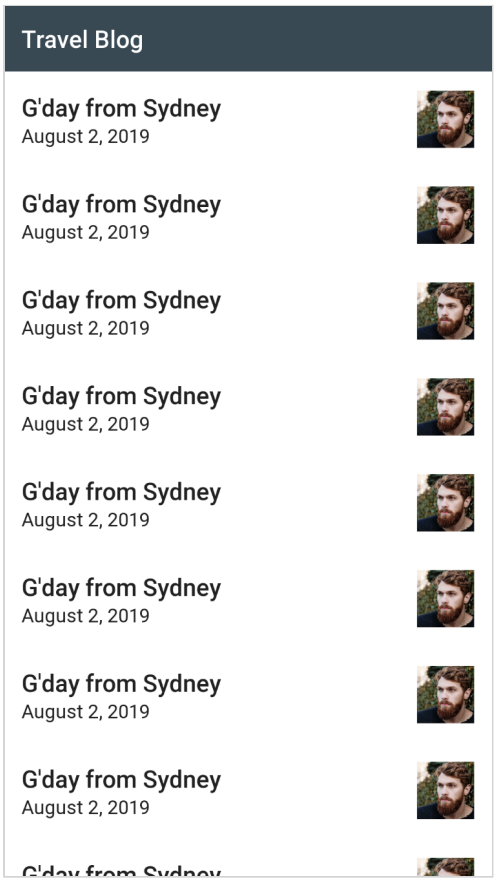
Building Layout

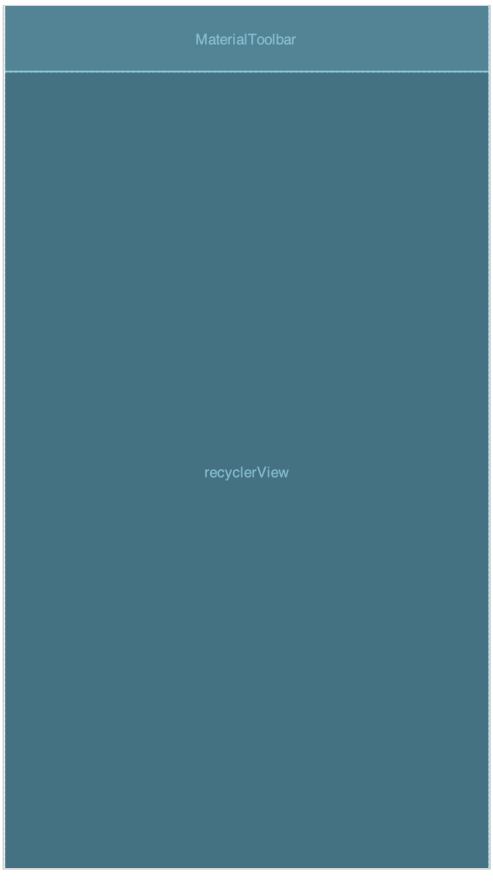
This lesson will cover how to create a layout for our blog list screen.

WE'LL COVER THE FOLLOWING

Final result preview

To make it easier to understand what we want to achieve, here is a preview of the layout which we are going to build.





List item layout

Let's start by creating a list item that is going to be a separate layout file.

Create a new *item_main.xml* layout file inside *app/src/main/res/layout* folder. As a root layout, we are going to use `LinearLayout`, which displays its children either horizontally or vertically depending on the `orientation` attribute value.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:padding="16dp">

...

