R.layout.list_item, parent,  attachToRoot: false)
    return ItemViewHolder(adapterLayout)
}
```

Implement onBindViewHolder()

1. Inside onBindViewHolder(), create a val item and get the item at the given position in the dataset.
2. With an Affirmation object instance, you can find the corresponding string resource ID by calling item.stringResourceId. However, this is an integer and you need to find the mapping to the actual string value.

In the Android framework, you can call getString() with a string resource ID, and it will return the string value associated with it. getString() is a method in the Resources class, and you can get an instance of the Resources class through the context.

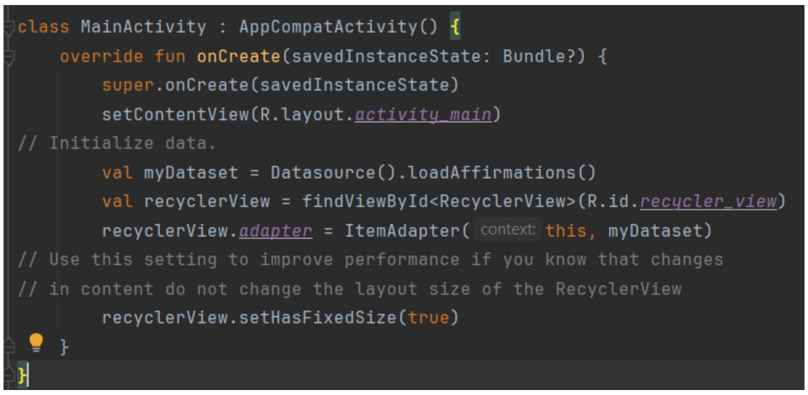
That means you can call context.resources.getString() and pass in a string resource ID. The resulting string can be set as the text of the textView in the holder ItemViewHolder. In short, this line of code updates the view holder to show the affirmation string.



```kotlin
override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
    val item = dataset[position]
    holder.textView.text = context.resources.getString(item.stringResourceId)
}
```

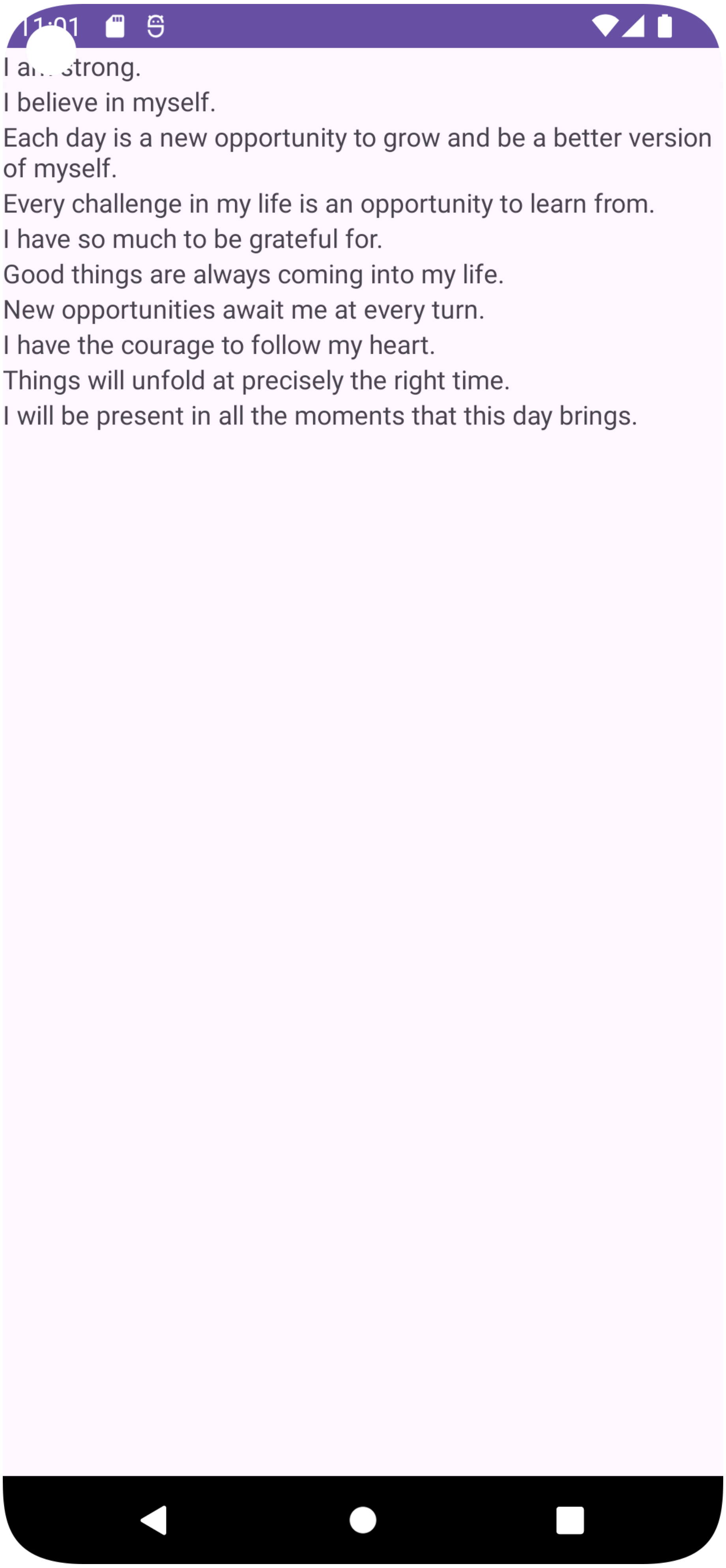
Modify the MainActivity to use a RecyclerView

1. Open MainActivity.kt.
2. In MainActivity, go to the onCreate() method. Insert the new code described in the following steps after the call to setContentView(R.layout.activity_main).
3. Create an instance of Datasource, and call the loadAffirmations() method on it. Store the returned list of affirmations in a val named myDataset.
4. Create a variable called recyclerView and use findViewById()to find a reference to the RecyclerView within the layout
5. To tell the recyclerView to use the ItemAdapter class you created, create a new ItemAdapter instance. ItemAdapter expects two parameters: the context (this) of this activity, and the affirmations in myDataset.
6. Assign the ItemAdapter object to the adapter property of the recyclerView.
7. Since the layout size of your RecyclerView is fixed in the activity layout, you can set the setHasFixedSize parameter of the RecyclerView to true. This setting is only needed to improve performance. Use this setting if you know that changes in content do not change the layout size of the RecyclerView.
8. When you are done, the code for MainActivity should be similar to the following



