
SUPER
SWEET
ANDROID
TIME

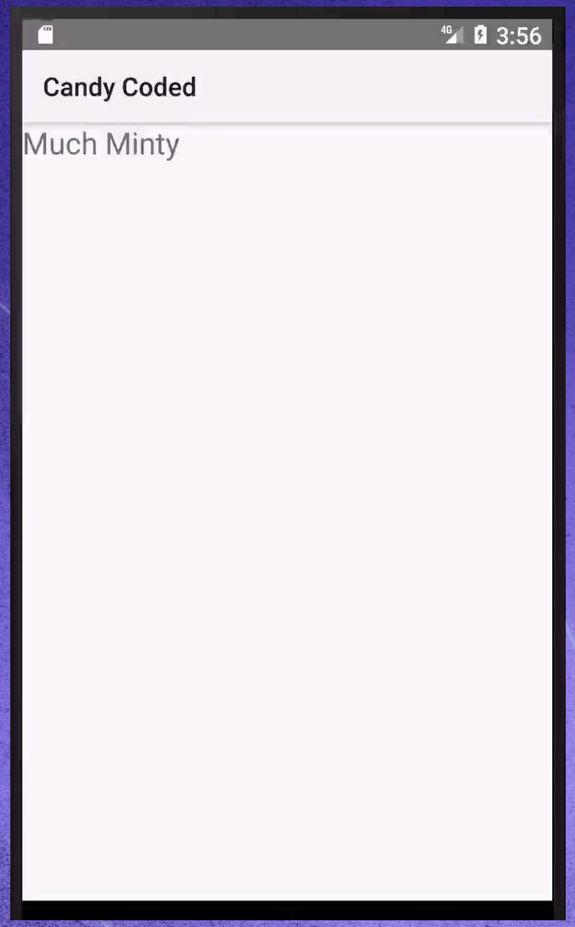
Level 3 – Section 1

The Detail Activity Layout

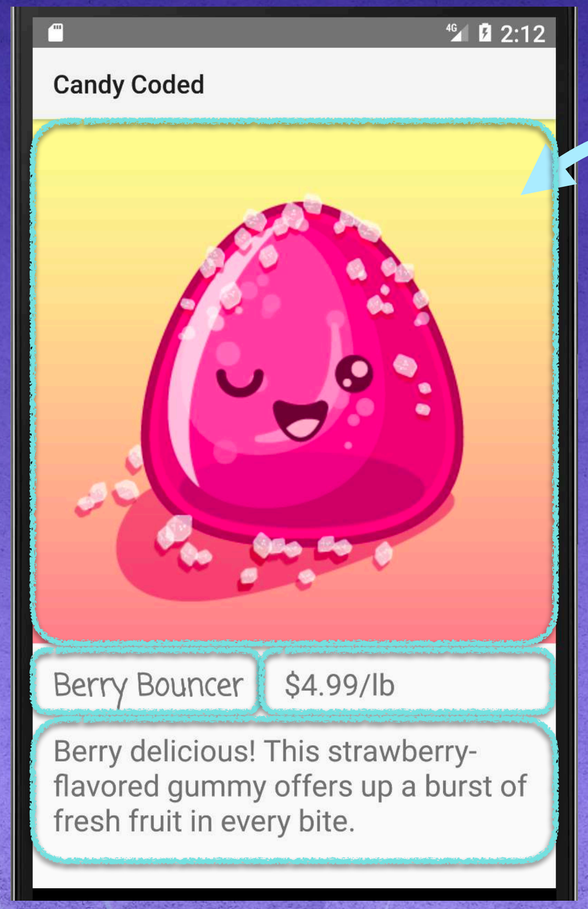
The Constraint Layout

What We Want DetailActivity's Layout to Look Like

Right now the Detail Activity **is only showing the Candy Name, we want to add** TextView's **for the price and description and an** ImageView **for the image.**



