
MASTER THESIS

Mr.
Peter Aoun

**Real-Time Database for patient
records on a Smartphone
including RAW Medical Images**

2019

Faculty of **Applied Computer Sciences and
Biosciences**

MASTER THESIS

Real-Time Database for patient records on a Smartphone including RAW Medical Images

Author:

Peter Aoun

Study Programme:

Applied Mathematics in Digital Media

Seminar Group:

MA15w2-M

First Referee:

Prof. Dr. Kristan Schneider

Second Referee:

Prof. Dr. Marc Ritter

Mittweida, April 2019

Bibliographic Information

Aoun, Peter: Real-Time Database for patient records on a Smartphone including RAW Medical Images, 53 pages, 42 figures, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer Sciences and Biosciences

Master Thesis, 2019

Abstract

The application described in this thesis has been created, built and designed to help nurses or any medical personnel all around the world in being able to access a real-time database to store patient records like Patient Name, Patient ID, Patient Age and Date of Birth, and the Symptoms that the patient is experiencing. A real-time database is a live database where all changes made to it are reflected across all devices accessing it. This application will be beneficial especially in countries where access to a computer or medical equipment is not always possible. A phone is always ready use and at the reach of the hand, users of this application will always be able to access the data at any given time and place. We will be able to add a new patient or search for existing patients. In addition, this application allows us to take RAW medical images that can be used to identify anomalies in the blood sample. RAW images are important for this application because they're uncompressed, which means, they do not lose any quality or details. The users of this application are the medical personnel that will be taking care of the patients. These users will have to create a profile on the database in order to use the application, since their data, like user ID, will be used in order to control the behaviour of the data retrieved and stored. We will also discuss the current and future features of this application, as well as, the benefits of this application when it comes to the medical personnel, as well as patients. Finally, we will also go over the implementation of such application from a hardware perspective, as well as a software one.

I. Contents

Contents	I
List of Figures	II
List of Tables	1
1 Introduction.....	1
1.1 Background	1
1.2 The Structure of the Thesis	2
2 Preliminaries	5
2.1 Basic Concepts	5
2.1.1 Nurse/ User Creation	6
2.1.2 Patient Creation	9
2.1.3 Medical images capturing	12
3 Current Features.....	17
3.1 Existing Patients	17
3.2 Profile edit	19
3.3 Transfer of Ownership	21
3.4 Request a transfer of Ownership.....	25
3.4.1 On the user side	25
3.4.2 The super user's side	28
3.4.3 Request History	32
3.5 Settings.....	32
3.6 User Information	33
4 Features in development	35
4.1 Symptom management (DONE)	35
4.2 E-mail verification	36
4.3 Fingerprint Authentication	36
4.4 Sorting option for requests	37
4.5 Case by case management (DONE through versions)	37
4.6 Offline Mode	40

5	Implementations	43
5.1	Preliminaries	43
5.2	Software Implementations	43
5.2.1	Android Studio	44
5.2.2	JAVA	45
5.2.3	Firebase	45
5.2.4	Technical terminologies	46
5.2.5	Android	47
5.3	Hardware implementation	47
6	Benefits	49
6.1	Benefits for the patient	49
6.2	Benefits for the nurses	49
	Bibliography	51

II. List of Figures

2.1	Welcome page.	7
2.2	The three rounded widgets.	7
2.3	User creation interface	8
2.4	Directory of user "Peter Aoun"	8
2.5	Log in interface.	9
2.6	User area.	10
2.7	The four rounded widgets.	10
2.8	Patient Creation form.	11
2.9	The "List of available symptoms" for that user.	11
2.10	Patient update alert dialog.	12
2.11	No image added alert dialog.	12
2.12	RAW directory.	13
2.13	The implemented camera interface.	13
2.14	A taken image being uploaded.	13
2.15	Image preview once an image is clicked in the recycler view.	14
2.16	The image links saved under "Uploads" under the patient ID.	15
3.1	The patient search interface.	17
3.2	A look at the fetched patient profile.	19
3.3	A look at the fetched patient profile (scrolled down).	19
3.4	Patient information page. The "Edit" button is highlighted.	20
3.5	Choose from a list of available versions.	21
3.6	Choose from a list of available edit dates.	21
3.7	Ownership verification algorithm.	22
3.8	Ownership transfer button.	24
3.9	The transfer interface.	24
3.10	"Transfer of Ownership" approach.	24
3.11	Transfer request button.	26
3.12	"Request a transfer" approach.	26

3.13	super_user_request sub-directory	28
3.14	Display of the Settings area.	29
3.15	A badge notification for how many requests are waiting for the user.	29
3.16	Display of the Request management area.	30
3.17	A display of new requests.	30
3.18	Display of the "Night mode"/"Day mode" toggle.	32
3.19	User information area.	33
3.20	Display of how the ID is represented.	34
4.1	Symptom Management activity.	36
4.2	Symptom management activity with editable list.	36
4.3	A display of the case by case management area.	37
4.4	Cases linked to each others.	38
4.5	Cases linked to each others - Interface.	39
4.6	Offline patient management interface.	41

1 Introduction

1.1 Background

Smartphones are a critical asset in our daily lives. They are portable and almost always ready to be used, they provide real-time communication and on-the-go database storage. They are equipped with touch screens, which make it simpler for the users to input data faster and more efficiently. They are also devices that connect their users to the online world, with endless possibilities, like data storage, data retrieval, communication, information sharing, etc.

People are able to use their smartphone as a powerful tool in their daily activities. Communication can range between different shapes, like vocal or visual, or both. Data storage varies for example from storing personal data to storing professional data, data for personal use and data on a bigger scale. A smartphone can store a large number of data about the users and their locations. It can enable real-time and asynchronous exchanges with other users via social networks and other forms of mobile communications. This feature makes it also possible for several users to access the same database at the same time and make changes to it, and then these changes would be visible to all users in real-time. This device includes features like: photography, location and other sensors, and wireless data service.

Those features are really important to us in this paper because they are the reason this application came to life. We will discuss them more in detail later to emphasize the importance of each feature.

Smartphones are offering a lot of opportunities in providing medical support when needed and wherever that may be. Large numbers and varieties of medical and health-related apps exist today. A 2012 estimate puts the number of health-related apps at no fewer than 40,000 [3]. There are apps that would send text message reminders to apply sunscreen when out in the sun, or apps that would help in managing diabetes, mobile apps are getting smarter and better as time goes by.

Mobile phones have several advantages in the medical field, especially when the mobile phone is the actual tool used in order to make this field simpler, more efficient and accessible by everyone, everywhere. Mobile applications make use of these aspects of mobile technology in several ways, these applications take advantage of the features a smartphone provides and produce a software beneficial to the user. In this paper we will focus on a mobile application developed by me, which aims on helping nurses in their daily medical tasks.

It has also been proved that combining a globally accessible database, with mobile phone based imaging, will provide objective, secure, automated, data collection and reported results. This mixing of existing technologies could also help in the detection and elimination of malaria. [5] The application that I created will be used as an environment for medical personnel in order to manage all patients on the go, as well as, take medical images of the blood for a deep image analysis leading to the pinpoint of the issue in most cases.

According to World Health Organization reports, some three quarters of the world population does not have access to medical imaging. In addition, in developing countries, over 50% of medical equipment that is available is not being used because it is too sophisticated or in disrepair or because the health personnel are not trained to use it. [6] That is why, a simple, handy, accessible, affordable, and well organized mobile application is needed for medical personnel all around the world, especially in the third world countries. And, the application presented in this thesis offers just that. A well structured mobile application with several handy and useful features, along with the most important one, RAW image capturing.

This application has been programmed in a way that it will force all compatible devices to capture RAW images and upload them on the server. The code contains a check, whether the phone's camera is compatible or not, and then produce a result. In all cases, nowadays, medical personnel have access to the latest technologies when it comes to mobile devices, especially that in our current date and time, an advanced smartphone is well within the affordable range.

1.2 The Structure of the Thesis

The thesis is organized as follows. In Chapter 2 we will present the preliminaries which are needed to understand the thesis and the application. In Chapter 3 we will look at six existing main features in the current build of the application.

In Chapter 4 we look at the features that are currently in development and are on the way of being implemented, some of them are important features and others are just added positive features for the application.

Chapter 5 treats the different implementation aspects of the thesis, namely software implementations and hardware implementations. The software implementations section will dive into the technicalities behind this application, like the reason behind the choice of Android over other operating systems, and the choice of Android Studio as a development tool, and so on. The hardware implementation section will describe what was needed as hardware in order for this application to work best.

The last chapter describes the several benefits for both nurses and patients.

2 Preliminaries

The use of smartphones by health care professionals has transformed the way clinical practice is done. Since mobile devices have become common in the health care environment, many medical software applications have been developed. Several applications are now available to assist health care professionals with many important medical tasks, such as: information and time management; health record maintenance and access; communications and consulting; reference and information gathering; patient management and monitoring; clinical decision-making; and medical education and training [4].

2.1 Basic Concepts

We will list here some basic concepts which will be used in the thesis. A smartphone contains the following elements: a **camera**, a **Wi-fi** connection and a **Database**. These elements can work independently for their intended function or they can work together to create results beneficial to the user.

A **camera** is a built-in device that the user can use to take images through the touch screen of the smartphone. A smartphone camera may not be as powerful as a digital camera on so many levels, but smartphone cameras have reached a level where they can take images that are almost at the same level of so many digital cameras.

A phone's **camera** does offer a lot of flexibility when it comes to what happens to the image after it is taken. When the image is taken, the user is able to share the image directly online, store it on a database, edit it, add information to it, and so on.

The biggest challenge would be to take images as close to the ones taken by a digital camera as possible. In order to achieve that, we would need to use specific features of the built in camera that would allow us to bypass the compression of images before they are saved.

This brings us to **JPEG** vs **RAW** format. A **JPEG** format is a format used by default to save images on a smartphone, it is usually the result of a compressed image file that would lose a bit of its quality while maintaining a decent result. The **JPEG** format is called a lossy format because some image quality is lost during the compression process. The **JPEG** format uses a compression algorithm in order to make the size of the image smaller and easier to store on a smartphone storage. For example, a **JPEG** image may reduce an image's file size by more than 80%, with little noticeable effect [1].

