Impatient Application

This document describes the main components of the Impatient Application.

1.Spring-Boot Servlet description

- **Servlet side classes.**

Both the patient and the admin app are for their data dependent on the Spring-Boot servlet. The queue behaviour is determined there. Lets look at some of the classes here :

Patient: Is present in all the app parts. It is the basic unit of transaction and is made Parcelable to make transfer from Activities to Services and vice versa easy. It can be put as an extra in an Intent now and just as easy be extracted. Its key member variables are gender, first name, last name, medical record id, access code (for secure registration),appointmendate, birthdate, and notified(to check whether a notification is necessary.), and status.

PatientStatus : An enumeration that holds the different states a Patient can have for the WaitingQueue :unavailable, available, temorary unavailable and under treatment. This object is a member variable of a Patient object.

WaitingQueue : Is an object that resides as a singleton in the server. It is @Autowired so that it can be accessed throughout the server app. Its key member is a CopyOnWriteArrayList<Patient> which we call the queue from now on; this type of list is chosen because it is thread-safe and maintains its order. The WaitingQueue has methods that are typical for ArrayLists, like inserting, deleting and swapping. These methods come from client requests. Another type of methods in the WaitingQueue are methods that have to do with evaluation and corresponding queue behaviour. These methods run in a separate thread that is started at the beginning of a treatment session.

Machine: Is a unit of calculation that represents a patient being treated; it's main method is a thread that counts down from the treatment time to zero. The remaining time is reported every second and is used to calculate waiting times. A machine also knows which patient is being treated at that moment.

AdminSettings: These hold the treatment time that the WaitingQueue had to use. Also the amount of Treatment Machines is set in this object. The adminsettings can be changed by a client but at the cost of a reset of the WaitingQueue.

- **WaitingQueue behaviour**

The WaitingQueue behaviour is roughly as follows :

- a session is started by an admin.

Server now looks in the appointment list and fetches the patients that have an appointment on that day. The treatment time and the amount of treatment machines are grabbed from the AdminSettings object. A fixed sized array of Machine objects is created, which will be accessed during the session when client requests come to put patients under treatment.

- A thread is started to monitor changes, calculate waiting times, and reorder the queue
- The underlying ordering principles are as follows :

-people "under treatment" are put at the front of the queue
-people "available" are put after people under treatment.
-people that are delayed come after that.
-when a person comes under treatment a countdown timer is started that counts down from the given treatment time down to zero; this concept is encapsulated in a "Machine" object

- It is only the first two available patients that have priority in the queue. When availability would be awarded front positions right away it would disturb making an appointment at a specific time. The first two is an arbitrary choice depending on the availability of one treatment machine and 15 minutes treatment time. (it wouldn't matter any more at what time you would appear at the center; people with an appointment at 10:00 would be faced with many people in front of them when they would not check in right away, it would invoke a run)

- Now the following formula is used to calculate the waiting time :

R = remaining treatment time in a currently running Machine.

Waiting time = R + (rowposition * treatmentTime)/amountOfMachines

- When an admin terminates the session this thread is interrupted and default values are restored.

- **Api logistics**

The Spring-Boot server uses Retrofit and Mongo to mediate between storage and dynamic use.

- Https request are streamlined as follows :

AdminSvcApi : interface with retrofit annotated methods.
AdminController : Main entry point for https requests, working with an @Autowired AdminControllerOps object. Requests come in here and return values are given. All requests are passed through to the Ops object. This pattern increases readability and workability a lot.
AdminControllerOps :implementing the AdminSvcApi methods and doing the heavy lifting. It is holding an @Autowired WaitingQueue object, as well as a @Autowired MongoPatientRepository, holding the list of Patients.