```kotlin
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
// Initialize data.
        val myDataset = Datasource().loadAffirmations()
        val recyclerView = findViewById<RecyclerView>(R.id.recycler_view)
        recyclerView.adapter = ItemAdapter( context: this, myDataset)
// Use this setting to improve performance if you know that changes
// in content do not change the layout size of the RecyclerView
        recyclerView.setHasFixedSize(true)
    }
}
```

9. Run your app



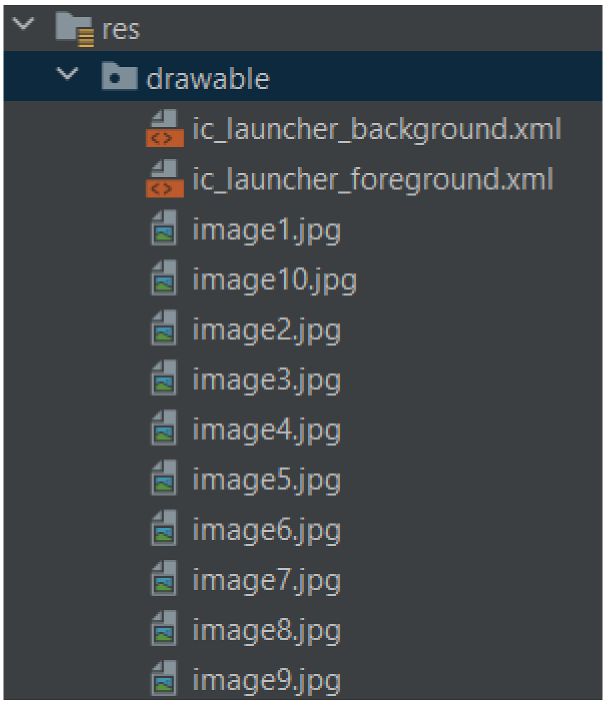
11:01

I am strong.
I believe in myself.
Each day is a new opportunity to grow and be a better version of myself.
Every challenge in my life is an opportunity to learn from.
I have so much to be grateful for.
Good things are always coming into my life.
New opportunities await me at every turn.
I have the courage to follow my heart.
Things will unfold at precisely the right time.
I will be present in all the moments that this day brings.

5.5 Display a list of images using cards

Adding images to the list items

Download the images



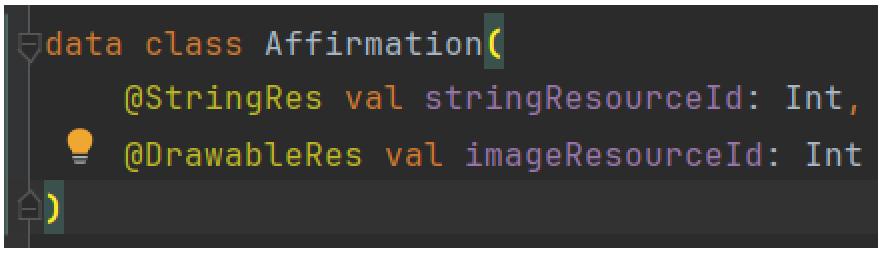
```
res
  drawable
    ic_launcher_background.xml
    ic_launcher_foreground.xml
    image1.jpg
    image10.jpg
    image2.jpg
    image3.jpg
    image4.jpg
    image5.jpg
    image6.jpg
    image7.jpg
    image8.jpg
    image9.jpg
```

Add support for images in the Affirmation class

1. Open the Affirmation.kt file within the model package.
2. Modify the constructor of the Affirmation class by adding another Int parameter named imageResourceId.

Using resource annotations

1. Add the @StringRes annotation to stringResourceId.
2. Add the @DrawableRes annotation to imageResourceId.
3. Make sure the imports androidx.annotation.DrawableRes and androidx.annotation.StringRes are added at the top of your file after the package declaration.



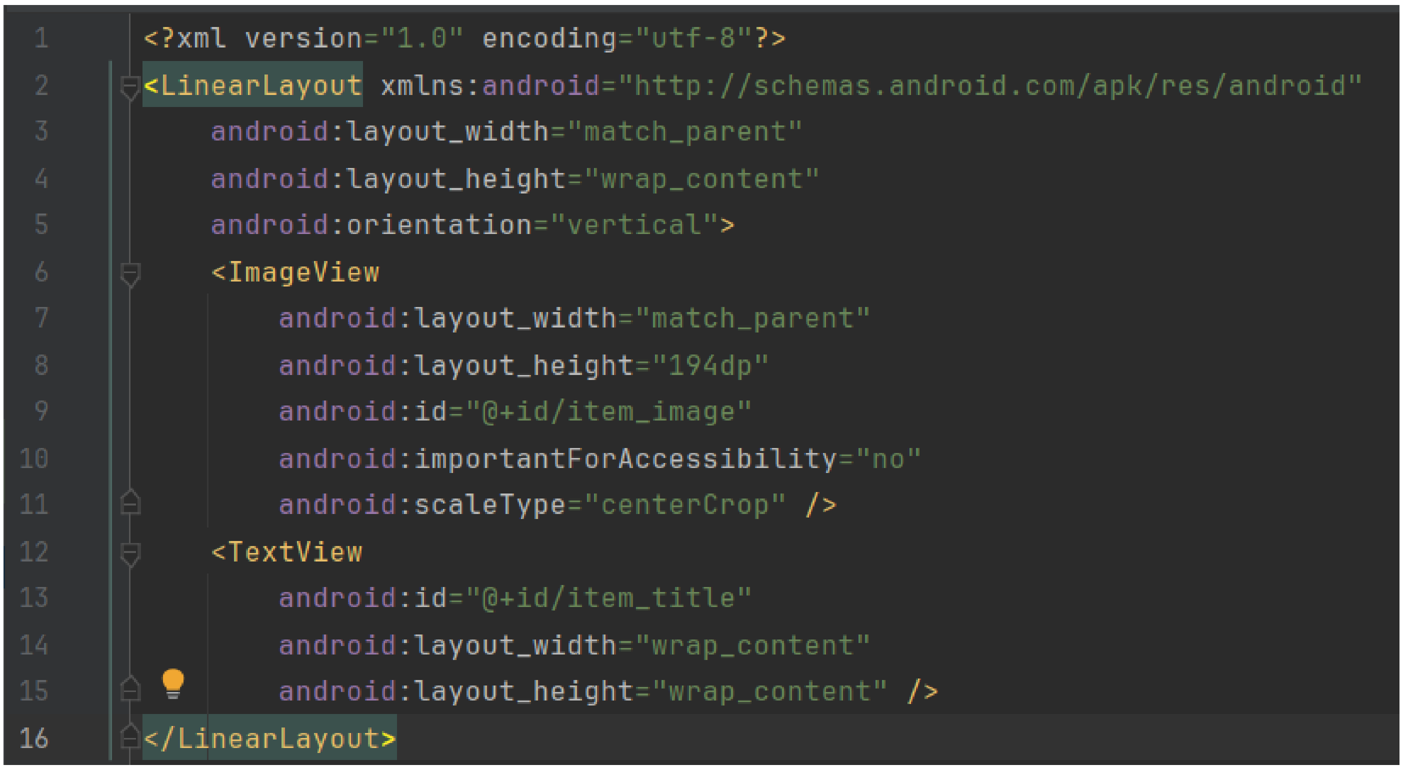
```kotlin
data class Affirmation(
    @StringRes val stringResourceId: Int,
    @DrawableRes val imageResourceId: Int
)
```

Initialize list of affirmations with images

1. Open Datasource.kt. You should see an error for each instantiation of Affirmation.
2. For each Affirmation, add the resource ID of an image as an argument, such as R.drawable.image1.

Add an ImageView to the list item layout

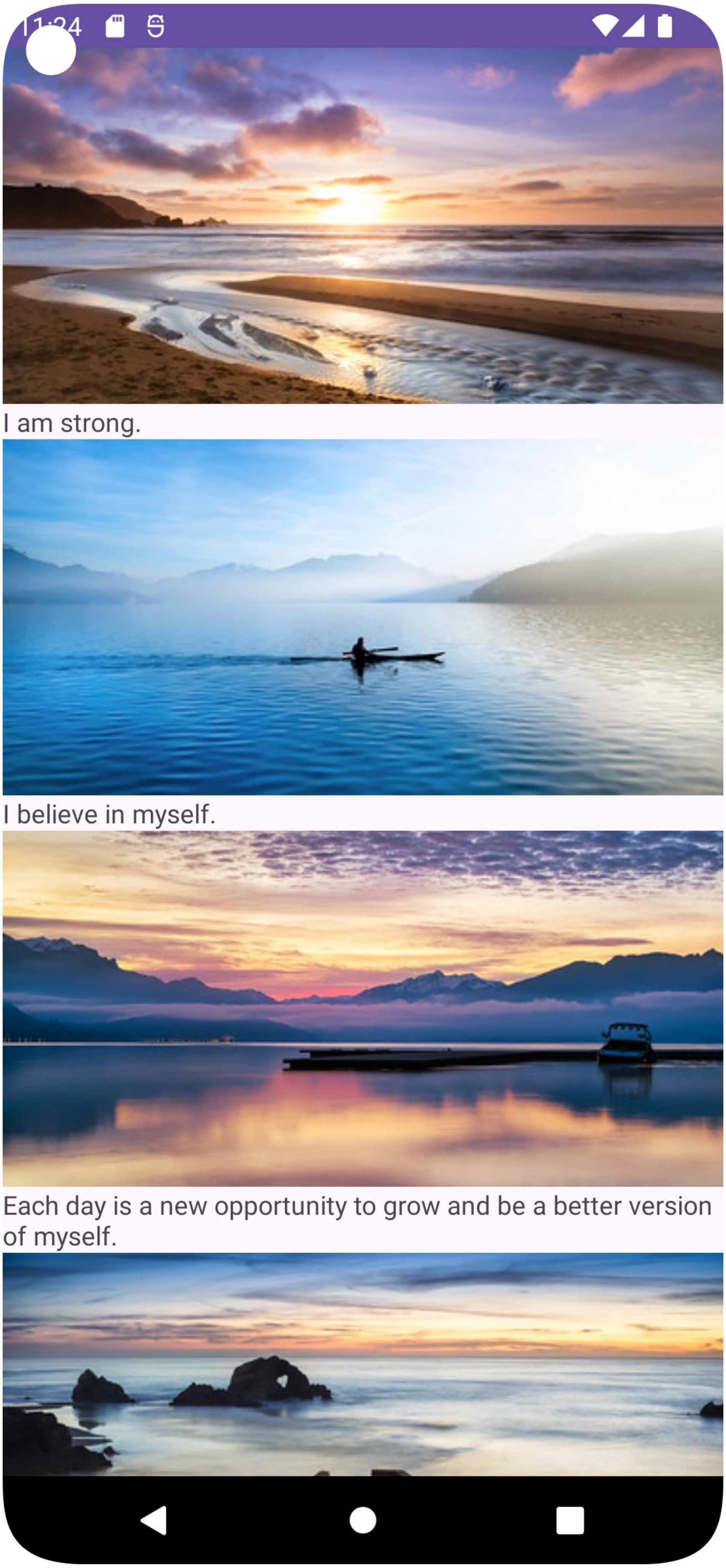
1. Open res > layout > list_item.xml. Add a LinearLayout around the existing TextView and set the orientation property to vertical.
2. Move the xmlns schema declaration line from the TextView element to the LinearLayout element to get rid of the error.
3. . Inside the LinearLayout, before the TextView, add an ImageView with a resource ID of item_image.
4. Set the ImageView's width to match_parent and height to 194dp. Depending on screen size, this value should show a few cards on screen at any given time.
5. Set the scaleType to centerCrop.
6. Set the importantForAccessibility attribute to no since the image is used for decorative purposes.



```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="194dp"
        android:id="@+id/item_image"
        android:importantForAccessibility="no"
        android:scaleType="centerCrop" />
    <TextView
        android:id="@+id/item_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Update the ItemAdapter to set the image

1. Open adapter/ItemAdapter.kt (app > java > adapter > ItemAdapter)
2. Go to the ItemViewHolder class.
3. An ItemViewHolder instance should hold a reference to the TextView and a reference to the ImageView in the list item layout. Make the following change.
4. Find the onBindViewHolder() function in ItemAdapter.
5. Previously you set the affirmation's stringResourceId on to textView in the ItemViewHolder. Now set the affirmation item's imageResourceId onto the ImageView of the list item view.
6. Run the app and scroll through the list of affirmations.



I am strong.