A **RAW** format would give the complete, uncompressed and lossless data from the camera's sensor. Which means that the **RAW** image saved will be exactly the one collected by the camera's sensor, no compression will be made on it and no data will be lost from it. This is beneficial in our case because we will be able to capture **RAW** medical images, without data loss, which will make those images ready to be analyzed and processed with the full data. For example, if we take a **RAW** image of a blood sample, the image will be full of data, with the proper lighting and the proper color. That's exactly what we need in order to study the medical images captured during the patient data collection.

Wi-Fi/ Cellular connections play a very important role in connecting smartphones to the world. A **Wi-Fi/ Cellular** connection is needed in order to upload/download images online, also this connection is used as a bridge between the smartphone and the online, real-time database. An **Internet** connection is needed so that the smartphone is able to communicate with the database and read/write data on it. Without the ability to connect to the **Internet**, the smartphones would have very limited capabilities, especially also in the world of medicine where nurses would need to keep records of their patients in a real-time database.

The **Database** is very important in our case because it is the place where the data of the users and the patients will be stored. First of all, a new user will need to create a new profile on the database in order to be able to log in when needed. Already existing users will need to pass the authentication step in order to use their profile and check which patients they are managing. Then the users will need to store the data of the patients; for the new patients, nurses will need to create a new profile, and for existing patients, nurses will need to retrieve existing profiles.

2.1.1 Nurse/ User Creation

First of all, we will need the users to create a new profile so that later it can be used for authentication and records.

When the user opens the application, they will be welcomed with a page where they can choose to either continue as guests, log in or register.

For this section we will discuss how new users/ nurses would register a profile upon first time use.

When the application is launched, a "Welcome." message would be on the screen to welcome the user on the application, and a rounded widget will be visible in the middle of the screen. This widget will have a "+" sign in its center, and once triggered, three rounded widgets will animate out of this central rounded widget.

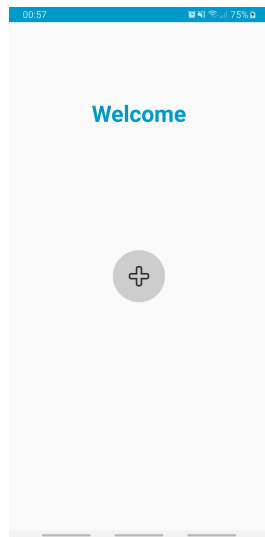


Figure 2.1: Welcome page.

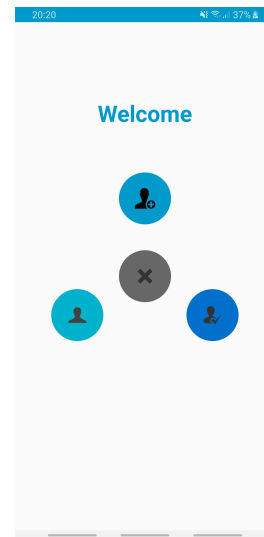


Figure 2.2: The three rounded widgets.

Three shades of blue rounded widget will be displayed, one to represent "Register User", another represents "Log In", and the last rounded widget represents "Guest User". For this section, we will assume that the user will press on the "Register User" widget.

When this widget is pressed, the user will be redirected to the "Register User" page. And there, the user will have to input the following:

- Name: It is required that this field is not empty.
- Family Name: It is required that this field is not empty.
- E-mail address: It is required that a valid e-mail address is entered, and that it does not exist on the database.
- Password: It is required that this field is not empty, that the password length is greater than 6 character, and that is hard (Consists of a minimum of 1 UpperCase, 1 LowerCase, and a digit).
- Confirm Password: It is required that this field matches the "Password" field.

Figure 2.3: User creation interface

Once the User clicks on "Register", a new directory will be created in the Database, this directory will be of the form:

```

master-thesis-78faf
├── OldPatients
├── Patients
├── Symptoms
└── Users
    ├── R66Z5We39ZaCLimv9YA9XYaCyyh1
    └── k98xhfv5djZpiGjF1utmt9L7a1Q2
        ├── NightMode: "true"
        ├── Symptoms
        ├── email: "peteraoun2@hotmail.com"
        ├── last_name: "Aoun"
        ├── name: "Peter"
        ├── patients_created: 142
        └── requests
    
```

Figure 2.4: Directory of user "Peter Aoun"

Each UserID will be created automatically by the database in a random way and it will be made of 28 elements. Each element is randomly picked from a set made of:

- Digits: Between 1 and 9.
- Lowercase characters: From 'a' to 'z'.
- Uppercase characters: From 'A' to 'Z'.

The final combination will form the unique UserID of each user. In addition, a child called "patients_created" will be added to this UserID directory, this child will keep track of all the patients that this user will create along the way.

2.1.2 Patient Creation

Our next point of discussion would be the creation of a new patient and how it would look like on the Database.

Once the user has created a profile, or if the user already owns a profile, they can log in to the application using the log in interface. In order to log in, a user will need their e-mail address and the password they have chosen earlier.

In the same interface, a switch button has been added in case the user made a mistake and would like to directly switch to the registration page. Also, a button is found at the left bottom of the page that would redirect the user to the 3 widget selection page where the user can again choose whether they would like to access the log in page, the registration page or the guest feature.

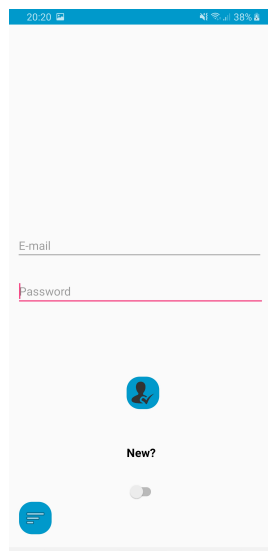


Figure 2.5: Log in interface.

If the log in is successful, the user will receive a toast message saying "Successfully signed in with —@—.—" . If the log in is not successful, the user will receive an error message.

Once the user is logged in, they will be redirected to the user activity interface with their name and family name fetched and displayed in the middle of the screen. Under their name, there will be a clickable widget that once clicked, will display four rounded

buttons. One button would redirect the user to **Patient creation**, the second button would redirect the user to **Existing patients**, the third button would **sign out** the user and redirect them to the **Welcome** screen, and the fourth would redirect the user to the **Settings** page where they can access some extra features.

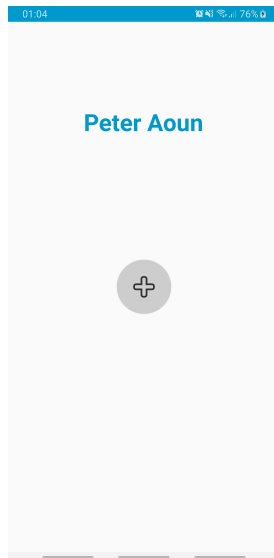


Figure 2.6: User area.

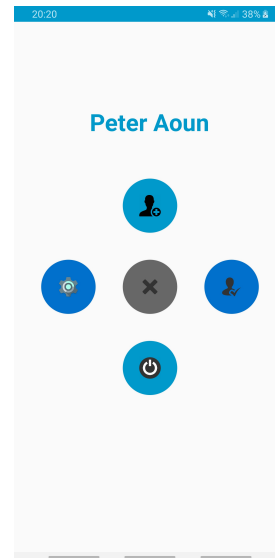


Figure 2.7: The four rounded widgets.

The "Patient creation" interface is where the user gets to input the data concerning the patient. This data consists of the following information:

- Name: The name of the patient is required. When the user chooses to avoid this field or forgets to input the information, an error message will be displayed informing the user that this information is necessary. This field is coded to force the first character of each word to be in uppercase like "Peter Aoun".
- Date of birth: The date of birth of the patient is required. When the user chooses to avoid this field or forgets to input the information, an error message will be displayed informing the user that this information is necessary. The age will be concluded from this information.

I coded this field so that it calculates the age based on the year, the month and the day. This means if today we are: 22/01/2019 and I was born on 24/01/1993, then $2019 - 1993 = 26$ years old. But technically, my birthday did not yet happen. The algorithm will also take into consideration the month and the day. So for this example, on the application, the displayed age will be "25" and in the case a patient was added with a date of birth on: 19/01/1993 then the displayed age would be "26".

- ID: Patient ID will be generated randomly, the ID will be created automatically by the implemented algorithm in a random way and it will be made of 28 elements.

Each element is randomly picked from a set made of:

- Digits: Between 1 and 9.
 - Lowercase characters: From 'a' to 'z'.
 - Uppercase characters: From 'A' to 'Z'.
- Symptoms: The symptoms experienced by the patient are required. When the user chooses to avoid this field or forgets to input the information, an error message will be displayed informing the user that this information is necessary. We need this information when the RAW medical images will be analyzed, and for patient records. A button is found next to this field, once that button "+" is clicked, an alert dialog will be displayed and a preset of pre-defined symptoms will be visible for the user to choose from. If the user already added custom symptoms, they will also be found in that list. This field has been coded in a way that every new line starts with "- " and that when a preset symptom is added, if the last line ends with a "- ", it will be appended with the first symptom added to that text box.
 - Medical images: There is a button at the bottom of the page that once clicked, would open a camera interface for the user to take RAW medical images. The button has a camera icon on it, which makes it self-explanatory for the users to know the purpose of such button. It won't redirect to the camera interface if the patient's necessary information are not filled. Also, once this button is pressed, the patient profile will be created and the database directory would be updated.

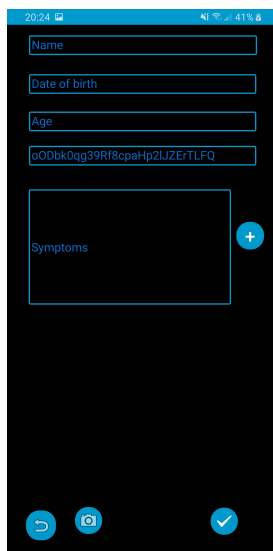


Figure 2.8: Patient Creation form.

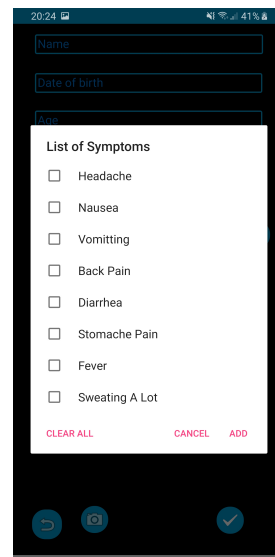


Figure 2.9: The "List of available symptoms" for that user.