The same pattern is followed by the methods that are used by the Patient app.
PatientSvcApi
PatientController
PatientControllerOps

- Security is handled as follows:

ImpatientUser : this is a class that implements UserDetails. Usernames, Passwords and Roles are stored in this class.
ImpatientUserDetailsService : this class implements UserDetailsService and is used by the Oauth framework .
OauthSecurityConfiguration : main class to handle User Authentication. It defines two roles "ROLE_ADMIN" and "ROLE_USER" , which makes sure that patients will never be able to access methods that list other patients' data.
ImpatientUserRepository : A Mongo repository that stores ImpatientUser objects. For testing purposes this repository is created and deleted when the server app starts up, and adds some default users.

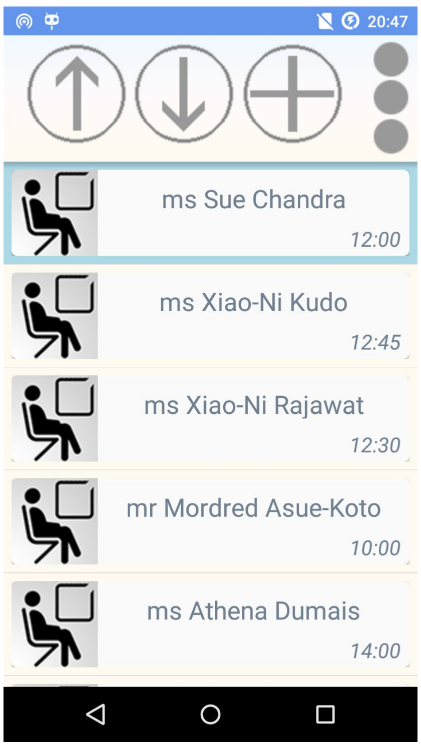
- Persistence is handled as follows:

MongoTemplate : A class that helps to make queries inside a Mongo Collection
MongoPatientRepo : A mongo collection that stores Patient objects.
ImpatientUserRepository : A mongo collection that stores ImpatientUser objects and holds credentials. It is used by the Oauth2SecurityConfiguration object.
Why Mongo?
MySql seems to be the most widely used persistence concept but I decided to give Mongo a shot cause I never feel at ease with MySql. It is like stepping into an other world with different laws and principles, to which one has to adapt. The concept of a non-relational database seems to be a likely solution for this unease. Mongo has a very object oriented style. To fetch an object from storage is like "give me this object", instead of "give me all the fields from this row". You miss the overhead of composing and decomposing all the fields of your object which can be a source of errors. When you look at how data are stored it looks like Json objects, one gets more then ever the idea that persistence is a natural part of Java, just as any other Api. And as soon one has the hang of working with the Mongo shell and starting the Mongo daemon, you start to realise that the non-relational concept is a very modern one, and one that will persist(;)).

<u>2.Admin App description</u>

The Patient app and the admin app follow a similar structure, although there are differences.

