UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Kristo Karp

Android Fog Gateway for Personalized Health Monitoring

Bachelor's Thesis (9 ECTS)

Supervisors: Chii Chang, PhD
Mohan Liyanage, MSc

Tartu 2019

Androidil Põhinev Hajusarvutuse Vahevara Kliendikeskseks Terviseseireks

Lühikokkuvõte:

Bakalaureusetöö kirjeldab hajusarvutusel põhineva ja üle õhu programmeeritava terviseseire süsteemi loomist ja kasutamist. Süsteem mõeldud arstidele ja hooldajatele enda patsientide pulsi jälgimiseks. Süsteem koosneb veebiserverist, Androidi aplikatsioonist ja Polar H7 pulsivööst.

Patsiendi pulsi hetkeseisu kuvatakse Androidi aplikatsioonis, kuid kogu pulsiväärtuste ajalugu on nähtav vaid arstidele mõeldud veebilehel. Lisaks oskab Androidi aplikatsioon patsiendi pulssi analüüsida iga patsiendi jaoks spetsiaalselt loodud algoritmiga ning vajadusel teavitab patsiendi terviserikkest nii veebiserverit, kui ka saadab arsti telefonile sõnumi. Patsiendi seisundi muutumisel on olemas ka võimalus luua patsiendi jaoks uus algoritm ning see üle interneti Androidi seadmesse edastada.

Võtmesõnad:

IoMT, Firebase, Android, pilvesõnumid, pulss

CERCS:

T120 - Süsteemitehnoloogia, arvutitehnoloogia

Android Fog Gateway for Personalized Health Monitoring

Abstract:

This bachelor′s thesis details the design, implementation, and deployment of a fog computing based Over-The-Air programmable medical system which is intended to be used by doctors and caretakers to monitor the heart rate of their patients. The system consists of a web service, an Android application, and a Polar H7 heart rate sensor. It displays the current heart rate in the Android application and also has the ability to display the patient′s whole heart rate history using a Web service.

With the customized algorithm built for each patient, the Android application analyzes the patient′s heart rate and notifies the doctor as well as the Web service when it detects an abnormal heart rate value. In case the patient′s condition changes, a new algorithm can be sent from the Web service to the application.

Keywords:

IoMT, Firebase, Android, cloud message, heart rate

CERCS:

T120 - Systems engineering, computer technology

Table of Contents

1. Introduction

Healthcare is an industry where treatment decisions often have to be made in a timely manner and have to be personalized per patient.

To reach this goal, more and more medical devices are being connected to the Internet, for example, heart rate monitors, fall detection systems and smart pills [1]. In fact, nearly 87% of healthcare organizations are planning to introduce Internet of Medical Things (IoMT) related services by the end of 2019 [2].

1.1 The Problem

Although the amount of medical devices is growing day by day then most of them use preconfigured AI-based algorithms to process the medical data [1]. However, this means that basically all of the treatment and monitoring decisions are made based on the common patient's profile.

Furthermore, due to the fact that all the data processing takes place in a central server then the systems often suffer from network latency related problems when more and more medical devices are connected to the system [3]. Moreover, this can significantly increase the amount of time a treatment decision is made in a life and death situation.

Based on that it is certain that for patients who require custom treatment decisions the IoMT devices aren't currently as beneficial as for other patients. This is something the author has decided to try and change with his own implementation of a network scalable IoMT system.

1.2 The Goal

The goal of this thesis is to design and implement a prototype of a medical system that would allow the patient's medical data to be processed on a Fog node using personalized algorithms to minimize the notification delay while not overflowing the server's network bandwidth.

The system will consist of a Web service, an Android fog node and a Polar H7 heart rate sensor. It will support the following functionalities:

1. The user's heart rate will be processed using personalized algorithms.
2. The system uses fog computing paradigm and therefore the heart rate processing will take place in the fog nodes.
3. The fog node algorithm can be swapped by sending a new algorithm from the Web service using over-the-air programming.
4. The user's heart rate history and current status can be monitored from the Web service.

1.3 Outline

Section 2 describes the paradigms used in the system and also covers other similar researches.

Section 3 describes the design of the built system.

Section 4 describes the usage of the system

Section 5 discusses the future development options for the system

Section 6 contains all the references

Section 7 contains the appendix

2. State of The Art

2.1 Internet of Medical Things

2.1.1 Definition

Internet of Medical Things (IoMT) is the usage of the Internet of Things (IoT) devices in healthcare. It provides services like real-time health monitoring, patient information management, diagnostic and treatment support, etc. [3].

This paper, however, focuses on using body sensors which are used to monitor different physiological parameters of the patients e.g. blood pressure and temperature.

2.1.2 The architecture of an IoMT system

The basic structure of an IoMT system consists of three parts:

1) A central server, which stores and processes the patient's sensory data [4].
2) A local gateway that is connected to the Internet, such as a smartphone. This can have a 2-way data stream with the sensor and with the server. Usually, some of the data is already pre-processed in this unit before it is sent to the server [5]. When there is an issue in the gateway then the whole system will be affected and as a result, the real-time data wouldn't be accessible from the cloud server [4].
3) A body sensor that is attached to the patient. This sensor is made as small as possible, but at the same time, it has to have a long lifespan as well. This is why it can't have much computing power in it. This can either be wearable or implantable [5]. Depending on the implementation, those sensors can have different communication protocols to transfer data such as Wi-Fi, Bluetooth[1], ZigBee[2] or 6LoWPAN[3] [4].

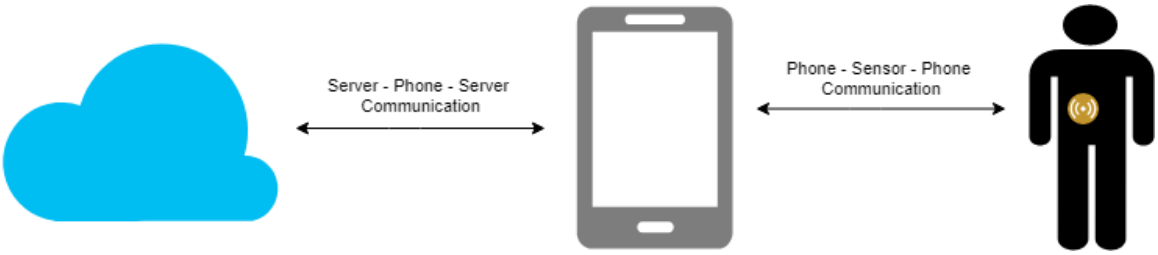

Figure 1: Architecture of an IoMT system

[1] https://en.wikipedia.org/wiki/Bluetooth
[2] https://en.wikipedia.org/wiki/Zigbee
[3] https://en.wikipedia.org/wiki/6LoWPAN

2.1.3 Wireless body sensors

Different IoMT wireless body sensors are used for an excessive amount of appliances, from post-surgery sensors that detect changes in white blood cell concentration to artificial pancreases, i.e. implanted continuous glucose monitors wirelessly interconnected with adaptive insulin pumps [6].

Sensors usually contain two core parts, a low power field programmable gate array (FPGA) and a microcontroller unit (MCU). Their combination provides low energy consumption while keeping the package very small [6]. As the sensors are made as small and energy efficient as possible, then they don't have the computing power to pre-process any data and send all collected data either to the cloud or to a gateway device.

Those sensors are also usually accompanied by a Radio Frequency (RF) module that allows them to communicate with other devices, such as smartphones [6].

In this system, the author decided to use a Polar H7 heart rate sensor to provide example sensory data. The decision to use this specific sensor was made due to the fact that the author already owned such a sensor and mostly because Polar provides a simple Android Application Programming Interface (API) to communicate between the Android device and the heart rate sensor. The Polar H7 sensor uses the low energy Bluetooth connection to send data to a connected smartphone.

2.2 Fog Computing

2.2.1 The concept of Fog computing

In fog computing, the fog distributes the resources and services of computation, communication, control, and storage closer to the users. Depending on the architecture the fog may be fully distributed, centralized, or somewhere in between [7].

In this thesis, the fog is completely decentralized, meaning that there exists a dedicated fog node for each of the heart rate sensors. All the fog nodes will use a centralized Web server as a controller to have an overview of the devices.

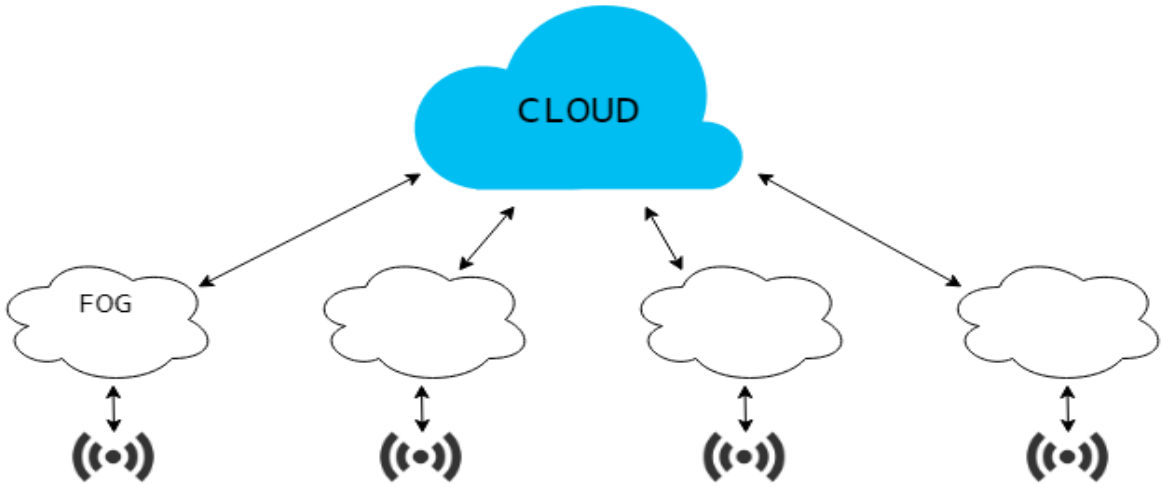

Figure 2: Concept of fog computing

2.2.2 The benefits of Fog computing in IoMT

As described by Gia, T.N et al. [4] there have been many efforts of designing smart gateways for healthcare applications. For instance, a smart gateway for health care system using a wireless sensor network, using a hybrid sensor-cloud framework for remote patient monitoring, using a personal gateway in mobile health monitoring, etc. None of the mentioned systems take the full advantage of the fog computing paradigm and basically use gateways to just to collect data and then dispense it to the remote servers. This is also why all of those system designs have some kind of flaw with them, either they are extremely limited by the network bandwidth or they don't scale well for larger systems [4].