A "Super User" field will be automatically added along with the patient information. The "Super User" will later be able to edit the patient information. This additional field will be

filled with the email address of the user creating the patient profile, as well as his first name and last name.

If the camera button is clicked, the patient will be created on the database, but first of course, the application will check if all the required fields are in order, and then once the patient created, the camera activity will open.

If the "Done" button is clicked before the camera activity was opened, a message will be displayed warning the user that no image was added and if they wish to continue anyway, and if the "Done" button is clicked after the camera was opened, an alert dialog will ask for a confirmation that the patient profile creation is done, and once confirmed, the patient will be added and the form refreshed for a new patient to be added.

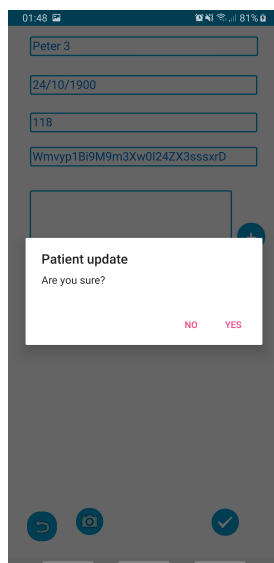


Figure 2.10: Patient update alert dialog.

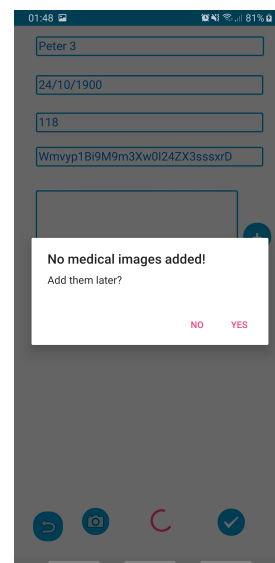


Figure 2.11: No image added alert dialog.

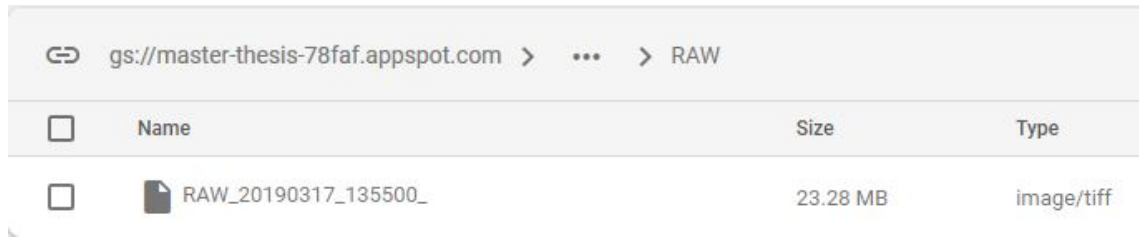
Once the patient is created, two subdirectories will also be added, "creationedit" to represent the time this patient profile was last edited, and "creationtime" to represent the time this patient profile was created.

2.1.3 Medical images capturing

As discussed earlier, a custom camera has been added to the application so that it captures RAW images. Once the user has been redirected to the camera interface, they will be able to see the name of the patient at the bottom of the screen, along with the ID. This information is merely there for cosmetic purposes.

The RAW image would be uploaded on the online storage under the directory "Photos" –> the user ID –> "RAW" –> the file name of the image will be the time

stamp that was added when the image was taken.



	Name	Size	Type
<input type="checkbox"/>	RAW_20190317_135500_	23.28 MB	image/tiff

Figure 2.12: RAW directory.

The RAW image will be uploaded automatically as soon as the user presses on the "Capture" button. It will take some time for each photo to be uploaded depending on the quality of the Internet connection since the RAW image is about 23 MB.

Compressing the image before uploading seemed like a bad idea since we want the uncompressed version of the picture and opting to compress it and then uncompress it might not guarantee a lossless conversion.

When a photo is being uploaded, the screen will be disabled, a message ("Uploading..") will be displayed, along with a rotating progress bar plus another progress bar in the background showing the percentage of the file that was already uploaded.



Figure 2.13: The implemented camera interface.

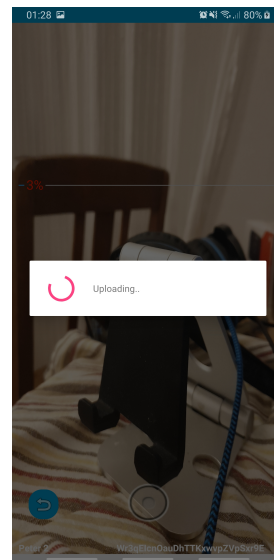


Figure 2.14: A taken image being uploaded.

On top of the screen, I implemented a scroll view where the thumbnails of the recently taken pictures would be displayed. The user can scroll right or left in order to see the available thumbnails. Once a thumbnail is pressed, a preview of that thumbnail would be displayed.



Figure 2.15: Image preview once an image is clicked in the recycler view.

A further explanation about what is displayed on the camera screen:

- A "Back" button at the left bottom of the screen. Once this button is pressed the user would go back to the "Patient Creation" page.
- An "X" button representing a delete function. If a user is not satisfied with the last photo taken, they can choose to delete it.
- The name and the ID of the patient are displayed at the bottom of the screen.
- The "Capture" button at the middle bottom of the screen. This button will be used to take photos.

When a user fills the patient information and then presses on the "Camera" button, as mentioned earlier, the patient profile will be created, but since the user has not yet finished setting up the patient profile, we will need to temporarily save the patient information on the device. As soon as the user presses on the "Camera" button, all the information about the patient will be assigned to different variables and then carried over in the form of a "Bundle" towards the next activity, the camera interface in that case.

Every time an image is taken, it will be assigned to a variable which will carry this image, in a "Bundle", to the previous activity ("Patient Creation" activity) in order to be displayed as a thumbnail at the bottom of the page. The thumbnails would be clickable so that the user can check the resulting images. Once the user is done taking the medical images, he/she will need to go back to the "Patient Creation" page.

Since the patient profile is created as soon as the "Camera" button is clicked, when the user finishes taking the medical images, they won't be **uploading** the profile, the profile will already be created and then the images will be added one by one as the user captures more and more images.

The thumbnails displayed on the camera interface will be the ones found in the directory of that specific patient.

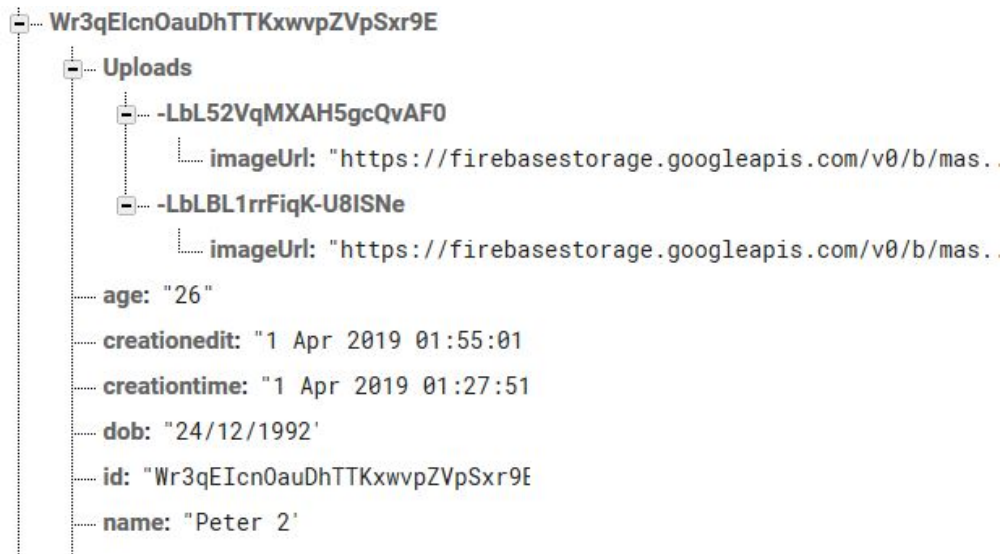


Figure 2.16: The image links saved under "Uploads" under the patient ID.

3 Current Features

In this chapter we will present the different features found in this application. Those features will make it easier for users to manage the patients, and they will offer more possibilities for the nurses.

3.1 Existing Patients

Here, the users will be able to browse through the existing patient profiles found on the database. Each user will be able to see all the patient profiles, regardless whether they are the ones who created them or not. The patients will be fetched from the database directory **Patients**.

In order for me to display these patient profiles, I had to initialize a recycler view which will be holding all the fetched profiles. And then I initialized a query pointing to the database reference **Patients**, sorted by child **name** (This is used to sort the results by **patient name**)

I then initialized a Firebase recycler using the initialized query as a parameter. Then I created an adapter that will use this Firebase recycler in order to inflate the recycler view with the patients' profiles.

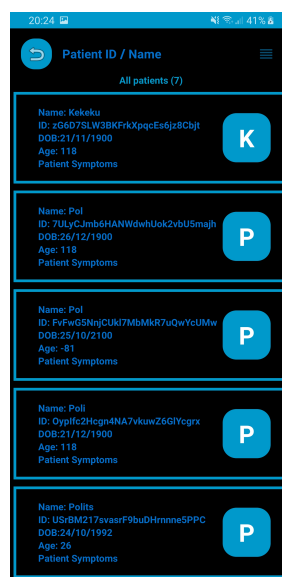


Figure 3.1: The patient search interface.

As shown in the figure above, the fetched information will be the following:

- **Name** of the patient.
- **ID** of the patient.
- **Date of birth** of the patient.
- **Age** of the patient.
- Initial letter of the patient's first name.

The symptoms will not be fetched as they don't need to be displayed during the patient search. Though they will be displayed once a patient is selected, of course.

Each patient profile is clickable and displayed in a box along with his/her first name's initials, which will separate them from each others and help us distinguish the patients.

Also, as seen in figure 3.1 above, there is a search box that the nurses can use in order to search for a patient by **name** or **ID**. The nurses won't need to do more than just start typing and the fetched results will be displayed. For example, if the nurse starts typing in order to get "AOUN", when she/he types the letter 'A', they will get all patients with 'A' in their names. Then "AO" will fetch patients with "AO" in their names, etc.

