PROJECT: DOGGLERS APP

1. Before you begin

This exercise introduces a new app called Dogglers that you'll build on your own. This exercise walks you through the steps to complete the Dogglers app project, including project setup and testing within Android Studio.

This exercise is meant to set up a project that you will complete independently, providing you with instructions on how to complete an app and check your work on your own.

There is a test suite provided as part of the app you'll download. You'll run these tests in Android Studio (steps to do this are shown later in this exercise) and see if your code passes. This may take a few tries—even professional developers rarely have all their tests pass on the first try! After your code passes all the tests, you can consider this project as complete.

This exercise require you to use different skills that you don't yet have a lot of practice with, such as:

- Googling terms, error messages, and bits of code in the app that you don't recognize;
- Testing code, reading errors, then making changes to the code and testing it again;
- Going back to previous content in Android Basics to refresh what you've learned;
- Comparing code that you know works (codes that you have done with the previous exercises) with the code you're writing.

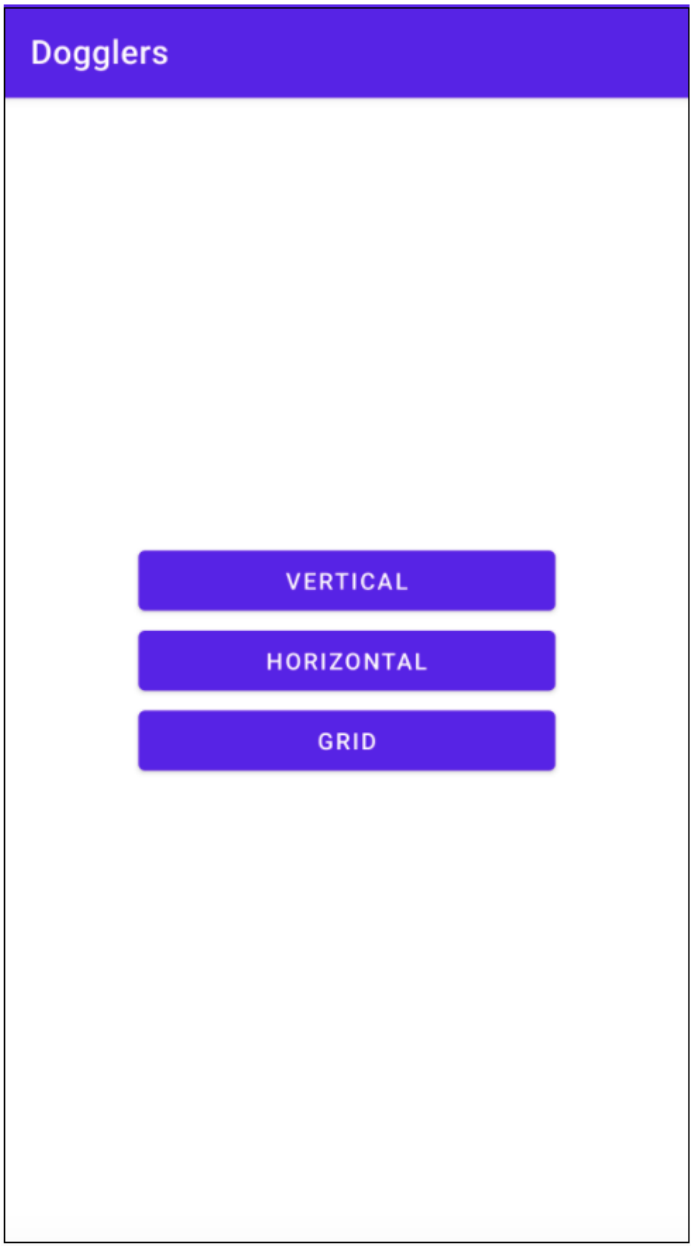
2. App overview

Welcome to Project: Dogglers app!

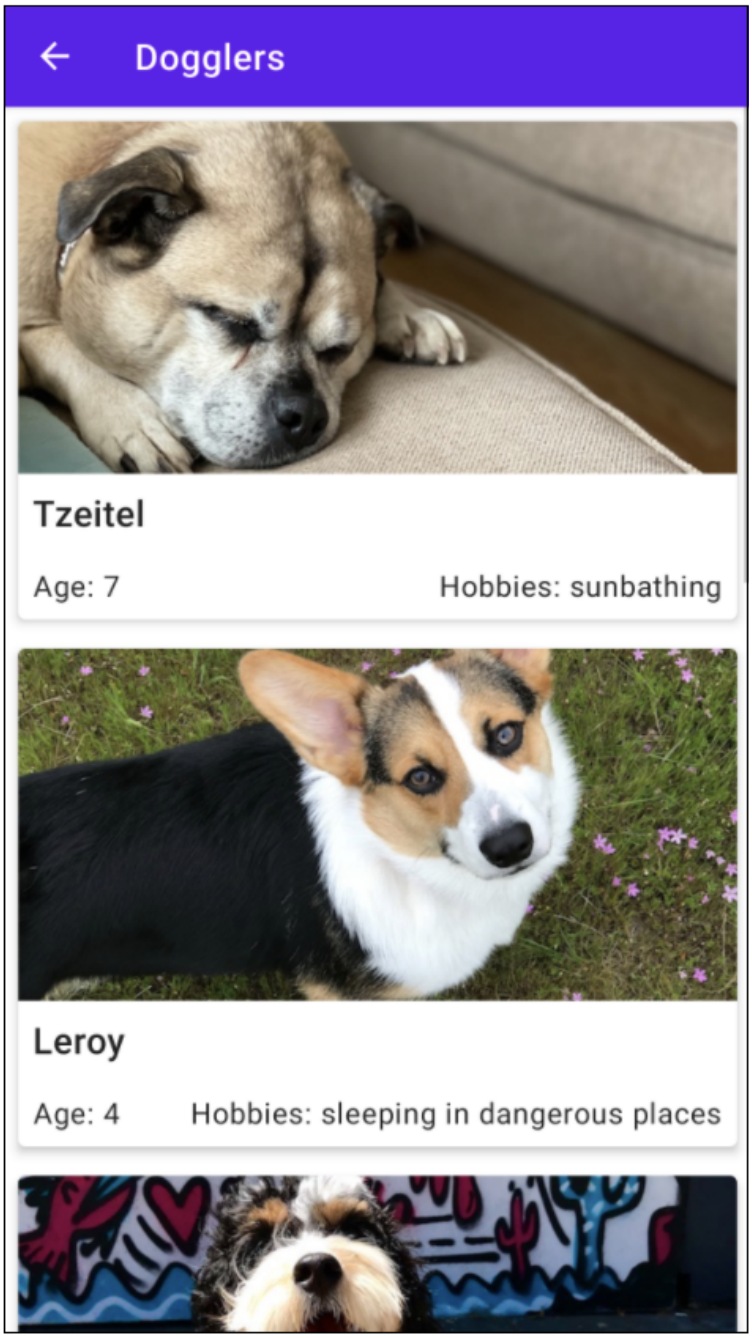
In this project, you will create an app for our canine friends, called Dogglers. Your task is to implement Dogglers, which shows scrolling lists of pet dogs along with a bit of information about each one, such as their name, age, hobbies, and a photo. In this project, you'll build layouts for the RecyclerView items in the Dogglers app, and

implement an adapter so that the list of dogs can be presented three ways: by horizontal scrolling, vertical scrolling, and vertically scrolling grid layout.