- Activities


20:47

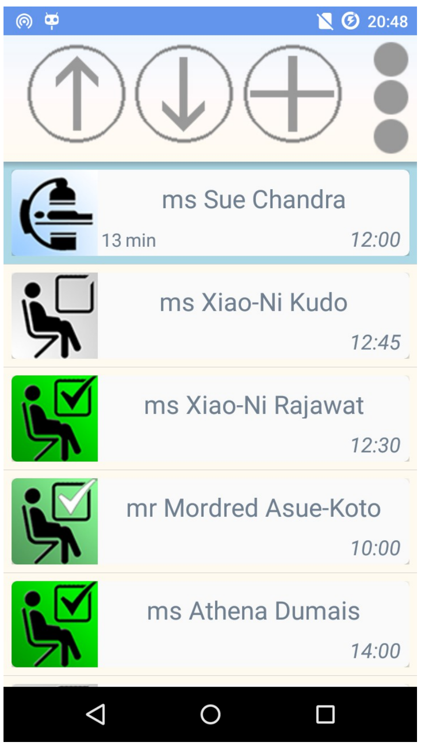
ms Sue Chandra 12:00

ms Xiao-Ni Kudo 12:45

ms Xiao-Ni Rajawat 12:30

mr Mordred Asue-Koto 10:00

ms Athena Dumais 14:00


20:48

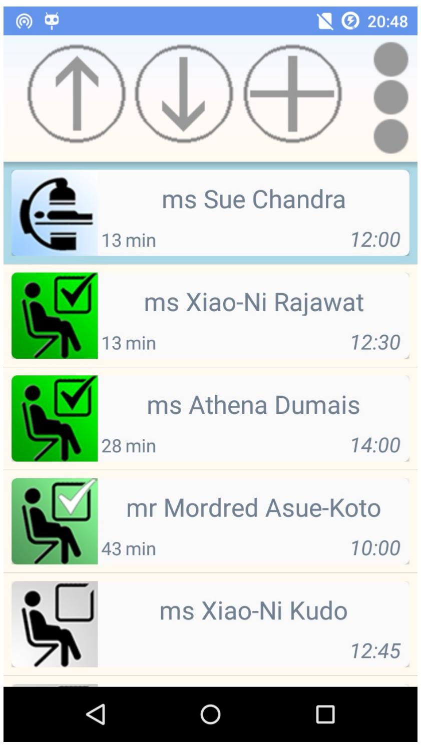
ms Sue Chandra 13 min 12:00

ms Xiao-Ni Kudo 12:45

ms Xiao-Ni Rajawat 12:30

mr Mordred Asue-Koto 10:00

ms Athena Dumais 14:00


20:48

ms Sue Chandra 13 min 12:00

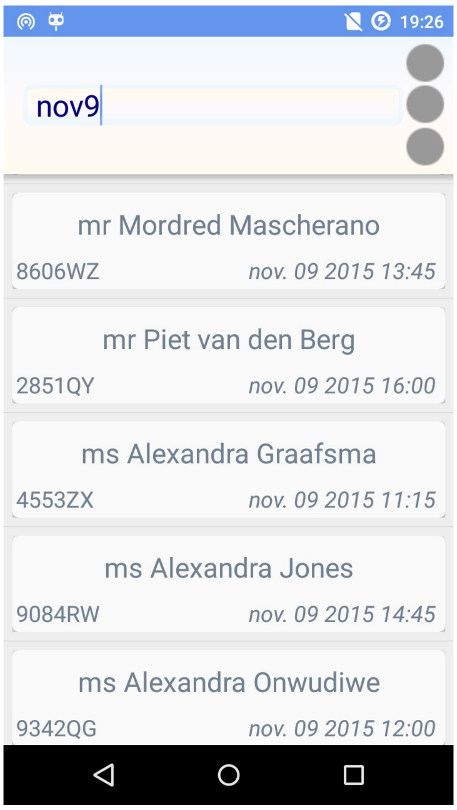
ms Xiao-Ni Rajawat 13 min 12:30

ms Athena Dumais 28 min 14:00

mr Mordred Asue-Koto 43 min 10:00

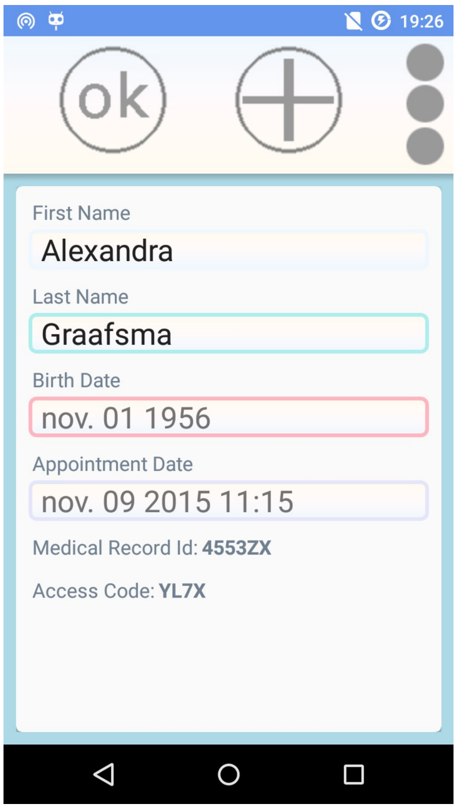
ms Xiao-Ni Kudo 12:45

➢ **SessionViewActivity** : The core of the impatient application. It contains the list of Patients to be treated for this day, and a set of controls to navigate through it. The list is backed by an ArrayAdapter which holds Patient objects. This Activity has a SessionViewOps object that functions as a guard for state that has to be maintained, as well as workhorse for operations that have not much to do with Views that have to be displayed, like setting up the aidl service connection or handling its callbacks. The main logic in this activity is about presenting the user a View that is consistent with the underlying data. For example a selector that highlights the current item, or handling clicks on the Patient icon. The list allows insert of Patients that are chosen from the list of all the patients. The Ops object refreshes the list every ten seconds with data it receives from the PollingService.