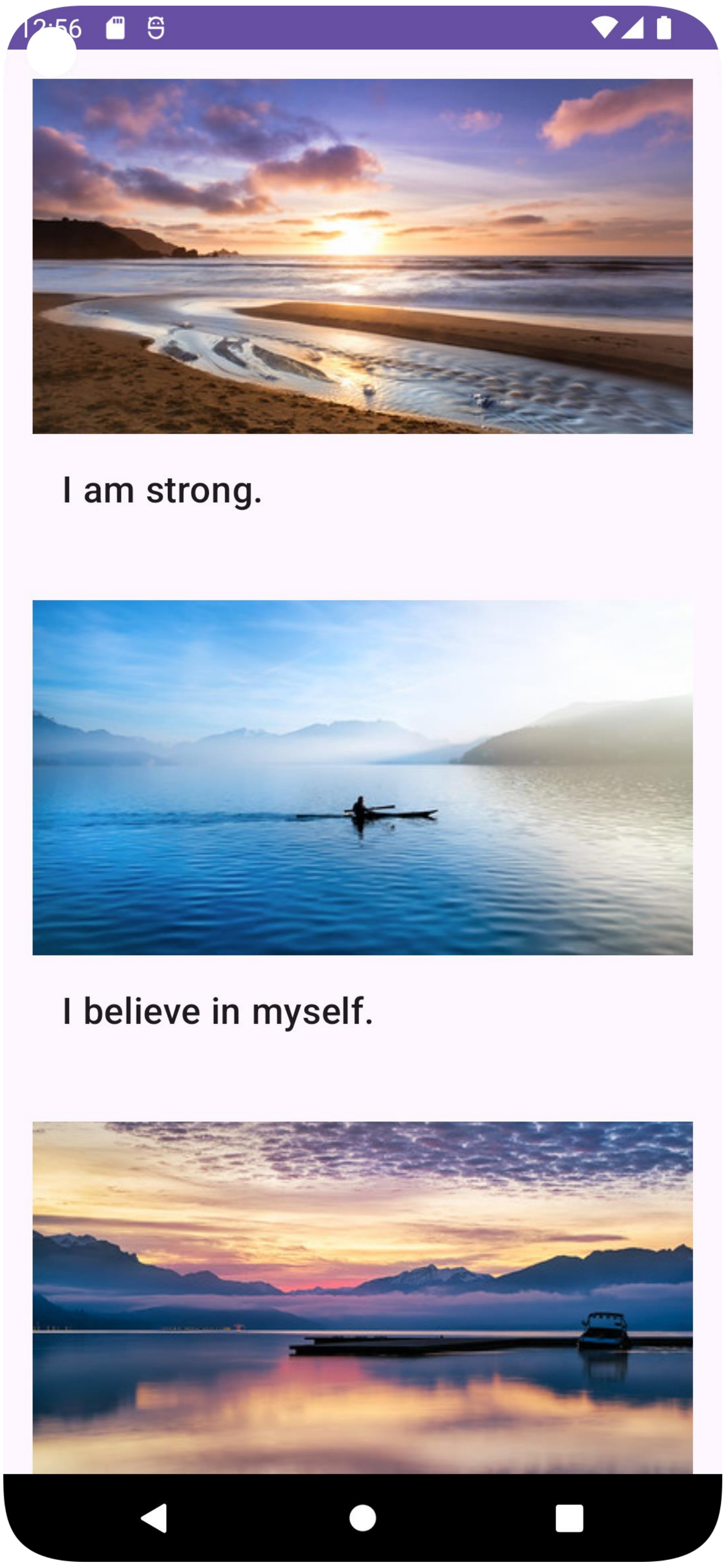
I believe in myself.

Each day is a new opportunity to grow and be a better version of myself.

Polishing the UI

Add padding

1. Open item_list.xml (app > res > layout > item_list.xml) and add 16dp padding to the existing LinearLayout.
2. Add 16dp padding to the item_title TextView.
3. In the TextView, set the textAppearance attribute to ?attr/textAppearanceHeadline6
4. Run the app.

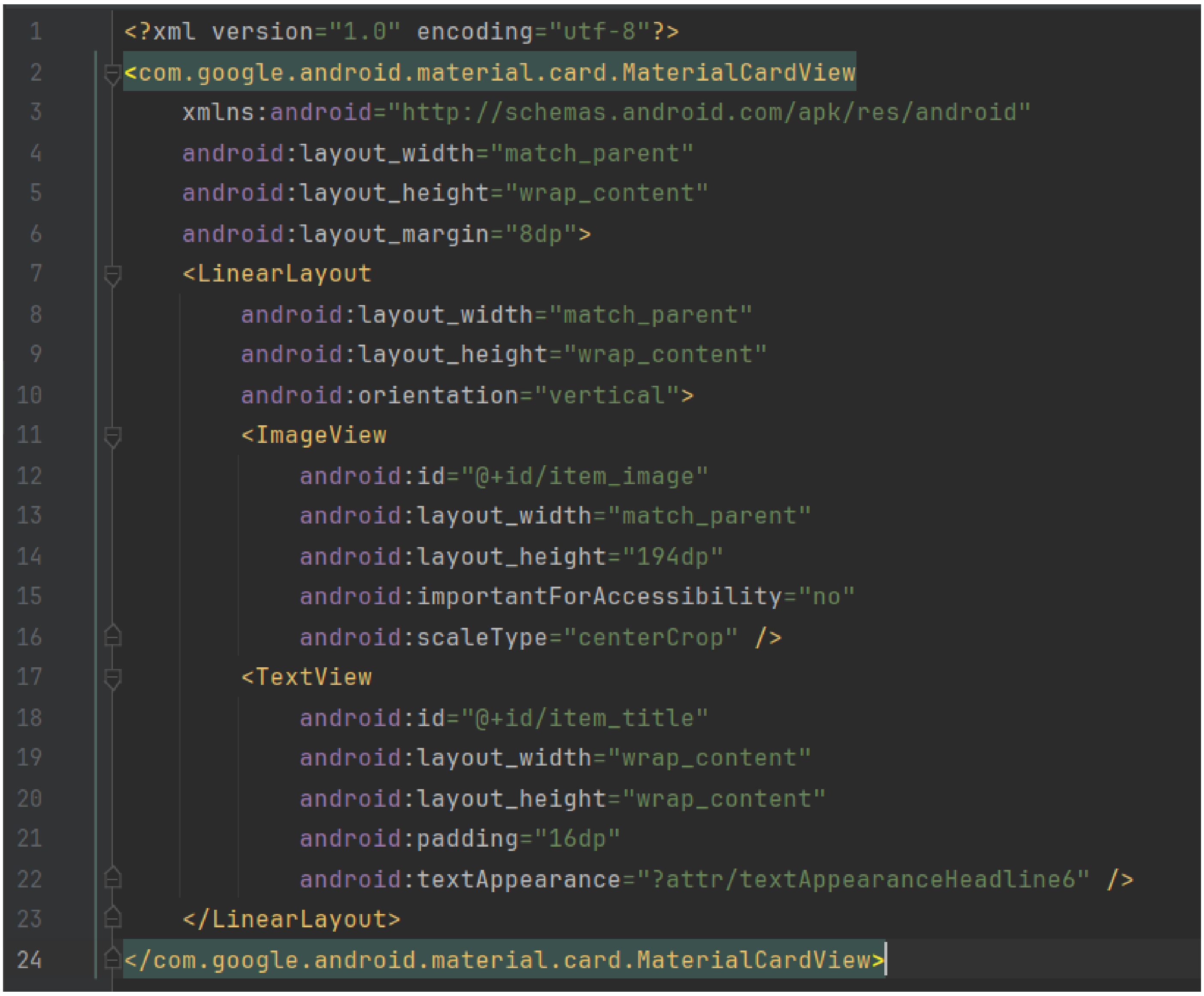


I am strong.

I believe in myself.

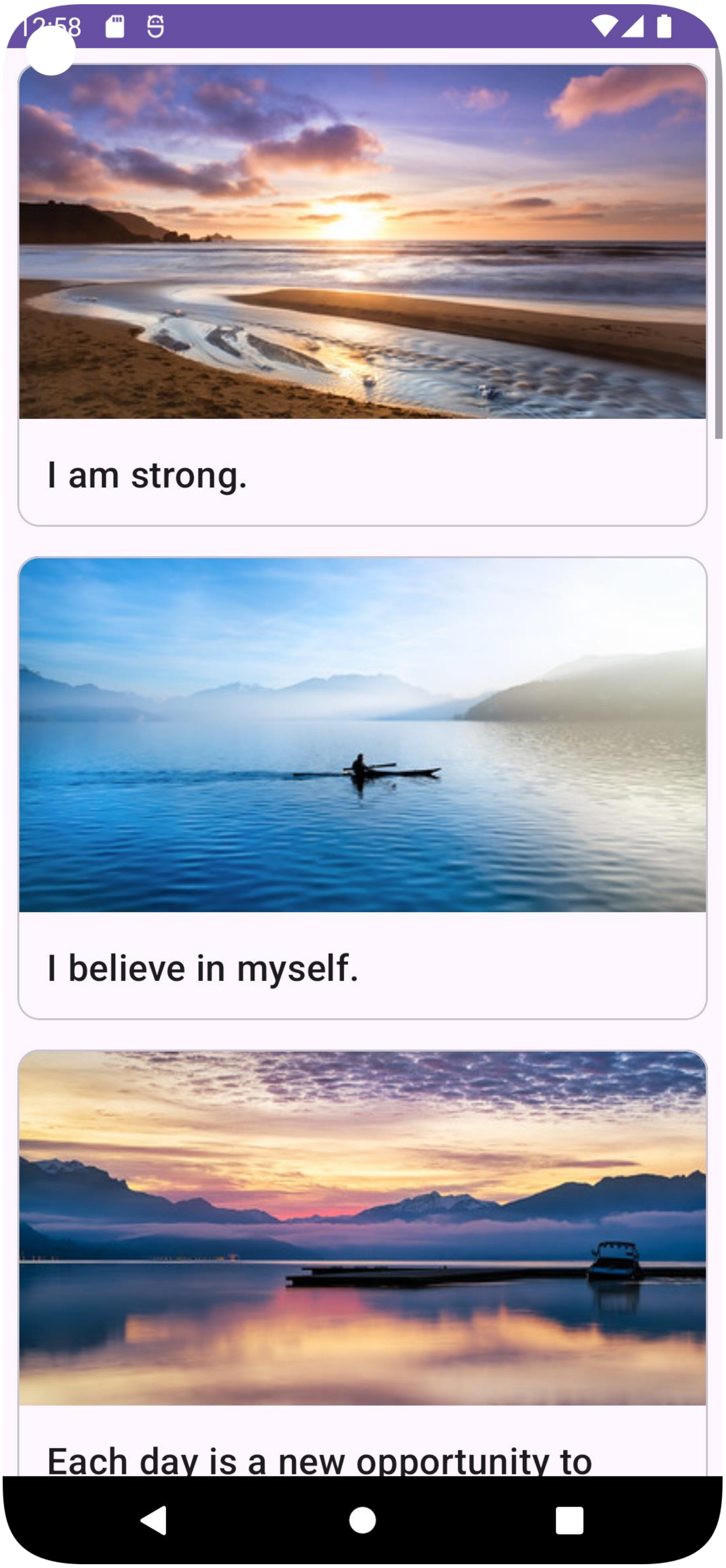
Use cards

1. Add a MaterialCardView around the existing LinearLayout.
2. Once again, move the schema declaration from LinearLayout into MaterialCardView.
3. Set the layout_width of the MaterialCardView to match_parent, and the layout_height to wrap_content.
4. Add a layout_margin of 8dp.
5. Remove the padding in the LinearLayout, so you don't have too much whitespace.



```xml
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.material.card.MaterialCardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="8dp">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <ImageView
            android:id="@+id/item_image"
            android:layout_width="match_parent"
            android:layout_height="194dp"
            android:importantForAccessibility="no"
            android:scaleType="centerCrop" />
        <TextView
            android:id="@+id/item_title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="16dp"
            android:textAppearance="?attr/textAppearanceHeadline6" />
    </LinearLayout>
</com.google.android.material.card.MaterialCardView>
```

6. Now run the app again.



I am strong.