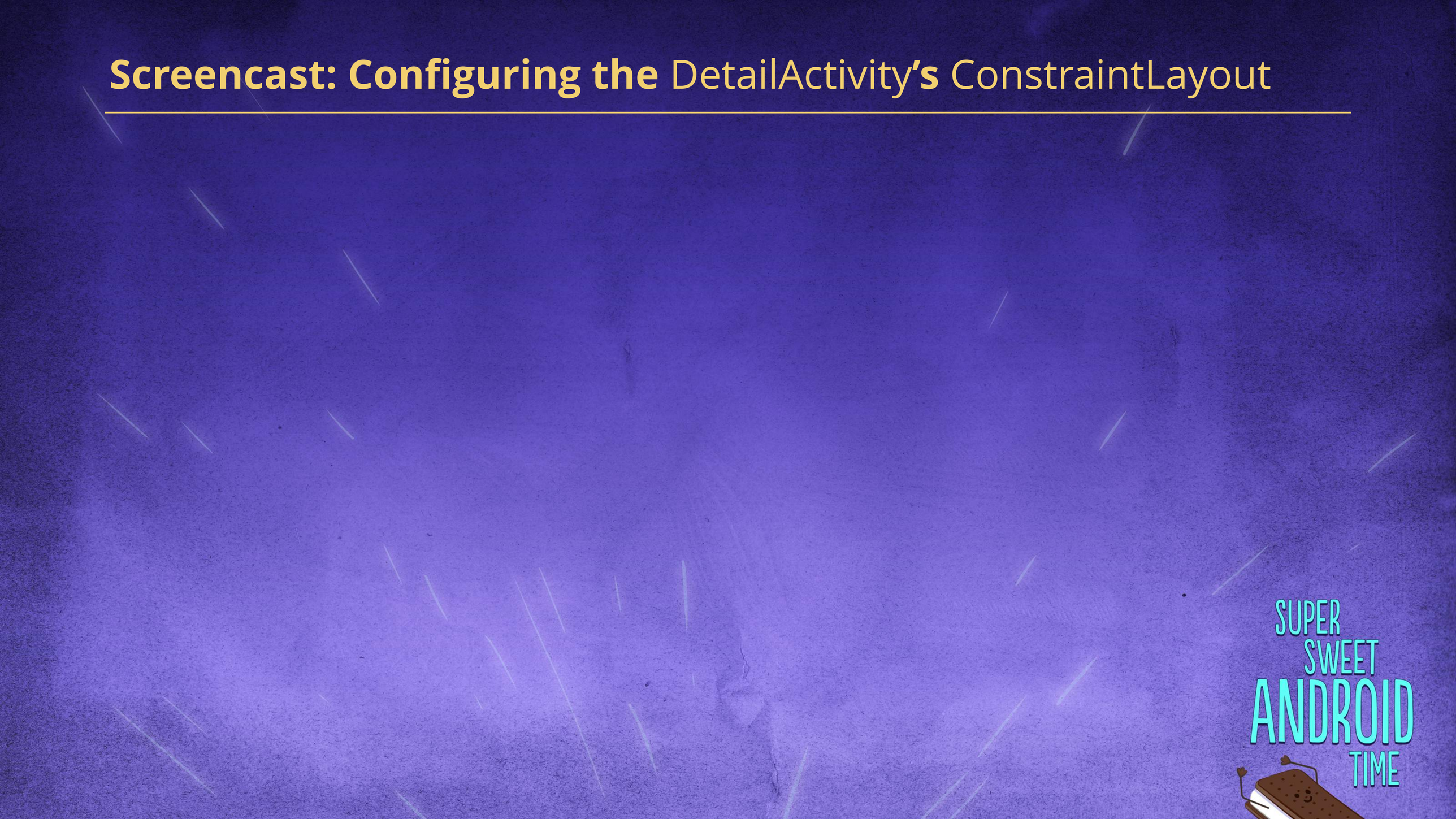
Our Detail Activity right now



Our final Detail Activity

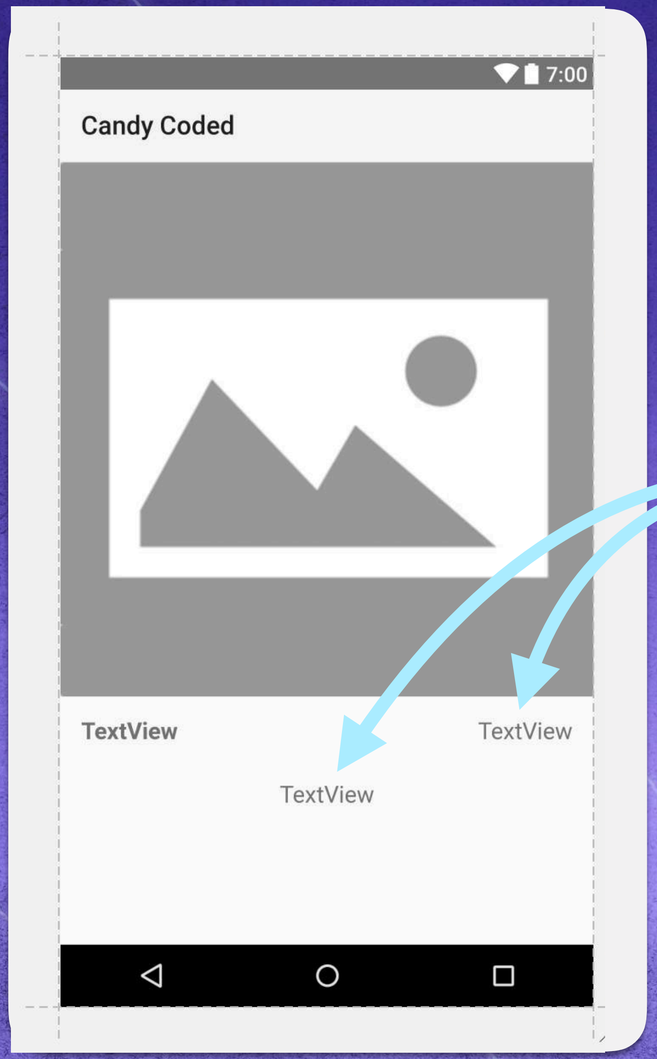
We can use a `ConstraintLayout` *to arrange all of these* `Views` *relative to each other*

Screencast: Configuring the DetailActivity**’s** ConstraintLayout



Our New Detail Activity Layout

Now that we have our new Detail Activity's layout in place, let's edit the Java code to fill the Views with data.



Let's write the code to fill the price and description `TextViews` *with actual data*

SUPER SWEET ANDROID TIME

Setting the TextView' text in DetailActivity

In DetailActivity, after we set the candy's name we can set the text for the other TextView objects.

DetailActivity.java

```
…
public class DetailActivity extends AppCompatActivity {
    @Override protected void onCreate(Bundle savedInstanceState) {
        …
        Intent intent = DetailActivity.this.getIntent();
        String candyName = "";
        if (intent != null && intent.hasExtra("candy_name")) {
            candyName = intent.getStringExtra("candy_name");
        }
        TextView textViewName = (TextView)this.findViewById(
                                                R.id.text_view_name);

        textViewName.setText(candyName);

        // Let's set the text for the other TextViews
        // here using the same 3 steps as above

    }
}
```

1. Get the value sent with the intent.
2. Find the TextView
3. Set the TextView's text

Setting the Description TextViews' text

Setting the text for the description TextView will be similar to setting it for the name.

DetailActivity.java

```
…
public class DetailActivity extends AppCompatActivity {
    @Override protected void onCreate(Bundle savedInstanceState) {
        …

        String candyDesc = "";
        if (intent != null && intent.hasExtra("candy_desc")) {
            candyDesc = intent.getStringExtra("candy_desc");
        }

        TextView textViewDesc = (TextView)this.findViewById(
                                                R.id.text_view_desc);
        textViewDesc.setText(candyDesc);


    }
}
```

Get the value sent with the intent

Find the **TextView**

Set the **TextView**'*s text*

Setting the Price TextViews' text

Setting the text for the description TextView will also be similar to setting it for the name.

