Gergely Györki

Ting-e-Ling – A Health Monitoring Device

Information Technology
2016

VAASA UNIVERSITY OF APPLIED SCIENCES
Information Technology – Software Engineering

ABSTRACT

Author	Gergely Györki
Title	Ting-E-Ling – A Health Monitoring Device
Year	2016
Language	English
Pages	36
Name of Supervisor	Timo Kankaanpää

With the world becoming focused on health on a personal level, individuals have the possibility to track their basic medical data, such as pulse, blood pressure, temperature, activity and hormone levels. The global trends also show general interest in personal health. This project aims to help people with tracking their individual health, by measuring and sending related data to the most common hand-held devices: smartphones. Not only this, but there are other implications of the collected data than health: the project will try to match people up (if allowed) with other people who share similar body traits and activity to them, helping them to find friends, partners or even love interests on Social Media sites like Tinder.

Delektre Ltd is developing a ring (named Ting-E-Ling), which is capable of measuring the previously mentioned health data (pulse, blood pressure and body temperature to begin with). It can transfer this data to other devices via a Bluetooth connection. The project needs to make a software level interface on those devices (Android smartphones – later IOS planned), to process, log and display the data. Other parts include displaying the data on Social Media. The ring will use Low-Energy Bluetooth for the communication, which must be taken into account during the development. The user interface of the application must be intuitive and easy to use.

The ring is currently in development phase. Initial testing of the hardware has begun, and a prototype device capable of mimicking the data stream from the ring has been produced and provided by the company. The development of an Android application which is capable of the previously listed features has been initiated.

PAGE OF CONTENTS

1 INTRODUCTION

The Ting-E-Ling Ring project is a project started by a company called Delektre Oy. Delektre is an innovative micro-company, which deals with everyday problems in new and imaginative ways. The idea and the features of the project were designed by Delektre, as well as the hardware and software of the ring itself, but the implementation and documentation of the phone application was done by the writer of this thesis, with the help of the company.

The aim of the project is to provide a compact health measurement device for every day use, which utilizes two needed pieces of hardware: a ring, which can be worn on the finger of the user, and the user's own smartphone.

The ring would house a set of sensors, which are used to collect data from the user's body. The types of data the sensors can measure are (but not limited to): pulse, blood pressure and temperature. The ring then uses a Low Energy Bluetooth interface to transmit the data to smartphones. The transmission utilizes the Attribute Protocol and the Generic Attribute Profile to transmit the data between devices. The phones need a specific software (the topic of this thesis), which listens to the broadcasts of the ring, and translates the raw data into the display. The application is able to differentiate between different data, and switch between different types seamlessly while keeping track of previous measurements in all types. The application is capable of saving the data for prolonged periods of time on the storage of the phone, and synchronise the data to an online media. It also makes a connection to Social Media services through the profile of the user, and even post**s** data in the name of the user, if the permission is given (for example, if the user was running for 15 minutes, and the ring detects the constant high level pulse, and it can assume that the user is working out, and when the pulse returns to a constant normal level, it can post a message about it). Naturally, the user has full control over to whom and when the data is shown.

The data gathered can be used for other areas of life as well. One is, to get the similarly active people together. The application can track other people who have similar health trends like the user, and can recommend meeting them. This can be used to get to know other, similar minded people.

The application initially is designed to be used on Android platforms. The development language is Java, and Android Studio was used as the development IDE. The smart phone needs to be able to utilize Low-Energy Bluetooth (BLE) technology to be able to receive the transmissions coming from the hardware.

In the 2nd Chapter, a background research is shown on human health, which is utilized in the project. It includes relevant information about Pulse, Blood Pressure and Body Temperature. The 3rd Chapter gives description about the relevant technologies used to develop and test the project. The 4th Chapter is all about the description of the project itself. It was done before the actual work has started, and consists of the modelling of data, the structure of the software and the relevant diagrams to help the description. The 5th shows the initial GUI plans and the final GUI, and describes what the components are expected to do. In 6th Chapter covers the implementation of the project, where the more difficult parts of the application are covered and shown with code snippets taken from the IDE. The 7th Chapter gives details about the testing phase of the project. In Chapter 8, the summary of the learning result is explained, and in Chapter 9, the conclusion of the project is done.

2 BACKGROUND ON HUMAN HEALTH

Since the project focuses heavily on tracking health data of individual persons, research was done on the data collected. The data currently tracked in the project are pulse, blood pressure and body temperature.

2.1 Pulse

Pulse is the number of times the heart pumps in one minute. It is measured in beats per minute (BPM). This value is influenced by the current needs of the body, including THE need of absorbing oxygen or excreting carbon-dioxide. In normal cases, the value rests between 50 and 90, with physically healthy people generally having A lower resting pulse. Values exceeding 90 are considered high and this state is called Tachycardia. Above 160 values can be considered very high, but during At heavy physical load (such as excercise), this rates can be considered normal. Constant, very high values can indicate health problems, and they are dangerous, since they can lead to cardiac arrest or stroke. Lower values are desired, but extremely low values (below 50) can result in dizzyness, fatigue or even fainting. During sleep, the pulse of a healthy adult can drop down to 40-50BPM. All of these values are subject to change based on age, physical attributes and activity of a person, with the younger and smaller people having higher resting values, older and physically fit people having lower resting values. (6)

Pulse can be measured with hand, if a person is trained to do it. It can be palpated at places, where one can push an artery against a bone. Easily palpable places are the wrists, the side of neck and the inside of the elbow. The most generally used method is to place the index-, middle- and ring-fingers on the place to the palpation place, and count the beats for a minute. Heart rate can also be measured by listening to the heart beats directly, with the common accessory being the stethoscope. (5)

Newer methods include wrist mounted devices, which automatically measure the pulse (and the blood pressure as well). Most of these can still be bulky and uncomfortable to use, since most of them exerts high pressure around the measurement place to compress the artery against the bones.

2.2 Blood Pressure

Blood pressure is the force the blood exerts on the wall of the blood vessels during flow. It is measured in millimeters of mercury (mmHg). The measurement is broken down to two parts: systolic and diastolic blood pressure. Systolic pressure is the pressure during the heart beat, and diastolic pressure is the pressure while the heart rests during beats. The notation is the systolic pressure followed by a slash followed by the diastolic pressure (such as 110/60). The desired ranges of values are 90-120 for the systolic cycle, and 60-80 for the diastolic cycle. Under the lower limit, the state is called hypotension. This state can cause dizziness or fainting, and long term low value can cause damage in the brain due to lack of enough blood. Hypertension, on the other hand, is the state if a person has systolic pressure above 140 or diastolic pressure above 90. This state has multiple stages, with above 160/100 signing a Stage 2 hypertension and above 180/110 a Hypertensic urgency. High blood pressure can lead to strokes, heart attacks and it is one of the main causes of kidney failure. Even moderate rise in the average blood pressure can reduce the life expectancy of a person, hence the need to carefully monitor this value. The mentioned values are lower for children under about 18 years old, with the younger the child, the lower the blood pressure. (3)

Measuring this value with sufficient accuracy is usually done with a stethoscope and sphygmomanometer. The sphygmomanometer consists of an inflatable cuff attached to a mercury (or aneroid) manometer. The measurement is done by attaching the cuff to the arm of the patient, roughly at the same height as the heart. Then, by inflating the cuff, the blood circulation is obstructed in the arm. The examiner listens to the heart beats of the patient with the stethoscope below the attached point of the cuff. Then, the cuff is slowly deflated, and the blood from the heartbeat starts to make a sound and exert pressure on the cuff, which pushes the attached liquid in the manometer higher. This measured value is the systolic pressure. Then during the rest of the heart, the sound silences and the pressure drops: this is the diastolic pressure. (3)

More modern home measuring devices can measure both pressure and pulse together with a similar inflatable cuff and an electronic device. While the size of these devices is smaller, and they are easier to use, even for untrained people, these devices are still too big to be used all day around without being inconvenient.