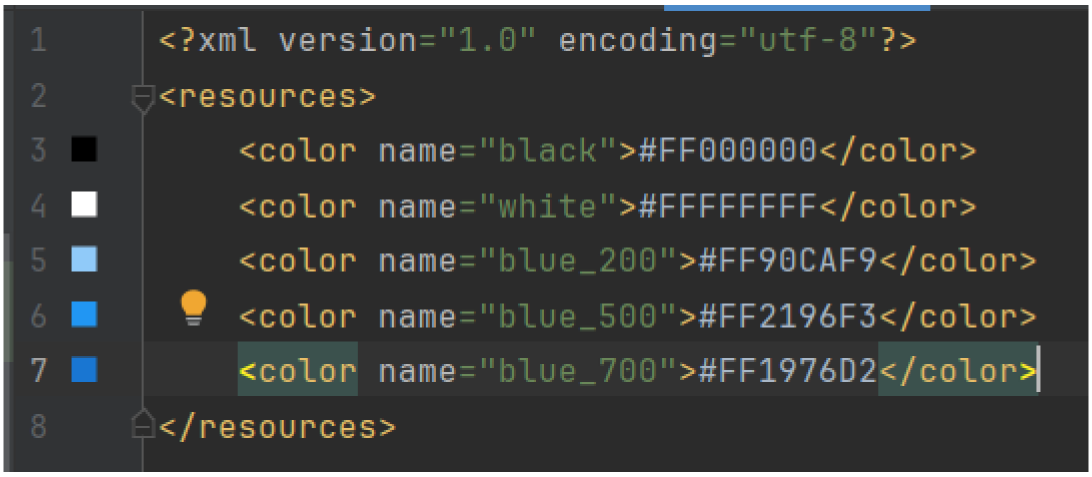
I believe in myself.

Each day is a new opportunity to

Change the app theme colors

Add color resources

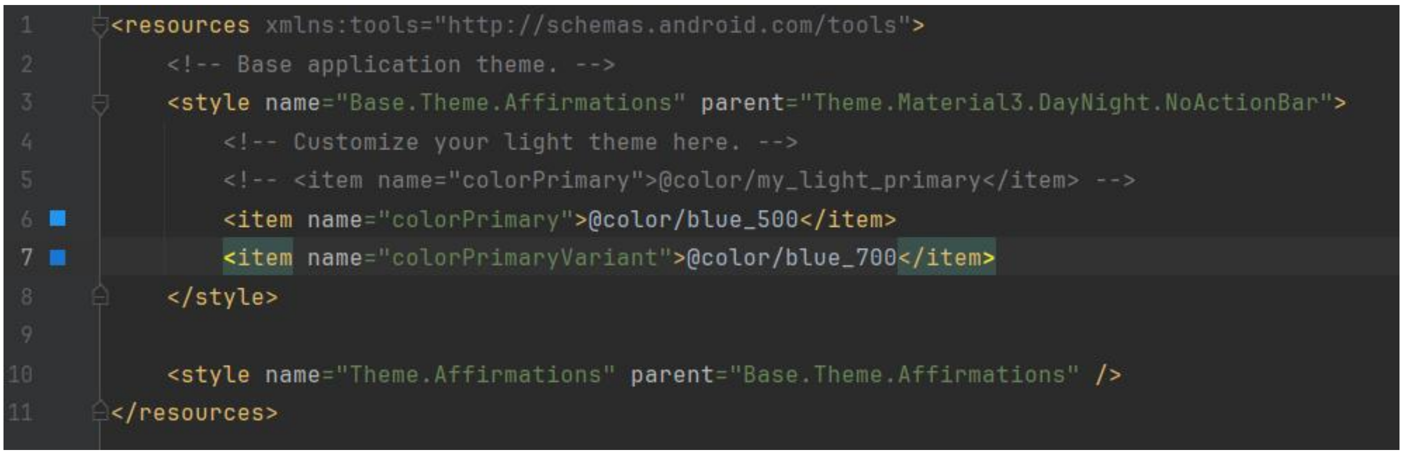
1. Open colors.xml (res > values > colors.xml).
2. Add new color resources to the file for the blue colors defined below:



```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="blue_200">#FF90CAF9</color>
    <color name="blue_500">#FF2196F3</color>
    <color name="blue_700">#FF1976D2</color>
</resources>
```

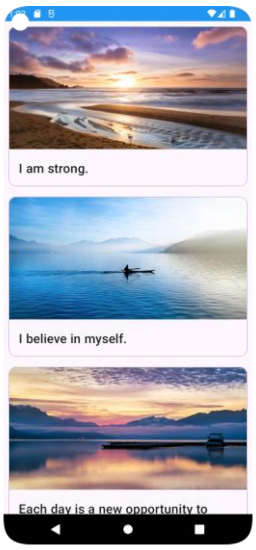
Change the theme colors

1. Open themes.xml (res > values > themes > themes.xml).
2. Find the section <!-- Primary brand color. -->
3. Add or change colorPrimary to use @color/blue_500.
4. Add or change colorPrimaryVariant to use @color/blue_700.



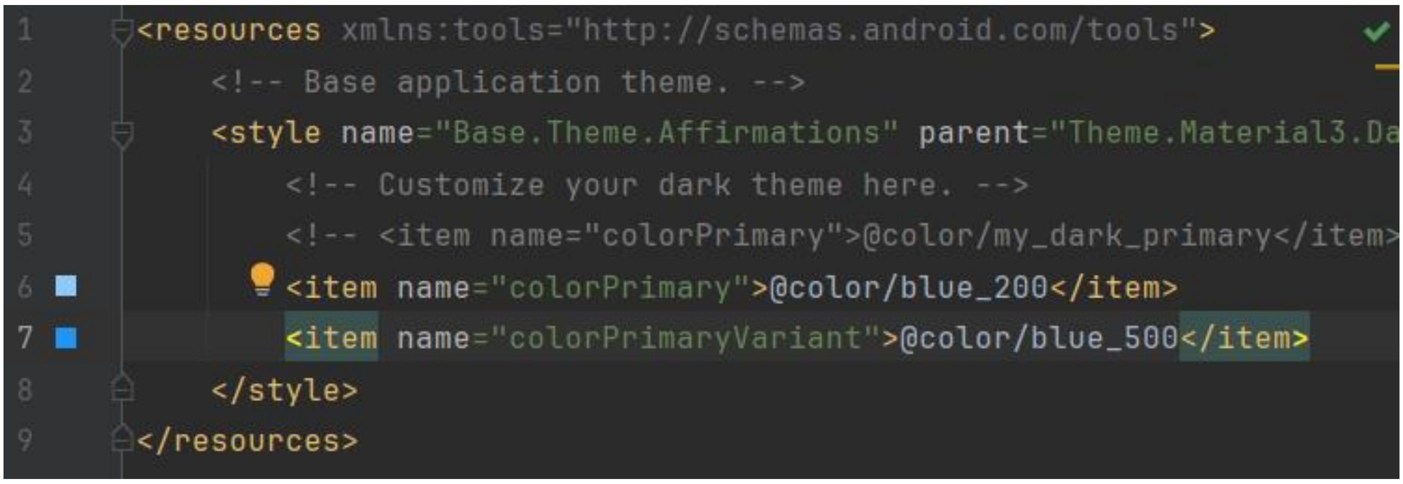
```xml
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Base.Theme.Affirmations" parent="Theme.Material3.DayNight.NoActionBar">
        <!-- Customize your light theme here. -->
        <!-- <item name="colorPrimary">@color/my_light_primary</item> -->
        <item name="colorPrimary">@color/blue_500</item>
        <item name="colorPrimaryVariant">@color/blue_700</item>
    </style>

    <style name="Theme.Affirmations" parent="Base.Theme.Affirmations" />
</resources>
```