nov9

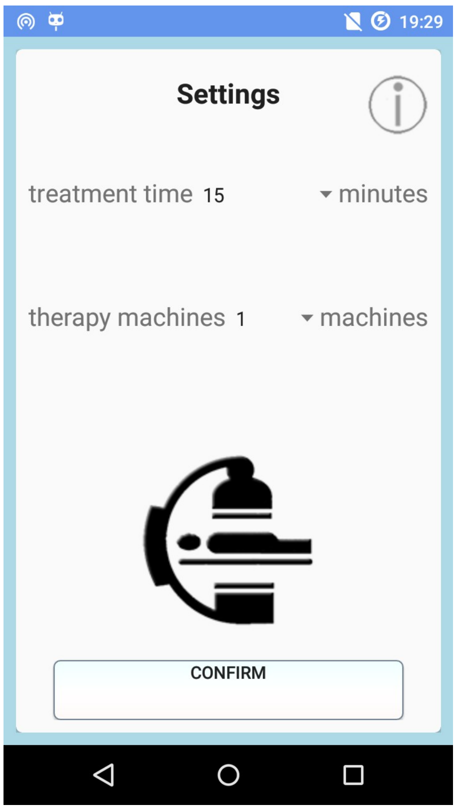
mr Mordred Mascherano	
8606WZ	*nov. 09 2015 13:45*
mr Piet van den Berg	
2851QY	*nov. 09 2015 16:00*
ms Alexandra Graafsma	
4553ZX	*nov. 09 2015 11:15*
ms Alexandra Jones	
9084RW	*nov. 09 2015 14:45*
ms Alexandra Onwudiwe	
9342QG	*nov. 09 2015 12:00*

- **PatientListActivity** : Presents the complete list of patients that is stored in the repository. It is used to insert patients into the running session, to lookup individual patients, or to review upcoming sessions. When this activity is started, the servlet is called for a list, which is temporary stored inside a ContentProvider. A LoaderManager and a CursorAdapter cooperate to retrieve these data from the Content Provider. This list now can be quite huge and it can be a real worry to retrieve a particular Patient. Therefore a search bar has been added to the activity, which cooperates with the CursorAdapter to immediately return search results. When the activity is done all the data will be deleted because no private data should persist on an individual device. Here also there is a PatientListOps object that takes care of the heavy lifting.

