

SYMPTOM MANAGEMENT

This is a design documentation for the Capstone Project Symptom Management for the Coursera Specialization on Mobile Cloud Computing with Android.

This pdf outline how I implemented the Symptom Management specifications and how my project implementation meets each requirement that is listed in the rubric we have from the Coursera seminar with title "Android Capstone Project". It will contain screen shots from my mobile phone plus a small description under each screen shot to understand what you see and what is happening. Also I will make a brief analysis of some mechanisms I use under the hood.

NOTE: This application is for the seminar under no circumstances use it for real world situations. The users, patient and doctor profiles will be hardcoded as this is correct as the Professors said. The patient and doctor profiles will use a dummy name and surname. All the information of the doctor and patient profile can be changed if the user selects edit profile from the action bar menu. All the changes will be saved in the database.

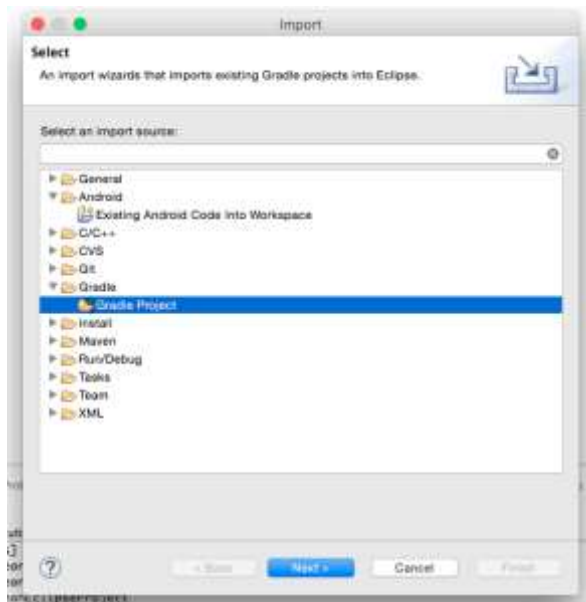
Also **NOTE:** The server side application uses a keystore I created. This keystore is for the seminar and it is NOT safe to use it in a real application. The keystore is placed inside the server side application (CapstoneProject) in the package src/main/resources inside the folder private with the name keystore.jks . To create the keystore I used the oracle documentation (<http://docs.oracle.com/cd/E19509-01/820-3503/ggfen/index.html>) In order to make the server side application to run you have to do the following:

IMPORT PROJECTS AND MAKE THEM RUNNABLE

Inside the zip file you will download from Coursera will be the following files:

- CapstoneProject (This is the server side application)
- Google-play-services_lib (This is the google play library. You need this in order to make the push notification to work)
- MPChartLib (This is the library for the Chart View that will display the patient data graphically)
- SymptomManager (This is the client application)
- A ReadMe file that will tell how to add all the projects correctly as I say below here.

First import the CapstoneProject as a gradle project. Right click in eclipse import. Select gradle project and browse to the CapstoneProject. After that build model and then click finish.



Second import the SymptomManager as Existing Android Code Into Workspace. Browse to SymptomManager folder and click finish



Third import the google-play-services_lib as Existing Android Code Into Workspace. Browse to google-play-services_lib folder and click finish.

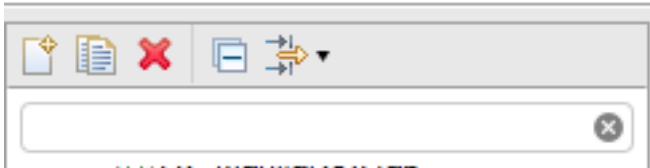
Forth import the MPChartLib as Existing Android Code Into Workspace. Browse to MPChartLib folder and click finish.

Fifth right click on SymptomManager application in eclipse -> Properties. Remove the Library references which will be with X red mark. Click Add, select the google-play-services_lib project from the window and click OK. Then click Add, select the MPChartLib project from the windows and click OK. Everything should be build OK now!

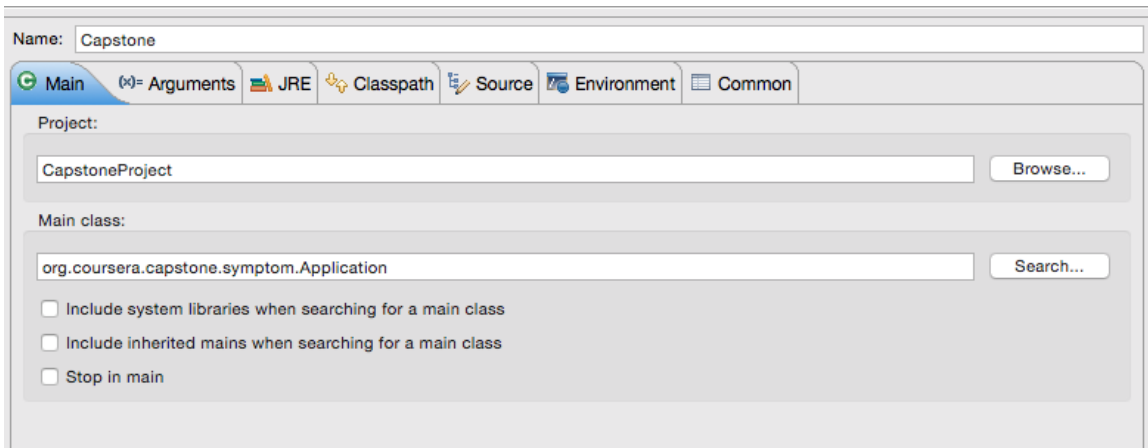
In eclipse open the tab Run -> Select the Run Configurations... -> there select Java Application from the list and Create New Launch Configuration from the first icon in the left of the window

Create, manage, and run configurations

Android Application



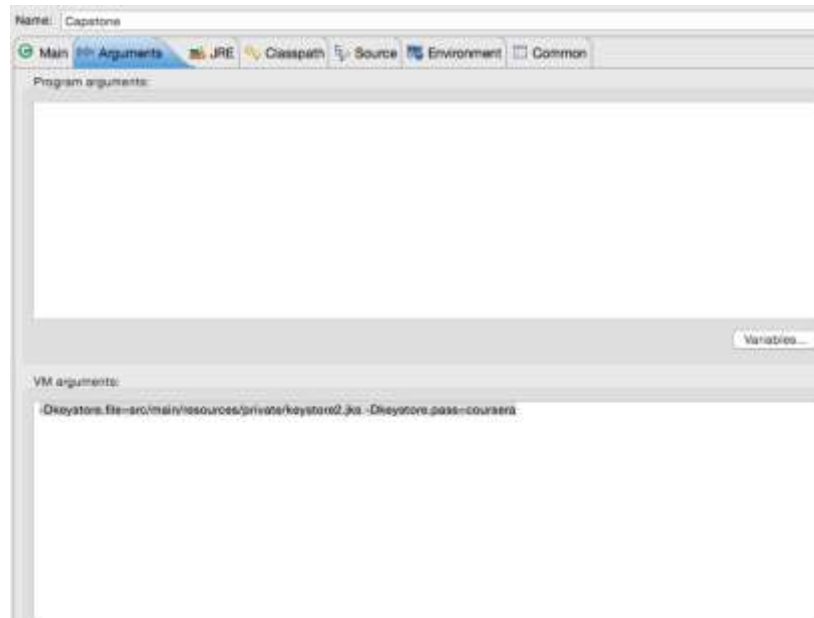
or right click select New. There in the tab Main (the first one) Browse the project select CapstoneProject and in the Main class add the Application class that contains the main method



Then in the Arguments tab (the second one next to the Main tab) in the VM arguments add the following:

```
-Dkeystore.file=src/main/resources/private/keystore2.jks  
-Dkeystore.pass=courser
```

Here we are declaring the path of the keystore inside the project and the password for the keystore.



Then select run and the server will execute.

Also **NOTE:** The application because is for the seminar the server side runs locally so you have to run first the server side and then run the android application (SymptomManager). Because the server runs locally you have to do one thing in order to make the client runs OK. After the server is up and running, open the terminal and write the following command:

ipconfig (if you using windows)

ifconfig (if you using mac)

Then, find the ip address your pc runs on (e.g. 192.168.10.224) this is the ip where the server is live because the server runs locally in your machine. You will need this ip to enter it in the client side when the application runs for the first time. In the login screen it tells you to enter the ip address there and you enter this ip.

NOTE: The application uses as external libraries the following: (Will be inside the zip file you will download from Coursera)

a) MPChartLib (which is for the chart view in order the doctor to see his/hers patient data graphically)

b) google-play-services_lib (which is for the Push Notifications from Google to work. This is an official library from Google)

Also it uses inside assets a file “countries.dat” which is the file that contains all the countries with their country codes for the telephone number.

The project consists of a server side application written in Java Spring and a client side application written in Java Android.

First I will explain the server side application:

The server will use Spring OAuth 2.0 to support multiple users via a custom UserDetailsService. Users will authenticate via the Spring OAuth 2.0 token endpoint and provide bearer tokens in an authorization header to prove user identity. In order to use the application you should be successfully connect as one of the hardcoded users. For the storing of the data I ‘ m using JPA Repositories. One repository per object. The objects I use are the following:

Account, which is the account of the current user and has the following parameters:

Long id	<i>This is the id of the account</i>
String username	<i>This is the username of the account</i>
String password	<i>This is the password of the account</i>
String regId	<i>This is the registration id for the push notifications from Google</i>
String role	<i>This is the role of the account (eg doctor, patient)</i>

Checkin which is a checkin object with questions the patient had submitted has the following parameters:

Long id	<i>This is the id of the checkin</i>
String raw	<i>This is the raw data that will contain the questions with the selected answer and the selected timestamp for the second question if the patient had taken his/hers medication. The format of this string is in JSON</i>

Long idPatient	<i>This is the id of the specific patient that did this checkin</i>
Long time	<i>This is the timestamp of the checkin</i>

DoctorDetail which is a doctor detail object with information about the doctor:

Long id	<i>This is the id of the Doctor</i>
String username	<i>This is username of the doctor</i>
String name	<i>This is the name of the doctor</i>
String surname	<i>This is the surname of the doctor</i>
String email	<i>This is the email of the doctor</i>
String phone	<i>This is the phone of the doctor</i>
String regId	<i>This is the registration id for the push notification from Google</i>

PatientDetail which is a patient detail object with information about the patient:

Long id	<i>This is the id of the patient</i>
String username	<i>This is username of the patient</i>
String name	<i>This is the name of the patient</i>
String surname	<i>This is the surname of the patient</i>
String email	<i>This is the email of the patient</i>
String phone	<i>This is the phone of the patient</i>
String regId	<i>This is the registration id for the push notification from Google</i>
Long dob	<i>This is the timestamp of the date of birth of the patient</i>
String gender	<i>This is the gender of the patient</i>
Long lastCheckIn	<i>This is the timestamp of the last checkin the patient did</i>
String notifyTime	<i>This is the selected notify times the patient had specified when the application will prompt him/her to make a checkin (by default is 4 times the patient have to specify. The format of this</i>

string is in JSON. It contains a JSONArray with JSONObjects that hold a timestamp for the selected notify time.

Long idDoctor

The id of the patient's doctor

Long severePain

This is a timestamp of when the patient had severe pain. This is used in order to inform the doctor if the patient had 12 hours of severe pain)

Long moderatePain

This is a timestamp of when the patient had moderate pain. This is used in order to inform the doctor if the patient had 16 hours of moderate pain)

Long cantEat

This is a timestamp of when the patient cant eat. This is used in order to inform the doctor if the patient cant eat for 12 hours)

String uid

This is the unique medical record number

Medication which is a medication object with information about the medication:

Long id

This is the id of the medication

String name

This is the name of the medication

String description

This is the description of the medication

Long idPatient

The id of the patient and represent which patient has this medication

I ' m using 5 repositories one for each object. The parameters of the objects represent the fields in each repository.

Account Repository

ID**	USERNAME	PASSWORD	REGID	ROLE
------	----------	----------	-------	------

Checkin Repository

ID**	RAW*	IDPATIENT	TIME
------	------	-----------	------

*The filed RAW because is going to contain a big string (that is the questions of the specific checkin with the answers and a timestamp if the question is about a medication and if the patient answered yes in JSON format) I specified the annotation Lob that mean this filed should be persisted as a large object. And also I specified the length of the filed to be 1024.

Medication Repository

ID**	NAME	DESCRIPTION	IDPATIENT
------	------	-------------	-----------

DoctorDetail Repository

ID**	USERNAME	NAME	SURNAME	EMAIL	PHONE	REGID
------	----------	------	---------	-------	-------	-------

PatientDetail Repository

ID**	USE RN AM E	N A M E	SU RN AM E	E M A I L	P H O N E	D O B	GE N D E R	LAST C H E C K I N	NOTI F Y T I M E*	IDD O C T O R	R E G I D	SEV E R E P A I N	MODE R A T E P A I N	CA N T E A T	U I D
------	----------------------	------------------	---------------------	-----------------------	-----------------------	-------------	------------------------	---	-------------------------------------	------------------------------	-----------------------	--	--	-----------------------------	-------------

**The parameter id in each repository is a generated value with strategy of generation type AUTO. So here are the repositories.

*The filed NOTIFYTIME because is going to contain a big string (that is the selected notify times the patient would like to be prompt to make a checkin in JSON format) I specified the annotation Lob that mean this filed should be persisted as a large object. And also I specified the length of the filed to be 512.

I ' m using 3 controllers one for the patient one for the doctor and one general. Below are a UML diagram for each controller showing this controllers:

Patient Controller



As you can see I have 9 web services here that are associated with the patient. I'm using some String variables that hold the web services urls and 4 repositories that are Autowired. The Checkin repository the PatientDetail repository, the Medication Repository and the DoctorDetail Repository.

getProfilePatient(String username)

GET /{username}/getPatientProfile

Returns the PatientProfile after search the database for the given PathVariable username. This is the patient username

getDoctorprofile(long id)

GET /getDoctorProfile/{id}

Returns the DoctorProfile after serach the database for the given PathVariable id . This is the doctor id.

createProfilePatinet(PatientDetail)

POST /createPatientProfile

Returns the created PatientProfile after save the given RequestBody PatientProfile. This is the PatientProfile of the patient

populatePatientsProfiles(ArrayList<PatientProfile>)

POST /populatePatients

Saves all the hardcoded patient profiles (with dummy data name, surname, gender) in the database. NOTE I use this web service because I use hardcoded patients.

updateProfilePatient(PatientDetail)

POST /updatePatientProfile

Returns the updated profile of the patient. The given PatientProfile is in RequestBody

getMedication(long id)

GET /getMedication/{id}

Returns an ArrayList<Medication> after search the database for the given PathVariable id . This is the patient id.

sendCheckIn(long id)

POST /sendCheckin

Saves in the database the given checkin object. The object is sent like: RequestParam(id), RequestParam(raw), RequestParam(idPatient), RequestParam(time). Where id is the id of the checkin raw are the raw data of the checkin (see checkin object in the start of this pdf) idPatient is the id of the patient that make this checkin and time is the timestamp of the checkin.

sendPushDoctor(long id, String title, String message)

POST /sendPush/{id}

Search the database with the given PathVariable id. This is the doctor id and finds the DoctorDetail object. There takes the registration id of the doctors device and send the push notification via the GCM Google servers with the title and message the RequestParam(title) and RequestParam(message)

addRegIdToPatient(long id, String regId)

POST /addRegId

Returns the patient profile after we have updated and saved it in the database with the given RequestParam(regId) and RequestParam(id)

Doctor Controller

<<Java Class>>	
 DoctorController	
org.coursera.capstone.symptom.controllers	
Δ ^F GET_DOCTOR_PROFILE: String	
Δ ^F UPDATE_DOCTOR_PROFILE: String	
Δ ^F CREATE_DOCTOR_PROFILE: String	
Δ ^F GET_PATIENT_LIST: String	
Δ ^F UPDATE_PATIENT_PROFILE: String	
Δ ^F REMOVE_PATIENT: String	
Δ ^F GET_AVAILABLE_PATIENTS: String	
Δ ^F GET_PATIENTS_CHECKINS: String	
Δ ^F GET_PATIENT_PROFILE: String	
Δ ^F ADD_MEDICATION: String	
Δ ^F REMOVE_MEDICATION: String	
Δ ^F SEND_PUSH_2_PATIENT: String	
Δ ^F REGID_PARAMETER: String	
Δ ^F POPULATE_DOCTOR_DETAILS_TABLE: String	
Δ ^F FIND_PATIENT_BY_NAME: String	
Δ ^F FIND_PATIENT_BY_SURNAME: String	
Δ ^F ADD_REGID: String	
□ checkInRepo: CheckInRepository	
□ doctorDetailRepo: DoctorDetailRepository	
□ patientDetailRepo: PatientDetailRepository	
□ pillRepo: MedicationRepository	
● ^C DoctorController()	
● getProfileDoctor(String,HttpServletResponse):DoctorDetail	
● createProfileDoctor(DoctorDetail,HttpServletResponse):DoctorDetail	
● populateDoctorsProfiles(ArrayList<DoctorDetail>,HttpServletResponse):ResponseEntity<Void>	
● updateProfileDoctor(DoctorDetail,HttpServletResponse):DoctorDetail	
● getPatientList(long,HttpServletResponse):ArrayList<PatientDetail>	
● addPatient(long,long):ResponseEntity<Void>	
● removePatient(long):ResponseEntity<Void>	
● getAvailablePatients(HttpServletResponse):ArrayList<PatientDetail>	
● getCheckins(long,HttpServletResponse):ArrayList<CheckIn>	
● getPatientProfile(long,HttpServletResponse):PatientDetail	
● addMedication(long,String,String,long):ResponseEntity<Void>	
● removeMedication(long):ResponseEntity<Void>	
● sendPushPatient(long,String,String):ResponseEntity<Void>	
● addRegIdToDactor(long,String,HttpServletResponse):DoctorDetail	
● findByNameContaining(String):ArrayList<PatientDetail>	
● findBySurnameContaining(String):ArrayList<PatientDetail>	

As you can see I have 16 web services here that are associated with the doctor. I'm using some String variables that hold the web services urls and 4 repositories that are Autowired. The Checkin repository the PatientDetail repository, the Medication Repository and the DoctorDetail Repository.

getProfileDoctor(String)

GET /{username}/getDoctorProfile

Returns the DoctorProfile after search the database for the given PathVariable username. This is the doctor's username

createProfileDoctor(DoctorDetail)

POST /createDoctorProfile

Returns the created DoctorProfile after save the given RequestBody DoctorProfile. This is the DoctorProfile of the doctor

populateDoctorsProfiles(ArrayList<DoctorProfile>)

POST /populateDoctors

Saves all the hardcoded doctor profiles (with dummy data name, surname) in the database. NOTE I use this web service because I use hardcoded doctors.

updateProfileDoctor(DoctorDetail)

POST /updateDoctorProfile

Returns the updated DoctorDetail after is been saved in the database. The given DoctorDetail is in RequestBody.

getPatientList(long id)

GET /{id}/getPatientList

Returns an ArrayList<PatientDetail> of the given PathVariable id. The id is the doctor id. Returns all the patients of the selected doctor

addPatient(long id, long id2)

GET /{id}/updateProfilePatient/{id2}

Save the doctor's id (id2) to the patient detail found by searching the database with the id . Both id and id2 are PathVariables

removePatient(long id)

GET /removePatient/{id}

Set the selected Patient found by searching the database with the given PathVariable (id) and setting the doctorId of the PatientProfile to -1.

getAvailablePatients()

GET /getAvailablePatients

Returns an ArrayList<PatientDetail> after searching the database for all the patient with doctor id = -1

getCheckins(long id)

GET /getCheckins/{id}

Returns an ArrayList<CheckIn> of the checking the patient did after searching the database with the given PathVariable (id). Id is the patient's id

getPatientProfile(long id)

GET /getPatientProfile/{id}

Returns the PatientDetail object after searching the database for the given PathVariable(id) . The id is the patient id.

addMedication(long id, String title, String description, long idPatient)

POST /addMedication

Saves to the database the given medication. The medication is in the form of RequestParam(id), RequestParam(title), RequestParam(description), RequestParam(idPatient).

removeMedication(long id)

GET /removeMedication/{id}

Deletes the selected medication which is found after searching the database with the given PathVariable(id). The id is the id of the medication in the database

sendPushPatient(long id, String title, String message)

POST /sendPush/{id}

Search the database with the given PathVariable id. This is the patient id and finds the PatientDetail object. There takes the registration id of the patient's device and send the push notification via the GCM Google servers with the title and message the RequestParam(title) and RequestParam(message)

addRegIdToDoctor(long id, String regId)