2.3 Body Temperature

Normal human body temperature depends on the place of measurement, and it can change slightly throughout the day. Commonly measurements are taken in the mouth, armpit or rectum. Normal human body temperature varies around 36,8°C (+-0.4°C). Under 35°C is the state called hypothermia, when the body temperature drops to dangerously low levels, which can cause shivers, confusion and eventually death. Over above around 37.5°C (there is no universally agreed value) is called hyperthermia. This state can be reached due to overheating or fever, and while they produce different symptoms, both can be dangerous and taken care of. Overheating can cause severe dehydration of the body, and fevers can be a result of illness. Above 40°C body temperatures can be extremely dangerous, especially for children, as at this level, the cells of the body start to get damaged, due to the proteins breaking down by the heat.

Old measurement devices used mercury in a sealed glass tube, with a metal tip, which heated the mercury with body heat, and the temperature could be read from the expansion of it. Newer devices use electrical sensors for the measurement, and are completely automatic. While they are small, they are still inconvenient if someone wishes to track their body temperature all around the day. Smaller devices have been developed (and are being developed) which can measure the body temperature on the fingers of the patient. (4)

3 RELEVANT TECHNOLOGIES

3.1 Low-Energy Bluetooth, Attribute Protocol and Generic Attribute Profile

The Ting-E-Ling project, as mentioned before, uses a small hardware, shaped as a wearable ring as the data collection device. This takes care of taking the measurements constantly. It has a Low-Energy Bluetooth device for broadcasting the collected data for paired smartphones. Because of this, the devices used to receive transmissions from the Ring must be capable of listening to Low-Energy Bluetooth broadcasts to pick up the signals. Normal (higher energy) Bluetooth is not used.

The application uses the Attribute Protocol (ATT) to rceive the signals from the ring. As the name of the ATT suggests, the sole building blocks of the transmissions are the attributes. Each attribute is made up by three parts:

- An UUID, which defines the type of the attribute.
- A value.
- A 16-bit handle, which identifies the attribute (as several attributes with the same UUID can exist within a device).

The ATT does not define or validate the value, it merely stores them in an array of bytes of given length. The meaning of the value is defined by the UUID, which in turn is defined in the higher level protocols, such as the Generic Attribute Profile (GATT). Permissions hidden inside the value are also left to handle by the higher protocols, as ATT does not try to test the values for these.

The ATT uses server-client hierarchy in the transmissions. The server stores the values, the client stores nothing; it only reads and writes the values. Most of the time, the client initiates the connection by inquiring the attributes from the server, but ATT has the capability to set indication flags, where the server takes the initiative, and notifies the client that a flagged value has been changed. In the case of this project, the server is the ring, and the client is the smartphone application. (2)

The Generic Attribute Profile (GATT), is built top on ATT. Its job is to interpret and translate the attributes into usable values. To do this, it uses the UUIDs and

handles to decide the type of attribute. The basis of these decisions is the UUID of 0x2800. This UUID, along with the handle, signs a new primary service definition. The GATT uses these service definitions to plot the boundaries of the services. The service identifier is supplied as the value for the attribute. The UUID 0x2800 is a signal for the GATT, that a new primary service is defined, and all attributes until the next 0x2800 UUID are part of that service. Secondary services, which are included in primary services, are defined with the UUID of 0x2801, but they work the same way. The service itself is described in the value by another UUID.

The attributes between the service definitions are the characteristics of the service. THE characteristics come as pairs of attributes:

- The Main Characteristic Attribute stores the UUID of the value, and the handle of the value. The main characteris**tc's** own UUID is 0x2803, to allow easy discovery of all characteristics.
- The value characteristic stores the actual value. Its UUID and handle is defined in the main characteristic, which allows for cross checking. The format is decided by the UUID, so the client always knows how to interpret the value.

Apart from the value, the ATT can add more description of the value (such as unit, range, human readable description etc). These values can be identified in the following way: it is not the value attribute, which can be found if it has the handle which is defined in the characteristic AND it falls into the characteristics range (before the next characteristic definition happens). Most of these are defined as standard descriptors in GATT, but services can define their own.

There is a special descriptor, called the Client Characteristics Configuration Descriptor (CCC or 3C). This descriptor always has the UUID of 0x2902, it has a 16-bit value, which is meant to be a bitmap. The server is required to store and serve a seperate instance of the value for each connected client, and each client can see only their copy. The first 2 bits of the CCC are reserved by the GATT specification, for notification and indication. By setting these bits, the client can ask the server to

notify whenever the characteristic changes, which is very useful for peripherial devices, such as the Ting-E-Ling Ring. The CCCs can be identified the same way as the other descriptors. The method of the notification is that when the server changes a charateristic, which the notifications has set, it starts to advertise. When the client picks up the signal, it initiates connection, and reads the characteristic. (2)

Table 1 below shows an example of attribute transmissions, and the description of the interpretation of them (the UUID for A thermometer is not this. These are just example values):

Handle	UUID	Description	Value
0x0100	0x2800	Thermometer service definition	UUID 0x1816
0x0101	0x2803	Characteristic: temperature	UUID 0x2A2B Value handle: 0x0102
0x0102	0x2A2B	Temperature value	20 degrees
0x0104	0x2A1F	Descriptor: unit	Celsius
0x0105	0x2902	Client characteristic configuration descriptor	0x0000
0x0110	0x2803	Characteristic: date/time	UUID 0x2A08 Value handle: 0x0111
0x0111	0x2A08	Date/Time	1/1/1980 12:00

Table 1 – Example ATT transmission

The first attribute sent is the service declartion of the thermometer. It has the compulsory 0x2800 as its UUID, and has the thermometer UUID as the value. The second attribute is a characteristic declaration, where it has the compulsory UUID of 0x2803, and has the UUID of the temperature and the handle for the value as its value. The third attribute is the temperature value. The fourth one is a descriptor, which defines the unit for the temperature. The fifth one is the CCC, which has the compulsory UUID of 0x2902 and has a bitmap as its value. The sixth and seventh attributes are again characteristic descriptions and a value, for a timestamp. The table was taken from epxx.co - Bluetooth: ATT and GATT.

The Low-Energy Bluetooth, the ATT and GATT are integral parts of the project. With AN understanding of these protocols and profiles, the interpretation of the transmitted data becomes much easier.

3.2 Java

The application itself is written in Java programming language, utilizing the Android SDK libraries. Java is a purely an Object Oriented Language, created in 1991 by Sun. It became well- known in the technological life in 1995, when it was integrated into the Netscape Navigator webbrowser.

Java sorts the application's files into packages. This allows logical seperation of the files, as well as allowing multiple objects of the same name existing in the same project, as long as they are in different packages.

The objects of Java are called classes. Every function or attribute is sorted logically into a class. The classes all originatE back to a core Object class by using inheritance. With inheritance, classes can inherit attributes and functions from their parent class, override them and add new attributes and functions. In other words, they extend (or specialize) the functionality of their parent. Most of the time, class objects must be instantiated before use. One class can have multiple number of instances (with different values given to their attributes) at any time.

The attributes and functions of the classes can have different access levels. The default access level is not signed in the code, and it allows the usage of the attribute or function inside the package the class is in. The other permission levels are:

- Public – the attribute/function can be accessed project-wide.
- Protected – the attribute/function can only be accessed from the class itself OR a child class.
- Private – the attribute/function can only be accessed from the class itself.

