Android Capstone

Introduction

The purpose of this document is to clearly describe how the “Symptoms Management” project is intended to implement all the requirements defined in Coursera.org site. The reader should be able to understand, a general overview on how the solution will be implemented, how the client app will look like, the use cases that will be implemented, the available api implemented on the backend and the traceability between the different areas.

General Approach

In the scope of this Capstone, two major actors are identified: a **Patient** and his/her **Doctor** and one single Android application (application from now on) will be created to handle the use cases for both actors.

When the application is started for the first time it will ask the user to login with his/her existing account id (Patient Record ID or Doctor ID). In this implementation it is not possible to create new accounts, the user is just limited to login with existing accounts.

The application will follow the general approach applications are following nowadays which means a top bar (action bar), a navigation drawer on the left which contains different categories depending on the account in use, menu options that change depending on the category on the action bar and search option to filter out contents per category

Categories are:

- For Doctor type: Patients, Medicaments, Profile, About and Sign out.
- For Patient’s type: Check-In History, Reminders, Profile, About and Sign out

The application will handle the reminders using the alarm manager to trigger reminders to the user at a predefined frequency for the patient.to perform the checkin. The patient can also

tweaks the frequency but cannot reduce them below the minimum frequency (4 hours). For the purpose of showing that the alarm mechanism is working, each time the user download the reminder configuration from the server and save it locally, the alarm mechanism is used to remind the user that he/she can check-in. After that initial reminder, the application will remind the patient after the number of hours the reminder was configured

Each time the application synchronizes with the backend it downloads all the needed information so it can operate offline until it synchronizes again.

When the user performs a check-in, the application will try to connect immediately to the backend. If the operation is not feasible, the check-in will be locally saved and will be sent to the cloud on the next sync.

Assumptions and Restrictions for the Capstone

The following restrictions and assumptions are:

- The application can be used only in the context of the treatment defined in the definition of this project. Nevertheless the application can easily be extended for other treatment's types.
- The Spring services will contains the list of patients, doctors, which patient is treated by each doctor, inventory of medicines, inventory of common questions and sample data for test cases and for demo usage. Nothing will be persisted on real DBs and changes will be lost if the server is tear down.
- Each patient can be treated by one doctor.at the time.
- All the added medicaments will be shared among all doctors. This means if any doctor chooses to add a new medicament, another doctor can make use of that.
- The system offers the doctor the possibility of setting up custom questions so it can be used for one or more patient. This means that each doctor can tweak the way he/she follows up their patients. Custom questions created by one doctor are also shared among all available doctors.
- The client application can be used in more than one device at the same time. The sync mechanism will ensure that devices get "in-sync" between each other with the Cloud state.

Technical Approach

The application will be organized in using the following package structure. The base package will be com.coursera.android.capstone.sm.

Package	
<base>.ui.activity <base>.ui.fragment	All the activities and fragment will be put here. Some activities will use more than one fragment.
<base>.controllers	Contains the classes that enable other classes to

	used the functionality covered by this application. For example, the PatientController class let other classes inside the system to interact with the functionality related with a Patient. Each class under this package is a singleton.
<base>.extras	Contains a class named BundleExtras with constants for the values that are used inside intents, bundles and arguments.
<base>.model	Contains all the classes that represent the entities used in this project. Samples are: Answer, CheckIn, Medicine, etc.
<base>.notification	Provide a helper class to launch notifications from the different places a notification needs to be launched.
<base>.persistence	Provide a simple persistence mechanism for all the check-in entries executed by the patient. A single check-in will be maintained into this storage until the application is successfully synchronized with the backend.
<base>.provider	Currently contains only the stub provider needed by the Sync mechanism.
<base>.receiver	Contains the receiver used by the Alarm manager when one alarm is executed. It also contains a helper class with utility methods used to during the communication between intent services, fragments and activities.
<base>.services	Contains the intent service used to interact with the backend and all the possible commands (runners) used. The command and factory pattern are used here.
<base>.stats	Contains the classes responsible for creating the graphical stats based on the check-in information.
<base>.sync	Contains the classes needed to make the sync mechanism work.
org.magnum.videoup.client.oauth & org.magnum.videoup.client.unsafe	Packages borrowed from the client application to perform oauth authentication using a self-signed certificate.

IMPORTANT!!! Since many used cases consist in executing an action from the UI that implies sending and retrieving data to and from the cloud an IntentService will be used to perform the rest operation against the Sprint backend using Retrofit framework and then notify the result using LocalBroadcastReceiver. The UI components (Activity/Fragments) has the responsibilty of register a broadcast receiver for the OPERATION that it is interested for that particular functionailty and update properly according to the messages received.

Supported Use Cases

In this section all use cases needed to support the Symptoms management specs are detailed.

Doctor

#	Description
1	List available patients
2	Retrieve Patient's profile from doctor's view
3	List Patient's medicines
4	Update Patient's medicines
5	List Patient's checkins history
6	View the answers for a particular checkin entry.
7	Display the statistics for all the checkin entries for a particular patient
8	View the custom questions that are related with a particular patient
9	Add new custom question
10	Update an existing custom question
11	Delete an existing custom question
12	View Doctor's profile
13	Update Doctor's profile
14	List available medicines
15	Add new medicine
16	Delete existing medicine
17	Login as Doctor

18	Display Warning for 12 hours or more of severe pain, 16 hours of moderate pain, 12 hours of can't eat.

Patient

#	Description
19	List Patient's Checkin History
20	List checkin details
21	List Reminders
22	Update Reminder
23	Perform CheckIn
24	Login as Patient
25	View Patient's Profile
26	Edit Patient's Profile

Both Patient and Doctor can sign out from their account.

