

Android practicum part 2

# Inleiding

Last week the focus was on te Android SDK installation. This week it’s time to start with a real app.

A common theme with police/crime series on TV is figuring out who is the actual perpetrator of the crime.

With CSI, the viewer is taken through a host of high-tech forensic gadgets, while Bones focuses on anatomy and Law and Order places its emphasis on interrogation techniques.

In the end however it all comes down to actually catching the bad guys.

And this is where the CrowdSource Criminal Tracer App comes in.

Every week a different TV series is taken as a theme for the app. The theme doesn’t have much impact on the functional behavior but it does on the look and feel.

This week we start with CSI (The original series). This means a gritty colorscheme: Green, blue and black.

# Main UI

Part 1: A new app.

Create a new project using your favorite IDE (for instance Android Studio or Eclipse).

(If you use a minimum required sdk version, you might have to download it through the SDK manager. If it is not convenient to do so now, use a minimum required sdk that you have already installed).

Call it CSI\_week\_1 and choose a good package name (for instance org.fhict.csi).

Use the default settings and select Blank activity.

Part 2:Add the face

The first thing the user should see is a face of the suspect. Find a suitable picture and copy it to the res/drawable resource directory of the android project.

You can also create different drawable directories besides res/drawable:

res/  
 drawable/  
 mybadguy.png   
 drawable-hdpi/  
 mybadguy.png

There are several drawable directories options: xhdpi, hdpi, mdpi and ldpi. What are they used for? Which directory should you use for the drawables?

………………………………………………………………………………………………………………………………………………………………..

DPI(Dots per inch) is a common measurement unit to measure the pixel density of a given display.

Android devices range from low DPI devices to (currently) extra-extra-extra-high DPI devices. To prevent devices from having blurry or pixelated image assets, Android supports multiple screen resolutions through the usage of drawable-dpi folders. Assets are loaded according to the DPI that corresponds to the used device.

It is preferred to use larger scale assets, as downscaling doesn’t distort an asset in the way upscaling does. Upscaling displays the image assets in a pixelated way.

Also see: <http://developer.android.com/guide/practices/screens_support.html>

………………………………………………………………………………………………………………………………………………………………..

Part 3: In your face.

Open the *activity\_*main.xml file and remove the textview.

Replace the relative layout with a linear layout and add an image view to it.

Select the image from the resources.

Make sure that the imageview has a small margin at the top and is centered horizontally.

Also make sure you know about the units to use: <http://developer.android.com/guide/topics/resources/more-resources.html#Dimension>

If you look at the XML code, you might get a warning.

Which warning is it?

………………………………………………………………………………………………………………………………………………………………..

………………………………………………………………………………………………………………………………………………………………..

This warning can be resolved by adding a content description.

The following XML attribute can be used to do this:

android:contentDescription =*"description"*

You might get another warning. Which one?

………………………………………………………………………………………………………………………………………………………………..

………………………………………………………………………………………………………………………………………………………………..

This warning can be resolved by moving the text of the content description to the string resources.

Open Stringx.xml and add a new string. Type the description of the image. Adjust the image view to use the string from the resources. Pro-tip: Use Quick fix or ctrl+1 to automate this process.

When you run the app, you should see something like this:



Part 4: Who’s that face

When making your layout, keep in mind that layouts can be nested.

Start by adding another linear layout below the imageview.

Add a tableLayout to this linear layout.

In this table you will show the details of the suspect.

To do this neatly, add even more linear layouts to the tableview rows, one for each row.

Android/Eclipse might give you a warning. Which one? Why?

…………………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………………

…………………………………………………………………………………………………………………………………

In these rows you can then add two textviews. One for the label, and one for the value.

Show the following data:

-Name

-Gender

-Age

-Bounty

Make sure that all elements are neatly separated with margins and that the displayed text is read from resources.

Also try to experiment a bit with the visual representation of the UI elements. Start with setting the top layout background color to black.

