# Abstract

This thesis describes development of an android application to do health monitoring of the elderly person and remotely controls a remote robot “TurtleBot” in order to allow caregivers to interact with elderly person in more convenient way.

A hybrid android application has been developed. The main services of the application are divided into two: i) Health Monitoring System which provides caregivers with elderly person's physiological data collected by various sensors installed in or around the house, and ii) Robot Telepresence System which controls the robot by using android application and provides video chat with elderly person.

The application is connected with a database server and is able to receive elderly person's physiological data by using HTTP Request. In order to remotely control the "TurtleBot", ROS (Robot Operating System) has been used. The video chat service is implemented by using WebRTC technology, which enables peer to peer connection between the application and the TurtleBot laptop.

The developed application is tested by using various performance testing tools. The results show that the application has relatively high performance. The only thing that could be problem is unexpectedly high battery consumption while using video chat service.

# Contents

# 1 Introduction

In the last few years, there are a growing number of robots being designed and built to interact with people in a specific setting. Above all, recent studies on interaction with robots focus on the importance of social intelligence even more so in a healthcare/eldercare sector. A number of researchers have been inspired by the expected growth in the elderly population and the labor shortages in the healthcare sector to explore the applicability of intelligent systems in general and robotic products in particular to be used in assisted-living environments. For robots, the functionalities are related to supporting independent living by supporting basic activities: eating, bathing, toileting, getting dressed and mobility, and providing household maintenance, monitoring of those who need continuous attention and maintaining safety [1].

Meanwhile, the mobile app business is growing rapidly, and many studies about apps development have been conducted. Now the mobile applications have become the one of thing we can't leave behind in our daily life. Mobile apps were originally offered for general productivity and information retrieval, including email, calendar, contacts, stock market and weather information [2]. However, public demand and the availability of developer tools drove rapid expansion into other categories, such as those handled by desktop application software packages [3].

In this thesis, these two main services are combined by making an android application to remotely control a remote “TurtleBot 2” robot [4] in order to allow caregivers to interact with their patient in a more convenient way than using laptop. The fundamental idea is from GiraffPlus project which allows the caregivers to remotely control a existing elderly care robot "Giraff" by using their laptop [5]. The more detailed description about the GiraffPlus project will be described in section 2.2.

In fact, almost all types and versions of smart phones have already included videoconference capabilities and technologies needed for controlling a robot, such as standard internet connection, Wi-Fi, Bluetooth and accelerometer etc. That helps us develop the application. Figure 1.1 shows us the basic concept of the application we are going to deal with in this

thesis.

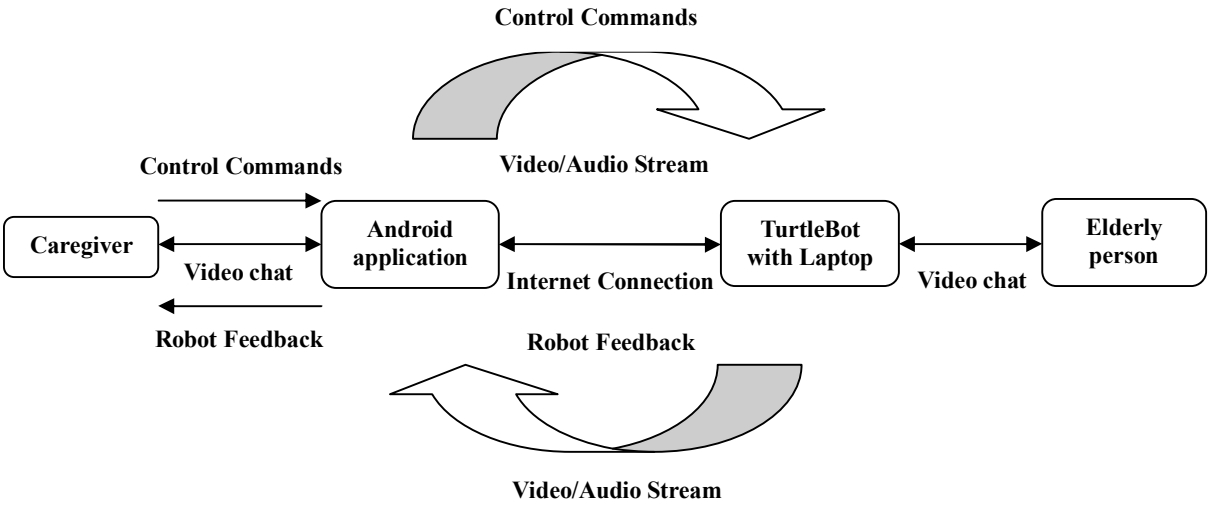



Figure 1.1 The basic concept of the application

This thesis describes the research about implementing two main services for assisting caregivers, e.g. Health Monitoring System and Robot Telepresence System. The systems are developed as combinations of several parts that work outside of the android application such as database server. The developed application will be a demo version aiming at investigating how to develop a complex system using different technologies and combine different parts into the android application in effective way.

The thesis consists of 8 chapters. The first chapter we introduce background of the thesis including the "GiraffPlus" project which is deeply related with our work. The second chapter describes about implementation of the project, from design of the application to actual development. In the last chapter the thesis describes the results of our work including performance testing of the application and our future work.

# Development of Android Application

# 3 Design for Android Application

Now we are going to start making design for android application we are going to develop. The design includes: i) which functions or services the application should have, ii) basic structure of the application and iii) general workflow of the application.

## 3.1 Main Services

At first, we start with choosing what functions or services the application should have. When we consider about the main user of the application, it will be basically caregivers and relatives of elderly people. The main services of the application are to allow caregivers to monitor elderly person's physiological measurements, and interact with the elderly person via Internet. The services will be based on DVPIS and Pilot software that are described in chapter 2.2.2 and 2.2.3, and three essential services are chosen such as:

i. Access to the central database and visualize elderly person's physiological parameters.
ii. Remotely control the remote robot and navigate the robot via application
iii. Initiate call, perform face-to-face video chat

i) Monitoring of elderly person's physiological measurements. The measurements are collected by various kinds of sensors placed in elderly person's home environment and stored in central database. The application sends request for specific data the user wants to the database via main server and the retrieved data is visualized as a graph in the android application.

In this thesis, we don't use real sensors, but we research how the GiraffPlus uses sensors and makes sensor network in the project, and we assume that the developed application use same sensor network as the GiraffPlus project.

Technologies used in the Health Monitoring System are almost same as the technologies used in the GiraffPlus project. The main server is developed as a web application server by using

Spring Framework [45] and Apache Tomcat [47]. MongoDB [28] is used in order to make a demo database. More detailed information about the technologies used in the Health Monitoring System will be described in chapter 4.1.

ii) Remote robot control service, including mapping and navigation. Since we use a remote robot "TurtleBot" in this thesis, the service is using ROS technology [50] in order to remotely control the robot. The "TurtleBot" using ROS technology has researched by many developers and there are a lot of open source codes we can use for our developing. In the rest of the thesis, this service will be called Pilot System.

While the Pilot software of the GiraffPlus project used mouse click to navigate robot, our application will use virtual controller with arrows to navigate the robot, because of less precision of mobile device's touch based input.

The application displays basically camera view like the Pilot software of the GiraffPlus project does, but as an optional function, a map view will also be implemented with a auto navigation function, which can navigate the robot by clicking or touching a point of the map.

Since our goal is to make an application, which can remotely control the robot via standard Internet, we are going to consider how we can communicate with TurtleBot laptop to control the robot without Wi-Fi connection.

iii) In order to implement the video chat system we use WebRTC (real time connection) technology [47], which makes peer to peer connection between two clients. The client program is developed as a web application, and is integrated into the android application by using Apache Cordova software [48]. So, the android application will be used by caregivers, while the web application will be used by elderly people. When the caregiver initiates call by using android application, the video chat system using WebRTC makes a peer to peer connection with elderly person's web application and exchange audio and video stream [49].

The video chat system relies heavily on the device’s network performance. The mobile devices have much less performance in both network and processing aspects, so quality of the

video chat may be unstable when the mobile device is moving around. The ideal condition is that the mobile device has Wi-Fi connection while using video chat system, but the application should provide stable video chat service without Wi-Fi connection as well.

## 3.2 Sensor Network of GiraffPlus project

The application in this thesis is developed under the assumption that using an existing sensor network developed as a part of GiraffPlus project.

The GiraffPlus system uses a number of sensors placed in patients' house and collects both physiological and environmental data. The collected data are intelligently visualized to for example diagram and graphs to provide useful indications to caregivers. Caregivers can see those indications through their laptop whenever they want and it gives caregivers ability to care their patients more convenient. The system can also rise alarms and send warnings for instance in case of falls or in case of abnormal physiological parameters [71].

The monitoring of physiological measurement are using sensors from IntelliCare Company. In particular Intellicare has developed the Look4MyHealth kit. Look4MyHealth is a system for remote monitoring of several biomedical parameters for example blood pressure, blood glucose, weight, oxygen saturation and temperature, and well-being monitoring for example daily routine, aid requests, and fall detection, in the home environment [72].

On the other hand, the GiraffPlus system is using sensors from Tunstall Company in order to collect environment measurement. Tunstall has also developed AD-life - activities of daily life monitoring solution. As the name represent, AD-life provides caregivers context recognition function, which monitors the residence's activity during the day and night, and the collected date are plotted over time to give graphical views to how change happens over time. The sensors used by AD-life includes door opening and closing sensors, which shows how many times are the doors used, bed/chair sensors for monitoring resident's location and activities, and electric usage sensors, which alert if an electrical device as a toaster is forgotten. There are also environmental sensors for fire detection, fall sensors and gas detection sensors [71][72].

The sensors of Giraffplus system are using different wireless communication technologies as Bluetooth, wi-fi and ZigBee to collect data. The collected data sent by those sensors are acknowledged by the gateway that also supports Bluetooth, wi-fi and ZigBee. The Gateway can use GPRS, 3G and Ethernet to send further the data. It works by connecting an Ethernet wire to the gateway or simply by configuring the Wi-Fi option in the gateway. The data sent from gateway are then divided into two types, long-term data and real-time data. Some real-time data are sent directly through the middleware for alarms and long-term data are sent to be stored in the server [71][72].

## 3.3 Basic Structure of Android Application

The android application has a basic structure, which consists of 5 components, Android part, Server part, Database part, Video Chat part and Robot Controlling part. The reason we divide the system into these parts is that each part is developed in different environment and use different technologies, for example the server part is developed by using Java and Spring Framework, while the Video Chat part is developed as a hybrid application by using HTML and JavaScript.

The general structure of the application is shown in figure 3.1. Server part and database part are connected to each other and work together as a database server in order to transfer the personal data stored in the database. The main purpose of the server part is transferring of data between android part and database part. In addition, since the application requires user id and password to login, the security system is implemented for protecting user's privacy.

Video chat part and robot controlling part are developed as a web application first, and then integrated in android part by using Cordova technology. Video Chat part allows user to connect with a chosen elderly person and enables video chat. Robot controlling part makes connection with TurtleBot via workstation and use ROS technology to interact and control the robot.

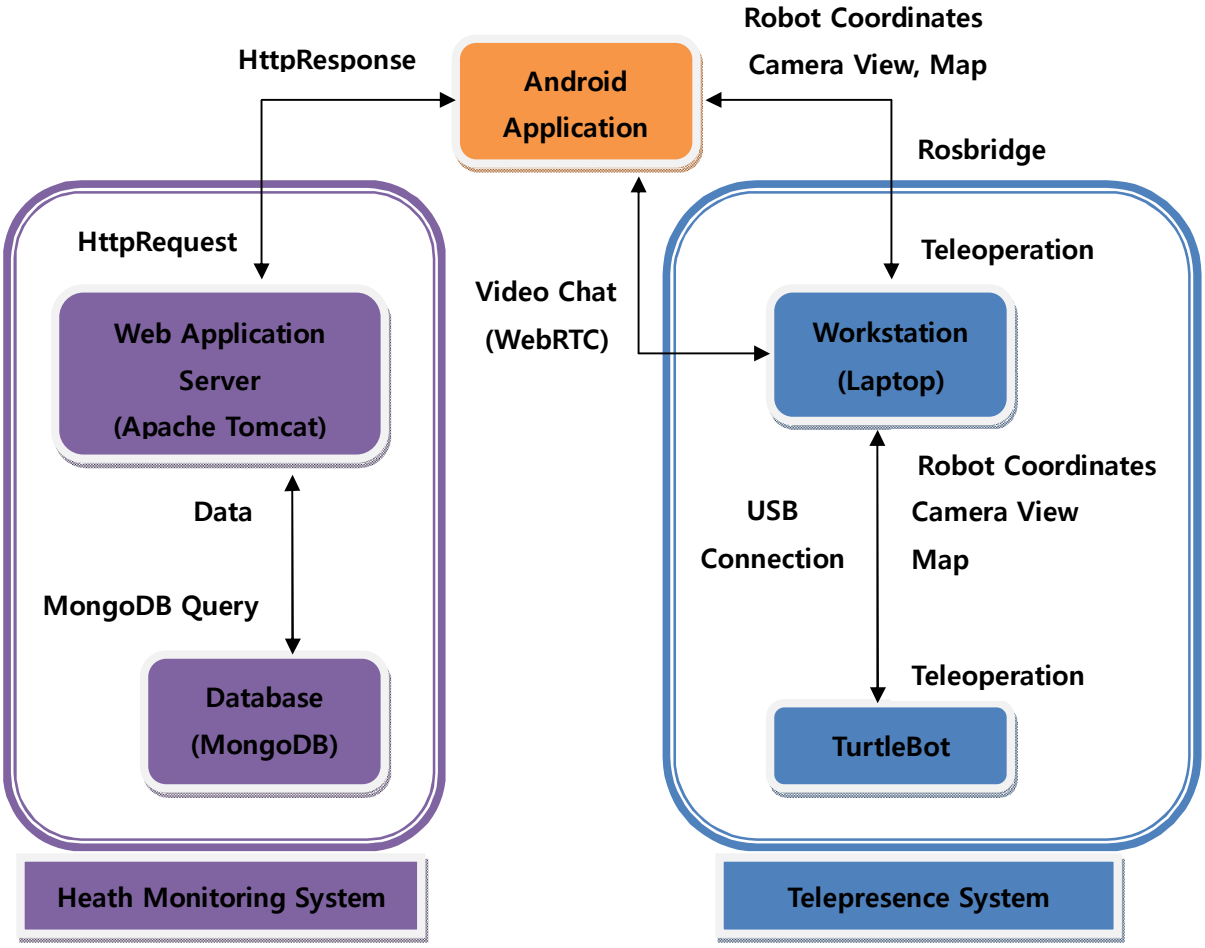



Figure 3.1 The general structure of the application

## 3.4 Overview of the Developed Android Application

This section introduces a simple overview of the developed android application before describing deeply about development of the application

Since the android application is designed to have mainly two systems: Health Monitoring System and Telepresence System. The name of the application is decided as, “HMT”, an acronym of the two systems. The figure 3.2 shows the log in page and main page of the HMT.