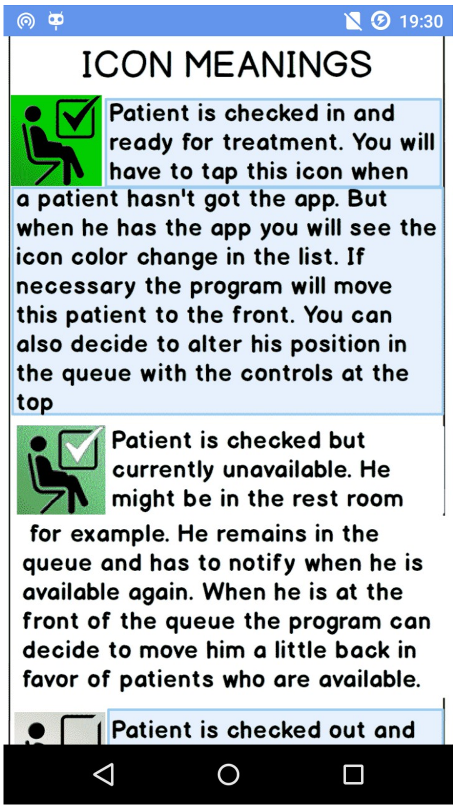


First Name: Alexandra
Last Name: Graafsma
Birth Date: nov. 01 1956
Appointment Date: nov. 09 2015 11:15
Medical Record Id: **4553ZX**
Access Code: **YL7X**

- **PatientActivity** : Used to view data of an individual patient, to edit them, or to add a new Patient. A PatientOps takes care to store a current patient Object. DialogFragments are used to get dates and times from a user. A broadcastreceiver reports back to supply new user credentials if necessary.



- **SettingsActivity** : Used to change critical settings on the server side. The server holds a default treatment time of 15 minutes and 1 machine. This is surely different in other use cases. When these settings are changed the session has to be reset, which is a drawback when one is halfway through a session. So a TODO item here is to raise the privilege level to a super admin, to avoid collisions. For the time being it is nice to be able to mess around a little with those values.



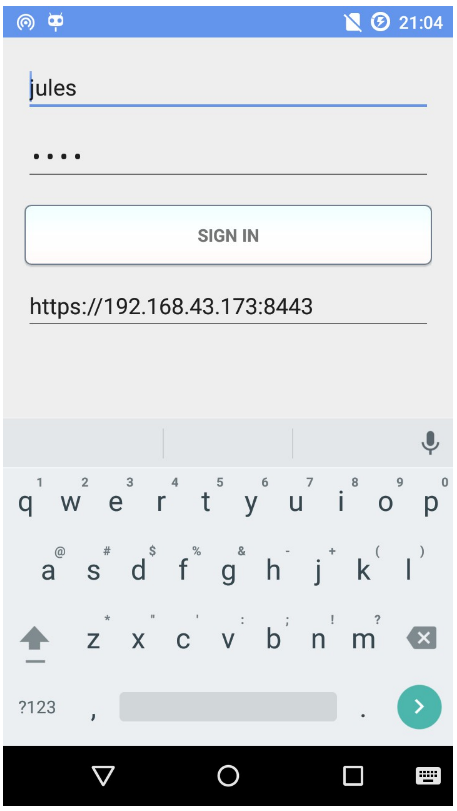
ICON MEANINGS

Patient is checked in and ready for treatment. You will have to tap this icon when a patient hasn't got the app. But when he has the app you will see the icon color change in the list. If necessary the program will move this patient to the front. You can also decide to alter his position in the queue with the controls at the top

Patient is checked but currently unavailable. He might be in the rest room for example. He remains in the queue and has to notify when he is available again. When he is at the front of the queue the program can decide to move him a little back in favor of patients who are available.

Patient is checked out and

- **HelpActivity** : In my opinion documentation is a must for each serious application. The logic of the developer often seems divergent from what users think. Each screen has a description that is made in an image editor. The format is a ViewPager that contains webpages. These webpages hold a single png image. Page transitions are done with a DepthPageTransformer. The Activity can only be exited via the back button, just like the SettingsActivity.



➢ **LoginActivity** : Used as the prime entrance to the application. Admins need to login again when they resume an application component. A BroadcastReceiver reports the succes or failure of the login attempt and acts accordingly.

- Services.