There are several other descriptors of the functins and attributes, which can influence the behaviour of them. If they are signed as ”static”, they can be accessed without instantiating the class, and all instances of the class share the same ”static” value. Static functions can only use static variables. Final variables must be given a value when the class is instantiated and (as their name suggests) their value cannot be changed in the future.

Java support abstract classes and interfaces. Abstract classes are incomplete, which must be subclassed, with their functions fully described in the child class. They can contain variables of any kind as usual. Interfaces contain only the footprint of the functions, and they can only contain final variables.

Java can be utilized in development of multiple projects. It has built in libraries to develop web servlets (servlet API), applications with GUI (Swing), Java Servlet Pages (JSP) and many more. It is not limited to these examples, as there are numerous libraries made by the community, which can enhance its usage enermously.

The Java language uses the Java Virtual Machine (JVM) as a platform to run. The JVM is an universal platform for all Java based programs. It allows easy porting between different operating systems.

3.3 Android SDK and Operating System

The Android Software Development Kit is a collection of libraries and development tools. It includes a debugger (Android Debugger Bridge, aka ADB), the libraries for development, an emulator, which enables testing of the applications on PC, documentation and sample codes. It does not require any IDE to run, as the developers can modify the source files of the projects, compile, run and debug them with command line tools included in the SDK, and even control attached test devices. However, the main supported IDE for the SDK is Android Studio by Google. Other mainstream IDEs used for Android development are IntelliJ Idea, which has built in Android support, and Eclipse and Netbeans, which support Android development via plugins. The compiled projects are packaged as APK (apk extension) files, which are stored in the /data/app folder on the Android devices (this folder requires root permission to access for security reasons). To install the applications, the APK files ask for the required permissions from the user on installation.

The Android projects are using Views for user display (Views are stored in XML description files, which the Android libraries translate into viewable objects), and Activities to process the information going in and out of the application. One application can have numerous views and activities. Activities can communicate with

each other and other applications with Intents. Intents (as the name suggests), signals a request from the Activity. It has a Request Code, a Return code, and it can hold extra variables declared by the user, however it is incapable of passing objects.

This project relies heavily on the Android SDK, using every aspect of it. The IDE used for development is Android Studio by Google.

3.4 XML

The Extensive Markup Language (XML) is a markup language, which is used very heavily in today's software development projects. It has a wide variety of use, ranging from making configuration files to describe Views in webpages and Android applications.

XML files are composed of Unicode character strings. The version and encoding of the file is stated in the beginning of the document, and while it can be omitted, it is highly recommended by the standards to state.

XML has usually two different objects: markups and contents. Markups usually start with a < character and end with >. These markups are known as tags. Tags can be start tags (<example>), end tags (</example>, or empty tags (<example />). Tags which have both start- and end-tags and have content, or they have empty tag without content are defined as elements.

Start-tags or empty tags can have attributes. The usage of these is usually to provide extra information about the content, for example giving the path of a file or giving the country code for a location.

XML is used in the project for several purposes. The Manifest of the application, which states the required permissions to run the app, the required version of the application, the Activities used in the app (and it specifies which is the starter activity) is one such file. Other XML files contain static variables for the GUI (strings, width and height values of objects in the Views, colors, etc.). Other usage of XML is the description of the Views of activities, which are interpreted by the operating system and translated to user viewable objects.

3.5 JSON

The Ting-E-Ling phone application uses JavaScript Object Notation (JSON) files to store the data of the measurements. JSON files are stored in the extension with json, and they are user readable, plain text files. JSON files can have objects and arrays stored inside of them. Arrays are made up by a list of objects, and objects can have arrays as a property. Arrays are marked with the [] symbols, and objects are put between {} symbols. Each object can have multiple properties. A property is made up by the name of the property and the value of it. The name must be put inside quotation marks. The values can omit the quotation marks if they represent a number, else they have to have it too. Properties are seperated by a comma. A simple JSON file can look in the following way:

```
{
”userlist”:[
{”Name”:”Gergely”,”Age”:24,”Hobbies”:[”Games”,”Reading”],”Grades”:[5,3]},
{”Name”:”János”,”Age”:23,”Hobbies”:[”Bowling”,”Pool”],”Grades”:[5,5,4]},
{”Name”:”Attila”,”Age”:23,”Hobbies”:[”Blogging”,”Cooking”],”Grades”:[5,3]},
]
}
```

In the example, userlist is the sole property of the top level object. It holds an array of users, which have the properties of ”Name”, ”Age”, ”Hobbies” and ”Grades”. Properties in an array do not have to match, for example if ”János” has been graduated, his ”Grades” property can be omitted and a ”Job” variable can be introduced, although if a program tries to interpret it, it must be handled properly (as null variable or such).

Most mainstream programming languages have built in capability or at least a popular library to handle JSON files. They can be easily written to and from, since they have no need for serialization like a binary file. Their properties are easily read and translated into objects in the application. In Java, the JSON reader and JSON writer classes are responsible for this purpose, but in this project an extra library, named GSON was used. GSON simplifies the task of translating back and forth between Java and JSON objects. It can be as simple as declaring a GSON object, and pass

the object or list of objects to it, and it returns the JSONized version of it, which is ready to be written in a file.

4 APPLICATION ANALYSIS

4.1 Requirement Specification

The main functions of the application, as described by the client:

ID	Description	Priority
1	Receive Low-Energy Bluetooth transmissions	1
2	Parse transmissions into human readable values	1
3	Differentiate between measurements based on ATT UUIDs	1
4	Store the data on the phone in a persistent way (JSON)	1
5	Display the collected data on the screen of the phone	1
6	Draw trends of the collected data	2
7	Alert the user for measurements exceeding the limit values (sound or visual alert)	2
8	Synchronise the offline storage to online media	2
9	Post the collected data on Social Media if allowed	3
10	Based on the collected data, search and offer people with similar traits	3

Table 2 - Functional Requirements

The focus of the functions was decided by the priority given to them in this table.

4.2 Functional Specification

From the requirement specification, the following use case table was drawn up for the application:

#	Case	Precondition	Input	Description	Expected result	Exception
1	Log in user	Application started.	User credentials	Get the user from online media or offline storage (if no internet).	User logged in. Previous session loaded from file or from online media.	User not found. Bad credentials. User profile does not exist.
2	Discover ring	Application started.	Physical address of found device.	Start bluetooth discovery to pair the ring to phone.	Return paired device.	No suitable device found.
3	Setup ring	Ring paired.	UUIDs of listened services and characteristics.	Set the Client Configuration Characteristics for the characteristics to ”notify”.	Ring automatically broadcasts when characteristic is changed.	Service/characteristic is not found.
4	Listen for characteristic	CCCs set on ring.	UUID of characteristic and returned value.	The phone is in standby until the ring sends notification about a characteristic. Then poll the value.	Get byte array value of characteristic.	Characteristic is not found. Data not returned.
5	Save Measurement	Characteristic change got.	Byte array value of characteristic.	Format the byte array to the correct format according to the type of measurement.	Save human readable value for the measurement type.	Unexpected number format.
6	Save to file	File system available and new value received.	User profile and list of measurements.	Save the updated list of measurements into JSON file on the phones filesystem for the user.	Values succesfully saved.	Profile does not exist for user. No access/permission for filesystem.
7	Sync offline storage with online media	Internet available and new value received.	User profile and list of measurements.	Save the updated list of measurements into online storage for the user.	Values succesfully updated online.	Profile does not exist for user. No internet access.
8	Display value	New value saved.	Latest measurement.	Display the latest measurement on the phone screen if it is the requested measurement type.	Value displayed succesfully.	Null value received. Unexpected format received.
9	Draw trend	Measurement type available.	Currently displayed measurement type.	Draw a trend from the currently requested measurement type for the requested time period.	Trend succesfully drawn.	Measurement type is empty or not existing. Time period is invalid.
10	Post data online	Permissions for Social Media available.	Selected value for measurement type.	Post a message on selected Social Media page about latest measurement results.	Message posted succesfully.	No permission to post online.