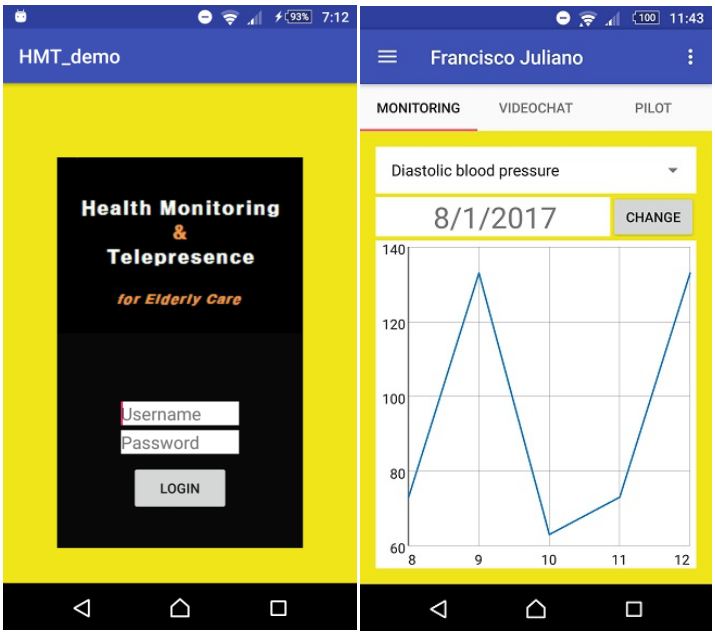



Figure 3.2 Login page (left) and Monitoring page (right) of the HMT

The application starts with login page (or activity in the android application) that requires valid id and password for using other functions. If the user logs in to the application the user can receive the personal data of elderly people who the user has responsibility to take care of.

In the main page of the application, a navigation bar is implemented with three elements: Monitoring, VideoChat and Pilot. Each element is connected to different services we have developed.

In the monitoring page the user can decide measurement type and period for requesting specific measurement data to the server, and received data is visualized in the application as a

graph. As we mentioned in chapter 2.3.5, the mobile phones has much smaller screen than laptop, so the graph should have a size that fits with the mobile screen.

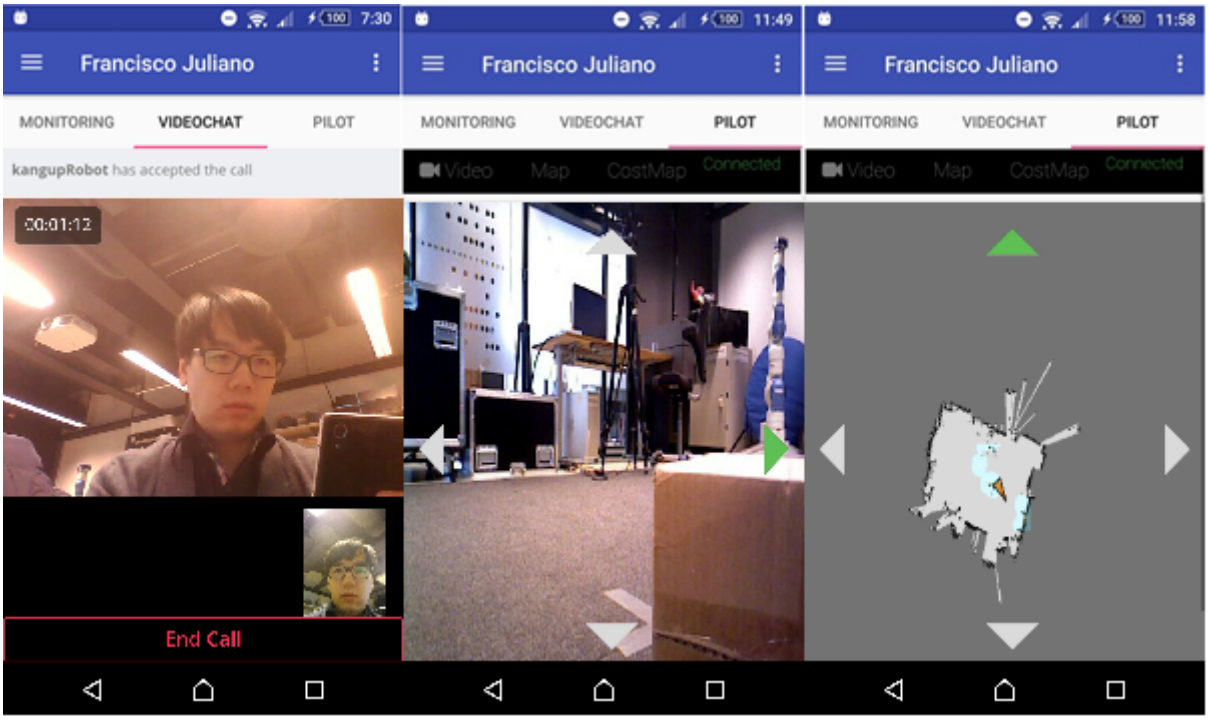



Figure 3.3 Video chat (left), Pilot with camera view (middle), and map view (right)

In the video chat page the user can initiate call to chosen elderly person and if the elderly person receives the call by using video chat web application, then video chat between two clients starts by using WebRTC technology with Skype-like interface.

Pilot system can be started from "Pilot" tap in the navigation bar. When the service starts, the application try to connect with TurtleBot using ROS. When the connection established, the application screen displays a front camera view of the TurtleBot. The display can be switched between the camera view and a map view. The robot can be controlled by using a virtual controller in the screen, and if the user touches a point on the map, the robot is navigated autonomously to the point.

# 4 Health Monitoring System

The health monitoring service is one of the core services of the application. The fundamental idea comes from the GiraffPlus project, as mentioned above. The system is designed as combination of 3 parts: server part, database part and android part. The figure shows the general structure of the Health Monitoring System.

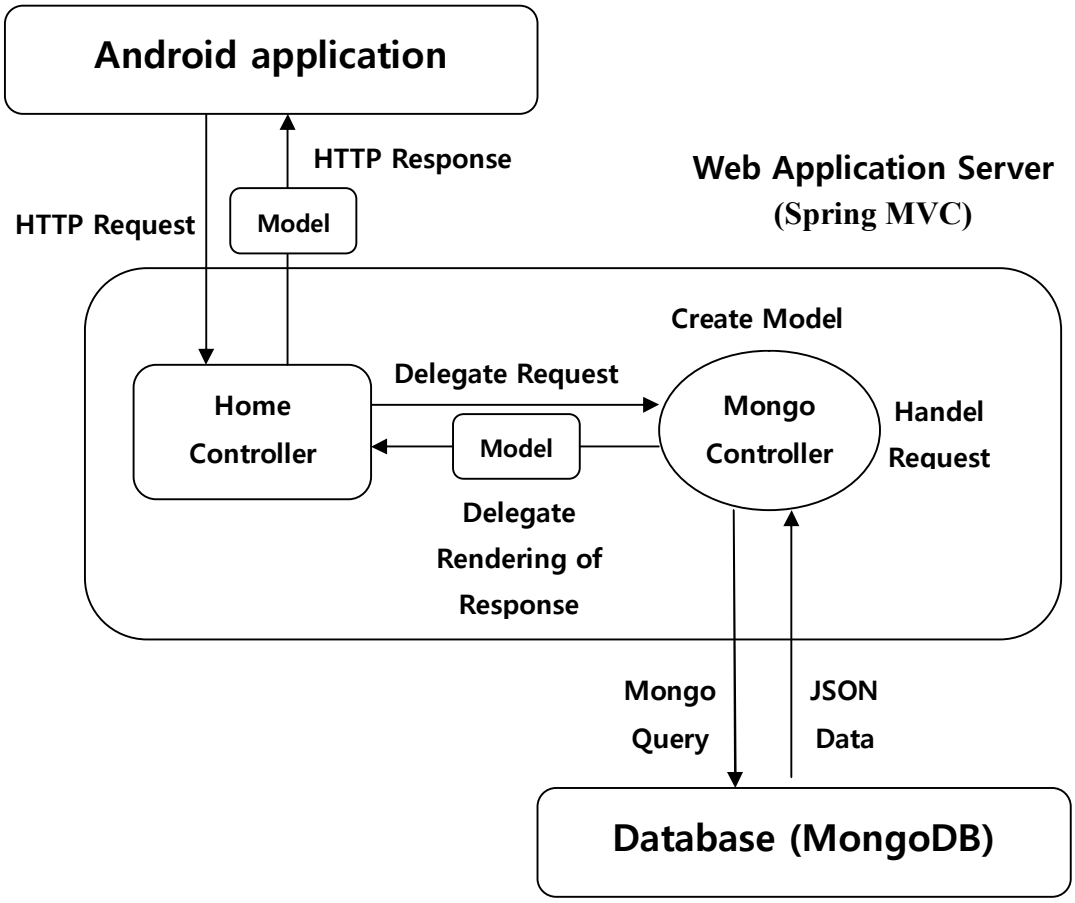



Figure 4.1 General structure of the Health Monitoring System

Since each part is developed by using different technologies, we need to look through which technologies are used before describing real development of the system.

## 4.1 Technologies

### 4.1.1 Web Application Server(WAS)

Before we even start to talk about Web Application Servers, let's look at term "web server" in general. A web server is generally a program that get request from clients and respond their requests using the Hypertext Transfer Protocol (HTTP). A protocol is a standard set of rules, which allows a server and clients to communicate that they must use the same protocol to communicate [55].

An Application Server is a term that sometimes is mixed with a web server. While a Web server mainly deals with sending HTML for display in a Web browser using mainly HTTP protocols, an application server provides access to business logic for use by client application programs through various protocols, including, but not limited to HTTP. The client application program can use this logic just as it would call a method on an object (or a function in the procedural world) [56].

Nowadays the application servers usually work together with web servers. It allows them to do more than just respond to simple requests for static documents, and many provide easy-to-use graphical user interfaces for administration and customization.

If we look at the request-response flow between client, web server and application server (see figure 4.1) when a client has requested for some other resources than normal web pages, such as database, the client's request first goes to the web server, which sends the required information to the application server. The web application accesses the databases servers to perform the requested task updating and retrieving the information lying within the database. The web application then presents the information to the user through the browser [57].

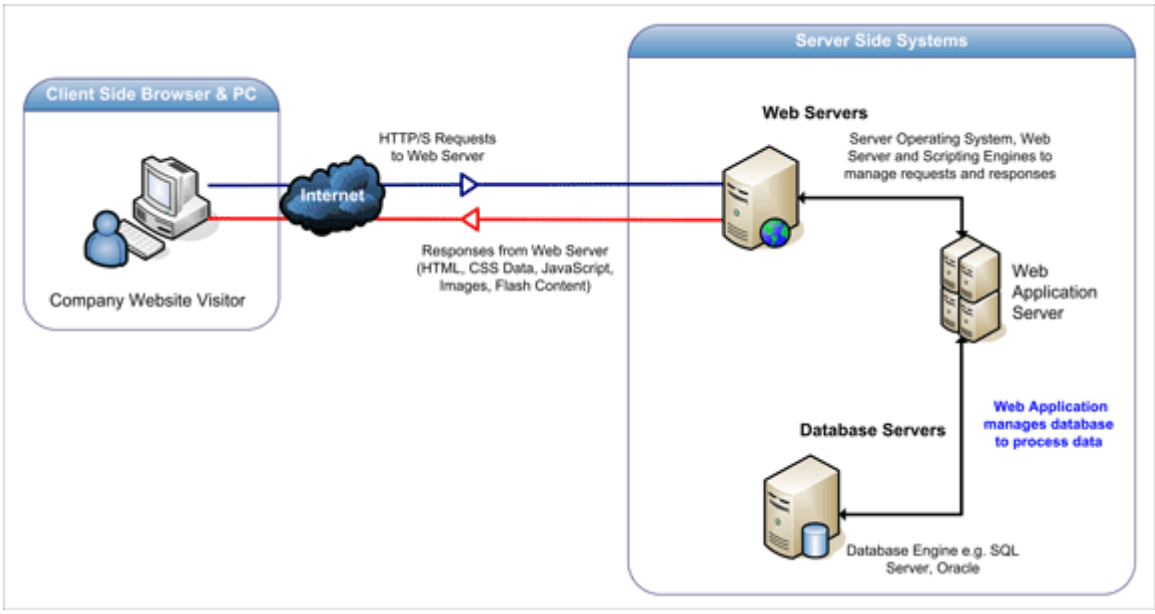



Figure 4.1 Workflow of a Web Application Server

The world's most used web application server software is Apache Tomcat, which is combination of web server Apache and Java application server Tomcat. As of July 2016, Apache Tomcat was estimated to serve 46.41% of all active websites [58], and Tomcat was estimated 63.9% in "Java application server market 2017" [59].

### 4.1.2 Apache Tomcat

As mentioned previously, in this thesis, world's most used web application server software Apache Tomcat is used. The Apache Tomcat server is an open source, Java-based web application container that was created to run servlet and JavaServer Pages (JSP) web applications. It is developed by the Apache Software Foundation and can be installed on almost all operating systems including Linux, UNIX, Windows, FreeBSD, Mac OS X and more [60].

As the web application server, the Apache Tomcat is combination of web server software Apache and application server Tomcat. The Apache Web server is cross-platform, lightweight, robust, and used in small companies as well as large corporations. The Apache, compared with other web server, has almost endless possibilities, due to its ability to pick and

choose various modules, which allows it to be integrated with numerous other applications. Moreover majority market share of the Apache has contributed strong user-community support. Meanwhile, Tomcat, as a web application server, provides additional functionality that makes it a great choice for developing a complete web application solution [61].

### 4.1.3 Spring Framework

The Spring Framework is a popular open source application framework that provides comprehensive infrastructure support for developing Java applications. It consists of a container, a framework for managing components, and a set of snap-in services for web user interfaces, transactions, and persistence. For developing the application, we can benefit from the Spring Framework with powerful and flexible collection of technologies such as :

- **Data access:** Support for traditional RDBMS as well as new NoSQL solutions (MongoDB), map-reduce frameworks and cloud based data services.
- **Security:** Authorization control for all tiers and authentication integration to dozens of providers.
- **Frontends:** Complete support for modern web technologies including REST, HTML 5, conversations and AJAX.

The Spring Framework is responsible to provide fast and secure access to the central data storage. It will expose RESTful web services and also host the CRUD web interface (CRUD - create, read, update, and delete).

The Spring Framework consists of features organized into about 20 modules to achieve different services and functionality for development of application, some popular modules are Spring Core(Base module), Spring JDBC(for persistence logic), Spring AOP(for cross context logic), Spring Transaction(For transaction support), Spring ORM and Spring MVC(Web aspect) [53][62]. However, the users do not have to use all of these modules. They are designed in such a way that no module is dependent to other module, except Spring core module. So based on their needs, Spring allows users to use only those parts that they need, without having to bring in the rest. In terms of web application, the Spring MVC

provides a full-featured MVC(Model View Controller) framework for creating web applications [53].

Spring MVC is a complete HTTP oriented MVC framework managed by the Spring Framework and based in Servlets. The most popular elements in it are classes annotated with @Controller and @RequestMapping annotations, where we implement methods we can access using different HTTP requests. With the introduction of Spring 3.0, the @Controller mechanism also allows us to create RESTful Web sites and applications, through the @PathVariable annotation and other features. Following figure shows the architecture diagram for Spring MVC [53][54],

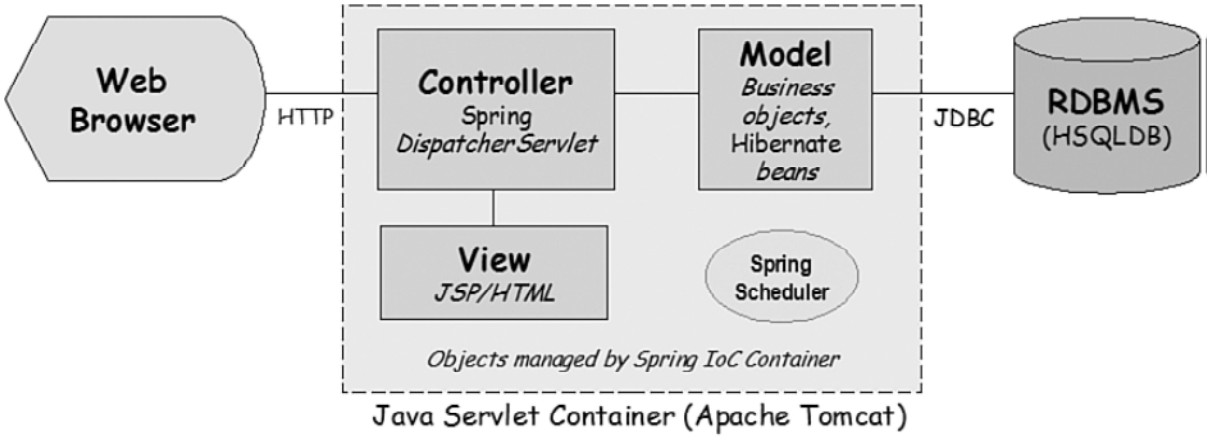



Figure 4.2 High-level architecture diagram for Spring MVC [54]

As we can see, all incoming HTTP requests from a web browser are handled by Controllers. A controller, as the name indicates, controls the view and model by facilitating data exchange between them. The key benefit of this approach is that the model can worry only about the data and has no knowledge of the view. The view, on the other hand, has no knowledge of the model and business logic and simply renders the data passed to it (as a web page, in our case). The MVC pattern also allows us to change the view without having to change the model.

