Android Capstone Project

Symptoms Management

30/11/2014
Coursera: https://class.coursera.org/androidcapstone-001

Table of contents

Requirements 3
Use Cases 5
Use Cases description 5
System design. General description 10
Communications between subsystems 11
Server design 15
Database schema 15
Class diagram 18
REST API 20
Mobile App design 23
Database schema 23
Class diagram 24
Patient classes description 26
Doctor classes description 30
User Interfaces 32

Requirements

First of all, these are the requirements to be met with the system. This list is taken from the project statement.

Functional Requirements

1. The *Patient* is the primary user of the mobile app.
2. The Patient will receive a *Reminder* in the form of an alarm or notification at some adjustable frequency, at least four times per day.
3. Once the Patient acknowledges a Reminder, the app will open for a *Check-In*.
4. During a Check-In, the Patient is asked, “How bad is your mouth pain/sore throat?” and can respond with “well-controlled,” “moderate,” or “severe.”
5. During a Check-In, the Patient is asked, “Did you take your pain medication?”
6. If a Patient is taking more than one type of pain medication, each medication will require a separate question during the Check-In (e.g., “Did you take your Lortab?” followed by “Did you take your OxyContin?”). The Patient can respond to these questions with “no” or “yes.”
7. During a Check-In, if a Patient indicates he or she has taken a pain medication, the Patient will be prompted to enter the time and date he or she took the specified medicine.
8. During the Check-In, a Patient is asked, “Does your pain stop you from eating/drinking?” To this, the Patient can respond, “no,” “some,” or “I can’t eat.”
9. A Patient’s *Doctor* is able to monitor a Patient’s Check-Ins with data displayed graphically.
10. A Doctor can see all his/her Patients and search for a given Patient’s Check-In data by the patient’s name.
11. A Doctor can update a list of pain medications associated with a Patient’s account. This data updates the tailored questions regarding pain medications listed above in (6).
12. A Doctor is alerted if a Patient experiences 12 hours of “severe pain,” 16 hours of “moderate pain or severe pain” or 12 hours of “I can’t eat.”
13. A Patient’s data should only be accessed by his/her Doctor(s) over HTTPS.

Technical Requirements

1. Apps must support multiple users via individual user accounts. At least one user facing operation must be available only to authenticated users.
2. App implementations must comprise at least one instance of at least two of the following four fundamental Android components: Activity, BroadcastReceiver, Service and ContentProvider.
3. Apps must interact with at least one remotely-hosted Java Spring-based service over the network via HTTP
4. At runtime apps must allow users to navigate between at least three different user interface screens. o e.g., a hypothetical email reader app might have multiple screens, such as (1) a ListView showing all emails, (2) a detail View showing a single email, (3) a

compose view for creating new emails, and (4) a Settings view for providing information about the user's email account.

5. Apps must use at least one advanced capability or API from the following list covered in the MoCCA Specialization: multimedia capture, multimedia playback, touch gestures, sensors, or animation. Experienced students are welcome to use other advanced capabilities not covered in the specialization, such as BlueTooth or Wifi-Direct networking, push notifications, or search. Moreover, projects that are specified by commercial organizations may require the use of additional organization-specific APIs or features not covered in the MoCCA Specialization. In these cases, relevant instructional material will be provided by the specifying organization.
6. Apps must support at least one operation that is performed off the UI Thread in one or more background Threads or a Thread pool.
7. There may also be additional project-specific requirements (e.g., required use of a particular project-specific API or service).

Use Cases

This section describes the Use Cases diagram that shows, at high level, the functionality of the system for Patients and Doctors. The Backend functionality to manage Patients or Doctors is not in the requirements of this System.

Use Cases description

For each Use Case, I am going to write a table of contents to describe in more detail the functionality of the System.

Name	Authenticate to the Server
Roles	Patient, Doctor
Description	
The user enters the login and password in the screen. This information is required every time the user starts the App for security reasons. The user and password is sent to the server via https protocol. The server authenticates the user and sends back the OAuth token. This token is used for every request to the server in this session. Once the user is authenticated, a background process communicates with the server and	

starts to download the initial information. If the user is a Patient the server sends back a list of his/her medications, personal information and role information in order to present Patient's screen. If the user is a Doctor, the server sends back only role information and personal information in order to present Doctors' screen.
Considerations
Data for Patients and Doctors and the relationships between them have to be defined at the database server with another computer system. The mobile device has to be connected to WIFI or cellular network. The patient's medication list will be saved internally in the mobile device's database

Functional requirements	Basic Project Requirements

Name	Starts a check-in
Roles	Patient
Description	
Once the user is authenticated successfully, another screen will be shown with the information of the next time reminder and below, a button to start manually a new check-in. Once the user presses the button, the Use Case "Answer check-in questions" is presented. If no medication has been associated with him/her, a message will be shown to alert patient that he/she can't do a check-in.	
Considerations	
The user is authenticated in the server	
Functional requirements	Basic Project Requirements

Name	Sets reminder frequency
Roles	Patient
Description	
The user will be able to change three parameters in the preference screen: 1) The frequency of the Reminders. The options will be: 4 times per day or 6 times per day. 2) The first hour in the morning to receive the first reminder. 3) The last hour to receive the last reminder in a day. This allow for example set a 4 or 6 reminders from 8:00 am to 22:00 pm and from 22:00 to 8:00 not to be disturbed. In this Use case, the user will be able to change other preferences like the interval of time to connect to Server for looking for his/her medications. This functionality will be accessed from the app menu with the label "set preferences"	
Considerations	
Functional requirements	Number 2 (of the list of App Requirements for Symptom Check)

Name	Receives a reminder
Roles	Patient
Description	
At the interval period of time defined by the user, the App will send a reminder in the form of Notification. The user will be able to see this reminder in the notification area of the device. The notification will show an icon and the text message similar to "Please, do a check-in". The user can press the notification at the notification area to open a screen with the use case "Answer check-in questions".	
Considerations	

The patient has a medication list defined by his/her doctor and saved at the mobile device database.	
Functional requirements	Number 2 and 3 (of the list)

Name	Answers check-in questions
Roles	Patient
Description	
The App shows a wizard with several screens with questions about his/her pain. Every screen has only one question with different options in a form of buttons. The questions are described at the requirements numbers 4 to 8. The last question allows users to take a photo of their mouth/throat using the camera in order to be send to the server with the rest of check-in information. Once the user has finished the wizard, all check-in data is sent in a background process via a service. During the wizard some questions appear about the list of medications that the patient is taken. This list of medication is updated from time to time on the server when the doctors change it and a process in background will download the new list from server and save it in the local database.	
Considerations	
The mobile device has to be connected to WIFI or cellular network to send the check-in data.	
Functional requirements	Numbers: 4, 5, 6, 7, 8

Name	Lists all of his/her patients
Roles	Doctor
Description	
Doctors will be able to list all of his/her patients. This list will be ordered by name and lastname and will be the first screen that doctors will see after login screen. From this list, pressing a patient, the doctor will access to the uses case " Shows patient's check-ins"	
Considerations	
In order to protect medical data information, the list of patients will be always requested to the server and will not saved locally. Check-in information is saved in the local database but referenced by the identifier of each patient and not by his or her name. The mobile device has to be connected to WIFI or cellular network.	
Functional requirements	Number 10

Name	Shows patient's check-ins
Roles	Doctor
Description	
It will be shown a list of patient's check-in ordered by date descending. In every row it will show the check-in date, the answer for How bad is your mouth pain? question and the answer for Does your pain stop you from eating/drinking?" question. Pressing any of these check-in rows, the doctor will be able to see all of the check-in data in another screen and they will be able to see a photo taken by patient pressing a button on the screen. The user will be able to zoom and drag the photo in order to watch it in more detail.	
Considerations	
In order to protect medical data information, the list of patients will be always requested to the server and will not saved locally. Check-in information is saved in the local database but referenced by the identifier of each patient and not by his or her name.	

The mobile device has to be connected to WIFI or cellular network.	
Functional requirements	Number 10

Name	Searches a patient
Roles	Doctor
Description	
Doctors will be able to search a patient by his/her exact name. This functionality will be accessible from an option menu that will appear in the patients list screen. If no patient is located by his/her name an error will be shown. If patient is located, the app will show the screen with the list of their check-in data filled by the patient. Pressing on each row, the doctor will be able to see all check-in data.	
Considerations	
The mobile device has to be connected to WIFI or cellular network.	
Functional requirements	Number 10

Name	Monitors a patient
Roles	Doctor
Description	
Doctors will be able to see the patients' pain data in a form of graphic. This will allow doctors to see the patients' pain evolution to take measures for the treatment. This functionality will be accessible from an option menu that will appear in the patient check-ins list screen.	
Considerations	
The medical information will be queried in the mobile device's local database.	
Functional requirements	Number 9

Name	Updates a list of patient's medication
Roles	Doctor
Description	
Doctors will be able to update a list of pain medication associated with a Patient. To do this, a screen will be presented with a list of pain medication. Doctors will be able to add new medication writing the name in an input text and pressing the button Add. Doctors will be able to delete a medication from the list pressing the row medication for a long time and selecting option delete or they will be able to activate again a deleted medication selecting option activate. This functionality will be accessible from an option menu that will appear in the patient check-ins list screen.	
Considerations	
All of the changes will be sent to the server and saved in the mobile device. The mobile device has to be connected to WIFI or cellular network.	
Functional requirements	Number 11

Name	Receives an alert of patient pain
Roles	Doctor
Description	
Doctors will receive a notification in the notification area of the mobile device when a Patient experiences 12 hours of “severe pain,” 16 hours of “moderate pain or severe pain” or 12 hours of “I can’t eat”. Doctors will be able to press the notification and the app will start	

showing a list of patients with monitor needs. Pressing on every patient a patient's checkin-list will be present in order to see this information. When the notification arrives, a sound will be played in order to alert the Doctor that patient needs a better medication.	
Considerations	
A background service in the Doctors' app will request from time to time for new patient's check-ins. When a new check-in comes, it will be analyzed by the app to find out if an alert needs to be notified in the notification area.	
Functional requirements	Number 12