POST /addRegId

Returns the doctor's profile after we have updated and saved it in the database with the given RequestParam(regId) and RequestParam(id)

findByNameContaining(String str)

GET /searchByName/{str}

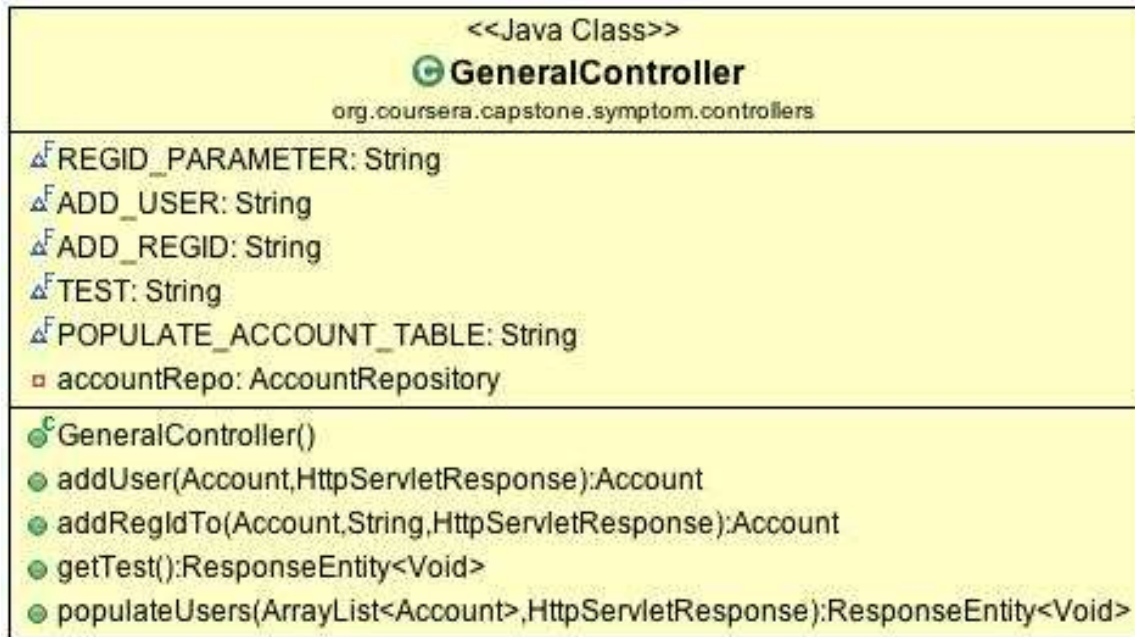
Searches the database and returns an ArrayList<PatientDetail> that their name contains the PathVariable(str)

findBySurnameContaining(String str)

GET /searchBySurname/{str}

Searches the database and returns an ArrayList<PatientDetail> that their surname contains the PathVariable(str)

General Controller



As you can see I have 4 web services here that are used by the doctor and the patient. I'm using some String variables that hold the web services url and one repository that is Autowired. Account Repository.

addUser(Account)

POST /addUser

Saves the given RequestBody Account object to the database

addRegIdTo(Account ,String)

POST /addRegIdTo

Returns the updated RequestBody(Account) with the RequestParam(regId)

getTest()

GET /test

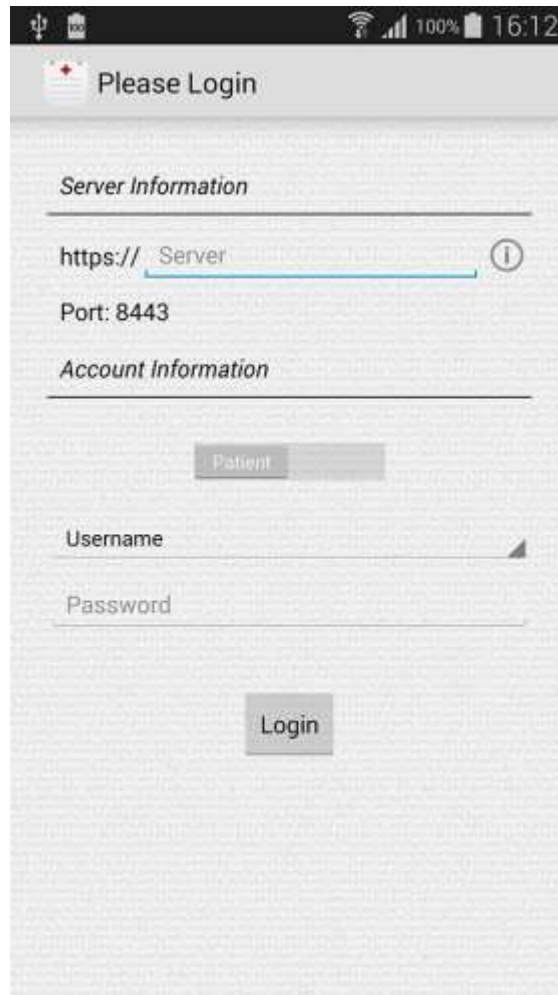
This is a test web service I use it in order to check if the user is authenticated

populateUsers(ArrayList<Account>)

POST /populateUsers

Saves all the hardcoded users in the database. NOTE I use this web service because I use hardcoded users.

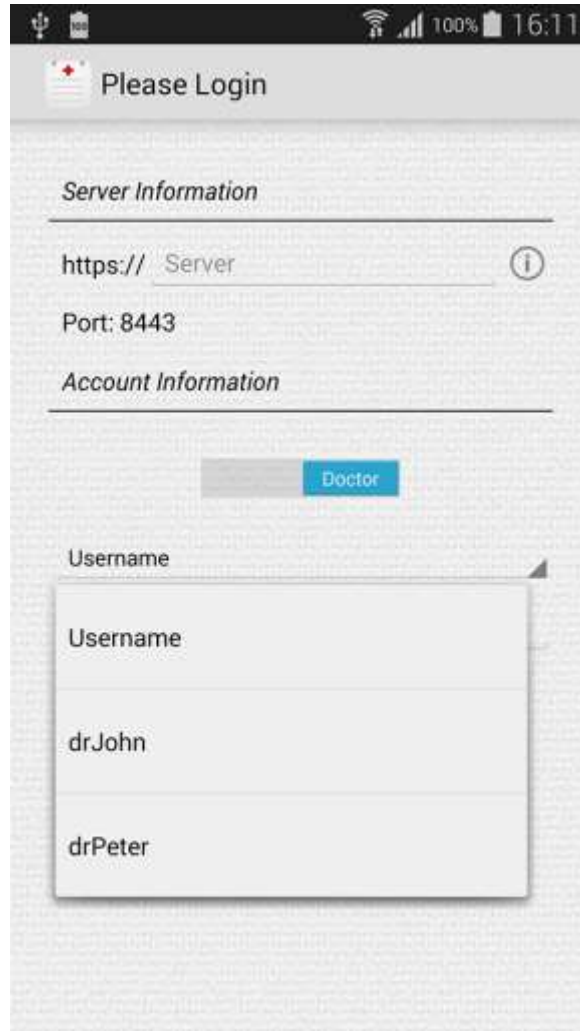
Second I will explain the client side application. Bellow, are some screen shot with the explanation. (First I will analyze a doctor use case)



pic 1

This is the first screen that appears after the splash screen. It prompts the user to log in as patient or as a doctor. First he/she must enter the server IP in which the server is online (as told in the start of this documentation). It uses by default https and the port 8443 that is the port for the https that we have specified in the oauth 2.0 implementation. If the user clicks the little i icon then a toast message

appears and tells the user to type only the IP of the server. e.g. 192.168.1.132 The user name and password fields are automatically filled because we use hardcoded users.



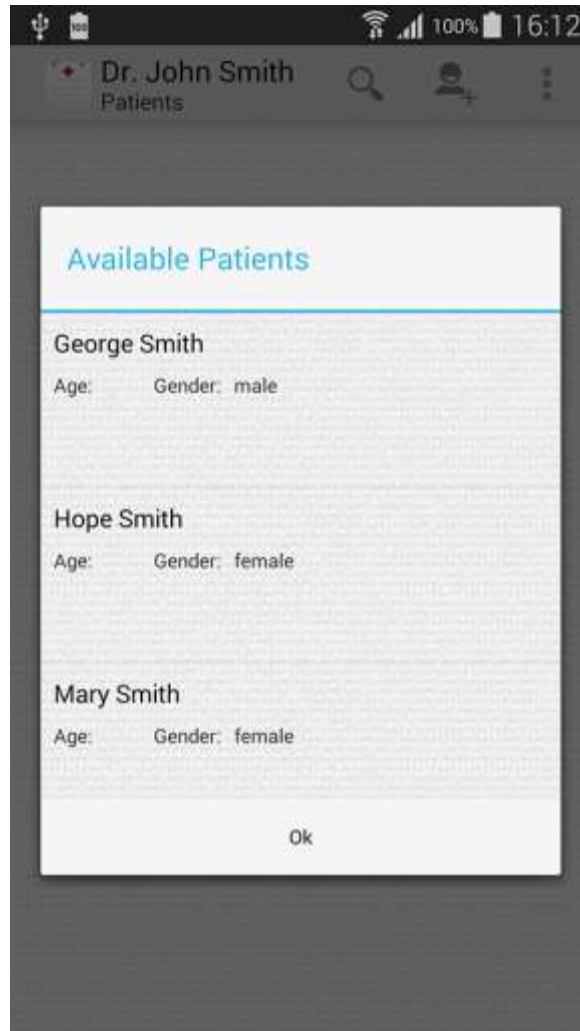
pic 2

This is the drop down menu that appears if the user clicks the spinner button username. Now he can select one of the hardcoded doctors account. When the user selects a doctor automatically the field password below the spinner username gets the appropriate password that is hidden that means that is in the form of "*****".



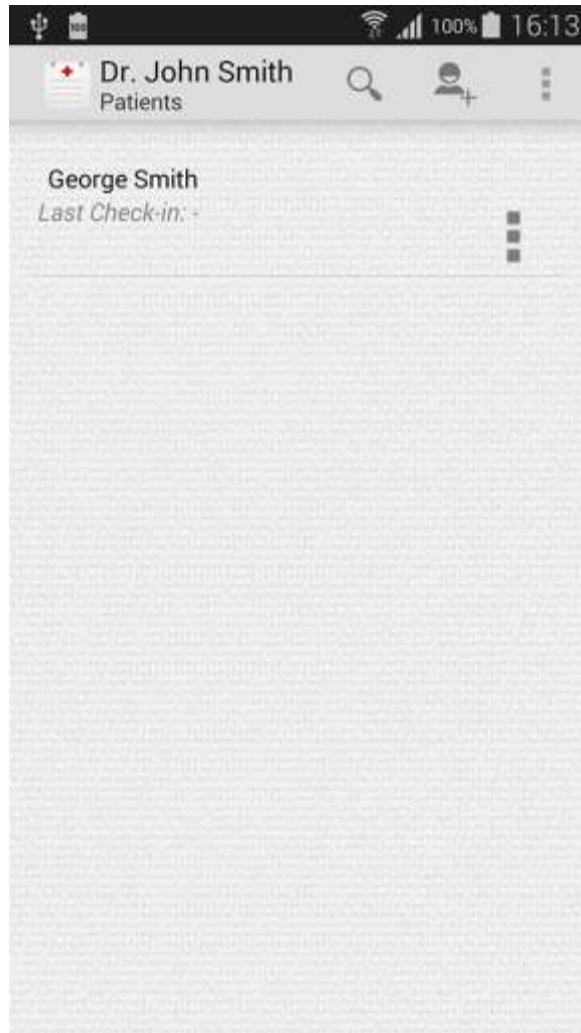
pic 3

This is the main screen of the doctor. It contains an action bar with title "Dr" and the name and surname of the doctor. A subtitle "patients" and some action bar menu icons. The rest of the screen is a list view that will hold all the patients the current doctor has. Now he/she doesn't have any patient. The action bar menu icons are: Search to search for the doctor's patients, add patient, edit profile (you cant see it now because is in the overflow menu) remove patient (you cant see it now because is in the overflow menu) and logout (you cant see it now because is in the overflow menu)

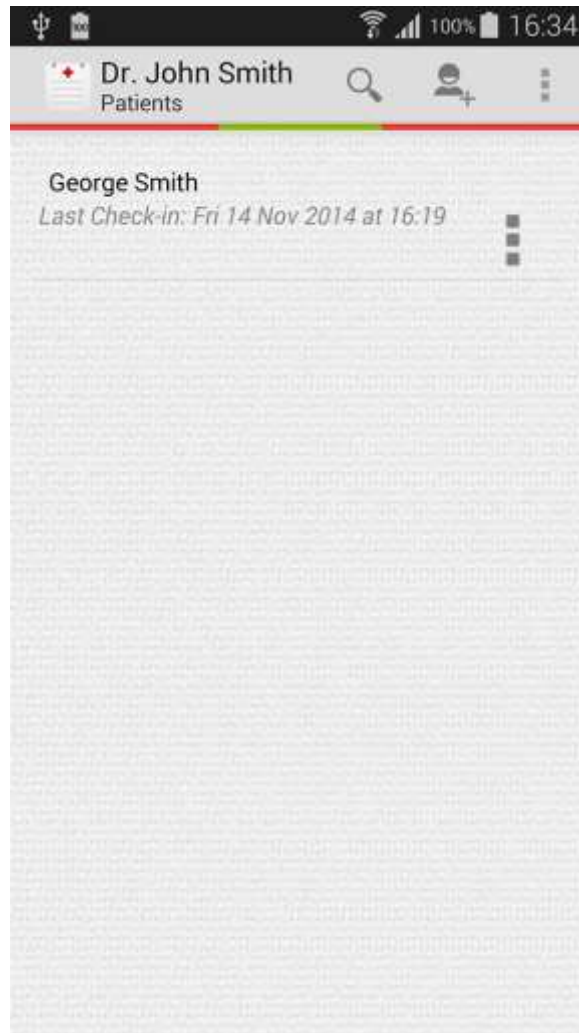


pic 4

If the doctor selects the add patient action from the action bar menu then a dialog will open up with a list with all the patients that are available and by that I mean that doesn't have any doctor yet. The list item here has the Name Surname of the patient below the Age and the Gender. If the patient doesn't have edited his/hers profile the information will be empty.

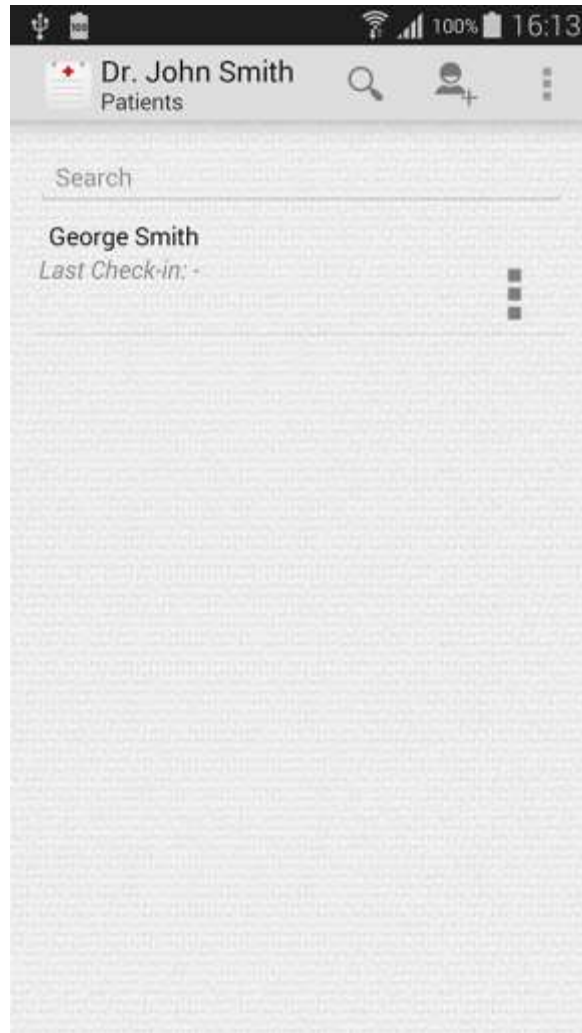


pic 5



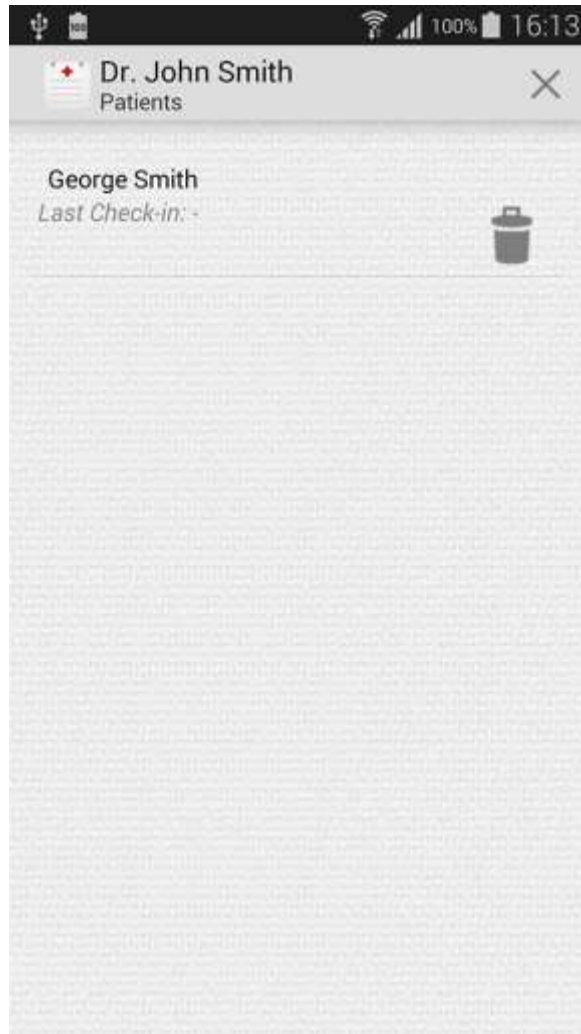
pic 5a

Here is the same screen as before only now the doctor has one patient the George Smith patient. Every item in the list has the following information. Name Surname of patient Last check in and there is the timestamp of the last check in in the form of Tue 14 Jan 2014 at 12:24 and a context menu icon that will open a menu with two actions. Add medication and remove medication. These actions are associated with the specific user. If the user slide down the list and it auto refresh with the above animation. For that it uses the Swipe Refresh Layout from the android developers documentation.



pic 6

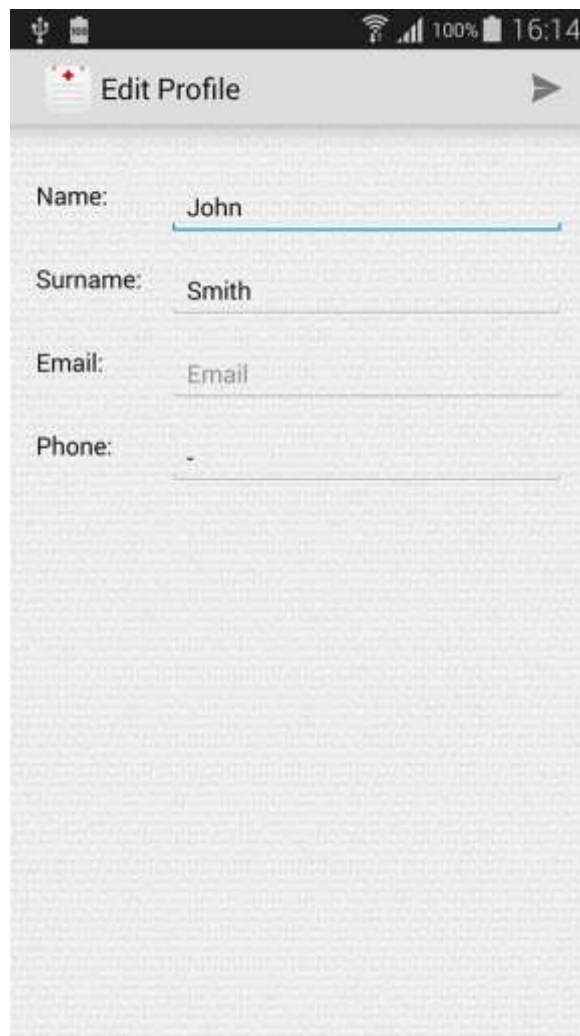
If the doctor selects the search action from the action bar menu an edit text appears. There you can type a patient name or surname and this will search through all the doctor's patients in the database and will populate the list with the patient that their name or surname contains the typed letters.



pic 7

If the doctor selects the remove patient from the action bar menu then in every patient in the list the option menu button changes to the remove button as you can see, also the action bar menu changes and has only one action which is the cancel as you can see, in order to close the remove function. If the doctor cancels the remove function then the option menu in every patient switch back to the overflow menu icon and the action bar menu changes back to the previews menu. If the doctor clicks button remove in a patient then a dialog appears and asks the doctor if he/she is sure to remove this patient.