### 4.1.4 REST API

Representational state transfer (REST) or RESTful Web services are a way of providing communication between computer systems on the Internet. RESTful Web services allow requesting systems to access and manipulate Web resources using a uniform and predefined set of stateless operations. “Web resources” were first defined as documents of files identified by their URLs, but today they have a much more generic and abstract definition as everything or entity that can be identified on the Web.

In a RESTful Web service, requests made to a resource's URL will elicit a response that may be in XML, HTML, JSON or some other defined format. The response may confirm that some alteration has been made to the stored resource, and it may provide hypertext links to other related resources or collections of resources. Using HTTP, as is most common, the kind of operations available include those predefined by the HTTP verbs GET, POST, PUT, DELETE and so on. By making use of a stateless protocol and standard operations, REST systems aim for fast performance, reliability, and the ability to grow, by re-using components that can be managed and updated without affecting the system as a whole, even while it is running [63][64].

HTTP-based RESTful APIs are defined with the following aspects:

- base URL, such as http://api.example.com/resources/
- an internet media type that defines state transition data elements, for example Atom, microformats, application/json, etc. The current representation tells the client how to compose requests for transitions to all the next available application states. This could be as simple as a URL or as complex as a Java applet.
- standard HTTP methods (e.g., OPTIONS, GET, PUT, POST, and DELETE)

### 4.1.5 MongoDB

MongoDB is an open source cross-platform document-oriented NoSQL database system, which stores structured data as JSON-like documents with dynamic schemas, making the integration of data easier and faster. MongoDB was founded in 2007 and since its inception. MongoDB has been downloaded 15 million times and is supported by more than 1,000

partners [28].

As a document-oriented database, MongoDB stores semi-structured data and descriptions of that data in document format. It allows developers to create and update programs without needing to reference master schema. MongoDB documents are similar to JavaScript Object Notation (JSON) objects, a data interchange format that has gained wide currency among web application developers, although XML and other data formats can be used as well. Document databases are used for content management and mobile application data handling [69].

NoSQL databases are useful for large sets of distributed data that relational databases aren't built to solve. They are most effective when an organization must analyze large chunks of unstructured data or data that's stored across multiple virtual servers in the cloud.
Furthermore, MongoDB provides high availability with replica sets. All the stored data will be automatically replicated to two or more copies. It provides automatic failover and automatic recovery of the data.

## 4.2 Development of Database Server

The first attempt to develop the Health Monitoring System is developing Database Server by creating the Server part and the Database part in parallel. A database server is simply the server connected with a database. The server is developed as a web application server by using Apache Tomcat software. The web application server is written in java and developed in Spring Framework + Maven environment by using Eclipse IDE (integrated development environment). The combination of Eclipse and Spring Framework is very popular choice for developing a web application server which is also called Spring MVC (Model-View-Controller) application [54]. This combination provides various plug-ins or dependencies for web server developing. For instance it allows the user to easily integrate Apache Tomcat web server technology in the project and provides REST API which provides simple way for request-response transactions of data.

The main purpose of the web application server is transferring data between Database and Android application. In order to accomplish this purpose, two controllers are developed in the server part: Home Controller and Mongo Controller. The Home Controller is responsible for communication with the android application, while the Mongo Controller is responsible for communication with the MongoDB database. The source codes of Home Controller and Mongo Controller can be found in Appendix B.

In order to guarantee the data security, the server is protected by combination of basic Spring Security Configuration [65] and JSON Web Token (JWT) [67] technologies that provide the user specific authorization token for each user when the user log in, and the user has to verify this token every time when the user wants to access to other services in the application.

In addition the server includes model structure that is used when the server receives data from the database and transfer data to the android application. The model structure is based on structure of the database, which will be described detailed in the next section. The structure of the project for server part and important source codes are introduced in Appendix A

### 4.2.1 Creating Demo Database

All the data including caregivers' username and password, and elderly people's physiological data are stored in the database. In the real world, the database requires sensors which collect various data from elderly people's home environment and send them to database. But in this thesis we assume that we use same sensor network as the GiraffPlus project (see chapter 3), so the monitoring system uses a demo database that is created for testing the application. The demo database is based on the database structure described in the document provided by GiraffPlus project [102]. The following UML shows basic structure of our demo database.

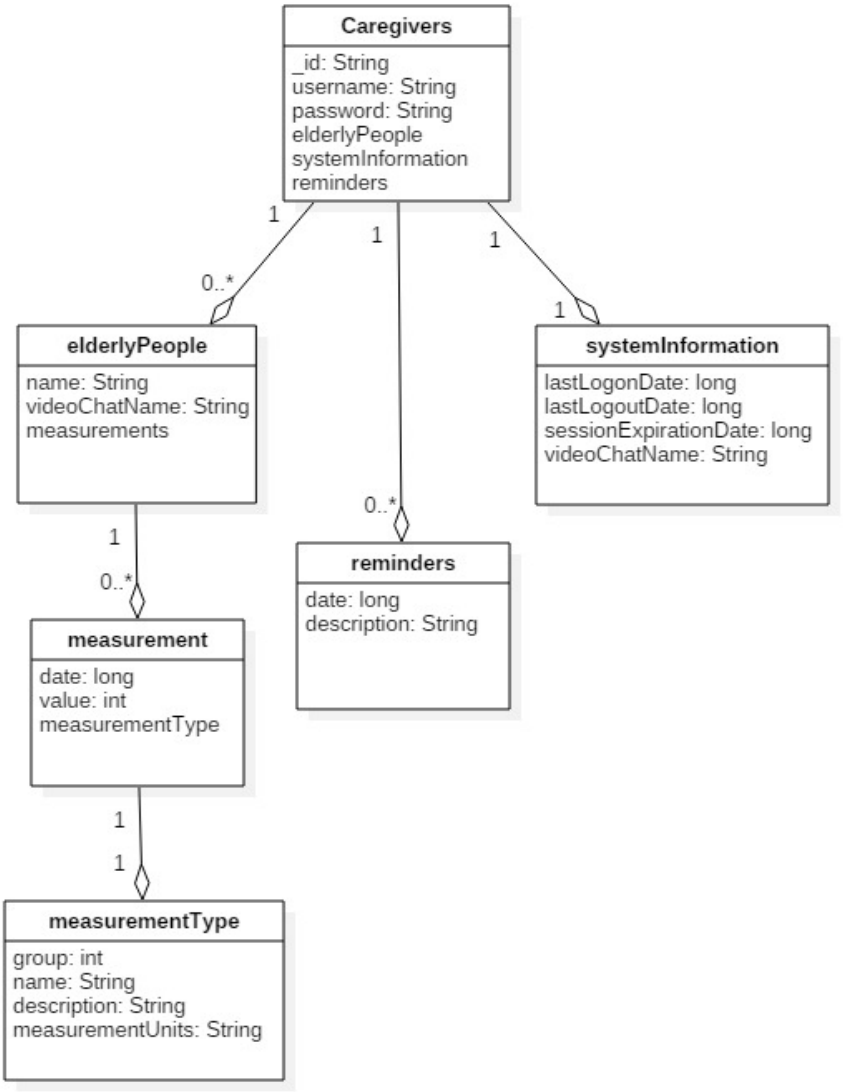



Figure 4.3 Basic Structure of the Demo Database

The demo database contains collections with JSON documents that contain user data for caregivers. Caregivers have their unique username and password that they use to log in to the application. Each caregiver has link to elderly people who the caregiver has responsibility to take care of. The most important data for our system is “measurement” since it contains physiological data for elderly people. The example of the "measurement" data is shown in the following example:

```
{
        "date" : NumberLong(1483574400000),
        "value" : 79.0,
        "measurementType" : {
                "group" : 3,
                "name" : "HR",
                "description" : "Heart rate",
                "measurementUnits" : "bpm"
        }
},
```

Example 4.1 The example of the "measurement" data in the database

The date value is indicated in the numerical value corresponding to the number of milliseconds since 1 January of 1970 00:00 GMT. For example [102]:

- "2012-09-12T15:24:47.657+0100" -> 1347459887664

The “measurementType” represents literally which type of the measurement the value belongs to. In the demo database there are 3 measurement types: Heart rate, Diastolic blood pressure and Systolic blood pressure. Appendix B contains an example of the demo database.

### 4.2.2 Communication Between Server and Database

In order to use the MongoDB database, the MongoDB server has to be started first. Starting the MongoDB server is arranged by executing mongodb.exe in the command line, and by default, the server starts at port 27017 [101].

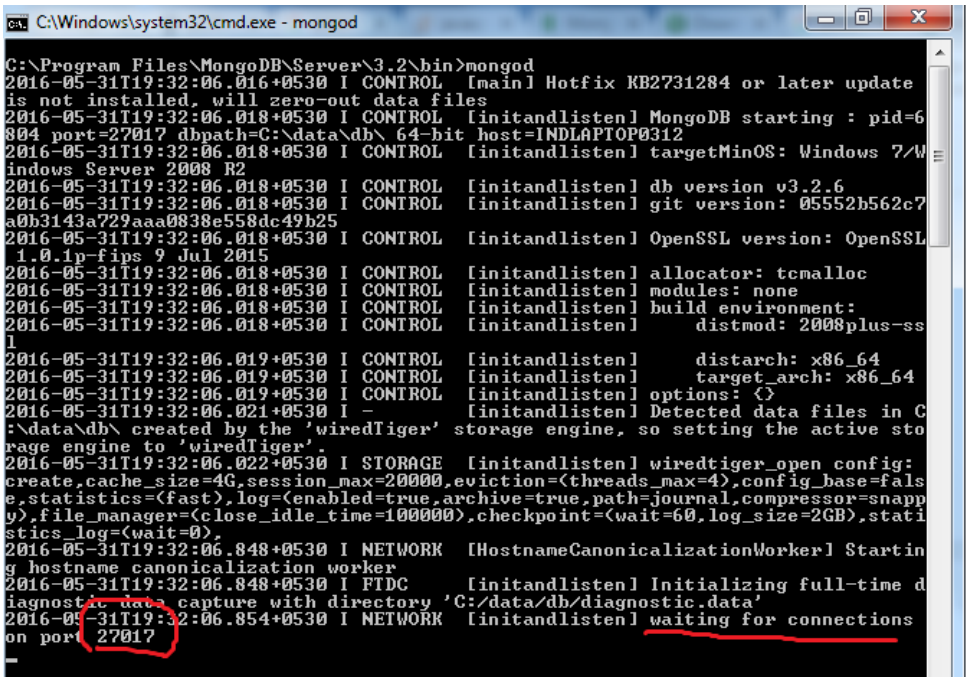



Figure 4.4 Starting MongoDB Server at Port 27017

In order that the server communicates with the database, “MongoTemplate” class is used in the server part, which is provided by Spring Framework. For using the "MongoTemplate", the class is declared in the "MongoController" with its configurations (see example 4.2). The "ServerAddress" contains host address and port of the MongoDB server. The name of the database, "hmt_database" is assigned in "Mongo DB Factory" [72][73].

```
ServerAddress serverAddress = new ServerAddress("localhost", 27017);

MongoClient mongoClient = new MongoClient(serverAddress,Arrays.asList(credential));

// Mongo DB Factory
SimpleMongoDbFactory simpleMongoDbFactory = new SimpleMongoDbFactory(
                                                    mongoClient, "hmt_database");

MongoTemplate mongoTemplate = new MongoTemplate(simpleMongoDbFactory);
```

Example 4.2 MongoDB configurations in MongoController class

By using the MongoTemplate simple Mongo query operations can be implemented like the following codes [74]:

```
Query query = new Query();
query.addCriteria(Criteria.where("username").is("testUser").and("password").is("testPass"));

Caregiver user = mongoTemplate.findOne(query, Caregiver.class, "Caregivers");
```

Example 4.3 Mongo query operation "findOne"

In example above, we create a query class that specifies certain document which has unique username (‘testUser”) and password (“testPass”). Then, the “findOne” method performs query on the “Caregivers” collection in the database, and create a “Caregiver” class (model) that contains results from the performed query.

The “MongoTemplate” provides not only “findOne” method, but also other methods that provide simple CRUD operations. For example we can find a certain caregiver and update his/her password to a new password (newPass) with following codes:

```
Query query = new Query();
query.addCriteria(Criteria.where("username").is("testUser").and("password").is("testPass"));

Update update = new Update();
update.set("password", "newPass");

mongoTemplate.updateFirst(query, update, "Caregivers");
```

Example 4.4 Mongo query operation "updateFirst"

By using these methods the server is able to implement almost all services that we need in this project, but there is still problem we have to consider that the methods described above are not able to perform more complicated query.

With the simple CRUD operations we described above, when we needs to find more specific data, such as “finding an array of measurement values of certain elderly person in certain period”, we have to receive all the data of the caregiver (document) from the database including all unnecessary data that we don't need to know, and then we have to find the data we want by using for example for-loop and if-statements in the server part. Actually this kind

of data is needed when we visualize a graph for providing certain measurement data to the caregiver. But if the caregiver has a large amount of data of for example 100 elderly people in the database, there may be some server lag every time the application receives measurement data from the server. So the server is supposed to be able to get only specific part of the data that the user actually want, not whole document.

To resolve this problem, the server part employs "Aggregations" operations provided by MongoDB [103]. The aggregation operations group elements from multiple documents together, and perform a various operations on the grouped data to return a single result. The aggregation operations are performed like a pipeline that consists of multiple stages. Each stage transforms the documents by using various operators as the data passes through the pipeline with filters. The operators used in the pipeline are called "Stage Operators". In this thesis we use mainly 4 stage operators:

- Match: Filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage.
- Unwind: Deconstructs an array field from the input documents to output a document for each element. Each output document is the input document with the value of the array field replaced by the element.
- Sort: Sorts all input documents and returns them to the pipeline in sorted order. Reorders the document stream by a specified sort key.
- Group: Groups documents by some specified expression and outputs to the next stage a document for each distinct grouping.

