

ACTIVITY 2.2.6

Let Me Take a Selfie

INTRODUCTION

In this activity, you will add the ability to take a picture and display it in the user profile of the CollegeApp.

Materials

- Computer with Android™ Studio
- Android™ tablet and USB cable, or device emulator

Procedure

Part I: Program College App to take a picture and save it in the Profile

- 1 In Android Studio, open your CollegeApp.

If you were unable to finish the activity, import *2.2.5CollegeApp_Solution* as directed by your teacher. (If you use the solution code, change `BE_APP_ID`, `BE_ANDROID_API_KEY`, and `MY_EMAIL_ADDRESS` in *ApplicantActivity.java* to reference your personal Backendless and email values.)

When College App takes a photo, the view of the image will be stored in an Android class called `ImageView`. The button used to initiate the picture taking will use a class called `ImageButton`. The XML must also be modified to accommodate the views for these two new parts of your app's user interface.

- 2 Define new widgets in *fragment_profile.xml*, as shown below. Make this a vertical, linear layout nested inside of the linear layout you already have set up. The code shown below is just one example of a way to make the screen look appealing. Play with the layout of your own screen to make it look presentable.

```
1: <LinearLayout
2:     android:orientation="vertical"
3:     android:layout_width="fill_parent"
4:     android:layout_height="wrap_content">
5:     <ImageView
6:         android:id="@+id/profile_pic"
7:         android:layout_width="100dp"
8:         android:layout_height="100dp"
9:         android:scaleType="centerInside"
10:        android:background="@android:color/darker_gray"
11:        android:cropToPadding="true"/>
12:     <ImageButton
13:         android:id="@+id/profile_camera"
14:         android:layout_width="match_parent"
15:         android:layout_height="wrap_content"
16:         android:src="@android:drawable/ic_menu_camera"/>
17: </LinearLayout>
18:
```

- 3 A Profile will have one profile picture associated with it, so add the following instance variables to your ProfileFragment class.

```
1: private ImageButton mSelfieButton;
2: private ImageView mSelfieView;
3: private File mSelfieFile;
```

- 4 Create a helper method `getPhotoFilename` in Profile that returns the value `"IMG_PROFILE.jpg"`. Remember to create a private constant to represent the name of the jpg file and reference the constant in your helper method.

5 Review the slideshow.

getPhotoFile

```
1 public File getPhotoFile(Context context) {  
2     File externalFilesDir = context.getExternalFilesDir(Environment.DIRECTORY_PICTURES);  
3     if (externalFilesDir==null){  
4         return null;  
5     }  
6     return new File(externalFilesDir, getPhotoFileName());  
7 }
```

Computer Science A

© 2016 Project Lead The Way, Inc.

Line 1 – The context that will be passed in will represent the Activity in which the Profile exists.

Line 2 – Uses the context to get information about where pictures can be found in external storage

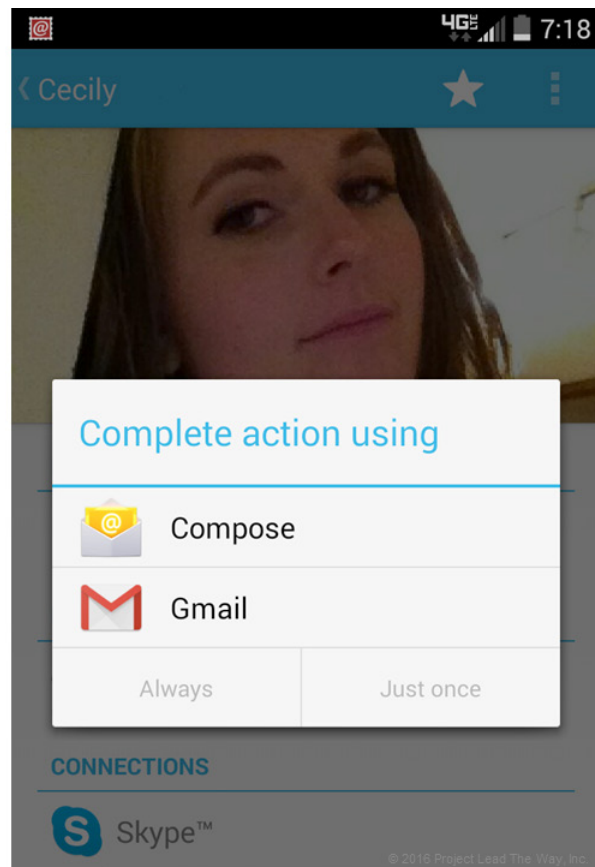
Lines 3–5 – If there isn't a location in external storage for pictures, return null