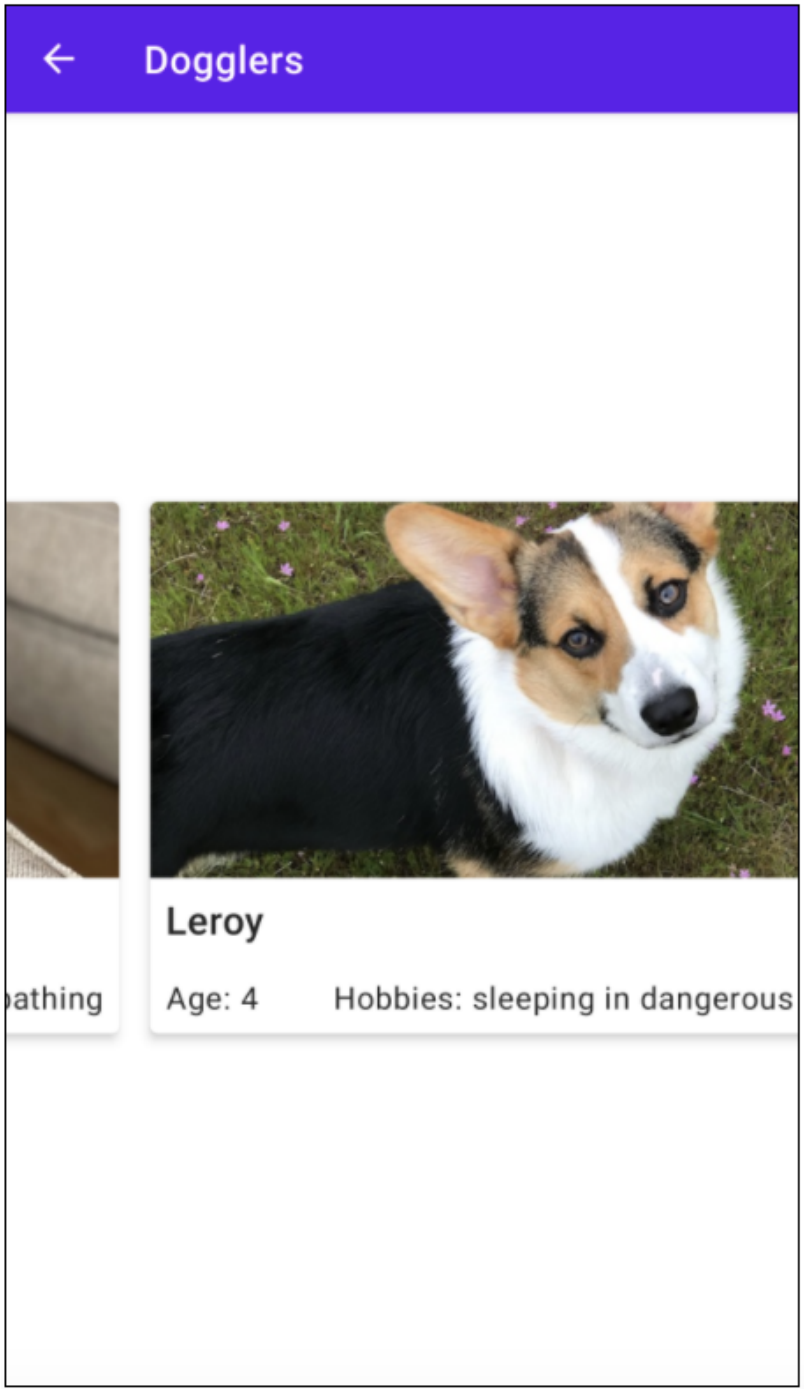
When you launch the app, you're provided with options for horizontal, vertical, and grid layouts.



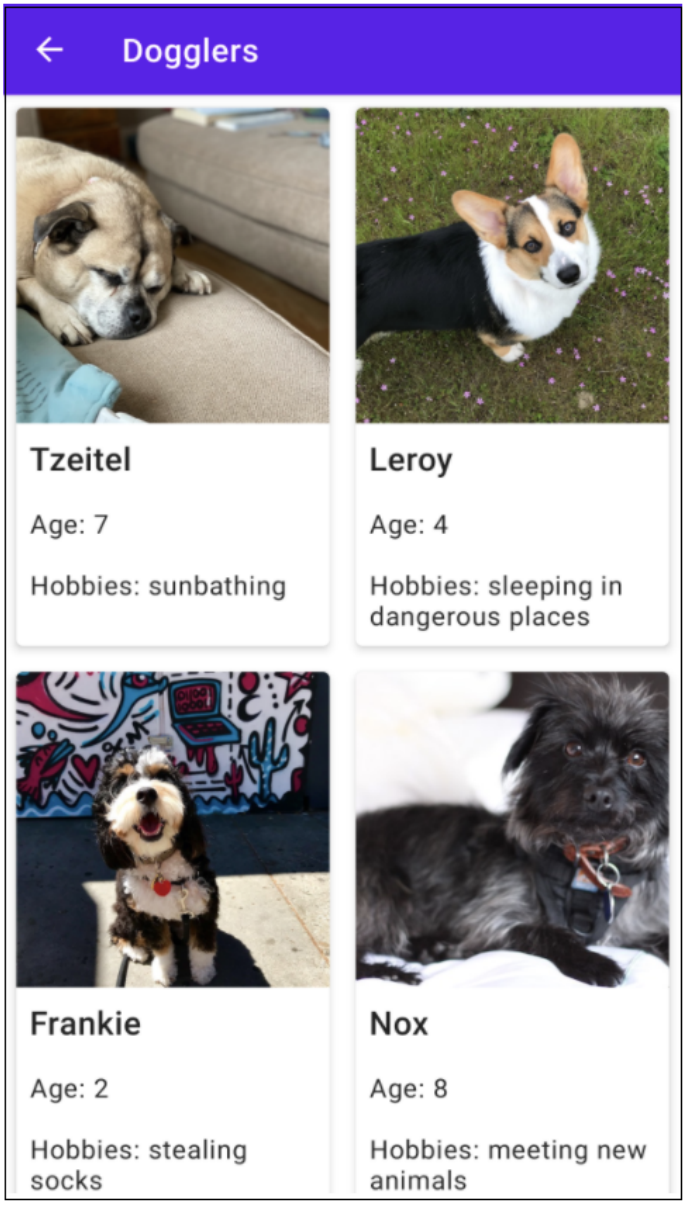
The first option is a vertical scrolling recycler view with items that take up the full width of the screen.