However, according to their experiment, using proper fog computing paradigm, as a part of IoMT systems helps to achieve more than 90% network bandwidth efficiency and offers low-latency real-time response times. Which helps to overcome both the bandwidth and the scalability problem [4].

2.3 Over-The-Air Programming (OTAP)

In general, over-the-air programming means the use of wireless mechanisms to update either the firmware or the software on a mobile device. More than often the wireless mechanism is just downloading a new package over the network, installing it, and starting to use it. Even though usually the updates require restarting the device or an application after the installation, then in this system, to minimize data loss, the restart isn't required [8].

2.3.1 OTAP and dynamic code execution in Android devices

Android native applications are usually written in Java or Kotlin programming languages which are then compiled into a byte-code that is runnable by Java Virtual Machine [9] [10]. Although this code is already runnable by JVM then Android uses its own custom Runtime (ART), which is designed to be more efficient to run multiple application instances in a single device. By design, ART is basically just a middle tier that creates an abstraction layer between the code and the underlying Linux kernel, this way the developers don't have to worry about what hardware is used to run the applications [10].

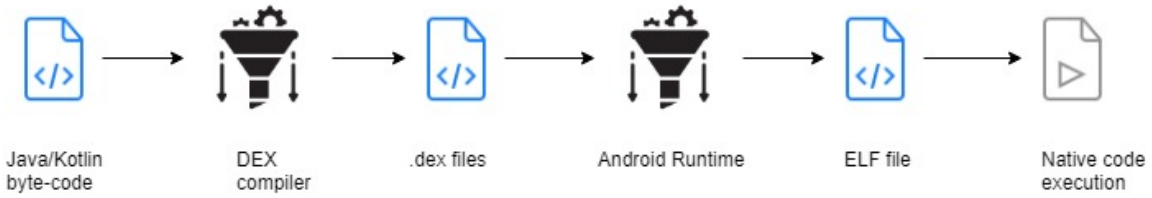

Figure 3: Code execution in Android Devices

As mentioned above, then Android Runtime is a layer built on the Linux kernel that handles low-level hardware interactions, e.g. drivers and memory management [10]. ART takes Dalvik executable files (Dex files) as input, compiles them to ELF files and then the Linux kernel executes the instructions from those [11].

This kind of architecture makes it possible to execute code that was not installed as a part of the application. For example, we can download new Dex files over the network and then use the DexClassLoader Kotlin class to load classes from a specified file path. Afterward, it is possible to execute the code from those classes [12].

2.4 Similar Work

This section will give an overview of similar research related systems and will compare them to the system that is being built as a part of this thesis.

Wireless heart rate monitor in personal emergency response system [13]

The system consists of a custom built heart rate sensor and two mobile phones (one for the doctor, one for the sensor). The sensor has a maximum heart rate threshold set and when then heart rate exceeds the threshold, then the sensor will trigger a call to the doctor's phone.

This system has no persistent way to monitor the patient's heart rate from a remote location.

Temperature and heartbeat monitoring system using IOT [14]

The framework built on this research is made of a body area network (BAN), a central server and a Web service. An Arduino is used as a middleware between the BAN and the server.

The overall workflow is that Arduino constantly forwards all the data sent by the sensors to the server. On the server, there are a few heart rate thresholds set that categorizes the user's heart rate status. The heart rates can be monitored from a Web interface, but there is no alert set up when one of the users reaches a critical heart rate status.

IoT-based patient monitoring and diagnostic prediction tool using ensemble classifier [15]

The tool the authors built is a system that predicts the user's heart rate status by using a machine learning algorithm.

First, they measure the user's heart rate with a sensor and forward the measurements to an Arduino mega. The Arduino forwards all data to a cloud server, which then stores it and calculates the risk level for the user. When the status is critical then an SMS is sent to the doctor's phone.

The doctor can also view the patient's status and previous measurements from a provided Web interface.

2.4.1 Comparison between the frameworks

System Authors / Functionality	Larkai, Wu	Kumar, Bharadwaja, Sai	Ani, Krishna, Anju, Sona, Deepa	This system
Notification method	Call	-	SMS	Web + SMS
Data processing	Sensor	Cloud Server	Cloud Server	Fog Node
HR processing algorithms	Max HR Threshold	HR value groups	Machine learning based	Personal
Uses fog nodes	-	X	X	X
Data persistence	-	Database	Database	Database
Database updates	-	Constant	Constant	After a preset time period
Monitoring	-	Web	Web	Web + App
Authentication	-	X	X	-
Supports over-the-air programming	-	-	-	X

Table 1: Comparison between similar systems

A great amount of work has already been done to develop a concept monitoring system that would provide the best possible experience for the patient and for the doctor. Some research groups focus more on the accuracy of the processing algorithms, while others try to achieve total data persistence. None of the previous works, however, seem to focus on each patient separately and for all patients, the notifications are triggered by the same requirements.

The system created in this thesis will specifically focus on each patient separately. The main difference with other systems is that for each user the heart rate processing algorithm has to be separately set up and can be swapped out without disturbing the running system.

3. System Design

In this section, you can find a description of the functionality that is required for fog computing based IoMT systems that provide over-the-air programming support. All major architectural steps are explained in the corresponding subsections and all the written code can be found in the author's Bitbucket repositories which are listed under the appendix.

3.1 System Requirements

The system requirements are divided into two parts: Web service and fog node (Android application instance).

The requirements for the Web service:

1. The user should be able to upload new Dex files to the service.
2. The user should be able to trigger an action that makes the selected fog node to use the new Dex file.
3. The service should be able to differentiate between different Android app users.
4. The service should be able to provide a different Dex file for each Android app user.
5. The service should be able to store basic information about Android app users.
6. The service should be able to do basic validation of input data.
7. The service should be able to display if the current heart rate is critical or not
8. The service should be able to display recent heart rates.

Requirements for the fog node:

1. The user should be able to enter a specific identification code.
2. The app should be able to read the heart rate from the Polar H7 sensor.
3. The app should be able to download new Dex files when the Web service sends a proper notification.
4. The app should use the logic from the downloaded Dex file to analyze the heart rate.
5. The app should work even when an improper Dex file is sent by the server.
6. The app should be able to notify the Web service when the heart rate analyzation requires so.
7. The app should be able to send an SMS to the supervising caretaker when the heart rate analyzation requires so.
8. The app should periodically send a collection of recent heart rates to the server.
9. The heart rate reading, Dex file downloading, and notifying the Web service should work even when the app is running in the background.
10. The app should display the user’s heart rate when the app is running in the foreground.

3.2 System Architecture

The built system consists of three major parts:

1. Web Service: used as a centralized server for the example system, displays patient data.
2. Fog node: acts as a processing middleware between the Web server and the Polar H7 heart-rate sensor.
3. Polar H7 heart rate sensor: provides user's heart rate data to the fog node.

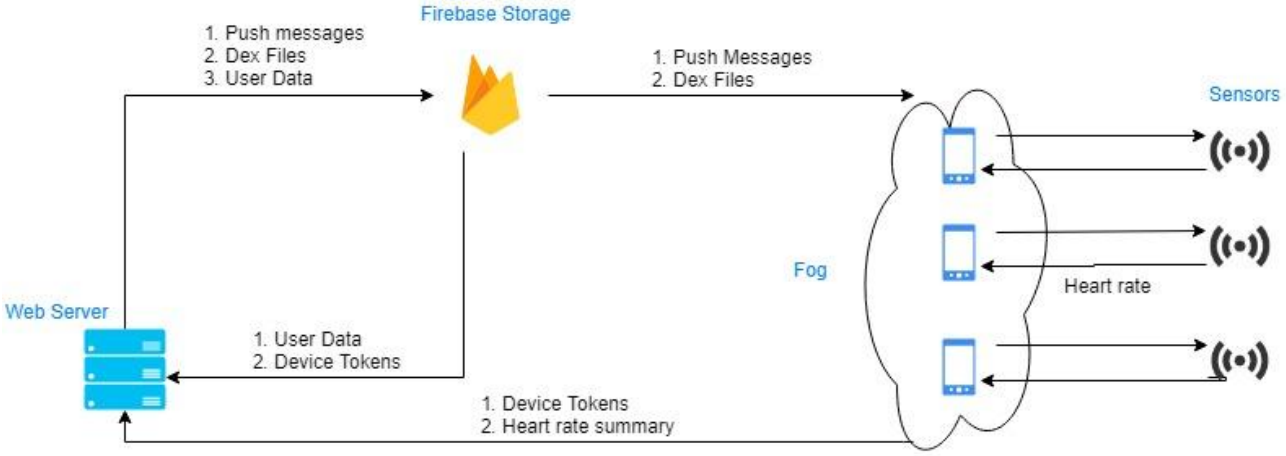

Figure 4: System Architecture

3.3 Web Service

The Web service consists of two separate parts: the server side which is where all the business logic is located and the client side, which provides the user interface.

3.3.1 Server side

The server is a REST-based API which is built using Node.js and Express. Node.js is an event-driven JavaScript runtime. The main benefits of Node.js are the facts that it is moderately fast due to using Google's V8 engine, which can compile JavaScript straight to machine code. Node also has a huge amount of community made tools accessible from the Node Package Manager (NPM)[4] [16]. Express is one of the Node packages that simplify building REST APIs.