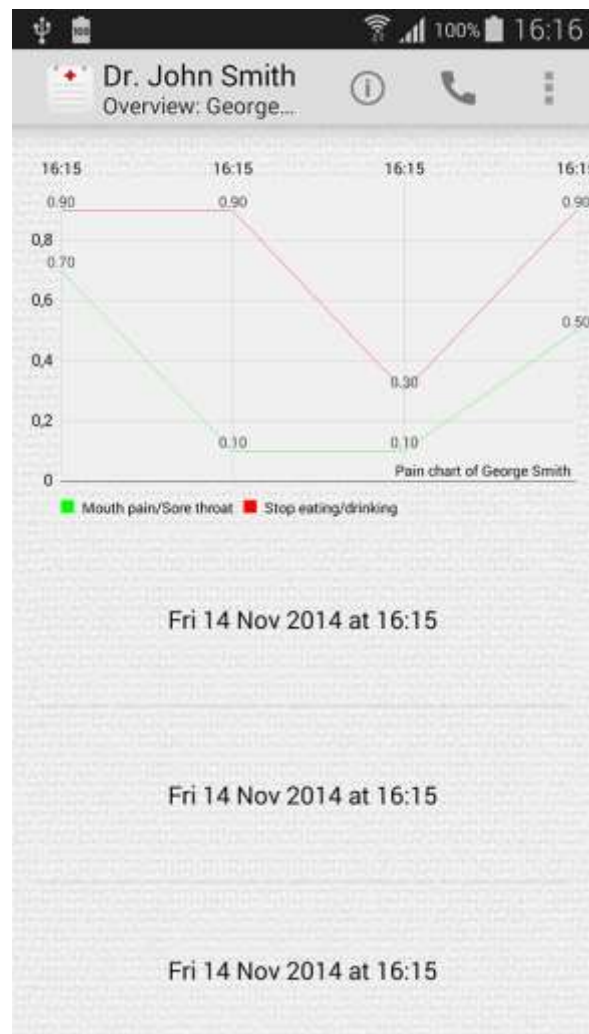
If the doctor adds a medication then a dialog appears and prompts the doctor to enter name and description (both mandatory) of the medication that is going to be given to the patient. When the doctor assigns the medication. A Push Notification arrives (as you will see below in the section with the notifications after the patient screen shots) in the patient's phone and inform him/her that "Your doctor just add the (name of medication) medication". If a doctor removes a medication then a dialog open up with a list view that will hold all the assigned medications to the patient and the doctor can choose which one to remove.

The image is a screenshot of a mobile application's 'Edit Profile' screen. At the top, there is a status bar with icons for USB, battery, and signal, along with the text '100%' and '16:14'. Below the status bar is a header bar with a red cross icon and the text 'Edit Profile', followed by a right-pointing arrow. The main content area contains four labeled input fields: 'Name:' with the value 'John', 'Surname:' with the value 'Smith', 'Email:' with the value 'Email', and 'Phone:' with the value '-'. Each input field has a light blue underline. The background of the screen is a light gray with a subtle grid pattern.

pic 8

If the doctor selects the edit profile action from the action bar menu then a new activity opens as you can see. Here the doctor is able to enter

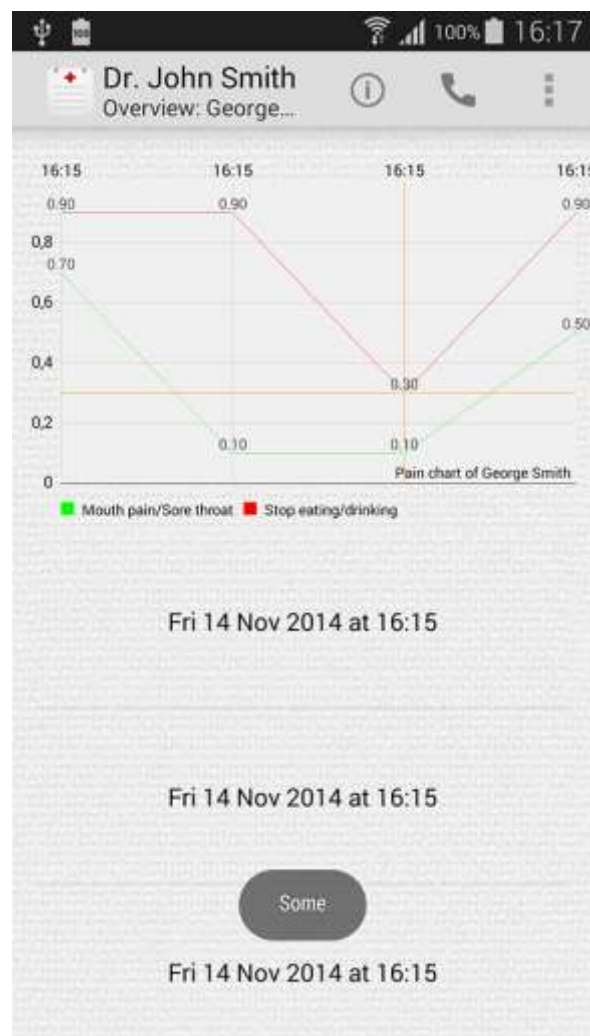
his/hers details like name, surname, email, phone (the email and phone are in case the patient want's to call or send an e-mail to the doctor for some emergency. Before the send action calls the appropriate web service it checks that the email and phone fields are entered correctly.



pic 9

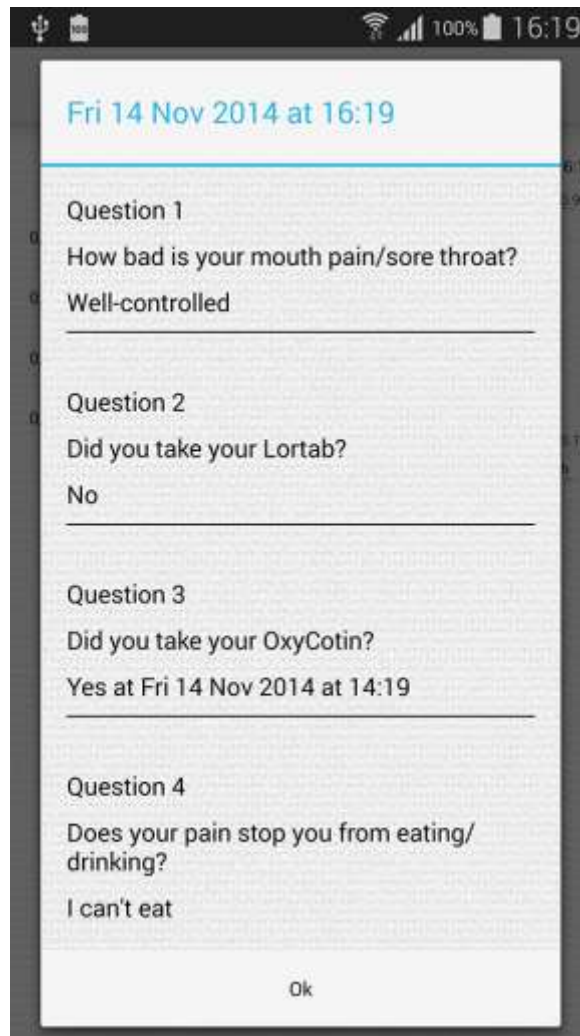
Now if the doctor click's to one of his/hers patient in the list this activity opens up that has information about the patient. This is a chart diagram from an open source library of Phil Jay MPAndroid Chart. Below that is a list with all the check in's the patient had done and you can see their timestamps in the form of e.g. Tue 14 Mar 2014 at 12:34. In the chart it has all the pain information of the patient. The green line represents the mouth pain/sore throat and the red line represents the stop eating/drinking pain. The values of pain are in the scale of 0.0 – 1.0 with the 1.0 is the worst pain. If the doctor clicks on a value a toast message appear with the selected pain. The values are:

0.00 = no	0.50 = moderate
0.10 = well controlled	0.70 = severe
0.30 = some	0.90 = I cant eat



pic 10

Here the doctor had selected the 0.30 value which is some. The data of the patient will be refreshed every X time where X is the time the patient had selected to be reminded to make a check-in + 5 minutes. e.g. if the patient gets informed at 15:00 o'clock the data will be refreshed automatically at 15:05 o'clock. Also when the doctor exits and enters this screen the data are updated.



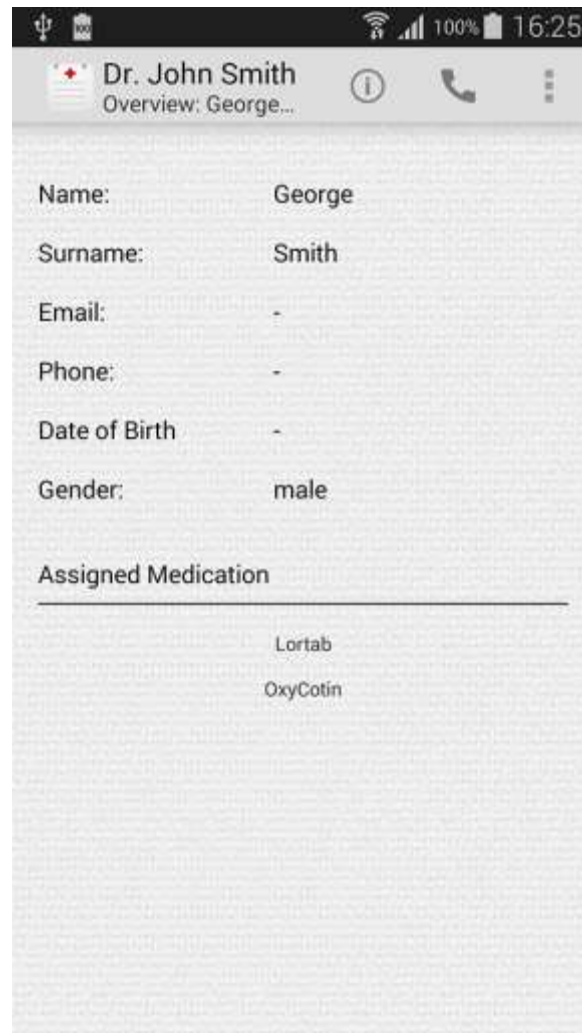
The screenshot shows a mobile application interface with a status bar at the top displaying icons for USB, signal, 100% battery, and the time 16:19. The main content area has a header "Fri 14 Nov 2014 at 16:19" in blue text. Below the header, there are four questions, each followed by an answer:

- Question 1: How bad is your mouth pain/sore throat? Answer: Well-controlled
- Question 2: Did you take your Lortab? Answer: No
- Question 3: Did you take your OxyCotin? Answer: Yes at Fri 14 Nov 2014 at 14:19
- Question 4: Does your pain stop you from eating/drinking? Answer: I can't eat

At the bottom of the form, there is an "Ok" button.

pic 11

When the doctor click's a check-in from the list this dialog pops up that has all the questions and their answers in detail.



pic 12

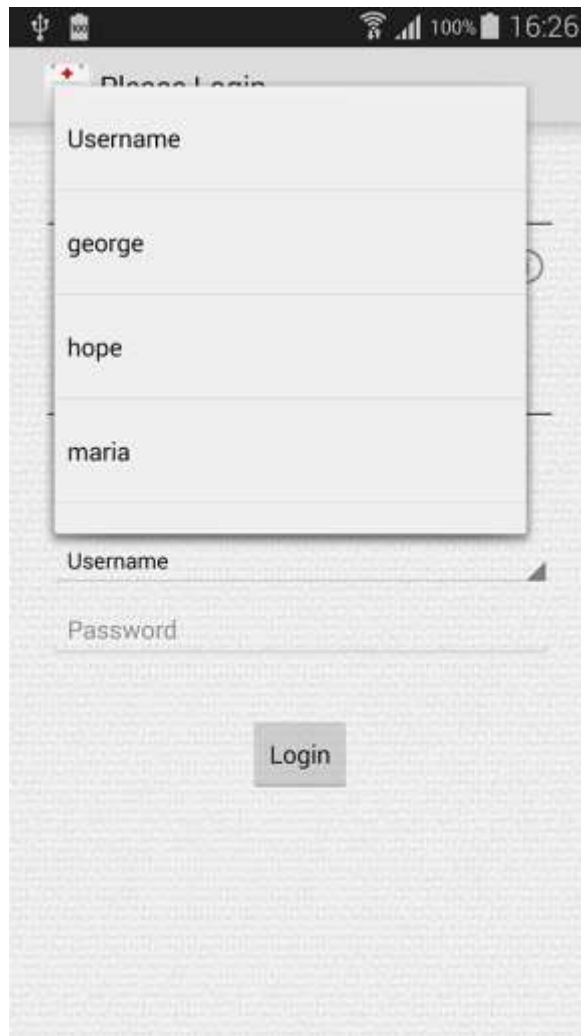
If the doctor selects the “i” icon in the action bar menu the screen flips and shows the patient profile info, with the data he/she had defined, and the assigned medications in list below. The list of medication is dynamically updated when the doctor removes or assigns a medication. The flip happens with an animation created in xml files in the recourses folder. For the animation I used two fragments the previous screen and this one. And I change the fragments using the flip animation.



pic 13

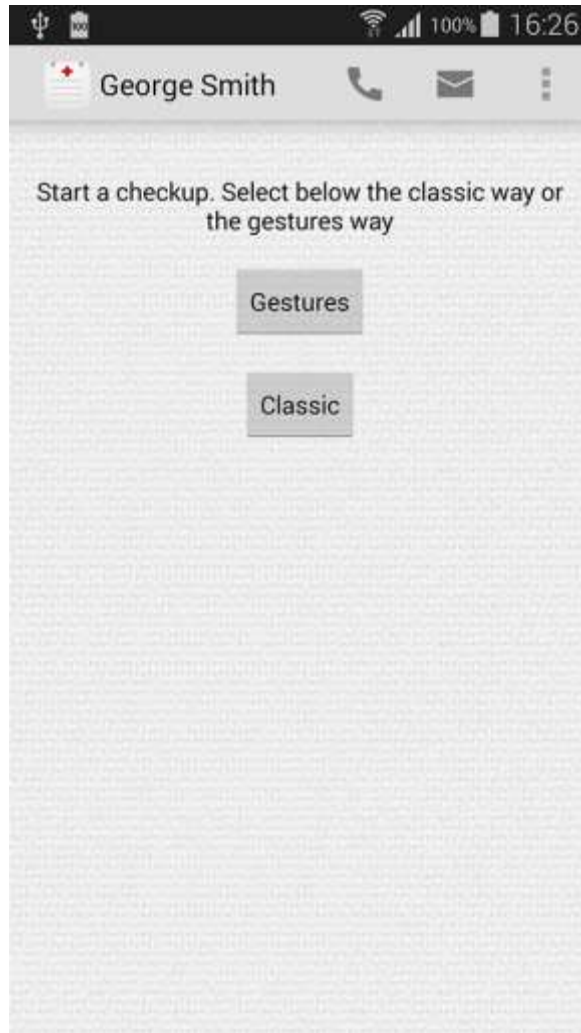
Here the doctor has these actions. I already said about the “i”, the next one call’s the patient and the other are shown in the image, and finally the doctor can logout.

Bellow are some screen shots with the explanation. (I will analyze a patient use case)



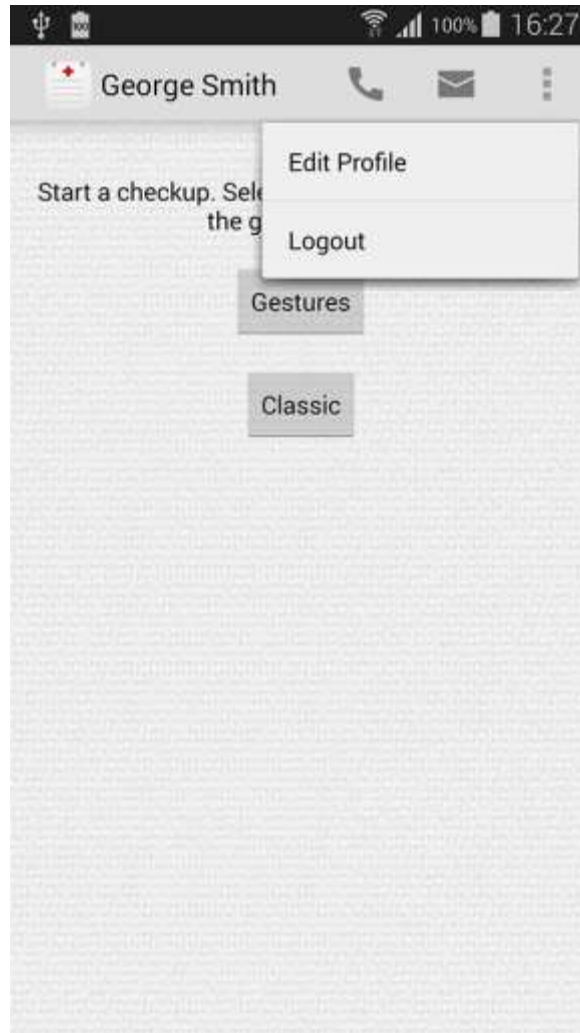
pic 15

This is the drop down menu that appears if the user clicks the spinner button username. Now he can select one of the hardcoded patient accounts. When the user selects a patient automatically the field password below the spinner username gets the appropriate password that is hidden that means that is in the form of *****



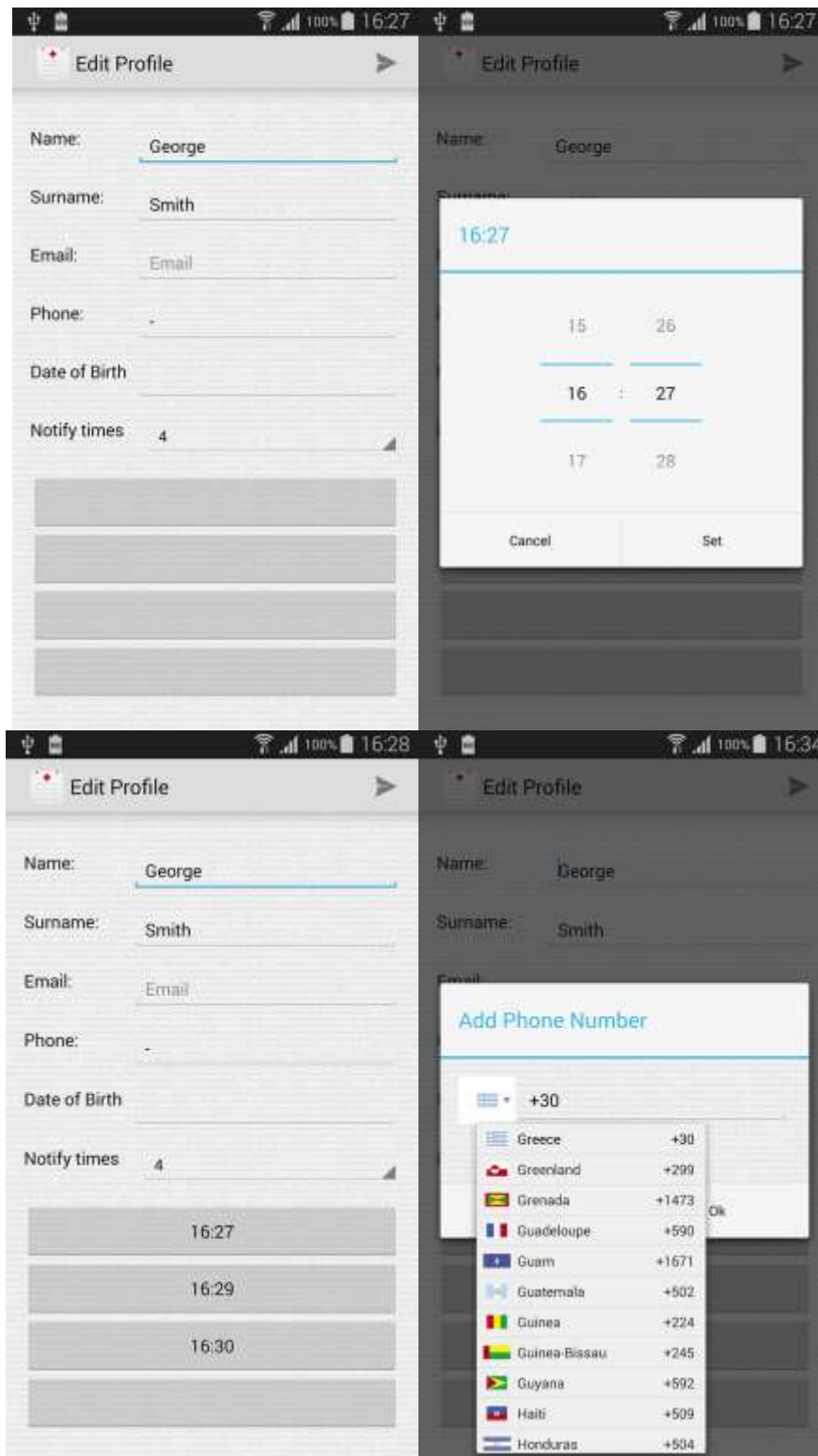
pic 16

Here the patient can make a checkup (check-in) with two ways. The one is the classic way and the other one is the gestures way. I will explain both later.