By using the aggregation operations, an example described above, "Finding an array of measurement values of certain elderly person in certain period", is performed with following codes:

```
AggregationOperation match1 =
                Aggregation.match(Criteria.where("username").is(secondaryName));
AggregationOperation unwind1 = Aggregation.unwind("primaryUsers");
AggregationOperation match2 =
                Aggregation.match(Criteria.where("primaryUsers.name").is(primaryName));
AggregationOperation unwind2 = Aggregation.unwind("primaryUsers.measurements");
AggregationOperation match =
                Aggregation.match(Criteria
                .where("primaryUsers.measurements.measurementType.group").is(group)
                .andOperator(Criteria.where("primaryUsers.measurements.date").gte(date),
                                Criteria.where("primaryUsers.measurements.date").lte(endDate)));

AggregationOperation sort =
                Aggregation.sort(Sort.Direction.ASC, "primaryUsers.measurements.date");
AggregationOperation group_query =
                Aggregation.group("null").push("$primaryUsers.measurements.value")
                .as("output_measurements");

Aggregation aggregation =
        Aggregation.newAggregation(match1,unwind1,match2,unwind2,match,sort,group_query);

AggregationResults<QueryOutput> result =
                mongoTemplate.aggregate(aggregation, "Secondary_Users", QueryOutput.class);
```

Example 4.5 The process of the Aggregations operations

In the example 7 operators are used in order of i) match, ii) unwind, iii) match, iv) unwind, v) match, vi) sort and vii) group. The detailed processes of the operations are:

i) First operation "match" filters the documents to find only the documents that have the username of the user. Since the username is a unique element in the database, the result will be a document with data of the user.

ii) Operation "unwind" deconstructs "primaryUsers" field and return several identical documents which have each elements of "primaryUsers" field.

iii) Operation "match" filters deconstructed documents and returns only a document which have specific primary user name.

iv) Operation "unwind" deconstructs "measurements" field in the "primaryUsers" field and returns several identical documents which have each "measurements" elements.

v) Operation "match" filters deconstructed documents and returns the documents which contain certain type of measurement values that are measured in certain period.

vi) Operation "sort" sorts out documents by "date" value in "measurements" elements.

vii) Operation "group" makes a new document that contains measurement values specified by operations.

The operations are performed by “aggregate” method and the result will be returned as an array of the measurements values.

## 4.3 Development of Android Part

In the monitoring system, the android part is responsible for handling general front-end services of the application, including design of the user interface. The main purposes of the android part in the monitoring system are: i) Requesting certain data the user wants to receive, and ii) Visualizing received data with a graph. The android part is written in native code by Android Studio, an official IDE for Android operating system announced on 2013 by Google.

### 4.3.1 UI of Android Application

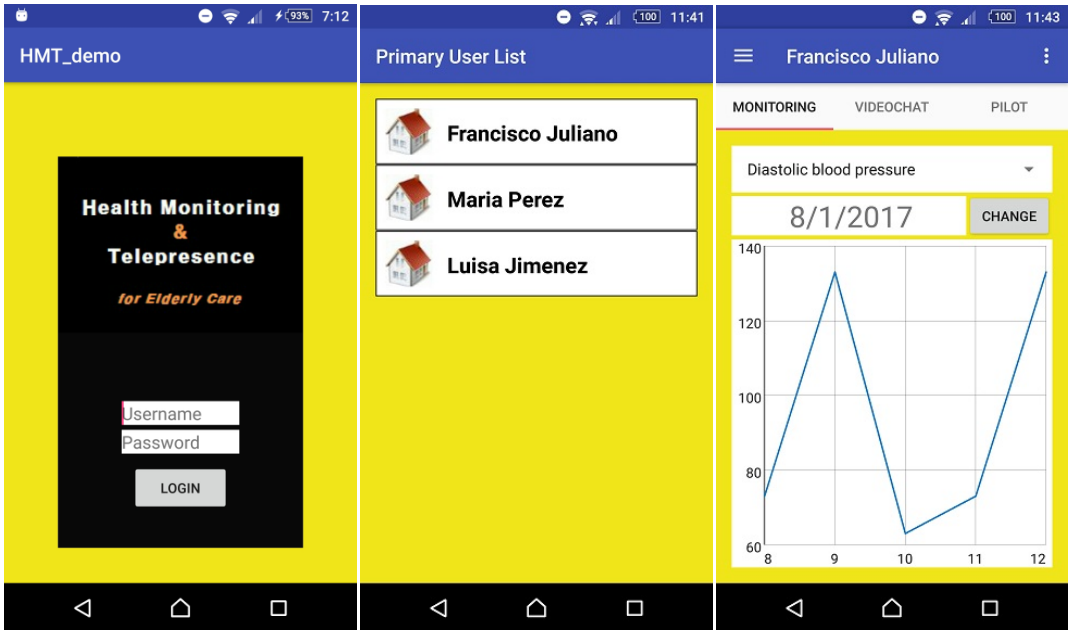



Figure 4.5 Basic UI of Android Application

Since the fundamental idea of the application comes from DVPIS software of GiraffPlus project (see figure 2.2, 2.3 and 2.4), the basic UI of the application is very similar with it. The monitoring service of the application includes the following components:

① Basic login system that requires username and password.

② List of elderly people currently under authorized access by the logged user. The list shows each elderly people's name.

③ Tap bar which has 3 taps for monitoring service, video chat service and pilot service. The tap can be changed by clicking or touching one of elements in the tap, or swiping the screen from side to side.

④ Graph that displays chosen elderly person's physiological data. X-axis presents date of the each point and y-axis presents the value of the data. Each axis displays most 5 points in order that user's readability.

⑤ Area where the user can change type of the physiological data with top-down menu, and date value from calendar.

⑥ Navigation bar which has three elements that lead to other functions.

⑦ Navigation drawer, a panel that displays several services on the left edge of the screen. It is appeared when the user touches the icon in the action bar (see the figure 4.7).

The design of the interface and the interaction with the application has the clean and simple feeling that was planned. The design lacks some testing from users outside of the development team and consequently some additional rework has to be accounted for in future iterations.

### 4.3.2 REST API in Server Part

The request-response transactions between android application and the server are performed in “HttpRequests” class of the android part and “HomeController” class of the server part. Access to the service will be allowed by using REST calls. Calls consist of a base URL of Apache web server, “http://localhost:8080/”, with addition of the resource identifier and the parameters used in the call. The data encapsulation supported is JSON. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own. The body of the message is typically the requested resource, although an error message or other information may also be returned.[2]

The "HomeController" class in the server part contains several methods that respond to HTTP requests from the android part. When the server receives a HTTP request, the server matches the URL of incoming request with access URL of the methods, and assigns the request to the matched method. Some important methods created in “HomeController” class are following:

- Access URL: {baseURL}/authenticateUser/{username}/{password}
  Description :
  GET method responsible for authenticating the user. Two string parameters “username” and “password” are provided from login page in the application. If the authentication successes, the server creates a temporary token providing the user identification when executing the other service calls. The token will be included in the header of HttpResponse and sent back to the application. The method returns a string array with elderly people’s names as well.

- Access URL : {baseURL}/getMeasurementsSetting
  Description :
  GET method responsible for obtaining the various data that are necessary to make monitoring activity in application. The method returns “MeasurementSetting” class that contains variables that we need to set up the graph

- Access URL : {baseURL}/getMeasurementsByPeriod/{group}/{date}/{periodType}
  Description :
  GET method responsible for obtaining the specific type of measurements in the certain period. Parameter “group” is number of measurement type the user wants to find. Parameter “date” represents start date of time period. Parameter “periodType” represents how many values the user wants from the start date. In the demo database the “periodType” are fixed with 5. The method returns int array with the values of measurements.

### 4.3.3 REST API in Android Part

The HTTP requests from the android application are sent by using "RestTemplate" class provided by Spring Framework. The "exchange" method of the "RestTemplate" provides simple way to send HTTP request to the server and receives the HTTP response automatically. The example codes of using "RestTemplate" are:

```
RestTemplate restTemplate = new RestTemplate();
restTemplate.getMessageConverters().add(new MappingJackson2HttpMessageConverter());
HttpHeaders header = new HttpHeaders();
header.add("Authorization", token);
header.add("primaryUser", primaryName);
HttpEntity<String> httpEntity = new HttpEntity<String>(header);

ResponseEntity<int[]> response =
restTemplate.exchange(baseURL+"/getMeasurementsByPeriod/" + group + "/" + date + "/" +
periodType, HttpMethod.GET, httpEntity, int[].class);
int[] measurements = response.getBody();
```

Example 4.6 The example of using "RestTemplate" class

The example codes send HTTP request to the server for calling "getMeasurementsByPeriod" method in the "HomeController" class in the server (see the previous chatper). The "RestTemplate" class includes HTTP headers that contains authentication token and name of the elderly person the user wants to access. In the example, the response from the server contains an array of measurement values and we can get the array by using "getBody" method.

## 4.4 General Workflow of Monitoring System

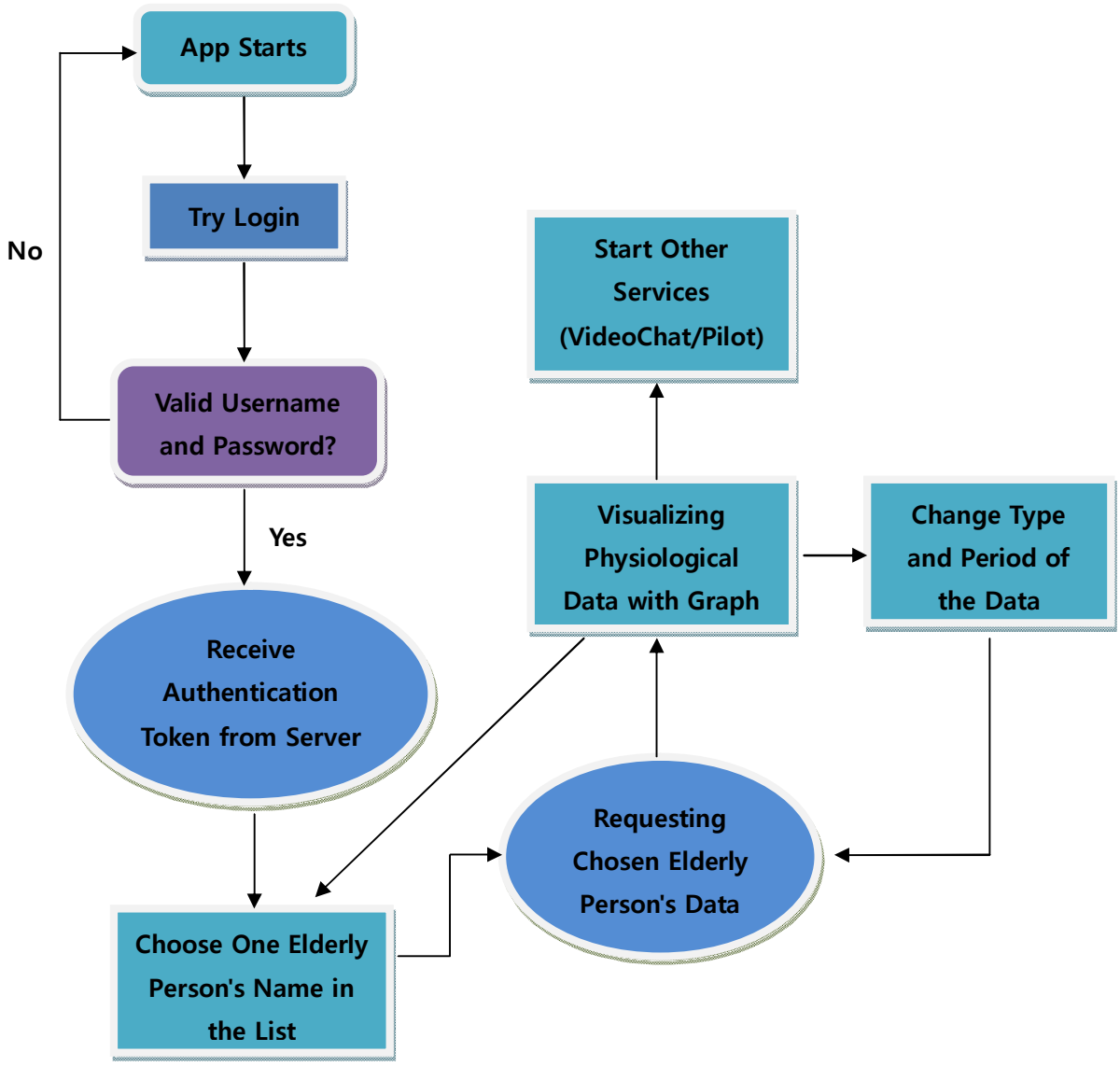



Figure 4.6 General workflow of the Monitoring System

1) Login system which requires valid user id and password

When we start the application the login page is displayed (figure 4.5). Before the user try to login, the web server and database naturally has to be turned on. The login system requires valid username and password, which are stored in demo database. When the user types in the username and password, and click the login button, the application sends request with the inputted username and password to the server. The server communicates with the database and check if the values are valid - in other words check if there are same username and

password in the database, and if the values are verified the server sends response including an array of elderly people's names and a token value as well. If the user types in invalid username or password, naturally an error message appears and the login fails. All other services than authentication service that are performed by server, require the token value the user received when the user login.

2) Choose one elderly person the user wants to access

When the user logs in successful, a list with names of elderly people is displayed in the application. By choosing one of those names, the application automatically requests the latest physiological data of the chosen elderly person to the server.

3) Visualizing physiological data

The application receives response from the server and the received data is visualized as a graph. For readability of the graph, only values of 5 days are plotted in the graph. Type and period of the values can be changed by multiple choices spinner and a calendar above the graph.

## 4.5 Implementation of Reminder Service

As a part of the Health Monitoring Service, a reminder service is implemented. The process of the service is very simple. When the user starts the service by touching "Reminder" in the navigation drawer (see the figure 4.7), reminder window is appeared (see the figure 4.7). The user can adjust the date and time, and the description of the reminder message. If the user touches "confirm" button, the application make a reminder model that contains "date" value with long number and String "description" value. The model is sent to the database via server and stored in the reminders field, as a following example:

```
"reminders" : [
      {
            "date": NumberLong(1510137900000),
            "description": "test description"
      }
]
```

Example 4.7 Data model of "reminders" field in the database

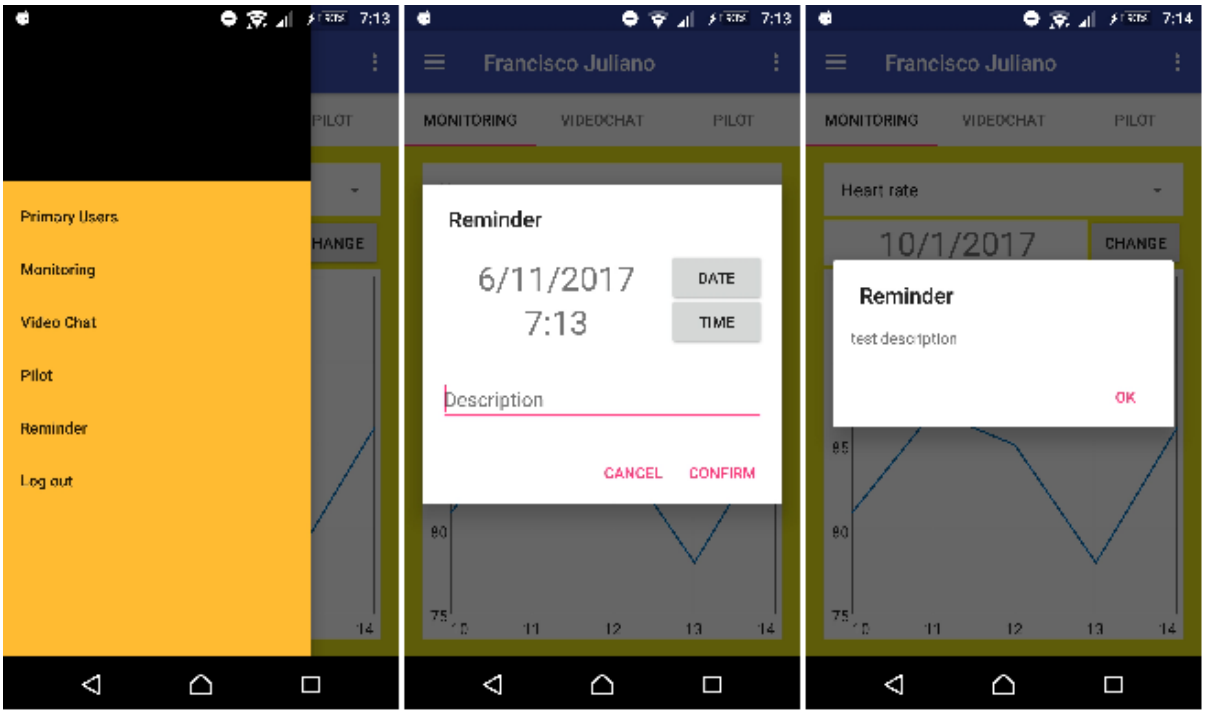



Figure 4.7 Navigation drawer (left), Set reminder window (middle), Get reminder window (right)

After the application sends the reminder data, the application checks continuously if there is any reminder data that has the less date value than current date value. If there is, the application displays reminder window with the description of the reminder data. The figure is showing the result of getting a reminder data.