In total the project uses six crucial Node packages:

1. Express – used for handling HTTP requests
2. Cors – used for allowing HTTP requests from the client side and fog nodes.
3. Multer – used for processing files that are sent over HTTP requests.
4. Firebase – used for communication with Firebase Realtime database
5. Firebase-admin – used to trigger sending cloud messages to fog nodes
6. @google-cloud/storage – used to upload files to google file storage

```
"dependencies": {
    "@google-cloud/storage": "^2.3.4",
    "cors": "^2.8.5",
    "express": "^4.16.4",
    "firebase": "^5.8.1",
    "firebase-admin": "^7.0.0",
    "multer": "^1.4.1"
  }
```

Figure 5: Dependency section from server side package.json

3.3.2 Client side

The client side is a separate application that is built using React.js. React is a JavaScript library meant for building user interfaces [17]. The overall purpose of this application is to request data from the server side and display it in a Web browser.

Similar to the server side, the client side can use packages from NPM as well. There are currently four types of packages:

[4] https://www.npmjs.com/

1. React, react-dom, react-router-dom, react-scripts – are required for the core React functionalities like rendering only a part of the page that has changed.
2. Axios – used for making HTTP requests to the server side
3. Materialize-css – used for styling
4. Recharts – used to display heart rate history on a chart

```
"dependencies": {
    "axios": "^0.18.0",
    "materialize-css": "^1.0.0",
    "react": "^16.8.5",
    "react-dom": "^16.8.5",
    "react-router-dom": "^4.3.1",
    "react-scripts": "2.1.8",
    "recharts": "^1.5.0"
  }
```

Figure 6: Dependency section from client side packge.json

3.3.3 Data storage

To better understand the functionality of the service, then it is best to know what kind of data is actually stored in the database.

All data is kept in a Firestore's real-time database, which is a NoSQL[5] cloud database that has the ability to keep the data synchronized in real-time on every connected client [18]. In this project, Firestore Real-Time Database is used just for storing the user data and Android tokens, the synchronization functionality is not used at all.

The database for this project consists of two collections:

1. **users**: Stores the basic information about the Android app user, user's heart rate status, the URL to the latest dex file that is uploaded for that specific user, Polar H7 device id, and the token that has been sent by the Android app to receive cloud messages.

[5] https://en.wikipedia.org/wiki/NoSQL

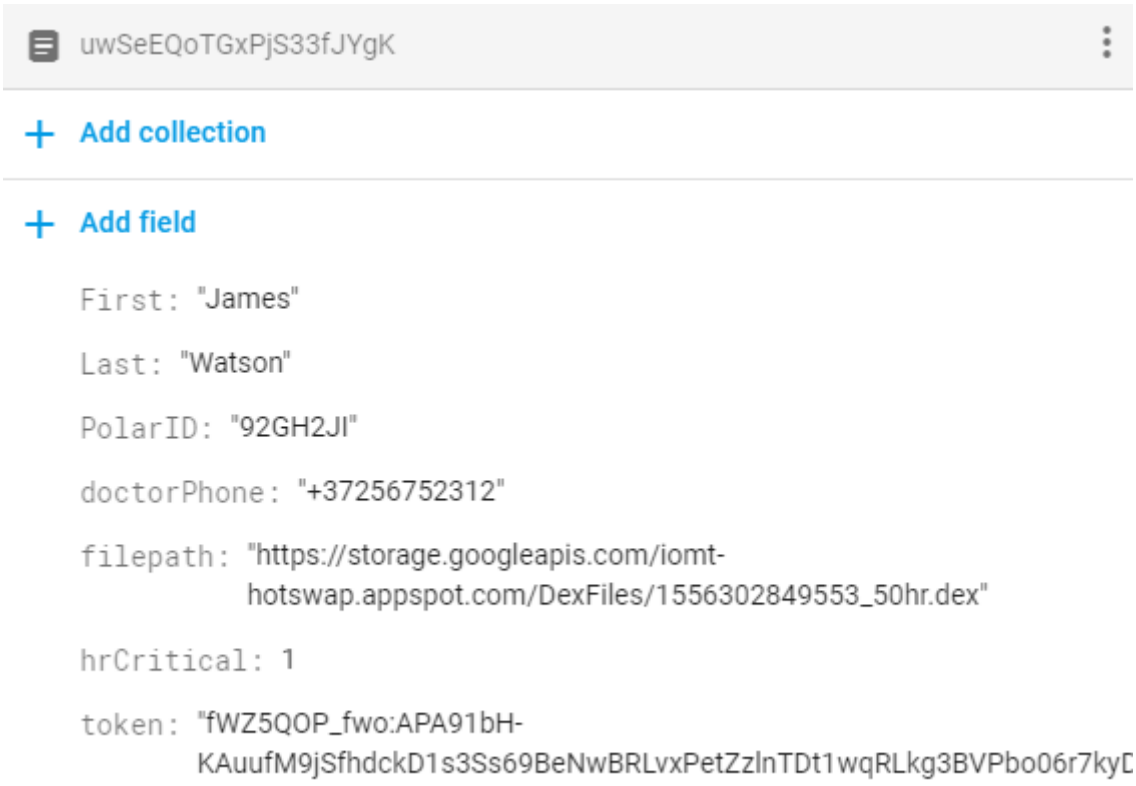

Figure 7: User's table entry

2. **hrdata**: Stores heart rate history that is sent from the fog node. This collection has three fields: heartrates – an array of heart rate values,

   timestamp – the time when server accepted the values (used for ordering the records),

   user – id of the user to whom the heart rates belong to.

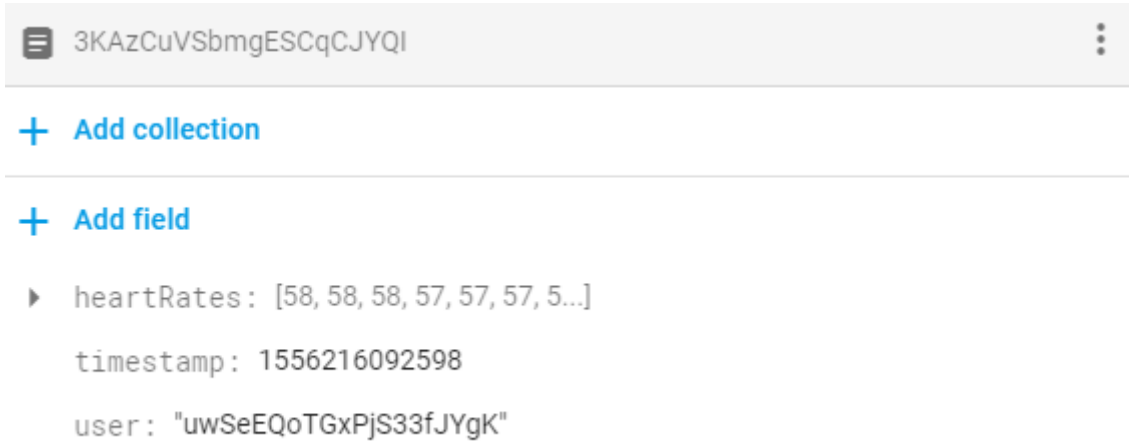

Figure 8: Hrdata table entry

3.3.4 The functionality of the Web service

The overall functionality of the service is divided into three different parts:

User data

As mentioned in section 3.3.3, all the user data is saved to the Firebase Realtime database, which is then accessed only by the server side of the service. The client side and the fog node can access the data by HTTP requests to the server. This way it will be easier in the future to implement different kinds of authorization methods to validate the users who access the data.

Dex files

All file storage in the system is handled by Firebase cloud storage. Firebase cloud storage is a service built on top of Google cloud storage. While Google cloud storage takes care of actually storing the files, then Firebase SDK takes care of simple developer access to those files [19].

After the files have been uploaded from the Web service to Firebase then the URLs for the files are auto-generated by the Google cloud storage. Although when initially setting up the storage the files can only be accessed by the Web service itself, then for simplicity reasons the author decided to make the uploaded files available for everyone to download.

Communication with fog nodes

The communication from the server to a fog node is done using Firebase cloud messaging. Firebase cloud messaging is a battery efficient way to have a connection between a server and a device, which allows you to send messages and notifications from your server to your device [20]. The benefit of cloud messages is that they are very easy to set up and they have built-in functionality to manage situations when there are network problems on the fog node side.

Cloud messaging is used to send a notification to the Fog node. The message contains an URL to the new Dex file, which is automatically downloaded to the device once the message has been received. There is currently no support on the server side to make sure that the message was actually received by the fog node, which allows for additional development opportunities.

3.4 Fog Node

The fog node functionality is provided by an Android application. In this section, the author will describe what are the requirements for the application to properly work, how the application is built.

3.4.1 Device requirements

The minimum supported Android version for the application is 6.0 e.g. Marshmallow. According to Google's 7-day data collection period ending on October 26, 2018, this Android version should cover at least 71% of current Android devices [21].

The necessary services for the application to work are Bluetooth, location info, Internet connection and an ability to send SMS messages.

Although the functionality described in this thesis doesn't use any location information, then the access is required for Polar's Android API to properly initialize and work.

3.4.2 Application architecture

The application consists of two modules: `dynamicCodeClient` and `dynamicModule`.

The `dynamicCodeClient` module is where the main application is located, it takes care of the communication with the server, reading and analyzing the user's heart rate and displaying the application in the device. The module consists of one activity and two services. One of the services, called `ClientFirebaseMessagingService` is taking care of the cloud messages sent by the Web service. The `HrService`'s purpose is to manage the connection and data flow with the Polar heart rate sensor. The `MainActivity` is the activity which is initialized when the app is opened and it is used to start and maintain the services.

The `dynamicModule` is where the logic for dynamically loading the Dex files is located. This consists of a base interface that all the Dex files should implement, a fallback module that is used when there is no Dex file or when the one sent by the server is corrupt, and a `ModuleLoader` class itself, which actually loads a dynamic class from the provided file.

