myHealth: a Self-Adaptive Android App in the domain of eHealth

Luigi Bisceglia*, Daniele Malitesta† and Berardino Prencipe‡
Department of Electric and Computer Engineering
Polythecnic University of Bari
Bari, Italy
Email: *l.bisceglia@studenti.poliba.it, †d.malitesta@studenti.poliba.it, ‡b.prencipe1@studenti.poliba.it

***Abstract*—In this paper we propose myHealth, a Self-Adaptive Android application in the domain of eHealth. Far from being a simple emulation of other similar well-known apps on Google Play, it is more a *collector* of user's health parameters coming from those similar apps. A user model is built accordingly to give suggestions about health.**

***Index Terms*—eHealth, self-adaptive, user model, collector, environment, third-party apps.**

I. Introduction

There has been a huge spread of devices such as smartphones and tablets in the last decade. According to Cisco[1], by 2020 there will be 5.5 billion mobile users, representing 70 percent of the global population [1].

One of the most important reason why users want a smartphone can be found in applications, whose number has grown exponentially in the last few years. Currently, the total number of apps on Google Play is about 2.5 million, with 95.1 percent of free ones [2].

eHealth applications have gained huge attention recently, because they represent a fast and easy way to take care of one's lifestyle. As a matter of facts, a study [3] reveals that approximately 3.2 million deaths every year are caused by insufficient physical activity. It is indeed quite challenging to motivate patients to lead an healthy life, especially because healthcare providers see them for only a brief moment of time; the rest of their daily life is left to uncontrolled habits [4].

In addition to this, software systems have evolved dramatically in the last few years, becoming more versatile, flexible and more prone to adapt to different operational contexts and environment. This has led to a new promising direction for these systems, which is known as *self-adaptation* [5].

Driven by these two domains, we decided to implement *myHealth*, a Self-Adaptive Android application in the domain of eHealth. However, our intention was not to simply build another eHealth app, especially because there are plenty of them on Google Play. What is more, lots of these solutions are often closed to the outside world and do not allow in any way to be interfaced.

To be more specific, we wanted to create a sort of *collector* for parameters related to health - *e.g.* eaten food, drunk water, step counting, spent km on a bike, measured weight - coming from already existing apps. Then, a user model is built accordingly. It describes the user in a *machine-readable* way, which is really helpful because it can be compared with *gold standards* data present in literature. So, we implemented a kind of health *recommendation system* which gives tips and suggests new apps to download from the store.

It is quite easy to use myHealth, thanks to its user-friendly and intuitive graphical interface. The only thing you need to do is to associate third-party apps you already have on your smartphone. You do not need to select or set any goal.

At the end of the day, our application is a *virtual personal trainer* who deeply knows you because it looks silently at your habits and lives inside your smartphone *environment*.

The rest of this paper is organised as follows. Section II focuses on the main topics of this project work, *i.e.* eHealth and self-adaptive applications. Section III is a complete description of our system architecture, analyzing every component and its implementation. Section IV deals with the application testing phase and the obtained outcomes. Section V presents works in literature which were really influential on our project work. Finally, Section VI tries to summarize the whole report, underlining strengths but also weaknesses of our application, which could be improved in future implementations.

II. State of art

This section aims at exploring the two main domains of this project work, *i.e.* eHealth and self-adaptive applications. Everything that has been written here comes from authoritative scientific papers inside digital libraries, such as IEEE Xplore Digital Library[2], ACM Digital Library[3] and Elsevier[4]. Another important source - especially for eHealth topic - was the Journal of Medical Internet Research[5].

This section is structured as follows: the first and second parts present the eHealth and self-adaptation domains respectively, considering their purposes and open issues, with some real world examples.