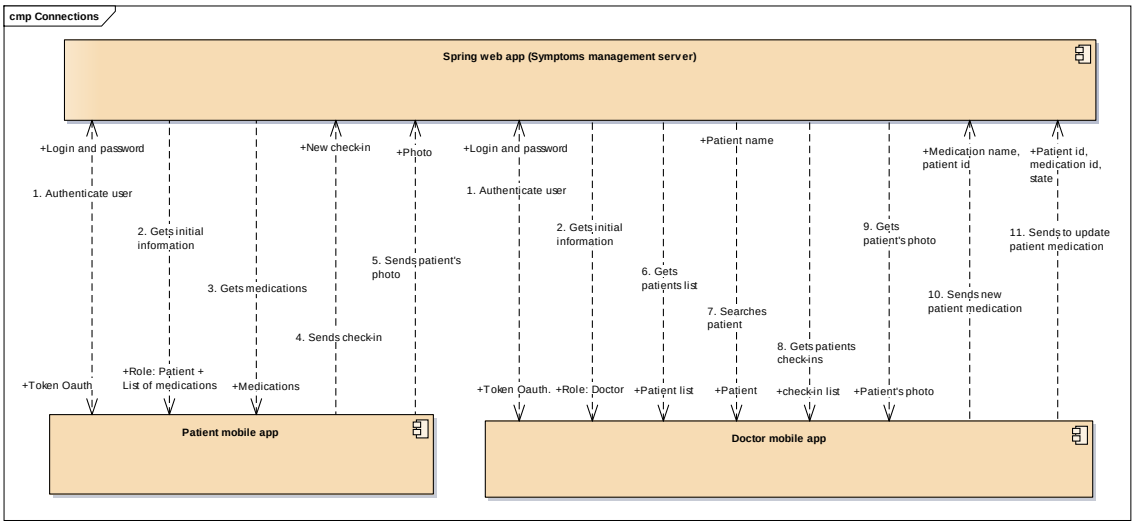
System design. General description

The system is composed by two main subsystems.

1. A Spring web App that exposes an API REST. This web app has a database to store all the information about doctors, patients and the relationships between them. In the chapter "Server design" you can read the technical information about this subsystem.

2. An Android mobile app for users (doctors and patients) to interact with the system. This App has two set of functionalities, one for doctors and another for patients. In the chapter "Mobile App design" you can read the technical information about this subsystem.

The next diagram shows how these two main subsystems communicate to each other. All of the communications are over HTTPS. Two independent boxes in the diagram represent the Android mobile App, despite that both boxes are the same mobile App, to show the communication between the server and the functionality for the doctors and the patients.



The mobile app has been designed to use a local database with the same database schema than the server. The main reasons for this decision are:

- It reduces the amount of traffic between the app and the server. This means cheaper costs for the users.
- It reduces the server overhead because there are fewer requests from clients.

In order to protect medical information, it's important to know that doctors' app will not save any information about the patients names.

Communications between subsystems

This section describes the communications between mobile app and the server.

Path	1. Authenticate user
From	Patient and doctor mobile App
Sent data	Login and password
Return data	Token OAuth
Description	
The user authenticates to the server with an OAuth grant password mechanism.	

Path	2. Get initial information
From	Patient and doctor mobile App
Sent data	
Return data	If authenticated user is a patient: The server returns in JSON format the list of patient medications, information that tells app the user is a patient, and finally patient and doctor information. {"patient": {"id":1, "name":"Chuck", "lastname":"Norris", "birthdate":"1973-06-24", "doctor": {"id":1, "name": "Gregory", "lastname":"House"}, "patientMedicationList":[{"id":1,"name":"Eferelgan","send":false,"active":true},{"id":2,"name":"Ibuprofeno","send":false,"active":true},{"id":3,"name":"Almax","send":false,"active":false}]}, "doctor":null, "role":"PATIENT"} if authenticated user is a doctor: The server returns in JSON format information that tells app the user is a doctor, and doctor information. {"patient":null,"doctor":{"id":1,"name":"Gregory","lastname":"House"},"role":"DOCTOR"}
Description	
The app gets the patient and doctor initial information from the server	

Path	3. Gets medications
From	Patient mobile App
Sent data	The patient Id in the request URL
Return data	The list of patient medications in JSON format [{"id":1,"name":"Eferelgan","send":false,"active":true}, {"id":2,"name":"Ibuprofeno","send":false,"active":true}] • Active: a false value indicates that his/her doctor has been deleted the medication. • send: It is not used at the moment.
Description	
The app gets the patient medications list from the server	

Path	4. Sends check-in
From	Patient mobile App
Sent data	The patient Id in the request URL and in the body a JSON format with all the fields that patient filled in during the check-in {"id":1, "checkinDate":"2014-11-25T16:33:39.697+01:00", "send":false, "howbad":"Severe", "painstop":"I can not eat", "takemedication":true, "alertDoctor":false, "photoPath":"/storage/sdcard0/Pictures/Symptoms Management/IMG_20141125_163330_-1205532288.jpg", "**checkinMedicationList**":[{"id":1,"takeit":true,"takeitDate":"2014-11-25","takeitTime":"16:33","patientMedication":{"id":1,"name":null,"send":false,"active":false}},{"id":2,"takeit":true,"takeitDate":"2014-11-25","takeitTime":"16:33","patientMedication":{"id":2,"name":null,"send":false,"active":false}}], "**patient**":{"id":1,"name":"Chuck", "lastname":"Norris", "birthdate":"1973-06-24", "doctor":null,"patientMedicationList":null}} • howbad values: “well-controlled,” “moderate,” or “severe” • painstop values: “no,” “some,” or “I can’t eat.” • date_checkin: date and time in ISO 8601 format • send: always must be false • photoPath: It does not matter the value. A second connection is needed from client to send the photo.
Return data	The same JSON data with the new id used to insert this record in the server database
Description	
The app sends a new check-in to the server	

Path	5. Sends patient's photo
From	Patient mobile App
Sent data	Binary data in the body, the doctor Id checkin Id and patientId in the request URL
Return data	Status 200
Description	
The app sends the photo to the server	

Path	6. Gets patients list
From	Doctor mobile App
Sent data	DoctorId in the request URL
Return data	A patients list in JSON format [{"id":2, "name":"Samuel", "lastname":"Jackson", "birthdate":"1985-07-21", "doctor": {"id":1, "name": "Gregory", "lastname":"House"}, "patientMedicationList":null}, {"id":1, "name":"Chuck", "lastname":"Norris", "birthdate":"1973-06-24", "doctor": {"id":1, "name":"Gregory", "lastname":"House"},"patientMedicationList":null}]
Description	

The app gets doctor's patient list from the server

Path	7. Searches a patient
From	Doctor mobile App
Sent data	Doctor Id and patient name in the request URL
Return data	A patient object in JSON format {"id":1, "name":"Chuck", "lastname":"Norris", "birthdate":"1973-06-24", "doctor": {"id":1, "name":"Gregory", "lastname":"House"},"patientMedicationList":null}
Description	
The app gets patient data from server	

Path	8. Gets patients check-ins
From	Doctor mobile App
Sent data	Doctor id and patientId in the request URL
Return data	A patients check-in list in JSON format [{"id":3, "checkinDate":"2014-11-25T16:40:55.000+01:00", "send":true, "howbad":"Severe", "painstop":"I can not eat", "takemedication":true, "alertDoctor":false, "photoPath":"photos/photo_3.jpg", "checkinMedicationList":[{"id":5, "takeit":true, "takeitDate":"2014-11-25", "takeitTime":"16:40", "patientMedication": {"id":1, "name":"Eferelgan", "send":false, "active":true}}, {"id":6, "takeit":true, "takeitDate":"2014-11-25", "takeitTime":"16:40", "patientMedication": {"id":2, "name":"Ibuprofeno", "send":false, "active":true}}], "patient":{"id":1, "name":"Chuck", "lastname":"Norris", "birthdate":"1973-06-24", "doctor": {"id":1, "name":"Gregory", "lastname":"House"}, "patientMedicationList":null}}] • show_alert: true or false. To indicate apps that a notification needs to be alert the doctor. • howbad values: "well-controlled," "moderate," or "severe" • painstop values: "no," "some," or "I can't eat." • date_checkin: date and time in ISO 8601 format • photoPath: It does not matter this value. A second connection will be needed in order to download the photo.
Description	
The app gets a new check-ins from the server	

Path	9. Gets patient's photo
From	Doctor mobile App
Sent data	Doctor Id, patient Id and checkin id in the request URL
Return data	Binary data
Description	
The app gets a patient check-in's photo	

Path	10. Sends a new patient medication
From	Doctor mobile App
Sent data	Doctor Id, patient Id and the patient medication name in the request URL
Return data	The medication object saved in the server database in JSON format {"id":7,"name":"Ibuprofeno","send":false,"active":true}
Description	
The app sends a new patient medication to the server	

Path	11. Sends to update a patient medication
From	Doctor mobile App
Sent data	Doctor Id , patient id, the medication id in the request URL, and Status HTTP request code: DELETE (to delete) or PUT (to activate)
Return data	Status 200
Description	
The app sends an update to patient medication to the server	