The second option shows the list of dogs in a horizontally scrolling recycler view.



The third option shows the dogs in a vertically scrolling grid-style layout where two dogs are shown on each row.



All of these layouts are powered by the same adapter class. Your task will be to build the layouts for the recycler view cards, and then implement the adapter so that each item is populated with the information about each dog.

3. Get started

Download the project code

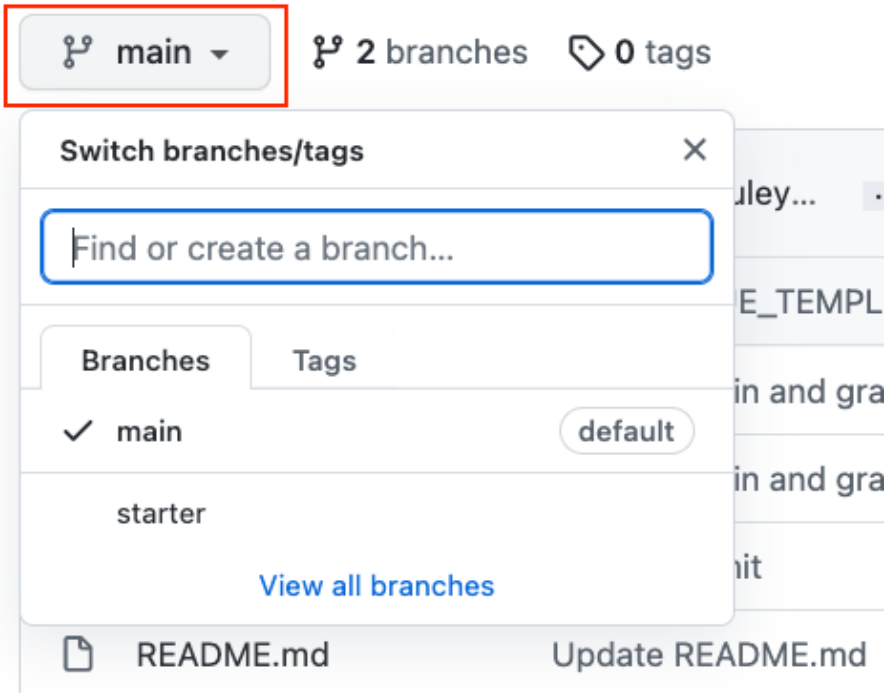
Note that the folder name is android-basics-kotlin-dogglers-app. Select this folder when you open the project in Android Studio.

Starter Code URL:

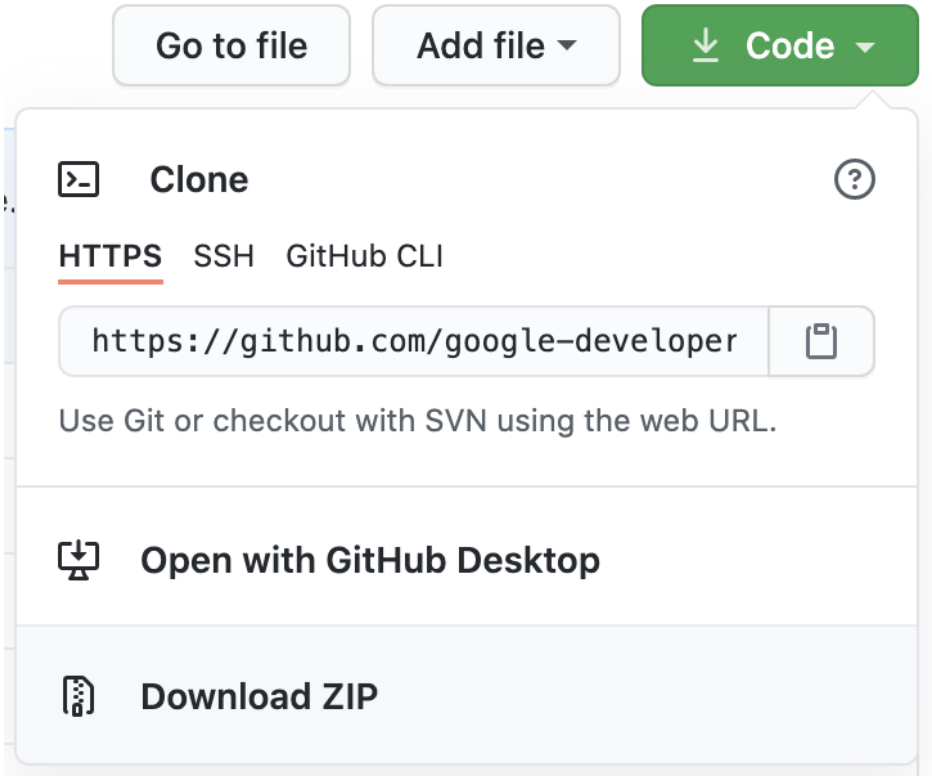
https://github.com/google-developer-training/android-basics-kotlin-dogglers-app/tree/main

Branch name with starter code: main

1. Navigate to the provided GitHub repository page for the project.
2. Verify that the branch name matches the branch name specified in the exercise. For example, in the following screenshot the branch name is **main**.