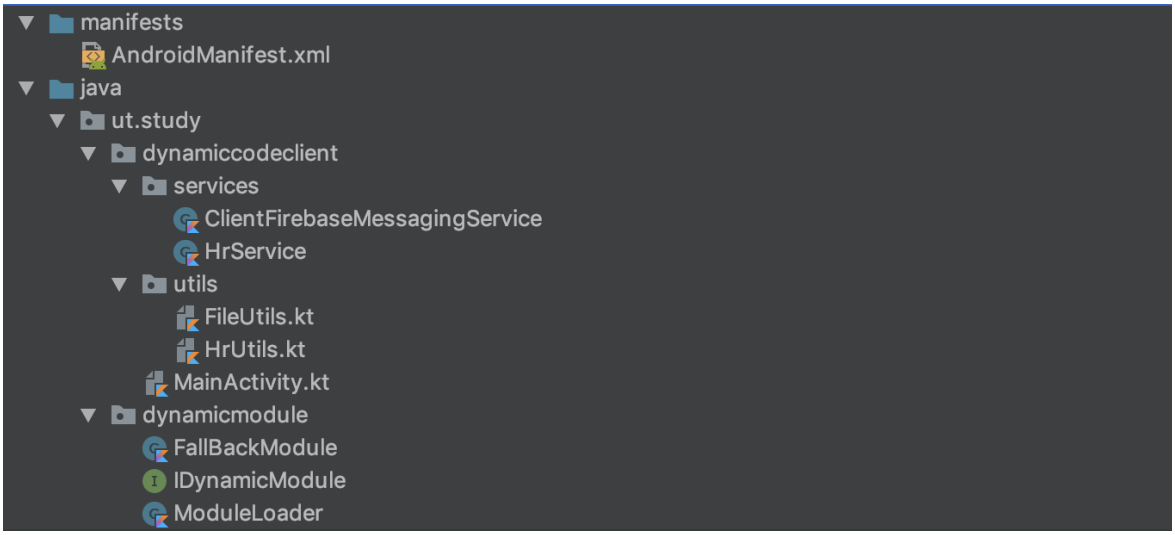

Figure 9: Android app file structure

3.4.3 Dynamic class loading

The dynamic class loading is based on an article by Dave Thomas called "Hot swapping code in Android" [22].

The requirement for all dynamic files that are loaded is that they should implement IDynamicModule interface, this way we can always load an instance of the interface, just from different files.

```
public interface IDynamicModule {
    Boolean shouldNotifyDoctor(int HR);
}
```

Figure 10: IDynamicModule interface

Once the dex file implementing the interface is present in a specified folder (in this case `applicationInfo.dataDir + "/files/"`) then we can actually start loading the class from the file.

The whole dynamic loading is based on a Kotlin class called `ModuleLoader`, which in its core uses Kotlin's built-in class called `DexClassLoader`. The `ModuleLoader` has a function `load`, which takes a file as an input and then tries to load an instance of `IDynamicModule`'s implementation from it.

```
class ModuleLoader(private val cacheDir: String) {

    fun load(dex: File?, cls: String = "ut.study.dynamicmodule.DynamicModule"):
IDynamicModule {
        try {
            val classLoader = DexClassLoader(dex!!.absolutePath, cacheDir,
                    null, this.javaClass.classLoader)

            val moduleClass = classLoader.loadClass(cls)
            if (IDynamicModule::class.java.isAssignableFrom(moduleClass)) {
                return moduleClass.newInstance() as IDynamicModule
            }
        } catch (e: Exception) {
            Log.w("ModuleLoader", e.message)
            Log.d("ModuleLoader", "using FallBackModule")
        }
        return FallBackModule()
    }
}
```

Figure 11: ModuleLoader class

When the `ModuleLoader` fails to load anything from the provided file, then it falls back to a build in `FallBackModule`, which is just a simple implementation of the interface.

```
class FallBackModule : IDynamicModule {
    override fun shouldNotifyDoctor(HR: Int): Boolean {
        return HR > 175
    }
}
```

Figure 12: FallBackModule class

3.4.4 Data flow

The Application itself has two different data flows, one related to the dynamic loading of the Dex files and another related to measuring the heart rate.

Dex file flow

The Dex file flow is quite simple, first, a message comes in from Firebase cloud messages, which contains an URL where the dex file can be downloaded from. The file is then downloaded from the provided URL and saved to `applicationInfo.dataDir + "/files/"` folder on the device.

When the file has been downloaded then the `ClientFirebaseMessagingService` broadcasts a message to the local broadcast. The main activity is constantly monitoring for messages in the local broadcast queue and when a message with the code `FIREBASE_FILE_RECEIVED` is found, then the main activity switches the active Dex file variable to the filename from the message and also saves the filename to shared preferences, so that the filename could be retrieved even when the application itself is completely killed and then reopened.

Once the active Dex file variable has been changed and the filename has been saved to the preferences, then the old files are deleted from the files folder and only the new just downloaded file is left to the folder.

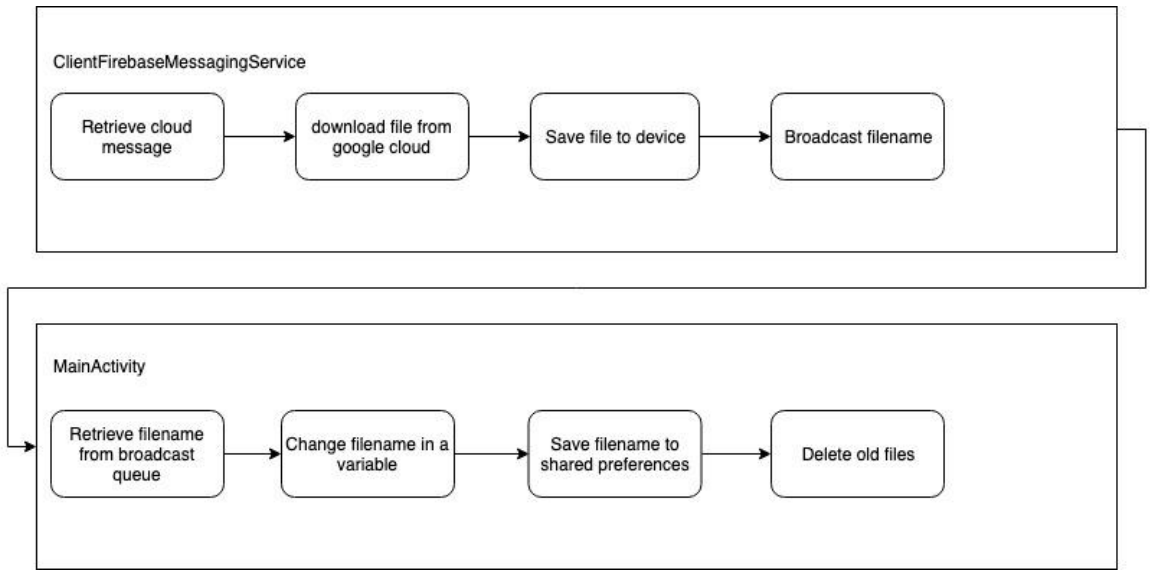

Figure 13: Dex file flow

Heart rate data flow

The heart rate flow starts with the heart rate sensor sending the data to the fog node. In the node, `HrService` will broadcast the heart rate value to the local broadcast using `HR_MESSAGE` identifier. At the same time, the heart rate is also added to a `HrHistory` class instance, which will later send a collection of the heart rate values to the Web service.

The `MainActivity` is constantly monitoring the local broadcast queue. When a message with `HR_MESSAGE` identifier is found, then a class instance is loaded from the active Dex file and the heart rate is analyzed. If the analyzation result is different from the previous one, then the Web service and the doctor's phone are notified that the heart rate has changed to critical/non-critical.

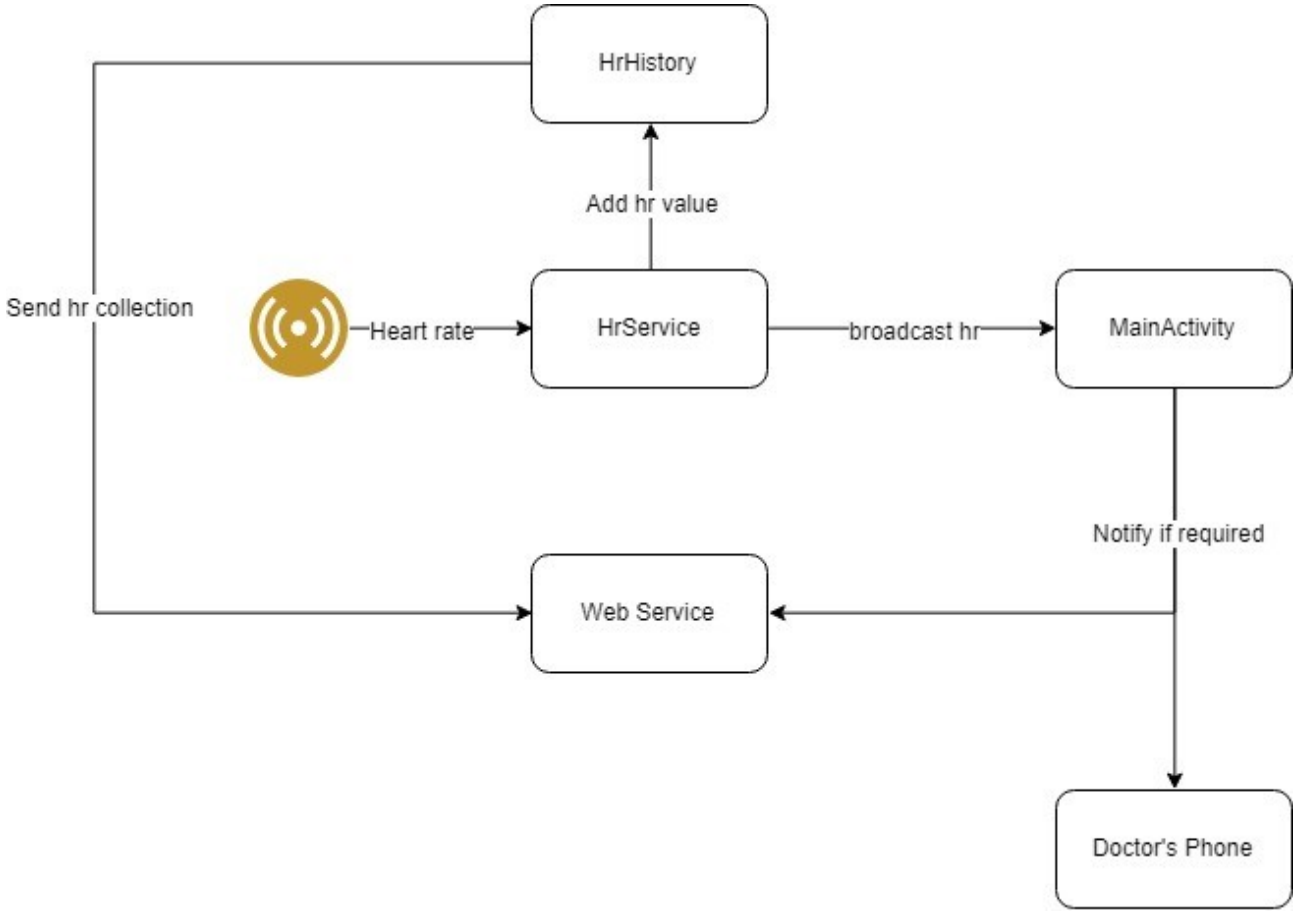

Figure 14: Heart rate data flow

4. Prototype Implementation And Performance Evaluation

4.1 Prototype Implementation

This section will demonstrate the usage of the system in a use case where the patient in focus is an elderly man with previous heart attacks. The main reason for the set-up is to monitor that there wouldn't be any excess stress on the heart.