5. Run the app.



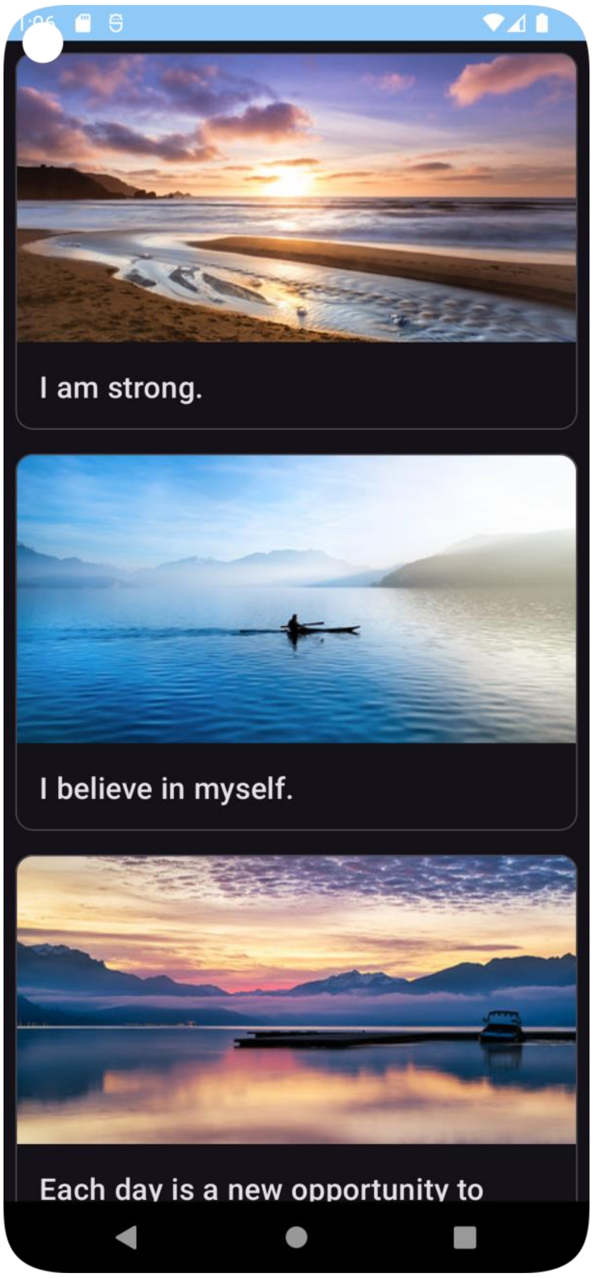
Update the dark theme colors

1. Open the dark theme themes.xml file (themes > themes.xml (night)).
2. Add or change the colorPrimary and colorPrimaryVariant theme attributes as follows:



```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Base.Theme.Affirmations" parent="Theme.Material3.Da
        <!-- Customize your dark theme here. -->
        <!-- <item name="colorPrimary">@color/my_dark_primary</item>
        <item name="colorPrimary">@color/blue_200</item>
        <item name="colorPrimaryVariant">@color/blue_500</item>
    </style>
</resources>
```

3. Run your app.
4. In the Settings of your device, turn on the Dark Theme.



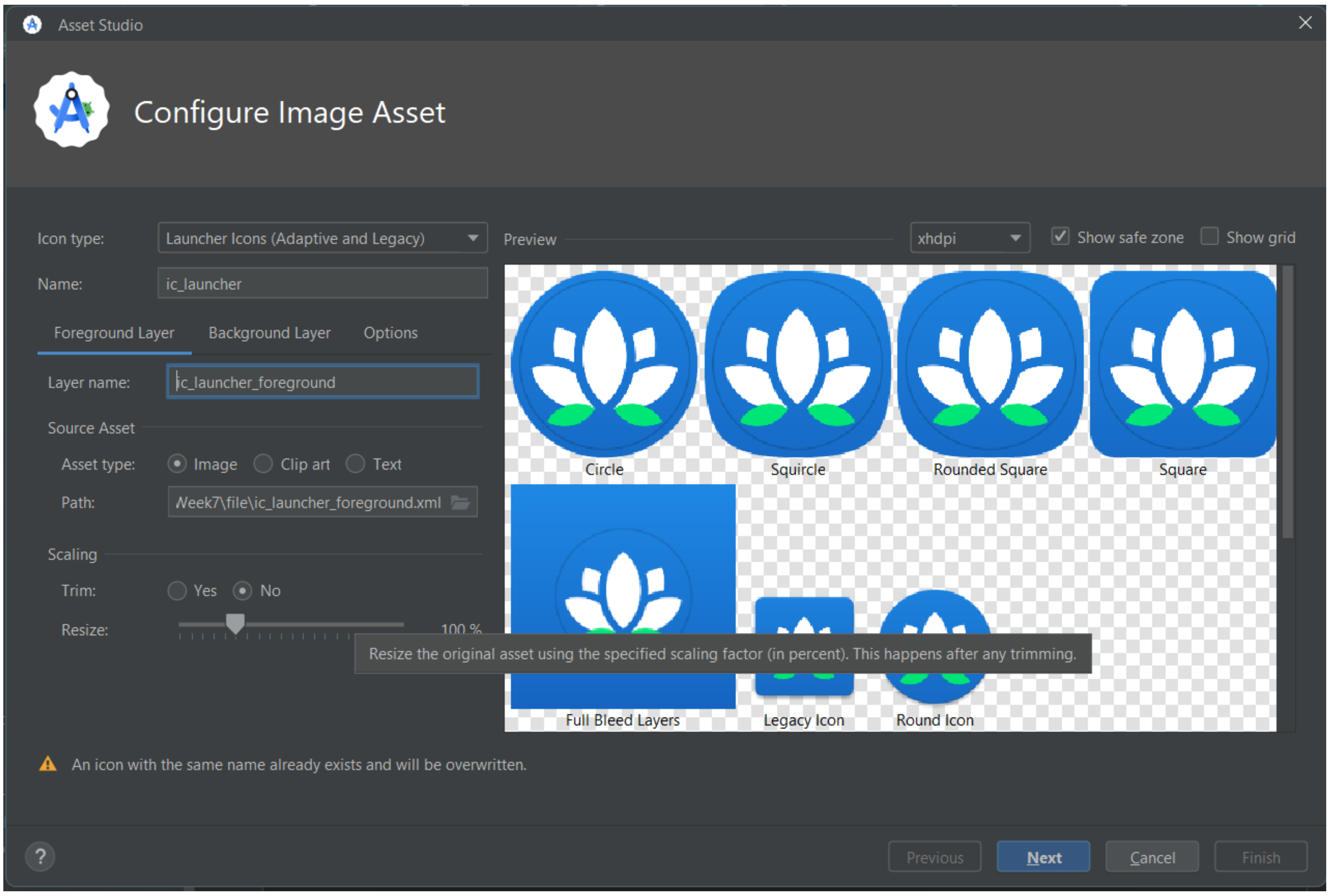
I am strong.

I believe in myself.

Each day is a new opportunity to

Change the app icon

1. Download the app icon files ic_launcher_foreground.xml and ic_launcher_background.xml. If your browser shows the file instead of downloading it, select File > Save Page As... to save it to your computer.
2. Within Android Studio, delete two files: drawable/ic_launcher_background.xml and drawable-v24/ic_launcher_foreground.xml files since those are for the previous app icon. You can uncheck the box Safe delete (with usage search).
3. Then right click on the res > drawable folder and select New > Image Asset.
4. In the Configure Image Asset window make sure Foreground layer is selected.
5. Below that, find the Path label.
6. Click the folder icon inside the Path text box.
7. Find and open the ic_launcher_foreground.xml file that you downloaded on your computer.


Asset Studio

Configure Image Asset

Icon type: Launcher Icons (Adaptive and Legacy)

Name: ic_launcher

Foreground Layer | Background Layer | Options

Layer name: ic_launcher_foreground

Source Asset

Asset type: [✓ Image] Clip art Text

Path: Week7\file\ic_launcher_foreground.xml

Scaling

Trim: Yes [✓ No]

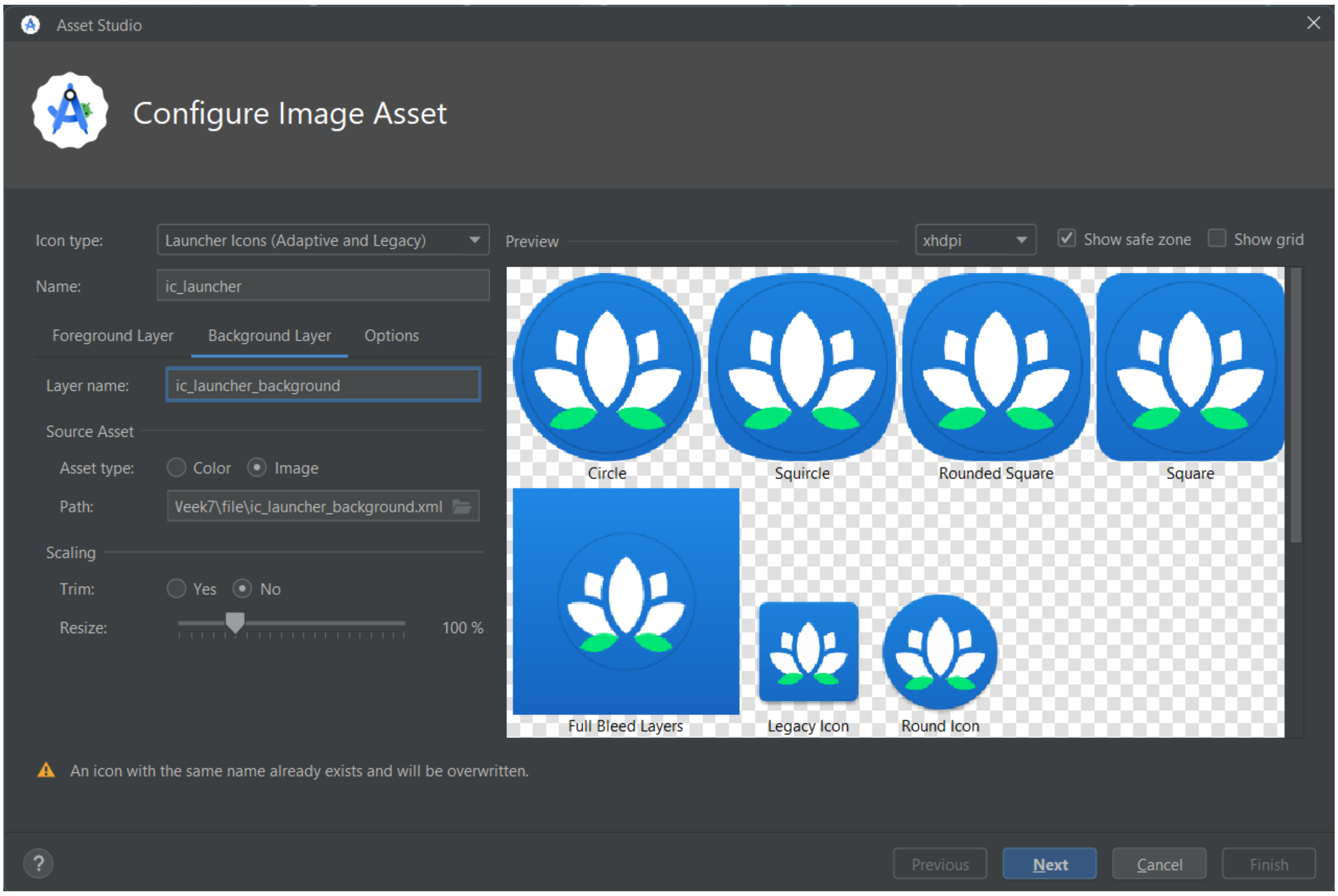
Resize: 100 %

Resize the original asset using the specified scaling factor (in percent). This happens after any trimming.