You could also use a background image for the tableview for example, or set the background of the tablerows to a semitransparent color. Be creative and try to capture the spirit of **CSI Las Vegas**. You should use at least one color (multiple preferred) that is read form the color resource.

Part 5 Details, details, details

Add another linearlayout below the tablelayout.

Add a textview on this linearlayout with the text: “Details”

Now place a scrollview below this textview.

A scrollview is capable of scrolling through its contents (using a scroll bar) when they don’t fit anymore.

This scrollview has a linearlayout placed in it by default.

Add a textview to this layout, which should display details about the suspect.

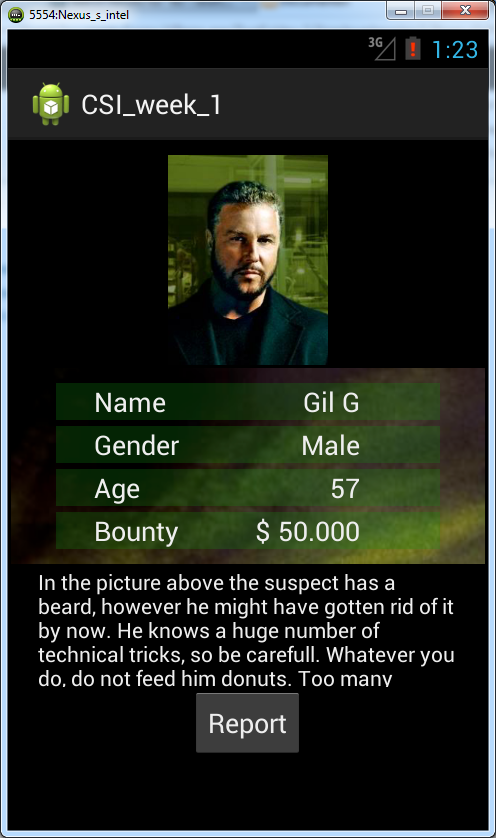
Make sure that the text is long enough so the scrollview is able to actually scroll.

To ensure that scrolling takes place, put a maximum to the layout\_height property of the scrollview, 90 dp might be a good value.

Part 6: Report

Add a button to the same LinearLayout that contains the ScrollView. It should have the caption: “Report”. Make sure that the button is horizontally centered. You can do this by adjusting the ‘gravity’ property of the element.

You should end up with something like the following screenshot:

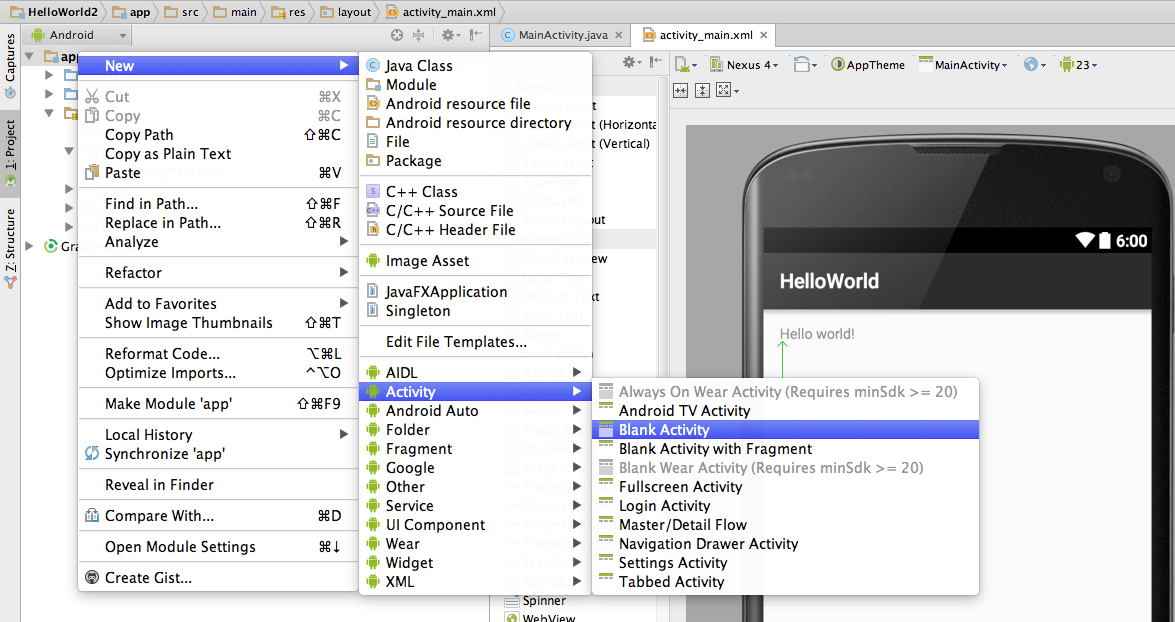


# Report UI

Part 1 Report activity

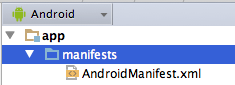
Add a new activity to your project (see example of Android Studio in screenshot below).

Call it ‘ReportActivity’. Make sure that you use the same package as the rest of your project.



Part 3 Report UI

This step will generate an ReportActivity.java and corresponding layout file for you in the res/layout folder. It will also add the newly created activity to the AndroidManifest.xml file:



Android studio will add the activity for you to the AndroidManifest.xml file:



part 5 Report UI 2

It’s time to create the report UI.

Open the activity\_report.xml file and construct the UI.

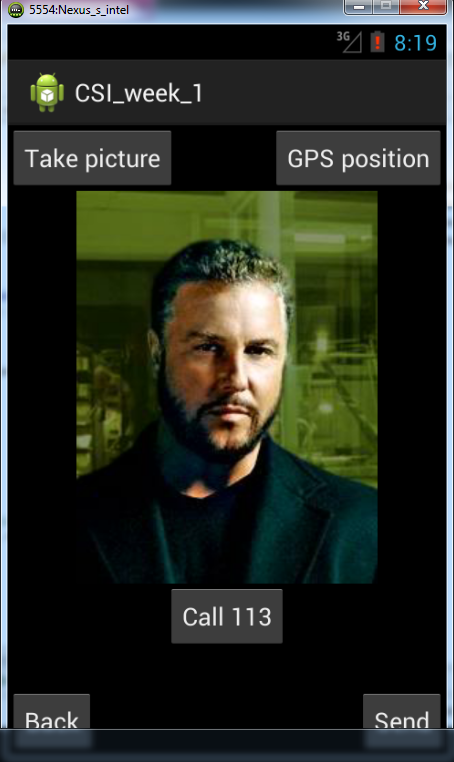
Take note of the following when doing so:

* The mugshot should be shown in the top of the UI
* Add three buttons on the mugshot:
  + One button top left, with the caption: Take Picture
  + One button top right, with the caption: GPS position
  + One button center bottom, with the caption: Call 113
* Add two buttons to the linearlayout. Align them bottom left and bottom right.

The left one should read : ”Back” and the right one “Send Report”.

Hint: Take a look at RelativeLayout.

Possible result:



# Buttons

Part 1 Go to Report.

Open MainActivity.java and make adjustments to the OnCreate function.

Get a reference to the Report button, by using findViewById.

Add a new OnClickListener to this button. (Hint: When you type **new** OnClickListener your development environment might be kind enough to create the code for you).

Create a new Intent in the OnClick method of the OnClickListener. Use this intent to show the next activity (ReportActivity).

Part 2 Go Back

When you press the back button you should return to the previous screen.

Add another OnClickListener to the correct button in the OnCreate function of the activity.

You could repeat the same as above by creating a new Intent, or you can use the finish() method to indicate that the activity is done.