In this use case there are 3 example requirements for the monitoring algorithm:

1. The heart rate can not exceed 120 beats per minute (BPM) during the daytime (8:00 – 20:00)
2. The heart rate can not exceed 80 BMP during the night time (20:00 – 8:00)
3. The heart rate can not fall below 40 BPM at any time of the day

4.1.1 Building the algorithm

The algorithm for the example patient is will be built in Java and will be compiled into a dex file using Android studio and the guide provided by Dave Thomas in his article about Hot swapping code in Android [22].

```
Import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;

public class DynamicModule implements IdynamicModule {

    @Override
    public Boolean shouldNotifyDoctor(int HR) {

        Calendar calendar = GregorianCalendar.getInstance();
        calendar.setTime(new Date());
        int hour = calendar.get(Calendar.HOUR_OF_DAY);
        24oolean isDay = (hour >= 8 && hour < 20);

        if (isDay && HR > 120) return true;
        else if (!isDay && HR > 100) return true;
        else return HR < 40;
    }
}
```

Figure 15: Personal algorithm

4.1.2 Initial setup

The initial setup for the patient is done using the Web service where the doctor adds a new patient (figure 16), enters the patient's name, the device id for the polar sensor and uploads the algorithm that has been created for the patient (figure 17). The healthcare organization will also provide the patient with an Android device that has the application pre-installed and a Polar H7 heart rate monitor.

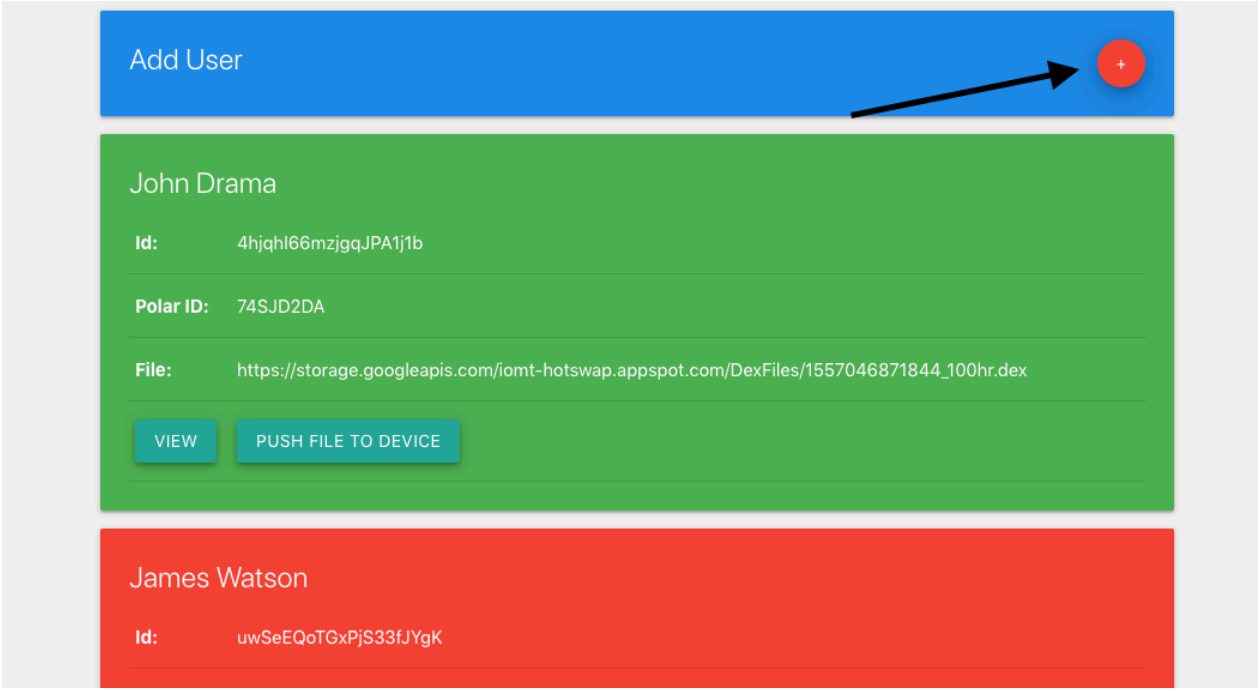

Figure 16: Add a new user button in the user's view

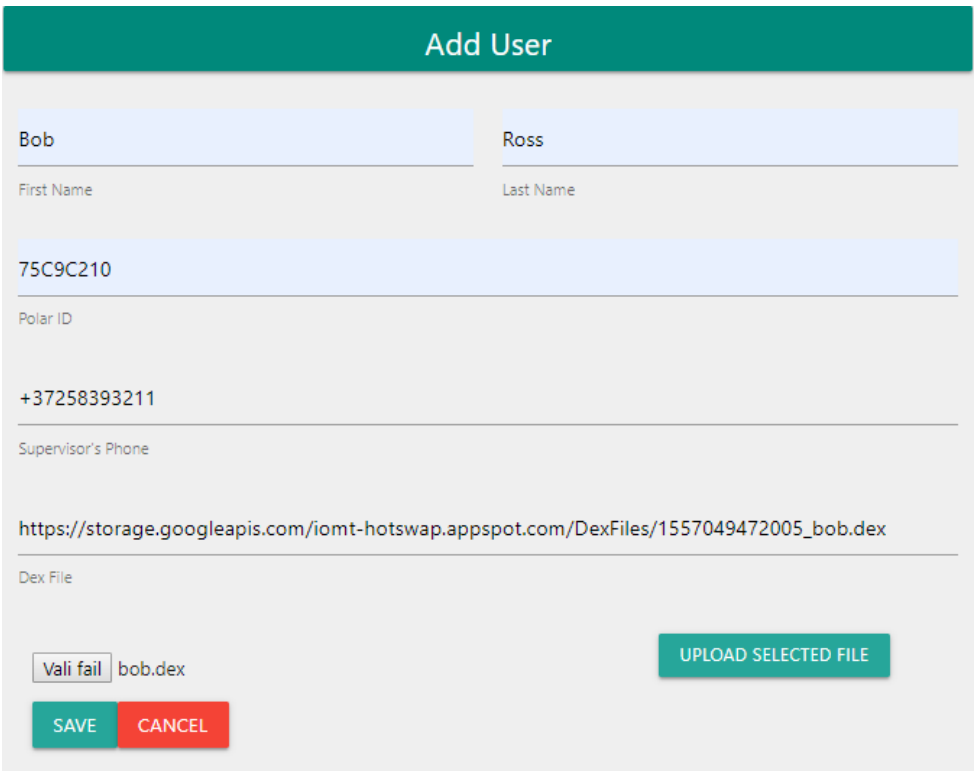

Figure 17: Add User view

Once the data has been saved in the Web interface, then the user can wear the sensor and start up the Android application.

Figure 18: Polar H7 heart rate sensor

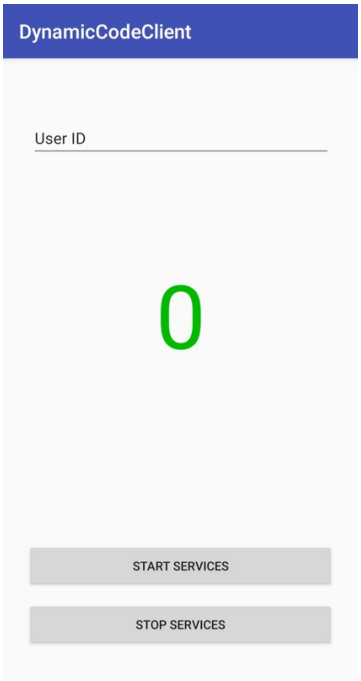

Figure 19: Android application

At this point, there is user data and heart rate algorithm set up at the Web service and the user is ready for monitoring. The user is now also visible in the overall user's list on the Web service.

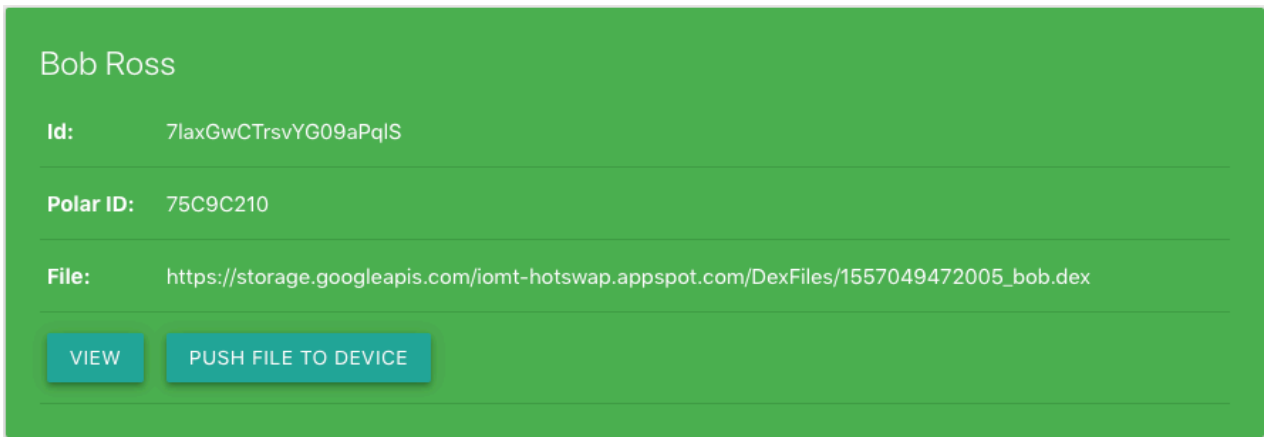

Figure 20: User profile in the users' list

The next step is to enter the auto-generated user id to the Android application and press the START SERVICES button at the bottom of the screen. The button press will trigger a request to the Web service and the application asks for a polar device id that it has to connect to. When the device id is returned, then the application will start the `HrService` and will connect to the provided Polar device. The number in the middle will start displaying the user's real-time heart rate.