Table 3 - Use Case Table

4.3 Modular diagram

The modular or package diagram shows the associations between the main components of the application. Each package represented can have multiple classes inside them. The following diagram (Figure – 1 Package diagram) is drafted based on the requirements of the application and the conventions of Android development:

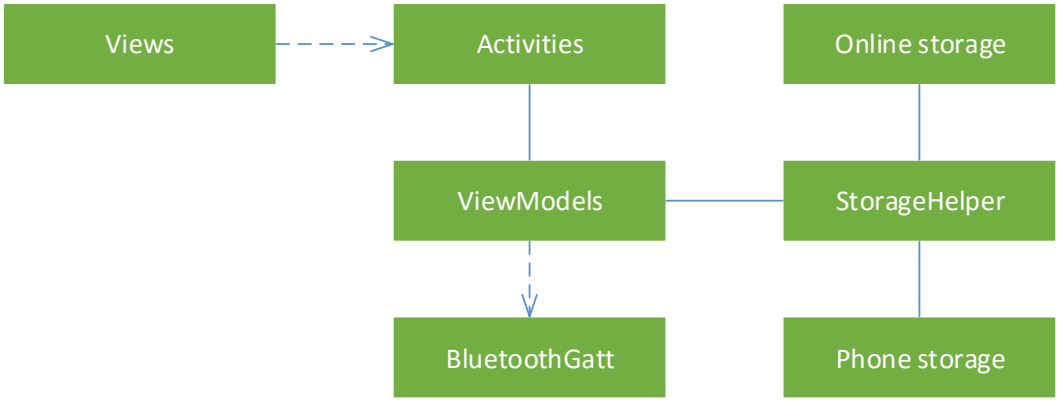

Figure 1 - Package Diagram

The Views on the picture are the collection of XML files which are translated by the operating system into user viewable objects. They describe the viewable objects in human readable UTF-8 string format. The Views are dependent on their corresponding Activities, who are serving as the connector to the application logic. They are used to handle the inputs and outputs from and to the Views, and process the data before sending them towards the ViewModels. The ViewModels can also send information to the Activities, for example when a value is received via the Bluetooth.

ViewModels serve as the representation of the data in the application. The Measurements incoming from the rings via the BluetoothGatt classes are initialized as ViewModels of their corresponding type, and saved into lists on the phones memory. Afterwards, these lists are saved into the active users Profile (which is also represented as a ViewModel), and then they are sent to the Storage Helper classes, which as their names suggest, serve as a mediator between the memory and the physical or online storage. Information is translated here into a JSON format, and saved into the hard drive o the phone, or synchronised to the online media.

4.4 GUI design

The following GUI mock up (Figure 2 – GUI Plan), which was used to plan the application layout and design, was received from the client:

Figure 2 - GUI Plan (align left correctly the figure text)

The current design is simplistic and focuses on presenting easily readable data. The main display shows the currently tracked health data value, along with a visual representation of the status of the value (the heart is green if the value is in the acceptable range, and turns red step by step when it is out of this range, finally getting fully red in the danger zones).

Beneath the value display, a graph view shows the measured values the trend of the health data for the last few minutes. This can alert the users about jumps in the values.

An action bar is built into the top of the application. From here, the user can switch the currently visually tracked data or change the settings of the application (change the good/danger limits of the app, for example).

5 APPLICATION DESIGN

5.1 Hardware

The hardware section of the project is called the Ting-E-Ling Ring. It is a small ring worn by the user, which reads the previously mentioned health data, and uses a Low-Energy Bluetooth technology to transmit these measurements to the smart devices. The data is sent in ATT attributes to the phone. The topic of this thesis, the phone application handling that data, only needs to care about receiving that data and processing it afterwards.

5.2 Use case methods

Based on the functional specifications and the table of use case, the description of each use case method was made.

The first step in using the application is to register a Ring to it. This can be done automatically by scanning for Low-Energy Bluetooth devices in the vicinity with the matching service and characteristic identifiers, or by selecting visible bluetooth devices from the phone. The connection itself is made by the application, and it notifies the user about errors, such as if the health services are not found on the connected bluetooth device.

In case of multiple available bluetooth devices, the currenly connected one must be unregistered from the application before a new one can be paired.

After registering the Ring, the software automatically listens to low-energy transmissions coming from it, and sorts the transmissions based on their type: Pulse, Blood Pressure and Body Temperature. The measurements are immediately stored on the Offline Storage of the phone in JSON files.

The user can select the currently displayed data type, which is viewable on the main screen of the application. By default, it shows the latest data got from the ring. It also draws a graph which visualises the trend of data in the past minutes.

Past time periods can also be selected for seeing raw measurements and drawing trends about them.

If the phone has internet access, and the user gives the permission, it automatically synchronises the offline storage to the online storage. It enables to access the data from multiple smart devices and even give the possibility of remote monitoring of the health of an individual for a doctor.

With internet access, the user can allow the application to post their collected health information on an online social media, like Facebook or Tinder. With this, other users of the Ting-E-Ling also can be found and get connected to the user. There can be implications of directed search of partners with similar data, to find training buddies, friends or even love interests.

The Actors identified for the project are:

- The User
- The Ting-E-Ling Ring

The main Functions identified for the application:

- Registering the ring
- Unregistering the ring
- Managing the ring
- Storing the data
- Querying the data
    - Drawing trend from the queried data
    - Posting the queried data on Social Media
- Synchronising the offline storage with the online storage.

Figure 3 shows the summary of the actors and functions of the project:

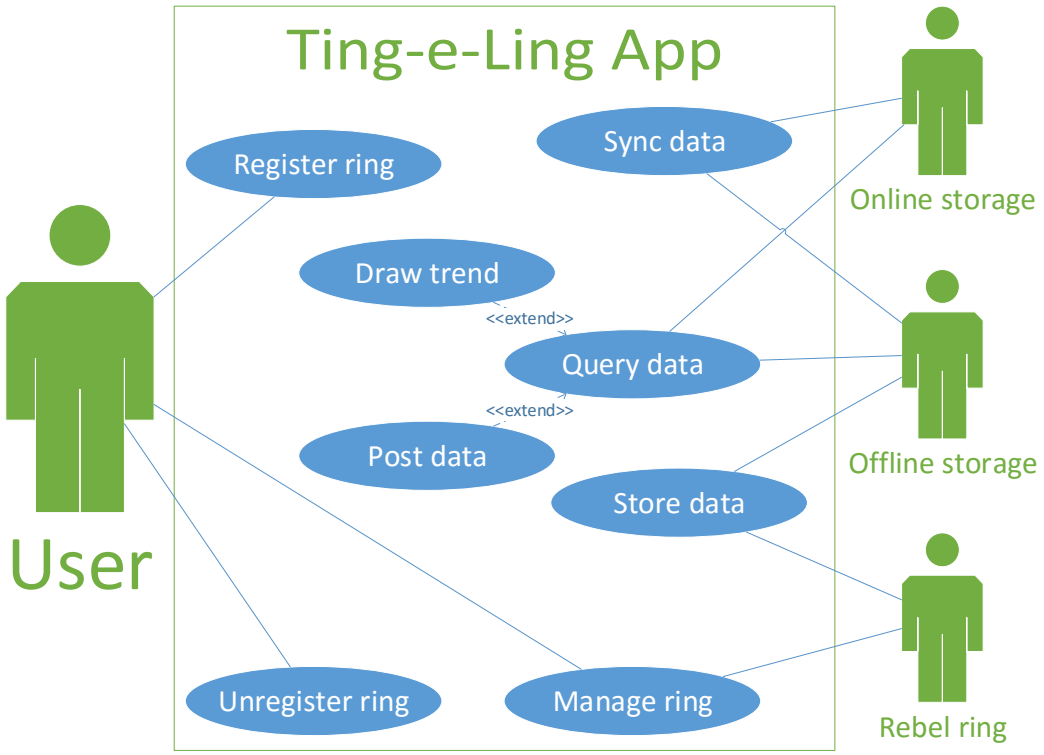

Figure 3 - Use Case Diagram

The user can register and unregister the ring, and manage the permissions given to the application. The ring sends the data to the phone for Offline Storage. The Offline Storage is then synchronised with the online storage. Data is queried by the phone based on the currently viewed health information, and shown on the screen and a trend is drawn from the past minutes measurements. The stored data then can be posted on Social Media.