[1]https://www.cisco.com/

[2]https://ieeexplore.ieee.org/Xplore/home.jsp
[3]https://dl.acm.org/
[4]https://www.elsevier.com/
[5]https://www.jmir.org/

A. eHealth

In literature, eHealth is mostly defined as a broad range of medical informatics applications for facilitating the management and delivery of health care [6]. In a broader sense, its main goal is to improve health care locally, regionally, and worldwide by using information and communication technology [7]. Even the World Health Organization could not help recognizing the importance of eHealth in modern society, stating that it is *(...) the cost-effective and secure use of information and communications technologies in support of health and health-related fields, including health-care services, health surveillance, health literature, and health education, knowledge and research* [8].

Besides the general definition, eHealth consists of different domains of application, each of them with its peculiar technologies. For istance, mobile Health (mHealth) has become one of the leading fields, especially thanks to the increased availability, miniaturization, performance, enhanced data rates, and the expected convergence of future wireless communication and network technologies around mobile health systems [9].

mHealth implies the use of mobile devices to collect patients' health parameters in real-time and store them in servers connected to the Internet. As a matter of facts, IoT technologies and approaches are one of the possible solutions in this context. The use of wearable sensors helps in monitoring patients' parameters such as sugar-level, bloodpressure, heartrate, etc [10].

In addition to all these solutions, there is still a huge and increasingly popular world to take into account, *i.e.* the one of eHealth applications for smartphones. Differently from the systems described above, eHealth apps mostly concern users' lifestyle.

1. eHealth for Smartphones. Papers in literature claim [11] there is a strong cause-and-effect relationship between noncomunicable diseases - NCD[6] - and various lifestyle modifications such as diet, physical activity, and sedentary behaviors. In such cases, it seems that only daily prevention can play a key role in treating NCD in an optimal way.

What is more, nowadays mobile devices and particularly smartphones are integral part of people's life and assist users in many ways with always innovative applications [11].

It has been found out [12], [13], [14], [15] that smartphone apps have the ability to facilitate behaviour change in users. There is evidence [16], [17] that young adults improved their eating habits and physical activity thanks to these solutuions, while there are some ongoing researches [18] to understand the possible impacts on elderly people.

The most common eHealth applications features are summed up here [4], [19]:

- they cover a wide range, including improving fitness activities, weight goals, nutrition and diet, sleep, relaxation, general healthcare information, pregnancy and alternative medicine;
- once they are downloaded, basic data are required to user such as gender, date of birth, weight, and height; they are used to build a model upon user;
- they collect data from smartphone inbuilt tools - such as GPS, accelerometer, microphone, camera - to measure health and fitness parameters and use them to provide individual plans and frequent feedback based on users' goals;
- some of them are also able to connect to wearable sensors like wristband sensor, heart rate sensor, belt sensor, shoe sensor, or more recently, smart clothing with wearable sensing technologies;
- some of them include a storyline that the user is immersed into, whose goal is to distract them while they are training;
- lots of them have virtual coaches which encourage the users so that they do not ever give up, like in real life;
- they display user calculated parameters in dashboards, such as total workout data (miles, speed, reps), body weight data, and sleep-quality data.

Authors in [4] also list what a good eHealth application should offer to users:

- ✓ have a user-friendly interface;
- ✓ offer a free trial version;
- ✓ easy to initiate and reliable during activity;
- ✓ allow for goal setting;
- ✓ provide real-time personalized feedback;
- ✓ customized to user's skill and fitness level;
- ✓ offer expert consultation;
- ✓ incorporate evidence-based behavior change tecniques;
- ✓ sync with other health and fitness apps and devices to allow easy sharing of statistics;
- ✓ support social networking;
- ✓ provide periodic summaries [20], [21].

eHealth applications can be found both on App Store[7] for iOS devices and on Google Play[8] for Android devices. However, the focus of this project work is on the Android application we built. So, this first part will end taking into account Android apps in the domain of eHealth.

2. Examples from Google Play. If we summed the number of Android applications belonging to HEALTH AND FITNESS, MEDICAL and FOOD AND DRINK categories, we would obtain a total amount of approximately 200 thousand apps, which is a quite remarkable result thinking that the most popular Google Play category - *i.e.* EDUCATION - counts approximately 220 thousand apps [22].

