Symptom Management - A Sample Capstone Project

"Sometimes our light goes out, but is blown again into instant flame by an encounter with another human being."
— Albert Schweitzer

Over half of head and neck cancer patients will receive chemotherapy and radiation therapy for 6-7 weeks. During this time, symptoms can flare suddenly. The worst symptom for patients is radiation-induced oral pain and sore throat. Between the 2nd and 4th week of treatment, pain may increase from nothing to severe (i.e., "I'm swallowing glass") in a matter of days. Once the pain has escalated to this point, patients stop eating and drinking and often must undergo a surgical procedure for the placement of a feeding tube.

Week to week, doctors rely largely on a patient's memory to intervene and manage symptoms, but patient recall is notoriously unreliable, and weekly visits can leave symptoms sub-optimally managed for days.

"Symptom Management" is a smartphone-based application which provides a more reliable method to assess symptoms in near real-time. It provides patients a simple way to work with their healthcare providers to improve management of pain and quality of care.

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:

- Apps must support multiple users via individual user accounts. At least one user facing operation must be available only to authenticated users.
- App implementations must comprise at least one instance of at least two of the following four fundamental Android components: Activity, BroadcastReceiver, Service and ContentProvider.
- Apps must interact with at least one remotely-hosted Java Spring-based service over the network via HTTP.

- 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.
- 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.
- 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 Symptom Check

1. The *Patient* is the primary user of the mobile app. A *Patient* is a unit of data which 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, and a medical record number.
2. The patient will receive a *Reminder* in the form of alarms or notifications at patient-adjustable times, at least four times per day.
3. Once the patient acknowledges a *Reminder*, the app will open for a *Check-In*. A *Check-In* is a unit of data associated with a *Patient*, a date, a time, and that patient's responses to a set of questions (outlined in items 4-8) at that date and time.
4. During a *Check-In*, the patient is asked, "How bad is your mouth pain/sore throat?" and can respond with "well-controlled," "moderate," or "severe."
5. During a *Check-In*, the patient is asked, "Did you take your pain medication?" and can respond with "yes" or "no".
6. If a patient is taking more than one type of pain medication, each medication will require a separate question during a *Check-In* (e.g., "Did you take your Lortab?"

followed by “Did you take your OxyContin?”). The patient can respond to these questions with “no” or “yes.”

7. During a *Check-In*, if a patient indicates he or she has taken a pain medication, the patient will be prompted to enter the time and date he or she took the specified medicine.
8. During a *Check-In*, a patient is asked, “Does your pain stop you from eating/drinking?” To this, the patient can respond, “no,” “some,” or “I can’t eat.”
9. A *Doctor* is defined as a different type of user with a unit of data including identifying information (at least first name, last name, and a unique doctor ID) and an associated list of *Patients* that the doctor can view a list of. A doctor can login.
10. A patient’s *Doctor* is able to monitor a Patient’s Check-Ins with data displayed graphically. The data is updated at some appropriate interval (perhaps when a *Check-In* is completed).
11. A *Doctor* can see all his/her Patients and search for a given Patient’s Check-In data by patient name.
12. A *Doctor* can update a list of pain medications associated with a Patient’s account. This data updates the tailored questions regarding pain medications listed above in (6).
13. A *Doctor* is alerted if a Patient experiences 12 hours of “severe pain,” 16 hours of "moderate" to "severe pain," or 12 hours of “I can’t eat.
14. A *Patient’s* data should only be accessed by his/her Doctor(s) over HTTPS.

Implementation Considerations

- How will Check-In data be transferred from a Patient’s device to his/her Doctor?
- How will you store the data on the server-side so that patients are associated with doctors? You may use a hard-coded set of patient and doctor user accounts provided by an in-memory UserDetailsService. You may also hard-code the relationship between patients and doctors.
- What will the user interface look like for a Patient so that it is quick and simple to use?
- How will Reminders be delivered to a Patient in a way that will help the Patient to use the app more frequently and consistently?
- How will a Patient’s data be securely transferred to the server?
- How will a Doctor be able to update medication lists for a specific Patient, and how will these updates be sent to that Patient’s device?
- What will the user interface look like for a Doctor?
- How will a Doctor be alerted if a Patient has alarming symptoms?

- 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 search 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 pictures of mouth sores when their pain is high? Will you use push notifications to prompt users for pain information? Will you allow users to employ simple gestures to snooze pain prompts for a set amount of time?
- 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.