## 4.6 Hosting the Database Server

As described above the database server is run on the localhost with certain ports. It means if the android application wants to communicate with the database server, the android device need to be connected with the same Wi-Fi network with the server. It's not that we want, and in order to reach one of our goals, "Allowing the caregivers to monitor physiological data of elderly people whenever and wherever they want", hosting the database server is necessary.

Web Hosting Service allows the users to make the server accessible via the standard Internet. In the database server, there are two servers we need to host, the web application server and the mongoDB server, and these two servers will be hosted in different ways.

The web application server is hosted by using Amazon Web Services (AWS) [104]. The AWS provides various services, each of which exposes an area of functionality. In this thesis we use "Elastic Beanstalk" service that allows us to upload our web application source codes, and provisions us a unique host URL for our server. Elastic Beanstalk supports applications developed in various languages such as Java, PHP, Node.js and Python. When we upload our web application source codes as a .zip file, the Elastic Beanstalk service automatically hosts our server with the assigned host URL. The hosted server URL needs to be declared as "baseURL" variable in the "HTTPRequests" class in the HTTPRequests class (see Appendix A).

For hosting the mongoDB server, "mLab" cloud database service [105] provides an easy way to host mongoDB server. The only thing we have to do is uploading the database to the "mLab" service, and the service provisions us a unique host URL for our mongoDB server. As we have seen in the example, the hosted server URL needs to be declared in the MongoController class as ServerAddress class (see Appendix B).

With the hosted database server, we can finally use the health monitoring system via Internet whenever we want.

# 5 Robot Telepresence System

The second goal of this thesis is developing a low cost telepresence system using an android application and existing mobile robot. Most of commercially available robots have specialized hardware for processing activities, and such robots increase the cost rapidly. Therefore, as the main cost reduction method, our design consists of a normal laptop mother board and a processor for the main processing hardware of the robot. In addition, the other peripherals of the laptop such as screen, microphone, speakers, webcam (if equipped), wireless network card are used with the streaming and networking activities at the robot.

The "TurtleBot" is chosen as the robot we use. The application will have services for interacting with the "TurtleBot" via standard internet. Our robot telepresence system requires two main components:

- Mobile robot control via android application (pilot system)
- Videoconferencing system between android application and mobile robot (video chat system)

The result is a hybrid application, which is written in combination of JavaScript and HTML, and integrated into android application by using Cordova software [48]. There are two reasons we use web based application, not using native application. The first one is for using web based open source codes and libraries available in internet and the second one is for allowing users to use the telepresence system both via android application and desktop with the same API. The second reason is also considering elderly people to use the telepresence system at home by using their desktop or laptop.

## 5.2 Development of Web Application for Pilot System

As the first attempt to develop a telepresence system, a web application which interacts with "TurtleBot" using ROS is developed.

### 5.2.1 UI of Web Application

The developed web application has very simple UI that consists of mainly 3 components: Navigation bar, Main view and Virtual controller.


Figure 5.2 Main UI of Web Application

On the top of the application there is a navigation bar which has 2 elements, Video and Map. Each element is connecting to different html pages that display camera view and map respectively from "TurtleBot" in the main view.

The virtual controller is used to control the robot. All html pages have same virtual controller on the main view. The controller has common shape with arrows for each direction e.g. "up" arrow driving the robot forward and "down" arrow backward.

## 5.3 Web Application for Video Chat System

The developed robot controlling system provides real time video streaming, but not audio streaming. Without audio streaming the video chat is not available. Since our system assumes that the caregiver uses android device and the elderly person uses Linux laptop connected with the robot, Skype, a most popular video chat application which is available in both android device and Linux laptop, can be a good solution. But considering allowing caregivers to remotely control the robot while video chatting, it would be better we integrate video chat system in the developed application.

For integrating a video chat service in our previously developed web application above, we will first develop a simple WebRTC based video chat web application, and it will replace "camera view" html page of the previously developed web application. The application for video chat has very simple structure and minimal functions, e.g. calling and receiving a call and exchanging video and audio streaming between two clients. The WebRTC technology actually provides multiple video chat system but in our application we implement only one-on-one video chat between caregiver and an elderly person.

In this thesis we use one of these open source video chat program as basement and develop a simple video chat program which has only essential functions initiating and receiving calls.

The video chat system we develop is going to be used between a smart phone of caregiver and the robot in elderly person's home environment. The call is basically initiated by the caregiver and the elderly person accepts the call the video chat starts. Initiating call from the elderly person is not going to be implemented in this thesis.

In this thesis, in order to implement a signaling server for establishing peer to peer connection between two clients, an open source back-end API provided by Quickblox Enterprise is used. The web application is based on open source sample application that uses Quickblox JavaScript SDK [107]. The sample application provides simple WebRTC video chat service with following UI:

The sample application allows user to make a room with specific name. The user can make a

new room or can join the room made by anyone else. When a participant enters a room and if there are other participants in the same room, they start to exchange signaling messages each other in order to begin a WebRTC session. The sample application exchanges signaling messages by using Quickblox signaling protocol with "QB.createSession" method [107].

### 5.3.1 Development of Video Chat Web Application

I'm not going to explain deeply about my codes of web application in this thesis because I didn't modify much from the sample codes. There are mainly two things I modified from the sample application:

- Deleted unnecessary UI and functions for making the application simple that has only minimum functions for performing one-on-one video chat.
- Assigned name of the room and user names for both caregivers and the elderly people.

The UI of the modified web application is shown in the following figure:

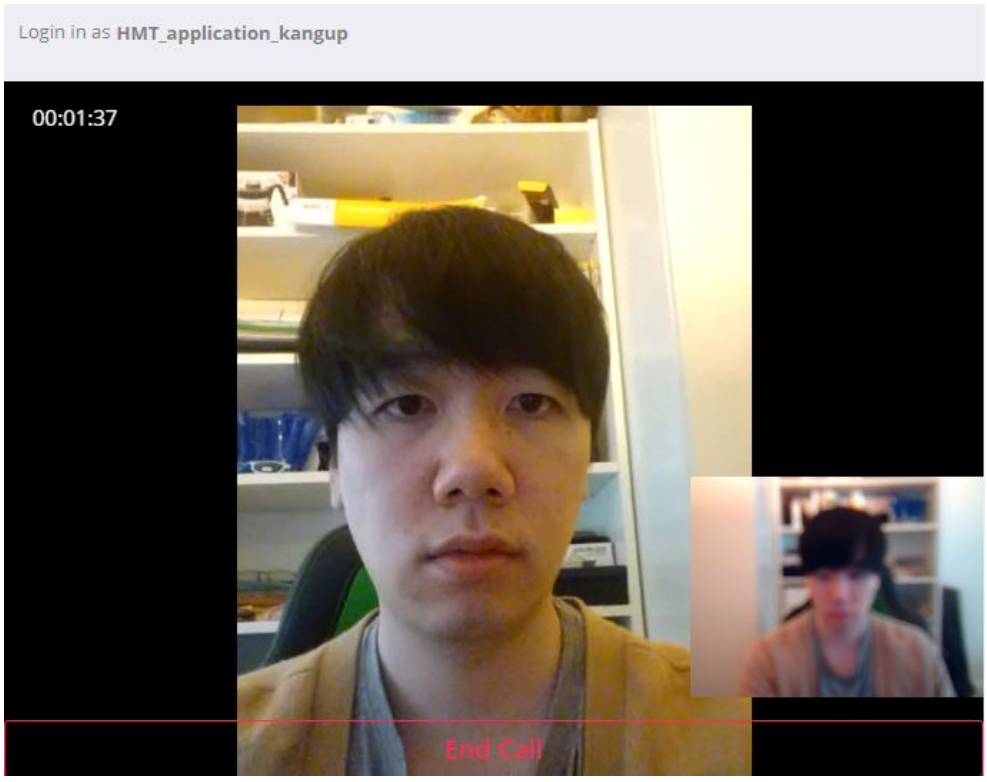



Figure 5.11 Developed video chat web application

Firstly, in the modified web application user don't need to type in both name of the room and user name when they enter the room. In this thesis the name of the room and usernames for caregivers and elderly people are assigned like [90][91]:

- Room name: HMT_room
- Caregiver's username: HMT_application_{user name of the application user}
- Elderly person 's username: HMT_robot_{last name of elderly person}
  (For example "HMT_robot_Francisco")

The assigned names for both caregivers and elderly people are stored in the database with name "videoChatName".

For evading confusion from calling to wrong person, the client doesn't have function to choose which person the user will initiate call to, but the client initiates call automatically to the elderly person, who is chosen from the list, when the user enters the room.

For performing the video chat with the developed application the two clients need to be turned on. In this thesis one client is integrated in android application for the caregiver and the other one is web application which will be turned on in Internet browser such as Firefox or Chrome browser.

### 5.4.1 Testing in Real World

To confirm that the developed telepresence system works well in the real world, the real TurtleBot was used. The test is performed by using developed android application without Wi-Fi connection. As a result the both video chat service and pilot service worked exactly the same as in the simulation without any problems.

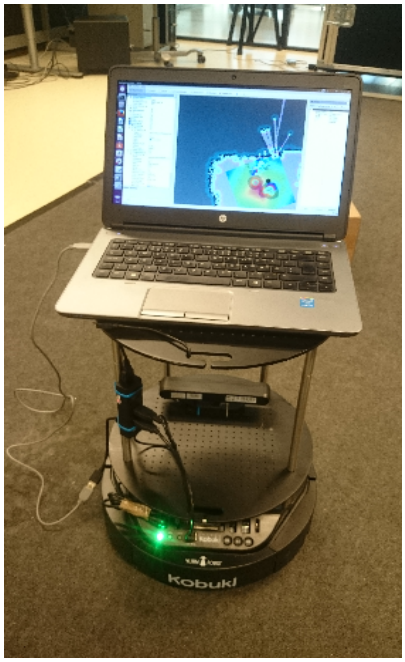



Figure 5.13 Interacting with Real TurtleBot

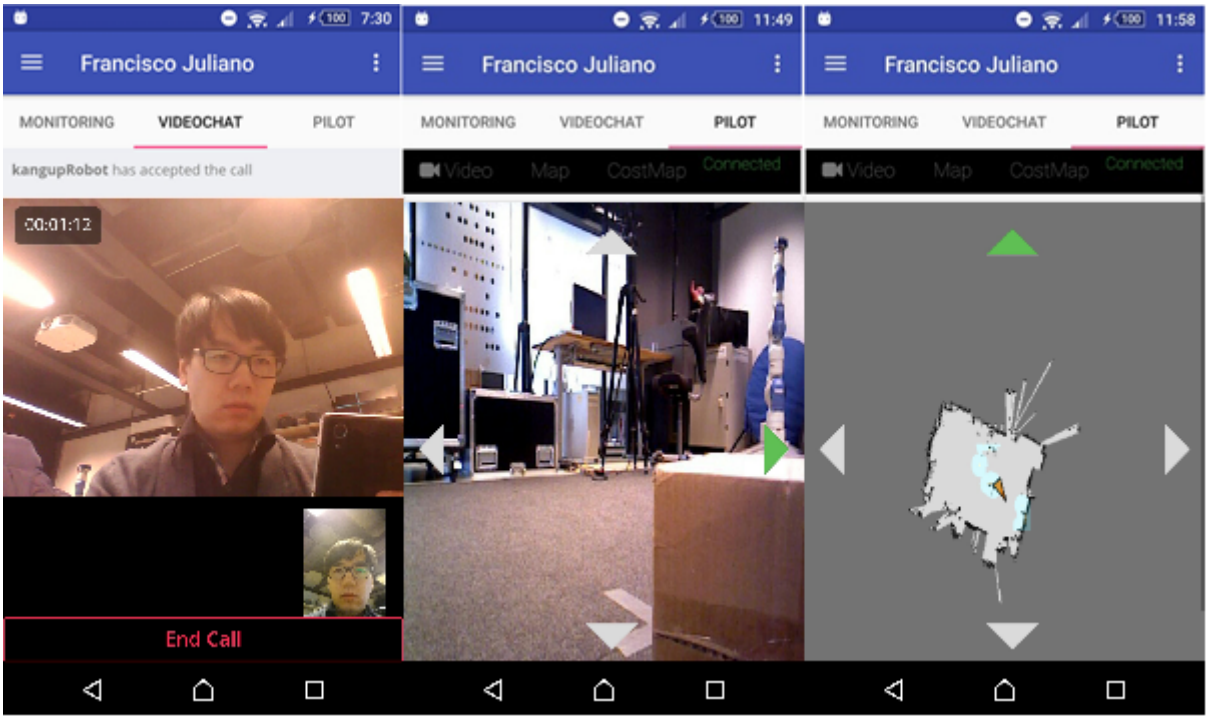



Figure 5.14 Video chat (left), Pilot with camera view (middle), and map view (right) in real world

# 6 Performance Testing

In this chapter the developed systems, Monitoring System and Robot Telepresence System will be tested in various ways by using several testing tools.

## 6.1 Methodology

As we have divided the systems into five parts, the performance testing will also be done with different methods for each divided part. The Monitoring System is divided into two parts: Server with Database and Android part, The Robot Telepresence System is divided into three parts: Pilot, Video Chat and Android part. Each part is tested by using different tools for analyzing different performance indicators. Due to the limited access to the TurtleBot robot, the performance test for Pilot system was performed in the simulation with virtual world.

In next two sections we introduce the definitions of performance indicators we are going to analyze and several performance testing tools we have used.

### 6.1.1 Performance Indicators

- *Latency*: The time an input reaches a system. Lower latency will simplify application development and increase web application scalability.
- *Response time*: The total time it takes from when a user makes a request until the user receives a response.
- *Memory usage*: A term used to describe how much memory is used in the android device for running the application.
- *CPU usage*: The percentage of CPU usage indicates how much of the processor's capacity is currently in use by the system.
- *Network traffic*: The amount of data moving across a network at a given point of time. Network traffic is also known as data traffic.

### 6.1.2 Performance Testing Tools

- *Apache JMeter* - An open source software, may be used to test performance both on static and dynamic resources, Web dynamic applications. This tool is used to test our

database server with "load testing" which simulates increasing number of concurrent users and transactions on a server and check the behavior of the server. By this test we can evaluate server's latency, response time and data transaction under a heavy load.

- *Android Device Monitor* - A standalone tool that is developed by Google and is installed together with Android Studio. It can be launched when the Android device is connected with desktop/laptop and the app is run by Android Studio. The tool helps us to profile the performance of our app by measuring and visualizing real time data such as memory, CPU, network and GPU usage of the Android device.