In what follows, some of the most popular Android applications in the domain of eHealth will be briefly presented. They belong to the Google Play categories mentioned before and they are in Italian language - just to keep things coherent with our project work.

- *Samsung Health* (Fig. 1), score = 4.3/5. It registers all daily activities and habits in order to make you lead

[6] *e.g.* cardiovascular and metabolic diseases.

[7] https://www.apple.com/lae/ios/app-store/

[8] https://play.google.com/store?hl=it

an healthy life. Main features are: step counter, spent distance on a bicycle, outdoor and indoor sports. User can provide info about the food she eats and water she drinks in order to receive diet suggestions. It also uses smartphone and third-part sensors to get some body parameters such as heart rate and blood pressure. What is more, user can enjoy free exercises programmes, as well as ask questions to experts.

- *Fitbit* (Fig. 2), score = 3.9/5. One of the most popular health and fitness apps on the Google Play Store. User can set a certain number of daily goals to be reached, such as sleep hours and quality of sleep. It measures all parameters linked to physical activity, like free running or generic training to calculate the impact of sport on user's daily life, how good or how bad it is. It allows to communicate with friends to compare performances and challenge them. It also checks user weight and level of hydration through food she eats and water she drinks.

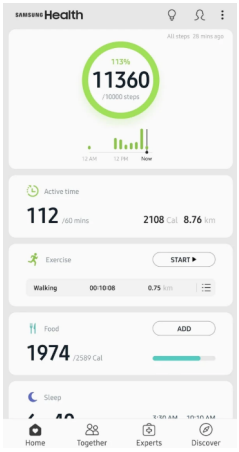

Fig. 1: Samsung Health [23], [24].

- *Just Eat* (Fig. 3), score = 4.4/5. After inserting home address, user can order food and drink from her favourite pubs in the nearby, specifying time of delivery and type of payment. Every pub page on Just Eat comes with other users' reviews, which is a really helpful way to choose among all of them.
- *Sanità km zero - Ricette* (Fig. 4), score = 4.0/5. This is an Italian experiment in Veneto Region. The user can get prescriptions directly inside the app and show them in pharmacies without taking the paper ones. In this way the application keeps records of all purchased drugs and each user can contact her doctor to renew prescriptions.

Fig. 2: Fitbit [25], [26].

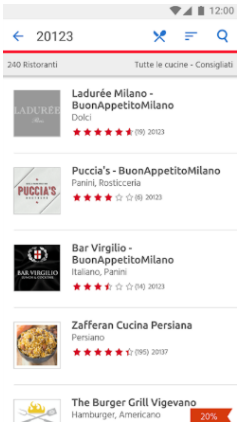

Fig. 3: Just Eat [27], [28].

Fig. 4: Sanità km zero - Ricette [29], [30].

B. Self-Adaptation

Generally speaking, self-adaptation stands for systems that are able to modify their behaviour and/or structure in response to their perception of the environment and the system itself, and their goals [5].

Although self-adaptive systems (SASs) were already the main topic in the field of *control theory*, the proper engineering and realization of self-adaptation in software still remains a formidable intellectual challenge [5].

In order to meet some quality attributes such as *fault-tolerance* and *replaceability*, these systems need to be able to self-reconfigure, self-heal, self-protect and self-optimize [31].

When dealing with the design of SASs, it is important not only to define the traditional requirements engineering phase for software development [32], but it is also needed a more specific preliminary study on what changes occur in

the environment the system lies in, what to adapt, when to adapt and how to adapt [33].