Once the nurse finds the patient they are looking for, they can simply click on that patient in order to have her/his information displayed. In addition, users can scroll through the patients' profiles in order to browse through the patients, but having the ability to search for a specific patient, makes the operation smoother and faster.

As an added feature, I have implemented a toggle (Seen in the figure, next to the search box) that the user can switch, and the choices will be:

- Patients managed by me.
- Patients managed by others.

When the user chooses **Patients managed by me**, the holder view will filter out only the patients that the current user is managing (The current user is their super user). On the other hand, when the user chooses **Patients managed by others**, the holder view will filter out only the patients that the current user is not managing (The current user is not their super user).

This small feature will give a better visual representation to the user in cases like:

- They would like to see the patients they are managing and go through them to make sure that the profile of each patient is up to date.
- They would like to see the patients they are managing for better managerial purposes and better organization.
- They would like to see the patients that other people are managing in order to request a transfer of ownership of one of the patients.
- They would like to see the patients that other people are managing in order to perhaps monitor how things are being handled by different nurses.

3.2 Profile edit

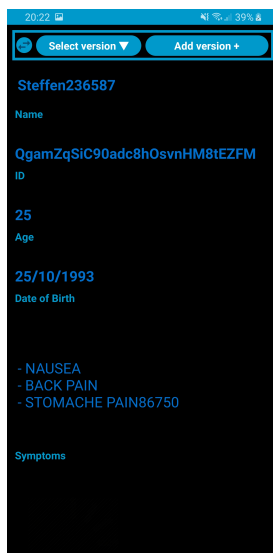


Figure 3.2: A look at the fetched patient profile.



Figure 3.3: A look at the fetched patient profile (scrolled down).

After the nurse has created a patient profile, they will be able to edit that profile and make some changes regarding the patient information saved on the database.

One thing to remember about editing a patient profile would be that only the super user of that profile can edit it. A super user of the profile is the user that created this profile. There is an "Edit" button at the bottom of the page, this button will only be enabled for the super user of that profile.

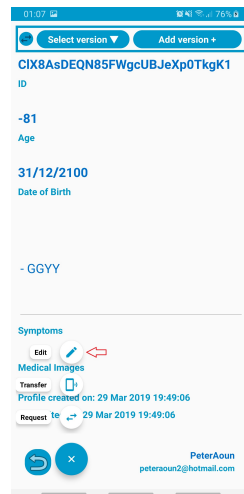


Figure 3.4: Patient information page. The "Edit" button is highlighted.

After the "Edit" button is clicked, and the algorithm verifies that this is the super user, this user will have the option to edit the following:

- Update the age after the birthday of the patient.
- Update the name of the patient.
- Add more symptoms experienced by the patient on a different day. The user will not be able to delete old symptoms.
- Add more medical images for this patient, or delete existing medical images.

Once a change is saved by the user, the new profile will replace the previous one found in the "Patients" directory and the old one will be added to the directory "OldPatients" under "Version" and the version number will be decided based on how many versions already exist under "OldPatients" for that patient. When the versions are displayed, it will be the number of versions found in "OldPatients" + 1, because we have to count the versions in "OldPatients" plus the latest version found in "Patients".

When the profile is saved, a time stamp of the current date and time will be added to the new value under "creationedit". That way, we will be able to keep log of all the changes that happened on the system and track all the changes made by the users. On the database level:

- If the age is updated by the user, it will be updated and replaced on the database.
- If the symptoms are updated by the user, the new symptoms will be added to the same directory as the old symptoms, update them, and then the old symptoms will be found in the older version. That way, when the symptoms are fetched to be

displayed as patient information, the user will see all symptoms from the oldest to the newest.

- If a medical image is deleted, it will be deleted from the database and thus, permanently deleted. If a medical image is added, it will be added to the existing images on their specific directory.

When a user is done updating the information, they will need to press on the "Update" button in order to trigger the changes. If the user tries to click on any button while a profile is still being updated, an error message will be displayed asking the user to finish editing the profile first.

If "Yes" is chosen, the database will be updated instantly, and if "No" is chosen, then all the changes will be cleared out and the patient information will be displayed as it was before the unsaved changes.

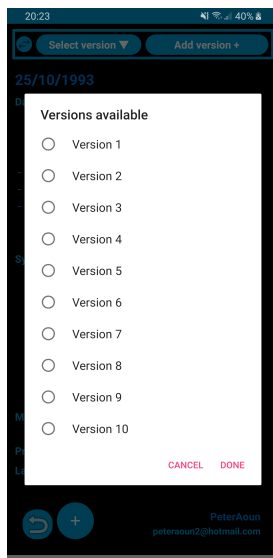


Figure 3.5: Choose from a list of available versions.

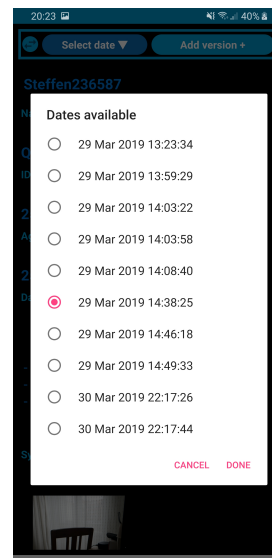


Figure 3.6: Choose from a list of available edit dates.

The user is able to switch by clicking on the switch button, between the ability to choose a specific version or to choose a specific edit date.

3.3 Transfer of Ownership

"Ownership" in this case means the ownership of the patient profile. When a patient's profile is created, the "Super user name" and "Super user email" values are added to the database as well. Those values correspond to the "First name + Last name" and the "E-mail address" of the user that created the patient profile. As mentioned earlier, only

the super user of a patient profile is able to edit that profile, and this is why we need a feature that helps the nurses transfer their ownership of a profile to another nurse.

This feature is useful because for example, when a patient has been transferred from one nurse to another, the new nurse will need to access and edit the information of that patient. The new nurse will be able to edit the information only if he/she is the super user of that patient.

As soon as the "Ownership" is transferred between nurses, the previous "owner" will not be able to edit the profile anymore and will receive a message saying "You are not the super user!" if they try to edit/ transfer "ownership" once more.

A sneak peek at the approach used to verify ownership:

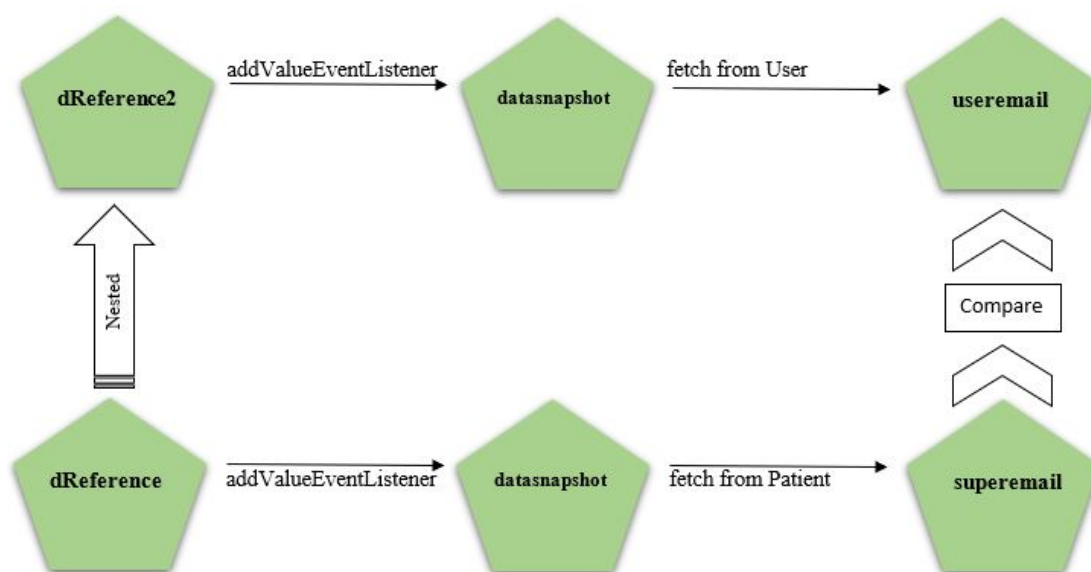


Figure 3.7: Ownership verification algorithm.

In this algorithm, we do the following:

- We initialize a database reference called "dReference2" as an instance of the current database's child, "Users". This means that "dReference2" will only contain the reference of the information that concerns the users on the database.
- We initialize "user_id" as the user ID of the current user logged into the system and making the operations.
- We then add a "addValueEventListener" function to "dReference2" which would be running at all times, so whatever code is written within "onDataChange" will be checked at all times.
- First thing that I did within this function is to initialize the "useremail" element. As shown in the algorithm, I took a datasnapshot of the current database reference. A datasnapshot consists of all the data found within this database reference. In the case of this database reference, the datasnapshot would be all the data concerning the users found on the database. I initialized the "useremail" to be equal to the child "email" of the child 'user_id' of the datasnapshot. This means, I fetched the e-mail address of the current user and assigned the retrieved value to the final fixed element "useremail".
- I then added a "addValueEventListener" function to "dReference". "dReference" was already initialized as an instance of the current database's child, "Patients". This means that "dReference" will only contain the reference of the information that concerns the patients on the database.
- First thing that I did within this function is to initialize the "superemail" element. As shown in the algorithm, I took a datasnapshot of the current database reference, in this case it would be "dReference". I initialized the "superemail" to be equal to the child "superemail" of the child 'id' of the datasnapshot. This means, I fetched the e-mail address of the current patient's super user and assigned the retrieved value to the element "superemail".
- I compared the values of "useremail" and "superemail".
- If "useremail" and "superemail" are identical regardless of the case (Uppercase/Lowercase), then the "Transfer" button will be enabled.
- If they are not identical, then when the "Transfer" button is clicked, the user will receive a toast message of the form "You are not the super user! "

Assuming the current user is indeed the super user and they are able to click on the "Transfer" button.

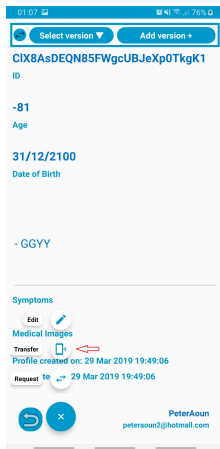


Figure 3.8: Ownership transfer button.

