"Since we cannot change reality, let us change the eyes with which we see reality."
- Nikos Kazantzakis

The Daily Selfie with Concurrent Image Processing

The Concurrent Daily Selfie application is an extension of one of the mini-projects in Professor Porter's MOOCs, which enables users to take pictures of themselves - selfies - over an extended period of time. It periodically reminds the user to take a selfie and presents the selfies in a list that makes it easy to see how the user has changed over time. This extended implementation also allows users to process their selfies to add effects, such as blurring or charcoaling. The image processing is done via a remote web service. In addition, all interactions between the device and the remote web service should be done concurrently in the background to ensure that the UI thread on the device is not interrupted.

Basic Project Requirements

Any Capstone project must support multiple users and should leverage one or more services running remotely in the cloud. Each project's specification clearly outlines the app's intended high-level behavior, yet leaves substantial room for individual creativity. Students will therefore need to flesh out many important design and implementation details. Basic requirements for all Capstone MOOC project specifications include:

1. Apps must support multiple users via individual user accounts. At least one user facing operation must be available only to authenticated users.
2. App implementations must comprise at least one instance of at least two of the following four fundamental Android components: Activity, BroadcastReceiver, Service, and ContentProvider.
3. Apps must interact with at least one remotely-hosted Java Spring-based service over the network via HTTP/HTTPS.
4. 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.
5. 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. In addition, experienced students are welcome to use other advanced capabilities not covered in the specialization, such as BlueTooth or Wifi-Direct networking, push notifications, or search, as long as they also use at least one advanced capability or API from the list above. Moreover, projects that are specified by commercial organizations may require the use of additional organization-specific APIs or features not covered in the MoCCA Specialization. In these cases, relevant instructional material will be provided by the specifying organization.

6. 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.

There may also be additional project-specific requirements (e.g., required use of a particular project-specific API or service).

Basic Functional Description and App Requirements for Concurrent Daily Selfie

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.
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.*
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.
4. The *User* can process *Selfies* by applying one 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.
5. *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.

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?
- 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.?
- 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?
- 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?
- 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)?

- 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.)?
- 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*?
- 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.