1. Design decision for SASs. There are five clusters of design decision for these systems, each of them answering a specific question. They are: observation, representation, control, identification and enabling adaptation, and their main features are [5]:

- *Observation*. It answers the question "what information about the external environment and about the system itself will need to be measured or estimated?". SASs usually observe the environment with the help of *sensors*. These measurements could be direct, indirect, extrapolated by early ones, etc. Finally, it is important to define what triggers observations, *i.e.* it is a continuous monitoring or it takes place at specific periods of time.
- *Representation*. It answers the question "what information is made available to the components of the self-adaptive system?". It is needed to define the way information is represented, for example explicit representations such as graph models, formulae, bounds, objective functions, etc., or implicit representations in the code.
- *Control*. It answers the question "what control loops are involved and how do those control loops interact?". The choice depend on the structure of the system. Control loops can be composed in series, parallel, multi-level (hierarchical), nested, or independent patterns. The decision about what triggers observations - seen in the *Observation* cluster - is closely related to this control cluster.
- *Identification*. It answers the question "what are the concrete sets of possible structures, behaviors, states, parameter values the system can undergo?". Technically speaking, the SAS is in one instantiation at every moment in time. So, the process of evolution consists of traversing the space of such instantiations.
- *Enabling Adaptation*. It answers the question "how the systems reacts to the events from the outside world?". It deals with the choice of which adaptation mechanisms the SAS employs, how they are triggered and how failure situations are handled. These mechanisms can be represented implicitly, explicitly, but also in an hybrid for which involves both of them. The decision about control loop (seen in the *Control* cluster) is closely related to this cluster.

2. New challenges in Software Engineering. It has been broadly accepted that softwares, which act inside real-world situations, must continuously evolve in order to avoid the risk of becoming outdated earlier than expected [34], [35].

Nowadays, software maintenance and evolution can be considered as one of the key fields of Software Engineering (SE). It separates the time before and the time after the delivery of a software, *i.e.* it divides its lifecycle in [36]:

1) development-time;
2) deployment-time;
3) run-time.

If any changes have to be made to the realised version of a software, they come as *patches* which replace or enhance the currently running version [37].

However, the key point here is that the common procedure consists of system maintenance performed *off-line* and it is guided by *humans* [5].

All these features are in contrast with SASs requirements, according to which a system operates in a highly dynamic world and must adjust its behavior automatically in response to changes in its environment, in its goals, or in the system itself.

In this new vision, humans should define how a system must react to changes, so that they could perform these adaptations autonomously at run-time [5].

An important new requirement introduced with SASs is the timing of verification and validation phases (V & V). They are traditionally performed at development-time, while in self-adaptation they must be considered continously during the whole process to build the software [5].

III. Proposed approach

The purpose of this section is to describe the implementation of *myHealth*, an Android App in the domain of the eHealth. Application's goal is to collect user's data generated by other apps - *e.g.* how many calories she ate, steps count, etc - to build a model upon it. Then, the app uses this model to give user suggestions to improve her personal lifestyle - *i.e.* it is self-adaptive.

The current section is structured as follows: the first part deals with the overall architecture, describing the single components and the way they communicate each other; the second and last part explains user's model and the logic to give suggestions.

A. Overall architecture

The architecture (Fig. 5) of the entire system involves different devices and platforms that interact with each other through *connectors* that allow the exchange of information between the different components.

Going more in-depth, the entire architecture consists of:

- myHealth application, which directly interacts with a central server to store data remotely in a persistent way on a MySQL[9] database (Fig. 6);
- third-party apps and/or devices that send data to our application through a publish/subscribe protocol (Fig. 7);
- the user model, which is built upon user's health parameters and allows the main app to adapt to changes in user's lifestyle and generate tips to lead an healthy life (Fig. 8);
- a crawler to retrieve the five most popular Google Play applications belonging to a certain category and suggest them to the user (Fig. 9).