5.3 Data Description

Based on the research done in Chapter 2, the following attributes were collected to describe any given measurement:

1. Name: the name of the measured physical data
2. Unit: unit of the data
3. Data type: the type which is used to store the value.
4. Minimum (Min): Minimum allowed value for the data (needed to set limits on trending representations).
5. Maximum (Max): Maximum allowed value for the data (needed to set limits on trending representations).

6. Good: a pair of values describing the healthy range of values.
7. Danger: a pair of values which sets the limits on low and high dangerous values. The first value shows the lower danger limit (below is unhealthy), the higher value shows the higher limit (above is unhealthy).
8. Value: the measured value.
9. Time: the measurement time.

These attributes are the basis of the model of the data represented in the application.

The actual values and limits which are not measured, are shown in the following Table 4:

Name	Unit	Datatype	Min	Max	Good	Danger
Pulse	bpm	integer	0	350	50-90	40/160
Blood pressure (systolic)	mmHg	integer	0	300	90-119	80/140
Blood pressure (diastolic)	mmHg	integer	0	300	60-79	50/90
Body temperature	°C	floating point	20	60	35-37	35/38

Table 4 – Attributes of data

Minimum and maximum values are set within sensible ranges of values. The good parameter is a pair of value, which shows the range of healthy values. The dangerous parameters show the lower and upper limits of unhealthy values.

5.4 Data Model

Based on the data described in the sub-chapter 4.4, the following model is drawn up to describe any measurement and any measurement type:

Measurement {

/* The measurement holds the individual measurements and a timestamp. It is also linked to the measurement type.*/

value: float; // the measured value coming from the ring
time: datetime; // the time of measurement, from coming from ring or the time of transmission

}

MeasurementType{

/* The measurement type holds common information for measurements. */

name: string; // user readable name for type
unit: string; // unit of measurements contained in type
min: integer; // minimum sensible value for type
max: integer; // maximum sensible value for type

```
        good: float[]; // range of values considered healthy
        danger: float[]; // limits of values considered dangerous for health
        measurements: Measurement[]; // list of measurements contained in this
        type

}
```

Profile{

```
/* The Profile holds the different types of measurements for a given user. This is
the highest level of the data, and this will have the capability to save these lists.*/
        name: string;
        measurement_lists: MeasurementType[];
}
```

These basic data models were extended during the implementation according to the needs.

5.5 Class diagram

Based on the data model and the functional specifications, the following figure (Figure 4 - Class Diagram) was drawn, containing the planned properties and functions (both of them with their expected types) for each class. The classes are sorted into their corresponding packages, and the associations between each class is marked.

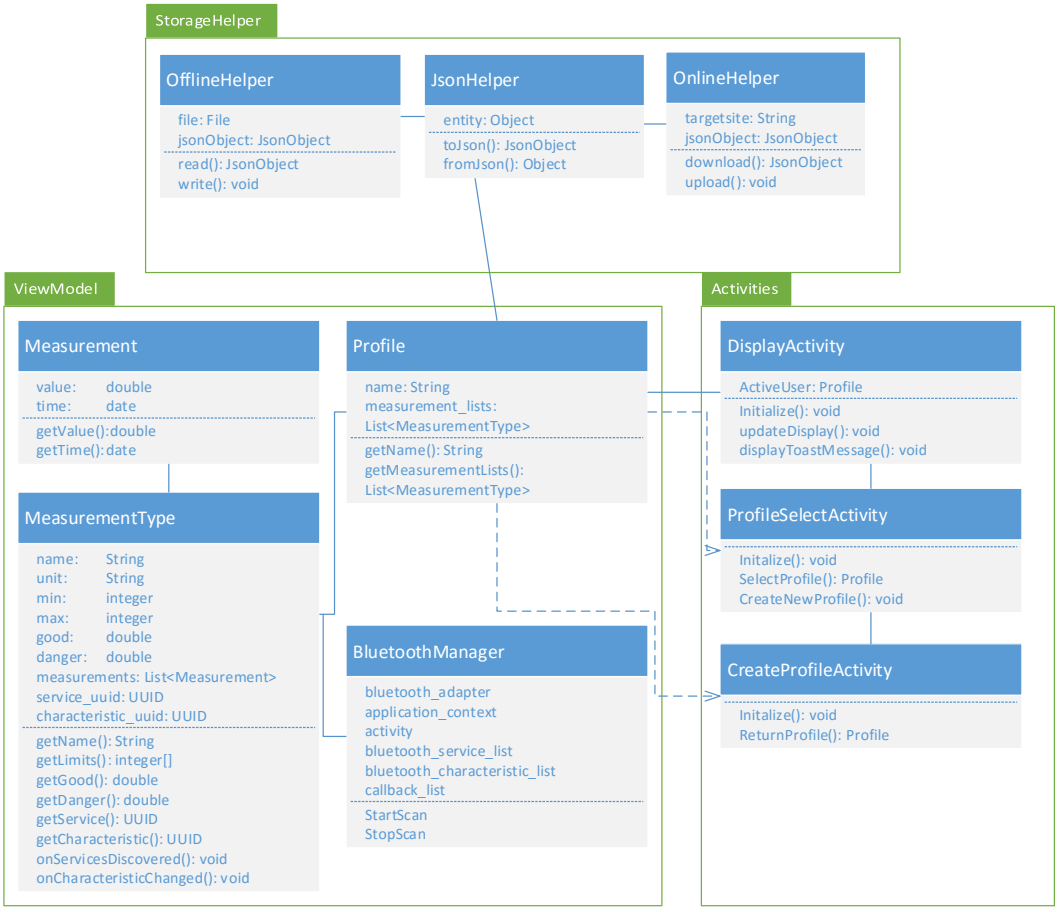

Figure 4 - Class Diagram

The lowest level of the application is the StorageHelper package. It includes three class. The Online- and OfflineHelper classes are similar in functionality. They both get a target (a site or a file), and with their functions they transmit the data to and from the target. They both convert their data into JsonObjects, and send it into the JsonHelper class, which job is to convert the received JsonObjects into Profile objects which are used within the application. Naturally, the process works in the reverse, when the application saves, Profile objects are converted into JsonObjects and then uploaded or saved.