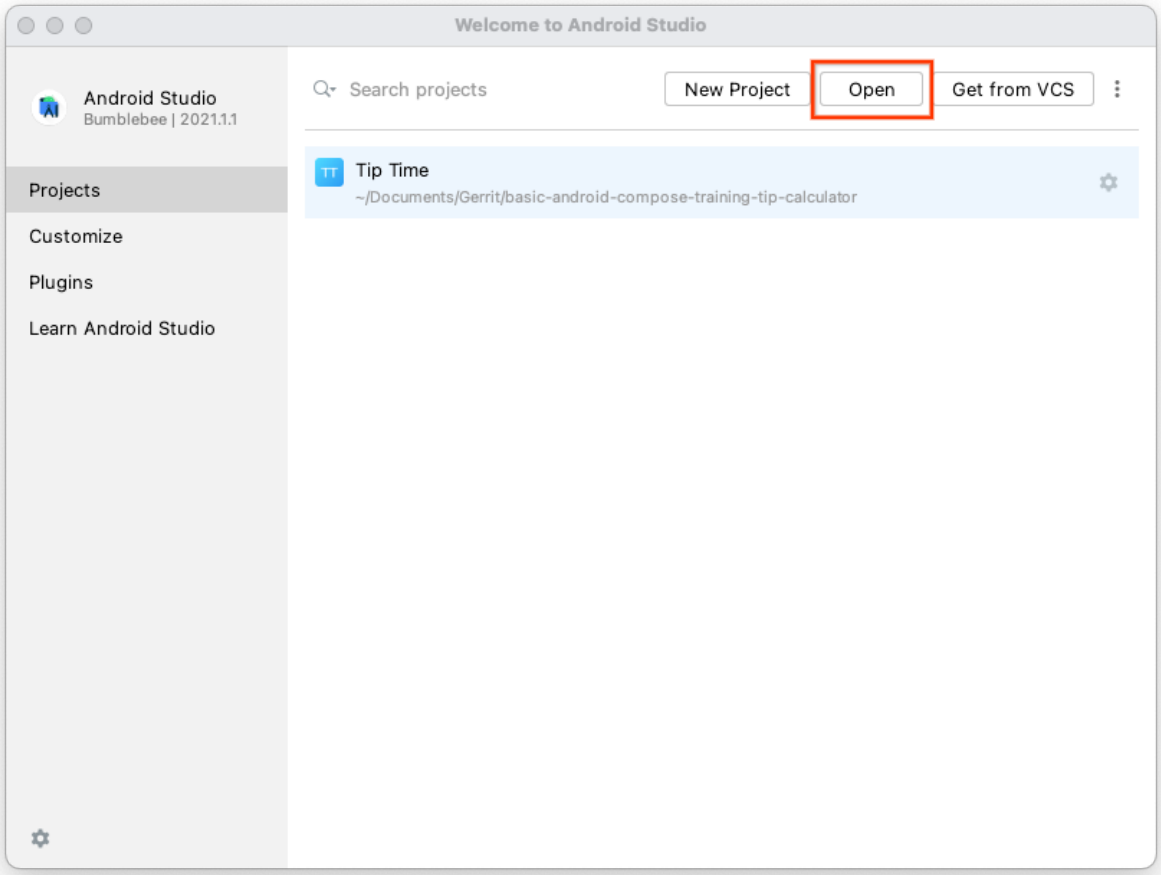
3. On the GitHub page for the project, click the **Code** button, which brings up a popup.



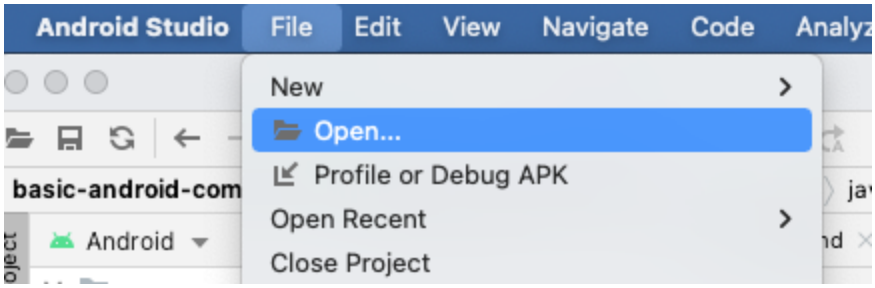
4. In the popup, click the **Download ZIP** button to save the project to your computer. Wait for the download to complete.
5. Locate the file on your computer (likely in the **Downloads** folder).
6. Double-click the ZIP file to unpack it. This creates a new folder that contains the project files.

Open the project in Android Studio

1. Start Android Studio.
2. In the **Welcome to Android Studio** window, click **Open**.



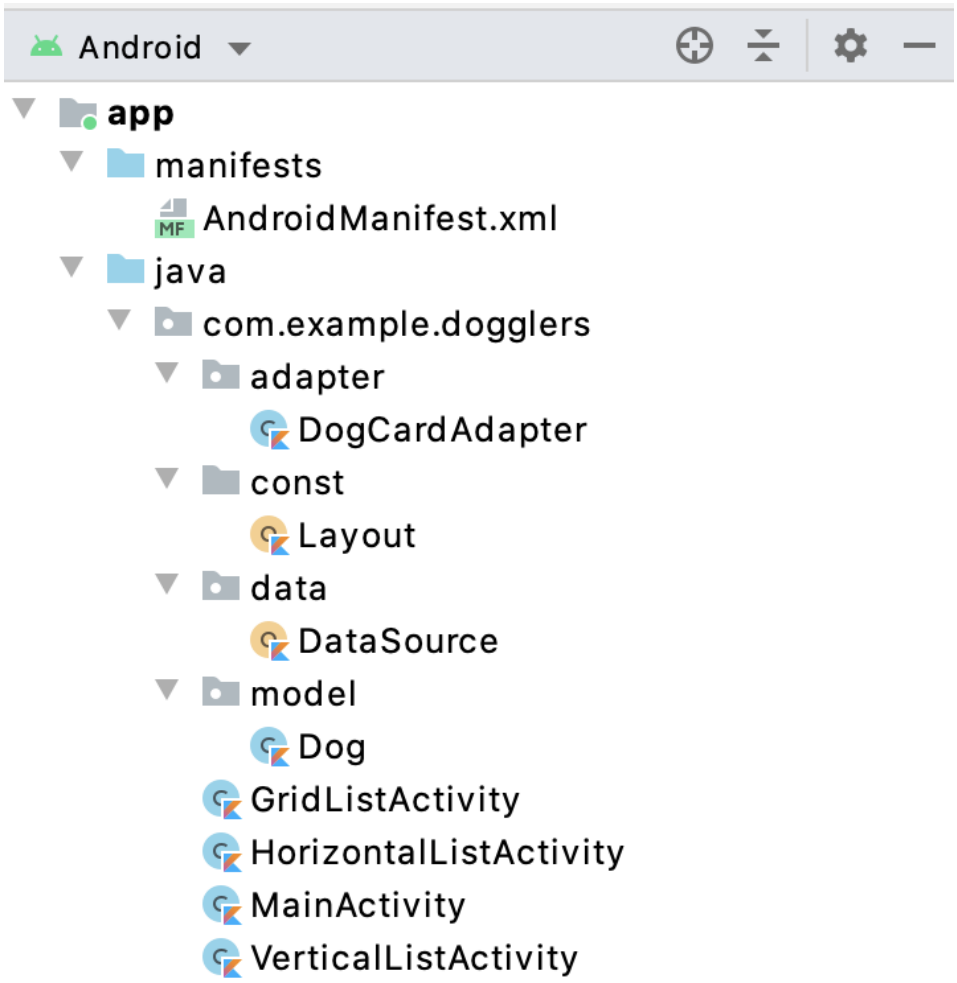
Note: If Android Studio is already open, instead, select the **File** > **Open** menu option.