DetailActivity.java

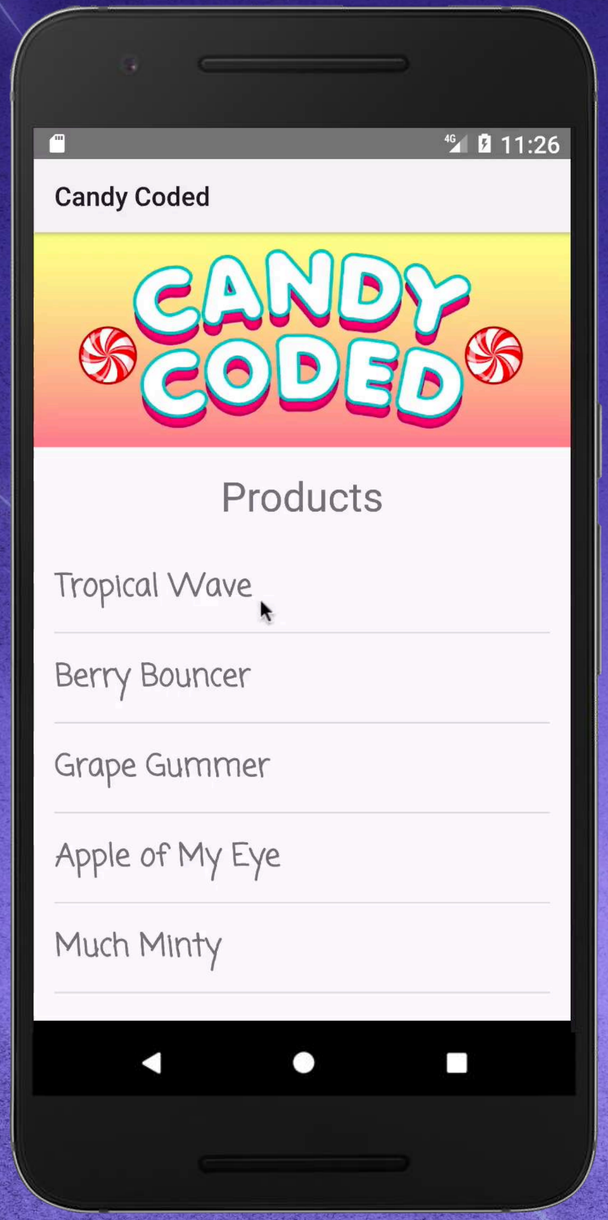
```
…
public class DetailActivity extends AppCompatActivity {
    @Override protected void onCreate(Bundle savedInstanceState) {
        …

        String candyPrice = "";
        if (intent != null && intent.hasExtra("candy_price")) {
            candyPrice = intent.getStringExtra("candy_price");
        }

        TextView textViewPrice = (TextView)this.findViewById(
                                                R.id.text_view_price);
        textViewPrice.setText(candyPrice);


    }
}
```

Get the value sent with the intent.

Find the **TextView**

Set the **TextView**'*s text*

Demo: The TextViews Have Live Data



Now our `TextViews` have live data!

Next we'll add our image!

