"Impatience is a virtue." Ursula Burns

ImPatient - A Sample Capstone Project

In a radiation therapy treatment center, patient appointments are typically scheduled in 15 minute slots, due to the nature of the equipment used in treatment. When patients arrive early or late, the appointment list often gets shuffled to accommodate the patients who are ready for treatment. Although this scheduling model is intended to make the best use of the equipment required for treatment, it can lead to confusion amongst patients and medical providers. The ImPatient app is designed to help optimize the time of both patients and medical providers. In particular,the overall goals of the ImPatient app are to (1) maximize the utilization of the radiation treatment machines, while (2) not unduly delaying patients from being treated at (roughly) their scheduled times. Thus, if a patient wants to take a short break the ImPatient app should try to reschedule another waiting patient (ideally in FIFO order, assuming this other patient has checked in), without making the original patient go to the end of the line - but instead reschedule them as soon as feasible to preserve as much "FIFOness" as possible.

Basic Project Requirements

Any potential Capstone project must support multiple users and should leverage 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.
2. At least one user facing operation must be available only to authenticated users.
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.
4. Apps must interact with at least one remotely-hosted Java Spring-based service
5. Apps must authorize the user via HTTP/HTTPS.
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 ListView 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.)

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. 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. 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.
8. Apps must support at least one operation that is performed off the UI Thread in one or more background Threads or a Thread pool.

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 ImPatient

1. The *Patient* is the primary user of the mobile app. A *Patient* is a unit of data that contains the core set of identifying information about a patient including (but not necessarily limited to) a first name, a last name, a date of birth, medical record number, and appointment time(s).

2. The *Patient* will use the app to *Check In* when they arrive at the waiting room for the department. The *Check-In* process will verify patient identity and appointment time and add the *Patient* to the *Queue*.

3. The *Queue* is a list of patients waiting to be seen. The app will update the *Queue* in first-in first-out (FIFO) order of arrival and message each *Patient* their expected waiting time based on their position in the *Queue*.

4. If the *Patient* needs additional time (to go to the bathroom, to get changed, etc.) they can delay their appointment time in 5 minute increments. The *Queue* is adjusted accordingly.

5. When a *Patient* approaches the treatment area their identity is again verified. After 15 minutes, their treatment is finished, and they are removed from the queue.

6. An *Admin* is a user type of the app that includes receptionists, doctors, nurses, etc. The *Admin* should be able to view the waiting room *Queue*, mark a *Patient* as waiting, being seen, or seen (removed from queue), and adjust *Patient* wait times manually.

Implementation Considerations

- How will a *Patient* enter identifying information upon *Check In*? In particular, what will the user interface look like for a *Patient* so that it is quick and simple to use?
- How will *Check-In* data be transferred from a *Patient*'s device to the *Admin*? Will this be done via the app or not?
- How will a *Patient's* data be securely transferred to the server?
- How will you ensure that *Patient* data is secure and shared only with the *Admins*?
- How will an *Admin* be notified that a *Patient* has performed a *Check-In*? In particular, what will the user interface look like for an *Admin* so that it is quick and easy to use and update?
- How, when, and how often will the user enter their user account information? For example, will the user enter this information each time they run the app? Will they specify the information as part of a preference screen?
- What user preferences can the user set? How will the app be informed of changes to these user preferences?
- How will the app handle concurrency issues, such as how will periodic updates occur - via server push or app pull? How will queries and results be efficiently processed? Will the data be pulled from the server in multiple requests or all at one time?
- How will the app use at least one advanced capability or API listed above? For example, will you create an animation to explain the app? Will you allow *Patients* to take/share pictures of themselves so that an *Admin* will know who they are and what they look like?
- 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 data, only when the device is connected to a WiFi network.