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.

The project is organized into separate packages. While most of the functionality is already implemented, you'll need to implement the DogCardAdapter. There are also two layout files that you'll need to modify. Other files are discussed as needed in the following instructions.



Implement the layout

Both the vertical and horizontal layouts are identical, so you only need to implement a single layout file for both. The grid layout displays all the same information, but the dog's name, age, and hobbies are stacked vertically, so you'll need a separate layout for this case. Both layouts require four different views to display information about each dog.

1. An ImageView with the dog's picture
2. A TextView with the dog's name
3. A TextView with the dog's age
4. A TextView with the dog's hobbies

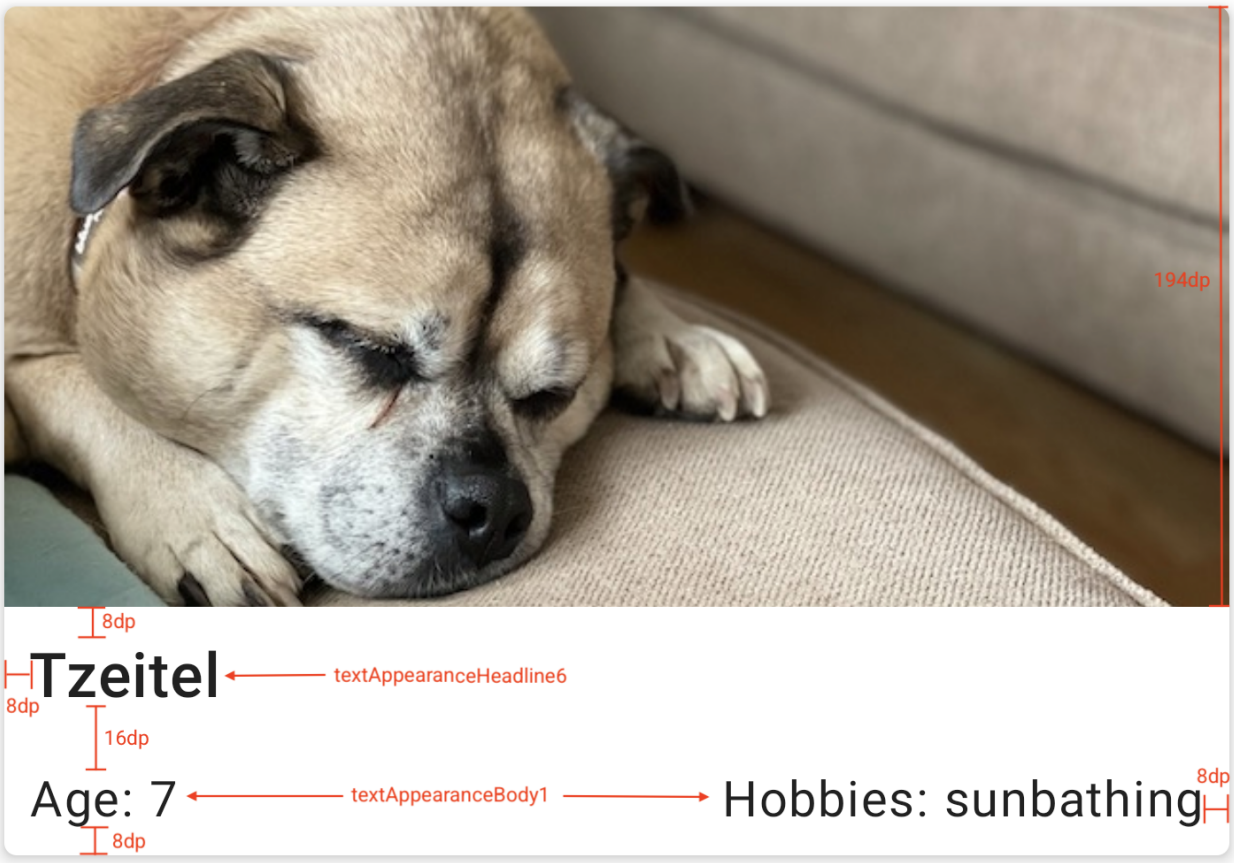
You'll also notice some styling on each card to show a border and a shadow. This is handled by MaterialCardView, which is already added to the layout files in the starter project. Within each MaterialCardView is a ConstraintLayout where you'll need to add the remaining views.

Tip: You can use any ID of your choice for each view, but keep in mind that both of these layouts will use the same ViewHolder class, so you'll want to make sure that corresponding views in each layout use the same ID. For example, both the grid layout and horizontal/vertical layout would have a TextView with the id dog_name.

There are two XML files you'll need to work with to implement the layouts: vertical_horizontal_list_item.xml for the horizontal and vertical layouts, and grid_list_item.xml for the grid layout.