Three separate services have been made to handle the concurrent exchange of data. One reason for this is their logical differences, another is simply readablility of class structures.

➢ **LoginService** : an IntentService that makes a first call to the servlet. It will obtain a token that is stored in a static AdminSvc object after it has passed through user credentials. These credentials are not stored in the admin app for security reasons. But in the patient app it is safe to store them, because the division in apps will guarantee shielding of private data.

➢ **PollingService** : this service is used to constantly check at the server about important changes. This has to happen every 10 seconds. For this reason an aidl service is setup, that retrieves a Collection of Patients and passes that back to the Activity. Polling is done in a continuous loop. When the underlying activity pauses this thread is interrupted to prevent unwanted use of cpu time and resources. When the activity resumes the PollingService is rebound again.

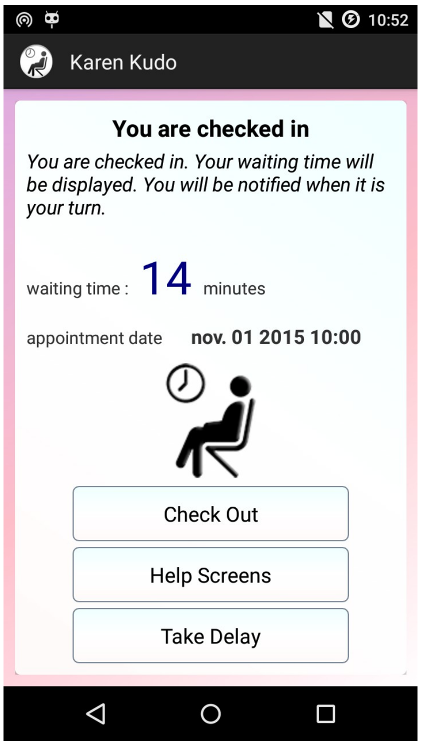
➢ **StatusService** : this is an IntentService which handles most of the requests that go to the servlet. It is triggered by an Intent that has a requestcode attached to it. This way the service knows which specific Api call to use. The onHandleIntent() method consists of a switch(requestcode) statement after which all the possible requestcodes are listed. When the servlet calls return the activity might be notified back via a BroadcastReceiver.

- **Content Provider**

The admin app has a feature in which the list of patients can be consulted. This can be a long list which would be impossible to navigate through. Therefore a content provider is used. The overall structure:

➢ **PatientProvider**: A core class that extends ContentProvider and does the actual CRUD operations
➢ **PatientDatabaseHelper** : Extends SqLiteOpenHelper and is used to create the table that holds the Patient data.
➢ **PatientContract** : Containing static Strings that hold Uri's, Authoritys, Paths, Columnames that are necessary to setup a sqlite db.
➢ **DataMediator** : A class that holds a ContentResolver that mediates between Activity and Content Provider. Methods to make ContentValues reside here, as well as a method that generates a String of all the values that are relevant to a certain patient. This String is used by the CursorLoader for a quick search through the PatientList

3.Patient App description


10:52

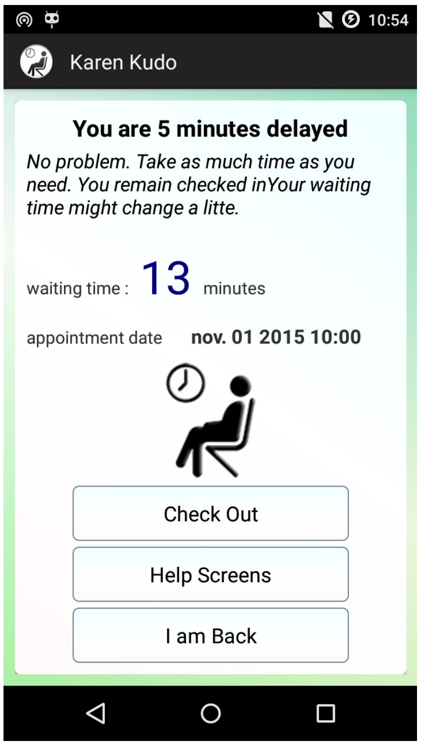
Karen Kudo

You are checked in

You are checked in. Your waiting time will be displayed. You will be notified when it is your turn.