Line 6 – Return a File type object constructed from the location of pictures in the external file structure and the name given to the profile photo, defined in Step 4 as "IMG_PROFILE.jpg"

Implicit Intents

- You have created explicit intents
- Implicit intents do not specify an Activity to start
- Let the operating system do the work for you

Computer Science A



© 2016 Project Lead The Way, Inc.

Implicit Intents

```
1 final Intent captureSelfie = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
2 boolean canTakeSelfie = mSelfieFile != null &&
3     captureSelfie.resolveActivity(getActivity().getPackageManager()) != null;
4 mSelfieButton.setEnabled(canTakeSelfie);
5 if (canTakeSelfie) {
6     Uri uri = Uri.fromFile(mSelfieFile);
7     captureSelfie.putExtra(MediaStore.EXTRA_OUTPUT, uri);
8 }
9 mSelfieButton.setOnClickListener(new View.OnClickListener() {
10     @Override
11     public void onClick(View v) {
12         startActivityForResult(captureSelfie, REQUEST_SELFIE);
13     }
14 });
15
```

Computer Science A

© 2016 Project Lead The Way, Inc.

Lines 1 – Notice the constant that is passed to the constructor of Intent. This value tells the OS what task will need to be completed by whatever Activity it provides.

Lines 2 and 3 – This Boolean value will be true only if mSelfieFile exists and the intent finds an appropriate activity to capture an image.

Line 4 – makes sure that the selfie button is enabled only if the conditions on the previous two lines are met.

Lines 5–8 – Again, if the two previous conditions are met, create a URI object (an object used to help the app locate a resource), in this case mSelfieFile, then store that URI in the intent.

Lines 9–14 – Create a new onClickListener using an anonymous inner class, overriding the default onClick(View v) method by calling startActivityForResult and passing in the implicit intent.

AndroidManifest.xml

```
1 package="org.pltw.examples.collegeapp" >
2
3 <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
4     android:maxSdkVersion="18"/>
5 <uses-feature android:name="android.hardware.camera"
6     android:required="false"/>
7
8 <application
```

Computer Science A

© 2016 Project Lead The Way, Inc.

Line 1 – Included as a reference point.

Line 3 – uses-permission is a tag that Android uses to let app users know what it is doing. In this case when the user installs the app, it will tell the user that it needs to read from External Storage.

Line 4 – This line indicates that the app will only ask for permission to read from external storage up to SDK version 18. This is because you don't need to ask for permission for this functionality post SDK 19.

Line 5 – uses-feature tells Google Play about features that your app uses... in this case the camera.

Line 6 – If the value of required were true, then Google Play would refuse to install the app unless the target device had a camera. Since our app can function to some extent without the camera, we set this value to false.

- 6 For CollegeApp, the image file that the camera produces will be stored in the device's shared storage area. The method you will use is `getExternalFilesDir`, but don't be confused by the word "external". Think of it as a storage area that is external to the app. To get the file from storage, add the following method to `ProfileFragment`.


```
1: public File getPhotoFile() {
2:     File externalFilesDir = getActivity().
        getExternalFilesDir(Environment.DIRECTORY_PICTURES);
3:     if (externalFilesDir == null) {
4:         return null;
5:     }
6:     return new File (externalFilesDir, mProfile.
        getPhotoFilename());
7: }
```

- 7 Initialize `mSelfieFile` within the `ProfileFragment` `onCreate` method to have the value returned by the `getPhotoFile` method.
- 8 Add the integer constant `REQUEST_SELFIE` to `ProfileFragment` with value 1.
- 9 Wire up your new widgets in `ProfileFragment`:
- Initialize `mSelfieView` and `mSelfieButton` to their reference ids defined in `fragment_profile.xml`.
 - (Slides 3 and 4) Add the following after you initialize `mSelfieView` and `mSelfieButton`. This code creates an **implicit intent** that starts a camera activity to handle taking pictures for you.