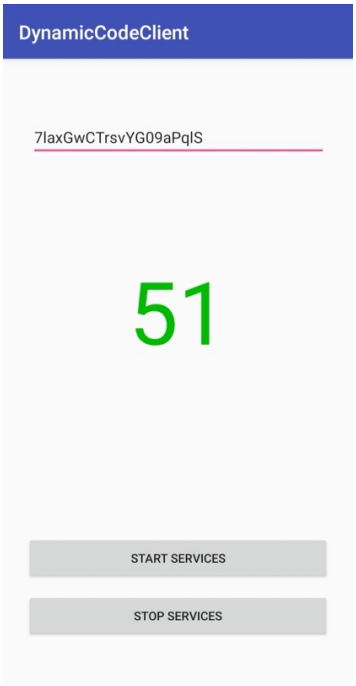

Figure 21: Application after starting the services

4.1.3 Using the customized algorithm

Even though a custom monitoring algorithm was uploaded to the Web service, then the application isn't using it yet and is using the provided `FallBackModule` instead. This is due to the fact that the Web service has not explicitly told the application that is should download the provided Dex file.

To trigger a cloud message from the server to the Android application, then the PUSH FILE TO DEVICE button has to be pressed in the Web service (See figure 20). The message will contain an URL from where the application will download the file and will then start using the new algorithm.

```
D/HrService: HR 61
D/ModuleLoader: using FallBackModule
D/HrService: HR 61
D/ModuleLoader: using FallBackModule
D/HrService: HR 60
D/ModuleLoader: using FallBackModule
D/firebaseService: Firebase started
D/firebaseService: From: 1092594600257
D/firebaseService: File Url: https://storage.googleapis.com/iomt-hotswap.appspot.com/DexFiles/1557049472005_bob.dex
D/HrService: HR 59
D/ModuleLoader: using FallBackModule
D/firebaseService: fileLoc: /data/user/0/ut.study.dynamiccodeclient/files/1557056895227155704947200 5_bob.dex
D/firebaseService: finished downloading file: 1557049472005_bob.dex
D/HrService: HR 58
D/HrService: HR 58
D/HrService: HR 59
D/HrService: HR 59
```

Figure 22: Android app logs after receiving the new algorithm

4.1.4 Monitoring the heart rate

Android application

In the Android device, the heart rate can always be checked by just opening the application. When the heart rate is normal, then the value is displayed as a green number at the center of the screen, but when the value is in the critical section then the number will turn red and SMS is sent to the doctor's phone.

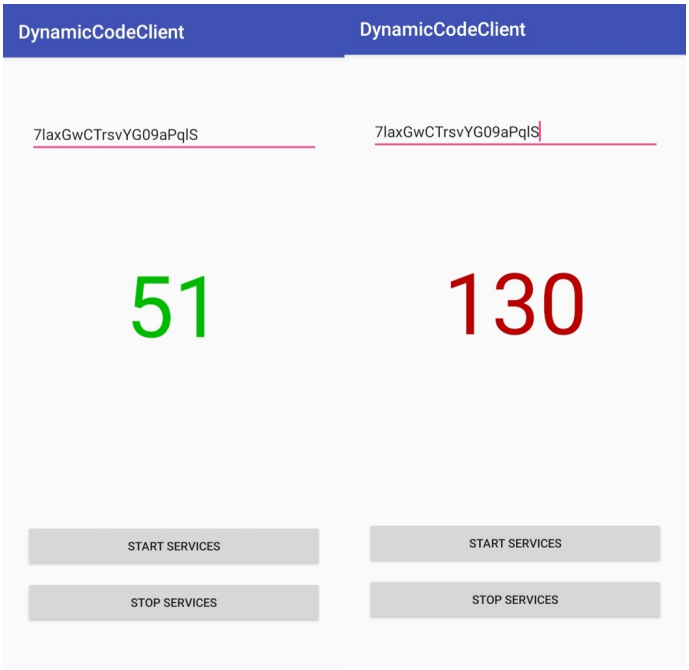

Figure 23: Non-critical and critical heart rates in the Application

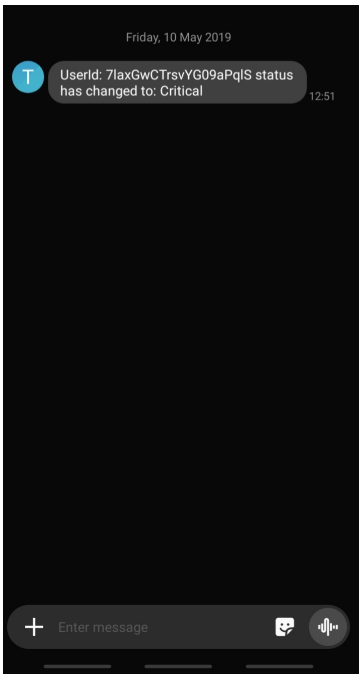

Figure 24: Notification SMS

Web service

Similar to the application, we can see if the heart rate is critical or not in the patient's overall view. When the heart rate is not critical then the user's card is green in the list, but when the heart rate is critical then the card turns red.

Figure 25: Non-critical and critical patients

After clicking the VIEW button on the patient's card, then more information about the patient will be displayed. In the detailed view, the doctor can view the patient's current heart rate status as a green/red indicator and can also see a graph showing how the patient's heart rate has changed over time. On this screen, the doctor can also upload new algorithms for the patient or change other patient-related information.

