Symptom Management - A Sample Capstone Project

Description

For this project I will create one Android Application and a Java server with Spring and a MongoDB database to store all the information for this project.

SPRING server and MongoDB database

Within the application, (server AND app side) data are stored with theses JAVA Classes.

- The class AppUser is a basic class describing all users for this project. Doctor and Patient classes extend this class by adding their own attributes. “Login” attribute is the credential used by each user in the App. For the push notification described later, I included also the attribute “regId” which is the registration Id on Google Servers. This attribute can be null because Patients don't need it.

```
public class AppUser {
    protected String name;
    protected String lastname;
    protected String login;
    protected String regid;
```

- A doctor is identified with his (unique) ID and his last and first names and his login (homonyms are supported with that). There is no patient’s list in the Doctor attributes. If I want to get all patients for one Doctor. I just had to make a request to Patient repository with the Doctor Entity.

```
public class Doctor extends AppUser{

    @Id
    private long id;
```

Requests available for the Doctor Repository are:
Get Doctor List : It sends the list of all Doctors available in database. (currently not used)
Save : It creates or update an Entity if it already exists or not.
Find by login : Return a Collection containing only one Doctor. User Login has to be unique to authorize each user individually. This request is used to fill the doctor profile after log in.

- Patients are represented with their ID, and names. There are also the birth date and medical Number as attributes. They have a medication list that contains the names of each medication they have to take. Each Patient has also the ID of his Doctor.

```java
public class Patient extends AppUser {

    @Id
    private long id;
    private String birthdate;
    private String medicalnumber;
    private ArrayList<String> medicationlist;
```

Request available:
Get Patient List : it give the list of all Patients in database. (not used)
Save : It creates or update an Entity if it already exists or not.
Find by login : return a Collection containing only the Patient who is trying to connect.
Find by Doctor : return the list of all Patients seen by a given Doctor.

- The Check-in contains the ID of the patient, the date+ time when it was created and all the answer given by the patient.

```java
public class CheckIn {

    @Id
    private long id;
    @DBRef
    private Patient patient;
    private String date;
    private PainState painstate;
    private Eating eating;
    private Set<Medication> medicationlist;

    public enum PainState{
        WELL_CONTROLLED,
        MODERATE,
        SEVERE
    }

    public enum Eating{
        NO,
        SOME,
        CANT_EAT
    }
```

The pain state and how the patient can eat are Enum structures:

For each medication, we need to create a sub-Classe that represents what and when the patient takes it.

```java
public class Medication {

    private String name;
    private boolean taken;
    private String date;
```

The Boolean “istaken” is here to store the answer “did you take your medication?”.

Medication Entity are not stored in a dedicated Document. They are stored directly in each CheckIn instance. That's because each patient can take a same medication a the same time.

Requests available for CheckIn repository are:
Get CheckIn List :
Add CheckIn : create a new CheckIn. For each new insertion, the server check all previous CheckIn for the patient who try to save this new one. And if the patient feels really bad. A notification is sent to his Doctor. (See Push Notification Section)
Find by Patient :

- UserRights class is here to manage users and theirs rights in the Application. The id is unique and based on the user login and the right for this user. Each user login is stored in this class and we give a Right for each. Rights type is an custom enum.

```
public class UserRights {

    @Id
    private long id;
    private String login;
    private Rights right;

    public enum Rights{
        DOCTOR,
        PATIENT,
        ADMIN
    }
```

Requests available:
get UserRights List : return the entire list of rights defined in the Application.
Save : create a new couple user login – Right.
Find by login : return the couple login – Right for a specified login. Used for retrieve the right when a user try to log in and choose the appropriate part of the application.

Android Application

Components used

- Activity to define all screens for the Application.
- BroadcastReceiver to handle alarms and push notification from Google Cloud Messaging
- IntentService to create and fire Android notification when alarms or notification are triggered.

- Initial view is a login screen in which the user (Patient or Doctor) give his authentication information. With this screen the app can select which mode it needs and which screen it can show to the user.
  A request is made to the Server. In the UserRights Table. Depending on the Rights type returned the next screen is chosen.

Illustration : LoginActivity.java

- The authentication procedure is shown when the user start the Application.

