Mid Point Project

Monday, October 12, 2015 8:10 PM

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 will 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 will build in three users (admin, user0) all with the password "pass".

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

   Authenticated users can only see their images on the server and 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.

   A service will be used to upload and download the images. Activities will be used to login and for the main activity of the application. ContentProvider will be used to store the images locally on the phone.

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

   Spring and Retrofit will be 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.

The three different UI screens will be:

- The landing page which will show all that users images on the server
- The taking a selfie page
- The setting of the alarm to take a selfie
- The downloading of a selfie
- The uploading of a selfie / and image processing.

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 will be using 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.

Uploading and downloading of an image will be performed in separate threads.

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.

See prototype of app - camera in upper left hand corner takes the picture using the camera on the device.

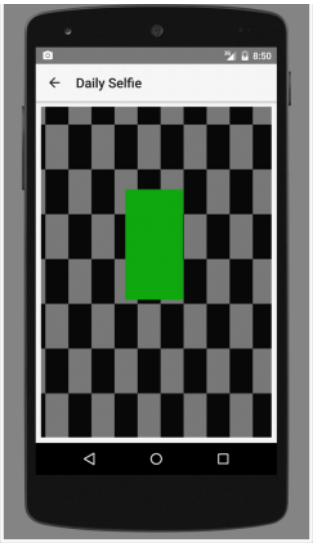
2. A *Reminder* is a unit that holds a time. The *User* is informed daily to take a *Selfie* at

the time specified in the *Reminder.*

Reminder will be set using the menu in the upper right hand corner (vertical …).

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 can be saved both on the server and the phone. The user can switch between which ones they are looking at.

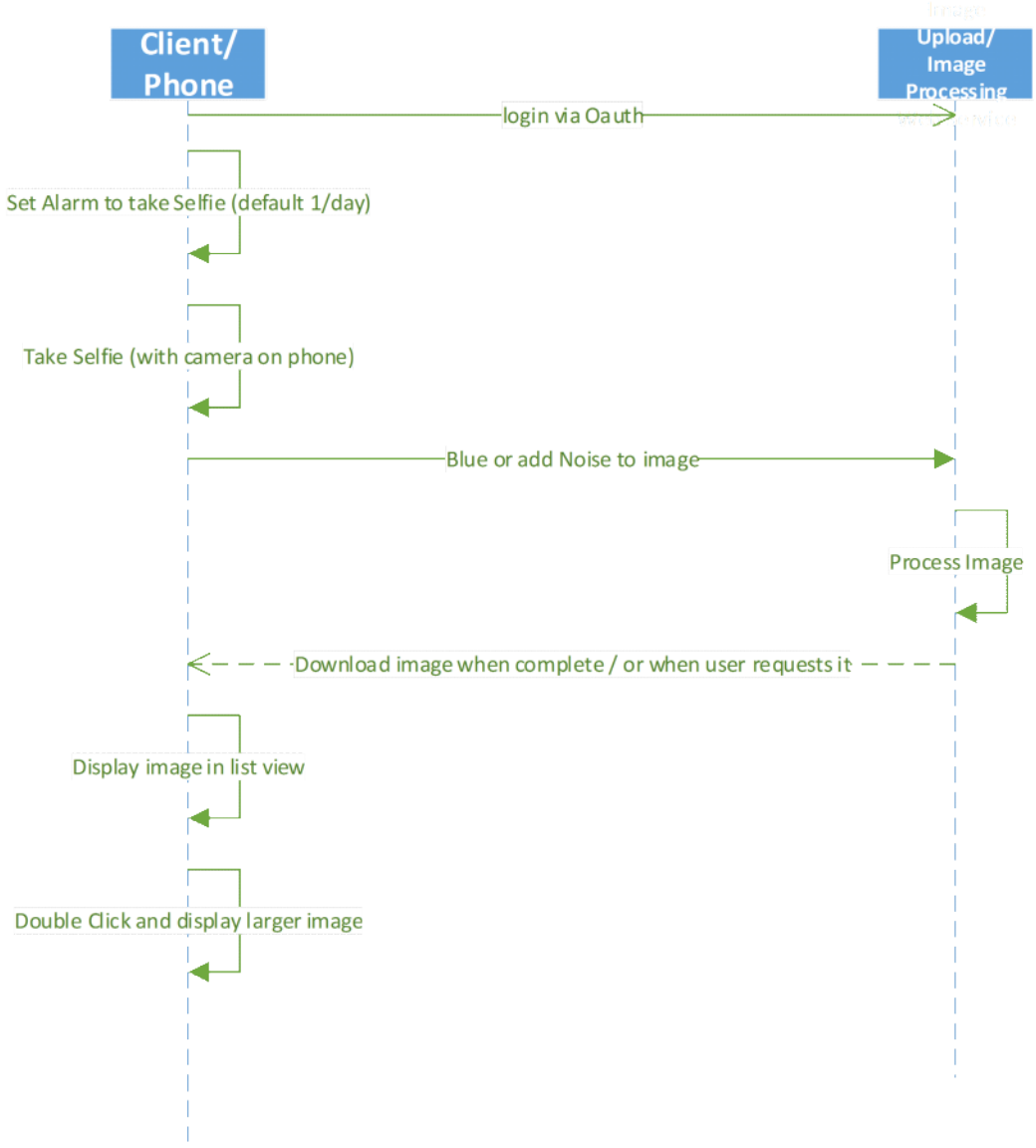
5. The *User* can process *Selfies* by applying two or more available effects, which can include adding noise, blurring, or adding a charcoal effect to the image. The *User* selects any number of *Selfies* to process, selects one or more effects, and then applies the selected effects.

This is a prototype of the server view. I will allow blurring and adding noise to the image.

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.

I plan to use ImageMagic to process the selfie.

See following sequence diagram for overall operation:



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.
- 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 am using ImageMagick - the Java interface to this.

- 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?

    I think I will have a screen for the server side, the client side of stored data, and buttons to do image processing (see above).

- 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?

    The only preference will be the alarm and that will be on the client side.

- 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 plan to use services with 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.

- 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.

    I plan to have the user initiate the syncrhonization but it may be a simpler UI to have the background service synchronize the images. I plan to use a Content Provider for the images and store them in the Gallery.