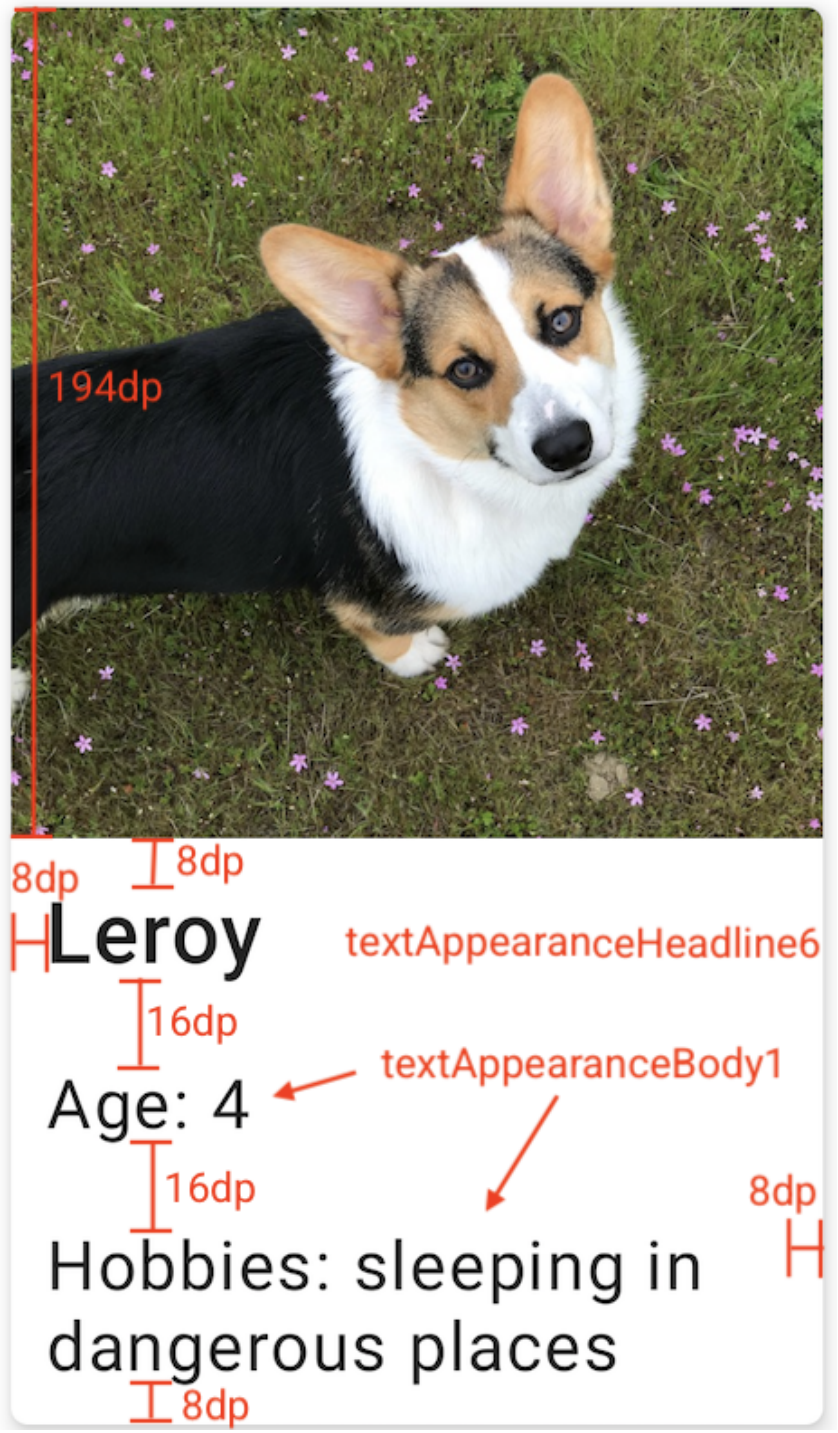
1. Build the layout for vertical and horizontal lists.

Open up vertical_horizontal_list_item.xml, and in the inner ConstraintLayout, build the layout to match the image.



2. Build the grid layout.

Open up grid_list_item.xml, and in the inner ConstraintLayout, build the layout to match the image.



Implement the adapter

Once you've defined your layouts, your next task is to implement the RecyclerView adapter. Open up DogCardAdapter.kt in the **adapter** package. You'll see there are lots of TODO comments that help explain what you need to implement.

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

    // TODO: Initialize the data using the List found in data/DataSource

    /**
     * Initialize view elements
     */
    class DogCardViewHolder(view: View?): RecyclerView.ViewHolder(view!!) {
        // TODO: Declare and initialize all of the list item UI components
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): DogCardViewHolder {
        // TODO: Use a conditional to determine the layout type and set it accordingly.
        //  if the layout variable is Layout.GRID the grid list item should be used. Otherwise the
        //  the vertical/horizontal list item should be used.

        // TODO Inflate the layout
```

There are five steps you'll need to 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<Dog> = listOf( ...

}
```

The dogs property is of type List<Dog>. The Dog class is found in the **model** package, and defines four properties: an image (represented by a resource ID), and three String properties.

```
data class Dog(
    @DrawableRes val imageResourceId: Int,
    val name: String,
    val age: String,
    val hobbies: String
)
```

Set the variable you define in DogCardAdapter to the dogs list in the DataSource object.

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<DogCardAdapter.DogCardViewHolder>() {
```

This corresponds to a value defined in the Layout object, located in the **const** package.