pic 17

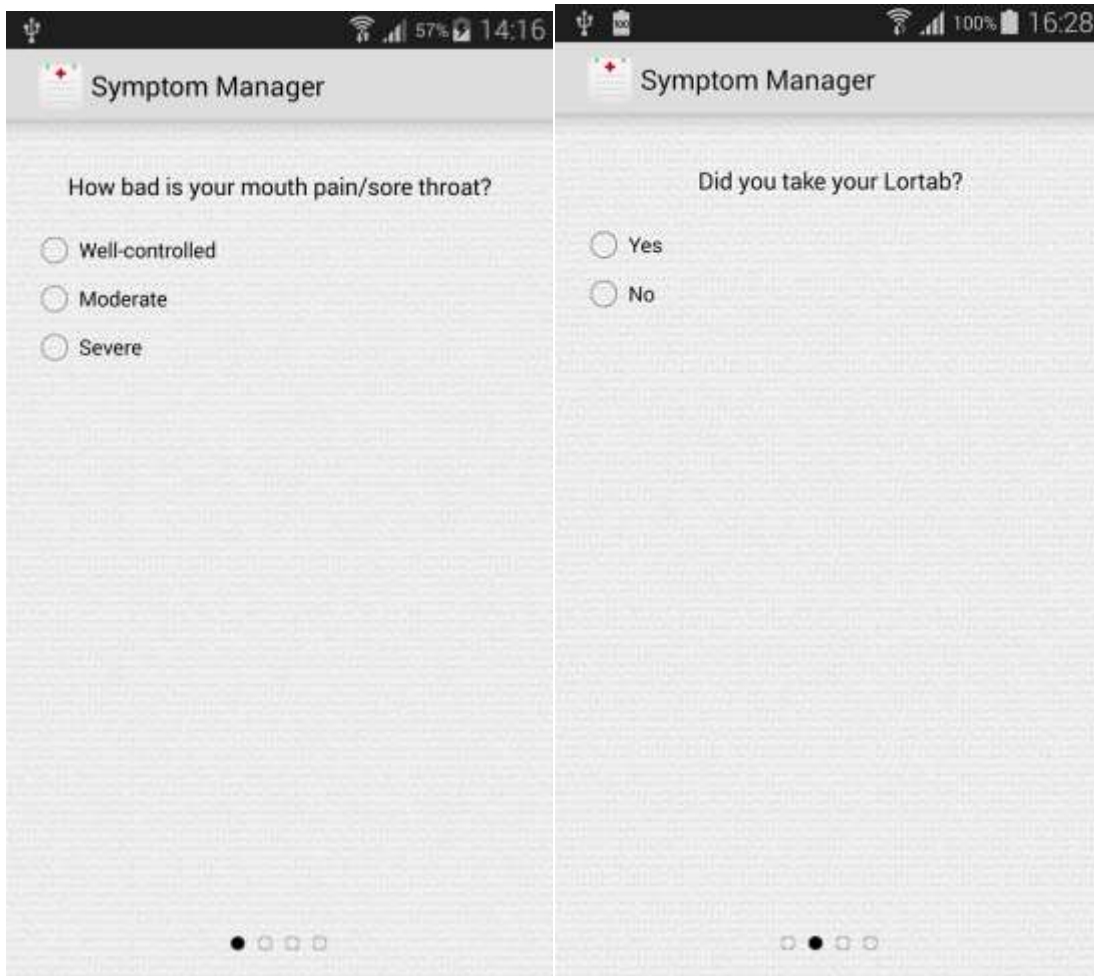
Here are the actions the patient can perform. Call his/hers doctor, send email to his/hers doctor (NOTE if the doctor doesn't have assigned an email or a phone number then a toast appears telling the patient that the doctor haven't assigned an email or phone number. The same happens and in the doctor's use case with the patient email and phone number), edit profile and logout.



pic 18

Here the patient can edit his/hers profile details like name, surname phone, email, date of birth, the amount of times the application will

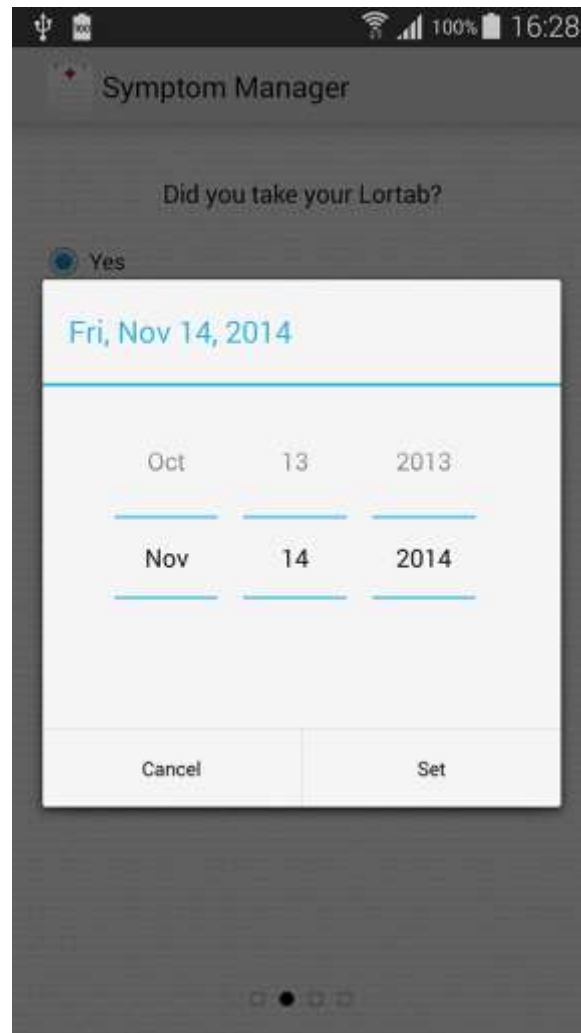
notify him/her to make a checkup (check-in) and of course the times the application will notify him. By default is 4 times a day and cant set less. Before the send action calls the appropriate web service it checks that the email and phone fields are entered correctly. And all the notify times are submitted. If the patient clicks to edit his/hers phone number then a dialog opens up and prompts the user to enter the area code as seen in the images above and then enter the phone number.



pic 19

These are the first and second questions in the checkup (check-in). These views are fragments and are inside a view pager that the user can swipe to go to next or select an answer and programmatically go to next question. The questions with the medications are dynamically added according to the patient's medications that are assigned by his/hers doctor. NOTE if the user (patient) doesn't answer a question then if

he/she tries to send the checkup (check-in) he/she will not be able to do that and a toast message will be shown t answer all the questions.



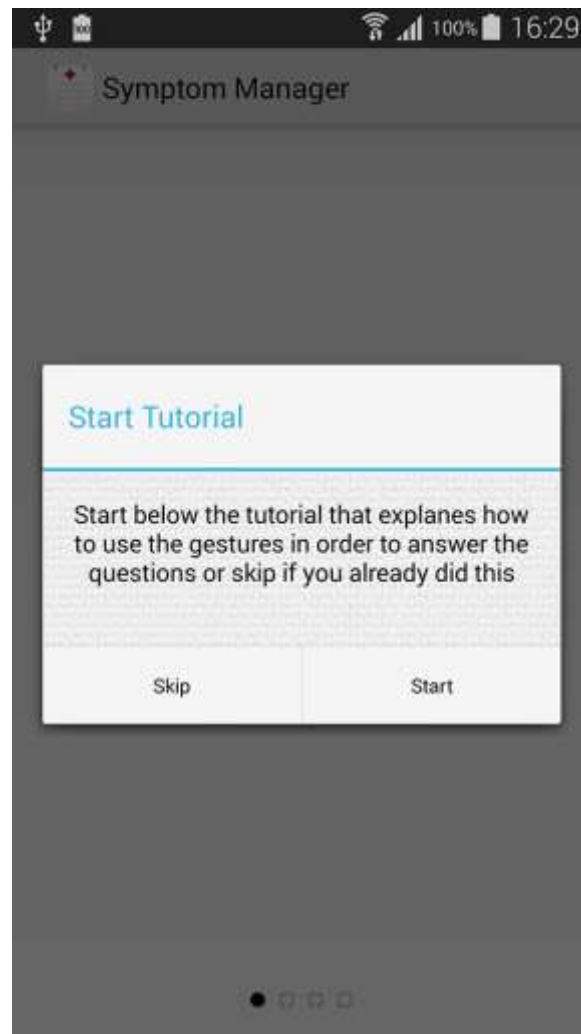
pic 20

If the patient selects yes to a medication then a dialog appears and ask the patient to enter the time he/she took the medication.

The image displays two side-by-side screenshots of a mobile application titled "Symptom Manager". The top status bar of both screens shows a signal strength icon, a battery icon at 100%, and the time 16:29. The app's header bar is grey with the title "Symptom Manager" and a red cross icon on the left, and a grey arrow icon on the right. The main content area is white. The left screen displays the question "Did you take your OxyCotin?" with two radio button options: "Yes" and "No". The right screen displays the question "Does your pain stop you from eating/drinking?" with three radio button options: "No", "Some", and "I can't eat". At the bottom of each screen, there is a progress indicator consisting of four circles; the second circle from the left is filled black in both screens, indicating the current question.

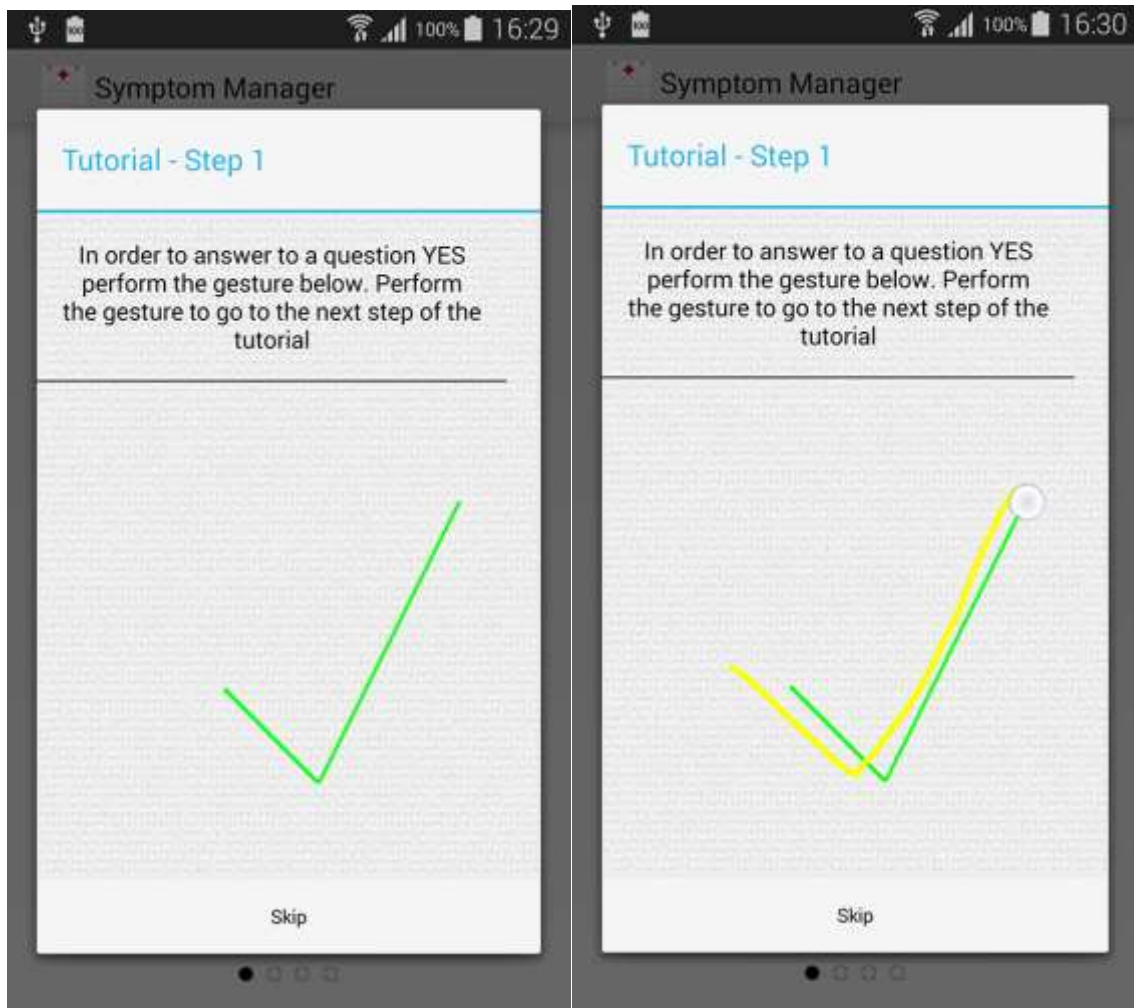
pic 21

And these are the last questions in my example a patient with two medications. Lortab and OxyCotin. As you can see the action bar menu button send appears only in the last question (fragment).



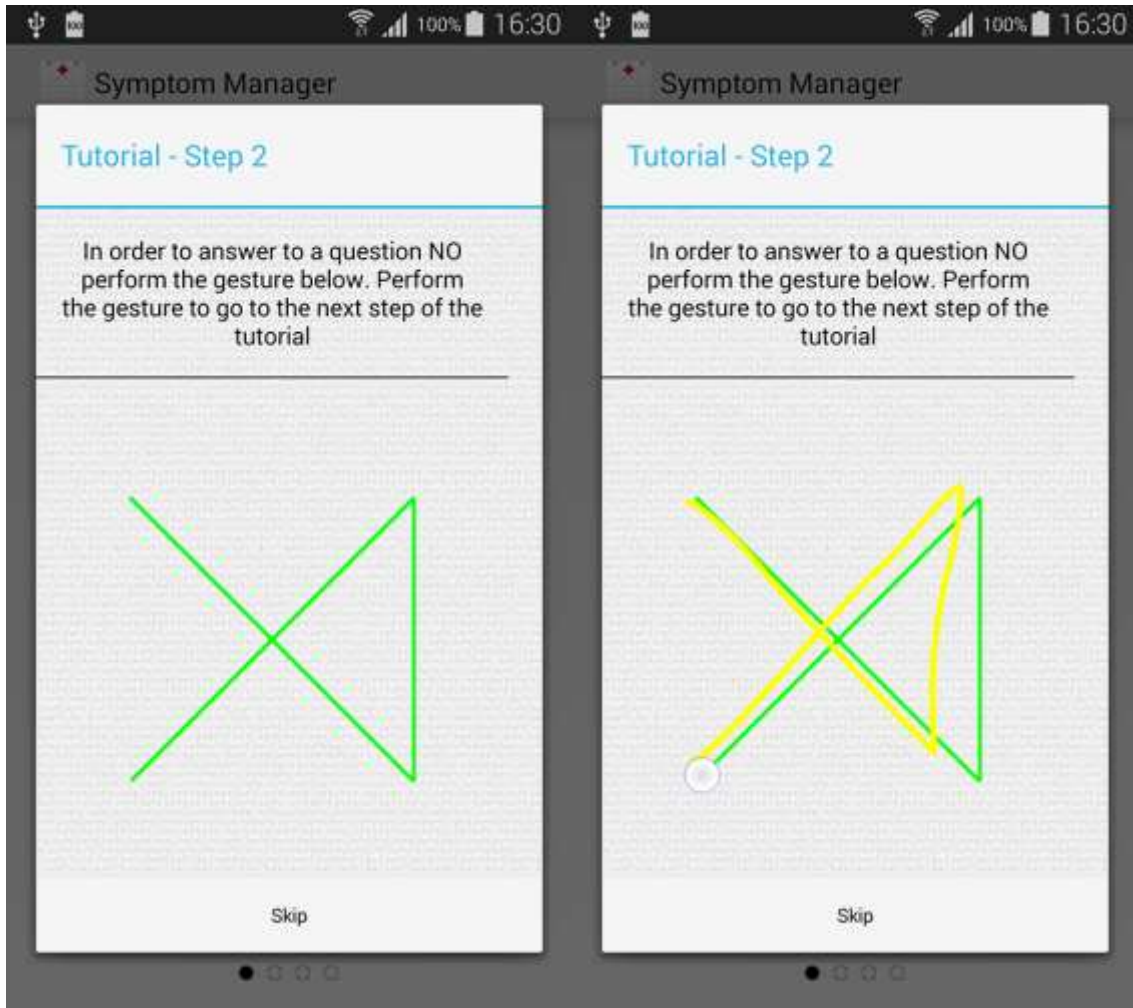
pic 22

If the patient selects the gestures to make a checkup (check-in) then a dialog appears that inform the patient to take the tutorial in order to know how to do the checkup (check-in) with gestures. If the patient knows he/she select skip otherwise select start.