[9]https://www.mysql.com/

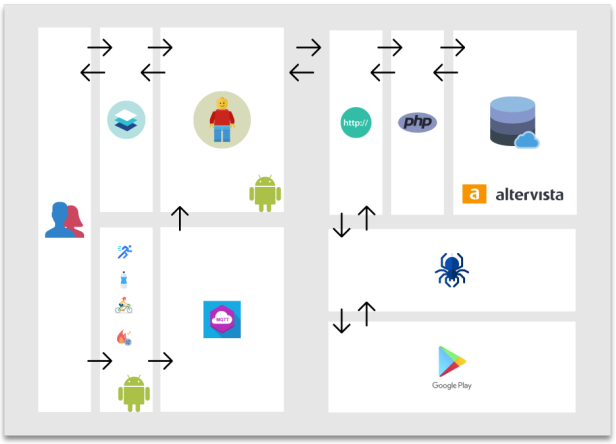

Fig. 5: Overall system architecture.

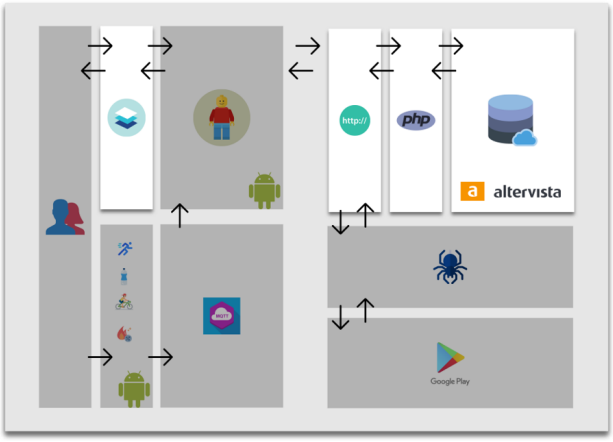

Fig. 6: myHealth app directly interacts with a central server to store data on a MySQL database.

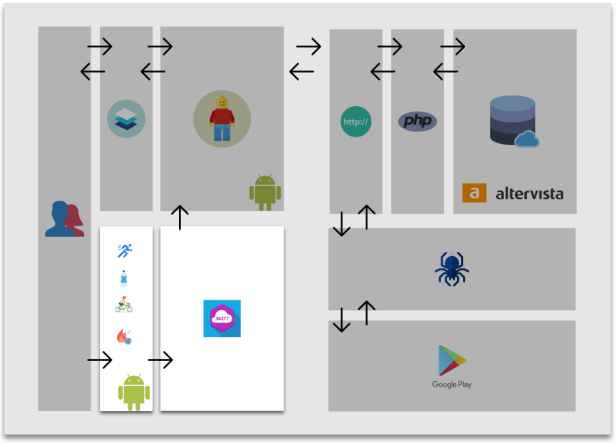

Fig. 7: Third-party apps send data to our application through a publish/subscribe protocol.

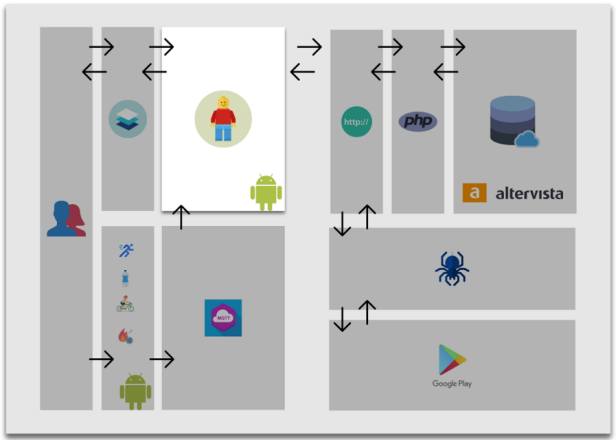

Fig. 8: User model is built upon user's health parameters and it is used to generate tips to lead an healthy life.

