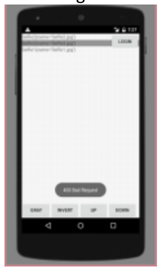
Final Project Report

Sunday, November 15, 2015 6:51 PM

Welcome to My Daily Selfie App. This app reminds you to take a selfie every minute (so you as an evaluator can see the alarms - a real app would do it only daily). When you take a picture it gets uploaded to a server. You then have the ability to do two different image processing operations - make the image grayscale or invert the colors of the image and then download it to your device and view a larger picture of the selfie.

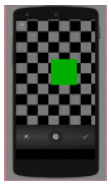
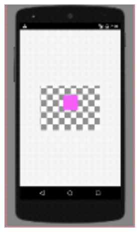
Below is a table of the project requirements from the Coursera grading page. I've created a table for ease

Requirement	***How does this App meet it?***	***Comments***
Basic Project Requirement		
App supports multiple users via individual user accounts	The app currently has two users built in "admin" and "user0" both with the password "pass". The screencast demonstrates both users logging in. Each user has their own account.	More users could be added. There is no limit to the number of users.
App contains at least one user facing function available only to authenticated users	Only owners can modify and download their selfies. But all selfies can be viewed in the list. The app requires the user to log in from the beginning	User will get a 400 error when trying to modify/image process or download a selfie they do not own. 

App comprises at least 1 instance of each of at least 2 of the following 4 fundamental Android components:
• Activity
• BroadcastReceiver
	The app uses several Activities and also uses a BroadcastReceiver for the Alarm mechanisms. See code fragments to the right.	`Public class AlarmReceive rextends BroadcastReceiver{`
		`Private static final StringTAG="AlarmReceiver";`
		`**************************`
		
		`Public class MainActivity extends ListActivity{`
		
		`//itemsrelatedtothealarmtotakeaselfie`

App interacts with at least one remotely-hosted Java Spring-based service.	The app interacts with one remotely-hosted Java Spring-based serve for uploading/downloading and image processing the Selfies. See code snippet to the right.	`import java.io.IOException;`
		
		`import javax.servlet.MultipartConfigElement;`
		
		`import org.magnum.mobilecloud.video.auth.OAuth2SecurityConfiguration;`
		`import org.magnum.mobilecloud.video.repository.VideoFileManager;`
		`import org.springframework.boot.SpringApplication;`
		`import org.springframework.boot.autoconfigure.EnableAutoConfiguration;`
		`import org.springframework.boot.context.embedded.MultiPartConfigFactory;`
		`import org.springframework.context.annotation.Bean;`
		`import org.springframework.context.annotation.ComponentScan;`
		`import org.springframework.context.annotation.Configuration;`
		`import org.springframework.context.annotation.Import;`
		`import org.springframework.data.rest.webmvc.config.RepositoryRestMvcConfiguration;`
		`import org.springframework.web.servlet.config.annotation.EnableWebMvc;`
		
		`//Tell Spring to automatically inject any dependencies that are marked in`
		`//our classes with @Autowired`
		`@EnableAutoConfiguration`
		`// Tell Spring to turn on WebMVC (e.g., it should enable the DispatcherServlet`
		`// so that requests can be routed to our Controllers)`
		`@EnableWebMvc`
		`// Tell Spring that this object represents a Configuration for the`
		`// application`
		`@Configuration`
		`// Tell Spring to go and scan our controller package (and all sub packages) to`
		`// find any Controllers or other components that are part of our applciation.`
		`// Any class in this package that is annotated with @Controller is going to be`
		`// automatically discovered and connected to the DispatcherServlet.`
		`@ComponentScan`

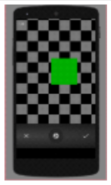
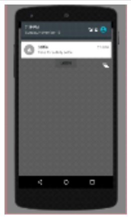
App interacts over the network via HTTP/HTTPS.	Yes the app uses Retrofit to simplify the RESTful calls over the network.	
App allows users to navigate between 3 or more user interface screens at runtime	Yes - there are four screens. The Login screen, the List Selfies screen, and the larger image when it is downloaded/clicked. There is also the screen when the picture is taken. See three screen shots to the right.	