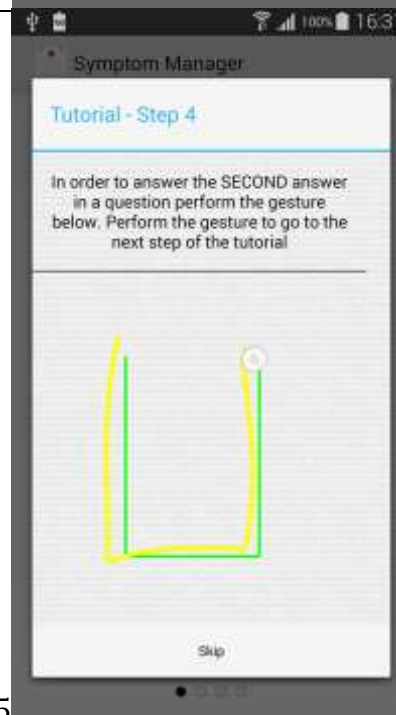
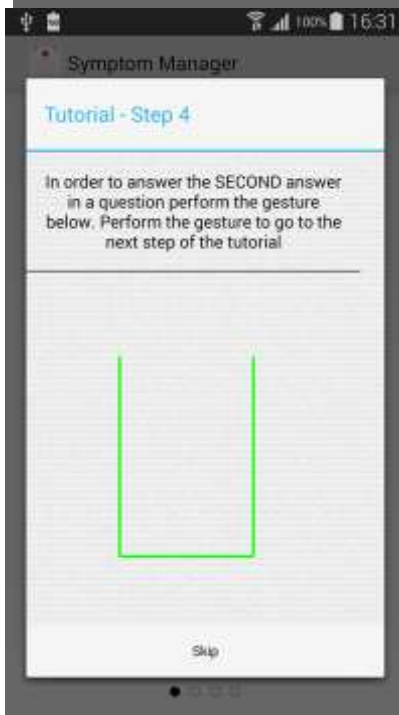
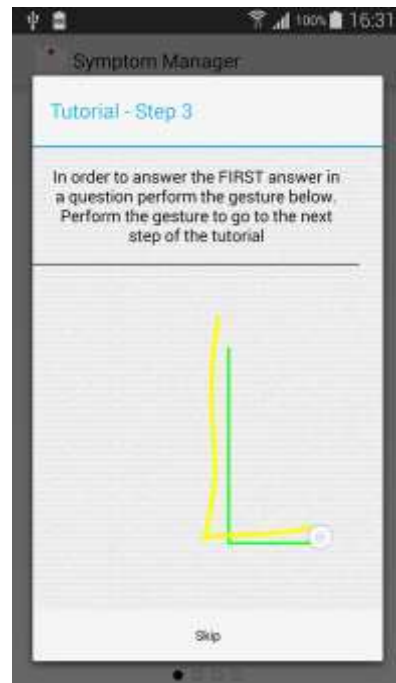
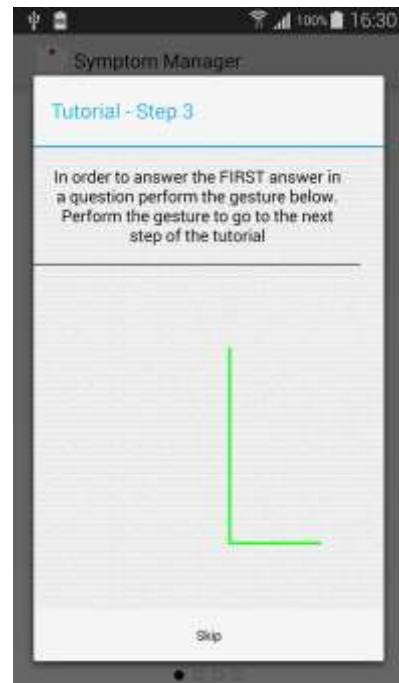
pic 23

The patient has to follow the instructions and perform the correct gestures. If he/she wants to answer yes to a question then he/she must make a “tic” gesture. In order to move to the next step of the tutorial he/she must make 2 correct gestures in each tutorial step. Any time he/she wants he/she can skip the tutorial by selecting skip



pic 24

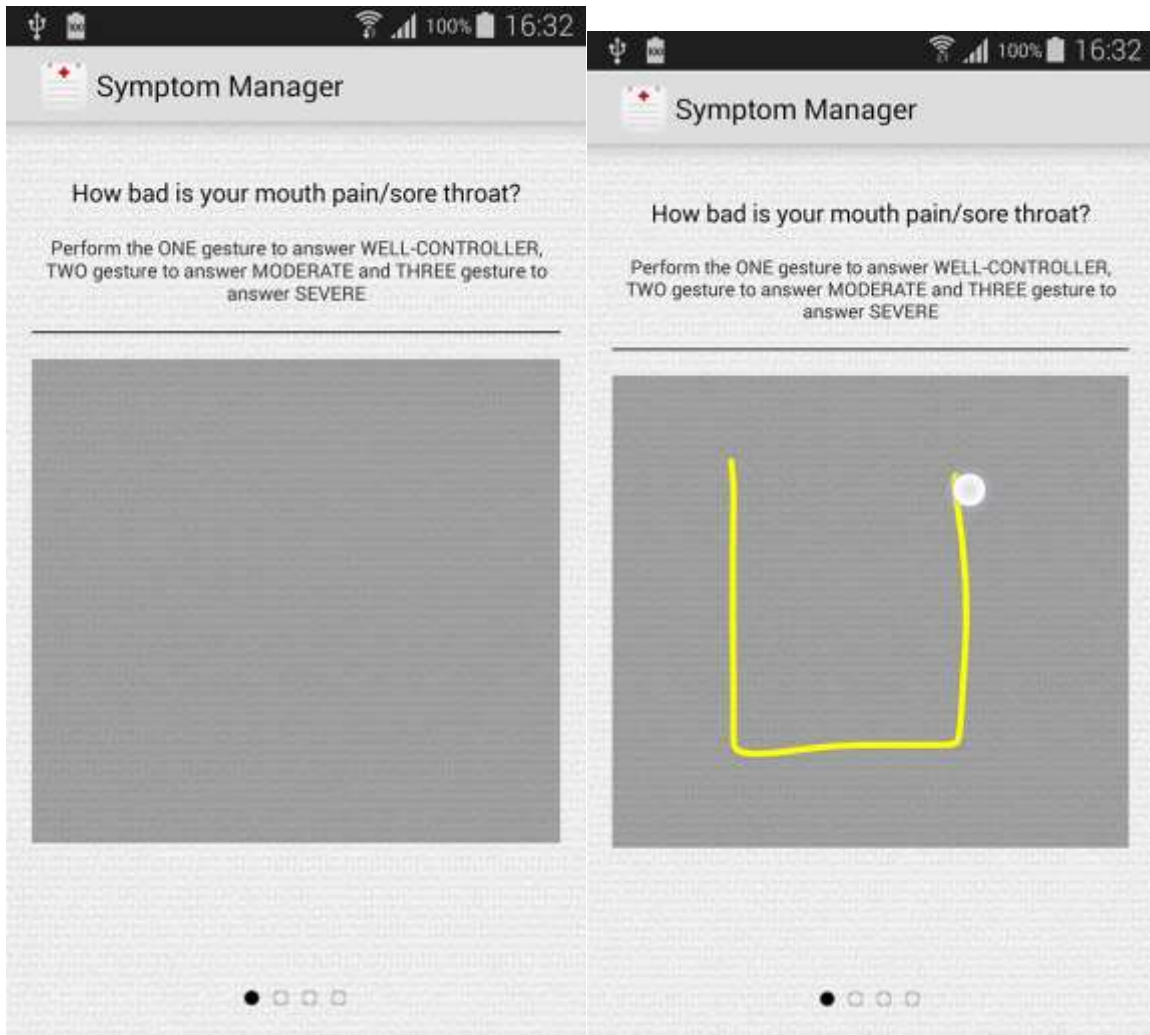
The patient has to follow the instructions and perform the correct gestures. If he/she wants to answer no to a question then he/she must make the gesture. In order to move to the next step of the tutorial he/she must make 2 correct gestures in each tutorial step. Any time he/she wants he/she can skip the tutorial by selecting skip



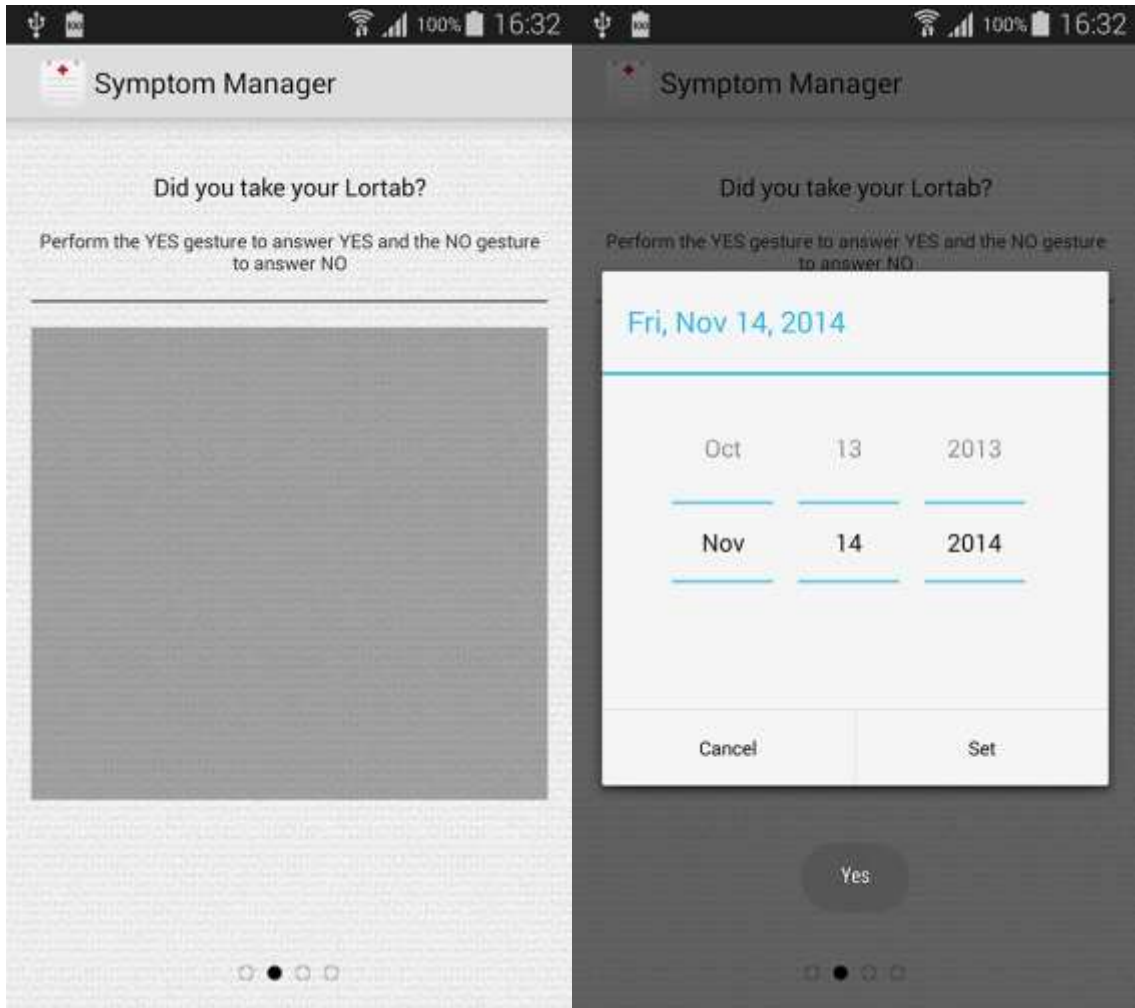


pic 26

After the step 5 the patient finishes the tutorial and sees the first question. The concept is the same and the question are the same, as in the classic way only the patient has to make a gesture in order to answer a question.

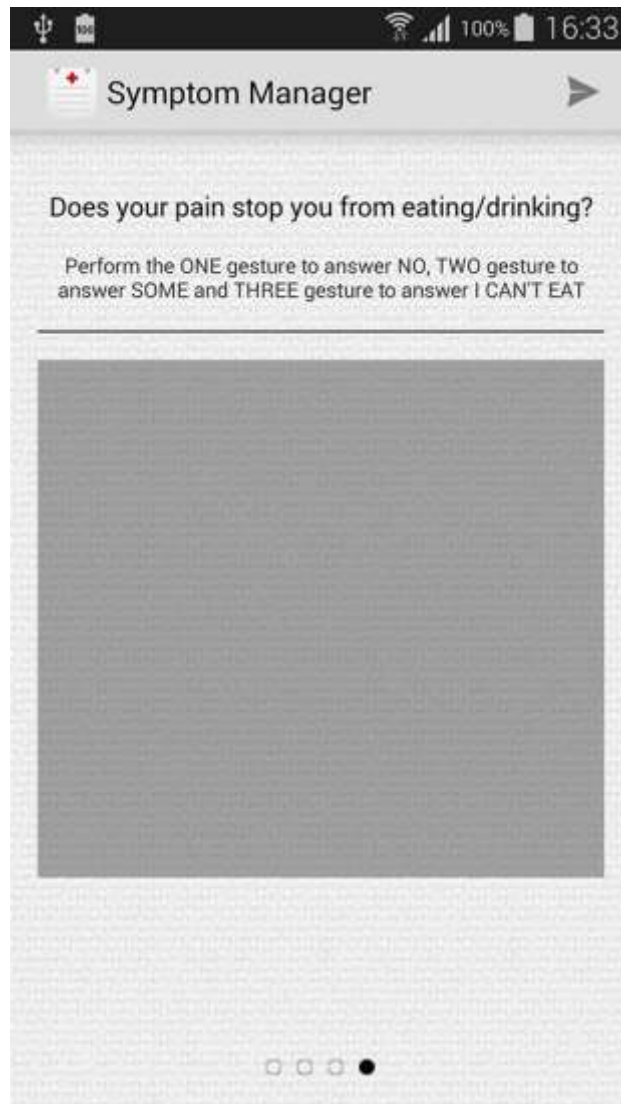


pic 27



pic 28

If the patient answers yes again has to enter the time he/she took the medication



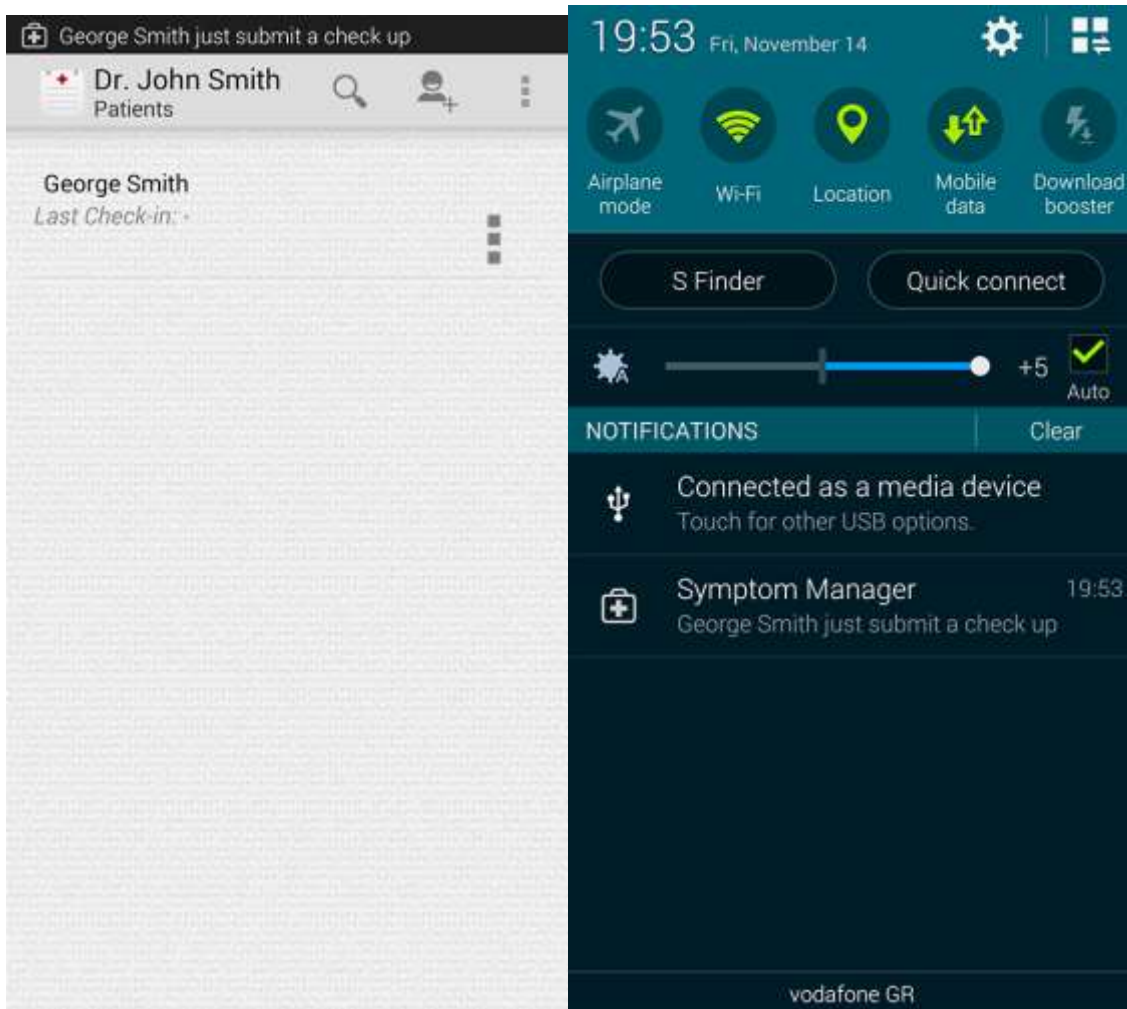
pic 29

And the last question, notice the action bar menu button again appears only in the last question.

Below are the screenshots with all the notification the user (doctor or patient) receive on their phones.

The push notifications are using the GCM servers of Google. The regular notifications are triggered by broadcast receivers.

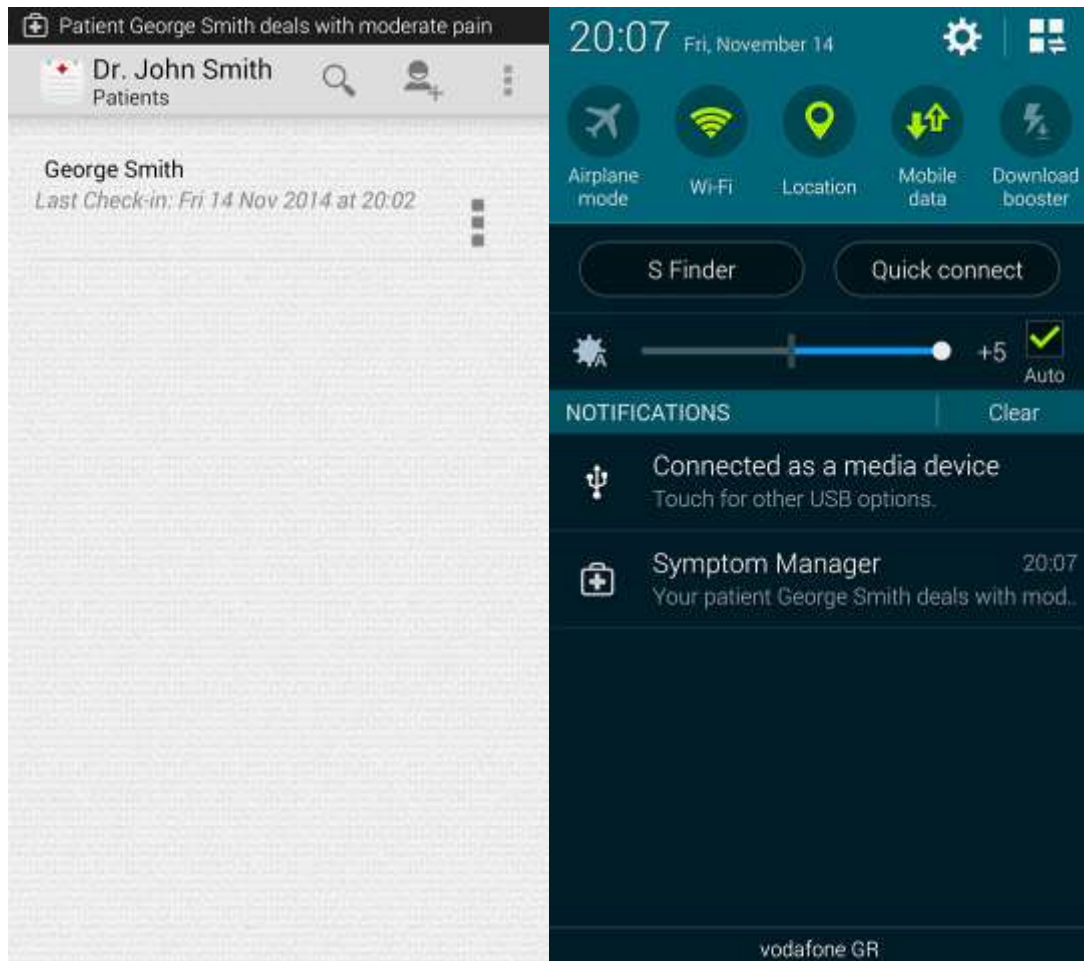
Notifications for Doctors



pic 14a

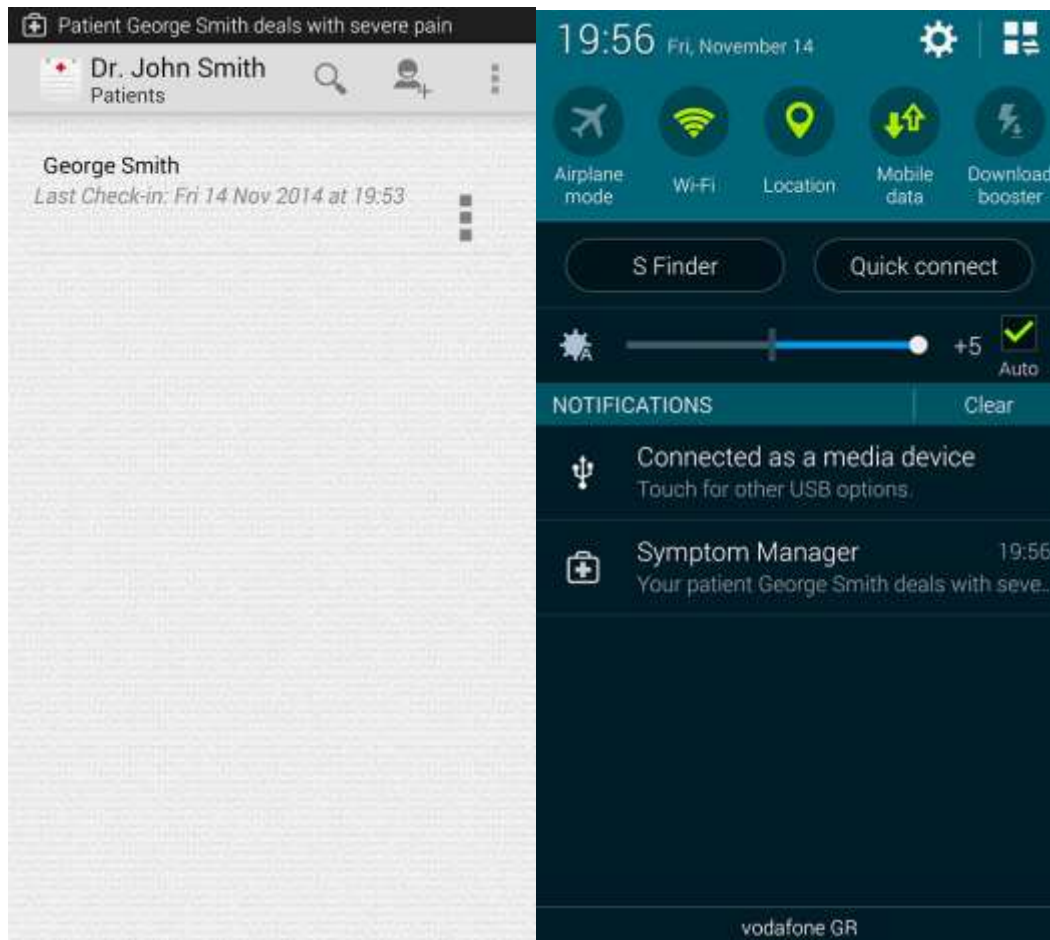
If a patient makes a checkup (check-in) then his/hers doctor gets informed with a push notification in the above form. Name surname of patient “just performed a check up”. Left image shows the notification in

the system bar and the right image shows the notification after we opened the drawer from the system menu.