waiting time : 14 minutes

appointment date **nov. 01 2015 10:00**


10:54

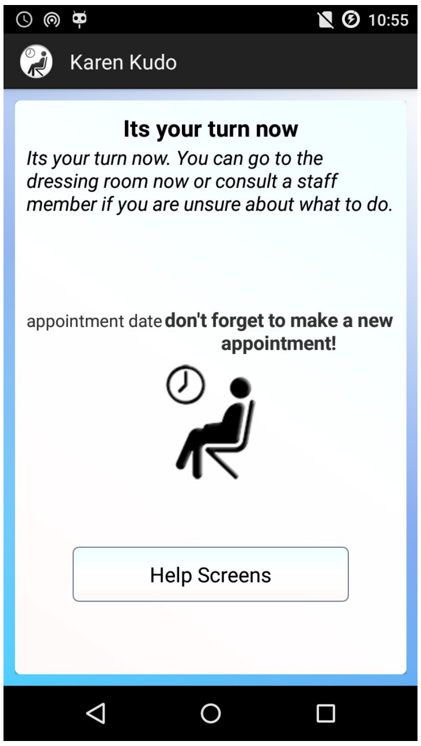
Karen Kudo

You are 5 minutes delayed

No problem. Take as much time as you need. You remain checked inYour waiting time might change a litte.

waiting time : 13 minutes

appointment date **nov. 01 2015 10:00**


10:55

Karen Kudo

Its your turn now

Its your turn now. You can go to the dressing room now or consult a staff member if you are unsure about what to do.

appointment date **don't forget to make a new appointment!**

The main difference with the Admin App is that this application is much smaller. It does not contain a Content Provider, and has only three instead of six Activities. However, the structure of three services is maintained, just for reasons of symmetry and reuse of code. So I refer to those sections for information on that, so I will focus here on the differences.
The patients app core Activity is PatientActivity who's main duty it is to display the Data that is relevant for a Patient at this particular moment. It relies on her PollingService to display the current waiting time and the PatientStatus. When the Patient is put under treatment a notification is sent that displays in the status bar at the top as a small clock. This notification will trigger the app if necessary.

- Animations

Both apps use animations but in the Patient App they are a bit more prominent.

- A TransitionDrawable is used to display the current status of a Patient. Each screen gets its own animated border, dependent on the ViewStatus. This is a little different then PatientStatus, because we have a state wherein the patient has an appointment but is unavailable, and a state wherein he has no appointment. This TransitionDrawable consists of two gradient Drawables that flow over into each other. These drawables are all formulated in xml, and reside in the res/drawable folder. The transition has to be performed as a separate Message on the MessageQueue. This is done via the postDelayed(runnable) method from the Handler class, which allows for timing. When the ViewStatus changes a new animated border has to be launched, and the old one cancelled. For this reason the Observer pattern is used : When the ViewStatus changes (for example when the Patient checks in) the update() method of the corresponding Observer (PatientOps in this case) is called, and the View is updated accordingly.
- Both Admin and Patient app have an animation to transit from one Activity to the other. These are scale and alpha transitions that are formulated in xml files that lie in the res/anim folder.
- Both Help Activities have animated page transitions. These are formulated in code. In both cases a ZoomOutPageTransformer class is used, but also a DepthPageTransformer is available.

- Artwork

Maybe a small note on the artwork. The used icons are all adapted from existing free icons. The toolbar icons of the Admin app are made from scratch. This is all done in Photoshop. The pngs for the Help screens are made in Balsamiq.