SUPER SWEET ANDROID TIME

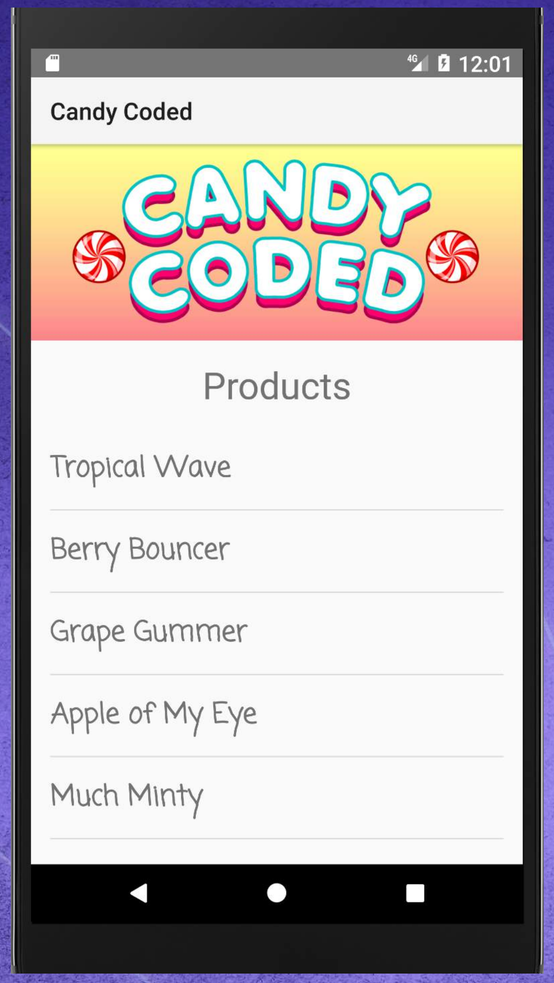
Level 3 – Section 2

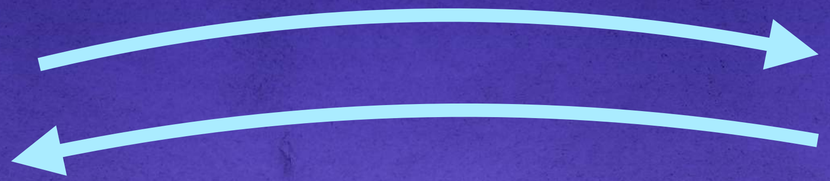
The Detail Activity Layout

Using the Picasso Image Library

Displaying an Image via a URL in an ImageView

Instead of writing somewhat lengthy code to download images into our app and cache them, Picasso is an Android library we can use to do this for us.



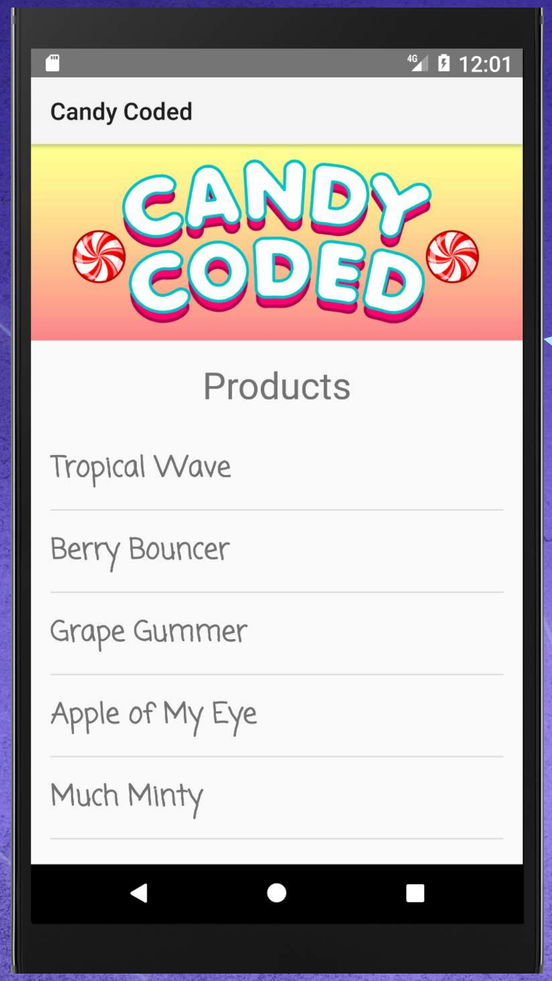


Web Server

SUPER SWEET ANDROID TIME

Caching

We need caching because getting files from a distant web server takes much longer than getting locally cached images from memory.



Cache Memory

If the image exists in our local cache memory, we can load it from there

(Only the most recently used items will be stored)

Otherwise, we'll download the image from the Web Server but also save it in our local cache for next time

Web Server

Adding the Picasso Library to our Project

build.gradle (app)

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 25
    buildToolsVersion "25.0.0"
    defaultConfig {
        applicationId "com.codeschool.candycoded"
        minSdkVersion 10
        targetSdkVersion 25 …
    } …
}
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile(…)
    compile 'com.android.support:appcompat-v7:25.3.0'
    ...
    compile 'com.squareup.picasso:picasso:2.5.2'
}
```

Add the Picasso Library to our project by adding to the bottom of dependencies.

Then when we sync our project, gradle will automatically download and compile any dependencies.

Using Picasso to Load an Image into an ImageView

Now that Picasso has been included in our project, it's really easy to use it in our code.