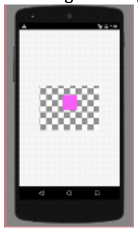
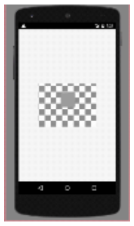
		 
App uses at least one advanced capability or API from the following list (covered in the MoCCA Specialization): multimedia capture, multimedia playback, touch gestures, sensors, animation	This app uses multimedia capture and multimedia playback - using the Android camera and then displaying the image later. See code sample to the right.	

```
private void dispatchTakePictureIntent() {
        Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        // Ensure that there's a camera activity to handle the intent
        if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
            // Create the File where the photo should go
            File photoFile = null;
            try {
                photoFile = createImageFile();
            }
            catch (IOException ex) {
                // Error occurred while creating the File
                Log.d("ERROR:", ex.getMessage());
            }

            // Continue only if the File was successfully created
            if (photoFile != null) {
                takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(photoFile));
                startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE);
            }
        }
    }
```

App supports at least one operation that is performed off the UI Thread in one or more background Threads of Thread pool.	All the calls made to the server are done in a separate thread. AsyncTasks are used. I used a thread pool on the server side.	

```
void login(final String login, final String password) {
            new AsyncTask<Void, Void, VideoSvcApi>() {
                @Override
                protected VideoSvcApi doInBackground(Void... params) {
                    mVideoSvcApi = new SecuredRestBuilder()
                            .setLoginEndpoint(Constants.TEST_URL + VideoSvcApi.TOKEN_PATH)
                            .setUsername(login)
                            .setPassword(password)
                            .setClientId(Constants.CLIENT_ID)
                            .setClient(new OkClient(UnsafeHttpsClient.getUnsafeOkHttpClient()))
                            .setEndpoint(Constants.TEST_URL).setLogLevel(RestAdapter.LogLevel.FULL).
build()
                            .create(VideoSvcApi.class);

                    Log.i(TAG, "mVideoSvcApi=" + mVideoSvcApi);
                    return null;
                }

                @Override
                protected void onPostExecute(VideoSvcApi videoSvcApi) {
                    getAvailableVideos();
                }
            }.execute();
```

Functional Description and App Requirement		
App allows user to take and save a Selfie.	Yes - see screen capture to the right.	
App reminds *User* to take a Selfie.	Yes - see notification to the right in screen capture.	
App allows user to view saved Selfies in a ListView.	Yes - see screen capture to the right.	

If the User closes and then reopens the App, the user has access to all saved Selfies.	Yes - this is demo'ed in the screencast.	
The App allows the User to select one or more Selfies, select one or more graphic effects, and apply the selected effects to the selected Selfies, and then view the processed Selfies in a ListView.	Yes - this is demo'ed in the screencast.	I used the image processing library called "Marvin" and the two effects are grayscaling the image and inverting the image. Here are two screen shots of a selfie with those effects:  
Some part of the operations corresponding to Requirement 5 (Applying Graphics Effects) must be executed concurrently. Therefore, App allows Users to start new a graphic effects operation, while a previous one is still in progress.	Yes - this operation is done on a server which is called in a separate thread. This is demo'ed in the screencast.	

Design

The Daily Selfie with Concurrent Image Processing

This project had two main pieces:
- A server developed in Eclipse which receives images, stores them, processes them, and downloads them to the client
- A client developed with Android Studio that uploads images to the server for storage and image processing, and will ask the server to download the image to the client.

Basic Project Requirements

- Below is a list of the project requirements and how my design has meet these requirements.

1. Apps must support multiple users via individual user accounts.

   I have implemented the Oauth framework as I did in previous MOOCs. I built in two users (admin, user0) all with the password "pass", but more could be added. Each user has their own individual account.

2. At least one user facing operation must be available only to authenticated users.

   Authenticated users can all images on the server but can only do image processing on their images. They also can only download their images.

3. App implementations must comprise at least one instance of at least two of the following four fundamental Android components: Activity, BroadcastReceiver, Service, and ContentProvider.

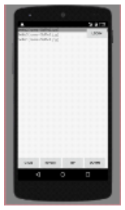
   This app has the following Activities: A Main Activity , a Login Activity , and an Activity for taking a picture. A BroadcastReceiver is implemented for using Alarms to notify the user when to take a Selfie.

5. Apps must interact with at least one remotely-hosted Java Spring-based service over the network via HTTP/HTTPS.

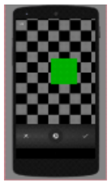
Spring and Retrofit have been used for the web service on the server side.