Preview: xhdpi [✓ Show safe zone] Show grid

An icon with the same name already exists and will be overwritten.

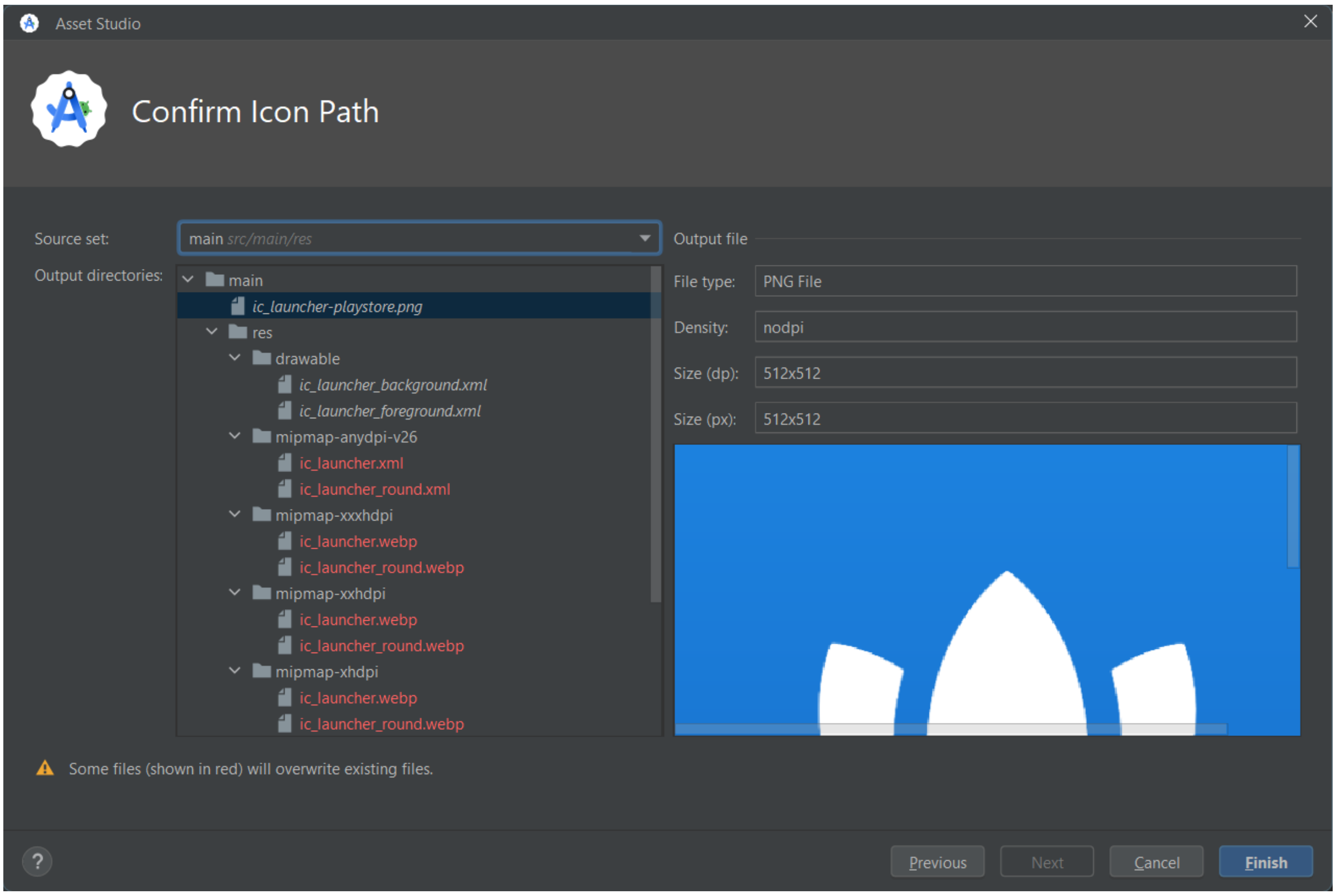
8. Switch to the Background Layer tab.
9. Click the Browse icon inside the Path text box.
10. Find and open the ic_launcher_background.xml file on your computer. No other changes are necessary.



Icon type: Launcher Icons (Adaptive and Legacy)
Name: ic_launcher
Layer name: ic_launcher_background
Source Asset
Asset type: Color [✓ Image]
Path: Veek7\file\ic_launcher_background.xml
Scaling
Trim: Yes [✓ No]
Resize: 100 %
Preview: xhdpi [✓ Show safe zone] Show grid

An icon with the same name already exists and will be overwritten.

11. Click Next.



Source set: main *src/main/res*
Output directories:

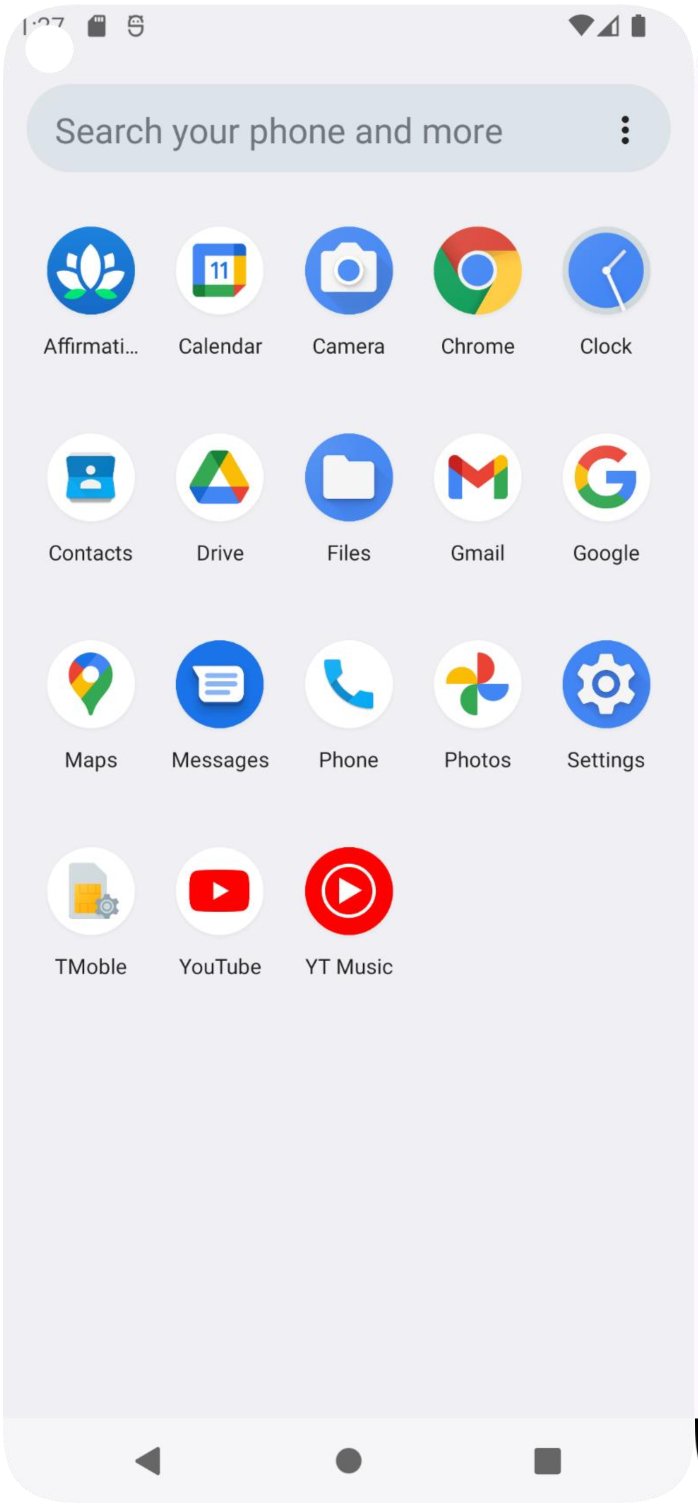
```
main
    ic_launcher-playstore.png
    res
        drawable
            ic_launcher_background.xml
            ic_launcher_foreground.xml
        mipmap-anydpi-v26
            ic_launcher.xml
            ic_launcher_round.xml
        mipmap-xxxhdpi
            ic_launcher.webp
            ic_launcher_round.webp
        mipmap-xxhdpi
            ic_launcher.webp
            ic_launcher_round.webp
        mipmap-xhdpi
            ic_launcher.webp
            ic_launcher_round.webp
```