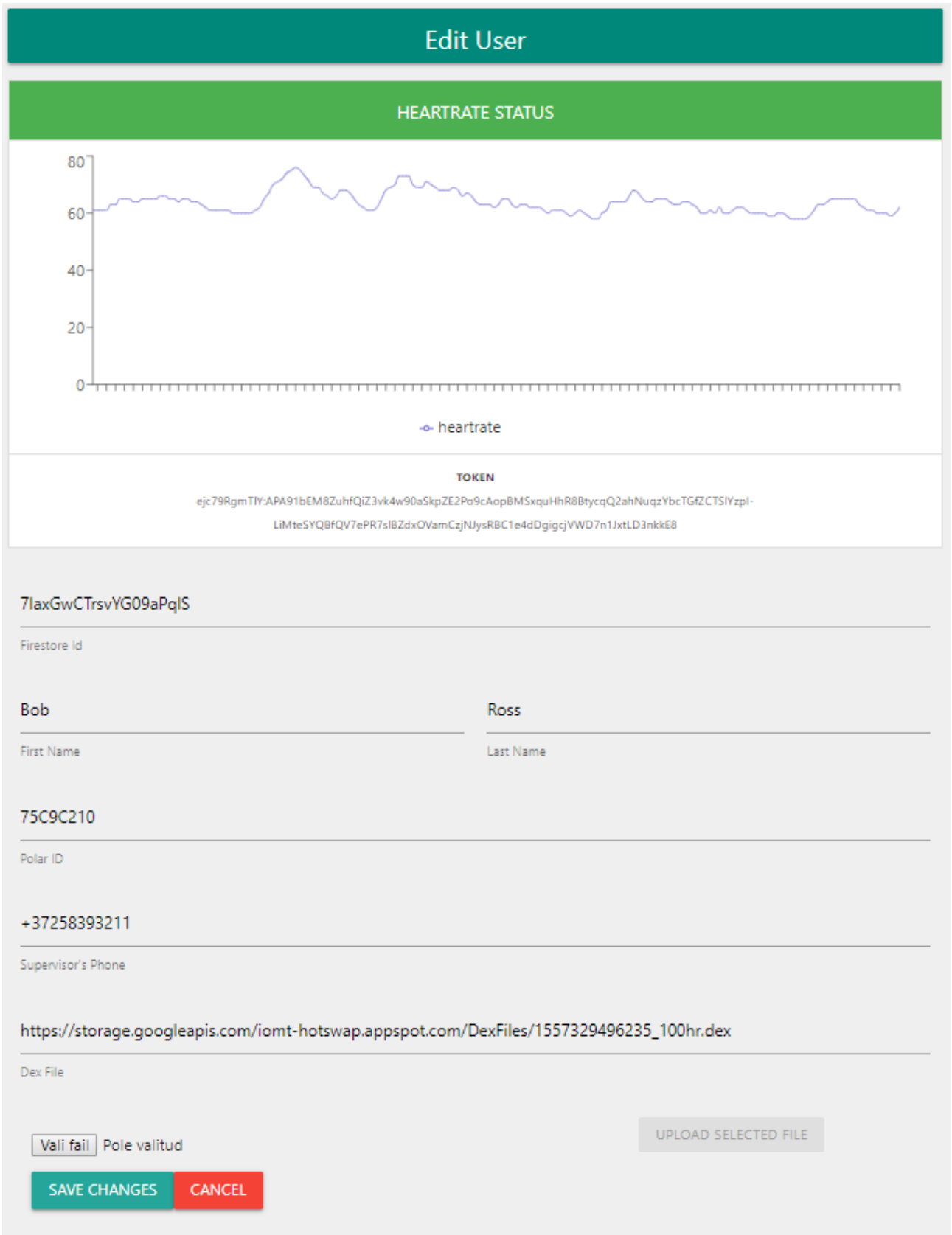

Figure 26: Detailed view with non-critical heart rate

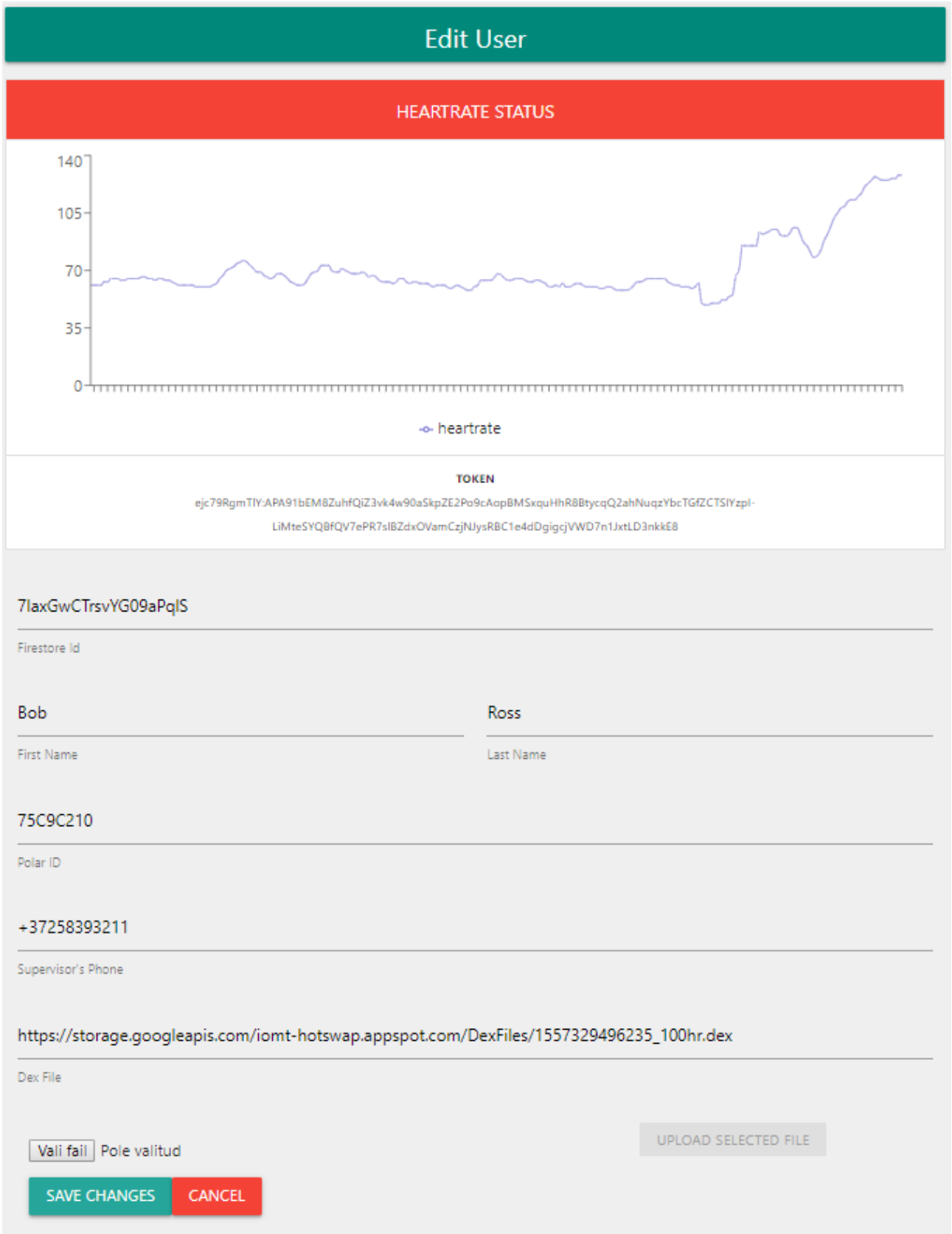

Figure 27: Detailed view with critical heart rate

4.2 Performance Evaluation

4.2.1 Benefits of Fog computing

This section presents a comparison to show how big of a performance gain the usage of the fog computing paradigm actually provides for the built system. Here the author will compare two different scenarios:

1. Heart rate processing takes place in a cloud server.
2. Heart rate processing takes place in the fog node.

To compare those two scenarios, then the author measured the time it takes for the critical heart rate notification to reach the doctor's phone in both cases. The starting time for the measurement was recorded when the fog node received the heart rate from the sensor and the end time was measured when the notification reached the doctor's phone.

Cloud computing based processing

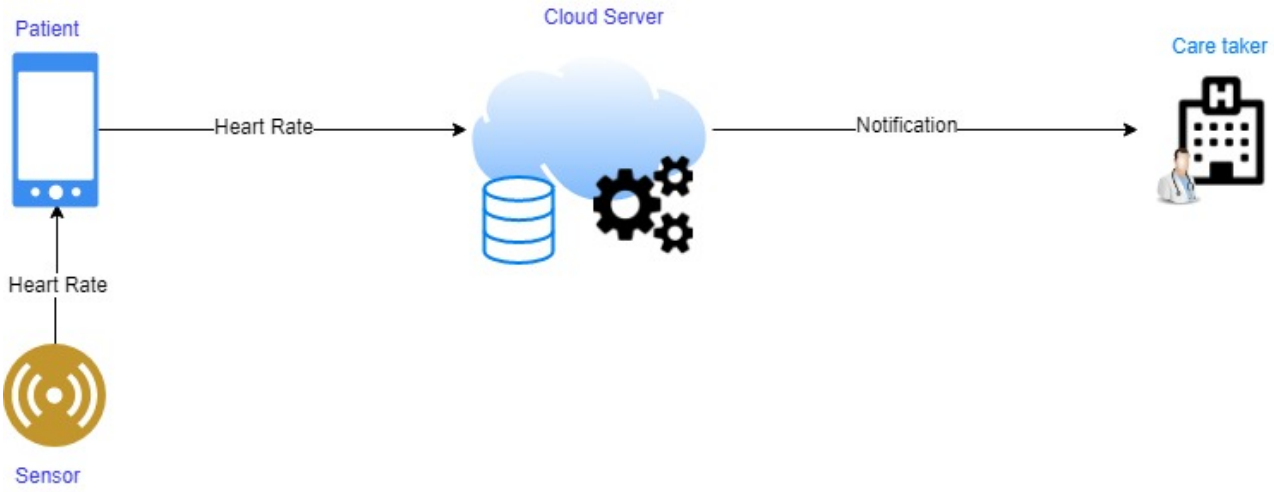

Figure 28: Cloud computing based processing

In this kind of architecture, the patient's phone forwards the heart rate to the cloud server, where the server runs the data processing algorithm to decide if a notification should be sent to the doctor. The heart rate status is also saved to the database when it has changed, this way the patient's status is updated in the web interface to notify caretaker in the hospital.

The main problem in a cloud computing based monitoring system is that the additional time it takes for the data to be sent from the patient's phone to the server can be crucial in a life and death situation.

Furthermore, when the network link between the patient's phone and the cloud server breaks, then the critical heart rate notification couldn't be sent to the doctor and also couldn't be displayed in the web interface.

Fog computing based processing

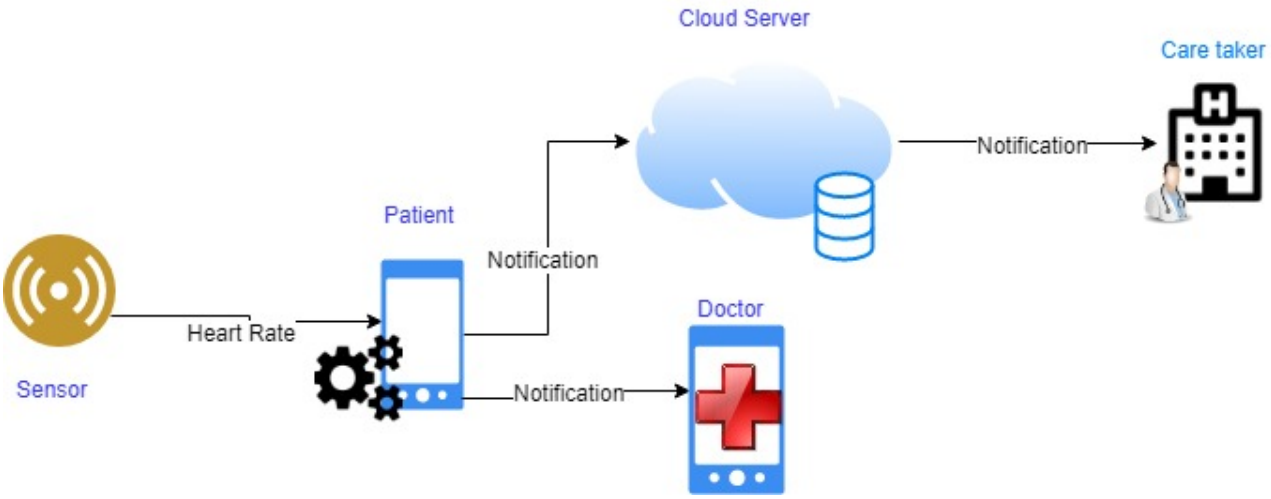

Figure 29: Fog computing based processing

In fog computing based processing, the fog node processes the heart rate itself and decides if a notification should be sent or not. When a notification is required, then the fog node will start two independent requests, first to send a notification to the doctor's phone and the second to notify the caretaker in the hospital by updating the patient's status in the web service.

The main benefit here is that as the processing happens really close to the sensor itself, then all delays are kept as minimal as possible. Moreover, as the two notifications are sent as separate requests, then there is a good chance that even when either the patient-doctor or patient-server network link breaks, the other one will stay intact and the notification won't be lost.