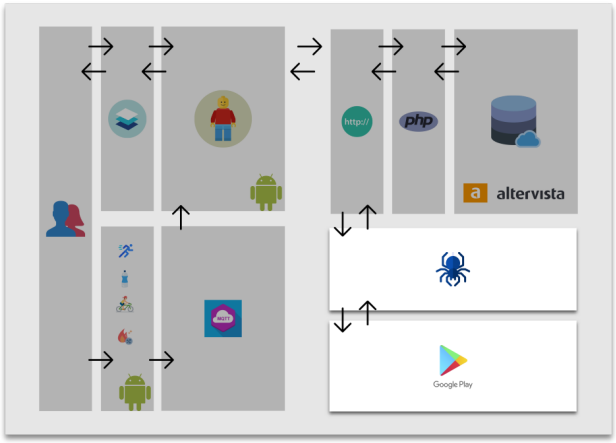

Fig. 9: A crawler is queried to retrieve the five most popular Google Play applications belonging to a certain category.

1. User data management: server and db. First of all, the user has the possibility to create an account, providing basic info - *e.g.* name, surname, birthdate, sex, height and of course username and password (Fig. 10). A first prototype of user model is calculated accordingly. Once the account is created, the user can login to the application (Fig. 11).

Through a very simple and user-friendly graphical interface one can add and remove third-party apps associated to the main application (Fig. 12). They will be in charge of sending user's health data to myHealth. It also shows the top five most downloaded apps on Google Play by category.

User's collected health parameters are shown in the dashboard, and user is suggested with tips about health according to the user model (Fig. 13).

Each time the application interacts with the server, an HTTP POST request is made, specifying the requested information. The server is hosted on a free Altervista[10] domain and interactions with MySQL database are performed by PHP scripts. Our project website is accessible at http://adaptapppoliba.altervista.org/index.html.

As regards parameters sent by the third-party apps, we used the publish/subscribe protocol MQTT that allows an

[10]https://it.altervista.org/

asynchronous communication. Every time data is sent by those apps, this event is notified to myHealth.

If the producers of a new eHealth app want to register their service to myHealth, they can do it. Just check our website under *Join us* section. Data received by our application is then processed, thus modifying the user model and making it consistent with the new data. Finally, the initial data is saved on the remote DB.

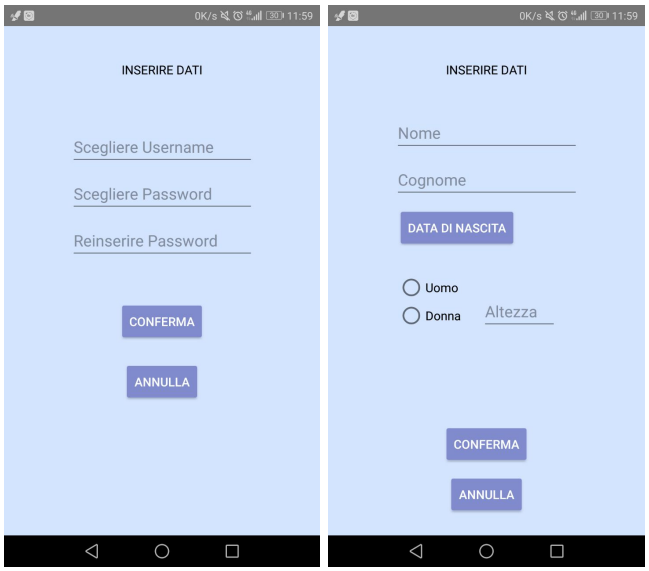

Fig. 10: User is asked to insert basic data to register.

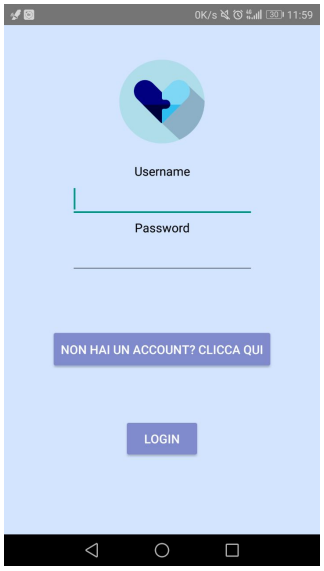

Fig. 11: Once the account is created, user can login.