Profile objects keep information about the user currently using the application. The hold all the MeasurementLists and Measurements associated with the user. The current user is set within the application, and all new measurement will be associated with his/her Profile. On creating a new user, a new Profile is created with empty lists. On loading from the storage, the user is initialized with all the previously saved

informaton. The MeasurementType lists are loaded with the measurements, and saved as BluetoothGattCallbacks. MeasurementTypes hold the corresponding list of Measurement lists of their type, and they have special callback functions which are used to set up the ring (for example, after service discovery, mark the characteristics of measurements with the notifications, so the ring knows which characteristics it needs to send broadcasts about) and listen to characteristic changes. Active MeasurementTypes are saved in the BluetoothManager class, which is responsible to initialize the Low-Energy Bluetooth, and keep track of the notifications of each MeasurementType. It can also start or stop the discovery of new devices in case there is no ring paired to the app. The three Activity existing with the application are all connected to the rest of the software through the Profile class. The application launches with the ProfileSelectActivity, where the user can pick the Profile which will be used as the active profile. Or he/she can create a new, empty profile through the CreateProfileActivity. After selecting the Profile, it is transmitted to the DisplayActivity, which role is to keep the GUI up to date with the currently active MeasuremenType. It also keeps track of the active user. The functions associated with display (such as Toast messages, deciding the color of the heart on the screen and updating the value) all reside within this class.

5.6 Sequence diagrams

Expanding on the class diagram and the specifications, sequence diagrams were drawn. Sequence diagrams are charts, which show the flow of information and lifetime of the objects in the project. Usually they show the flow from actor to actor, with the classes in between where the data is passed through. The following three diagrams (Figure 5, 6 and 7) were drawn for the projects more important sequences:

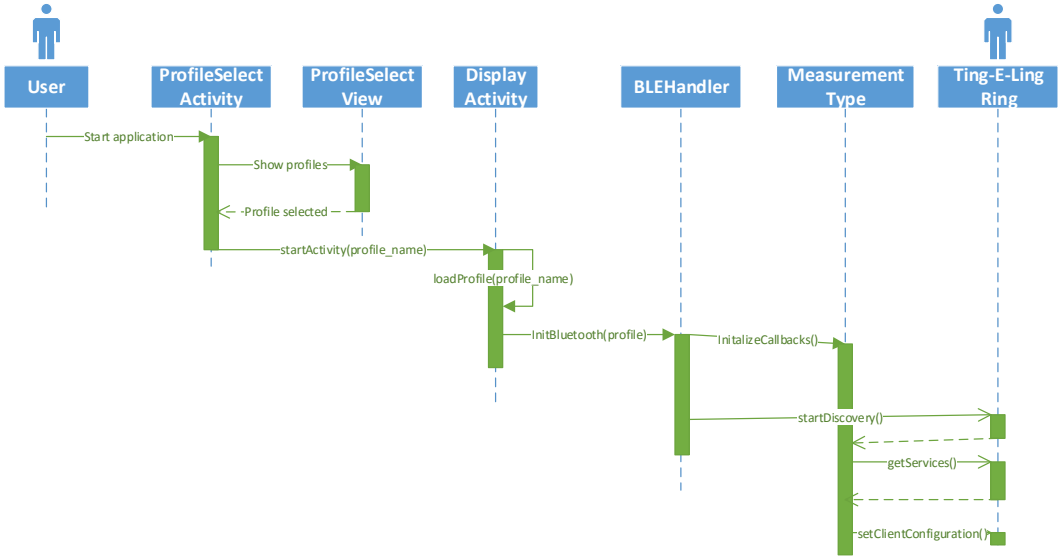

Figure 5 – Startup Sequence

Th sequence in Figure 5 shows the functions which the application goes through each startup. First the user is navigated to the ProfileSelectActivity. It searches for the available profiles, and displays it on the ProfileSelectView, where the user can select his/her profile. It will be used throughout the application. After profile selection, the DisplayActivity is loaded with the profile. It initalizes the Bluetooth, where the MeasurementTypes are read from the profile and initalized as callback functions. The Bluetooth then starts the discovery of devices. If a suitable ring is found, it is paired, and a callback is sent to the MeasurementTypes, that they can scan for services. Once the services are found, the MeasurementTypes loop through the characteristics of the services, and set the Client Characteristic Configuration for each desired characteristic. After this setup, the ring will automatically broadcast characteristic updates.

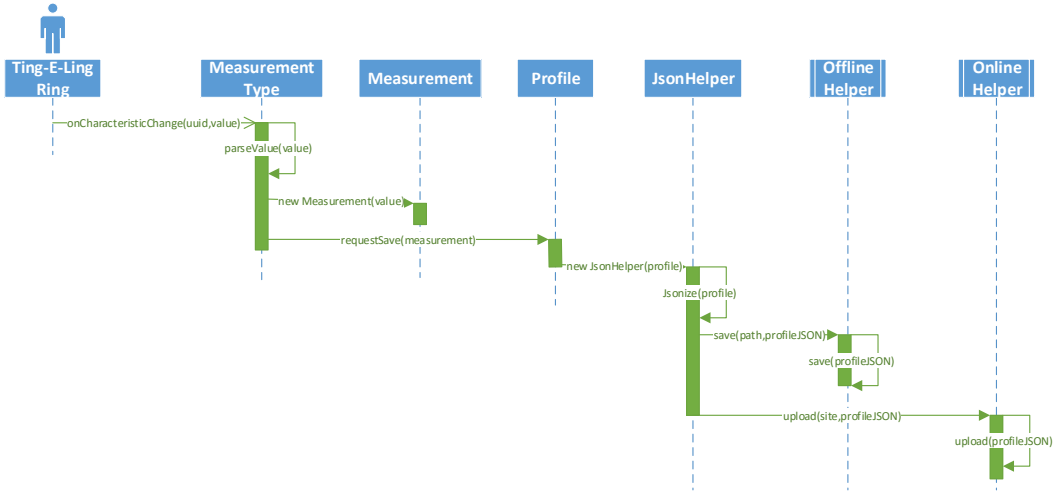

Figure 6 - Measurement save sequence

The sequence in Figure 6 shows the process, when the ring has a new measurement ready. It broadcasts a message about a characteristic update, which the smart phone picks up, since the CCC is set. The phone parses the value into usable format (integer or double), and initalizes a new Measurement object. The measurement is saved into the MeasurementType-s list. Then, the user's profile is notified about a new measurement. The updated list of measurement is sent to the Profile object, which then creates a new instance of the JsonHelper class. This class turns the whole Profile object, and all of its subobjects into a JsonObject. It is then sends the JsonObject to the OfflineHelper, which saves it into a .json file onto the phone's offline storage (it will prioritize saving onto the external drive, but if it is not attached, then it will save into the internal storage). The default path is the Android/data/com.delektre.tingeling/files folder. The default file name is the username plus "_profile.json", for example Gergely_profile.json. After saving offline, the application will check if there is internet connection and the online storage is enabled. It will send the JSON format data to the selected site, where a webservice will pick up the JsonObject, and handle the saving. The user gets no extra notifications about this sequence process, since it can happen dozens of times in under a minute, with each new measurement.