implicit intent

A way of signaling to the Android OS that your app needs it to provide some external service to accomplish a given task; also allows the user to select which app is used at run time.

```
1: final Intent captureSelfie = new Intent(MediaStore.ACTION_IMAGE_
    CAPTURE);
2: boolean canTakeSelfie = mSelfieFile != null &&
3:     captureSelfie.resolveActivity(getActivity().
        getPackageManager()) != null;
4: mSelfieButton.setEnabled(canTakeSelfie);
5: if (canTakeSelfie) {
6:     Uri uri = Uri.fromFile(mSelfieFile);
7:     captureSelfie.putExtra(MediaStore.EXTRA_OUTPUT, uri);
8: }
9: mSelfieButton.setOnClickListener(new View.OnClickListener() {
10:     @Override
11:     public void onClick(View v) {
12:         startActivityForResult(captureSelfie, REQUEST_SELFIE);
13:     }
14: });
```

- 10 Test your program. You should be able to take pictures and navigate back to your app, though you won't see pictures display in your app yet.
- 11 To display a picture in your app:
- In `ProfileFragment`, create a new method `updateSelfieView` that takes no parameters and has a `void` return type.
 - If `mSelfieFile` is not null and `mSelfieFile.exists()` is true, create a `Bitmap` object, assigning to it the object returned by `BitmapFactory.decodeFile(mSelfieFile.getPath())`.
 - To discover the call that will set `mSelfieView`'s image to the `Bitmap` you just created, explore  **ImageView**.



What did the value of the parameter passed to `BitmapFactory.decodeFile` represent and why is it needed?

What did the `BitmapFactory.decodeFile` return and how is it used?

- 12 Just before you return the `rootView` in `onCreateView` of `ProfileFragment`, call `updateSelfieView`.

- 13 In `onActivityResult` in `ProfileFragment`, add a conditional to check whether the value of `requestCode` indicates that a selfie is being requested and if so, call `updateSelfieView` there as well.
- 14 (Slide 5) In `AndroidManifest.xml`, add lines 4–7 below.

```
1: <uses-permission android:name="android.permission.INTERNET" />
2: <uses-permission android:name="android.permission.ACCESS_
   NETWORK_STATE" />
3:
4: <uses-permission android:name="android.permission.READ_
   EXTERNAL_STORAGE"
5:                 android:maxSdkVersion="18" />
6: <uses-feature android:name="android.hardware.camera"
7:              android:required="false" />
8:
9: <application
```

- 15 Test your app to make sure that it displays a thumbnail image of the picture that you take within `ProfileFragment`: Make sure to accept the picture you take (for example, click **OK**). If you navigate with the back button from the Camera app, your picture will not display in your app. Also, note that if you take a picture when the device is vertical, the thumbnail will be incorrectly rotated.
- 16 To fix the rotation problem, explore the XML attributes that `ImageView` inherits from  **View** and make the change in `fragment_profile`.
- 17 *Challenge*: Add a component to your app that allows an applicant to submit a video essay. You may want to use the following link as a resource.  <http://developer.android.com/training/camera/videobasics.html>

Part II: Maintenance and End-of-Life Software

After an app has been developed and released, programmers must maintain the software. You will not be maintaining the software you write in class, but in the “real world,” programmers spend much of their time maintaining the apps they (or others) have written.

When a new app replaces an old one, the old one has reached its “end-of-life.” Often, programmers stop maintaining the end-of-life application and only maintain the new, replacement version. This can pose a security risk. If users do not uninstall and remove end-of-life applications, hackers can take advantage of bugs that may still exist in the old software.

- 18 How are bugs, software testing, patches, (discussed in *Activity 1.2.3 Data Storage*) and end-of-life software related?

CONCLUSION

1. Investigate to find out what features and permissions one of your favorite apps uses. List the features and permissions and explain the difference.