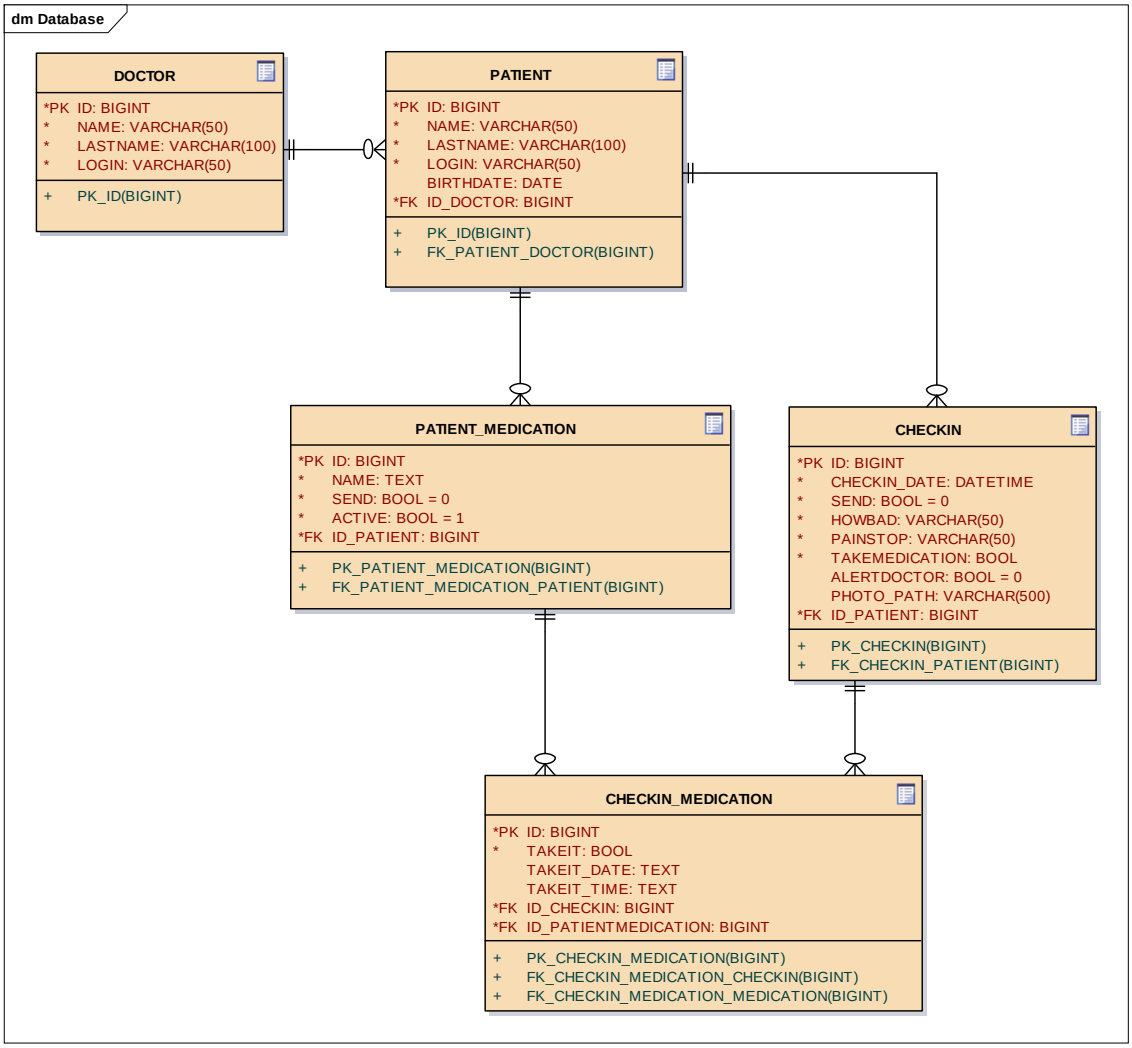
Server design

The server is implemented in Spring and will run in a Tomcat server with MySQL database to store all the information about the system. The server allows to authenticate users with login and password with OAuth password grant mechanism. Every user has an independent account and all of their information is related in the same database. All the doctors have assigned DOCTOR_ROLE and all patients have PATIENT_ROLE, and of course doctors functionality are protected by DOCTOR_ROLE and patients functionality re protected by PATIENT_ROLE. This blocks any attempt to patients to access the doctors functionality and vice versa. If a doctor is a patient too, then he/she will have an account as a patient and a different account as a doctor.

The next three sections describe the server database schema, server class diagram and the REST API.

Database schema

The next diagram shows the system database schema.



This database schema is replicated at mobile app to allow users to reduce traffic and to improve data access from the app. Despite that, not all the information is replicated at mobile

app. The chapter "Database" from "Mobile App design" describes what information is saved for the mobile app.

The next list describes the database tables.

- **DOCTOR**. Represents the doctors of the system. This table is maintained by an external system. There is a one-to-many relationship with patient table.
- **PATIENT**. Represents the patients of the system. A patient has a check-ins list and a medications list. Each patient has only one doctor (I assume this decision in my design)
- **PATIENT_MEDICATION**. Represents the patient medications. Each patient medication row belongs to one patient. This table is maintained by doctors app.
- **CHECKIN**. Represents the patients check-in. Each checkin row belongs to one patient. This table is maintained by patients app.
- **CHECKIN_MEDICATION**. Represents the answers for each patient medication during each check-in. It's a many-to-many relationship between CHECKIN and PATINET_MEDICATION tables. This table is maintained by patients app.

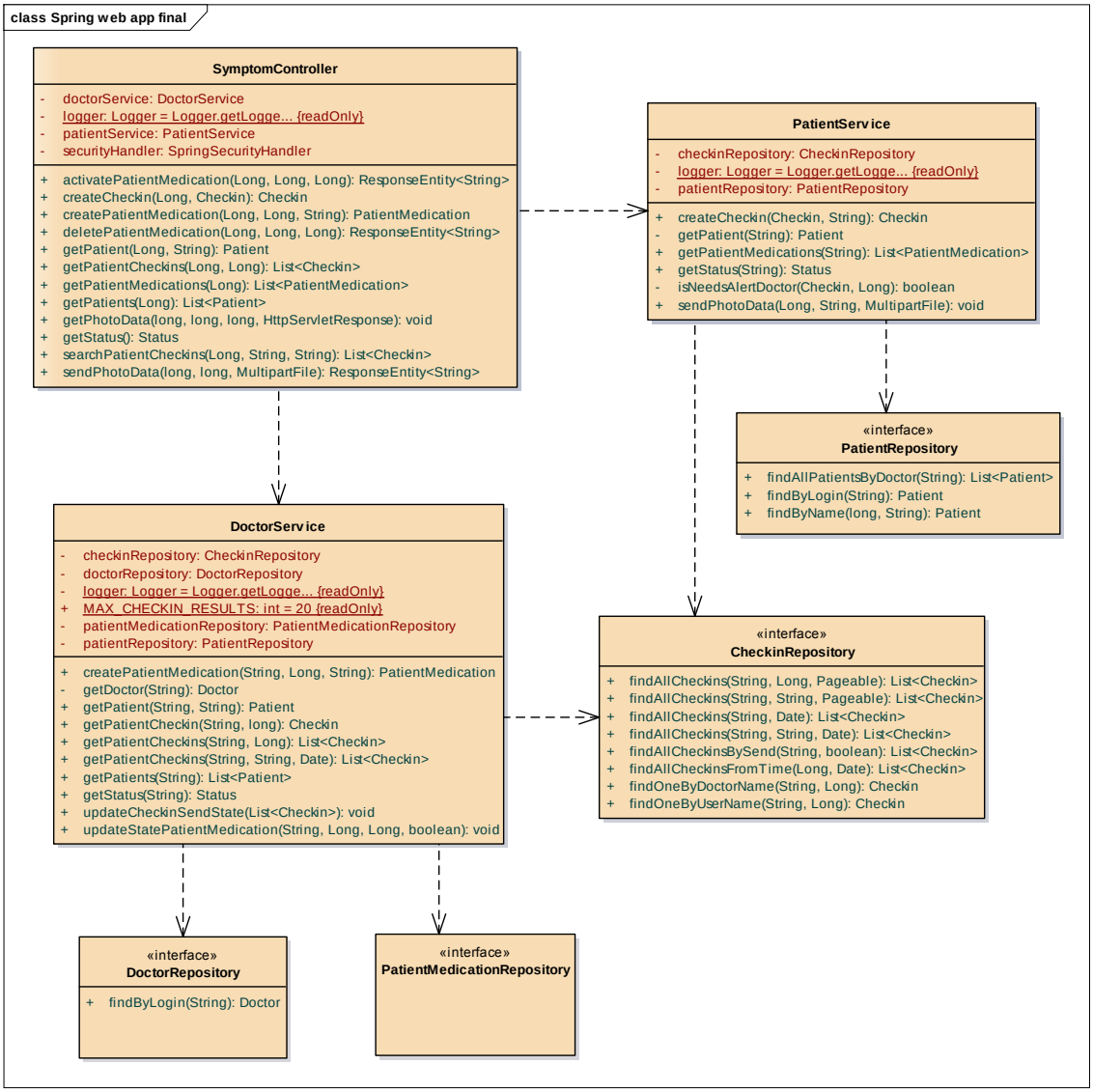
This section describes some fields to clarify the model. The rest of the fields are clear enough.

- **SEND**: BOOL. This field is used to indicate if the information has been sent to mobile app. The default value is 'N'. At the moment is only used in CHECKIN table. When new check-ins arrive to the server and they have not been sent yet, the value is 'N'. When the doctors' mobile app requests the new check-ins that belongs to their patients and the server send these check-ins to mobile app, then the value is updated to 'Y'.
  In the PATIENT_MEDICATION table is not used at the moment because every request from patients' mobile app always sends all patient medications to every patient because I assume it represents a few amount of data.
- **ACTIVE**: BOOL. This field is used to indicate that patient medication has been deleted by doctor. The default value is 'Y' to indicate that patient medication is active and the patient has to answer if they are taking it. If the doctors delete a patient medication the value changes to 'N'. Doctors can active the pain medication again from their apps and set the value again to 'Y'.
- **CHECKIN_DATE**: DATE. Represents the date and time of the patient's check-in. This information contains the patient time zone.
- **ALERT_DOCTOR**: BOOL. This field is very important and indicates that in this check-in, doctor needs to be alerted that their patient needs medical attention. This field is calculate on the server when the patient sends their current check-in. When doctor's app downloads the new check-ins, it realizes if some check-in has this field to true in order to send a notification alert.
- **HOWBAD**: TEXT. Represents the answer for: How bad is your mouth pain/sore throat?". The values can be: "well-controlled", "moderate", "severe".
- **PAINSTOP**: TEXT. Represents the answer for: Does your pain stop you from eating/drinking?". The values can be: "no," "some," or "I can't eat."
- **TAKEMEDICATION**: BOOL. Represents the answer for: "Did you take your pain medication?".

- **TAKEIT**: BOOL. Represents the answer for: “Did you take your ....?" for each patient medication.
- **TAKEIT_DATE**: DATE. Represents the date when patient took his/her medication
- **TAKEIT_TIME**: DATE. Represents the time when patient took his/her medication

Class diagram

The next diagram shows the server class diagram.



The server is designed by three layers:

- Controller layer: This layer publishes an REST API for mobile apps. Only authorized users will access REST API methods. This layer contains a **SymptomController** class. This class accepts requests from mobile apps and calls service layer in order to access to the business logic. Other function in this layer is transform JSON objects to Java and Java objects to JSON in order to communicate with mobile apps. All JSON to Java transformations are made by Jackson library. The next section describes the REST API in more detail.
- Service layer: This layer represents the business logic. This layer accepts calls from Controller layer and invokes the next layer classes in order to access to the database. This layer contains **DoctorService** class to implement doctor business logic and **PatientService** class to implement patient business logic. In this layer, all business methods verify that every user can only access their own information. For example, if a

malicious user invokes an URL and changes the ids to access information that don't belongs to that user, a Security exception will be thrown.