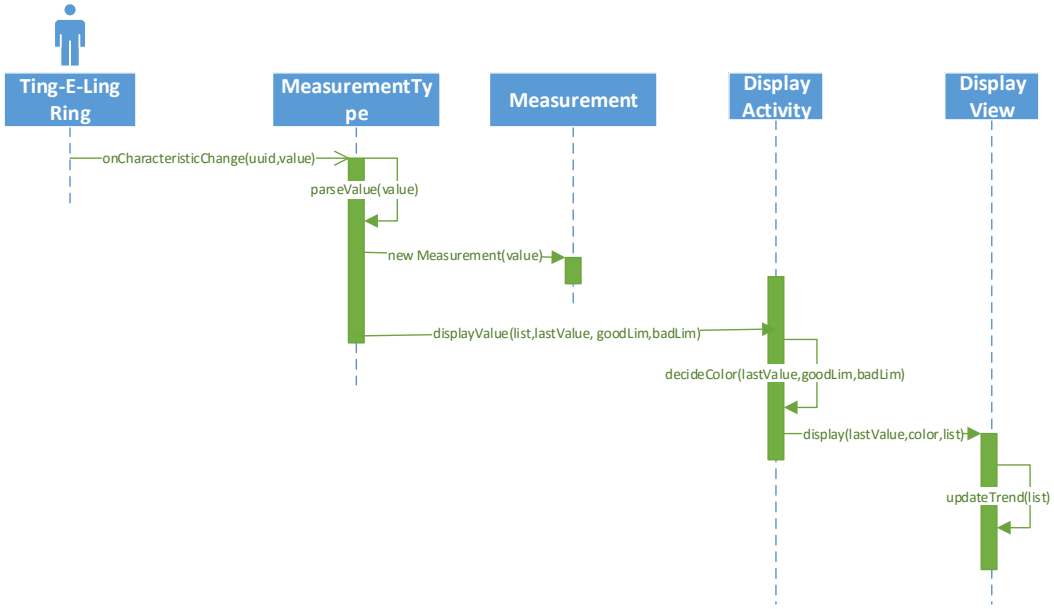

Figure 7 - Update display sequence

The sequence (Figure 7) is started as well, when a new value is created. For details the initial parts (ring sending data, MeasurementType creating a new Measurement object) please refer to the previous sequence diagram (Figure 3). The MeasurementType object, after sending request to the Profile to save, forwards the new Measurement object to the DisplayActivity, which handles functions related with updating the GUI. The DisplayActivity gets the Good and Bad limits from the MeasurementType, along with the value and the list of previous measurements, and decides the color of the warning display with a formula. The decision is made by the following way: if the value is within the range of Good values, the color is green. If not, find the closest Good limit, substract the value from it, and take the absolute value of the result. With this value, a percentage of green and red value mixture can be calculated for the RGB color (red: (255 * base) / 100; green: (255 * (100 - base)) / 100, blue is 0 always). After the color is set, send the value, the color, the list of measurements and the latest measurement to the DisplayView. With the usage of the GraphView library, a trend can be drawn from the list (the used measurements are the last 200 by default). The latest value is displayed in the center of the heart on the screen, and the color of the heart is modified to be the calculated color.

6 IMPLEMENTATION

The two major challenges in the implementation of the project were the Low-Energy Bluetooth (BLE) connection and choosing the correct model of objects to use for the data. Special consideration had to be taken for making the application responsive, and minimizing the amount of load it presents to the hardware.

6.1 Implementing the Bluetooth connection

The bluetooth code was seperated into two sections. The first section is the general setup of the bluetooth interface, including the enabling of the adapter and setting up the listeners. The second section was incorporated into the MeasurementType class, since they receive the data individually from the different services. For this, they are subclassing the BluetoothGattCallback object, which allows the MeasurementType class to be target of callbacks whenever the data with their service UUID arrives on the bluetooth interface. For accessing these functions, the MeasurementType have to override its superclass's 3 functions, which functions are shown in Figures 8, 9 and 10.

The first function is called when the GATT transmission returns with a new status. It provides the transmission, the status of the current transmission (most common is success or failure, but there are several other statuses in between, like insufficient authentication or request not permitted) and the new status (which can be connected, connecting, disconnected and disconnecting).

For the application's purpose, most important is to get the services after the Ting-E-Ling Ring has connected to the smart phone. The discovery of the services must be turned on constantly while the data transmission is happening.

```
// Each BluetoothGattCallback and its extended classes have to implement the onConnectionStateChange function
// This function is fired, if a suitable BLE device is found and the connection has returned a state.
@Override
public void onConnectionStateChange(final BluetoothGatt gatt, final int status, final int newState) {
    if(status == BluetoothGatt.GATT_SUCCESS) {
        switch(newState){
            case BluetoothGatt.STATE_CONNECTED:{ // display message of succesful connection.
                gatt.discoverServices();
                btadapter.stopLeScan(scancallback); // stop the discovery of new devices.
                display.ToastDisplay("Connection succesful to "+gatt.getDevice().getName()+"!");
            }break;
            case BluetoothGatt.STATE_DISCONNECTED:{ // display message if we disconnected from device
                display.ToastDisplay("Disconnected from "+gatt.getDevice().getName()+"!");
                btadapter.startLeScan(scancallback); // start the discovery of new devices again
            }break;
            default:{ // if we run into any other state, display it, so it can be debugged.
                display.ToastDisplay("Unhandled state: "+newState);
                break;}
        }
    } else if(status == BluetoothGatt.GATT_FAILURE){ // if the connection failed, display message
        display.ToastDisplay("Connection failed to "+gatt.getDevice().getName()+"!");
    } else {                                         // any other case, display status id for debug
        display.ToastDisplay("Unhandled status: "+status);
    }
}
```

Figure 8 - Connection state change function

Tthe measurement types were created as Bluetooth Callbacks, so they can be easily notified when a new measurement concerning them comes in a transmission. When the GATT fires the state change event, this functon checks the status and the new state. If the connection is succesful, the device turns off the discovery of new devices and starts to listen to service transmissions from the ring. The user is notified about the state of the connection. The switch statement inside the callback function provides an easily extendable platform if the need to handle different statuses arise. The outer IF statement only should ever respond to GATT_SUCCESS and GATT_FAILURE, so if there is some other status comes, we can see if there is a problem.

The second step in getting the data is to set which characteristics are the application is listening to in the transmission.

After the gatt.discoverServices()-command is issued, the event ”service discovered” is fired when the application receives the GATT transmission which has any services. With the GATT handle, the application can go through each service which was included in the transmission. At this point, flags can be set in the service of which characteristics is requested by the application. After setting these flags, whenever the characteristic changes, it will issue a new event for the application.

```
// The onServicesDiscovered must be implemented in every BluetoothGattCallback and its extended classes.
// This event is fired after the phone has connected to the ble device, and discovered its services.
@Override
public void onServicesDiscovered(final BluetoothGatt gatt, final int status) {
    for (BluetoothGattService service : gatt.getServices()) { // we go through all the services
        if(service.getUuid().equals(serviceuuid)){ // if it is the actual measurement, we start to configure
            BluetoothGattCharacteristic characteristic = service.getCharacteristic(characteristicuuid);
            if(characteristic!=null){ // if we find the correct characteristic. Else, we dont want to do anything.
                UUID ccc = UUID.fromString("00002902-0000-1000-8000-00805f9b34fb"); // The CCC uuid, 0x2902
                BluetoothGattDescriptor descriptor = characteristic.getDescriptor(ccc); // we get the CCC descriptor
                // the next two lines modify the CCC, so the ring will send notifications of the value without querying
                descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
                gatt.writeDescriptor(descriptor);
                // the next line is needed, so the phone is listening to the broadcasts of the characteristic.
                gatt.setCharacteristicNotification(characteristic, true);
            }
            break;
        }
    }
}
```

Figure 9 - Service discovered function

For each measurement type, the application listens individually only for their services. Once the service is found, it loops through the characteristics, until it finds the expected one in the measurement type. Then it enables the notifications for the characteristics, which enables the software to listen for the characteristic value changes.