</LinearLayout>
```

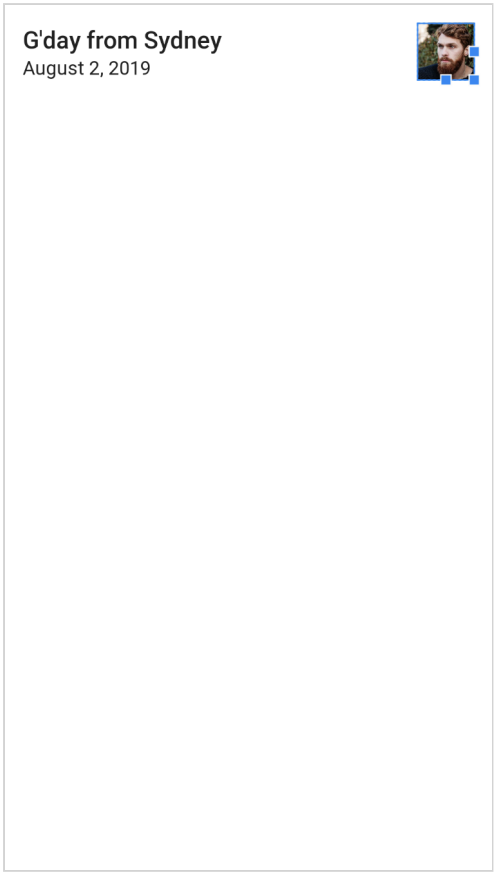
Next, let's add `ImageView` to display the avatar and another `LinearLayout` with `layout_width="0dp"` and `android:layout_weight="1.0"` attributes to push the `ImageView` to the right.

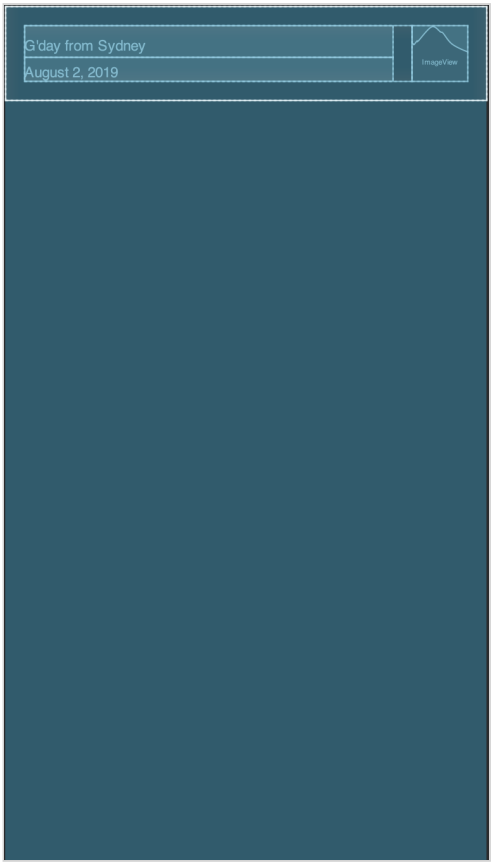
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:padding="16dp">

    <LinearLayout
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1.0">
    ...

    </LinearLayout>

    <ImageView
            android:id="@+id/imageAvatar"
            android:layout_width="48dp"
            android:layout_height="48dp"
            android:scaleType="centerCrop"
            tools:src="@drawable/avatar" />

</LinearLayout>
```

Finally, let's add two more `TextView`'s to the `LinearLayout` with the `android:orientation="vertical"` attribute to display a title and a date vertically.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:padding="16dp">

    <LinearLayout
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1.0"
            android:orientation="vertical">

        <TextView
                android:id="@+id/textTitle"
                style="@style/TextAppearance.MaterialComponents.Headline6"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginRight="16dp"
                tools:text="G'day from Sydney" />

        <TextView
                android:id="@+id/textDate"
                style="@style/TextAppearance.MaterialComponents.Body1"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginRight="16dp"
                tools:text="August 2, 2019" />

    </LinearLayout>

    <ImageView
            android:id="@+id/imageAvatar"
            android:layout_width="48dp"
            android:layout_height="48dp"
            android:scaleType="centerCrop"
            tools:src="@drawable/avatar" />