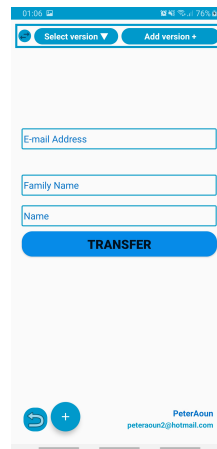


Figure 3.9: The transfer interface.

The super user will then need to input the name, last name, and e-mail address of the recipient user. The name and last name are required because they guarantee that the user knows to which person they are transferring the ownership to, and in order to verify that they indeed know this information, they will need to input the correct values in the required fields.

I used the following approach:

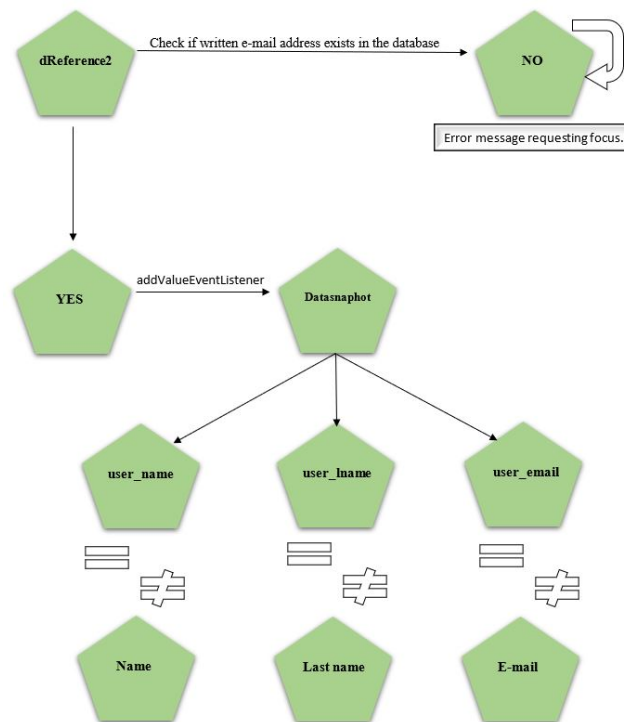


Figure 3.10: "Transfer of Ownership" approach.

- First, I took a specific portion of **dReference2**, this portion will be the child **email** equal to the e-mail address the user has entered as the recipient user e-mail.
- Second, I checked if this portion is empty. If it is empty then it means that the e-mail address does not exist, and the user will receive an error message requesting focus on the e-mail address field with the following message "User e-mail not found."
- Third, and after making sure the e-mail address exists in the database, I fetched the **datasnapshot**'s value and took a specific sub-string out of it. This sub-string will represent the user ID that holds the recipient's e-mail address.
- I then added the **addValueEventListener** to **dReference2**, and used the fetched user ID in order to get the **user_name**, **user_lname** and **user_email** of that user.
- Finally, I made sure that those values are all identical to name, last name and e-mail (Fetched from user input). Any inequality will result with an error message and a focus request.

3.4 Request a transfer of Ownership

Any user can request a transfer of ownership as long as they are not the actual super user of the selected patient. So first of all, before jumping into the details of this feature, there exists a check that makes sure that the user wishing a transfer of ownership is not actually the super user.

This feature will be used by users who would like to request a transfer of ownership of a certain patient profile, so that they are able to edit it. It can be the case of when a patient is transferred over to a new nurse and this new nurse needs access to the patient profile as soon as possible. Or when something happens and a new nurse needs to step in and edit the profile of a certain user.

3.4.1 On the user side

In this feature, and as mentioned earlier, the first thing that is checked by the algorithm, would be whether the current user is the super user or not. If the current user is the super user, then the button "Request" will be disabled, and whenever the super user clicks on that button, they will receive a message of the form "You are the super user!". This message will be informing the user that he/she is the super user and they don't need to request for a transfer of ownership.

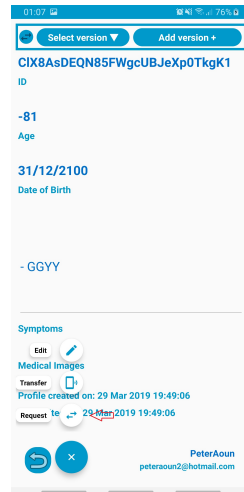


Figure 3.11: Transfer request button.

If the user is not the super user, another check will be triggered. This time, the algorithm will check whether this user already requested a transfer of that specific patient. If yes, the user already requested a transfer for that patient, he/she will receive a message of the form "Request already sent!" and no further requests will be sent to the super user. This is done in order to protect the super user from a spam of requests, a user is only allowed for one request for a specific patient at a time. If the user did not already send a request for that patient, then a new request will be sent to the super user.

The approach used for this feature is the following:

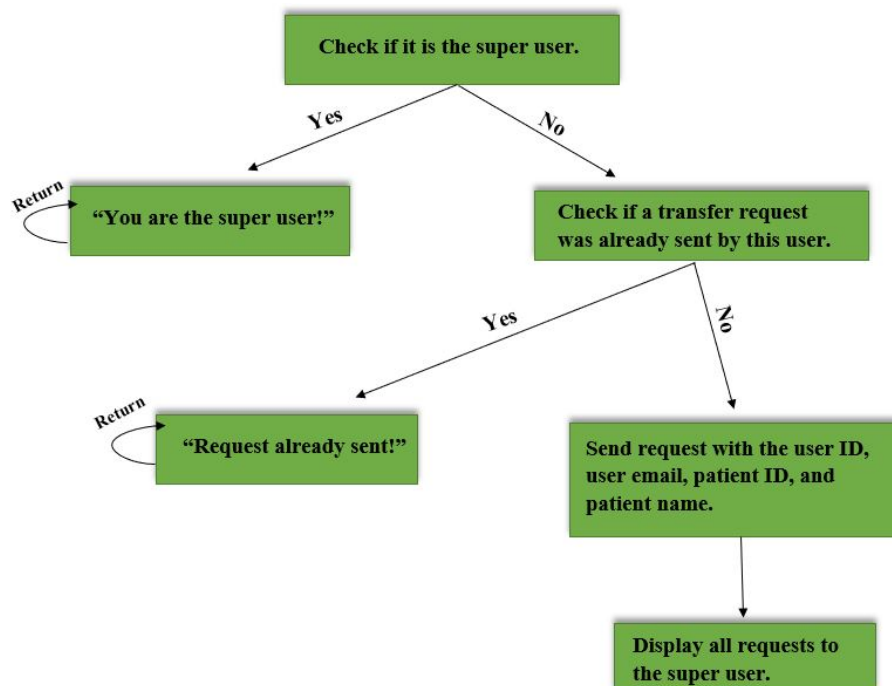


Figure 3.12: "Request a transfer" approach.

In the algorithm of this approach:

- I took **dReference2** (database reference pointing at **Users**), then I took a **datasnapshot** of the user that holds the **super_user_request**. This **datasnapshot** will help me obtain the **userID** of the super user of the current patient.
- I then added an **addListenerForSingleValueEvent** (differs from the other one in one aspect, this function stops at one value, the other keeps running several times through multiple values).
- Within this function, I fetched the e-mail address of the current user.
- I then initialized a database reference **dReference3** that points towards the child **super_user_request** (where the requests are found) of **dReference2**, and added a **addListenerForSingleValueEvent** to it.
- Then, I needed to check if that super user actually has at least one transfer request. To do so, I checked if the **datasnapshot** of **dReference3** exists. If it doesn't, then that means that the directory does not exist and the super user has no requests at the moment. We then need to create the first child directory for **super_user_request**, it would have value "1" since the **super_user_request** directory will parent children ranging from 1 to 20. Letting the **super_user_request** have children between 1 and 20 is better because it helps keep track of the number of requests and also, it helps limit the requests to 20. A super user is allowed to only have 20 requests at a time.
- If the **datasnapshot** exists, we sort the database reference **dReference3** by the **userID** of the current user. So in a way, we only take the data of users where this **userID** is found.
- Again, we first make sure that the **datasnapshot** exists. In this case, if it does exist, then it means that this user already made a request from this super user. We will then need to check if any of those requests were made towards the current patient profile.
- We compare the **datasnapshot** as a String and check if it contains the current **patientID**. Since this **datasnapshot** contains all the requests made by the current user, if this **datasnapshot** contains the current **patientID**, then that means that the current user already has a request pending towards that patient. If that is the case, the user will receive the message "Request already sent!" and no further requests will be sent to the super user.
- If the **datasnapshot** does not contain the current **patientID**, we create a child for **super_user_request** with the current **userID**, **user e-mail**, **patientID**, **patient name**, and **request number**.
- **Request number** is just a number that keeps track of the current request number. It is just a better visual representation for the super user.

- A check is implemented to make sure that once requests have been removed, the next child number assigned to a request will be the available number starting from **1**. So that means, whenever a request is deleted, or a request is added, the check will always make sure that the added request has a child number between 1 and 20, depending on which number is available. The **request number** of added requests will continue increasing. That means, the first request a super user receives after 60 closed requests will have child number **1** and **request number 61**.
- The user will receive a confirmation message "Request sent." once the request has been sent.

3.4.2 The super user's side

Things happen differently on the super user's side, many of the things that happen here may not be seen or noticed by the super user. That is why, I will discuss the magic that happens behind the scene to ensure that the super user is receiving the best results.

When a request has been made by a user for a transfer of ownership, the algorithm will either create or update the directory of the user that owns the patient profile.

If the directory does not exist, a new one will be created and if the directory exists, it will be updated with the information that was sent over through the user request.