- Data access layer: This layer accesses to the database to insert, update, delete and query patient and doctors' data. This layer contains **DoctorRespository** class to access DOCTOR table, **PatientRespository** class to access PATIENT table, **CheckinRespository** class to access CHECKIN table and **PatientMedicationRespository** class to access PATIENT_MEDICATION table.

REST API

The server publishes an API that is used by the mobile App. This API will be accessible at https://server:8443/symptom.

All /doctors/xxx paths are protected by DOCTOR ROLE and All /patients/xxxx paths are protected by PATIENT ROLE. The protected mechanism is OAuth.

These are the operations that API implements:

- /oauth/token
  Operation type: Authenticated
  Description: The user sends username and password and receives a token if he/she is a valid user or an error if not.

- /getstatus
  Operation type: GET
  Roles allowed: DOCTOR, PATIENT
  Description: If the user is a patient then receives a list of his/her medications, his/her personal information stored in the system and a field named ROLE with PATIENT value. If the user is a doctor then receives only his/her personal information and a field named ROLE with DOCTOR value.
  JSON_FORMAT: see Path: 2. Get initial information at "Communications between subsystems section".

- /patients/{patient_id}/medications
  Operation type: GET
  Roles allowed: PATIENT
  Description: The user with {patient_id} identifier receives a list of his/her medications in JSON format.
  JSON_FORMAT: see Path: 3. Gets medication at "Communications between subsystems section".

- /patients/{patient_id}/checkins
  Operation type: POST
  Roles allowed: PATIENT
  Description: The user with {patient_id} identifier sends a check-in to the server in order to be saved in the central database.
  JSON_FORMAT: see Path: 4. Sends check-in at "Communications between subsystems section".

- /patients/{patient_id}/checkins/{checkin-id}/data
  Operation type: POST
  Roles allowed: PATIENT
  Description: The user with {patient_id} identifier sends a photo associated to check-in with identifier {checkin-id} to the server in order to be saved in the central database.

JSON_FORMAT: see Path: 5. Sends patient's photo at “Communications between subsystems section”.

- /doctors/{doctor_id}/patients
  Operation type: GET
  Roles allowed: DOCTOR
  Description: The user with {doctor_id} identifier receives a list of his/patients in JSON format.
  JSON_FORMAT: see Path: 6. Gets patient list at “Communications between subsystems section”.

- /doctors/{doctor_id}/patients/search?patientName==”patientName”
  Operation type: GET
  Roles allowed: DOCTOR
  Description: The user with {doctor_id} identifier receives a patient information that correspond to patientName in a JSON format.
  JSON_FORMAT: see Path: 7. Searches patient check-ins at “Communications between subsystems section”.

- /doctors/{doctor_id}/patients/{patient_id}/checkins
  Operation type: GET
  Roles allowed: DOCTOR
  Description: The user with {doctor_id} identifier receives a check-in list of his/her patient with {patient_id} identifier in a JSON format. The server only will send the last 20 check-ins.
  JSON_FORMAT: see Path: 8. Gets patients check-ins at “Communications between subsystems section”.

- /doctors/{doctor_id}/patients/checkins/search?patientName=”patientName”& dateFrom=”date”
  Operation type: GET
  Roles allowed: DOCTOR
  Description: There are several possibilities:
  1. Parameter patientName is not null and dateFrom is not null: The user with {doctor_id} identifier receives a check-in list of his/her patient whose name is "patientName". The check-ins returned were created after date "date".
  2. Parameter patientName is not null and dateFrom is null: The user with {doctor_id} identifier receives a check-in list of his/her patient with name "patientName". The server only will send the last 20 check-ins.
  3. Parameter patientName is null and dateFrom is not null: The user with {doctor_id} identifier receives a check-in list that were created after date "date" of **all** his/her patients.
  4. Parameter patientName is null and dateFrom is null: The user with {doctor_id} identifier receives a check-in list of **all** his/her patients that have not been sent

yet. Once the checkins are sent to clients, they will not send it again with this method.

JSON_FORMAT: see Path: 8. Gets patients check-ins at “Communications between subsystems section”.

- /doctors/{doctor_id}/patients/{patient_id}/checkins/{checkin-id}/data
  Operation type: GET
  Roles allowed: DOCTOR
  Description: The user with {doctor_id} identifier gets a patient checkin's photo with {patient_id} and {checkin-id} identifier.
  JSON format: see Path: 9. Gets patient's photo at “Communications between subsystems section”.

- /doctors/{doctor_id}/patients/{patient_id}/medications
  Operation type: POST
  Roles allowed: DOCTOR
  Description: The user with {doctor_id} identifier sends a new medication for his/her patient with {patient_id} identifier in a JSON format.
  JSON format: see Path: 10. Sends new patient medication at “Communications between subsystems section”.

- /doctors/{doctor_id}/patients/{patient_id}/medications/{medication_id}
  Operation type: DELETE or PUT
  Roles allowed: DOCTOR
  Description: The user with {doctor_id} identifier updates the medication's state with {medication_id} for his/her patient with {patient_id} identifier.
  JSON format: see Path: 11. Sends to update patient medication at “Communications between subsystems section”.

Mobile App design

The mobile app is the same app for all users. Users have to authenticate with their username and password in order to allow them to work with the system. The mobile app uses a local database that is accessed via a **ContentProvider** class. The communications with the server uses the Retrofit framework. The mobile App has two **services**, one for download medications, check-ins and patients checkins' photos and another for send check-in information. Finally, the app uses two repeating **alarms**, one for start the download service in a period of time, and another to send a broadcast in a period of time that is received by a **BroadcastReceiver** that sends a notification for a new check-in reminder.

The next three sections describe the mobile app database schema, the class diagram and the user interfaces.

Database schema

The Mobile App uses the same database schema than the server because the app has been designed to reduce data traffic. Because of this, both patients and doctors app need to save and query to this local database, and a process in background (services) synchronize the local data with the server. Of course, each user will only have the data that belongs to each user.

From the **patients'** app point of view, the information that is saved locally is:

- **DOCTOR table**: A row that represents his/her doctor. I assume in my design that a patient has only one doctor.
- **PATIENT table**: A row that represents his/her information.
- **PATIENT_MEDICATION table**: Only his/her medications. Maybe less than ten rows.
- **CHECKIN table**: Only his/her check-ins. Maybe less than a five hundred rows.
- **CHECKIN_MEDICATION table**: Only his/her answers for each medications in every check-in. Maybe less than a thousand rows.

From the **doctors'** app point of view the information that is saved locally is:

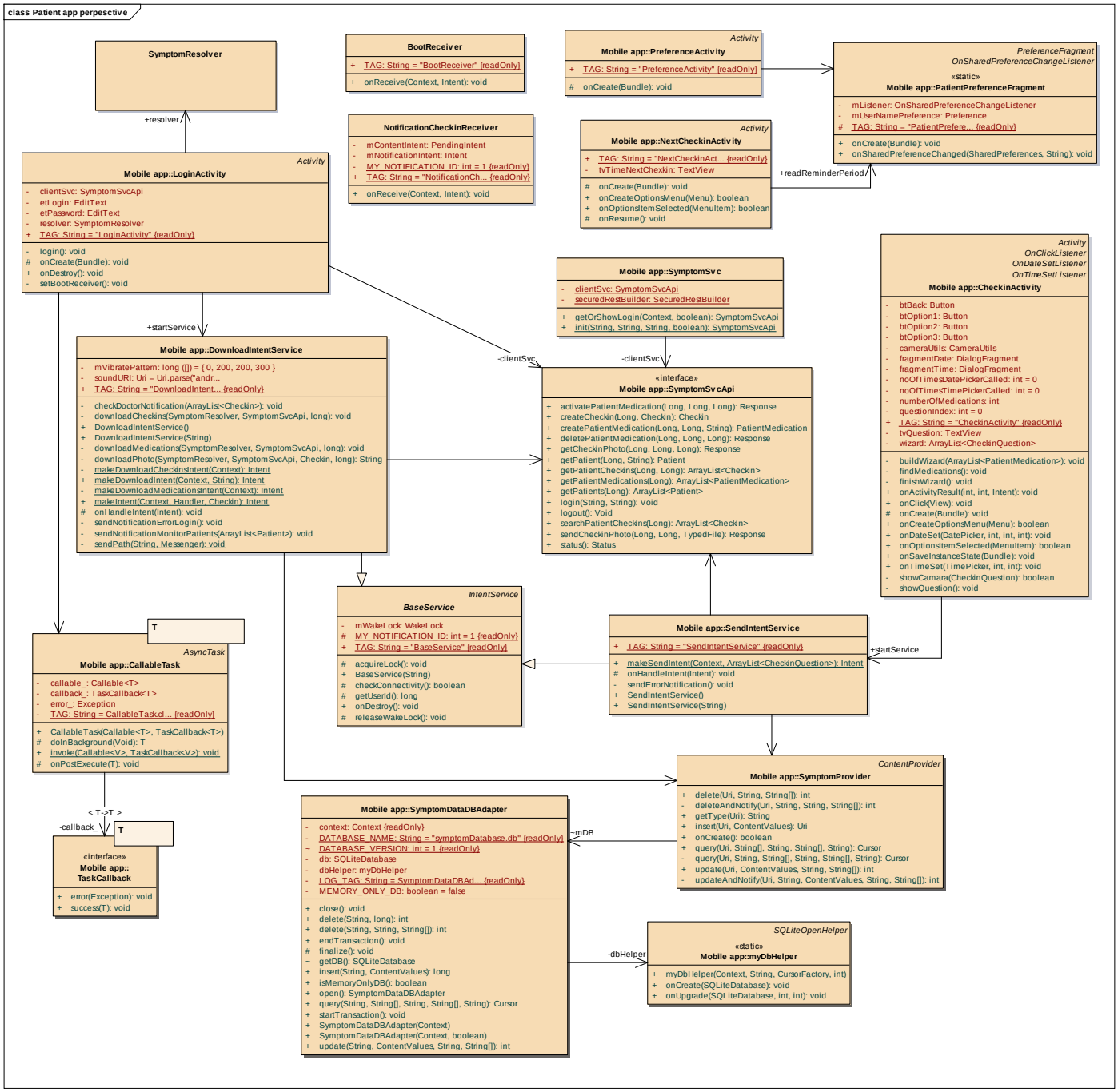
- **DOCTOR table**: Only one row that represents his/her information.
- **PATIENT table**: No information is saved for security reasons.
- **PATIENT_MEDICATION table**: Their patients' medications list. Maybe less than a thousand rows.
- **CHECKIN table**: Their patients' check-ins list. Maybe less than ten thousand rows.
- **CHECKIN_MEDICATION table**: Their patient check-ins' answers list for each medication. Maybe less than ten thousand rows.