  TODO:If a had enough time, the idea is to let Android Account Manager handles App authentication. With this feature, the user will not be prompt to enter his credentials. But when, he need to log with another username. He has to delete his account in the Settings App. Java classes for this features are ready to use (SymptomMgmtAuthenticator and SymptomMgmtAuthService)

- The authentication is made by OAuth2.0 protocol. There are 8 patients and 3 doctors hard coded. To facilitate tests, username and password are identical.

  username/password Couples available:
  doc1/doc1
  doc2/doc2
  doc3/doc3
  sdurand/sdurand
  psmith/psmith
  jbeck/jbeck
  mdupond/mdupond
  jjacques/jjacques
  mtelo/mtelo
  tpumba/tpumba

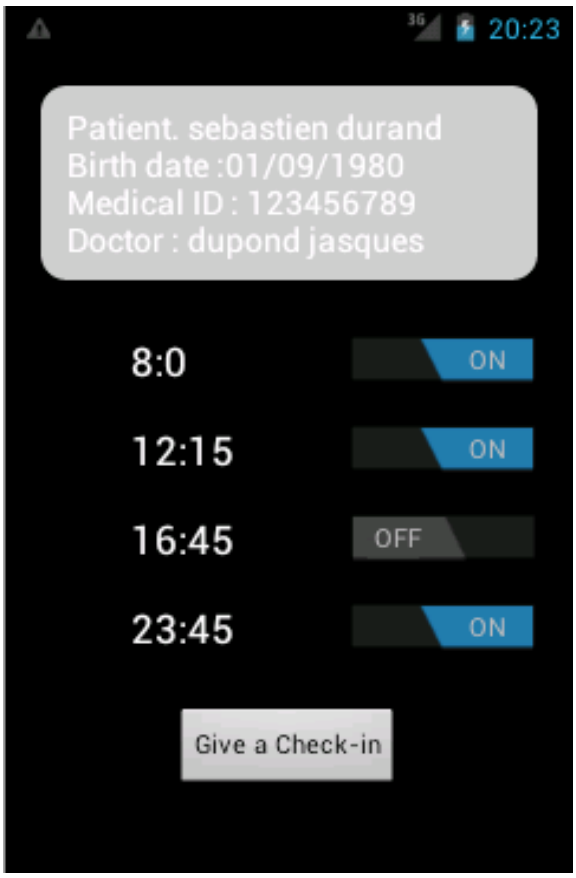
- Data (a new check-in created by a patient or the complete list of check-in for one patient) are downloaded in a background thread. (CallableTask.java and TaskCallback.java)

- To alert a doctor when a Patient is very bad, the server can push notification to the application.

- The application communicates with the server with HTTPS requests. (EasyHttpClient.java, SecuredRestBuilder and SecuredRestException)

- There are 2 modes in this application: the patient mode and the Doctor mode. For each mode, a list of screens is defined.

Patient mode



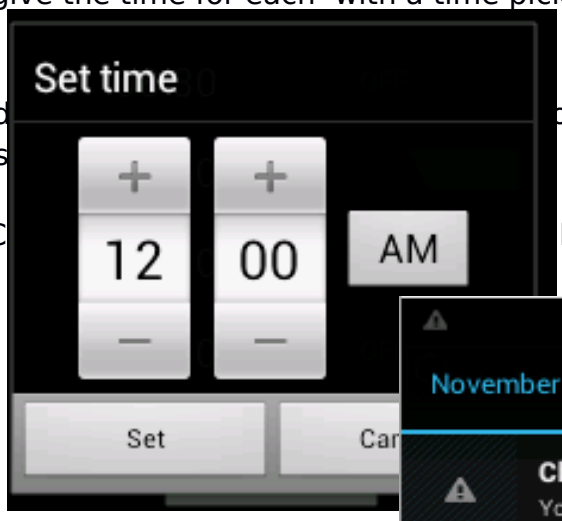
- In this mode, the user can only configure his daily alarms and achieve a check-in.

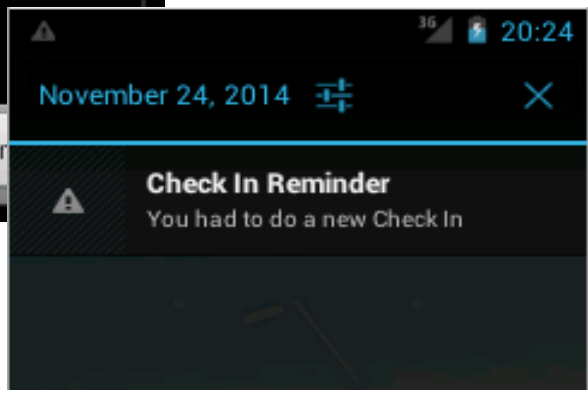
- The patient can see his personnal data stored in the database. (birth date, Medical number and his doctor)

- The configuration is made with the settings page shown above. We can manually configure 4 alarms and give the time for each with a time picker. Switches are here in order to activate

  – Each alarm is triggered d on is generated (the phone rings and vibrates

  – The broadcastReceiver C handle Alarms.