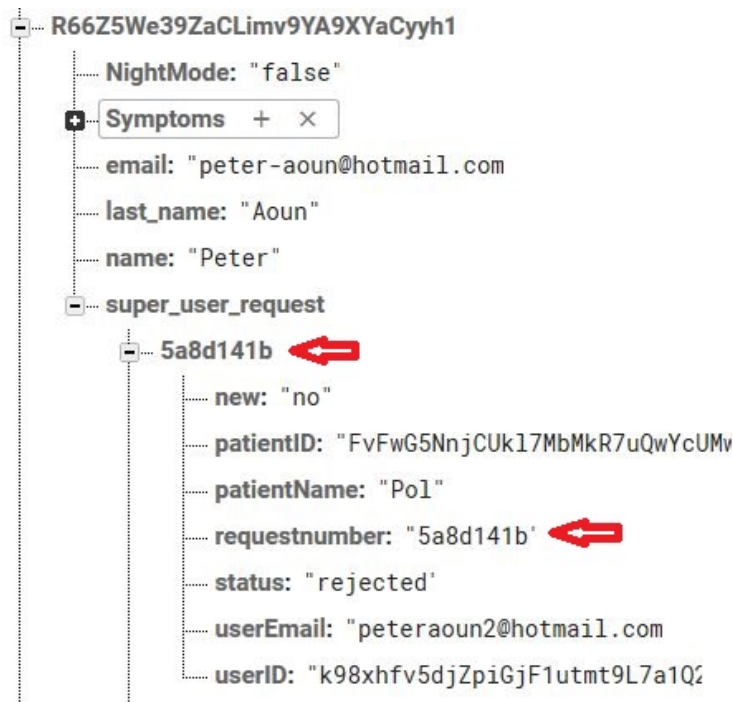


Figure 3.13: super_user_request sub-directory

When the super user launches the application, and after they get logged in, on the user area interface, they will have four choices. They will need to click on **Settings** in order to access the area where the extra features are found.

In this area there will be **five** buttons distributed as follows:

- User Information
- Symptoms Management
- Settings
- Requests Management
- Requests History

If the **user** has pending transfer of ownership requests, there will be a notification badge on top of the "Requests Management" icon indicating the number of pending requests the current user has. If no requests are found at that moment, the notification badge will be hidden and won't show a '0'.

The notification badge displays the number of children found in **super_user_requests**, which makes sense because the number of children found in that sub-directory is the actual number of requests the super user would have at that time.

The number in the notification badge has been programmed to change dynamically and reflect the current state of the **super_user_requests**.

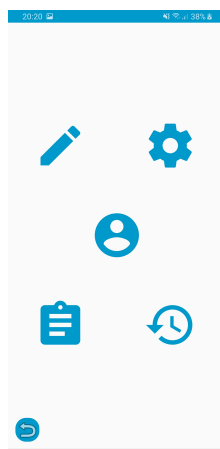


Figure 3.14: Display of the Settings area.

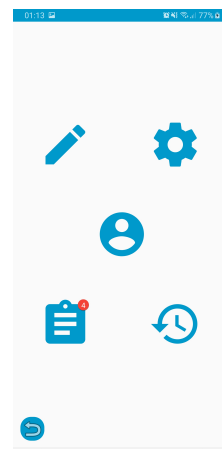


Figure 3.15: A badge notification for how many requests are waiting for the user.

Once the user clicks on "Requests Management", the process that displays the requests work as follows:

- We initialize a database reference **mDatabase2** pointing to the reference of the child **super_user_requests** of the current user.
- Initialize a query equal to **mDatabase2** sorted by the child **userEmail**, this is used to sort results by the e-mail address of the requesting users.
- Now we use that query in order to initialize a Firebase recycler.
- We use that recycler to initialize an adapter that would display the following information: **User name** (Sender of the request), **User e-mail**, **Patient name** (The patient whose profile is being requested), **Patient ID**, and **Request number**.

Once the information has been displayed, the super user will have two image views displayed next to each patient's information. One button represents a **Thumbs Down** and the other represents a **Thumbs Up**. Also, the user is able to minimize each category "Rejected", "Pending", and "Accepted", as well as, minimize each request independently by simply clicking on the blue area of that request.

In addition, if the pending request is new, a "NEW" message will be displayed and a counter of how many new requests will be seen. And once a request has been "seen" the counter will decrease and the request will stop being new. Plus, once a request is rejected or accepted, it will be moved to its corresponding category.



Figure 3.16: Display of the Request management area.

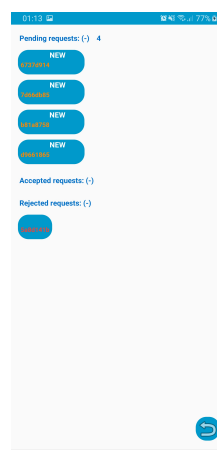


Figure 3.17: A display of new requests.

If the **Thumbs Down** has been clicked, that means that the super user has rejected the **Transfer request** for that user and will receive a confirmation message saying "Transfer of ownership rejected".

If the **Thumbs Up** was chosen, the ownership will be transferred to the user that made the request, and the super user will receive a message of the form "Transfer of owner-

ship confirmed". This will update the fields on the database to ensure that the accepted user is now indeed the new owner.

An issue arises in case the super user has several requests for the same patient. Assuming the super user accepted ownership transfer and no longer has ownership over a patient, if no check is implemented, that super user will still be able to accept/reject the requests for that patient's profile and that would affect the whole concept of ownership.

I have implemented a function that, after an ownership request has been accepted, would automatically reject all requests found concerning the same patient. This would guarantee a smooth operation and a correct definition of ownership.

A message will be displayed to the super user telling him/her the number of requests that got automatically rejected. The message will be of the form "Automatically rejected X requests for the same patient".

All the rejects and accepts will be updated on the database.

A check has also been implemented so that when a child of **super_user_request** is deleted, the next child will have either that number or the number available between 1 and 20. This has been implemented to make sure that the children stay organized under a maximum of 20 requests and for the ease of fetching these children. On the other side, the **request number** will never be the same and will keep increasing, as mentioned earlier. So when a request is rejected or accepted, the request number of the next requests will never match the number of the request that got handled, but the number of the request on the database might be the same as a request that got removed.

Another check has been implemented for another issue that may arise, and that is when the super user manually transfers ownership of a patient while having transfer requests for that patient. Since again, this would contradict the meaning of ownership, a super user might transfer ownership manually of a patient and still be able to change the ownership of that patient via the pending requests.

The algorithm will make sure to check if the super user has any pending transfer requests for that patient and then, give a warning message for the user telling him/her that transfer requests have been found for that patient and if they wish to continue.

If the user decides to continue, the algorithm will automatically reject all pending requests for that patient. Once that is done, the user will be informed of such action and of the number of requests that were rejected.

3.4.3 Request History

Every user will have a **request history**. It is used to keep record of the requests that have been requested, transferred, approved, or rejected.

A **request history** will display requests in the same fashion as the "Requests Management" form, requests will be sorted under "Pending", "Rejected", and "Accepted". The only difference would be that in this case, the "NEW" elements will be the rejected and accepted requests, while in "Requests Management", the "NEW" elements are the pending requests.

When a user requests ownership, his/her request will be displayed in the **requests history**. It will have one of these available statuses: **PENDING**, **ACCEPTED**, and **REJECTED**.

PENDING would be when the user is waiting for an answer. **ACCEPTED**, when the super user accepted the transfer of ownership. **REJECTED**, when the super user rejected the transfer of ownership.

All requests in the **request history** hold information about the targeted super user, the user that sent the request and the patient profile.

3.5 Settings

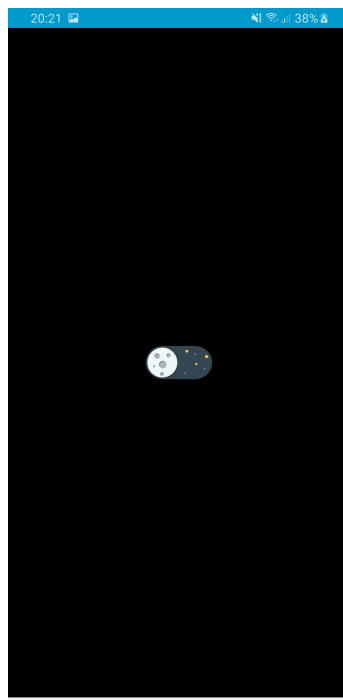


Figure 3.18: Display of the "Night mode"/"Day mode" toggle.

The **Settings** section is found from within the **Settings** interface. It is a section where the user is able to edit purely cosmetic behaviours. Some of these changes are as follows:

- If the user has an issue with white mode background and theme on the application, as it tends to hurt the eyes, they are able to switch to dark mode.

3.6 User Information

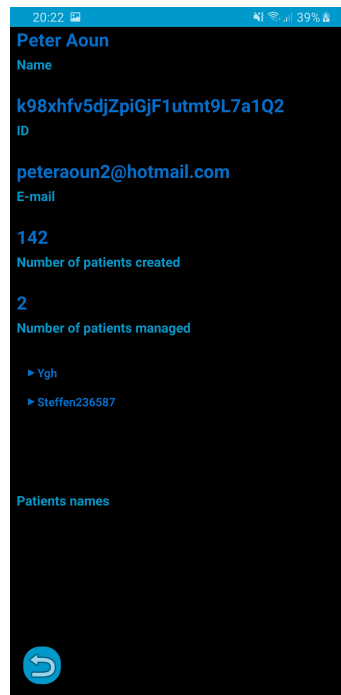


Figure 3.19: User information area.

Another section found in the **Settings** interface, the **User Information** section includes the usual user information, like e-mail address, first name, last name, and ID. Though, these information are not editable by the users and that is for security reasons.

In case the user would like to change the e-mail address, the name or the family name, this user will have to request the change from the database administrator.

An additional information found in that section, would be the number of patients this user created. No matter how many patients they transferred or received, this number is purely determined on how many patient profiles they actually created from 0. This number is static as it can only go up.

One last information found there, is the number of patients the current user is managing. This number represents purely the number of patients where the current user is

their super user. This number is dynamic as it can go up or down depending on the ownership.

The user will be able to click on the patient name and that will display the ID of that patient, and once the ID is clicked, the patient information of that patient will be displayed.

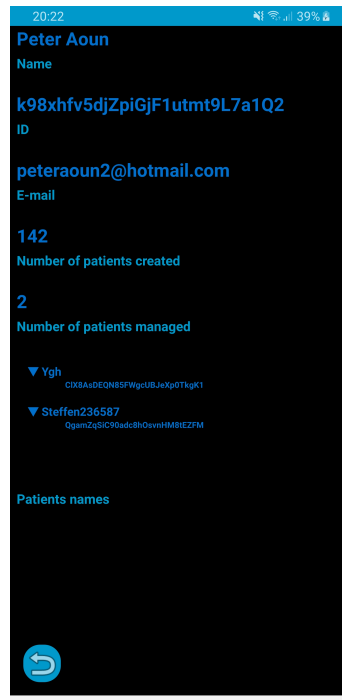


Figure 3.20: Display of how the ID is represented.