The third function to be overwritten is the mentioned characteristic change callback. It is called whenever the previously mentioned event happens. Since the data is transmitted in a binary format, it has to be converted to double. After this,a Measurement object is initalized with the value and the current timestamp, and saved to the MeasurementType's list of values. After this, the displayed value is updated, if the user is currently tracking it.

```
// The OnCharacteristicChanged function must be implemented in every BluetoothGattCallback and its extended classes.
// The event is fired, when the ring sends a notification of a value, which has the CCC set to enable_notifications.
@Override
public void onCharacteristicChanged(BluetoothGatt gatt, final BluetoothGattCharacteristic characteristic) {
    // We have to parse the byte format value into integer. The ring always sends 8-bit format integers.
    int value = characteristic.getIntValue(BluetoothGattCharacteristic.FORMAT_UINT8, 1);
    // We initalize a new measurement object, and we save it to the measurement list.
    Measurement measurement = new Measurement(value, new Date());
    if(measurement!=null){ // safety double check, that the measurement object was created.
        measurements.add(measurement);
        // Finally, we update the display of the value, if it is the currently displayed measurement type.
        if(displaycaller.currentDisplay.equals(this)){
            displaycaller.ValueDisplay(measurement.getValue(), good, danger);
        }
    }
}
```

Figure 10 - Characteristic updated function

When a characteristic changes, the application receives the data in a byte array. Since it is known, that the Ring sends the data in a single byte, unsigned integer

format, the application has no need to check the value besides getting it and saving it into a new Measurement.

6.2 Graphical Layout

Based on the Graphical User Interface plan received from the client (see Figure 2 for reference), a straightforward and simple GUI was implemented to serve the purpose of displaying the measurements to the user. The result did not go through many iterations, and it roughly looked the same for the major part of the development as in Figure 11.

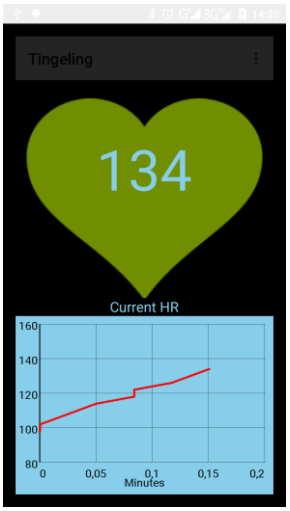

Figure 11 - GUI Layout

The top action bar is used for the main interactive features. The heart in the focus of the screen displays the currently selected value. It changes its color based on how close the value is to the healthy range or the dangerous limits. The healthy values are displayed as vibrant green, and the unhealthy values slowly turn into red color, with the combination of the two colors in between.

6.3 Testing

For testing purposes, manual black-box testing was used. When a block of the application was made with well seperated inputs and outputs, different inputs were inserted into it, and the output was observed. Since the outputs were easily understandable, conclusions could be drawn about if the module is functoning properly,

or if it has some malfunctions, like running into exceptions or resulting in unexpected results.

The following main cases were tested for:

Switching between activities and passing variables between them

Problem #1: Objects cannot be simply passed with Intents.

Solution #1: Objects must implement the Serializable or Parcelable interface. Serializable objects are automatically serialized, which can be sent with Intents. Parcelable is a configurable serialization (marhaling and unmarshaling in the jargon) in Android development, with specific parts serialized. It is more desired than the Serializable interface, since it generates less garbage in the memory.

Connection to Bluetooth Device

Problem #1: Checking if the handheld device is capable of Low-Energy Bluetooth communication.

Solution #1: After researching, it was found out, that all devices are capable of BLE transmissions, so checking for this problem is not necessary.

Problem #2: Establishing the correct communication type.

Solution #2: Initially, the standard Bluetooth libraries were used to try and make the connection to the ring. After unsuccesful attempts, and research on the topic, the BluetoothGatt and its dependent libraries were found, and used to provide correct results.

Problem #3: Data arriving from the Ring is encoded in byte arrays.

Solution #3: The Gatt libraries provide a characteristic converter to different standard formats, which can be used to convert the byte arrays, IF their expected standard format (integer, floating point etc) is known already.

Saving and loading from JSON files

Problem #1: Converting from and to JSON files results in a very complicated code, where nesting objects within objects often not clear and results in translation problems when converting back to objects.

Solution #1: The GSON library automatizes the conversion between Java and JSON objects. It provides a standardized format for all objects to use, and shortens the customized code parts significantly, increasing code readability.

7 SUMMARY

Since there was no previous experience in Android development and in wireless communications, parts of the application challenged me to learn new things.

It was especially hard to get the Bluetooth communications right. At first, it was challenging to use the normal Bluetooth connection of the phone, which of course resulted in failure, as no transmissions came through that channel. After several days, I realized the need for the Low-Energy Bluetooth and the ATT protocol.

A lot was learned about the usage of callback functions. I have used them before, but never really relied on them too much. In Android development, they play a big part, as a lot more responsive and interactive applications can be made with listening to every (necessary) event and interaction the user makes.

I got familiar with the general Android development architecture. This was an easier area, since the structure is very similar to me, if not the same as the MVC model which was used previously in web projects. The few little differences were, for example, how the strings of the UI are preferred to keep in a seperate XML file and that it is easier to create an interactive app than an interactive page, since there is no need for JavaScript and the user interactions can be listened directly in the Activity classes.

Finally, I learned a lot about documentation during the writing of this document. I considered, and still consider, that to be one of my weak points, but I believe that I have improved upon it significantly since the beginning of the development.

To summarise the experience, the topic of the project was really interesting, as it included a bit of everything which I had already learned about, while it also offered new areas in which I could improve my skills. I am sure that the knowledge I earned during the development and documentation will come in handy during my future career.

8 CONCLUSION

As said before, general awareness on personal health is increasing. The trends show that there are more and more people acquiring some form of health-monitoring device. But as also mentioned, these devices are still bulky and often inconvenient for everyday use. The Ting-E-Ling Ring aims to change that. It will offer a new level of personal information ready at the most commonly used devices, while retaining a small frame. The user can just put the Ring on IN the morning and ”forget about it”. The smart device will take care of the rest of the work, enabling to track the health status of its owner, alert him/her in case of problematic values, or even post the user’s good health or regular workouts online. This can also promote health in a higher number of people.

To improve the Ring the possibilities are numerous. There might be future iterations with more diverse sensors and even smaller hardware.

For the Android application and my personal knowledge, I plan to look into the Widgets of Android OS. I believe the application would benefit from a widget, so the user can use it in an even more convenient way. This would also provide an area for me to learn about. There might be usage for remote tracking as well, primarily for doctors, who can be alerted automatically if their patients’s ring sends alarming information. It could even provide an easy connection between the doctor and the patient!

For my closing words, I believe, that if people will continue to be interested in their health (which of course, is everybody’s concern), the Ting-E-Ling Ring has a future. It offers what people are looking for in a convenient way, at their fingertip.

9 LIST OF REFERENCES (FIX ACCORDING TO ANOTHER EMAIL)

(1) *Bluetooth.org: Generic Attribute Profile (GATT)*. (2016). Forrás: Bluetooth.org: https://developer.bluetooth.org/TechnologyOverview/Pages/GATT.aspx

(2) Attribute protocol and Generic Attribute Profile (2016). *https://epxx.co/*. Forrás: https://epxx.co/: https://epxx.co/artigos/bluetooth_gatt.html

(2) *Practical Clinical Skills: Blood Pressure Measurement*. (2016). Retrieved from Practical Clinical Skills: http://www.practicalclinicalskills.com/blood-pressure-measurement.aspx

(4) *WebMD: First Aid - Body Temperature*. (2014. November 14). Forrás: WebMD: http://www.webmd.com/first-aid/body-temperature

(5) *WebMD: Heart Disease Center - Pulse Measurement*. (2014, September 9). Retrieved from WebMD: http://www.webmd.com/heart-disease/pulse-measurement

(6) *WebMD: Hypertension/High Blood Pressure Health Center - Know Your Numbers*. (2014. July 27). Forrás: WebMD: http://www.webmd.com/hypertension-high-blood-pressure/guide/diastolic-and-systolic-blood-pressure-know-your-numbers