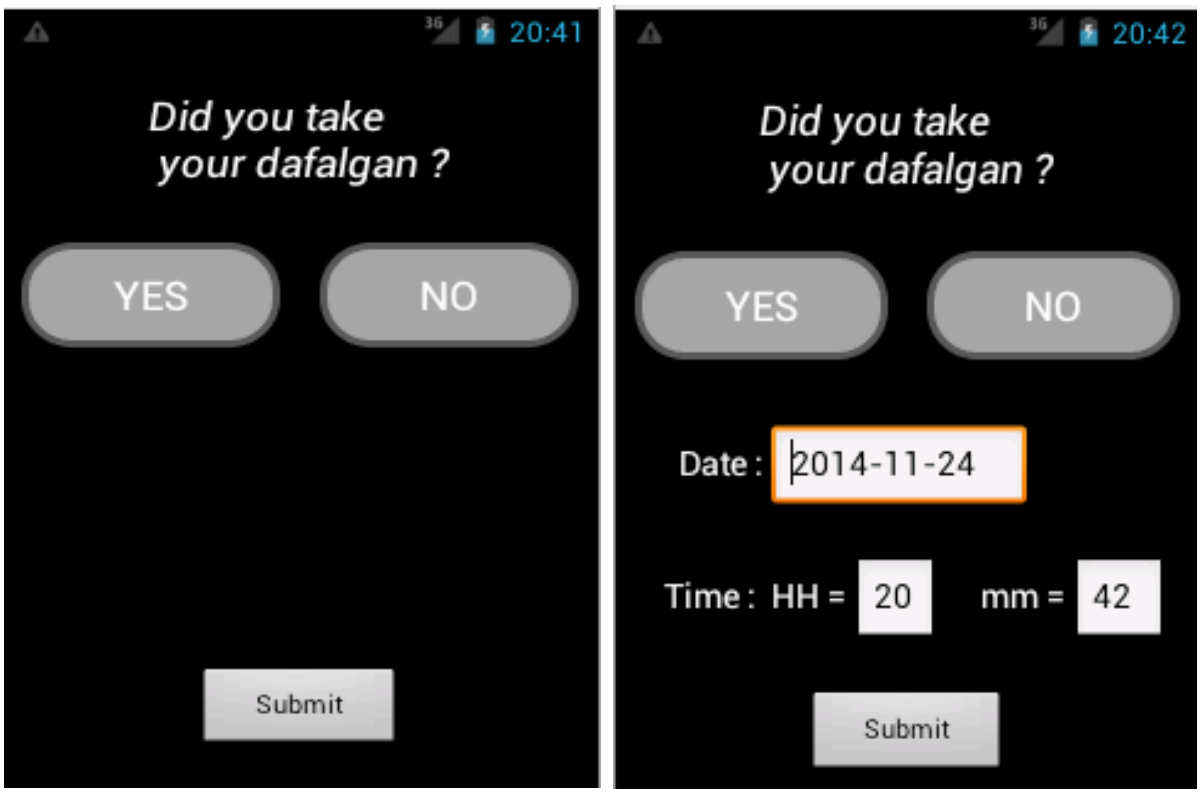


- When an alarm rings, the user need to acknowledge it and the application opens with the check-in screens. This is a set of at least 3 screens that let the patient give his feelings about his pain. The checkInActivity use a Pager Viewer. It lets the user switch between all the screens with a Right-to-Left or a Left-to-Right gesture.

- It uses CheckInActivity using CheckInPagerAdapter to display Fragments described bellow.

– The user needs to choose one answer. When he click on one choice, the next screen is shown immediately.
(CheckInQuestionFragment)

This last screen may be duplicated if there is more than one medication. For each element, the patient needs to say YES or NO and give the time when he takes his medication. If patient clikcs on “YES”, a date and a time picker is shown and automatically filled and the current time.