4 Features in development

In this chapter, I will discuss the features that are currently in development and will be released in the future.

They may be features that have been postponed for a future release, but that does not make them any less important. Contrarily, many of these features are very important but were not vital for the initial development of the application.

I will further discuss these features and elaborate the importance of each.

4.1 Symptom management (DONE)

This feature has been implemented for operation simplification, and to make the daily tasks easier and faster for the nurses.

There exists a preset of most used symptoms found under the directory "Symptoms" on the database, and the user will be able to choose one or many symptoms. This is much faster than the nurses having to enter the information manually, and this feature will greatly decrease the time needed for a nurse to create / edit a profile.

In addition to the already preset symptoms, each nurse will be able to create and edit their own presets. Each created symptom will only be displayed to the nurse that created it, and each added symptom will be added to the list of symptoms the nurse sees when adding symptoms to a patient profile.

The symptom's list is manageable from within the **Settings** interface, after the user clicks on the **Symptoms Management** button.

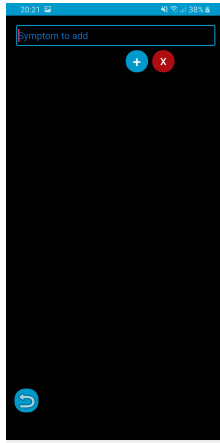


Figure 4.1: Symptom Management activity.

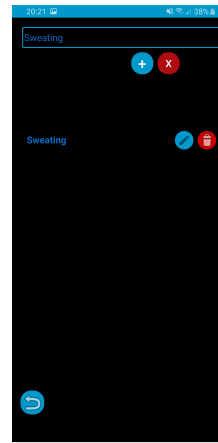


Figure 4.2: Symptom management activity with editable list.

4.2 E-mail verification

A small addition with a big importance. This feature will help the user make sure that the e-mail address used to register is correct, and that will be done by making the user verify his e-mail address via e-mail verification.

In addition, this feature will be useful in case the user forgets his password and would need to reset it via e-mail reset.

There will be a **Forgot password** button. When the user clicks that button, an e-mail will be sent to the user in order for him/her to reset the password via the instructions in the e-mail.

4.3 Fingerprint Authentication

Rather than having the traditional e-mail address + password log in, a user will have the option to opt for fingerprint authentication.

Once a user opts for this option, they will simply need to scan their fingerprint in order to quickly log in.

There will be, of course, an alternative log in method, in case fingerprint authentication is not possible at that time. Also, the user can enter several fingerprints, for further flexibility and smoothness.

This feature will be found under **settings**, where the user can decide whether to activate or disable this feature.

4.4 Sorting option for requests

This is a sorting option for requests found on the super user profile, and it will allow the user to sort requests by **request number** rather than the requester e-mail address.

This can be beneficial for nurses who would like to see which request came before the other.

4.5 Case by case management (DONE through versions)

This feature will allow the nurse to differentiate between different cases and have a good, organized overview of patient information and changes.

Once a nurse clicks on a patient profile, found in the **Existing Patient** interface, a page for case by case management will open, display something as the following:

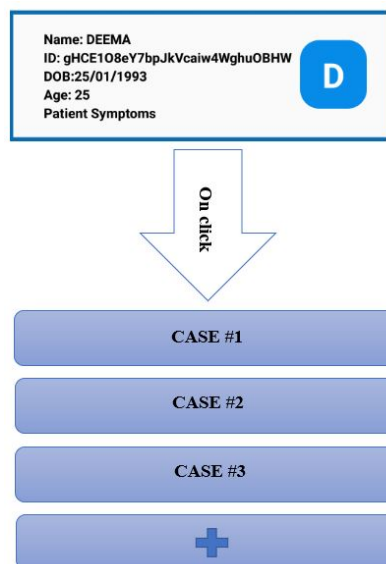


Figure 4.3: A display of the case by case management area.

When a user clicks on a **CASE**, a patient information page will be displayed. There will be a time stamp on that page to show the date this case was added, and the information on that page will be specific for that **CASE** on that date.

Each **CASE** will either have new information, or a more correct and accurate version of a previous case. I will make this feature in a way that, information in a specific **CASE** is not editable, only seen, but when a new case is added, there will be an option that allows the user to link the new case to an old one. Once cases are linked to each others, the link will be visible as follows:

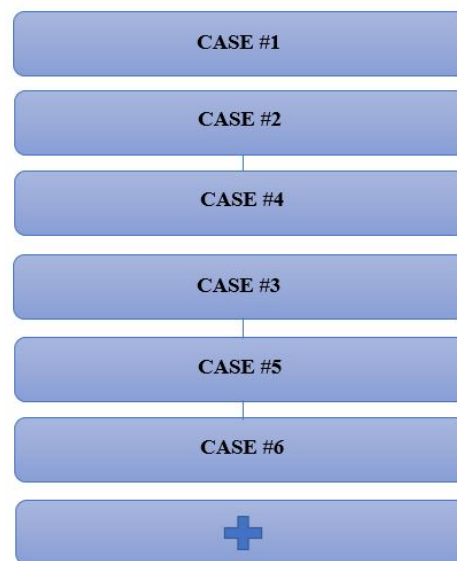


Figure 4.4: Cases linked to each others.

In addition, if we follow the example I illustrated in figure 4.2, if the user clicks on **CASE 2**, there will be an information at the bottom of the page indicating that an updated / newer version of this case is available and this newer case will be displayed for the user as follows:

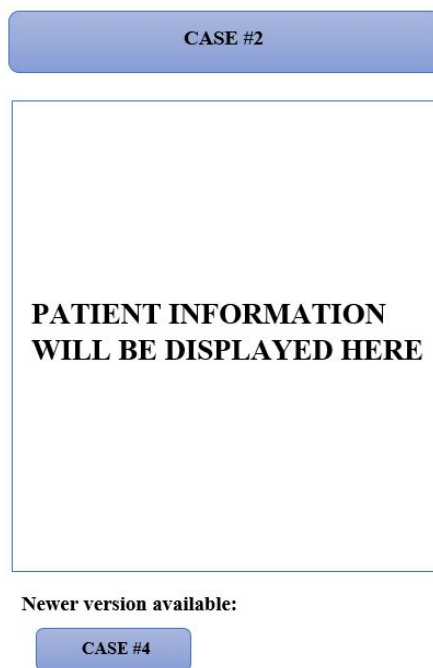


Figure 4.5: Cases linked to each others - Interface.

And the same thing will happen if the user clicks on **CASE 4**, the user will have an information at the bottom of the page indicating that an older version of this case is available, and in this example it will show **CASE 2** as the old version available.

The **+** button displayed in figure 4.1 and figure 4.2 represents the **Add a case** button. It is a button used to add a new case to the patient profile. Only the super user of that patient will be able to add cases, and the option to remove cases will not be granted to the users, in order to make sure that all changes remain for logging purposes.

A check will be implemented once the **case management** page is loaded, if the user is not the super user, the button **Add a case** will be disabled and once the user clicks on it, he/she will receive a message of the form "You are not the super user!". And if the user is the super user, the button will be enabled.

This feature will help the nurses distinguish the different cases of a patient and monitor the progress or the regress of that patient.

The cases will of course be sorted by creation date, as it makes it easier for the nurses to follow up on them, and also they will be sorted based on the existing case links.

If **CASE 3** was created before **CASE 4** but **CASE 4** was created and linked to **CASE 2**, then **CASE 4** will be displayed directly after **CASE 2** and then **CASE 3** will be displayed.

4.6 Offline Mode

It is a mode where the user does not need an Internet connection in order to take advantage of the application.

If the user wishes to manually access this mode, it is found by clicking on the black widget found in figure 2.2. Otherwise, there will be a check at the start of the application that will do the following:

- If no Internet connection is found, the offline (Guest) mode will automatically be triggered.
- If the user has internet and would like to access the offline mode, a message will pop up informing the user that they have Internet connection and they are about to connect to the offline mode of the application.

Once the user is in the **Offline mode**, only three options will be available, **Patient Creation**, **Patient Management**, and **Picture Mode**.

If **Patient Creation** is selected, the user will have access to the usual patient creation interface, and then the patient profiles will be stored locally as forms and will be accessible via the **Patient Management** interface.

If the **Picture Mode** interface is selected, the user will have to first enter the patient name, and then will be able to take images and they will be stored temporarily on the phone, under a folder named after the patient's name. Naming the folder as the patient name will make it easier for the user to access the images later on, and to know to whom these images belong.

When the user goes online, he/she will be able to see the patients they created and then upload them with the click of a button. Users will be able to choose to upload one patient, many patients, or all patient at a time.

Afterwards, once these profiles have been uploaded, the super user will be able to access these profiles and manually upload the images that belong to each new patient.

In addition, once a patient profile is uploaded, the information will automatically be deleted from the local storage, and the same thing will happen for the images.

This feature is very important because there is no guarantee that the nurses will be online at all times, and this application has to be ready to deliver at any time, no matter the connectivity.

Finally, if the **Patient Management** interface is selected, the user will be able to manage the local patient profiles and it will be as follows:

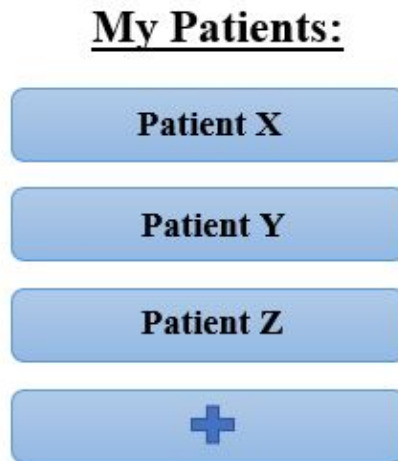


Figure 4.6: Offline patient management interface.

In this mode, each locally stored patient will be editable. The nurse managing patients found on the local database can edit

- The name.
- The date of birth.
- The symptoms.

Once the profile is uploaded on the database, the only information that can be edited would be **the symptoms**.