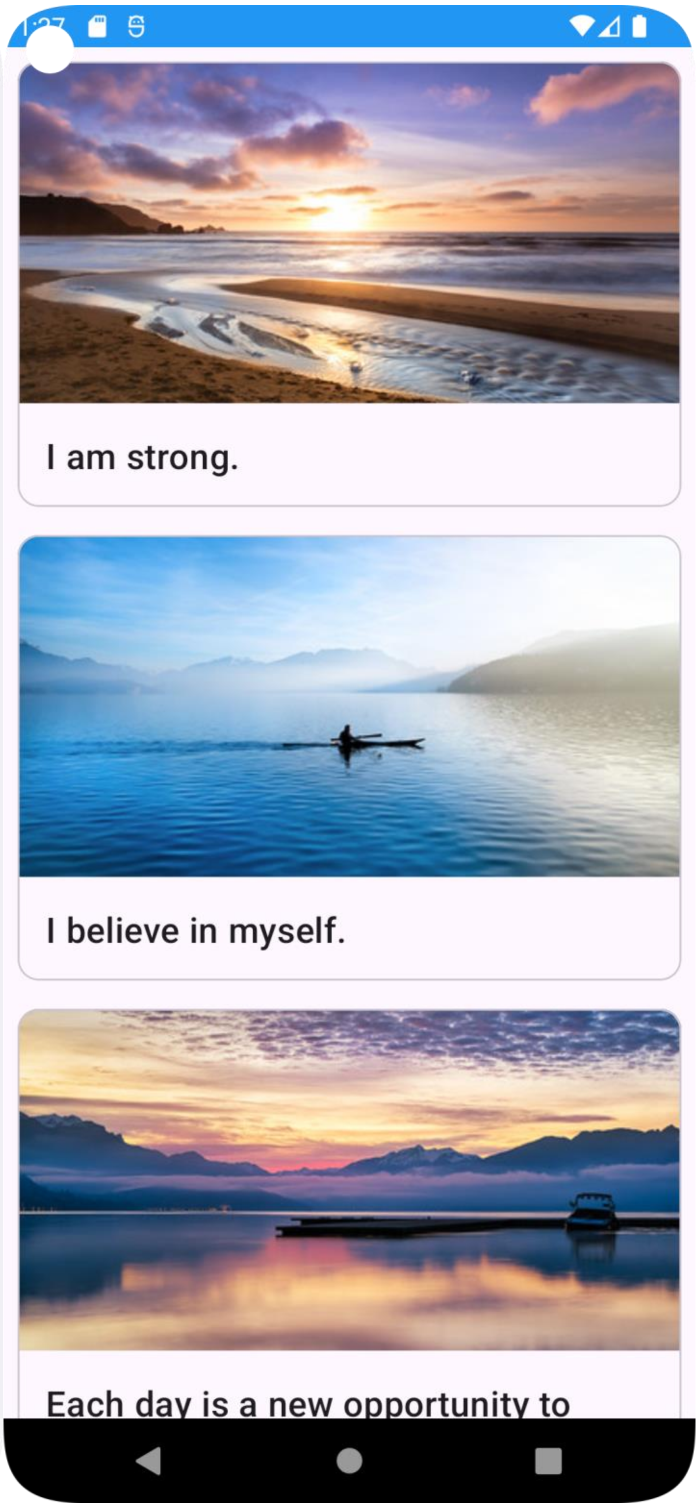
Output file
File type: PNG File
Density: nodpi
Size (dp): 512x512
Size (px): 512x512

Some files (shown in red) will overwrite existing files.

12. In the Confirm Icon Path dialog, click Finish. It's OK to overwrite the existing icons.

13. For best practices, you can move the new vector drawables ic_launcher_foreground.xml and ic_launcher_background.xml into a new resource directory called drawable-anydpi-v26. Adaptive icons were introduced in API 26, so these resources will only be used on devices running API 26 and above (for any dpi).
14. Delete the drawable-v24 directory if there's nothing left there.
15. Run your app and notice the beautiful new app icon in the app drawer!



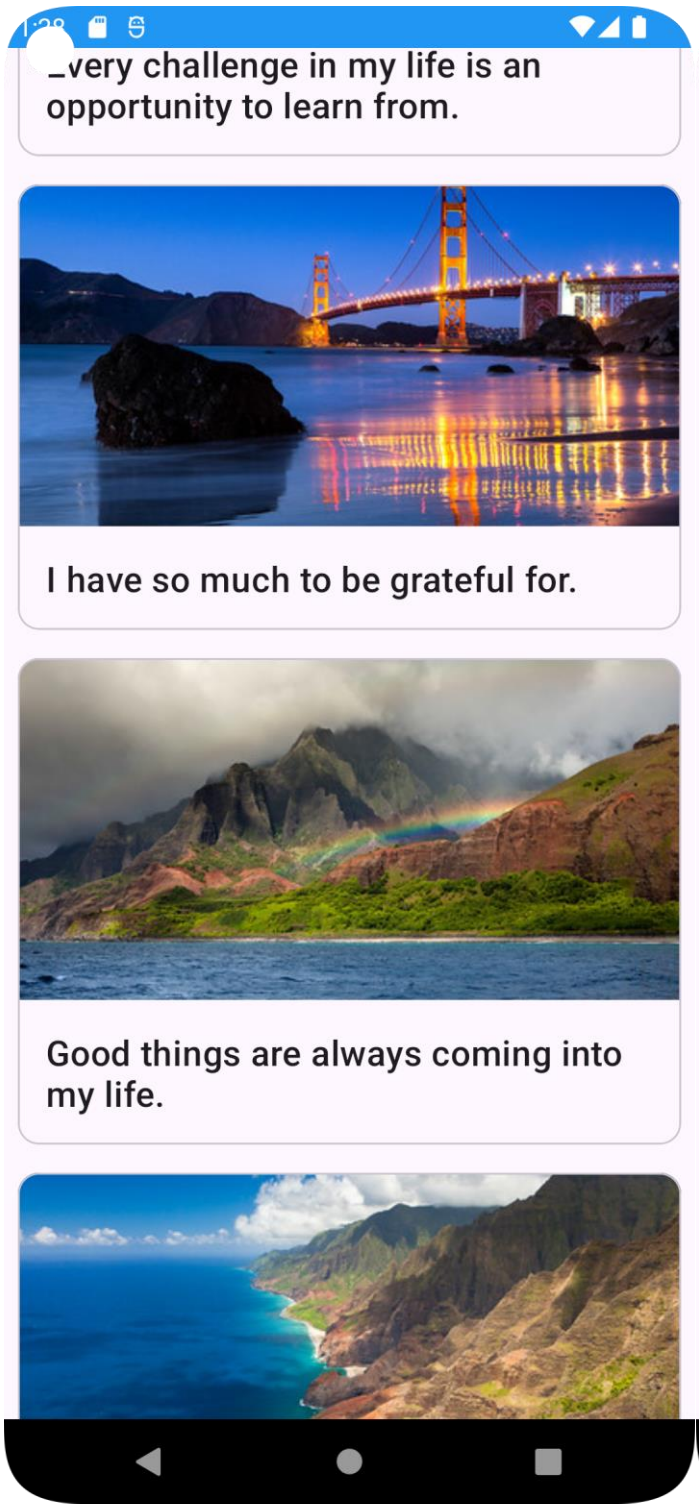


I am strong.

I believe in myself.

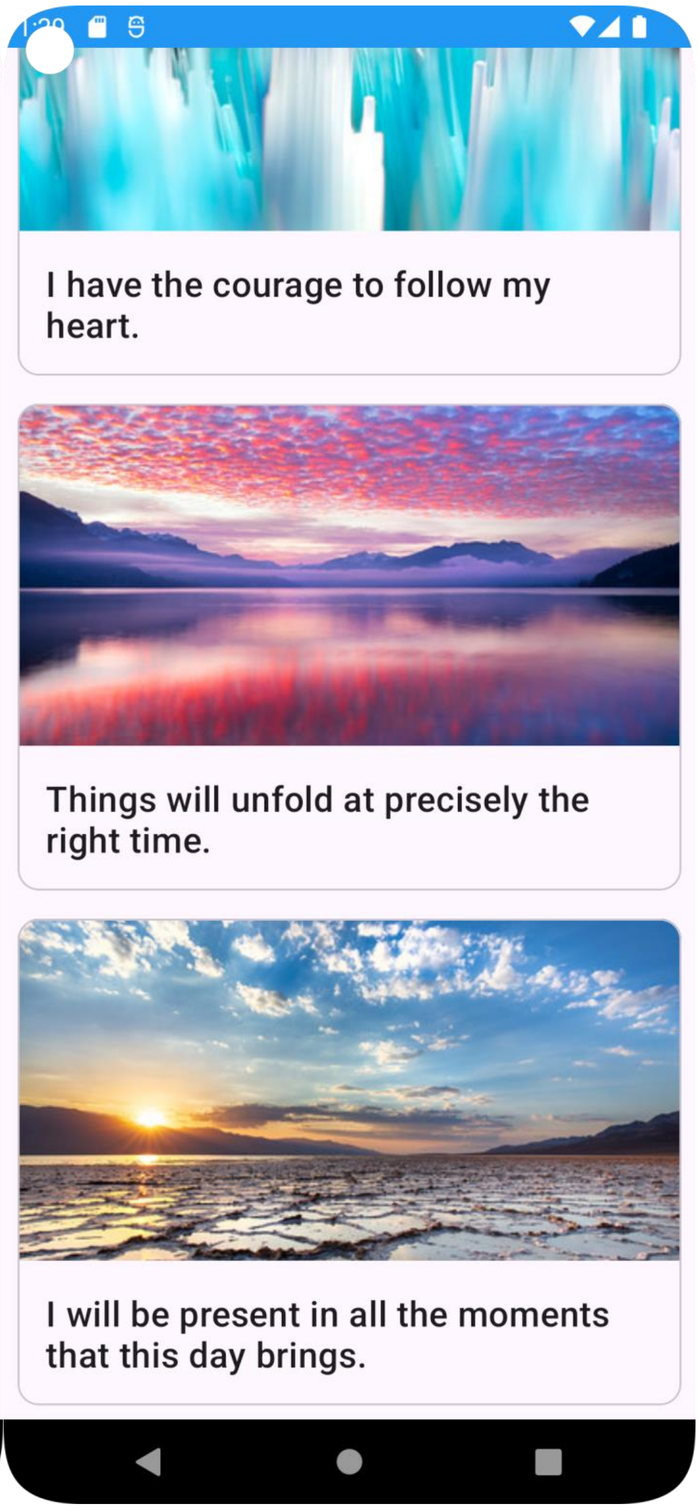
Each day is a new opportunity to

16. As a last step, don't forget to reformat the Kotlin and XML files in the project so your code is cleaner and follows style guidelines. Congratulations! You created an inspiring Affirmations app.