Wireframes

Wireframes are explained in the separate document and are splitted in the doctor's view and the patient's view.

Each use case listed in the previous section contains a 1-1 match with the screens presented in the wireframes.

Backend API

All the following APIs will point to a single ENDPOINT for the purpose of this demo. Initially, this will be the IP of the box where the solution is tested. This means that if the box has an IP 192.168.0.100, one or more phones in the same network will be able to test the solution end to end. If this endpoint happens to be on the Internet, anyone can access the solution from anywhere in the world.

#	Use Case	Action + URI	Description
BE-1	1	GET /patient	Retrieve the list of patients.
BE-2	2	GET /patient/<id>/profile	Retrieve the profile for a particular patient.
BE-3	26	POST /patient/<id>/profile	Change profile's fields.
BE-4	5,19,20	GET /patient/<id>/checkin	Retrieve the check-in history for a particular patient.
BE-5	23	POST /patient/<id>/checkin	Perform a new check-in.
BE-6	6	GET /patient/<id>/checkin/<checkin-id>/response	Retrieve all the responses for a particular checkin
BE-7	12	GET /doctor/<id>/profile	Retrieve the profile of the doctor
BE-8	13	POST /doctor/<id>/profile	Update the profile of the doctor
BE-9	21	GET /patient/<id>/reminder	Retrieve the list of reminders. There will be at this point one per patient that can be modified by himself/herself.
BE-10	22	POST /patient/<id>/reminder	Update the reminder with the specified data.
BE-11	3	GET /patient/<id>/medicine	Retrieve the list of all the medicines the patient is taken.
BE-12	4	POST /patient/<id>/medicine	Update the list of medicines the patient have to take.
BE-13	4	DELETE /patient/<id>/medicine/<medicineId>	Remove a medicine from the patient profile
BE-14	14	GET /medicine	Retrieve the list of medicines available for all the patients
BE-15	15	POST /medicine	Add a new medicine to the list of available medicine. This can be added by any doctor.
BE-16	16	DELETE /medicine/<id>	Remove existing medicine
BE-17	17,24	POST /oauth/token	Login the user (patient/doctor) depending the ID used.
BE-18	8	GET /questions	Retrieve all available custom questions that can be asked to a patient

BE-19	9,10	POST /questions	Add new custom question
BE-20	8	GET /patient/<id>/question	Retrieve the custom questions that a patient can take
BE-21	9,10	POST /patient/<id>/question	Add new custom question to the patient's profile
BE-22	11	DELETE /question/<id>	Remove existing question
BE-23	11	DELETE /patient/<id>/question/<questionId>	Remove a quesiton from the patient profile
BE-24	17,24	GET /role	Identify whether the passed ID represent a DOCTOR or a PATIENT

For the purpose of this capstone, all the communication between the android clients and the servers will work over HTTPS.

Answers to implementation Considerations

- **How will Check-In data be transferred from a Patient's device to his/her Doctor?**
  Using a combination of techniques. Check-In data will be sent immediately. If there is no network at the time of Check-in, data will be synchronized offline using Android sync mechanism.

- **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.**
  Using an in-memory implementation. Account creation will NOT be permitted. Doctors, Patients and Doctor-Patient's relationships will be hardcoded. The initial Reminder configuration will also be hardcoded.

- **What will the user interface look like for a Patient so that it is quick and simple to use?**
  Check Wireframes Section for this.

- **How will Reminders be delivered to a Patient in a way that will help the Patient to use the app more frequently and consistently?**

Reminders will have an initial configuration and then can be customized by the Patient. After the Patient changes the Reminder, changes will be saved locally.

- **How will a Patient's data be securely transferred to the server?**
  Using HTTPS

- **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?**
  Once the doctor updates the medication list and update the cloud, on the next sync, the patient will receive the updates on such changes.

- **What will the user interface look like for a Doctor?**
  Check the wireframes

- **How will a Doctor be alerted if a Patient has alarming symptoms?**
  After a synchronization occurs, the downloaded data will be evaluated to identify scenarios that requires alerts for the doctor. A notification will be launched indicating which patient has which alert.

- **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?**
  The patient will enter the account information only once per device.

- **What user preferences can the user set? How will the app be informed of changes to these user preferences?**
  The patient will be able to customize the frequency of the alarms that needs to be used when the alarm is fired.

- **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?**
  Each type of request to the cloud will be performed using an IntentService. For this demo,

app will do data pull via sync. Once the responses are received from the cloud, they will be processed in the IntentService's thread (or a proper delegate) and then the content provider will save such data and notify the proper URI for changes. The UI will have the responsibility to update the data accordingly.

- **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?**
  Animations will be use for fragment transitions. Android Sync will be used.

- **Will you allow patients to take pictures of mouth sores when their pain is high?**
  No

  Will you use push notifications to prompt users for pain information?
  No for the capstone. Having GCM implemented is a good improvement. and should be used in a real implementation.

  Will you allow users to employ simple gestures to snooze pain prompts for a set amount of time?
  No for the capstone.

- **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**
  The application will contains Activities, Fragments, BroadcastReceivers and IntentServices. The app will sync also data on both Mobile and WIFI data. Android synchronization mechanism, LocalBroadcastManager and NavigationDrawer are also used.