This will ensure clean online records, and will give the nurse the opportunity to correct any mistakes made offline. And the same applies for the images, the nurse can take as many images as he/she wants, and then upload whichever images they deem best.

5 Implementations

In this chapter, I will discuss the hardware and software requirements that made and would make this application feasible and usable.

I will discuss what will be needed as hardware component, and what was used and implemented as a software approach.

5.1 Preliminaries

We live in a world where there are a lot of hardware and software choices and options, and that makes it difficult for us to choose the best option. That is why, in order to pinpoint the best direction for me, I had to break down all the choices and options into smaller parts in order to take a decision and follow it throughout the thesis.

I had the option to choose either a database based on Linux, where I will need to create everything from scratch in order for the application to store the data, or to opt for Firebase which is supported by Google and has all the utilities included. I will discuss further in this chapter, why Firebase seemed like the best option for me.

In parallel, I had the option to choose out of several coding applications, yet I had to pick Android Studio. It is an Android coding software that offers **JAVA** as a programming language. I will also further discuss in this chapter why Android Studio stood out for me.

I have mentioned a lot of technical words in this thesis up until this point, and this is where I will break down each term and explain what it does and why it was used.

Finally, I will discuss why this application is solely developed for Android Devices and there are no plans, for the moment, to develop this application for other operating systems.

5.2 Software Implementations

In this section, I will dissect the software implementations and technicalities that were used, mentioned or needed in order to develop this application.

5.2.1 Android Studio

Out of all the available android development tools, Android Studio seemed to me as the most flexible, simple and easy to use tool available on the market today.

Based on a list of the top 20 android development tools, Android Studio is at the number one spot and has been there for a while. [2]

Android studio was released in 2013 by Google, which also, released the **Android** operating system back in 2007-2008. So using Android studio seemed like a great idea, since the targeted operating system was developed by the same company as the one that released the development tool.

This tool offers also a lot of features and benefits. These features are found on Android's official website (<https://developer.android.com/>), some of them are as follows:

- Fast coding and iteration of code. This development tool offers the option to **Instant Run** a code which pushes the changes made on the code to the running application.
- A lot of code templates and Github codes can be found in order to help the developer in his/her coding journey and to facilitate the coding process.
- There are tools to catch performance, usability, version compatibility, and other problems. The tool automatically checks for such issues and informs the user so that he/she can be aware of them.
- Built in support and features for Firebase, which is one of the reasons this database has been used in the development of this application.
- Intelligent code editor that helps the developer code faster. As the user types the code, the code editor will start suggesting codes to the user and once the user chooses a suggestion, it will automatically be implemented.
- It is the fastest tool today that allows the emulation of an application on an Android device.
- Unified for all Android devices, which means that building and coding an application using Android studio will allow this application to run on all Android devices.
- This tool has a UI editor that makes it easy to create, edit, and control the visual representations of the application. Views, text boxes, buttons, etc. can simply be added through a drag and drop motion, and then instantly edited either on screen by simply re-sizing the object for example, or through back-end coding.
- It also has a Vector Asset studio, and this allows the users to create there own image assets regardless of the density size. Icons to be used for visual representation can either be selected from a preset group or they can be imported from user selected files.

In conclusion, these features among some others made me choose Android Studio over other available tools on the market.

5.2.2 JAVA

There are plenty of programming languages today, from C++ to C to VB.net to JAVA, etc. And all of them have their own advantages and disadvantages, and I've had the chance to experience most, if not all of them. There are mainly three reasons why I chose JAVA over the others for my current application, and they are as follows:

- Android Studio, my number one choice for Android development, supports and encourages JAVA coding, as JAVA seems to run the best on Android devices.
- As a computer science graduate, JAVA has been my main programming language since the early days of this coding adventure. It was the first language I learned, and thus the language I'd like to focus on the most.
- As I've learned throughout my Bachelor studies, JAVA is very easy to use, compile, debug, etc., compared to other languages. Also that it is multi-threaded, meaning that it allows multiple tasks to run at the same time. Those are some of the features, among many others offered by JAVA

5.2.3 Firebase

As I mentioned earlier, I had the option between Firebase and another more traditional database structure, but I opted for Firebase. Based on the official website (<https://firebase.google.com/products/>), some of the many reasons I opted for Firebase are as follows:

- As mentioned earlier, Firebase features and options are implemented within Android Studio, which means that using Firebase on Android Studio was going to be easier, faster, and more efficient than implementing a more traditional database structure and having to create everything from zero.
- The authentication feature is very easy to implement, it provides authentication by email, password, id, username. That feature is essential in my application since each nurse will need to be authenticated before using the application. Firebase also allows authentication via Google accounts and Facebook, but that feature was not implemented in my application, but could be added later on.
- It allows the sending of notifications, and to customize them.
- Cloud Firestore, and that means that the data is stored and synced between the users at a real-time level, along with an offline support. That feature was very important for my application because live synchronization and offline availability are what it needs.

- It offers cloud storage which is used to store media files, and in my case, medical images. Cloud storage is also a very important feature as one of the key functionalities of my application is the RAW medical image capturing.
- The cost to scale up the database capacity for Firebase is very cheap and small compared to other solutions.
- Administrating the Firebase database is very easy to use, learn, and execute. The interface to manage the Firebase database is very informative and very user friendly, it also offers statistics and graphs on how much space was used per day, the number of the downloads made, the used storage space, etc.
- Data is available everywhere and on the go, and also, since the database will be updated by a lot of users, it is capable of handling the real-time data updates between devices.

In conclusion, Firebase seemed to be the most flexible, simple, efficient, cheap, and user friendly solution.

5.2.4 Technical terminologies

Throughout this thesis, some technical terminologies have been used, and with the help of the official Android and Firebase websites mentioned earlier, I will explain this terminologies.

Earlier in the explanation of some algorithms, the word **Query** was mentioned, and this interface sorts and filters the data found at a Database location so only a subset of the child data is included. This was useful to order a collection of data taken earlier, where we ordered the results by the name of the fetched patients.

Also a **holder** was used earlier, and that describes a **ViewHolder** which is used to describe an item view about its place within the RecyclerView (A dynamic list that gets updated with data). This **holder** is very essential because it helps the algorithm know where the user clicked and which information to show. So when, for example, a user clicks on **Patient X** in the list, only the information about **Patient X** is displayed.

Another interface that works together with the **holder** in order to fetch the right data is the **Adapter**. The **Adapter** is used to provide access to the data items. The **Adapter** is also responsible for making a View for each item in the data set.

So the **holder** holds the data and returns the position of each, and the **Adapter** provides access to the data and for making a view for each data set held in the **holder**.

Database reference was also used earlier, and it is used by the algorithm to access a particular location on the Database and then read or write data to that Database location. It is important because we need to start from there in order to move forward with any **Database** related operations.

And finally, I will explain the **datasnapshot**. It contains data from a **database reference**, and every time a data is read from the database, the data will be received as a **datasnapshot**.

5.2.5 Android

As mentioned earlier, the application is only for Android, and for the moment, there are no plans on going further than that. That is because Android devices are more flexible to use, more accessible to the majority and they have a lot of options within different price ranges.

Those are mainly the reasons why **Android** was chosen over other operation systems.

5.3 Hardware implementation

Along with the application and its software implementations, we will need to have some physical hardware in order for the whole concept of this thesis to work properly.

We will first need an **Android** device, of course, but it has to be a modern **Android** device able to run this application and able to take decent images. A phone with a good camera quality will be needed to accomplish high resolution medical images.

Also, a stand will be built for a stable image capturing, which means that an actual stand will be 3D printed and it's where the phone can be placed in order to, and by the click of a button, take medical images.

An additional element will be added to that stand, and that would be a lens placed on the camera lens of the phone, in order to take focused, zoomed in, high quality **RAW images**.

6 Benefits

In this chapter, I will discuss the benefits of this application for the nurses and the patients. There are several noticeable benefits for both parties, and this is where I will shed the spotlight on some of them in order to show the importance of this application in the daily lives of the nurses and the patients.

6.1 Benefits for the patient

The benefits for the patients are many, some are direct and some are indirect. Some of these benefits are as follow:

- The data is kept in an organized and structured way, and the records are tidily stored. The patient can rest assured that their data is properly stored.
- Fast and efficient data handling, and that's because digital records are created and maintained faster than physical records. When it's a physical record, a lot of paper work will be involved, yet on this application, it's all in there, in one place.
- Since records can be fetched easily, the patient can be sure that his profile will not be forgotten or lost between the records.
- The patient can rest assured that the information about him/her will always be up-to-date and synchronized. Even if they were transferred from one nurse to another.
- Since the data will be in one place, the patients can rest assured that they will be receiving a more focalized assistance.

6.2 Benefits for the nurses

The benefits for the nurses are many, they help make their lives easier, less complicated and more productive. Some of these benefits are as follow:

- **Flexibility:** This application will be flexible to the user's needs, tailored to match many possible needs for a nurse's daily tasks.
- **Simplicity:** The application is user-friendly and easy to use, learn and understand. The design was made in way to be self-explanatory.
- **Availability:** The application will be available at all times, accessible at the click of a button, and ready to be used as soon as it is executed.
- **Universality:** All the features for patient management, monitoring, in addition to RAW medical images capturing can be found in this application.

Bibliography

- [1] Christensson, P. *JPEG Definition*. TechTerms, 2016.
- [2] S. Stone. (2018, May 4) Top 20 Tools for Android Development [Web log post]. Retrieved January 31, 2019 from <https://www.altexsoft.com>
- [3] M. N. K. Boulos, A. C. Brewer, C. Karimkhani, D. B. Buller and R. P. Dellavalle. Mobile medical and health apps: state of the art, concerns, regulatory control and certification. *Online J Public Health Inform*, 5(3): 229, 2014.
- [4] C. L. Ventola. Mobile Devices and Apps for Health Care Professionals: Uses and Benefits. *P T*, 39(5): 356–364, 2014.
- [5] T. F. Scherr, S. Gupta, D. W. Wright, and F. R. Haselton Mobile phone imaging and cloud-based analysis for standardized malaria detection and reporting. *Sci Rep.*, 6: 28645., 2016.
- [6] Y. Granot, A. Ivorra, and B. Rubinsky A New Concept for Medical Imaging Centered on Cellular Phone Technology. *PLoS One.*, 3(4): e2075., 2008.

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, im April 2019