- *Chrome DevTools* - A set of debugging tools built into Google Chrome. The tool is used to test web applications we developed which can be run in the Chrome browser. The tool has a "Network" panel that provides insights into resources that are requested and downloaded over the network in real time, and "Performance"

- *Battery Historian* - An open source tool developed by Google. With the tool the user can receive a report that provides insight into a device's battery consumption over time. The tool is going to be used for analyzing battery consumption of the developed application.

## 6.2 Testing on Monitoring System

### 6.2.1 Load Testing on Database Server

The testing on Database Server is done by using Apache JMeter. The testing method is a load test that we make an use case scenario that sending HTTP Requests to the server and receiving responses and perform the test by a number of users. The tool allows us to make a thread group that can adjust number of users, loop count and duration of the test. As we can see in the following figure, we set up that 20 users run the scenario as many times as possible in 60 seconds.

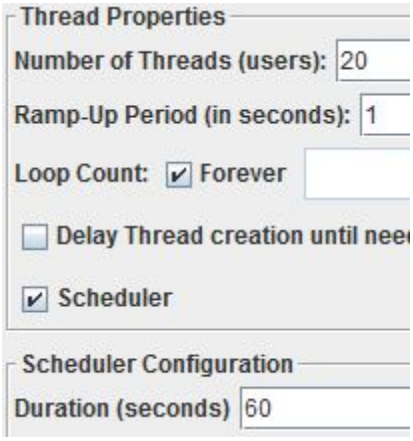



Figure 6.1 Configuration of the load test

The scenario of the test is basic use case of the monitoring service which consists of 6 steps :

1. Login with username and password.
   Request URL: {baseURL}/authenticateUser/{username}/{password}
2. Choose one elderly person in the list and receive measurement setting for making graph.
   Request URL: {baseURL}/getMeasurementsSetting
3. Request specific data of measurement of the elderly person for a certain period of time.
   Request URL: {baseURL}/getMeasurementsByPeriod/{group}/{date}/{periodType}
4. Change measurement type or period and request again.
   Request URL: {baseURL}/getMeasurementsByPeriod/{group}/{date}/{periodType}
5. Set reminder service at the current time.
   Request URL: {baseURL}/setReminder
6. Get up-to-the-minute reminder message
   Request URL: {baseURL}/getReminder

As the result we got 17135 samples in 60 seconds, and it means there were average 285 requests sent to the server every second. The summary of the tests is shown in the following figure:

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | Received KB/sec | Sent KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| HTTP Request | 17135 | 69 | 51 | 1377 | 38.77 | 0.00% | 285.2/sec | 143.84 | 113.72 | 516.4 |
| TOTAL | 17135 | 69 | 51 | 1377 | 38.77 | 0.00% | 285.2/sec | 143.84 | 113.72 | 516.4 |

Figure 6.2 Summary of the samples

The result shows that the average latency is low enough with 69ms. The minimum and maximum latency are 51ms and 1377ms respectively, but as we can see in the next figure (see figure 6.3), only 20 samples have over 800ms latency and the samples with high latency are run at the almost same time period, about 14:07:52 ~ 14:07:53. So we can infer that there was a "server lag" at that period.

| Sample # | Start Time | Thread Name | Label | Sample Time(ms) | Status | Bytes | Sent Bytes | Latency | Connect Time(ms) |
|---|---|---|---|---|---|---|---|---|---|
| 15010 | 14:07:52.768 | Thread Group 1-15 | HTTP Request | 1377 | | 409 | 450 | 1377 | 0 |
| 14997 | 14:07:52.772 | Thread Group 1-14 | HTTP Request | 1358 | | 409 | 450 | 1358 | 0 |
| 14988 | 14:07:52.682 | Thread Group 1-12 | HTTP Request | 981 | | 409 | 450 | 981 | 0 |
| 14996 | 14:07:52.742 | Thread Group 1-20 | HTTP Request | 968 | | 510 | 383 | 968 | 0 |
| 14994 | 14:07:52.742 | Thread Group 1-10 | HTTP Request | 963 | | 491 | 383 | 963 | 0 |
| 14995 | 14:07:52.768 | Thread Group 1-18 | HTTP Request | 942 | | 510 | 383 | 942 | 0 |
| 14993 | 14:07:52.743 | Thread Group 1-6 | HTTP Request | 933 | | 510 | 383 | 933 | 0 |
| 14985 | 14:07:52.734 | Thread Group 1-13 | HTTP Request | 927 | | 465 | 413 | 927 | 0 |
| 14992 | 14:07:52.749 | Thread Group 1-4 | HTTP Request | 920 | | 460 | 413 | 920 | 0 |
| 14984 | 14:07:52.745 | Thread Group 1-2 | HTTP Request | 913 | | 483 | 413 | 913 | 0 |
| 14990 | 14:07:52.752 | Thread Group 1-8 | HTTP Request | 913 | | 465 | 413 | 913 | 0 |
| 14979 | 14:07:52.735 | Thread Group 1-17 | HTTP Request | 912 | | 472 | 413 | 912 | 0 |
| 14989 | 14:07:52.753 | Thread Group 1-3 | HTTP Request | 911 | | 460 | 413 | 911 | 0 |
| 14986 | 14:07:52.755 | Thread Group 1-19 | HTTP Request | 907 | | 465 | 413 | 907 | 0 |
| 14987 | 14:07:52.761 | Thread Group 1-7 | HTTP Request | 902 | | 460 | 413 | 902 | 0 |
| 14991 | 14:07:52.763 | Thread Group 1-5 | HTTP Request | 902 | | 460 | 413 | 902 | 0 |
| 14983 | 14:07:52.756 | Thread Group 1-11 | HTTP Request | 895 | | 460 | 413 | 895 | 0 |
| 14980 | 14:07:52.773 | Thread Group 1-16 | HTTP Request | 874 | | 630 | 392 | 874 | 0 |
| 14981 | 14:07:52.776 | Thread Group 1-9 | HTTP Request | 872 | | 648 | 392 | 872 | 0 |
| 14982 | 14:07:52.780 | Thread Group 1-1 | HTTP Request | 869 | | 648 | 392 | 868 | 0 |
| 15013 | 14:07:53.659 | Thread Group 1-2 | HTTP Request | 519 | | 409 | 450 | 519 | 49 |
| 15014 | 14:07:53.661 | Thread Group 1-13 | HTTP Request | 518 | | 409 | 450 | 518 | 53 |
| 15015 | 14:07:53.662 | Thread Group 1-19 | HTTP Request | 517 | | 409 | 450 | 517 | 0 |
| 15016 | 14:07:53.665 | Thread Group 1-8 | HTTP Request | 514 | | 409 | 450 | 514 | 52 |
| 15011 | 14:07:53.648 | Thread Group 1-9 | HTTP Request | 498 | | 668 | 394 | 498 | 51 |
| 15007 | 14:07:53.647 | Thread Group 1-16 | HTTP Request | 495 | | 668 | 394 | 495 | 0 |
| 15008 | 14:07:53.647 | Thread Group 1-17 | HTTP Request | 495 | | 465 | 413 | 495 | 0 |
| 15012 | 14:07:53.663 | Thread Group 1-12 | HTTP Request | 494 | | 510 | 383 | 494 | 51 |
| 15009 | 14:07:53.651 | Thread Group 1-11 | HTTP Request | 491 | | 472 | 413 | 491 | 55 |
| 15002 | 14:07:53.649 | Thread Group 1-1 | HTTP Request | 490 | | 668 | 394 | 490 | 56 |
| 15006 | 14:07:53.664 | Thread Group 1-3 | HTTP Request | 478 | | 472 | 413 | 478 | 53 |
| 15004 | 14:07:53.663 | Thread Group 1-7 | HTTP Request | 477 | | 472 | 413 | 477 | 51 |
| 15003 | 14:07:53.665 | Thread Group 1-5 | HTTP Request | 474 | | 472 | 413 | 474 | 54 |
| 15001 | 14:07:53.669 | Thread Group 1-4 | HTTP Request | 469 | | 472 | 413 | 469 | 51 |

Scroll automatically? Child samples? No of Samples 17135 Latest Sample 157 Average 69 Deviation 38

Figure 6.3 Detail information about each samples

The following graph is showing response time of the samples over time. Most of the values are around 60ms~70ms, but we can see the graph rises at point 14:07:50, when the server lag occurred.

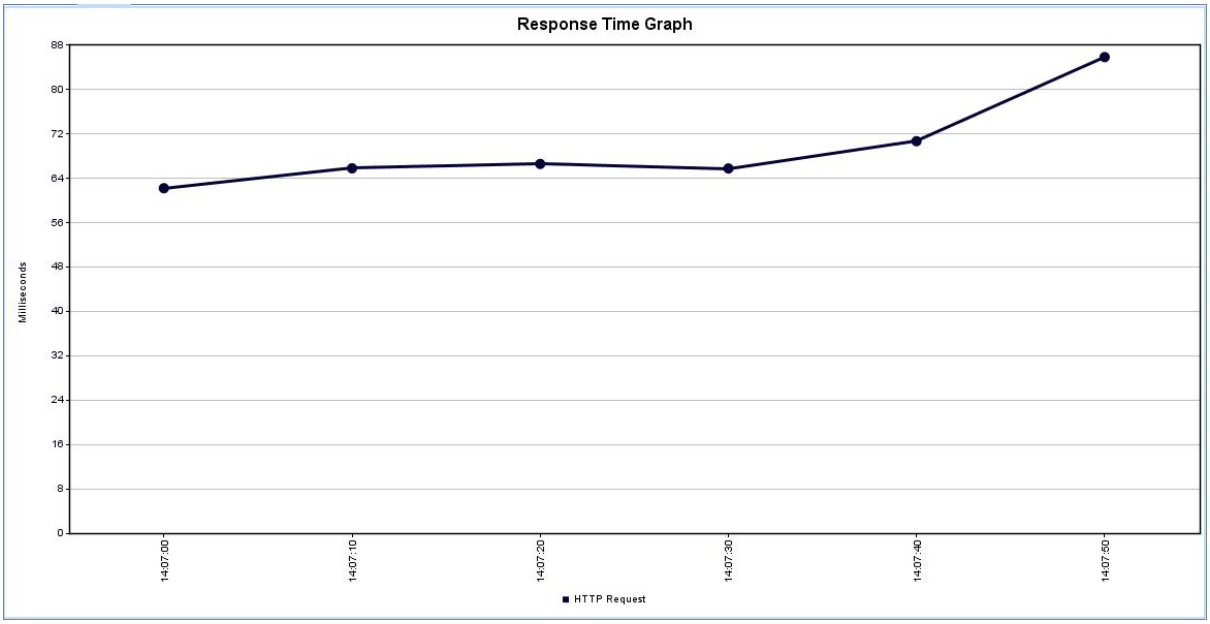



Figure 6.4 Response Time Graph of Load Test

The results of the load test indicate that the performance of the database server is fine enough to be used. There were some points that occurred server lag, but the average latency and the response time were low enough, and it shows the server has potential to tolerate more than 200 users simultaneously.

### 6.2.2 Monitoring System in Android Application

The following figure is Android Device Monitor that shows memory, CPU and Network usage of the Android device when we use Health Monitoring System in the application.

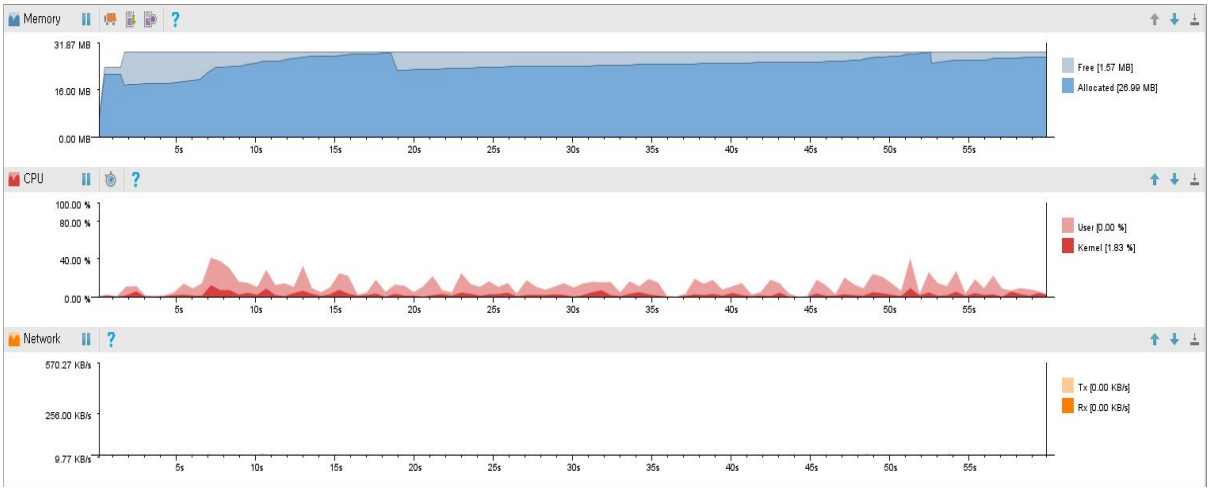



Figure 6.5 Performance for Monitoring System Analyzed by Android Device Monitor

In the graph of Memory usage, the y-axis displays the free and allocated RAM in megabytes and the x-axis shows the time elapsed. The amount of free memory, measured in megabytes, is shown in a light color, and allocated memory is a darker color. When there's a sharp drop in allocated memory that indicates a garbage collection event. The memory usage of our application was around 20 MB at first and there was some fluctuation in the graph, but the average memory usage was around 25MB.

The graph of CPU Monitor displays the CPU usage in real time used in the y-axis. The value is the percentage of total CPU time used in user mode and kernel mode. In user mode, the code must use system APIs to access hardware or memory, has access to virtual memory addresses only, and crashes are usually recoverable. In kernel mode, the code can directly access hardware, including physical memory addresses and crashes halt the device. In the graph there are small peaks that are occurred when the application sends HTTP request or receive responses. The average values are around 10~20% with the most of in user mode.

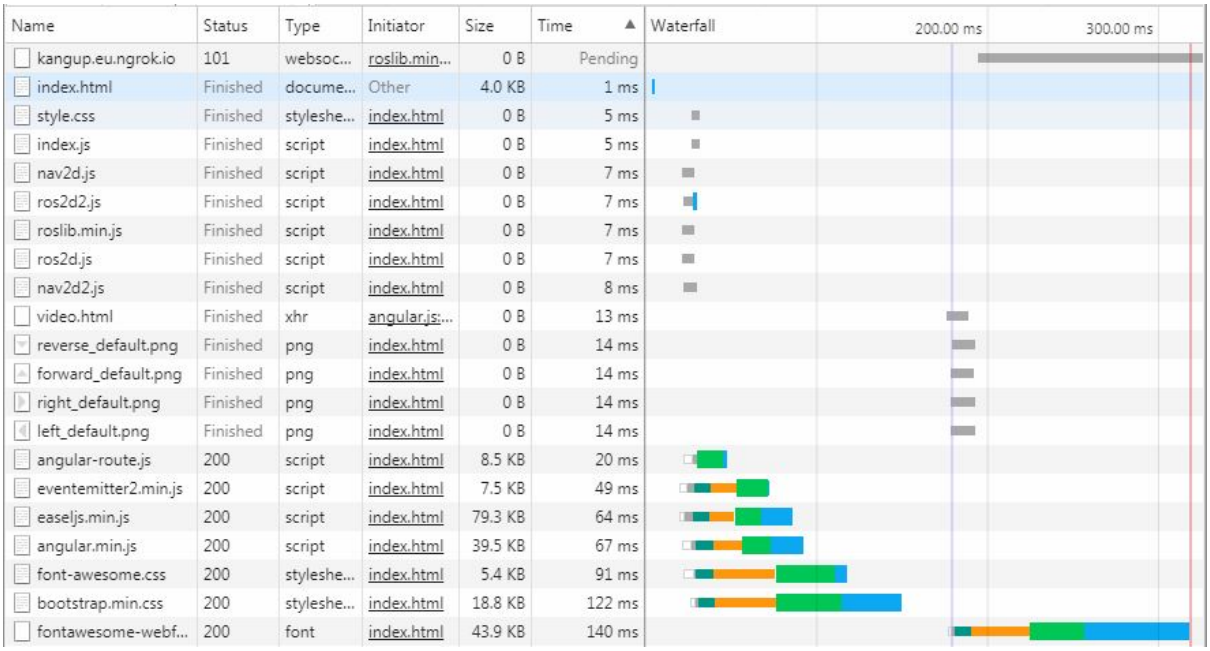
The Network Monitor tracks when our application is making network requests. Using this tool, we can monitor how much data is transferred when the application communicate with the database server. It shows the amount of time it takes for the device to transmit and receive kilobytes of data in x-axis, and the y-axis is in kilobytes per second. But as we saw in figure 6.3, the average bytes of transferred data is about 500bytes, and it's so small that we can't see any peak in the graph.