To avoid uncontrolled growth of some tables of the doctor mobile app, it would be necessary a functionality that deletes all historical information about patients that leave the treatment. This is not a requirement of this system and for academic reason it won't be implemented.

Class diagram

This section shows the app class diagram in two different diagrams for a better explanation.

The following diagram shows the mobile app class diagram from the patient point of view.


class Patient app perpesctive
SymptomResolver
+resolver
BootReceiver
+ TAG: String = "BootReceiver" {readOnly}
+ onReceive(Context, Intent): void
Activity
Mobile app::PreferenceActivity
+ TAG: String = "PreferenceActivity" {readOnly}
# onCreate(Bundle): void
PreferenceFragment
OnSharedPreferenceChangeListener
«static»
Mobile app::PatientPreferenceFragment
- mListener: OnSharedPreferenceChangeListener
- mUserNamePreference: Preference
TAG: String = "PatientPrefere... {readOnly}
+ onCreate(Bundle): void
+ onSharedPreferenceChanged(SharedPreferences, String): void
NotificationCheckinReceiver
- mContentIntent: PendingIntent
- mNotificationIntent: Intent
- MY_NOTIFICATION_ID: int = 1 {readOnly}
+ TAG: String = "NotificationCh... {readOnly}
+ onReceive(Context, Intent): void
Activity
Mobile app::NextCheckinActivity
+ TAG: String = "NextCheckinAct... {readOnly}
- tvTimeNextChexkin: TextView
onCreate(Bundle): void
+ onCreateOptionsMenu(Menu): boolean
+ onOptionsItemSelected(MenuItem): boolean
# onResume(): void
+readReminderPeriod
Activity
Mobile app::LoginActivity
- clientSvc: SymptomSvcApi
- etLogin: EditText
- etPassword: EditText
- resolver: SymptomResolver
+ TAG: String = "LoginActivity" {readOnly}
- login(): void
onCreate(Bundle): void
+ onDestroy(): void
- setBootReceiver(): void
Mobile app::SymptomSvc
- clientSvc: SymptomSvcApi
- securedRestBuilder: SecuredRestBuilder
+ getOrShowLogin(Context, boolean): SymptomSvcApi
+ init(String, String, String, boolean): SymptomSvcApi
Activity
OnClickListener
OnDateSetListener
OnTimeSetListener
Mobile app::CheckinActivity
- btBack: Button
- btOption1: Button
- btOption2: Button
- btOption3: Button
- cameraUtils: CameraUtils
- fragmentDate: DialogFragment
- fragmentTime: DialogFragment
- noOfTimesDatePickerCalled: int = 0
- noOfTimesTimePickerCalled: int = 0
- numberOfMedications: int
- questionIndex: int = 0
+ TAG: String = "CheckinActivity" {readOnly}
- tvQuestion: TextView
- wizard: ArrayList<CheckinQuestion>
- buildWizard(ArrayList<PatientMedication>): void
- findMedications(): void
- finishWizard(): void
+ onActivityResult(int, int, Intent): void
+ onClick(View): void
# onCreate(Bundle): void
+ onCreateOptionsMenu(Menu): boolean
+ onDateSet(DatePicker, int, int, int): void
+ onOptionsItemSelected(MenuItem): boolean
+ onSaveInstanceState(Bundle): void
+ onTimeSet(TimePicker, int, int): void
- showCamara(CheckinQuestion): boolean
- showQuestion(): void
+startService
Mobile app::DownloadIntentService
- mVibratePattern: long ([]) = { 0, 200, 200, 300 }
- soundURI: Uri = Uri.parse("andr...
+ TAG: String = "DownloadIntent... {readOnly}
- checkDoctorNotification(ArrayList<Checkin>): void
- downloadCheckins(SymptomResolver, SymptomSvcApi, long): void
+ DownloadIntentService()
+ DownloadIntentService(String)
- downloadMedications(SymptomResolver, SymptomSvcApi, long): void
- downloadPhoto(SymptomResolver, SymptomSvcApi, Checkin, long): String
- makeDownloadCheckinsIntent(Context): Intent
+ makeDownloadIntent(Context, String): Intent
- makeDownloadMedicationsIntent(Context): Intent
+ makeIntent(Context, Handler, Checkin): Intent
# onHandleIntent(Intent): void
- sendNotificationErrorLogin(): void
- sendNotificationMonitorPatients(ArrayList<Patient>): void
- sendPath(String, Messenger): void
-clientSvc
-clientSvc
«interface»
Mobile app::SymptomSvcApi
+ activatePatientMedication(Long, Long, Long): Response
+ createCheckin(Long, Checkin): Checkin
+ createPatientMedication(Long, Long, String): PatientMedication
+ deletePatientMedication(Long, Long, Long): Response
+ getCheckinPhoto(Long, Long, Long): Response
+ getPatient(Long, String): Patient
+ getPatientCheckins(Long, Long): ArrayList<Checkin>
+ getPatientMedications(Long): ArrayList<PatientMedication>
+ getPatients(Long): ArrayList<Patient>
+ login(String, String): Void
+ logout(): Void
+ searchPatientCheckins(Long): ArrayList<Checkin>
+ sendCheckinPhoto(Long, Long, TypedFile): Response
+ status(): Status
IntentService
BaseService
- mWakeLock: WakeLock
MY_NOTIFICATION_ID: int = 1 {readOnly}
+ TAG: String = "BaseService" {readOnly}
# acquireLock(): void
+ BaseService(String)
# checkConnectivity(): boolean
# getUserId(): long
+ onDestroy(): void
# releaseWakeLock(): void
Mobile app::SendIntentService
+ TAG: String = "SendIntentService" {readOnly}
+ makeSendIntent(Context, ArrayList<CheckinQuestion>): Intent
# onHandleIntent(Intent): void
- sendErrorNotification(): void
+ SendIntentService()
+ SendIntentService(String)
T
AsyncTask
Mobile app::CallableTask
- callable_: Callable<T>
- callback_: TaskCallback<T>
- error_: Exception
- TAG: String = CallableTask.cl... {readOnly}
+ CallableTask(Callable<T>, TaskCallback<T>)
# doInBackground(Void): T
+ invoke(Callable<V>, TaskCallback<V>): void
# onPostExecute(T): void
< T->T >
-callback_
T
«interface»
Mobile app::
TaskCallback
+ error(Exception): void
+ success(T): void
ContentProvider
Mobile app::SymptomProvider
+ delete(Uri, String, String[]): int
- deleteAndNotify(Uri, String, String, String[]): int
+ getType(Uri): String
+ insert(Uri, ContentValues): Uri
+ onCreate(): boolean
+ query(Uri, String[], String, String[], String): Cursor
- query(Uri, String, String[], String, String[], String): Cursor
+ update(Uri, ContentValues, String, String[]): int
- updateAndNotify(Uri, String, ContentValues, String, String[]): int
Mobile app::SymptomDataDBAdapter
- context: Context {readOnly}
- DATABASE_NAME: String = "symptomDatabase.db" {readOnly}
~ DATABASE_VERSION: int = 1 {readOnly}
- db: SQLiteDatabase
- dbHelper: myDbHelper
- LOG_TAG: String = SymptomDataDBAd... {readOnly}
- MEMORY_ONLY_DB: boolean = false
+ close(): void
+ delete(String, long): int
+ delete(String, String, String[]): int
+ endTransaction(): void
# finalize(): void
~ getDB(): SQLiteDatabase
+ insert(String, ContentValues): long
+ isMemoryOnlyDB(): boolean
+ open(): SymptomDataDBAdapter
+ query(String, String[], String, String[], String): Cursor
+ startTransaction(): void
+ SymptomDataDBAdapter(Context)
+ SymptomDataDBAdapter(Context, boolean)
+ update(String, ContentValues, String, String[]): int
-mDB
SQLiteOpenHelper
«static»
Mobile app::myDbHelper
+ myDbHelper(Context, String, CursorFactory, int)
+ onCreate(SQLiteDatabase): void
+ onUpgrade(SQLiteDatabase, int, int): void
-dbHelper