</LinearLayout>
```

Here is a preview of the layout.





Main layout

Now that we have a list item layout, we can modify *activity_main.xml* layout. The main layout is going to consist of two main elements: the toolbar and the list.

Android toolbar, also known as top app bar, is a material design component that displays information and actions relating to the current screen.

While Android SDK provides Toolbar view, we are going to use `MaterialToolbar` from Material Components library because of the richer API and better visual parity with Material Design. The toolbar is often used in pairs with `AppBarLayout` which provides several scrolling features.

Let's define `AppBarLayout` and its child `MaterialToolbar` with two `app` namespace attributes:

- `title` attribute specifies the toolbar title, in our case *Travel Blog*
- `titleTextColor` attribute specifies toolbar title color, in our case white

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/
        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">

    <com.google.android.material.appbar.AppBarLayout
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintLeft_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent">

        <com.google.android.material.appbar.MaterialToolbar
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                app:title="Travel Blog"
                app:titleTextColor="@android:color/white" />

    </com.google.android.material.appbar.AppBarLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```

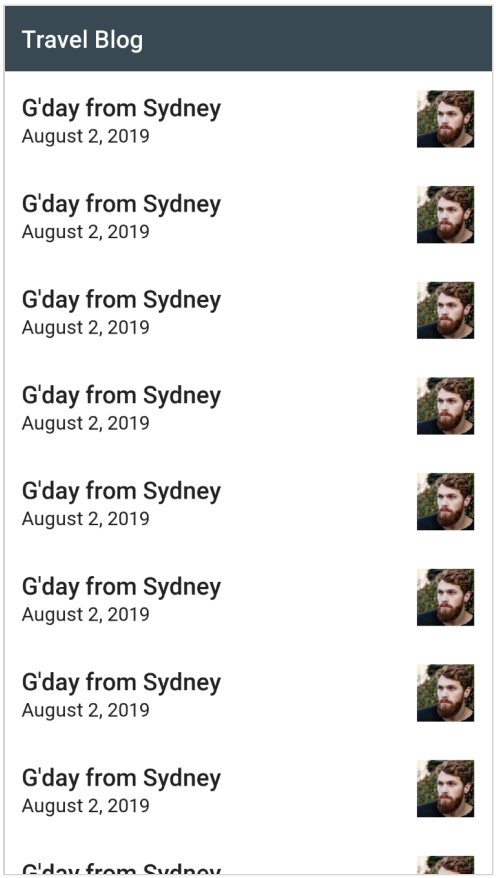
Now we can define our view to render a list; in Android, it's a `RecyclerView`. It's called a `RecyclerView` because it *recycles (reuses)* the list items which are not visible to the user during scrolling. To do so `RecyclerView` has an adapter class which we are going to create in the next chapter.

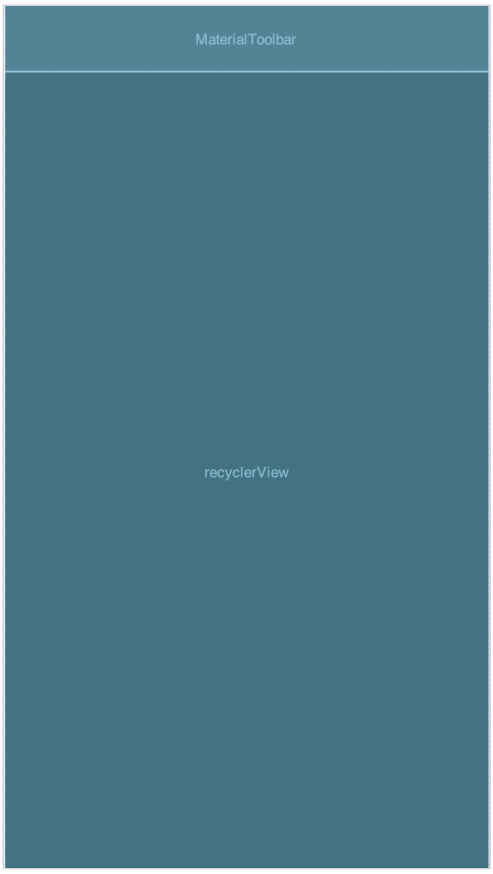
Right now, let's add a `RecyclerView` below the toolbar and stretch it to all available space. By default, our `RecyclerView` is going to be empty in the AndroidStudio preview, but we can use `listitem="@layout/item_main"` attribute from `tools` namespace to fill it with preview items.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/
        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"
        android:orientation="vertical">

<com.google.android.material.appbar.AppBarLayout
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintLeft_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <com.google.android.material.appbar.MaterialToolbar
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                app:title="Travel Blog"
                app:titleTextColor="@android:color/white" />
```

```
</com.google.android.material.appbar.AppBarLayout>

<androidx.recyclerview.widget.RecyclerView

        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintLeft_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/toolbar"
        tools:listitem="@layout/item_main" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Here is a preview of the layout.





The next lesson will go over the recycler view adapter implementation to render the list of blog articles.