## 6.3 Testing on Telepresence System

### 6.3.1 Performance of Pilot Web Applications

The Telepresence system that has been developed is a hybrid application. So before we test the system in the android device, we can use a Google Chrome Devtool to analyze details about network traffic when we start the video chat and pilot systems. With this tool we can analyze what kinds of network transactions occurred when we use the application. In the Network panel we can see detailed information of each transaction the application performed as the “Time” shows how long time is spent by each transaction and the "size" shows the size of the transferred data. However, the speed of data transferring depends on the network performance of the desktop/laptop or mobile device, so it can vary from the results we are going to see. The testing is performed by two laptops with different Wi-Fi connections for making more real world situations.

The figure 6.6 shows transactions from when the application started to when the application finished with rendering the page. Here we can see those transactions is performed to load libraries and some image files. The times spent by the most of transactions are less than 50ms, and only two transactions spent time more than 100ms. The time spent by all the transactions for loading web page is less than 1 second as about 650ms.



| Name | Status | Type | Initiator | Size | Time | Waterfall 200.00 ms 300.00 ms |
|---|---|---|---|---|---|---|
| kangup.eu.ngrok.io | 101 | websoc... | roslib.min... | 0 B | Pending | |
| index.html | Finished | docume... | Other | 4.0 KB | 1 ms | |
| style.css | Finished | styleshe... | index.html | 0 B | 5 ms | |
| index.js | Finished | script | index.html | 0 B | 5 ms | |
| nav2d.js | Finished | script | index.html | 0 B | 7 ms | |
| ros2d2.js | Finished | script | index.html | 0 B | 7 ms | |
| roslib.min.js | Finished | script | index.html | 0 B | 7 ms | |
| ros2d.js | Finished | script | index.html | 0 B | 7 ms | |
| nav2d2.js | Finished | script | index.html | 0 B | 8 ms | |
| video.html | Finished | xhr | angular.js:... | 0 B | 13 ms | |
| reverse_default.png | Finished | png | index.html | 0 B | 14 ms | |
| forward_default.png | Finished | png | index.html | 0 B | 14 ms | |
| right_default.png | Finished | png | index.html | 0 B | 14 ms | |
| left_default.png | Finished | png | index.html | 0 B | 14 ms | |
| angular-route.js | 200 | script | index.html | 8.5 KB | 20 ms | |
| eventemitter2.min.js | 200 | script | index.html | 7.5 KB | 49 ms | |
| easeljs.min.js | 200 | script | index.html | 79.3 KB | 64 ms | |
| angular.min.js | 200 | script | index.html | 39.5 KB | 67 ms | |
| font-awesome.css | 200 | styleshe... | index.html | 5.4 KB | 91 ms | |
| bootstrap.min.css | 200 | styleshe... | index.html | 18.8 KB | 122 ms | |
| fontawesome-webf... | 200 | font | index.html | 43.9 KB | 140 ms | |

Figure 6.6 Information of Transferred Data while Starting Pilot Web Application

The next two figures shows data transferring occurring when we use pilot system with camera view and map image respectively. The time spent by receiving data is very short as less than 1ms. The data for camera view is transferred continuously more than one time in every second, while the data for map image is transferred only when they were updated.

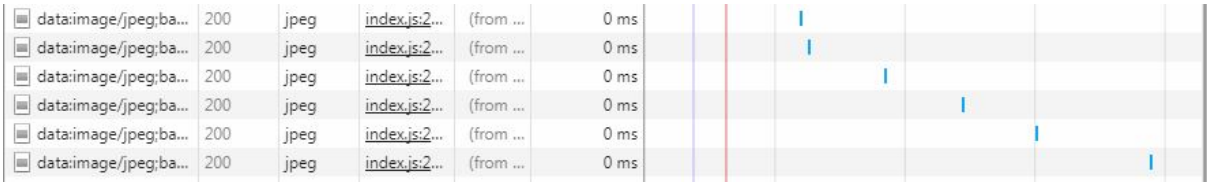



Figure 6.7 Transferred Camera View Data From TurtleBot

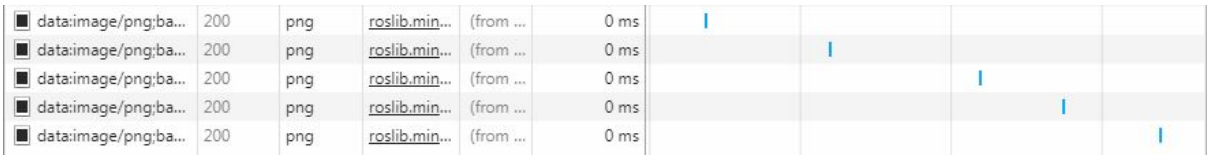



Figure 6.8 Transferred Map Data From TurtleBot

## 6.3.2 Performance of Video Chat Web Application

The video chat web application is also tested by the same method with the pilot web application. The figure 6.9 shows data transactions occurred when the web page is loaded. Like the pilot system the transactions were performed for using JavaScript libraries and CSS file and the time spent for rendering web page is shorter than pilot web application as 300ms.


Figure 6.9 Information of Transferred Data while Starting Video Chat Web Application

The figure 6.10 shows the data transactions occurring when the application tries to enter a room with certain room name and certain username. As we can see there were three transactions. The first one is making session for establishing peer to peer connection between two clients by using signaling server. The second and third one are transactions for making user data and login with that user data. Those three transactions spent relatively more time than other parts but it is hard to make them faster since they are performed by the back end API QuickBlox.

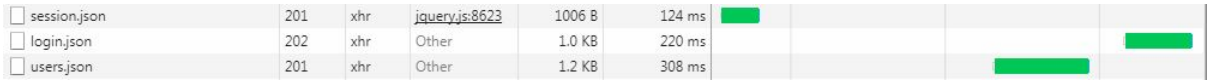



Figure 6.10 Information of Transferred Data while Establishing p2p connection

### 6.3.3 Telepresence System in Android Application

The Telepresence system is also tested in Android Device Monitor for analyzing memory, CPU, Network and GPU usage of the Android device. The figure 6.11 shows the result of the test for pilot system. As like the monitoring system the memory usage is between 20~30 MB, and there was a garbage collection. CPU usage was higher than monitoring system as up to 40%. In the Network traffic panel there were many peaks that shows data the application received continuously from TurtleBot.

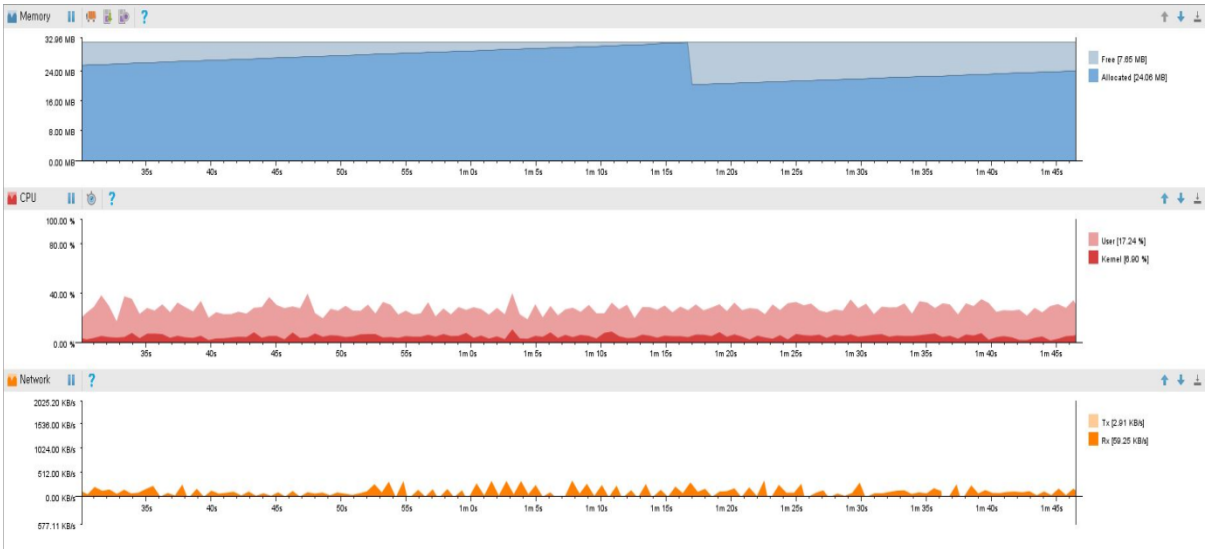



Figure 6.11 Performance for Pilot System Analyzed by Android Device Monitor

The figure 6.12 shows the result of the test for video chat system. The thing that is beneath our notice is that the CPU usage was higher than other systems as up to 50 %. In addition, as the network traffic there are both receiving and sending data as 100~300 KB/s. From these reasons we can infer that the video chat system uses more resources of the android device than other systems.

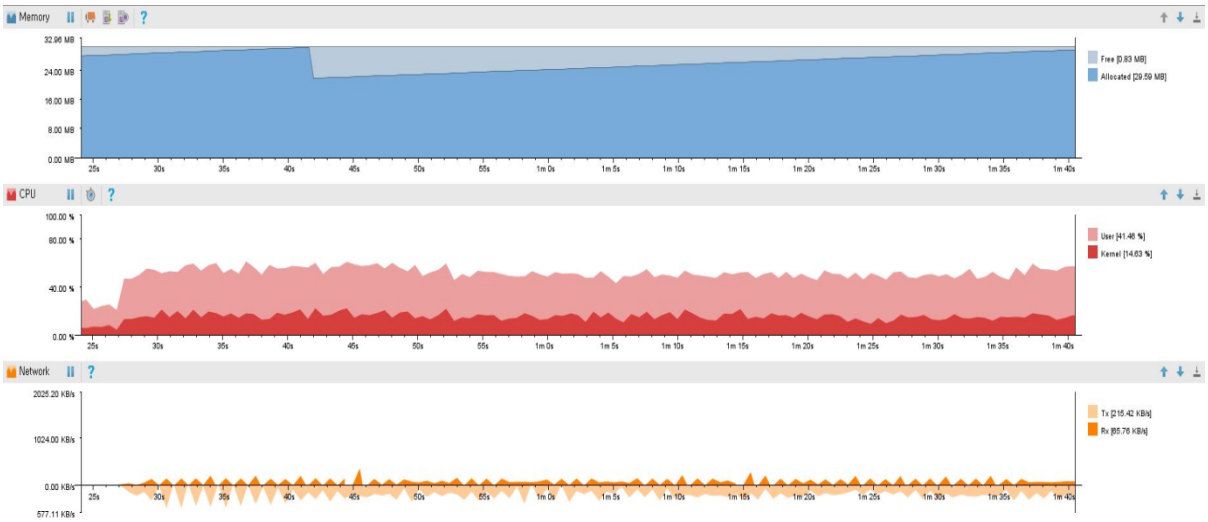



Figure 6.12 Performance for Video Chat System Analyzed by Android Device Monitor

## 6.4 Battery Usage of the Application

"Battery Historian" is a tool provided by Google, which enables the user evaluate battery usage of the android device in with time line. In order to test the battery usage I used each system in 1 hour. The black line in the following figure shows all the battery usage in 3 hours and where the battery is used for. The x-axis shows the time and the y-axis shows how the battery is used. For comparing with the other application, I used "Facebook" application in 1 hour, and the result is shown in the figure 6.14, the test was from 8 pm to 9 pm and battery consumption was about 10%.

My test was performed from 7 pm to 10 pm as we can see in the figure 6.14. From 7 pm to 8 pm the monitoring system was running in foreground of the device and I used some functions intermittently such as requesting other type of physiological data and reminding service. The battery usage was low as 2%.

From 8 pm to 9 pm, the video chat system was used as performed video chat with another

client. The battery usage was unexpectedly high as about 40%. This high battery usage can cause a problem for using the application in the real world.

From 9 pm to 10 pm, the pilot system was running in foreground. Since we know that the camera view uses relatively more resources than the map view, the test was performed with camera view. During the test, the robot was controlled by virtual controller intermittently. The test used about 10% of battery and it is low enough.

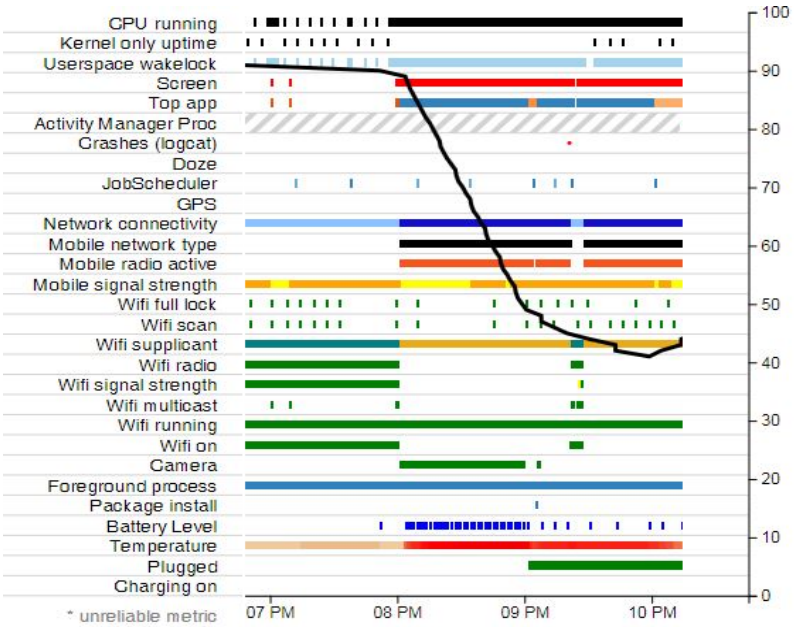



Figure 6.13 Battery usage for using the HMT application

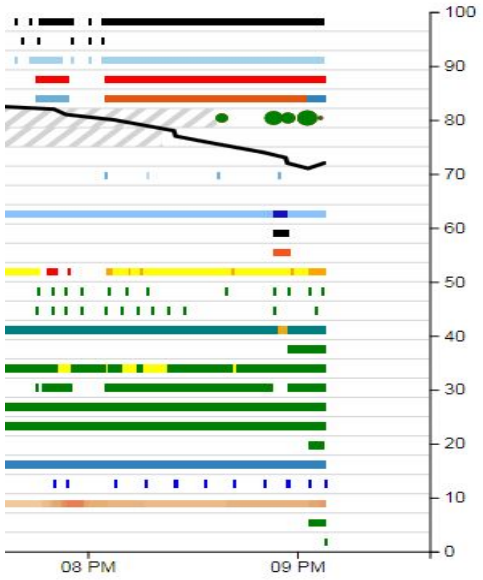



Figure 6.14 Battery usage for using "Facebook" application

# Conclusions and Future Works

# 7 Conclusions

This thesis work intended to develop an android application for allowing caregivers to monitor elderly person's physiological data and remotely control remote robot "Turtlebot" for visiting patient virtually and performing video chat with elderly person. The fundamental idea is from "GiraffPlus" project and is designed to have mainly two systems: Health Monitoring System and Robot Telepresence System.