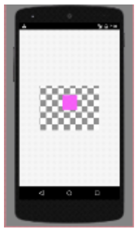
6. At runtime apps must allow users to navigate between at least three different user interface screens, e.g., a hypothetical email reader app might have multiple screens, such as (1) a List view showing all emails, (2) a Detail view showing a single email, (3) a Compose view for creating new emails, and (4) a Settings view for providing information about the user's email account.

There are actually four different UI screens

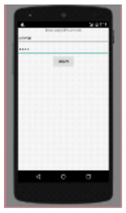
- The landing page which shows all that users images on the server
- 

- The taking a selfie picture



- The blown up image screen
- 

- The login page



7. Apps must use at least one advanced capability or API from the following list covered in the MoCCA Specialization: multimedia capture, multimedia playback, touch gestures, sensors, or animation

I have used the camera capture.

8. Apps must support at least one operation that is performed off the UI Thread in one or more background Threads or in a Thread pool. Communication between background Threads and the UI Thread should be handled by at least one of Android concurrency frameworks, such as the HaMeR or AsyncTask framework.

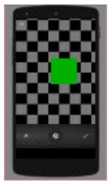
All server operations are performed in separate threads on the client using AsyncTask framework. I used a thread pool on the server side.

```
void login(final String login, final String password) {
          new AsyncTask<Void, Void, SelfieSvcApi>() {
              @Override
              protected SelfieSvcApi doInBackground(Void... params) {
                  mSelfieSvcApi= new SecuredRestBuilder()
                          .setLoginEndpoint(Constants.TEST_URL + SelfieSvcApi.TOKEN_PATH)
                          .setUsername(login)
                          .setPassword(password)
                          .setClientId(Constants.CLIENT_ID)
                          .setClient(new OkClient(UnsafeHttpsClient.getUnsafeOkHttpClient()))
                          .setEndpoint(Constants.TEST_URL).setLogLevel(RestAdapter.LogLevel.FULL).build()
                          .create(SelfieSvcApi.class);

                  Log.i(TAG, "mSelfieSvcApi=" + mSelfieSvcApi);
                  return null;
              }

              @Override
```

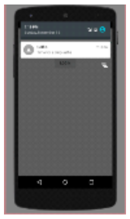
```
                protected void onPostExecute(SelfieSvcApi, selfieSvcApi) {
                    getAvailableSelfies();
                }
            }.execute();
```

Particular Selfie Project Requirements

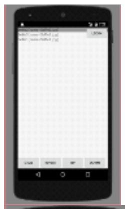
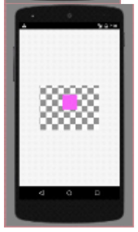
1. A *Selfie* is an image captured using the Daily Selfie application by the *User*. The *User* creates selfies by taking a picture using the phone's built in camera app, which saves the image to the device at a designated location.
Yes here is my app taking a picture:



2. A *Reminder* is a unit that holds a time. The *User* is informed daily to take a *Selfie* at the time specified in the *Reminder.*
Yes see the notification:



3. The *User* can view their saved *Selfies* in an Android ListView. Clicking on a *Selfie* in the ListView will open a large view in a separate Activity, showing the *Selfie* in a larger form.
Yes, this is the original list view and the expanded view.

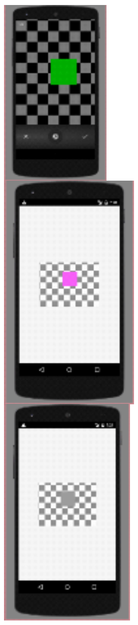



4. If the *User* closes and then reopens the app the user has access to all saved *Selfies*.

Selfies are saved on the server and displayed when the user logs in again.