Patient classes description

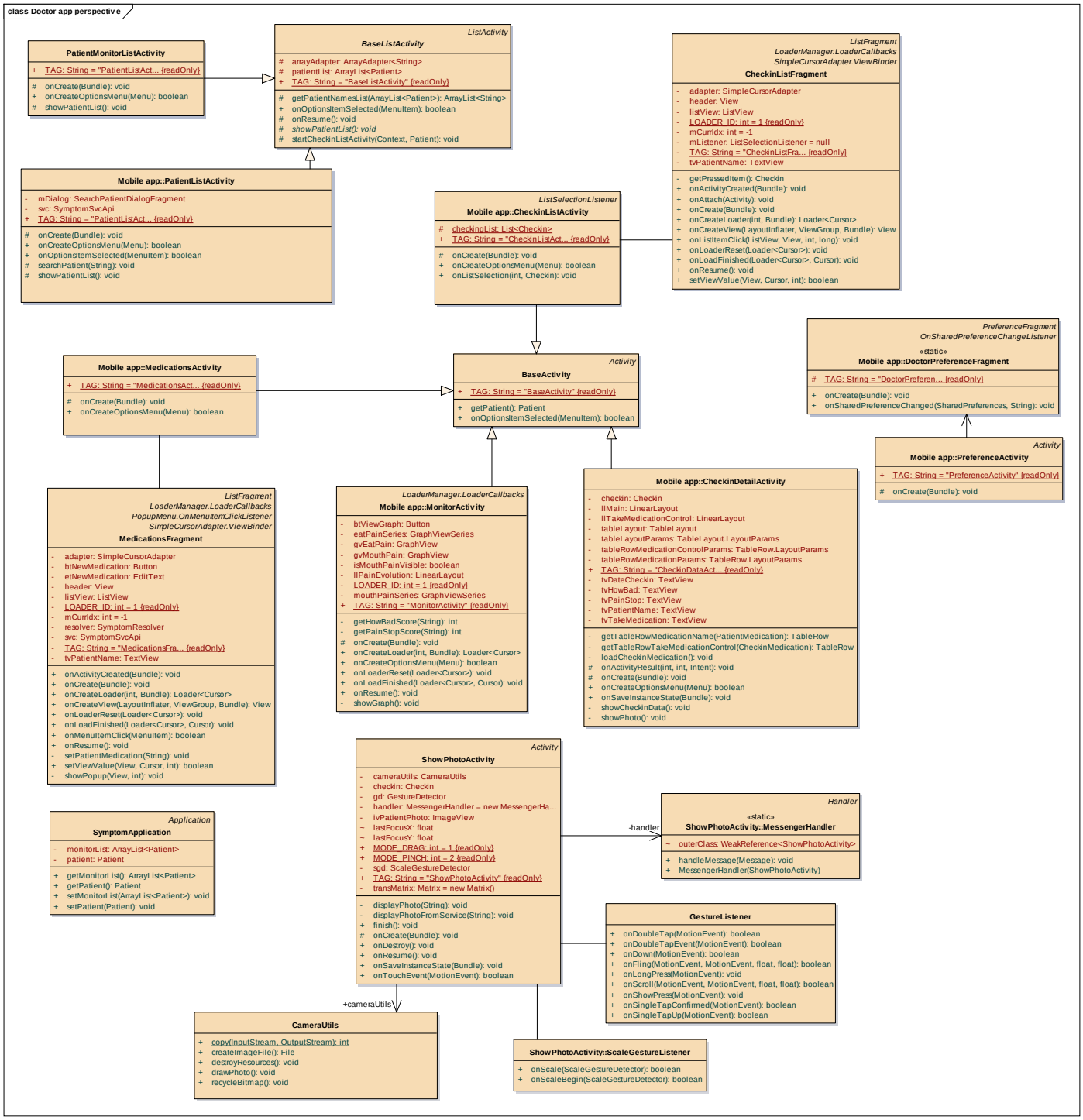
- **LoginActivity:** This activity class has these responsibilities:
    - It shows a Login screen.
    - Once the user inserts his/her username and password and press the Login button, the activity calls **init**() method of **SymptomSvc** class to build a SymptomSvcApi object to communicate to the server using a REST API and **OATUH** authenticated mechanism. The activity calls **getStatus**() from **SymptomSvcApi** class to retrieve who is this user, his/her role in the system and the medication list if the user is a patient. This method calls REST API method: */getstatus* at sever. Once the information is returned and only if it is different from local status information, it is saved in the local database using **SymptomResolver** class that uses a **SymptomProvider** for futures uses. In this case all information from local database is deleted because a different user than the last one is using this mobile device and it is necessary to prevent that it can access to another user's information. The OAuth token that has been returned from the server is saved as a private preference to allow services in Background to make calls to the server. All of this process is made via **CallableTask** class for doing it in background.
    - If the user is a patient, the activity starts a repeating alarm for check-ins reminders. To do this, the activity associates a broadcast to alarm and when the alarm is fired, Android sends a broadcast. This broadcast is received by **NotificationCheckinReceiver** class that sends a notification to the notification area. At this point, because it is a repeating alarm, a new alarm will be scheduled for the next reminder at the interval frequency defined by the user at user preferences. The activity needs to read the SharedPreference object to know that value.
    - if the user is a patient or doctor, the activity starts a repeating alarm for download new medications or new checkins depending if the user is a patient or doctor. To do this, the alarm is associated to the service **DownloadIntentService**. When the alarm is fired, Android starts the **DownloadIntentService** service to start a process in background to download medications or checkins. This is done sending a different intent object to the service. Once the alarm is fired, a new alarm will be scheduled for the next download process at the interval frequency defined by the user at user preferences. The activity needs to read the SharedPreference object to know that value.
    - Finally the activity starts the **NextCheckinActivity** class if the user is a patient or **PatienListActivity** if the user is a doctor.
- **PreferenceActivity** and **PatientPreferenceFragment**: This activity allows user to change the reminders frequency, the first hour of a day to receive a reminder, the last hour of a day to receive a reminder and finally the data download frequency.
- **NextCheckinActivity:** This activity shows a screen with the date and time of the next check-in. The activity reads the SharedPreference reminder frequency value defined in **PatientPreferenceFragment** class. Additionally allows user to press a button on the

screen to start a new check-in manually. If the user presses this button the activity starts the **CheckinActivity** class.

- **CheckinActivity:** This activity shows a screen with a wizard that allows user to answer all questions defined in the check-in step by step. The first action is to read the patient medication list from local database using **SymptomResolver** class. This is done in background using **CallableTask**. If no medication is present in the local database, an alert is shown to the user and the process can't continue. In other case, the activity presents a wizard screen with all the questions. In the last question the user can take a photo of their mouth/throat. Once the user has finished, the activity starts the **SendIntentService** class to send the check-in information, the photo and finally saves the check-in data into the local database using **SymptomResolver** class**.**
- **DownloadIntentService:** This service downloads medications, check-ins or patient checkin's photo depending of the intent object sent to the service. The service calls **getPatientMedications**(), **searchPatientCheckins**() or **getCheckinPhoto()** from **SymptomSvcApi** class. These methods call REST API methods: /patients/{id_patient}/medications,
  /doctors/{doctor_id}/patients/checkins/search,
  /doctors/{doctor_id}/patients/checkins/{checkin_id}/data respectively at server.
  Once the information is returned, it is saved in the local database using **SymptomResolver** class. If the user is a doctor he/she is alerted if a patient experiences 12 hours of "several pain", "16 or more hours of moderate or severe pain" or 12 hours of "I can't eat". The service finds out this scenario in the returned check-in list information and in this case it sends a notification to the doctor mobile's notification area.
  The service uses the OAuth token saved in the private App preferences to be authenticated to the server. Another important consideration is that this service and the **sendIntentservice** checks three things in order to do they work:
  1) Is there Internet connection availability in order to connect to the server?. If it is not available the downloadIntentService does not do anything but the sendIntentService sends a notification to alert patient that the check-in could not be send to the server.
  2) Can I connect to the REST Api Server?. If it is not, both services sends a notification to the user to alert that maybe it is needed to authenticated again.
  3) Acquire a Wake lock to keep the mobile CPU running to prevent Android kills the service while it is working. Once the service finishes its work, release the lock.
- **SendIntentService:** This service class sends the current check-in filled by the patient and the photo if it was taken to the server. To do this, the service uses **createCheckin**() from **SymptomSvcApi** class, that calls REST API method: /patients/{patient_id}/checkin in order to sent the check-in to the server, and finally, the service saves the new check-in in local database with **SymptomResolver** class. To send the photo the service calls **sendCheckinPhoto**() from **SymptomSvcApi** class that calls REST API method:
  /doctors/{doctor_id}/patients/{patient_id}/checkin/{checkin_id}/data
  The service uses the OAuth token saved in the private App preferences to be authenticated to the server.

- **NotificationCheckinReceiver:** This is a **BroadcastReceiver** class to receive the broadcast that is sent by the check-in reminder alarm when the alarm is fired. Once the broadcast is received, a new notification is sent to mobile device notification area to alert patient to make a new check-in.
- **BootReceiver:** This is a **BroadcastReceiver** class to receive the broadcast when the mobile device is reboot. In this class the alarms are set up again.
- **CallableTask** and **TaskCallback:** The **CallableTask** is an AsyncTask for doing work in the background. This class accepts two parameters. The first one is a **Callable** object with a method named call, that it will be executed in the **doInBackground**() method of the **CallableTask**. The second one is an **TaskCallback** object with two methods: **success**() that it will be executed if the **CallableTask** finishes ok and **error**() if not. This class allows other classes to execute work in background.
- **SymptomResolver, SymptomProvider, SymptomDataDBAdapter** and **myDBHelper:** Theses four classes allow other classes to access to the local database for reading and writing information.
- **SymptomSvc** and **SymptomSvcApi:** These two classes allow the other classes to access to the server for requesting or getting information.

The following diagram shows the mobile app class diagram from the doctor point of view. Despite that these classes use the LoginActivity, DownloadIntentService, CallableTask, SystomResolver, SymptomProvider and SymptomSvcApi classes, all of them are not shown because they are explained in the patient class diagram.