Health Monitoring System was developed as combination of 3 parts: server part, database part and android application. In this thesis we didn't develop a new sensor network, but we assumed the system use the same sensor network as the GiraffPlus project. The database server was developed as web application server (WAS) by using Apache Tomcat software and is run by Eclipse. The database was created by using MongoDB and connected with the server by using MongoTemplate API. Communication between android application and the server is performed by using REST API as the application can send and receive specific data to/from the server.

Robot Controlling System is composed of pilot system, video chat system and android application. The pilot system and video chat system was developed as web application and integrated into the android application. For the pilot system the application and "TurtleBot" is connected through "rosbridge" server and the robot is remotely controlled by using virtual controller. In order to teleoperating the robot, the application receives camera view, map image and local costmap image from TurtleBot. In costmap view the map image and the local costmap image is overlapped to allow the user navigate the robot more easily by seeing obstacles around the robot. The video chat system is developed by using WebRTC technology which provides peer to peer connection between two clients: android application for caregiver and web application for elderly person. An open source back-end API QuickBlox is used to implement signaling server for establishing peer to peer connection.

The developed systems were tested and analyzed in various aspects by using several testing

tools. The results shows that the developed application and systems have relatively high performance, but the one thing we have to concern is that the high resource consumption of the video chat service, in particular, the battery usage and network traffic were unexpectedly high.

# 8 Future Works

The android application and systems developed in this thesis still have a lot of potential to be improved by future works. In this chapter, I describe the remained problems of my systems and possible projects that can build upon the work I have done.

## 8.1 Usability Testing

The testing usability is very important for developing new software. By usability testing we can assess how easy user interfaces are to use. The usability of the application is not considered in this thesis, so the low fidelity UI of the developed application need to improved. By watching people trying to use the application in the real world, from the small things such as the positions of buttons or non-critical errors, to some critical errors that interrupt normal use or halt the application can be detected.

## 8.2 Improvement of the Telepresence System

The one of things that is needed to be improved for making more high fidelity application is the Telepresence system. The problem is that the two main components of the system, Pilot service and Video Chat service are working independently of each other. The biggest reason that the two services are developed separately is that they use different network connection from each other as the Pilot service uses "rosbridge" server and the Video Chat service uses WebRTC connection. Using those services separately causes some problems in the system, for example the video chat service can't be used before a call initiated by the application is accepted by the elderly person, while the pilot service can be used without any permission to control the robot. Actually combining two services into one system is not so difficult, as the

pilot service is not available before the video chat service establish connection with the elderly person, but in that case the system becomes too complicated and using both services at the same time may cause the application uses unnecessary resources. In the future work, it is necessary to find an efficient way to combine these two systems.

# References


1. Heerink, M., Krose, B., Wielinga, B., Evers, V., Human-Robot User Studies in Eldercare: Lessons Learned, 2006, p. 1.
2. Qiu, M., Dai, W., Gai, K., Mobile Applications Development with Android Technologies and Algorithms, 2017, p. 6.
3. Amalie F.A. Fadzlah., Readiness Measurement Model (RMM): Mathematical-Based Evaluation Technique for the Quantification of Knowledge Acquisition, Individual Understanding, and Interface Acceptance Dimensions of Software Applications on Handheld Devices, 2016, p. 2.
4. Turtlebot 2., URL: http://www.turtlebot.com/turtlebot2/
5. GiraffPlus project., URL: http://www.giraffplus.eu/
6. Broekens, J., Heerink, M., Rosendal, H., Assistive social robots in elderly care: a review, 2009, pp. 1-2.
7. Sabelli, A. M., Kanda, T., Hagita, N., A Conversational Robot in an Elderly Care Center: and Ethnographic Study, 2011, pp. 1-2.
8. Villarreal, J. J., Ljungblad, S., Experience Centred Design for a Robotic Eating Aid, 2011, p. 1.
9. Wynsberghe, A. V., Healthcare Robots: Ethics, Design and Implementation, 2014, p. 75.
10. Pollack, Martha E. et al., Pearl: A Mobile Robotic Assistant for the Elderly, 2002.
11. Giraff., URL: http://www.giraff.org/?lang=en
12. Costa, A., Julian, V., Novais, P., Personal Assistants: Emerging Computational Technologies, 2017, p. 79.
13. Wada, K., Shibata, T., Living with Seal Robots-Its Sociopsychological and Physiological Influences on the Elderly at a Care House, 2007, pp. 972-980.
14. Wada, K., Shibata, T., Social and Physiological Influences of Living with Seal Robots in an Elderly Care House for Two Months, 2008.
15. Stiehl, W.D., et al. The huggable: a therapeutic robotic companion for relational, affective touch, 2006.
16. Fujita, M., AIBO: Toward the Era of Digital Creatures, 2001.
17. Plaumbo, F., Ullberg, J., et al., Sensor Network Infrastructure for a Home Care Monitoring System, 2014, p. 3834.

18. Frennert, S. A., Forsberg, A., Östlund, B., Elderly people's perceptions of a telehealthcare system: Relative advantage, compability, complexity and observability, 2013.
19. Torbjørn S. D., Maged N. K. B., Robots in Health and Social Care: A Complementary Technology to Home Care and Telehealthcare?, 2013, p. 3.
20. Coradeschi, S., et al., GiraffPlus: Combining social interaction and long term monitoring for promoting independent living, 2013, pp. 1-2.
21. GiraffPlus project., URL: http://www.giraffplus.eu/
22. Cesta, A., Cortellessa, G., Tiberio, L., Long-term Evaluation of a Mobile Remote Presence Robot for the Elderly, 2011.
23. J. Gonzalez-Jimenez., et al., Technical Improvements of the Giraff Telepresence Robot based on Users' Evaluation, 2012, pp. 1-2.
24. Maria Linden., et al., GiraffPlus Project: D1.2 Technological Component Specifications, 2014, pp. 15-39.
25. Amedeo Cesta., et al., GiraffPlus Project: D4.1 The Interaction and Visualization Service and Personalization Module – Alpha Release, 2014, pp. 8-10.
26. Silvia Coradeschi., et al., GiraffPlus Project: D1.3 System Reference Architecture, 2014, pp. 9, 65, 69-72
27. Spring Framework., URL: https://spring.io/
28. MongoDB., https://www.mongodb.com/
29. GiraffPlus project: Pilot 2.0 User Guide., pp. 7-14.
30. What is application? - Definition from WhatIs.com., URL: http://searchsoftwarequality.techtarget.com/definition/application
31. What is Mobile Application? - Definition from Techopedia., URL: https://www.techopedia.com/definition/2953/mobile-application-mobile-app
32. IDC: Smartphone OS Market Share., URL: https://www.idc.com/promo/smartphone-market-share/os
33. What is Android? Applications in Android, Android Mobile App Development - Arokia IT., URL: http://www.arokiait.com/whatis-android.htm
34. Jobe, W., Native Apps vs. Mobile Web Apps, 2013, pp. 27-32
35. Web, Hybrid Or Native Apps? What's The Difference?., URL: https://www.mobiloud.com/blog/web-hybrid-native-apps/
36. Web app vs. Native app - App Press., URL: https://www.app-press.com/blog/web-app-

vs-native-app
37. Types of Mobile Applications., URL: https://www.linkedin.com/pulse/types-mobile-applications-kaushik-bhatti-
38. What's the difference between a web site and a web application? - Stack Overflow., URL: https://stackoverflow.com/questions/8694922/whats-the-difference-between-a-web-site-and-a-web-application
39. What is a web application? - 3Squared Ltd., URL: https://www.3squared.com/2011/02/what-is-a-web-application/
40. Mobile: Native Apps, Web Apps, and Hybrid Apps., URL: https://www.nngroup.com/articles/mobile-native-apps/
41. Mobile apps vs. PB desktop apps., URL: https://www.appeon.com/support/documents/appeon_online_help/2.0/development_guidelines/ch02s01.html
42. Touch vs. Mouse - The Hipper Element., URL: http://thehipperelement.com/post/47937273876/touch-vs-mouse
43. Mobile vs. Desktop: 10 Key Differences - ParadoxLabs., URL: https://www.paradoxlabs.com/blog/mobile-vs-desktop-10-key-differences/
44. 10 Ways the Mobile Web is Different., URL: https://www.elated.com/articles/10-ways-the-mobile-web-is-different/
45. Spring Framework., URL: https://spring.io/
46. Apache Tomcat., URL: http://tomcat.apache.org/
47. WebRTC Home., URL: https://webrtc.org/
48. Apache Cordova., URL: https://cordova.apache.org/
49. Getting Started with WebRTC - HTML5 Rocks., URL: https://www.html5rocks.com/ko/tutorials/webrtc/basics/
50. ROS.org - Powering the world's robots., URL: http://www.ros.org/
51. Jakobsson, C., Peer-to-peer communication in web browsers using WebRTC, 2015.
52. rosbridge_suite - ROS Wiki., URL: http://wiki.ros.org/rosbridge_suite
53. Johnson, R., et al., Spring java/j2ee Application Framework, 2004.
54. The Spring Web MVC Framework - TechTarget., URL: http://media.techtarget.com/tss/static/articles/content/AgileJavaDev/SpringMVC.pdf
55. What is Web server? - Definition from WhatIs.com., URL:

http://whatis.techtarget.com/definition/Web-server
56. App server, Web server: what's the difference? - JavaWorld., URL: https://www.javaworld.com/article/2077354/learn-java/app-server-web-server-what-s-the-difference.html
57. Srivastava, A., Bhargava, A., Understanding Application Servers, 2003, pp. 1-4.
58. DR. M. Hoche., Structure Capital, Assessment, Compliance and More, 2016, p. 100.
59. Most popular Java application servers: 2017 edition, URL: https://plumbr.eu/blog/java/most-popular-java-application-servers-2017-edition
60. Press, W., Tomcat for beginning Web developers, 2005, p. 1.
61. Dedoimedo., Apache Web server Complete Guide, p.11.
62. Introduction to Spring Framework., URL: https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/overview.html
63. Fredrich, T., RESTful Service Best Practices., 2012.
64. Richardson, L., Ruby, S., RESTful Web Services., 2007.
65. Stateless Spring Security on REST API - Complex to Simple., URL: https://malalanayake.wordpress.com/2014/06/30/stateless-spring-security-on-rest-api/
66. Alex, B., Taylor, L., Winch, R., Hillert, G., Spring Security Reference, 2015
67. JSON Web Tokens - jwt.io., URL: https://jwt.io/
68. JSON Web Token Tutorial: An Example in Laravel and AngularJS., URL: https://www.toptal.com/web/cookie-free-authentication-with-json-web-tokens-an-example-in-laravel-and-angularjs
69. Sapthami, R., Anisha, P. Rodrigues., Study on Handling Semi-structured Data using NoSQL Database, 2016.
70. Palumbo, F., et al., GiraffPlus project: D2.1 First Prototype of sensors, Giraff platform and network system report, 2014.
71. Palumbo, F., et al., GiraffPlus project: D2.2 Second Prototype of sensors, Giraff platform and network system, 2014.
72. Spring Data MongoDB., URL: https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/
73. Spring 4 MongoDB Example (MongoTemplate CRUD)., URL: http://www.technicalkeeda.com/spring-tutorials/spring-4-mongodb-example
74. Introduction to Spring Data MongoDB - Daeldung., URL:

http://www.baeldung.com/spring-data-mongodb-tutorial
75. Thambawita, V., et al., Low Cost Telepresence Robot, 2011, p. 2.
76. Cesta, A., et al., Enabling Social Interaction Through Embodiment in Excite, 2013, pp. 1-3.
77. Foustanas, N., Helping Elderly Users control a Telepresence Robot With a Touch Screen, 2015, pp. 4-14.
78. TurtleBot Archives - JetsonHacks., URL: http://www.jetsonhacks.com/tag/turtlebot/
79. Jose Manuel Ferrandez Vicente., Natural and Artificial Models in Computation and Biology, 2013, p. 142.
80. Quigley, M., ROS: and open-source Robot Operating System.
81. Martinez, A., Fernandez, E., Learning ROS for Robotics Programming, 2013, pp. 8-13.
82. Jason M. O'Kane., A Gentil Introduction to ROS, 2014.
83. Sam Dutton, WebRTC in the real world: STUN, TURN and signaling, 2013, URL: https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/
84. roslibjs - ROS Wiki., URL: http://wiki.ros.org/roslibjs
85. ros2djs - ROS Wiki., URL: http://wiki.ros.org/ros2djs
86. nav2djs - ROS Wiki., URL: http://wiki.ros.org/nav2djs
87. rviz - ROS Wiki., URL: http://wiki.ros.org/rviz
88. gmapping - ROS Wiki., URL: http://wiki.ros.org/gmapping
89. amcl - ROS Wiki., URL: http://wiki.ros.org/amcl
90. QuickBlox Backend: Cloud Communication Backend API As A Service For Mobile And Web Apps., URL: https://quickblox.com/
91. Web video / voice calls on WebRTC, code samples., URL: https://quickblox.com/developers/Sample-webrtc-web#Integrate_video_calling_to_your_application
92. S. Coradeschi., GiraffPlus: Combining social interaction and long term monitoring for promoting independent living, 2013, p. 1.
93. Atallah, L., Lo, B., Yang, G.Z., Siegemund, F., Wirelessly Accessible Sensor Populations (WASP) for Elderly Care Monitoring, 2008, pp. 1-2.
94. Yang, G.Z., Body Sensor Networks, 2006.
95. Mutha, N., et al., Patient Health Monitoring Using Android Application, 2007.
96. How to Check Your Heart Rate on Any Android Phone - Android., URL:

https://android.gadgethacks.com/how-to/check-your-heart-rate-any-android-phone-0164070/
97. Apache JMeter., URL: http://jmeter.apache.org/index.html
98. Nimbledroid., URL: https://nimbledroid.com/
99. Technology use among seniours - Pew Research Center., URL: http://www.pewinternet.org/2017/05/17/technology-use-among-seniors/
100. GitHub - markwsilliman/turtlebot-web-app: Web app for CoffeeBot., URL: https://github.com/markwsilliman/turtlebot-web-app
101. How do I start Mongo DB from Windows? - Stack Overflow., URL: https://stackoverflow.com/questions/20796714/how-do-i-start-mongo-db-from-windows
102. Silvia Coradeschi., et al., GiraffPlus Project: D5.1 Preliminary technological integration (Pre-T-Int) report, 2012, pp. 27-49
103. Aggregation Pipeline - MongoDB Manual 3.4., URL: https://docs.mongodb.com/manual/core/aggregation-pipeline/
104. Elastic Beanstalk Applications., URL: https://eu-west-1.console.aws.amazon.com/elasticbeanstalk/home?region=eu-west-1#/applications
105. MongoDB Hosting: Database-as-a-Service by mLab., URL: https://mlab.com/
106. Packages - ROS Wiki., URL: http://wiki.ros.org/Packages
107. GitHub - QuickBlox/quickblox-javascript-sdk: JavaScript SDK of QuickBlox cloud backend platform., URL: https://github.com/QuickBlox/quickblox-javascript-sdk
108. ngrok - secure introspectable tunnels to localhost., URL: https://ngrok.com/product
109. Robot Web Tools., URL: http://robotwebtools.org/