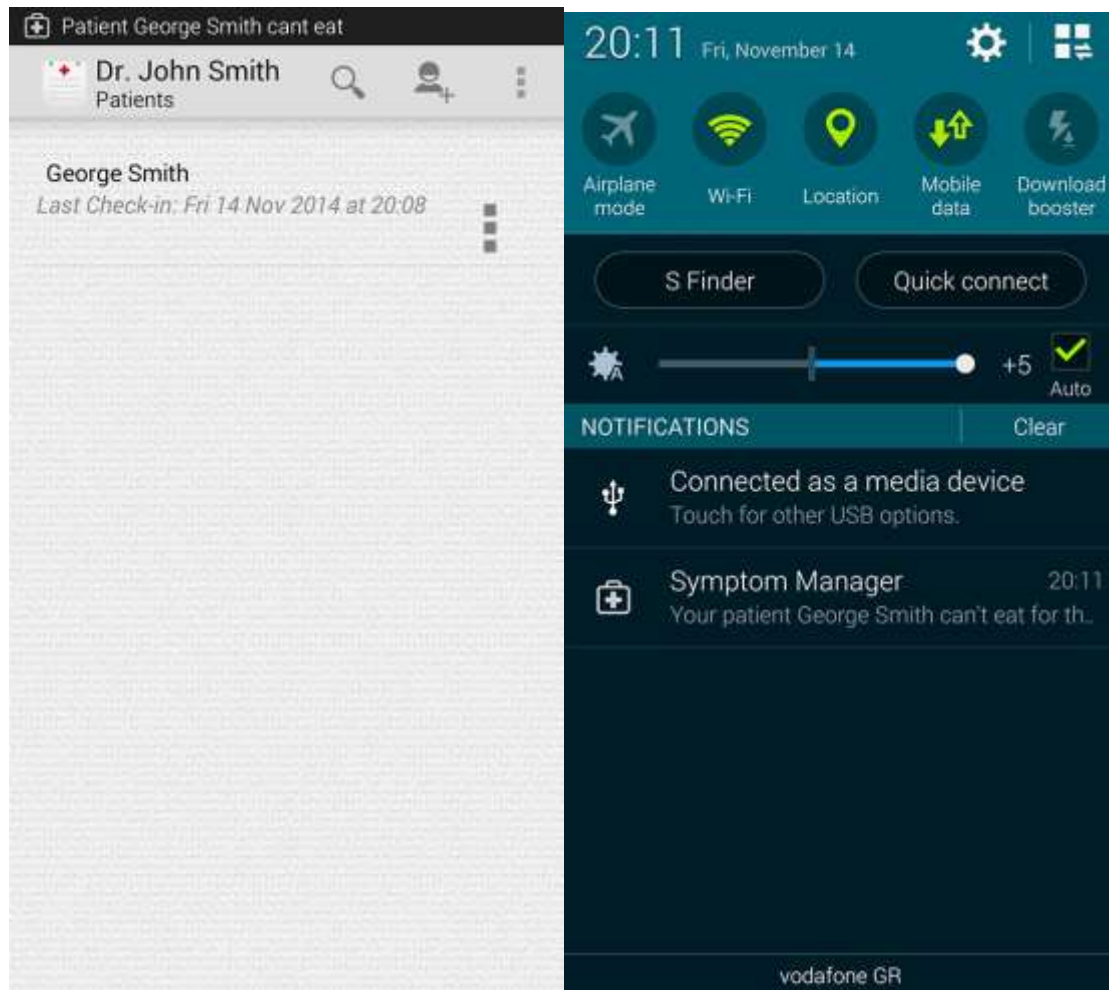
pic 14b

If a patient deals with moderate pain for 16 hours then the patient's doctor receives a push notification in the above form. "Your patient" Name Surname "deals with moderate pain for the past 16 hours". Left image shows the notification in the system bar and the right image shows the notification after we opened the drawer from the system menu.



pic 14c

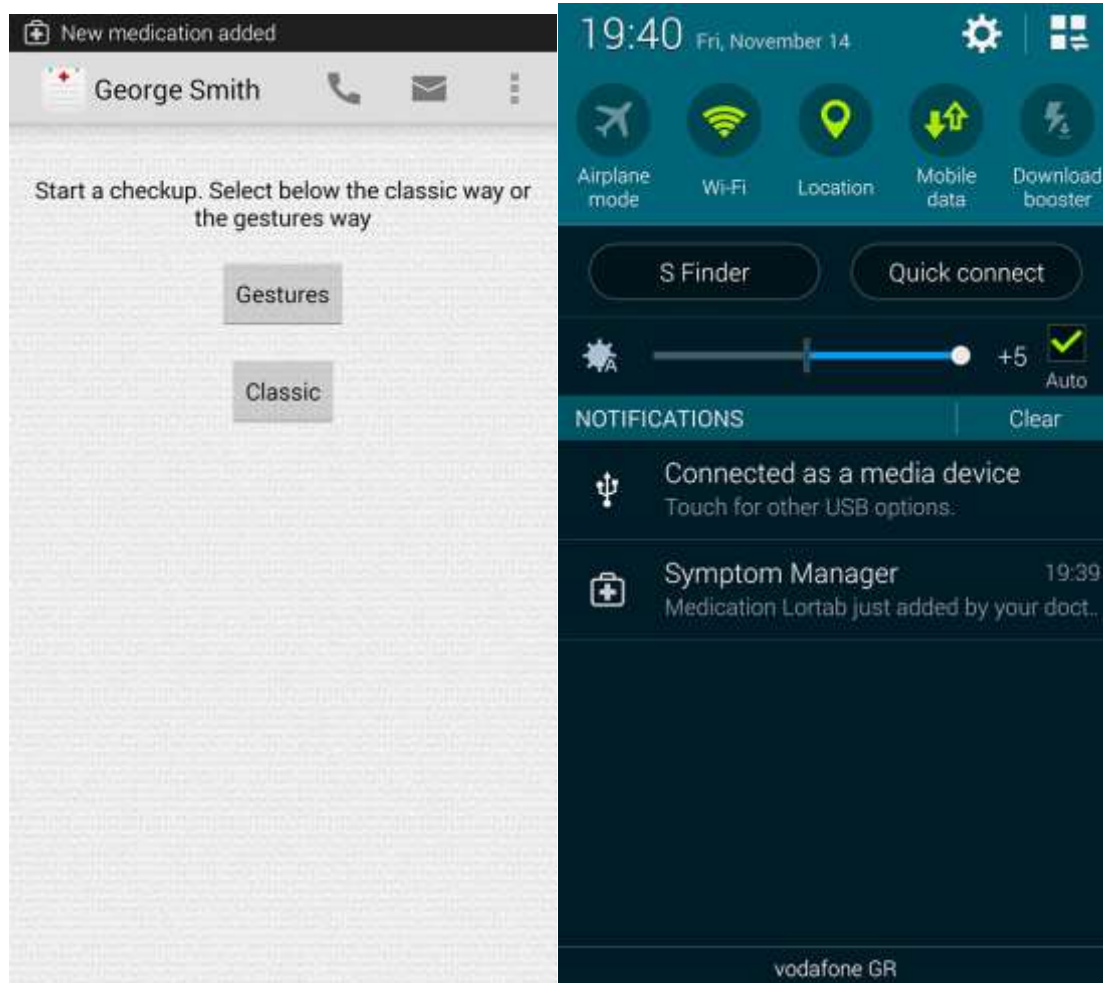
If a patient deals with severe pain for 12 hours then the patient's doctor receives a push notification in the above form. "Your patient" Name Surname "deals with severe pain for the past 12 hours". Left image shows the notification in the system bar and the right image shows the notification after we opened the drawer from the system menu.



pic 14d

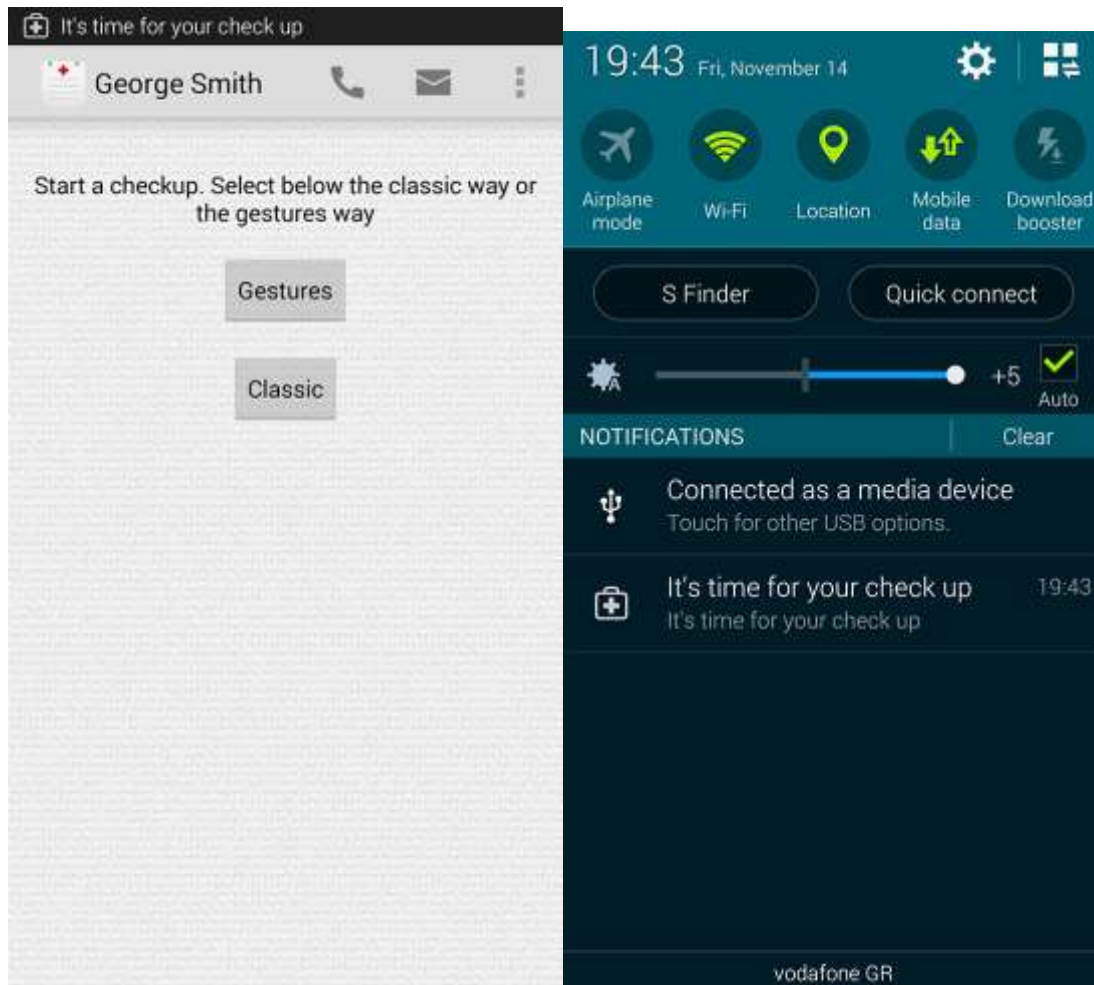
If a patient can't eat for 12 hours then the patient's doctor receives a push notification in the above form. "Your patient" Name Surname "can't eat for the past 12 hours". Left image shows the notification in the system bar and the right image shows the notification after we opened the drawer from the system menu.

Notifications for Patients



pic 14e

If a patient's doctor adds a medication to a patient then the patient receives a push notification in the above form. "Medication" Name or medication "just added by your doctor". Left image shows the notification in the system bar and the right image shows the notification after we opened the drawer from the system menu.



pic 14f

If the time the patient had select to be notified by the application in order to have a checkup (check-in) comes, then an alarm manager is triggered and creates an event, then a broadcast receiver that listens to that event creates a notification to the patient's device and informs the patient to have a checkup (check-in) in the above form. "It's time for your check up". Left image shows the notification in the system bar and the right image shows the notification after we opened the drawer from the system menu.

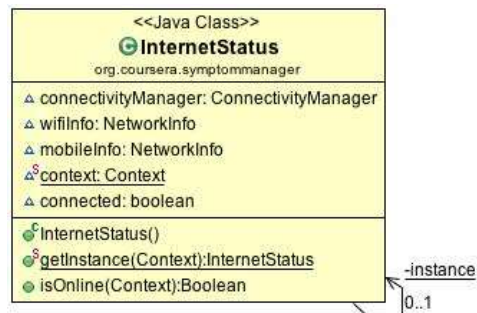
Below are a UML diagram for each class of the application:

All the Services (GCMIntentService ScreenReceiverService)

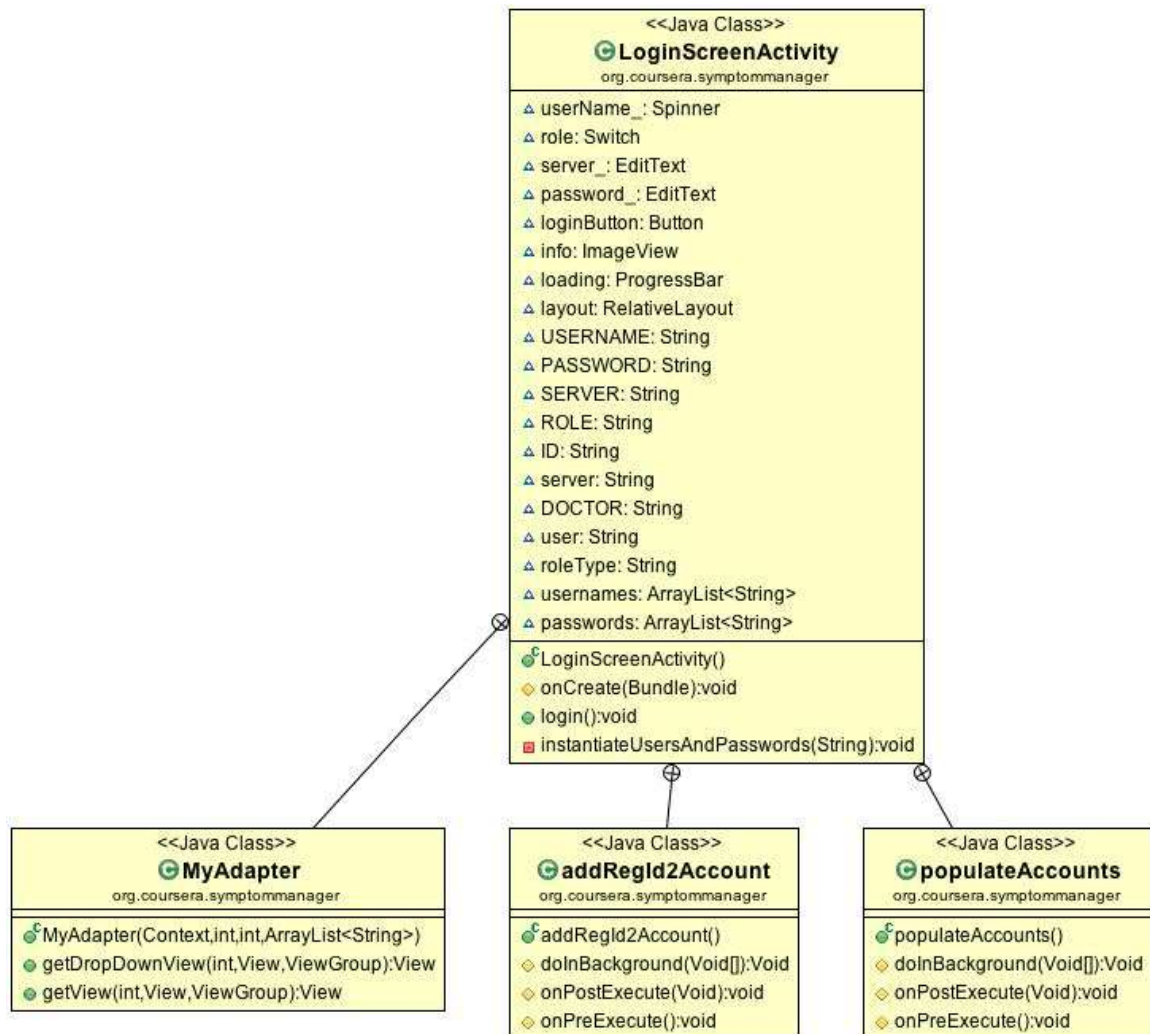
Constants



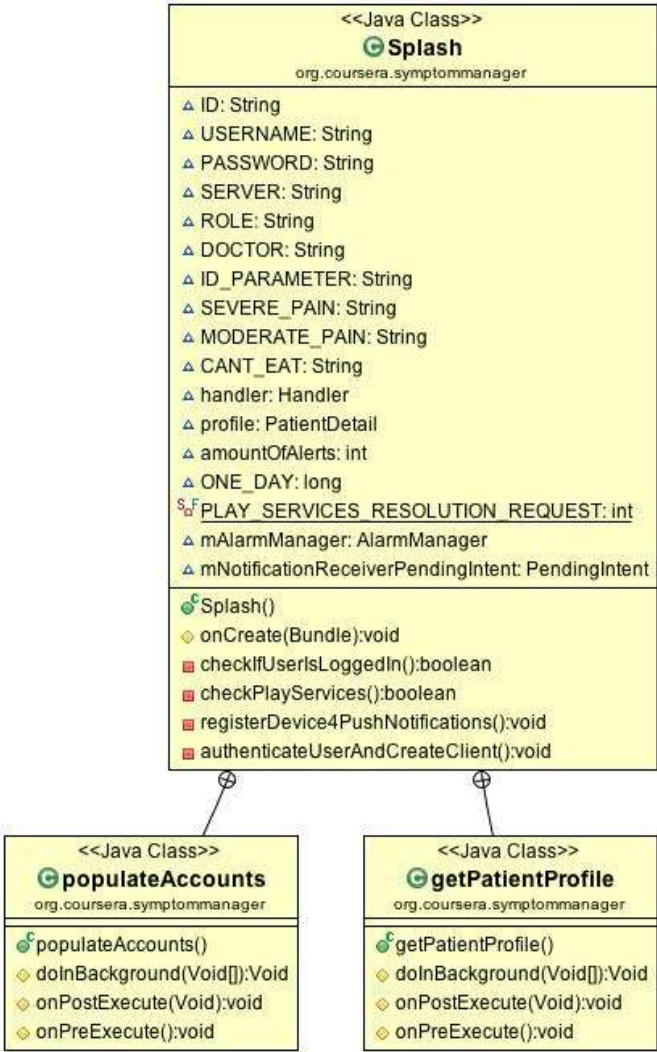
InternetStatus



LoginScreenActivity



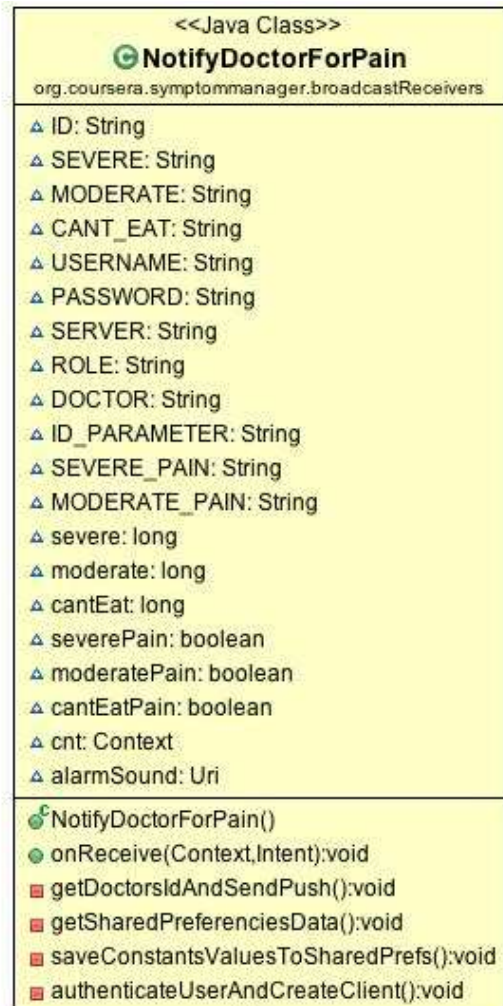
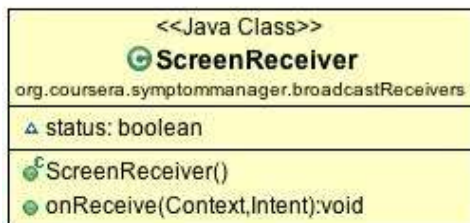
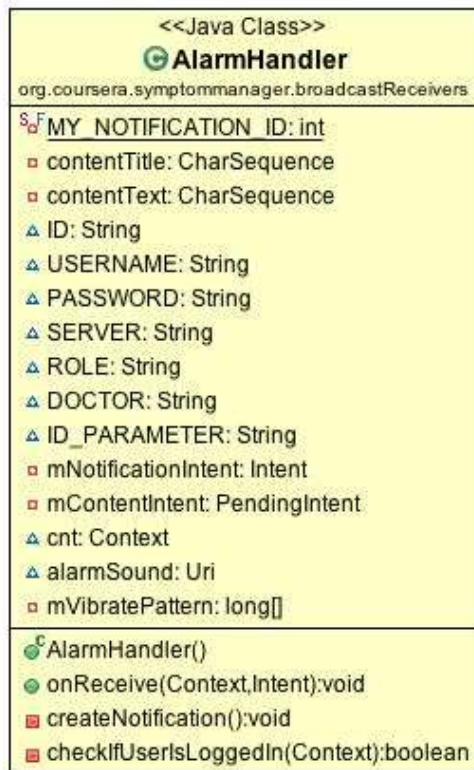
Splash