class Doctor app perspective
PatientMonitorListActivity
+ TAG: String = "PatientListAct... {readOnly}
# onCreate(Bundle): void
+ onCreateOptionsMenu(Menu): boolean
# showPatientList(): void
ListActivity
BaseListActivity
arrayAdapter: ArrayAdapter<String>
patientList: ArrayList<Patient>
+ TAG: String = "BaseListActivity" {readOnly}
# getPatientNamesList(ArrayList<Patient>): ArrayList<String>
+ onOptionsItemSelected(MenuItem): boolean
# onResume(): void
# showPatientList(): void
# startCheckinListActivity(Context, Patient): void
ListFragment
LoaderManager.LoaderCallbacks
SimpleCursorAdapter.ViewBinder
CheckinListFragment
- adapter: SimpleCursorAdapter
- header: View
- listView: ListView
- LOADER_ID: int = 1 {readOnly}
- mCurrIdx: int = -1
- mListener: ListSelectionListener = null
- TAG: String = "CheckinListFra... {readOnly}
- tvPatientName: TextView
- getPressedItem(): Checkin
+ onActivityCreated(Bundle): void
+ onAttach(Activity): void
+ onCreate(Bundle): void
+ onCreateLoader(int, Bundle): Loader<Cursor>
+ onCreateView(LayoutInflater, ViewGroup, Bundle): View
+ onListItemClick(ListView, View, int, long): void
+ onLoaderReset(Loader<Cursor>): void
+ onLoadFinished(Loader<Cursor>, Cursor): void
+ onResume(): void
+ setViewValue(View, Cursor, int): boolean
Mobile app::PatientListActivity
- mDialog: SearchPatientDialogFragment
- svc: SymptomSvcApi
+ TAG: String = "PatientListAct... {readOnly}
# onCreate(Bundle): void
+ onCreateOptionsMenu(Menu): boolean
+ onOptionsItemSelected(MenuItem): boolean
# searchPatient(String): void
# showPatientList(): void
ListSelectionListener
Mobile app::CheckinListActivity
checkingList: List<Checkin>
+ TAG: String = "CheckinListAct... {readOnly}
# onCreate(Bundle): void
+ onCreateOptionsMenu(Menu): boolean
+ onListSelection(int, Checkin): void
PreferenceFragment
OnSharedPreferenceChangeListener
«static»
Mobile app::DoctorPreferenceFragment
TAG: String = "DoctorPreferen... {readOnly}
+ onCreate(Bundle): void
+ onSharedPreferenceChanged(SharedPreferences, String): void
Mobile app::MedicationsActivity
+ TAG: String = "MedicationsAct... {readOnly}
# onCreate(Bundle): void
+ onCreateOptionsMenu(Menu): boolean
Activity
BaseActivity
+ TAG: String = "BaseActivity" {readOnly}
+ getPatient(): Patient
+ onOptionsItemSelected(MenuItem): boolean
Activity
Mobile app::PreferenceActivity
+ TAG: String = "PreferenceActivity" {readOnly}
# onCreate(Bundle): void
ListFragment
LoaderManager.LoaderCallbacks
PopupMenu.OnMenuItemClickListener
SimpleCursorAdapter.ViewBinder
MedicationsFragment
- adapter: SimpleCursorAdapter
- btNewMedication: Button
- etNewMedication: EditText
- header: View
- listView: ListView
- LOADER_ID: int = 1 {readOnly}
- mCurrIdx: int = -1
- resolver: SymptomResolver
- svc: SymptomSvcApi
- TAG: String = "MedicationsFra... {readOnly}
- tvPatientName: TextView
+ onActivityCreated(Bundle): void
+ onCreate(Bundle): void
+ onCreateLoader(int, Bundle): Loader<Cursor>
+ onCreateView(LayoutInflater, ViewGroup, Bundle): View
+ onLoaderReset(Loader<Cursor>): void
+ onLoadFinished(Loader<Cursor>, Cursor): void
+ onMenuItemClick(MenuItem): boolean
+ onResume(): void
- setPatientMedication(String): void
+ setViewValue(View, Cursor, int): boolean
- showPopup(View, int): void
LoaderManager.LoaderCallbacks
Mobile app::MonitorActivity
- btViewGraph: Button
- eatPainSeries: GraphViewSeries
- gvEatPain: GraphView
- gvMouthPain: GraphView
- isMouthPainVisible: boolean
- llPainEvolution: LinearLayout
- LOADER_ID: int = 1 {readOnly}
- mouthPainSeries: GraphViewSeries
+ TAG: String = "MonitorActivity" {readOnly}
- getHowBadScore(String): int
- getPainStopScore(String): int
onCreate(Bundle): void
+ onCreateLoader(int, Bundle): Loader<Cursor>
+ onCreateOptionsMenu(Menu): boolean
+ onLoaderReset(Loader<Cursor>): void
+ onLoadFinished(Loader<Cursor>, Cursor): void
+ onResume(): void
- showGraph(): void
Mobile app::CheckinDetailActivity
- checkin: Checkin
- llMain: LinearLayout
- llTakeMedicationControl: LinearLayout
- tableLayout: TableLayout
- tableLayoutParams: TableLayout.LayoutParams
- tableRowMedicationControlParams: TableRow.LayoutParams
- tableRowMedicationParams: TableRow.LayoutParams
+ TAG: String = "CheckinDataAct... {readOnly}
- tvDateCheckin: TextView
- tvHowBad: TextView
- tvPainStop: TextView
- tvPatientName: TextView
- tvTakeMedication: TextView
- getTableRowMedicationName(PatientMedication): TableRow
- getTableRowTakeMedicationControl(CheckinMedication): TableRow
- loadCheckinMedication(): void
onActivityResult(int, int, Intent): void
onCreate(Bundle): void
+ onCreateOptionsMenu(Menu): boolean
+ onSaveInstanceState(Bundle): void
- showCheckinData(): void
- showPhoto(): void
Activity
ShowPhotoActivity
- cameraUtils: CameraUtils
- checkin: Checkin
- gd: GestureDetector
- handler: MessengerHandler = new MessengerHa...
- ivPatientPhoto: ImageView
~ lastFocusX: float
~ lastFocusY: float
+ MODE_DRAG: int = 1 {readOnly}
+ MODE_PINCH: int = 2 {readOnly}
- sgd: ScaleGestureDetector
+ TAG: String = "ShowPhotoActivity" {readOnly}
- transMatrix: Matrix = new Matrix()
- displayPhoto(String): void
- displayPhotoFromService(String): void
+ finish(): void
# onCreate(Bundle): void
+ onDestroy(): void
+ onResume(): void
+ onSaveInstanceState(Bundle): void
+ onTouchEvent(MotionEvent): boolean
Handler
«static»
ShowPhotoActivity::MessengerHandler
~ outerClass: WeakReference<ShowPhotoActivity>
+ handleMessage(Message): void
+ MessengerHandler(ShowPhotoActivity)
-handler
Application
SymptomApplication
- monitorList: ArrayList<Patient>
- patient: Patient
+ getMonitorList(): ArrayList<Patient>
+ getPatient(): Patient
+ setMonitorList(ArrayList<Patient>): void
+ setPatient(Patient): void
GestureListener
+ onDoubleTap(MotionEvent): boolean
+ onDoubleTapEvent(MotionEvent): boolean
+ onDown(MotionEvent): boolean
+ onFling(MotionEvent, MotionEvent, float, float): boolean
+ onLongPress(MotionEvent): void
+ onScroll(MotionEvent, MotionEvent, float, float): boolean
+ onShowPress(MotionEvent): void
+ onSingleTapConfirmed(MotionEvent): boolean
+ onSingleTapUp(MotionEvent): boolean
+cameraUtils
CameraUtils
+ copy(InputStream, OutputStream): int
+ createImageFile(): File
+ destroyResources(): void
+ drawPhoto(): void
+ recycleBitmap(): void
ShowPhotoActivity::ScaleGestureListener
+ onScale(ScaleGestureDetector): boolean
+ onScaleBegin(ScaleGestureDetector): boolean

Doctor classes description

- **PreferenceActivity** and **DoctorPreferenceFragment**: This activity allows user to change data download frequency.
- **PatientListActivity:** This activity shows the doctor patient list screen. The activity calls **getPatients**() method from **SymptomSvcApi** class that calls REST API method: /doctors/{doctor_id}/patients in order to download this information. Once the information is returned is showed in the screen with a String array adapter. All of this process is done in background using **CallableTask** class. If the user presses one row of the patient list the **CheckinListActivity** is started in order to show patient check-in list. From the menu of this activity the user can search a patient by his/her name. A dialog is showed to enter the patient name. Once the user enters the patient name and presses the search button, it calls **getPatient**() from **SymptomSvcApi** class that calls REST API method: /doctors/{doctor_id}/patients/search. If no information is returned from the server an error will be shown. If the information is returned, the **CheckinListActivity** is started.
- **PatientMonitorListActivity:** This activity shows a patient list with medical attention. This activity is started when the user presses the notification that was sent by the download service when it realizes that some patients need medical attention. If the user presses one row of the patient monitor list the **CheckinListActivity** is started in order to show patient check-in list.
- **CheckinListActivity and CheckinListFragment**: This activity shows the patient's check-ins. The activity loads a **CheckinListFragment** that makes all the work. The fragment reads patient's check-ins from the local database using **a LoaderManager**. If the user presses on one row of the check-in list, the **CheckinDetailActivity** will be started.
- **CheckinDetailActivity:** This activity shows the check-in data information on the screen that it was send by **CheckinListActivity**. In this activity the user can press a button to see patient check-in's photo. When the user presses this button the **ShowPhotoActivity** is started.
- **ShowPhotoActivity:** This activity shows patient check-in's photo. The activity tries to load the photo from the local file system. If it is not locally saved, the activity starts the **DownloadIntentService** to get the photo from the server in the background. This is done calling RES API method: /doctors/{doctor_id}/patients/{patient_id}/checkin/{checkin_id}/data. Once the photo is downloaded, the service saves the photo in the local file system and updates check-in column path photo in the local database using **SystemResolver** class. Finally, the service sends the photo path to the activity via a handler, and the activity loads the photo and display it using **CameraUtils** class. The user can make zoom and drag on the photo. To allow this, the activity uses ScaleGestureDetector and GestureDetector classes.
- **MonitorActivity:** This activity shows a graph with the patient pain evolution. The doctor can access this activity from the menu of CheckinListActivity. The activity reads this information from local database using **a LoaderManager**. The activity uses a third

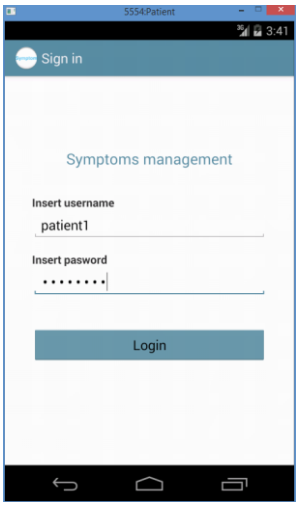
library to draw the graph. You can find more information of this library at this link: http://android-graphview.org/

- **MedicationsActivity and MedicationsFragment:** This activity shows the patient medication list. The doctor can access this activity from the menu of **CheckinListActivity** class. The activity loads a fragment that makes all the work. The fragment reads this information using **LoaderManager** class. The doctor can add new medication writing the medication name into the field and pressing Add button. The activity calls **createPatientMedication**() method from **SymptomSvcApi** class that calls REST API POST method: /doctors/{doctor_id}/patients/{patient_id}/medications. Once the server returns the success of the operation, the new medication is saved into the local database and is shown into the medication list screen. This process is done in background using **CallableTask** class. Another possibility is to delete or activate a patient medication pressing a long time on a medication row. A contextual menu will be shown with a delete and activate options. If the doctor presses once of these options, the activity calls **deletePatientMedication**() or **activatePatientMedication**() method from **SymptomSvcApi** class that calls REST API DELETE method: /doctors/{doctor_id}/patients/{patient_id}/medications/{medication_id} or PUT method: /doctors/{doctor_id}/patients/{patient_id}/medications/{medication_id}. Once the server returns the success of operation, the patient medication's state is updated in the local database. This process is done in background using **CallableTask** class.
- **CameraUtils**: This is a Utility class to draw the photo in an ImageView.
- **SymptomApplication**: This class extends from Application and it is used to save in memory the Patient object that was selected on **PatientListActivity** and **PatientMonitorListActivity** and to save the patients list that needs medical care. This allow activities to share this information and to avoid them to save it to disk in the case the activity's method onSaveInstanceState() is called.