Results

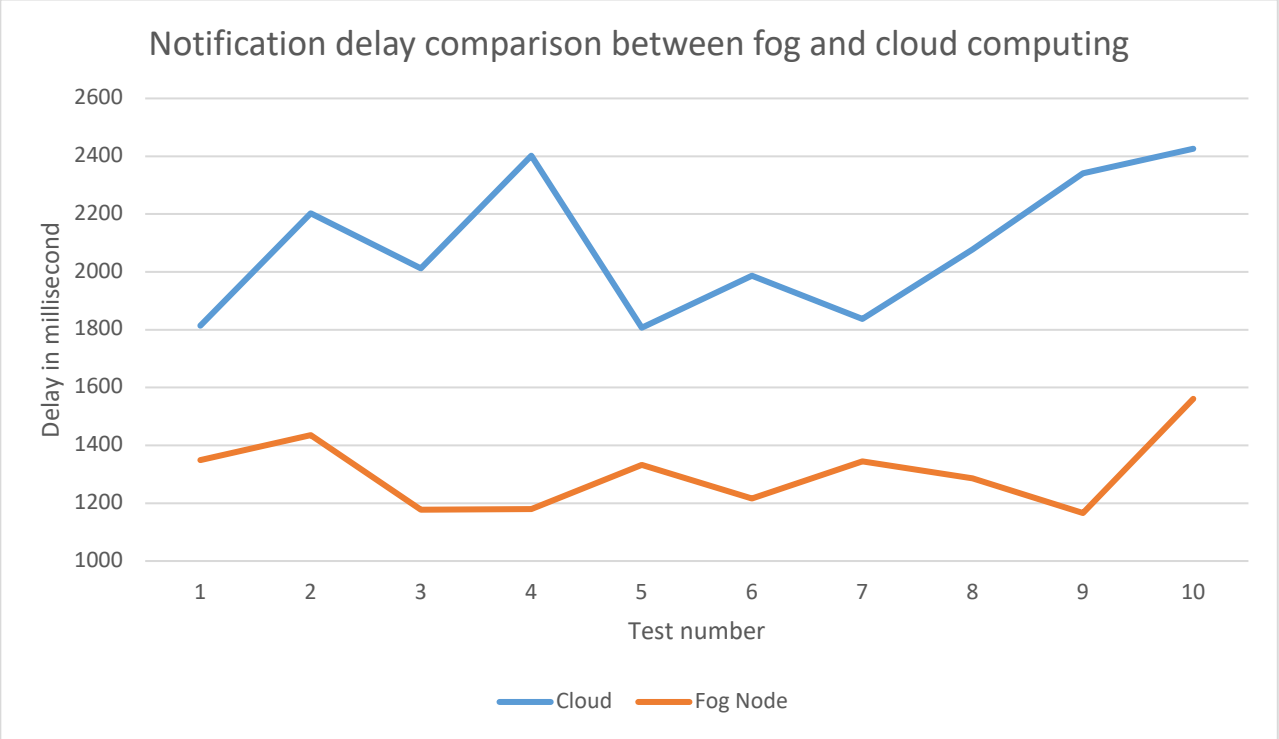

Figure 30: Notification delay in fog and cloud computing

As seen from Figure 30, in cloud computing based solution sometimes the notification delay can exceed two seconds, which is compared to the fog based system where the delay is just half of it. Moreover, as this experiment was conducted using a strong network high-bandwidth signal then this delay can grow exponentially in locations where the signal strength or network bandwidth is lower. In the medical world those second, however, can be the crucial parameter to decide if the outcome of the situation is positive or not.

The usage of cloud computing in this kind of system would only make sense when the data processing algorithm is exceptionally resourced hungry and the fog node wouldn't be able to process the data in a timely manner.

4.2.2 Enhancements over the other similar systems

The main enhancement over other similar systems is the fact that for each patient the monitoring algorithm is handcrafted especially for them. This allows the highest possible level of personalization for the patient. A great example would be that a patient needs different heart rate limits for daytime and nighttime, in a regular configuration based systems this would be an edge case and would require a whole update the system, but in the Dex file based system this is fairly simple as seen in the section 4.1.1.

4.2.3 Limitations of the system

The main limitation of the system is that each time a new user is added to the system, then the algorithm for the user has to be developed as well. This, however, requires that all the personnel, working with the system, need to have great programming knowledge as code bugs can be fatal in this situation.

Due to the fact that the application loads some of its code dynamically then it also can't be published in the Google Play store, as according to the Google play policy "An app distributed via Google Play may not modify, replace, or update itself using any method other than Google Play's update mechanism. Likewise, an app may not download executable code (e.g. Dex, JAR, .so files) from a source other than Google Play." [23].

5. Conclusion And Future Work

As a result of the thesis, a fog computing based heart rate monitoring system was developed, providing doctors and caretakers with a way to monitor their patients. The system consists of a Web service for the doctors, an Android application for the patients and a Polar H7 sensor that provides the heart rate to the application. The Android application displays the heart rate and periodically sends heart rate data to the Web service. The application also notifies the Web service when the heart rate status changes to critical.

The main difference with other similar systems is the possibility to develop a custom heart rate processing algorithm for each patient without requiring any modifications in the underlying Android application. The algorithms can be pushed from the Web service to the Android application over the network.

5.1 Future Work

Similar to other medical and software related projects, the system can always be improved to be more reliable and user-friendly. There are several major features that would make the system much more feasible for real-life usage:

User authentication

In the Web service, the data should be accessible after the doctor has authenticated himself/herself and should display only information about patients who belong to the doctor.

More health measurements

In the future the system should support more than one health measurement, great additional measurements of the heart rate would be body temperature and blood pressure, as those already have great Bluetooth sensors developed.

Browser code editor and dex compiler.

The Web interface could have a built-in code editor that would allow the doctors to write and compile the algorithms in the Web service. This way there would be no need for external applications and the personnel would save a lot of time.

6. References

[1] V. Kamani, "A Detailed Guide To IoMT Implementation in 2019," [Online]. Available: https://arkenea.com/blog/iomt/. [Accessed 3 May 2019].

[2] R. Parker, "Internet of Things in Healthcare: What are the Possibilities and Challenges," [Online]. Available: https://readwrite.com/2018/01/13/internet-things-healthcare-possibilities-challenges/. [Accessed 27 April 2019].

[3] A. K. Chattopadhyay, A. Nag and D. C. K. Ghosh, "A Secure Framework for IoT-Based Healthcare System," in *Proceedings of International Ethical Hacking Conference 2018*, Kolkata, 2019.

[4] T. N. Gia, M. Jiang, A.-M. Rahmani, T. Westerlund, P. Liljeberg and H. Tenhunen, "Fog Computing in Healthcare Internet-of-Things: A Case Study on ECG Feature Extraction," in *IEEE International Conference on Computer and Information Technology*, Liverpool, 2015.

[5] K.-H. Yeh, "A Secure IoT-Based Healthcare System With Body Sensor Networks," in *IEEE Access, vol 4*, 2016, pp. 10288-10299.

[6] G. E. Santagi and M. Tommaso, "An Implantable Low-Power Ultrasonic Platform for the Internet of Medical Things," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, Atlanta, 2017.

[7] "Fog and IoT: An Overview of Research Opportunities," in *IEEE Internet of Things Journal*, IEEE, 2016, pp. 854-864.

[8] ""Over-The-Air Programming,"," [Online]. Available: http://portablecontacts.net/wiki/development/over-the-air-programming/. [Accessed 4 May 2019].

[9] L. Ndwaru, Introduction to Android ART; The next generation of Android Runtime, Nakuru: Egerton University, 2014.

[10] R. Meier and I. Lake, Professional Android, Wrox, 2018.

[11] A. Frumusanu, "AnandTech," [Online]. Available: https://www.anandtech.com/show/8231/a-closer-look-at-android-runtime-art-in-android-l/. [Accessed 20 April 2019].

[12] Google, "DexClassLoader | Android Developers," Google, [Online]. Available: https://developer.android.com/reference/kotlin/dalvik/system/DexClassLoader. [Accessed 20 April 2019].

[13] D. L. Larkai and R. Wu, "Wireless Heart Rate Monitor in Personal Emergency Response System," in *IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, Belgrade, 2015.

[14] G. V. Kumar, A. Bharadwaja and N. N. Sai, "Temperature and heart beat monitoring system using IOT," in *International Conference on Trends in Electronics and Informatics (ICEI)*, Tirunelveli, 2017.

[15] R. Ani, S. Krishna, N. Anju, M. Sona Aslam and O. S. Deepa, "Iot based patient monitoring and diagnostic prediction tool using ensemble classifier," in *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Udupi, 2017.

[16] R. Hegde, "What are the advantages of using Node.js compared to other backend scripting languages?," [Online]. Available: https://www.quora.com/What-are-the-advantages-of-using-Node-js-compared-to-other-backend-scripting-languages-What-are-the-advantages-against-Python-and-Ruby. [Accessed 29 April 2019].

[17] Facebook Inc, “A JavaScript library for building user interfaces,” [Online]. Available: https://reactjs.org/. [Accessed 29 April 2019].

[18] Google inc, “Firebase Realtime Database,” [Online]. Available: https://firebase.google.com/docs/database/. [Accessed 21 April 2019].

[19] Google inc, “Firebase Cloud Storage,” [Online]. Available: https://firebase.google.com/docs/storage/. [Accessed 21 April 2019].

[20] Google Inc, “Firebase Cloud Messaging,” [Online]. Available: https://firebase.google.com/docs/cloud-messaging. [Accessed 21 April 2019].

[21] Google Inc, “Distribution Dashboard,” [Online]. Available: https://developer.android.com/about/dashboards/index.html#Screens. [Accessed 29 April 2019].

[22] D. Thomas, “Hot swapping code in Android,” [Online]. Available: https://medium.com/quick-code/hot-swapping-code-in-android-3043ccf6dd9b. [Accessed 14 November 2018].

[23] Google Inc, “Developer Policy Center,” [Online]. Available: https://play.google.com/intl/en-US/about/developer-content-policy-print/. [Accessed 4 May 2019].

7. Appendix

7.1 Code Repositories

Web service server side: https://bitbucket.org/karpUT/iomt_hotswap/

Web service client side: https://bitbucket.org/karpUT/iomt-hotswap-frontend/

Android application: https://bitbucket.org/karpUT/iomt-hotswap-android/

7.2 License

Non-exclusive licence to reproduce thesis and make thesis public

I, Kristo Karp,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Android Fog Gateway for Personalized Health Monitoring,
supervised by Chii Chang and Mohan Liyanage.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Kristo Karp
15/05/2019