```
object Layout {
    val VERTICAL = 1
    val HORIZONTAL = 2
    val GRID = 3
}
```

The value of layout will either be 1, 2, or 3, but you can check it against the identifiers VERTICAL, HORIZONTAL, and GRID, from the Layout object.

For the GRID layout, inflate the grid_list_item layout, and for HORIZONTAL and VERTICAL layouts, inflate the vertical_horizontal_list_item layout. The method should

return a DogCardViewHolder instance representing the inflated layout.

Tip: You can use a conditional statement, like if or when, to set a variable based on the layout type (grid_list_item for GRID, and vertical_horizontal_list_item for VERTICAL or HORIZONTAL). Once you have the correct layout ID, simply pass it into the method to inflate the layout.

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.

Tip: In the onBindViewHolder() method, we've defined a resources variable that you can use to reference string resources, without using context?.resources every time.

Once you've finished implementing the adapter, run your app on the emulator to verify that everything is implemented correctly.

3. Additional Task

For the images of the dogs to be displayed in the Doggler's app, use the images provided for you (dogs.zip) and the details below.

Dogs Name	Age	Hobbies	Filename
Nala	3	Strolling in the garden	Dog1.jpg
Kali	1	Sunbathing	Dog2.jpg
Snow	2	Playing in the snow	Dog3.jpg
Jake	1	Surfing	Dog4.jpg
Oreo	1	Playing tennis ball	Dog5.jpg
Rollo	4	Reading	Dog6.jpg
Chase	3	Playing frisbee	Dog7.jpg

Finn	2	Swimming	Dog8.jpg

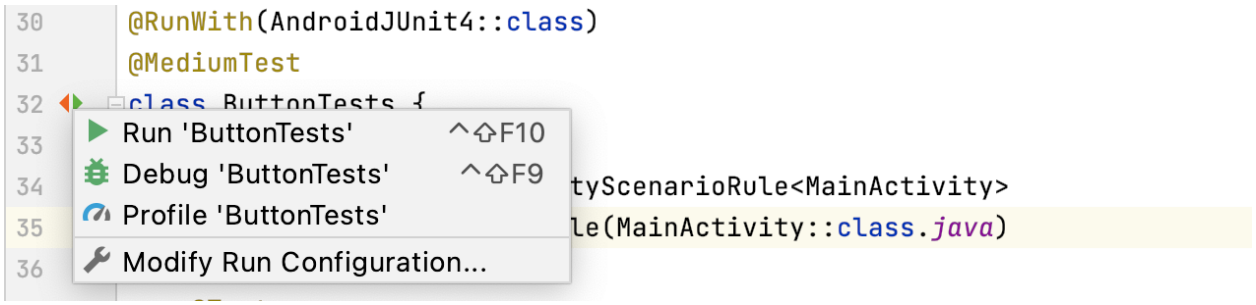
4. Test your app

The Dogglers project contains an "androidTest" target with several test cases.

Running your tests

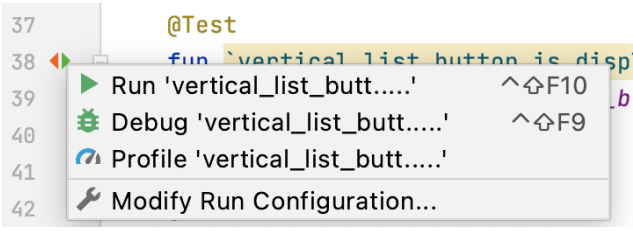
To run your tests, you can do one of the following:

For a single test case, open up a test case class and click the green arrow to the left of the class declaration. You can then select the Run option from the menu. This will run all of the tests in the test case.

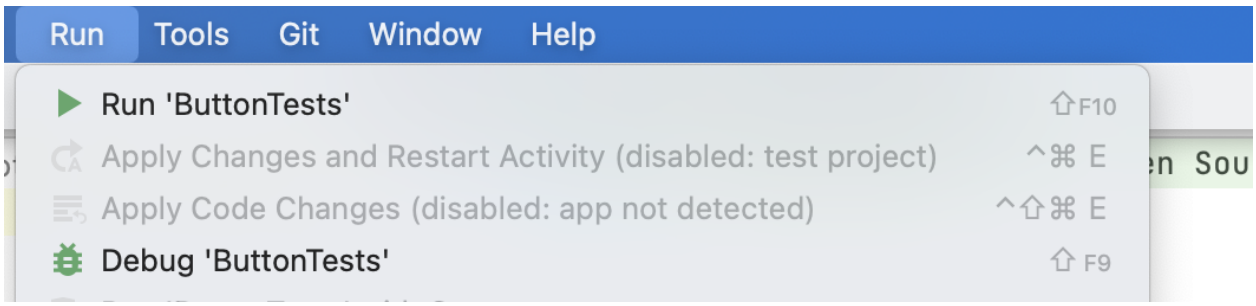


Often you'll only want to run a single test, for example, if there's only one failing test and the other tests pass. You can run a single test just as you would the entire test case.

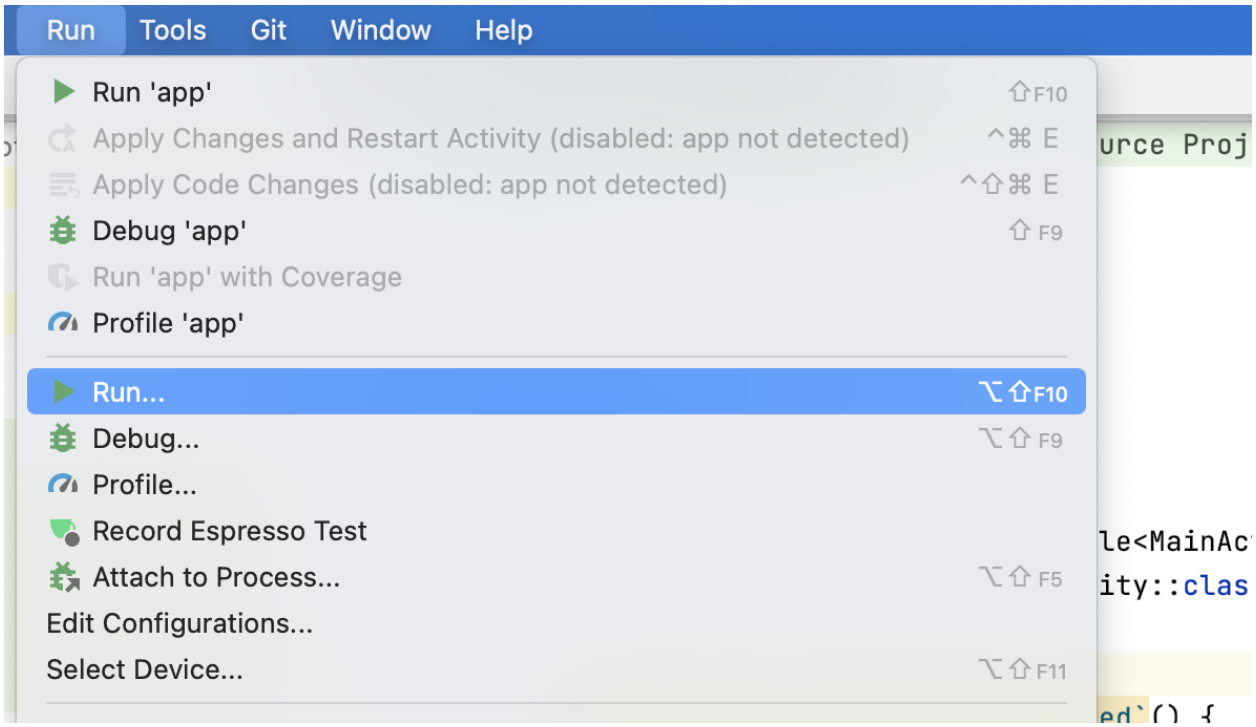
Use the green arrow and select the **Run** option.



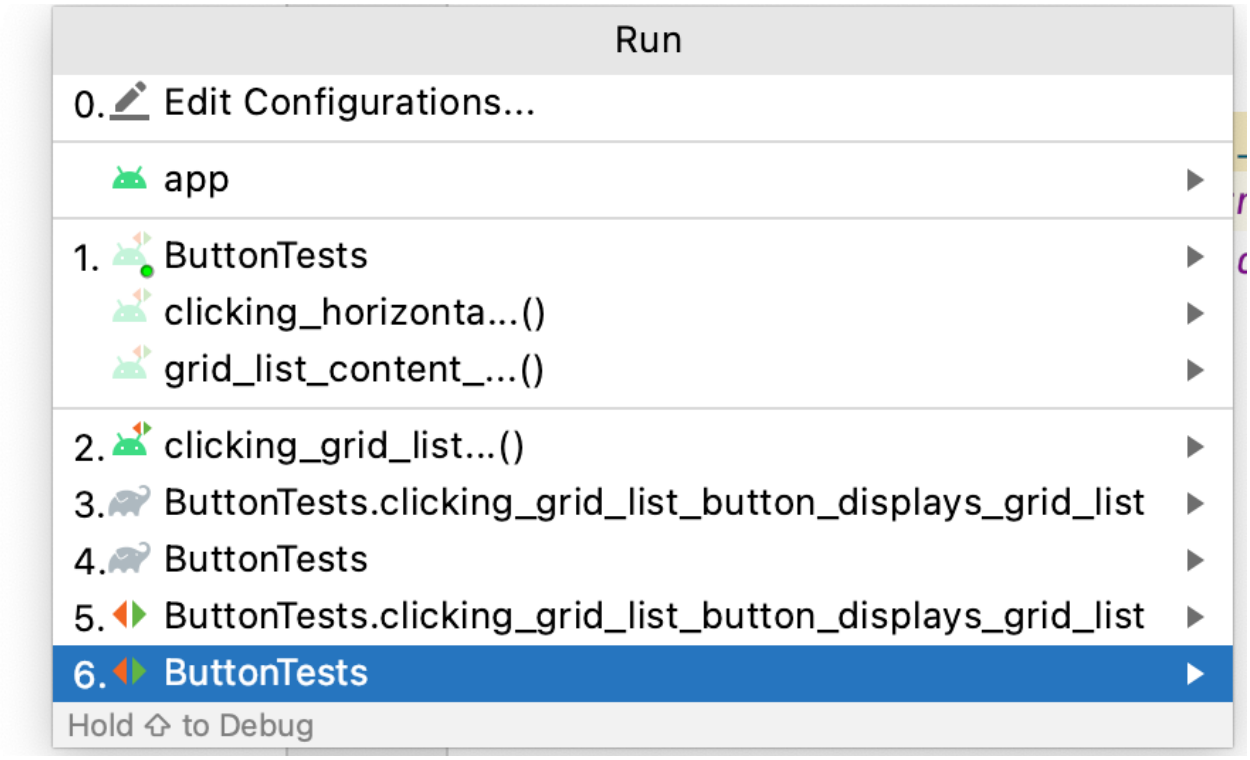
If you have multiple test cases, you can also run the entire test suite. Just like running the app, you can find this option on the **Run** menu.



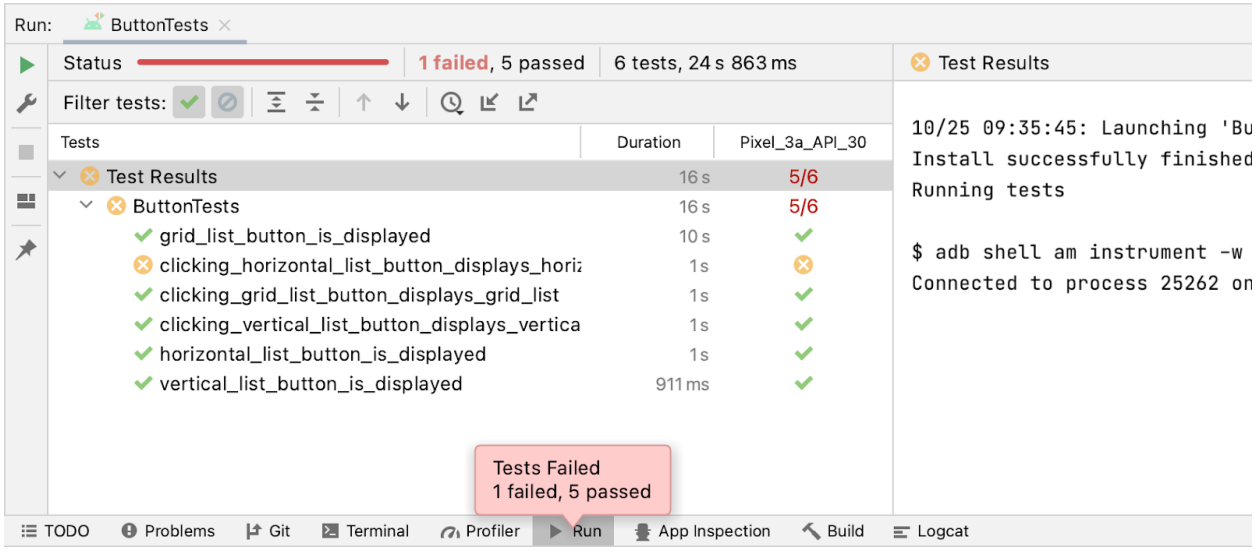
Note that Android Studio will default to the last target that you ran (app, test targets, etc.) so if the menu still says **Run > Run 'app'**, you can run the test target, by selecting **Run > Run**.



Then choose the test target from the popup menu.

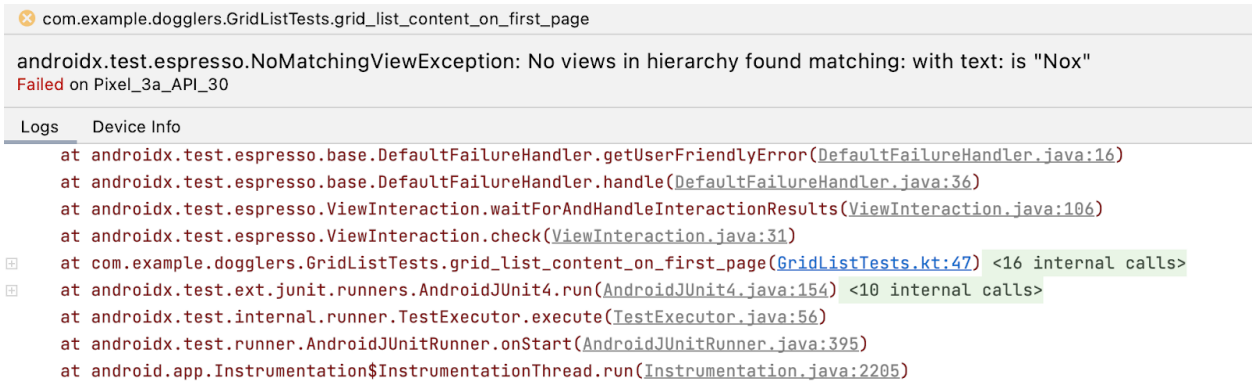


The results of running the tests are shown on the **Run** tab. In the pane on the left, you'll see a list of failing tests, if any. Failing tests are marked with a red exclamation point next to the function name. Passing tests are marked with a green check mark.



Tip: To display passing, failing, or both passing and failing tests in the left pane, you can use the two buttons at the top left. Selecting the checkmark will show passing tests, selecting the icon of a circle with a line through it will list the failing tests. By default, only the failing tests are listed.

If a test fails, the text output provides information to help you fix the problem that caused the test to fail.



For example, in the above error message, the test is checking if a string containing the word "Nox" is displayed. However, the test is failing. The text could not be found, meaning it is likely not being displayed yet.