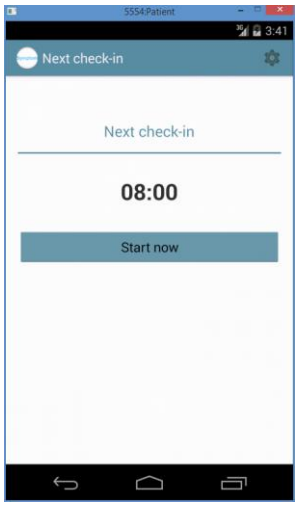
User Interfaces

These are the app user interfaces.

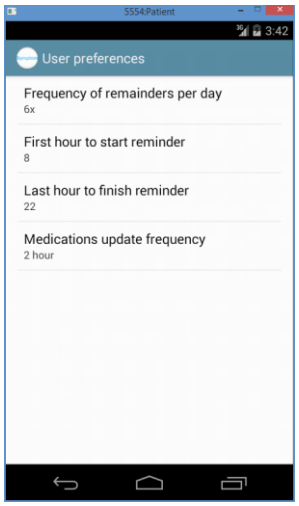
Patients



Login screen (same for doctors)



Next check-in

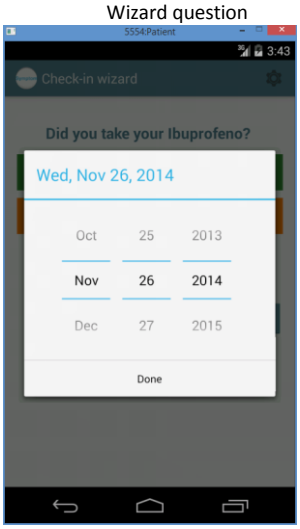


Patients Preference screen

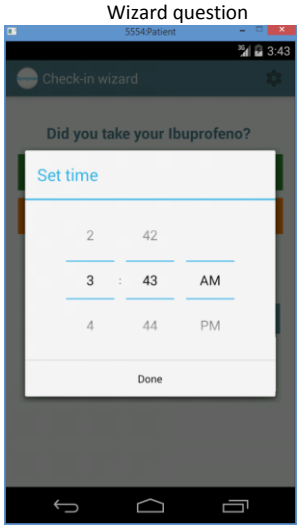
Wizard question

Wizard question

Wizard question

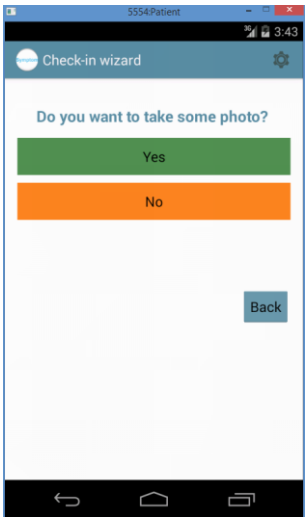


Picker date for wizard question

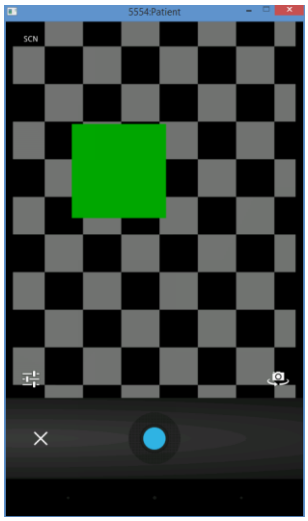


Picker time for wizard question

Wizard question



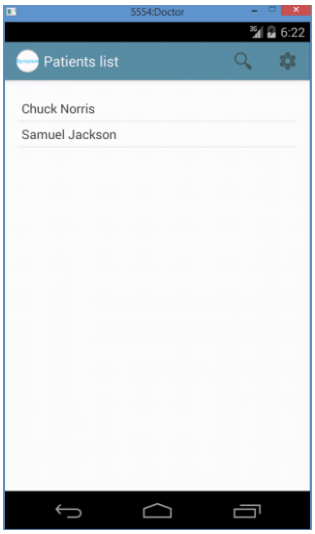
Wizard extra question



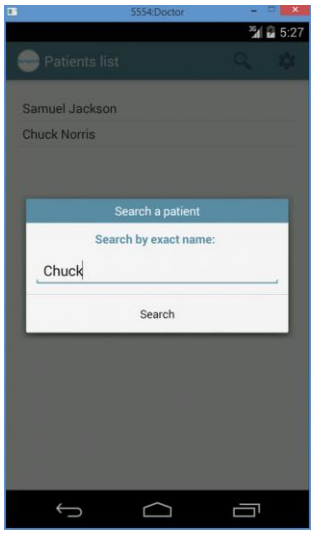
Taken photo

Check-in reminder

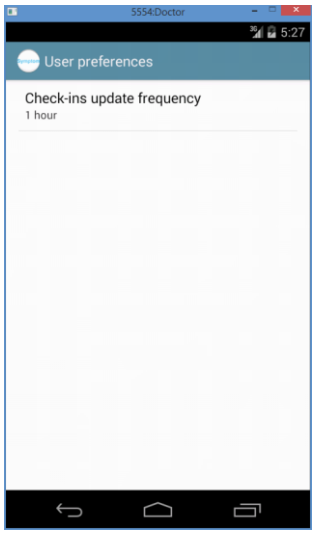
Doctors:



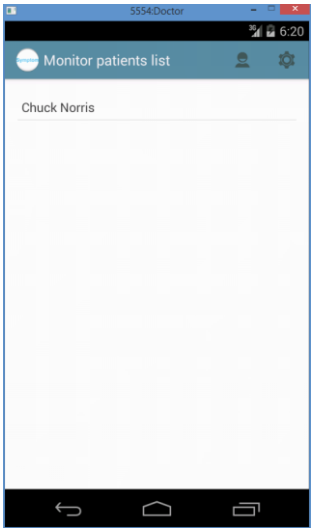
Patient list



Search a patient



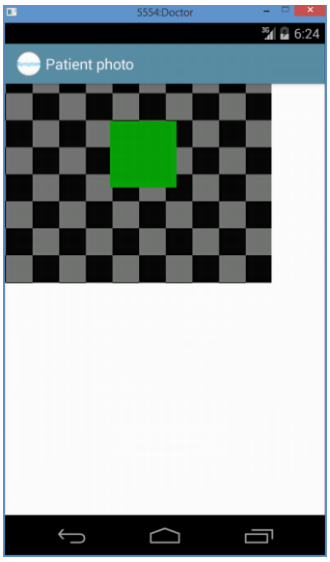
Doctor Preferences screen



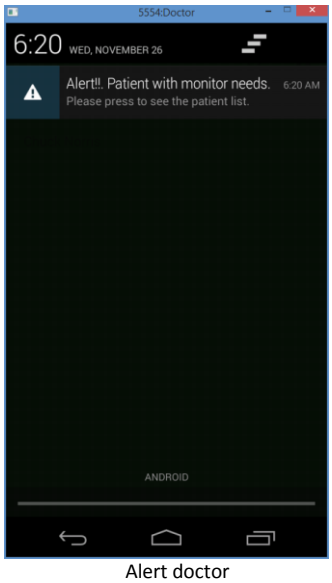
Monitor patient list

Patient list check-ins

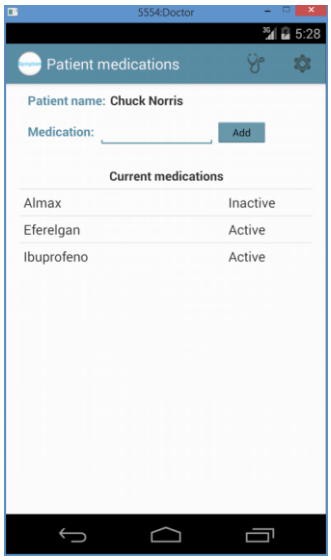
Check-in data



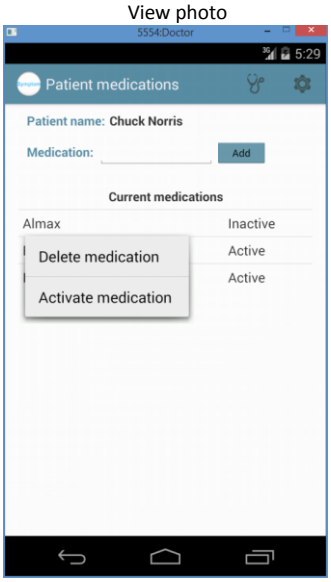
View photo



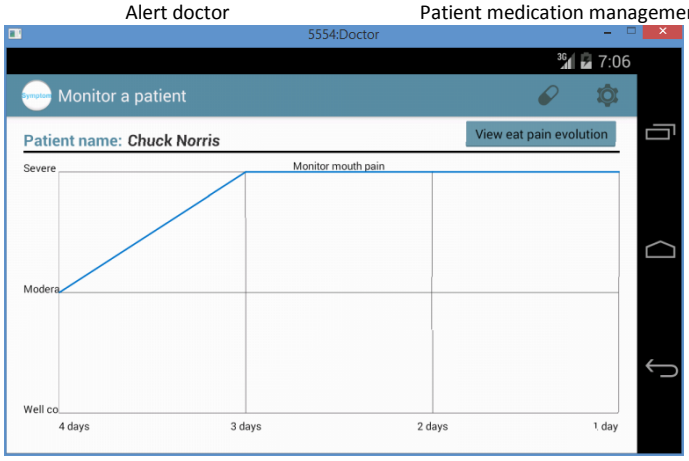
Alert doctor



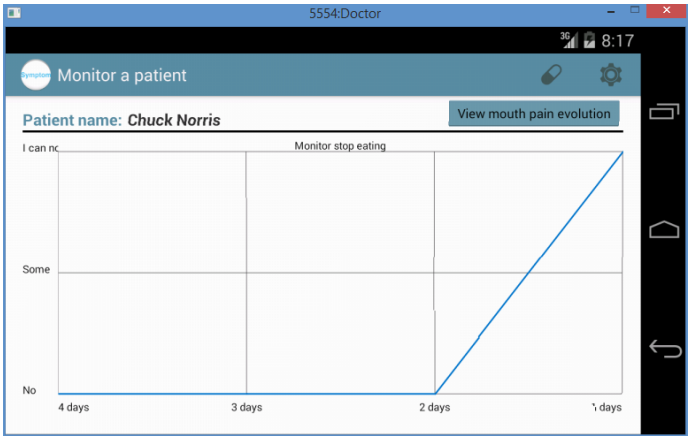
Patient medication management



Patient medication management



Monitor a patient. Graph pain mouth evolution



Monitor a patient. Graph stop eating evolution