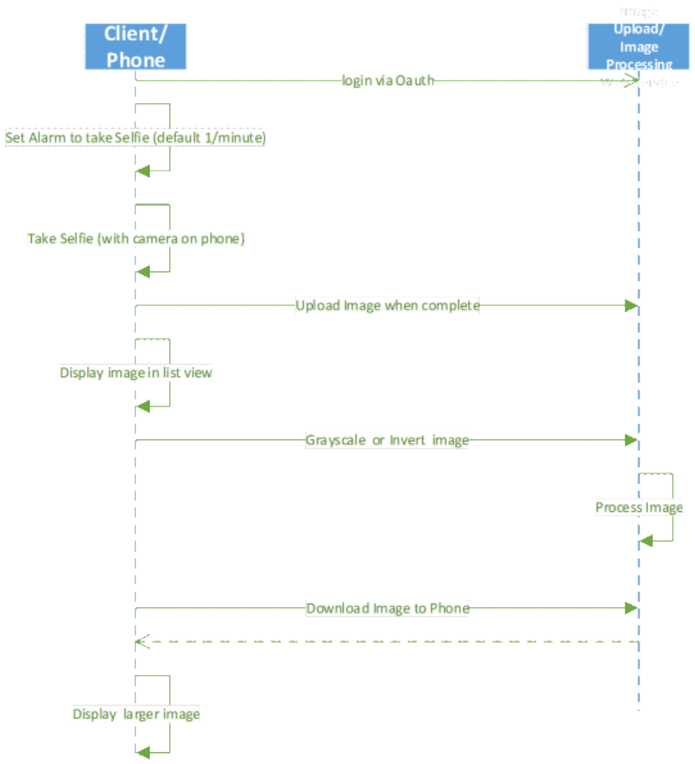
5. The *User* can process *Selfies* by applying two or more available effects, which can include adding noise, blurring, or adding a charcoal effect to the image. The *User* selects any number of *Selfies* to process, selects one or more effects, and then applies the selected effects.

I used the Marvin imaging processing java library and I use the effects of grayscale and inverting the picture. Here is a screen shot of the two effects and the original picture:



6. *Image Processing* operations on a *Selfie* are performed concurrently in a remote web service. Once processed, the images are returned to the device, where they are saved and displayed in a ListView.

See following sequence diagram for overall operation:


Client/ Phone
Upload/ Image Processing
login via Oauth
Set Alarm to take Selfie (default 1/minute)
Take Selfie (with camera on phone)
Upload Image when complete
Display image in list view
Grayscale or Invert image
Process Image
Download Image to Phone
Display larger image

Implementation Considerations

- How will *Reminders* be used to inform the *User* that it's time to take a *Selfie*? For example, will an Android notification be used, so that clicking on the notification will bring the user to the Daily Selfie application?

  Yes a notification will be used.

- How will you store the images on the device? For example, will you use a directory of files, a ContentProvider, etc.? Likewise, will copies of both the original and filtered images be stored, just the filtered images, etc.?

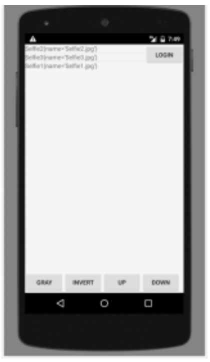
- Both copies will be stored on the phone and remotely on the server, but the remote server is what the user interacts with.

- What image processing effects/algorithms will you support on the remote web service? Will you write these in Java or in a native language like C/C++ accessed via the JNI?

  I used a package called Marvin because I found it easier to understand compared to ImageMagic and they had lots of code examples that I could use to learn how the library worked.

- What will the user interface look like for the Daily Selfie app so that it is quick and simple to use? How will there be at least three different user interface screens?

  Here is my user interface - it is not professional looking but it helped with debugging.


Selfie2{name='Selfie2.jpg'}
Selfie3{name='Selfie3.jpg'}
Selfie1{name='Selfie1.jpg'}
LOGIN
GRAY
INVERT
UP
DOWN

- What, if any, user preferences can the *User* set? How will the app running on the device, as well as the remote web service, be informed of changes to these user preferences?

  I don't have any preferences - I tried to implement an alarm menu but ran out of time to get it implemented. This would allowed the user to set the time of the alarm.

- How will the device and the remote web service implement the concurrency requirements, for instance, so that images are processed concurrently (e.g., will you use an AsyncTask on the device and a Java ExecutorService thread pool on the web service)?

  I used AsyncTask on the client side and a thread pool on the server side.

- What interface will the remote web service provide to the app running on the device and how will you communicate between the device and the remote web service (e.g., using RetroFit, Android HTTP client, etc.)?

  I'm using Retrofit - see code snippet above.

- How will the app use at least one advanced capability or API listed above? For example, will you create an animation to explain how to use the app? How will you allow *Users* to take pictures? Will you use push notifications to prompt *Users* when to take *Selfies*?

  I'm using the camera on the phone to take pictures and using push notifications when the alarm goes off.

- Does your app really require two or more fundamental Android components? If so, which ones? For example, this app might benefit from using a ContentProvider or from using a background Service that synchronizes local and remote images to a cloud-based server only when the device is connected to a WiFi network.

  As explained above I am using Activities and BroadcastReceiver.

My screen cast is here:

https://youtu.be/oMqpw8eG5ag