4.Mapping of Project Requirements

Abbreviations :
SS = SpringBoot Servlet : package prefix afr.iterson.impatient
PA = patient app : package prefix afr.iterson.impatient_patient
AA = admin app : package prefix afr.iterson.impatient_admin
For readablility only the short package names are used, and only the last folder name and the class that is addressed.

Requirement	**To be found where**
1. *1. Apps must support multiple users via individual user accounts.*	**PA** & **AA** : .activities.LoginActivity , .services.LoginService **SS** : .auth.OAuth2Configuration, .auth.ImpatientUserDetails, .model.ImpatientUser
2. *2. At least one user facing operation must be available only to authenticated users.*	**SS** : .controller.AdminController , .controller.PatientController
3. *3.App implementations must comprise at least one instance of at least two of the following four fundamental Android components: Activity, BroadcastReceiver, Service, and ContentProvider.*	**PA** & **AA** : AndroidManifest.xml , local BroadcastReceivers in .activities.PatientListActivity,.activities.PatientActivity, .activities.LoginActivity
4.Apps must interact with at least one remotely-hosted Java Spring-based service over the network.	**PA** & **AA** : retrofit.AdminSvc and retrofit.PatientSvc
5. Apps must interact over the network via HTTP/HTTPS.	**PA** & **AA** : retrofit.AdminSvc and retrofit.PatientSvc
At *6. At runtime apps must allow users to navigate between at least three different user interface screens.*	**PA** : .activites.LoginActivity, .activites.PatientActivity, .activites.HelpActivity **AA** : .activites.LoginActivity, .activites.SessionViewActivity, .activites.PatientListActivity, .activites.PatientActivity, .activites.SettingsActivity, .activites.HelpActivity
7. 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.	**PA** : .activities.PatientActivity , .activities.PatientActivity : TransitionDrawable , ActivityTransition, ZoomOutPageTransformer **AA** : ActivityTransition and ZoomoutPageTransformer
8. Apps must support at least one operation that is performed off the UI Thread in one or more background Threads or a Thread pool.	**PA** : .activities.PatientActivity : line 196(intentservice), 204(intentservice), 252(handler) , .operations.PatientOps : line 214(handler), 240 (aidl call) **AA** : Various calls to .services.StatusService from LoginActivity, SessionViewActivity, PatientListActivity, SettingsActivity, and PatientActivity.
The Patient is the primary user of the mobile app. A Patient is a unit of data that contains the core set of identifying information about a patient including (but not necessarily	**AA** : .aidl.Patient **PA** : .aidl.Patient **SS** : .model.Patient

limited to) a first name, a last name, a date of birth, medical record number, and appointment time(s).	
The Patient will use the app to Check In when they arrive at the waiting room for the department. *The Check-In process will verify patient identity and appointment time and add the Patient to the Queue.*	**PA** : .activities.PatientActivity : Line 212 checkInOrOut(View v)
The Queue is a list of patients waiting to be seen. The app will update the Queue in first-in first-out (FIFO) order of arrival and message each Patient their expected waiting time based on their position in the Queue.	**AA** : .activities.SessionViewActivity **SS** : .model.WaitingQueue (see above.)
If the Patient needs additional time (to go to the bathroom, to get changed, etc.) they can delay their appointment time in 5 minute increments. The Queue is adjusted accordingly.	**PA** : .activities.PatientActivity : line 187 delayOrUndelay(View v)
When a Patient approaches the treatment area their identity is again verified. After 15 minutes, their treatment is finished, and they are removed from the queue.	This re-verification is done via staff. App provides functionalitiy to remove people from the queue.
An Admin is a user type of the app that includes receptionists, doctors, nurses, etc. The Admin should be able to view the waiting room Queue, mark a Patient as waiting, being seen, or seen (removed from queue), and adjust Patient wait times manually.	**AA** : login requires admin credentials; by tapping the icons in the app patients status is changed. Waiting times are manipulated by changing a patients order in the queue.