SymptomSvcApi

<<Java Interface>>	
 SymptomSvcApi <small>org.columeri.symptommanager</small>	
<ul style="list-style-type: none"> TOKEN_PATH: String PATIENT_SVC_PATH: String DOCTOR_SVC_PATH: String ADD_USER: String ADD_REGID: String ADD_REGID_DOCTOR: String ADD_REGID_PATIENT: String REGID_PARAMETER: String TEST: String POPULATE_ACCOUNT_TABLE: String POPULATE_PATIENTS_TABLE: String POPULATE_DOCTOR_TABLE: String GET_PATIENT_PROFILE: String USERNAME: String CREATE_PATIENT_PROFILE: String UPDATE_PATIENT_PROFILE: String FIND_PATIENT_BY_NAME: String FIND_PATIENT_BY_SURNAME: String GET_MEDICATION: String ID: String ID2: String SEND_CHECKIN: String SEND_PUSH_2_DOCTOR: String TITLE: String MESSAGE: String GET_DOCTOR_PROFILE: String CREATE_DOCTOR_PROFILE: String UPDATE_DOCTOR_PROFILE: String GET_PATIENT_LIST: String GET_AVAILABLE_PATIENTS: String GET_PATIENTS_CHECKINS: String GET_PATIENT_PROFILE_DOCTOR: String GET_DOCTOR_PROFILE_PATIENT: String ADD_MEDICATION: String REMOVE_MEDICATION: String SEND_PUSH_2_PATIENT: String UPDATE_PATIENT_PROFILE: String REMOVE_PATIENT: String 	<ul style="list-style-type: none"> addUser(Account):Account populateUsers(ArrayList<Account>):Void populatePatientsProfiles(ArrayList<PatientDetail>):Void populateDoctorsProfiles(ArrayList<DoctorDetail>):Void addRegIdTo(Account,String):Account getTest():Void getProfilePatient(String):PatientDetail createProfilePatient(PatientDetail):PatientDetail updateProfilePatient(PatientDetail):PatientDetail getMedication(long):ArrayList<Medication> sendCheckin(long,String,long,long):Void sendPushDoctor(long,String,String):Void getProfileDoctor(String):DoctorDetail createProfileDoctor(DoctorDetail):PatientDetail updateProfileDoctor(DoctorDetail):DoctorDetail getPatientList(long):ArrayList<PatientDetail> getAvailablePatients():ArrayList<PatientDetail> getCheckins(long):ArrayList<CheckIn> getPatientProfile(long):PatientDetail getDoctorProfile(long):DoctorDetail addMedication(long,String,String,long):Void removeMedication(long):Void sendPushPatient(long,String,String):Void addPatient(long,long):Void removePatient(long):Void addRegIdToDoctor(long,String):DoctorDetail addRegIdToPatient(long,String):PatientDetail findByNameContaining(String):ArrayList<PatientDetail> findBySurnameContaining(String):ArrayList<PatientDetail>

All the broadcast Receivers (AlarmHandler, NotifyDoctorForPain, ScreenReceiver)



All the Java Objects that are used (DoctorDetail, PatientDetail, Medication, Account, CheckIn, Question)

<<Java Class>> 👤 DoctorDetail org.coursera.symptommanager.objects	
username: String name: String surname: String email: String phone: String regid: String id: long	
DoctorDetail() DoctorDetail(String,String,String,String,String,String,Long) getUsername():String setUsername(String):void getName():String setName(String):void getSurname():String setSurname(String):void getRegid():String setRegid(String):void getEmail():String setEmail(String):void getId():long setId(long):void getPhone():String setPhone(String):void	

<<Java Class>> 💊 Medication org.coursera.symptommanager.objects	
id: long name: String description: String idPatient: long	
Medication() Medication(long,String,String,Long) getName():String setName(String):void getDescription():String setDescription(String):void getIdPatient():long setIdPatient(long):void getId():long setId(long):void	

<<Java Class>> 👤 Account org.coursera.symptommanager.objects	
id: long username: String password: String regid: String role: String	
Account() Account(long,String,String,String,String) getUsername():String setUsername(String):void getPassword():String setPassword(String):void getId():long setId(long):void getRegid():String setRegid(String):void getRole():String setRole(String):void	

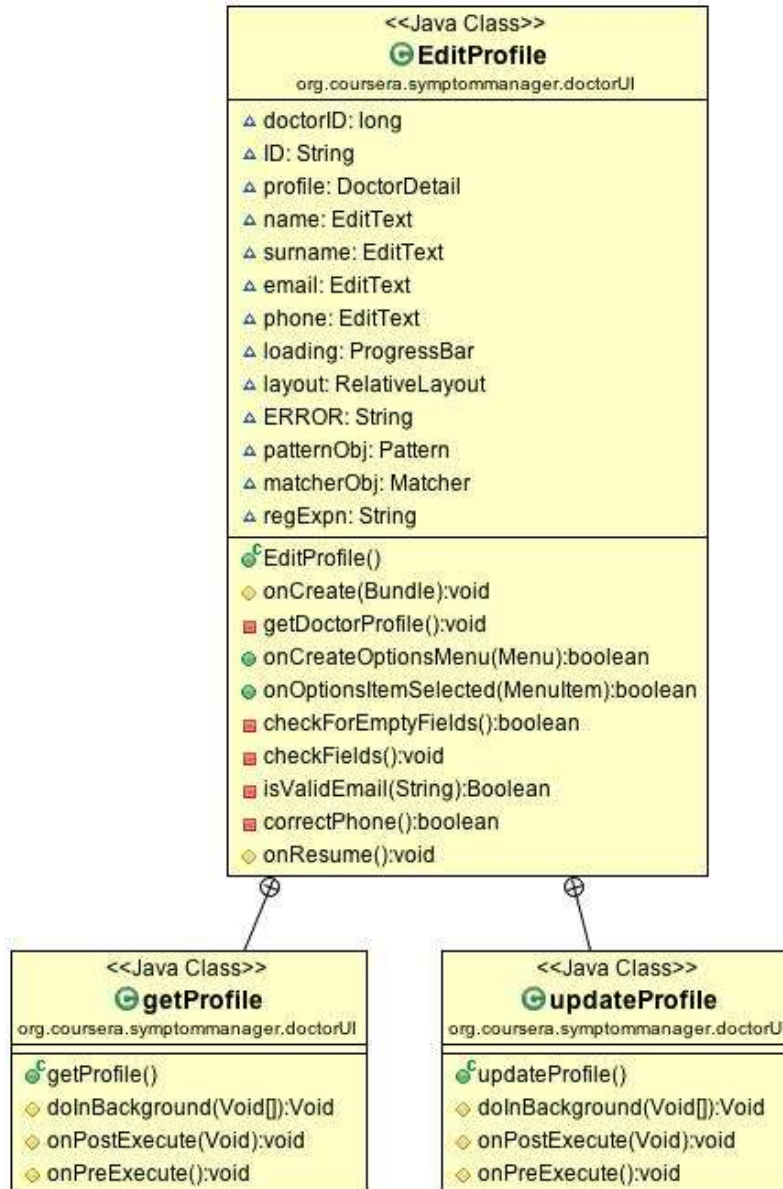
<<Java Class>> 👤 PatientDetail org.coursera.symptommanager.objects	
id: long username: String name: String surname: String email: String phone: String dob: long gender: String lastCheckIn: long notifyTime: String idDoctor: long regid: String severePain: long moderatePain: long cantEat: long uid: String	
PatientDetail() PatientDetail(String,String,String,String,String,Long,String,Long,Long,String,Long,String,Long,Long,Long,String) getUsername():String setUsername(String):void getName():String setName(String):void getSurname():String setId():long setId(long):void setSurname(String):void getEmail():String setEmail(String):void getPhone():String setPhone(String):void getDob():long setDob(long):void getRegid():String setRegid(String):void getGender():String setGender(String):void getLastCheckIn():long setLastCheckIn(long):void getIdDoctor():long setIdDoctor(long):void getNotifyTime():String setNotifyTime(String):void getSeverePain():long setSeverePain(long):void getModeratePain():long setModeratePain(long):void getCantEat():long setCantEat(long):void getUid():String setUid(String):void	

<<Java Class>> 👤 CheckIn org.coursera.symptommanager.objects	
id: long raw: String idPatient: long time: long	
CheckIn() CheckIn(long,String,Long,Long) getRaw():String setRaw(String):void getIdPatient():long setIdPatient(long):void getId():long setId(long):void getTime():long setTime(long):void	

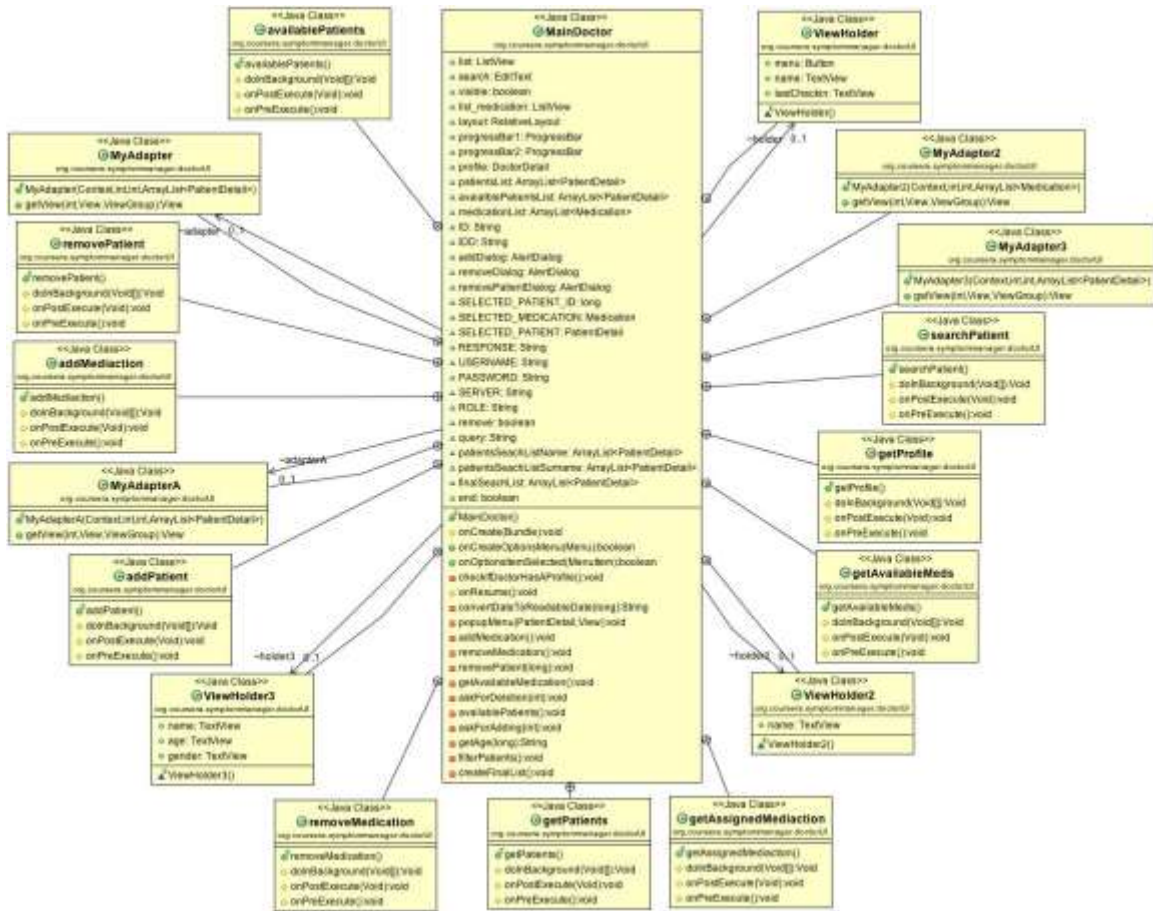
<<Java Class>> ❓ Question org.coursera.symptommanager.objects	
id: long title: String response: String timestamp: long	
Question() Question(long,String,String,Long) getTitle():String setTitle(String):void getResponse():String setResponse(String):void getId():long setId(long):void getTimestamp():long setTimestamp(long):void	