```
Picasso.with(this).load(candy_image).into(imageView);
```

The `context`*, this means that we're going to display the image in* `this DetailActivity`

The `url` *where* `Picasso` *will download the image from*

The `ImageView` *where we're going to display the image*

Remember by using `Picasso`*, Android Studio will automatically add the following* `import` *line to the top of your file. Or you can add it yourself.*

```
import com.squareup.picasso.Picasso;
```

Using Picasso to Load an Image into an ImageView

DetailActivity.java

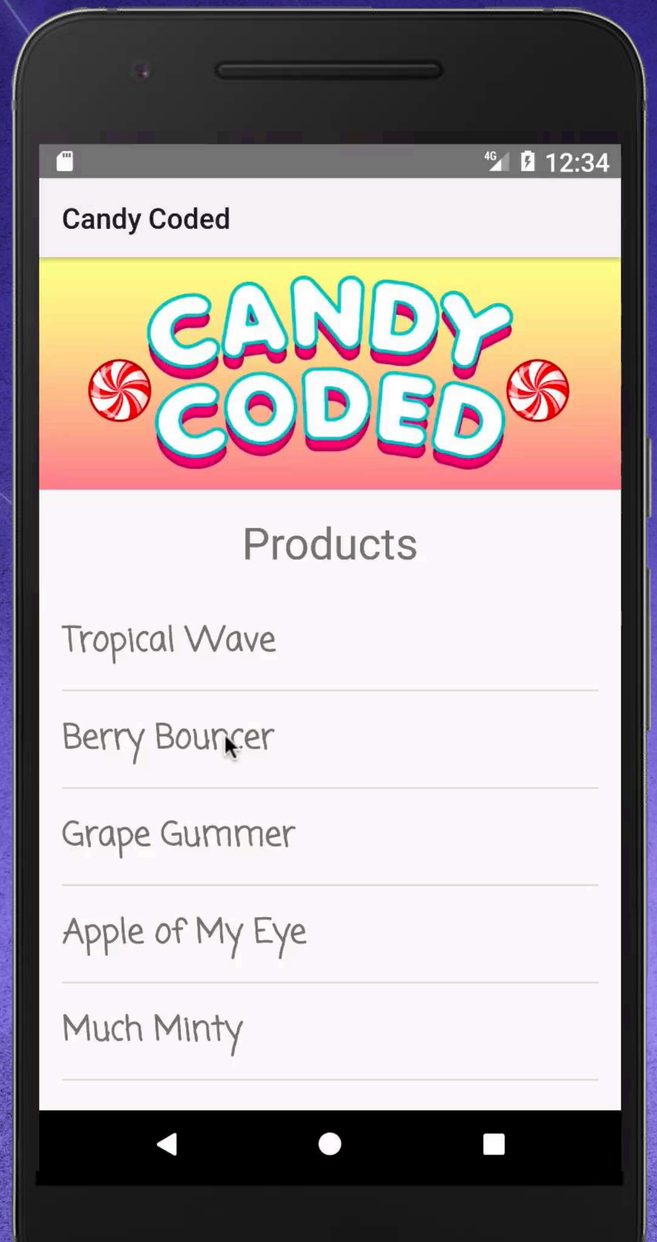
```
…
    Intent intent = DetailActivity.this.getIntent();
    …
    String candy_image = "";
    if (intent != null && intent.hasExtra("candy_image")) {
        candy_image = intent.getStringExtra("candy_image");;
    }

    ImageView imageView = (ImageView)this.findViewById(
                                        R.id.image_view_candy);
    Picasso.with(this).load(candy_image).into(imageView);
…
```

First we need the image url passed in with the `Intent`

Then we need to find our `ImageView`

We can then use `Picasso` *to load and cache our image from the URL and display it in our* `ImageView`

Picasso in Action



Now we can see Picasso loading our images dynamically from the image URL!


SUPER
SWEET
ANDROID
TIME