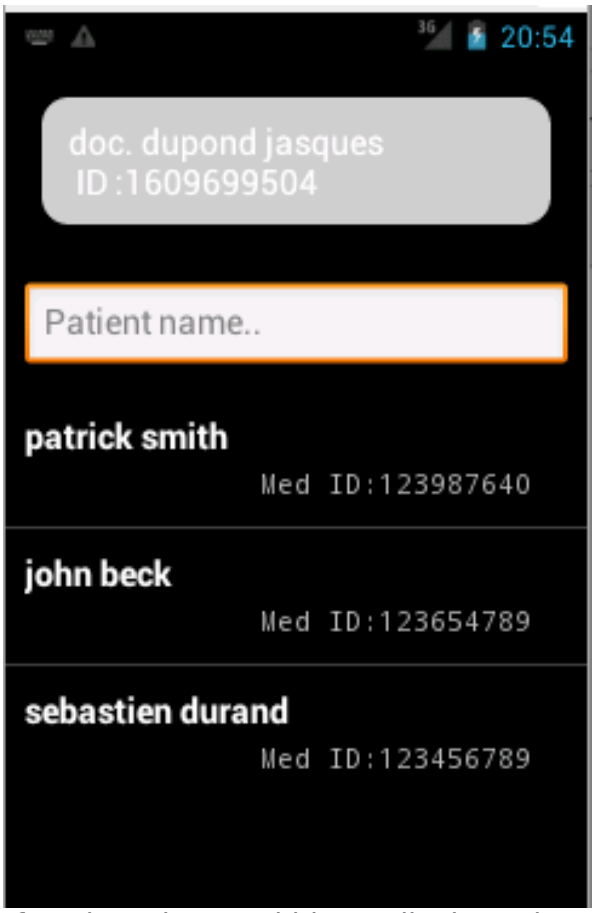
(CheckInMedicationFragment)

- When the check-in is complete, Toast message confirms that everything is OK and information will be sent to he doctor. It also gives a possibility to change his answer. The user can switch manually between all question if he wants to modify something.In this case, the information updated in the database.

Doctor mode

- This mode is a monitor for reviewing the patient’s problems. We can also see Doctor personnal data. (DoctorHomeActivity using the adapter PatientArrayAdapter to fill the listview)

- The home screen is a global view with a scrollable list of patients for the authenticated doctor. He can search for a patient by name. It’s a listView that

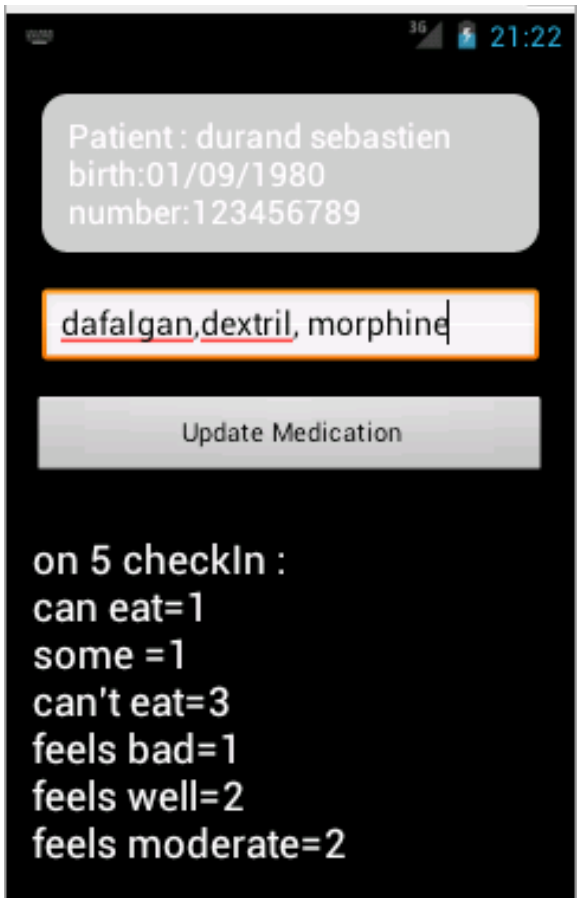


display the name of each patient and his medical number.

– There is a TextBox which lets the Doctor filter by Patient's name.

- Each element is clickable and its send the user to a screen in which we can view graphically the result of the last check-ins. (DoctorCheckPatientActivity)

- There is also a button to update the list of medication for this patient. All medication have to be written and separates with a coma.



Push Notification

Each time a CheckIn is created, the other old check-ins for this patient are scanned and if the patient is very bad. (see the following criterion) the server push a notification to the concerned Doctor.

The criterion are defined like that:

- If for the last 12h, the patient felt a “severe” pain
- If for the last 16h, the patient felt a “moderate” pain
- If for the last 12h, the patient answer “I can’t eat”

How does it work ?
A new project need to be created on Google Services. When a user logs in the App for the first time, a request is send to GCM servers to register this user. He receives a "registration id" which identifies the user. This id is sent back to our own server where it'll be stored. (UserApp “regId” attribute). The server will use this id as an address where to send a notification.

The server sends a POST request to Google's Server with the title and the message and the registration Id of the device that needs to receive the notification. Google takes this POST and transfer information to the device registered with the given Id.

Our App defines a Broadcast receiver that listen to these notifications (DoctorNotificationReceiver). When a notification is received, it's sent to the IntentService in which we call the Notification Manager the generated an Android notification. (DoctorNotificationIntentService)