Doctor UI Classes

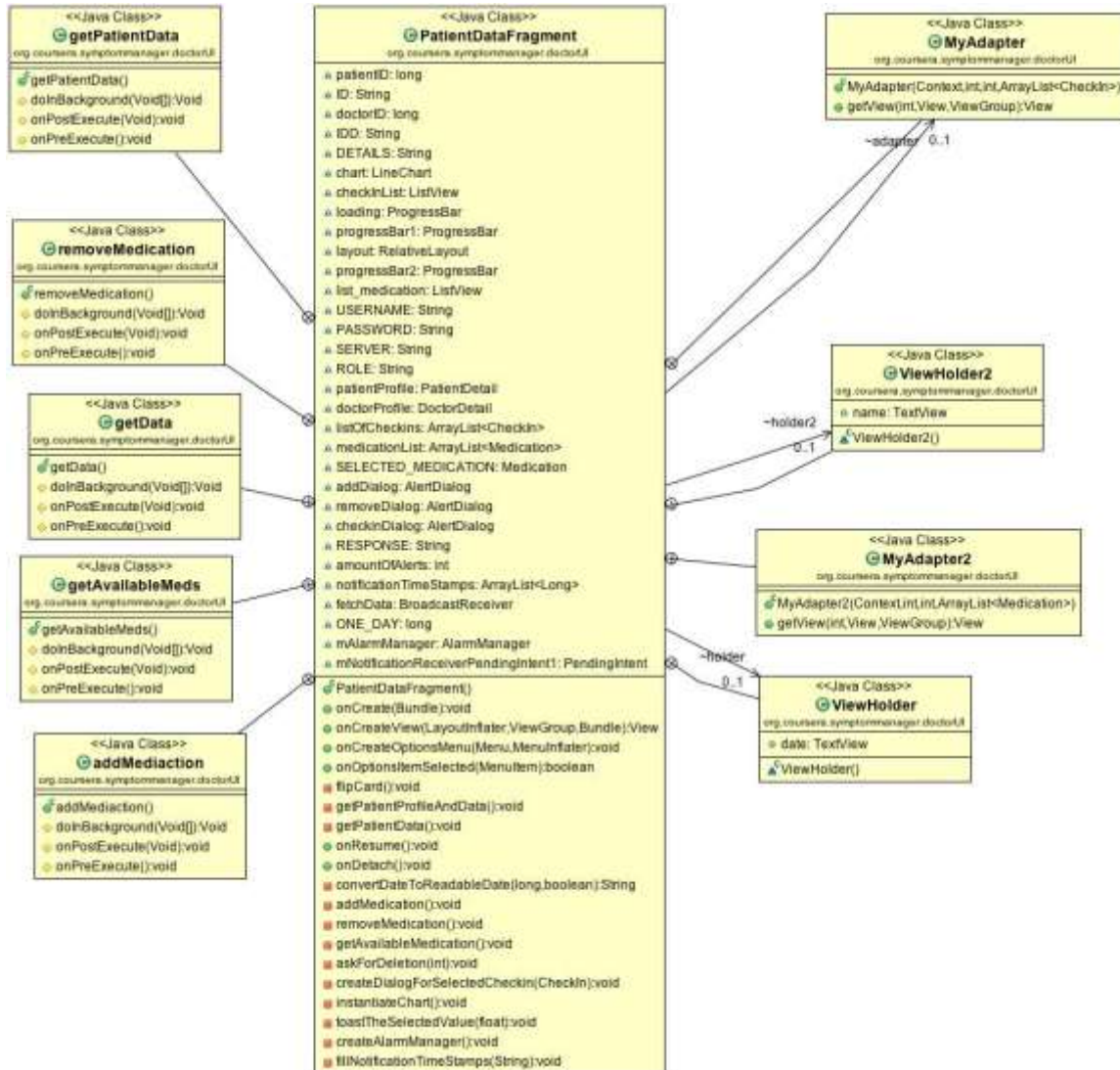
EditProfile



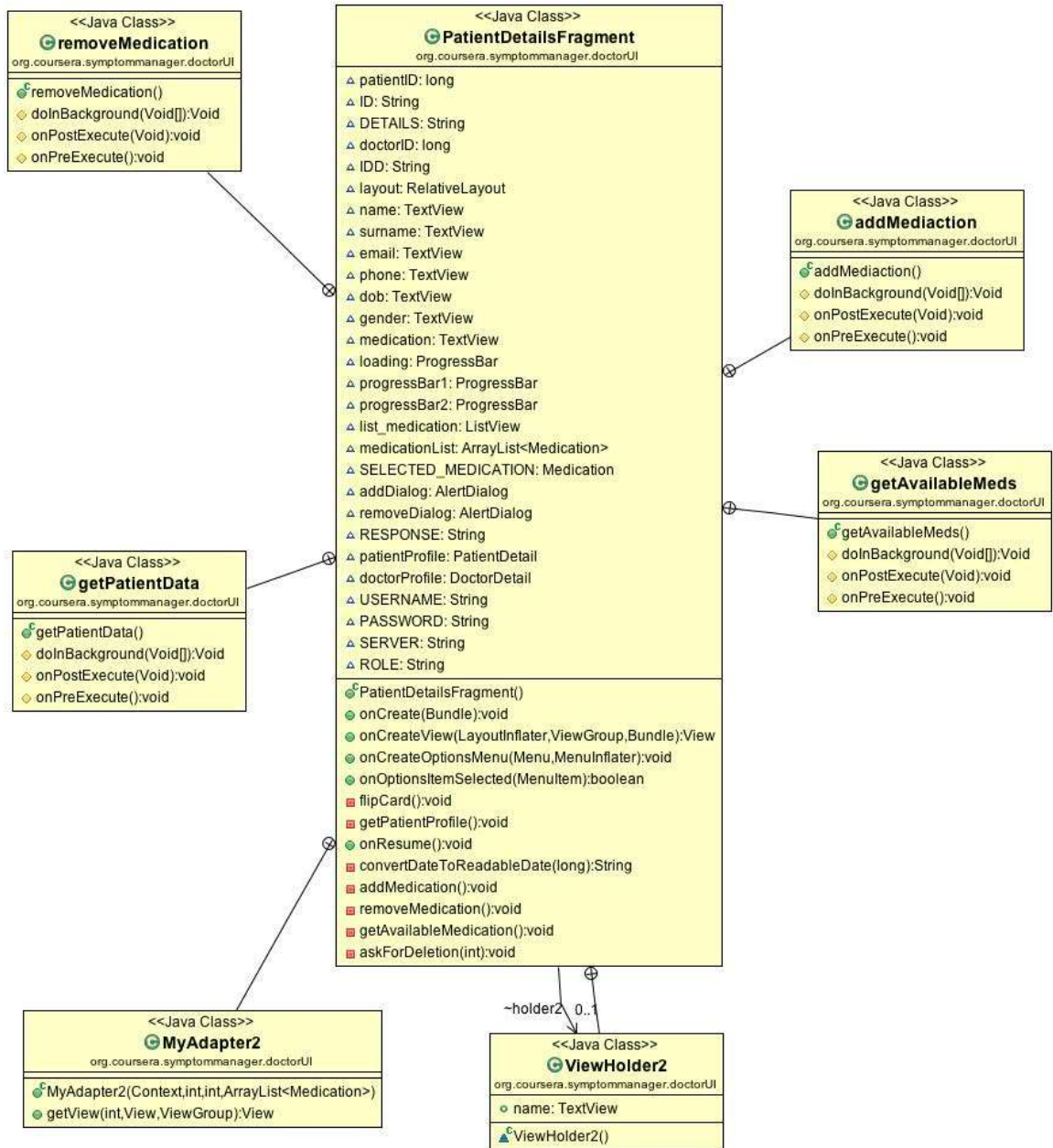
MainDoctor



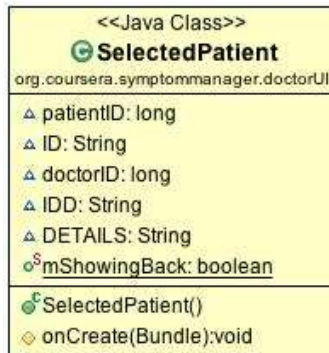
PatientDataFragment



PatientDetailsFragment

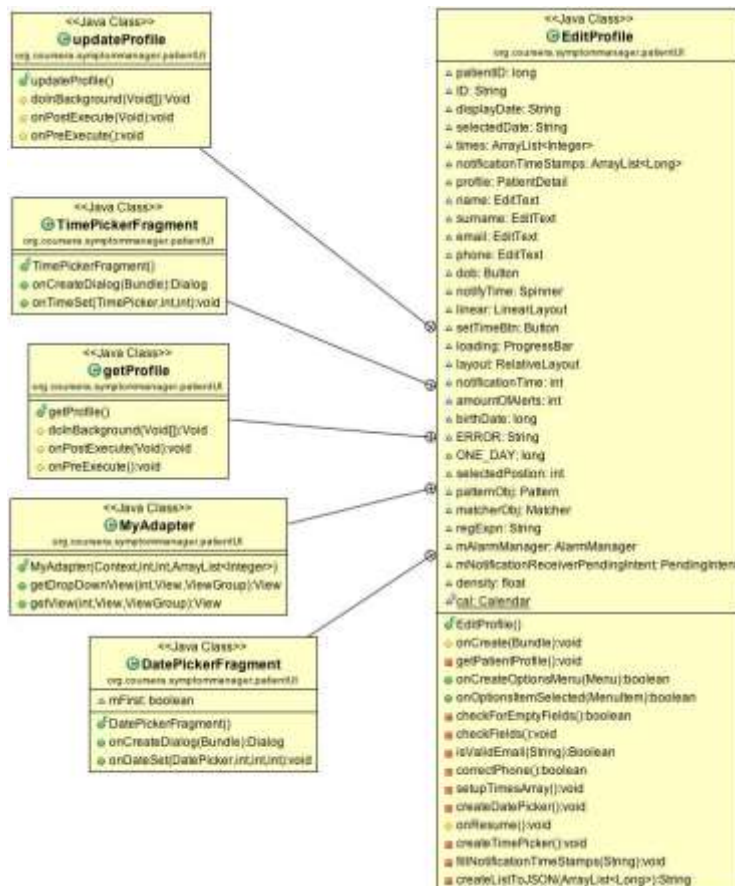


SelectedPatient

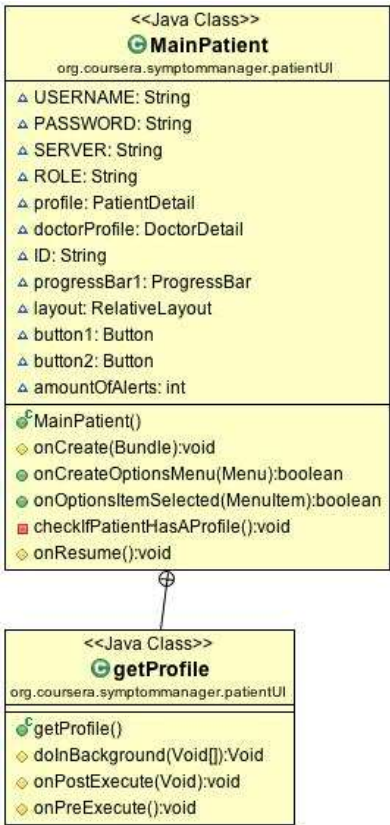


Patient UI Classes

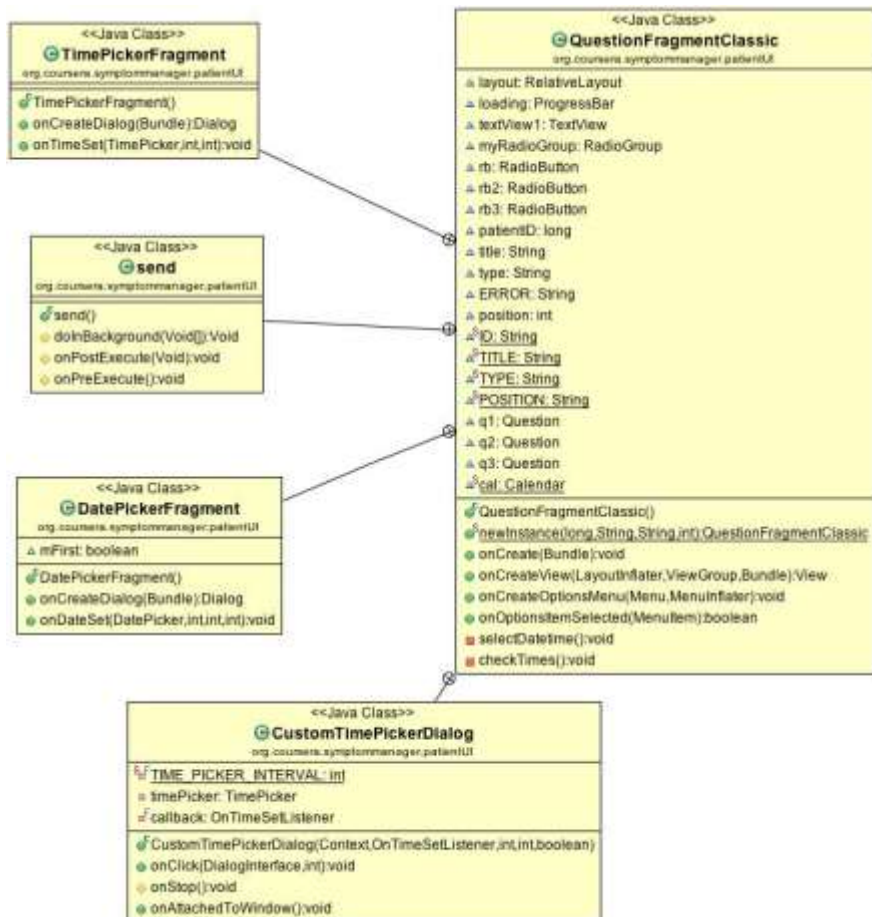
EditProfile



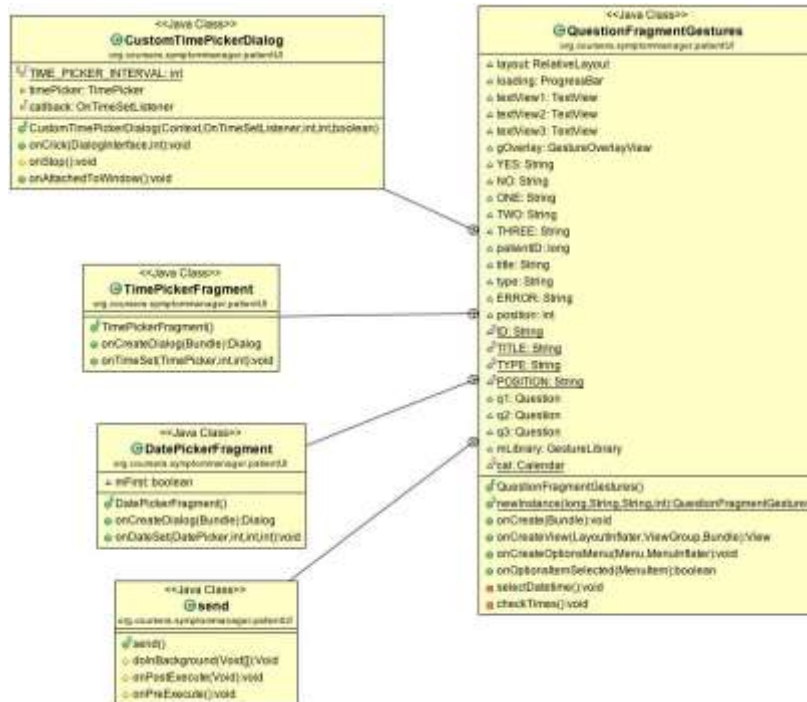
MainPatient



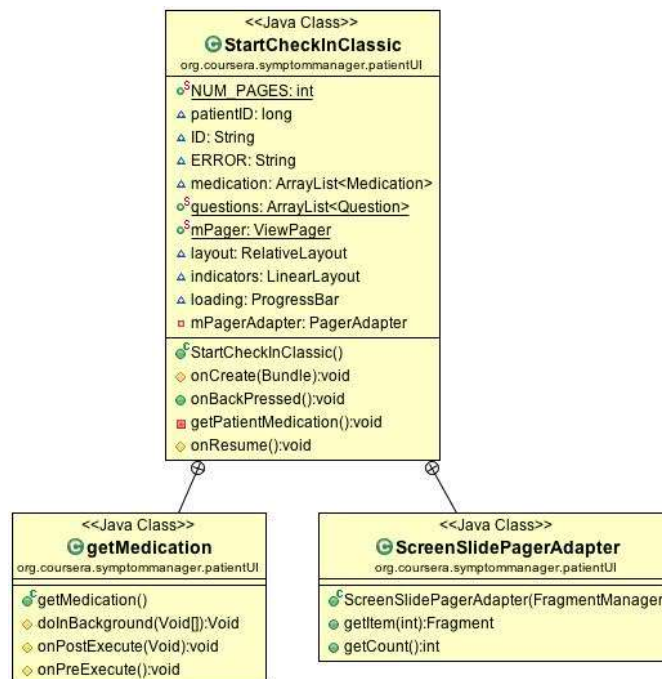
QuestionFragmentClassic



QuestionFragmentGestures



StartCheckInClassic



StartCheckInGestures



Below are the rubric questions with the answers of what I do and how in my project
(Inside the zip file I uploaded there is a second pdf with the mapping of the questions to classes and code lines)

1. Basic Project Requirement:

App supports multiple users via individual user accounts

Yes it does the server will use Spring OAuth 2.0 to support multiple users via a custom UserDetailsService (The users doctors, patients will be hardcoded as this is correct as the Professors said). Users will authenticate via the Spring OAuth 2.0 token endpoint and provide bearer tokens in an authorization header to prove user identity

2. Basic Project Requirement:

App contains at least one user facing function available only to authenticated users

Yes it does in order to use the application you should be successfully connected as one of the hardcoded users. All the user facing functions are available to the authenticated users

3. Basic Project Requirement:

App comprises at least 1 instance of each of at least 2 of the following 4 fundamental Android components:

- Activity
- BroadcastReceiver
- Service
- ContentProvider

Yes it does the app uses:

9 activities

Doctor (Splash, LoginScreenActivity, EditProfile, MainDoctor, SelectedPatient)

Patient (Splash, LoginScreenActivity, EditProfile, MainPatient, StartCheckinClassic, StartCheckinGestures)

4 fragments

Doctor (PatientDataFragment, PatientDetailsFragment)

Patient (QuestionFragmentClassic, QuestionFragmentGestures)

4 broadcast receivers

AlarmHandler (used in order to notify the patient at the times he/she specified in his/hers profile. The default is 4 times a day that means every 6 hours) The AlarmHandler is triggered by a repeating alarm with the specified time by broadcasting an intent at these times a custom action the receiver is registered to listen to.

NotifyDoctorForPain (Used to notify the doctor when one of the following happens. 12 hours of severe pain, 12 hours of I cant eat, 16 hours of moderate pain) The NotifyDoctorForPain is triggered by an alarm manager that fires when one of the criterias are true by broadcasting an intent with a custom action.

ScreenReceiver (used to catch screen on screen of events in order to show an animation every time the screen turns on and a user is logged in either a doctor or a patient. The animation is a welcome message) The ScreenReceiver is triggered by the system when the screen turns off and on. This receiver is registered by a service in order to work because it cant be registered in the manifest as the others)

fetchData (used in order to fetch the patient data every X time, where X is the time the patient specified for the app to notify him to make a checkup (checkin) + 5 minutes, when the doctor is viewing the patient graph.) The fetchData is triggered by an alarm that fires every X time. This broadcast receiver is registered in the life time of the fragment PatientDataFragment because only then we want the data to be refreshed every X time in order to see the changes in the graph)

2 services

GCMIntentService that will handle the push notification. The registration of the device and the income of a push notification

ScreenReceiverService that will register the ScreenReceiver broadcast and will handle the changes of the flags for the screen statuses.

4. Basic Project Requirement:

App interacts with at least one remotely-hosted Java Spring-based service

Yes it does as you see in the UML diagrams for the controllers in the start of this document I use 27 Java Spring-based web services.

5. Basic Project Requirement:

App interacts over the network via HTTP

Yes it does in the setup of the client we use an EasyHttpClient that establishes an HTTP connection

6. Basic Project Requirement:

App allows users to navigate between 3 or more user interface screens at runtime

Yes it does in the doctor UI we have 8 screens (6 activities and 2 fragments) as you can see from the screen shots)

In the patient UI we have 9 screens (7 activities and 2 fragments which are dynamically created according to the patient questions that means we might have more screens in the patient)

7. Basic Project Requirement:

App uses at least one advanced capability or API from the following list (covered in the MoCCA Specialization): multimedia capture, multimedia playback, touch gestures, sensors, animation.**

**Learners are welcome to use ADDITIONAL other advanced capabilities (e.g., BlueTooth, Wifi-Direct networking, push notifications, search), but must also use at least one from the MoCCA list.

Yes it does the application uses the following advanced capability covered in the MoCCA specialization:

Touch gestures (see patient checkup (checkin) with gestures)

Animation (see doctor flip animation and both doctor and patient welcome message on every screen on event)

ADDITIONAL advanced capabilities:

Push notifications

8. Basic Project Requirement:

App supports at least one operation that is performed off the UI Thread in one or more background Threads of Thread pool.

Yes it does all the calls to the web services are made using Async Tasks and in the Splash class the very first activity I use a handler with a runnable in order to execute the setting up of the client and the authentication of the user if the user has already logged in before and hasn't logout. Also I use a delay on the handler that causes the runnable to be added to the message queue, to be run after the specified amount of time elapses in order to make the splash to appear for some seconds.

1. Functional Description and App Requirement:

App identifies a *Patient* as a user with first name, last name, date of birth, a (unique) medical record number, and possibly other identifying information). A patient can login to their account.

Yes it does as you can see in the beginning of this document the PatientDetail object that has id, firstname, lastname, date of birth, gender, phone, email, username, last checkin, notifytime, registration id, severe pain, moderate pain, cant eat, uid. The uid is generated by UUID.randomUUID().toString() that gives us a unique ID.

2. Functional Description and App Requirement:

App defines a *Reminder* as an alarm or notification which can be set to patient-adjustable times (at least four times per day).

Yes it does the application use the AlarmHandler broadcast receiver that is triggered by a repeating alarm that the time of the repeats is being specified by the patient in his profile (see pic 18) default is 4

times. He/she can set 4 – 24 alarms inside a day. Each alarm will fire at the selected time from the patient and will repeat itself every day.

3. Functional Description and App Requirement:

A *Reminder* triggers a *Check-In*, which is defined by the app as a unit of data associated with a *Patient*, a date, a time, and that patient's responses to various questions at that date and time.

Yes it does check the Checkin object that has id, raw (patients responses in raw JSON format), timestamp of the date and time, and patient id in order to know the application which patient made this checkup (checkin)

4. Functional Description and App Requirement:

Check-In includes the question, "How bad is your mouth pain/sore throat?" to which a patient can respond, "well-controlled," "moderate," or "severe."

Yes it does is the first question check pic19 right, pic 20, pic 21 left image and pic 27

5. Functional Description and App Requirement:

Check-In includes the question, "Did you take your pain medication?" to which a Patient can respond "yes" or "no".

Yes it does is the second question check pic19 left image and pic 28

6. Functional Description and App Requirement:

A Check-In for a patient taking more than one type of pain medication includes a separate question for each medication (e.g., "Did you take your Lortab?" followed by "Did you take your OxyContin?"). The patient can respond to these questions with "yes" or "no."

Yes it does is the second question check pic19 left image and pic 28 as dummy data I added (as a doctor) to the patient the Lortab medication and the OxyCotin

7. Functional Description and App Requirement:

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.

Yes it does check the pic20 and pic28

8. Functional Description and App Requirement:

During a Check-In, the patient is asked "Does your pain stop you from eating/drinking?" To this, the patient can respond, "no," "some," or "I can't eat."

Yes it does is the last question check pic 21 right image and pic 29

9. Functional Description and App Requirement:

App defines a Doctor 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.

Yes it does as you can see in the beginning of this document the DoctorDetail object that has id, firstname, lastname, phone, email, username, and registration id.

10. Functional Description and App Requirement:

App allows a patient's Doctor to monitor Check-Ins, with data displayed graphically. The data is updated at some appropriate interval (perhaps when a Check-In is completed).

Yes it does check pic 9 and pic 10. The data displayed in a Chart and in the list view below. The data are updated every by the fetchData receiver that is triggered by a repeating alarm manager every X time which X is the time the patient set the application will inform him/her to have a checkin + 5 minutes

11. Functional Description and App Requirement:

A doctor can search for a given Patient's Check-In data by the patient's name (an exact text search hosted server-side).

Note: Non-exact text searching is not required (e.g. you don't have to suggest, "Did you mean...")

Yes it does. Check pic 6. I'm using a web service that searches the database PatientDetail's Name and Surname columns if they contain the typed string if yes it returns the results and populate them in the list view.

12. Functional Description and App Requirement:

A doctor can update a list of pain medications associated with a Patient. This data updates the tailored questions regarding pain medications listed above in (6).

Yes it does. Check pic 5 description patient's option menu with actions Add medication and Remove Medication. Also in pic 12 and pic 13 you can see the action in the action bar menu. Also in these pic the medication of the patient is dynamically updated in the list below. And also dynamically updates the questions in the patient checkup (checkin) as you can see in the pic 19 right , pic 20 , pic 21 left, pic 28.

13. Functional Description and App Requirement:

A doctor is alerted if a patient experiences 12 of "severe pain," 16 or more hours of "moderate" or "severe pain" or 12 hours of "I can't eat."

Yes it does. This logic is being handles by the NotifyDoctorForPain broadcast receiver. Every time a patient makes a checkup (checkin) the receiver is triggered and checks the selected pain. Also checks the previews declared pain and checks the times then and now if any criteria is true it notifies the doctor. Also when the patient finishes a checkup (checkin) an alarm is set to run every one hour and check the current time with the current patients pain and if a criteria is true notifies the doctor.

14. Functional Description and App Requirement:

A patient's data should only be accessed by his/her doctor over HTTPS.

Yes it does the tomcat is set up over Https on port 8443 and all the data are transferred safely with decryptions.