_very challenge in my life is an opportunity to learn from.

I have so much to be grateful for.

Good things are always coming into my life.


I have the courage to follow my heart.

Things will unfold at precisely the right time.

I will be present in all the moments that this day brings.

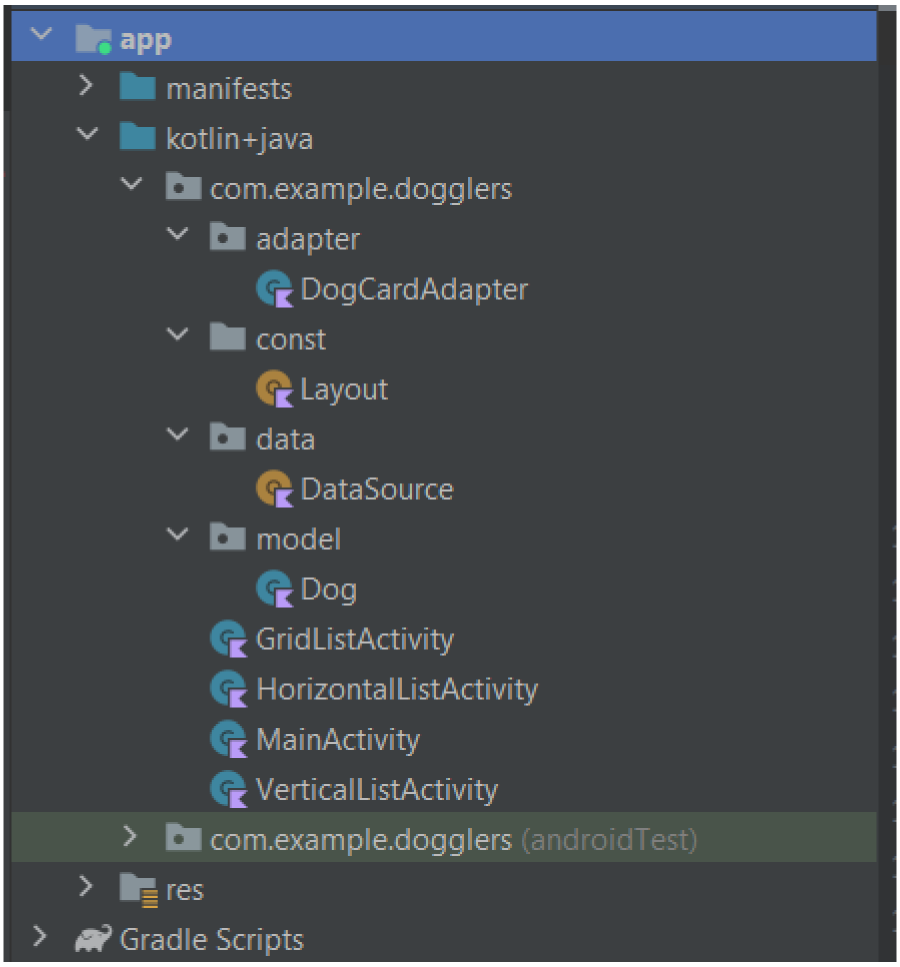
5.6 Project: Dogglers app

Get started

Open the project in Android Studio

1. Start Android Studio.
2. In the Welcome to Android Studio window, click Open.
3. In the file browser, navigate to where the unzipped project folder is located (likely in your Downloads folder).
4. Double-click on that project folder.

5. Wait for Android Studio to open the project.
6. Click the Run button to build and run the app. Make sure it builds as expected.



Implement the layout

1. Build the layout for vertical and horizontal lists.
2. Build the grid layout.

Implement the adapter

1. Define a variable or constant for the list of dog data. The list can be found in the data package in an object called DataSource, and looks like the following:

object DataSource { val dogs: List = listOf(... }

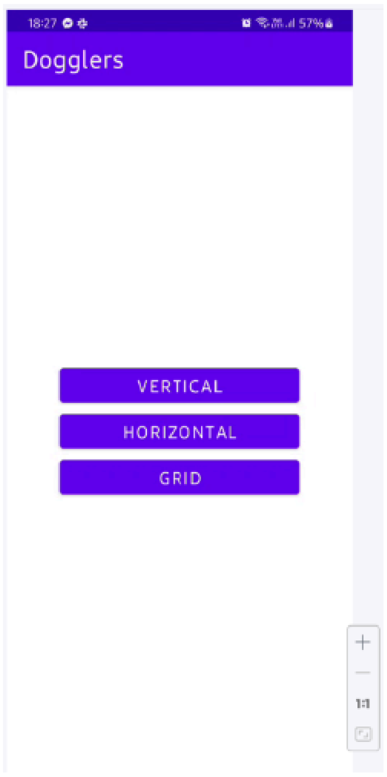
2. Implement the DogCardViewHolder. The view holder should bind the four views that need to be set for each recycler view card. The same view holder will be shared for both the grid_list_item and vertical_horizontal_list_item layouts, as all the views are shared between both layouts. The DogCardViewHolder should include properties for the following view IDs: dog_image, dog_name, dog_age, and dog_hobbies.
3. In onCreateViewHolder(), you want to either inflate the grid_list_item or vertical_horizontal_list_item layout. How do you know which layout to use? In the adapter's definition, you can see that a value called layout of type Int is passed in when creating an instance of the adapter.

class DogCardAdapter(private val context: Context?, private val layout: Int): RecyclerView.Adapter() {

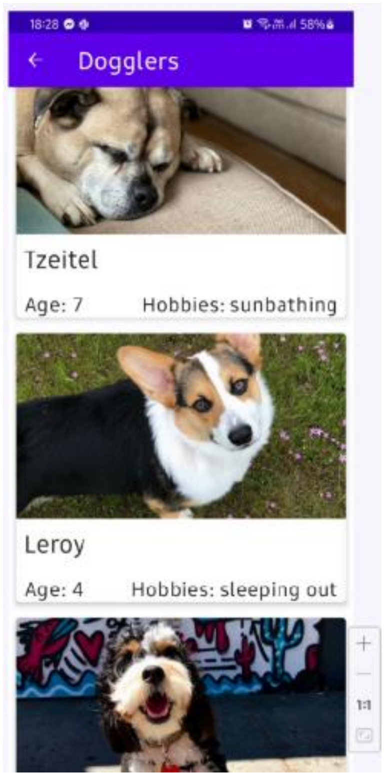
4. Implement getItemCount() to return the length of the list of dogs.

5. Finally, you need to implement onBindViewHolder() to set data in each of the recycler view cards. Use the position to access the correct dog data from the list, and set the image and dog name. Use the string resources, dog_age, and dog_hobbies to format the age and hobbies appropriately.

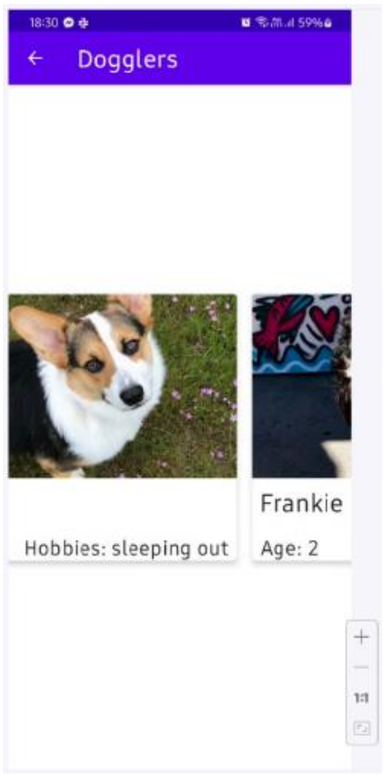
Test your app



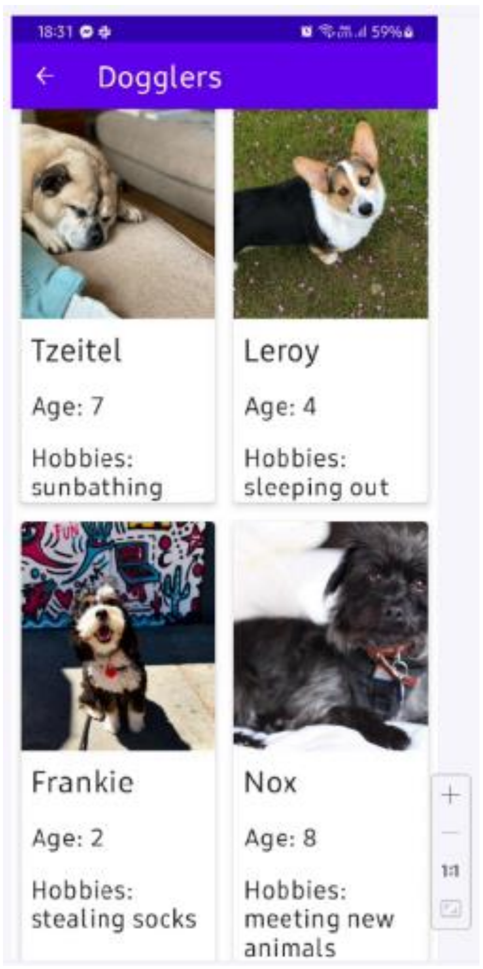
Vertical style



Horizontal style



Grid style


18:31
59%
Dogglers
Tzeitel
Age: 7
Hobbies: sunbathing
Leroy
Age: 4
Hobbies: sleeping out
Frankie
Age: 2
Hobbies: stealing socks
Nox
Age: 8
Hobbies: meeting new animals
1:1