2. Apps communication. At this point, something to link all the parts of the architecture seems to be missing.

To begin with, let us consider the following situation: Daniele, a 23-year-old guy, wants to order some food with his Android app *FoodApp*. He has already downloaded myHealth on his smartphone to take note of food he daily eats.

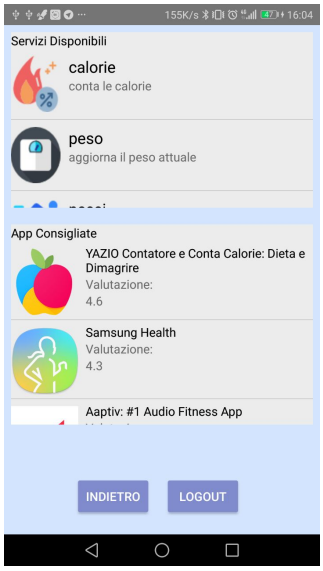

Fig. 12: User can associate third-party apps to myHealth. Other apps from Google Play are also suggested.

Fig. 13: User's collected health parameters are displayed inside a dashboard. Tips about health are also shown.

However, he is quite a lazy boy so he does not want to notify this purchase event to myHealth himself. He wishes these two apps were *connected* in some ways, so that myHealth is aware of everytime he buys food.

After about one hour, Daniele has just finished eating what he had previously ordered on *FoodApp*. He feels full because he had a lot of junk food, so he would like to know how his weight changed in the meanwhile to update it on myHealth.

He has got a smart scale which is connected to the Android *WeightApp* application. So he measures his weight, but again he wishes there was a kind of *link* between the scale and myHealth.

What is missing in the two situations is an *actor* able to

notify particular events - the food purchase and the weight measurement - to myHealth.

With this goal in mind, we thought about a *publish/-subscribe* protocol. Eventually, we went for *MQTT* for the following reasons:

- ✓ it is one of the most adopted solutions among state-of-art publish/subscribe protocols;
- ✓ it is well documented both for server side implementations and for client side ones (we needed them all);
- ✓ it is lightweight since it is often adopted in IoT systems.

1) MQTT protocol: Message Queuing Telemetry Transport (*MQTT*) [38] is a *machine-to-machine* connectivity protocol, designed as a lightweight *publish/subscribe* messaging protocol.

The publish/subscribe pattern is an alternative to *client/server* model. While in the latter client communicates with an endpoint directly, in the former there is a decoupling among those who send messages - *i.e. publishers* - and those who receive the messages - *i.e. subscribers* [39].

A third component called *broker* acts as intermediary for publishers and subscribers, so that they are never aware of each other. Publishers send messages to the broker, who is in charge of distributing those messages to the interested subscribers.

Broker has several options to choose subscribers to be notified for the messages. The most common one is based on *topics*.

Topics [40] are strings of characters and numbers divided by / which are unique for a certain broker server. Each string composing a topic represents a *topic level*. Examples of possible topics are: *calorie/daniele* or *bici/daniele*.

As a matter of facts, subscribers subscribe to the topics they are interested in, so that the broker is aware of the clients to notify every time it receives a new message from publishers on those specific topics.

To sum up, these are the most relevant features of publish/-subscribe model [39]:

- it decouples *spatially* publishers and subscribers, *i.e.* they only need to know the IP address and the port number of the broker in order to communicate;
- it decouples *by time*, which means that a broker can store messages received from publishers until the correspondent subscribers go online again;
- it is an *asyncronous* model, *i.e.* tasks on clients are not blocked while waiting for a message or publishing a message.

All those features are part of the MQTT protocol, which is an *application-layer* protocol of the TCP-IP stack.

Every client [41] - publishers and subscribers - in MQTT need to be identified with a unique *clientId*, which is stored by the broker. Its purpose is to let the broker keep a link between each subscriber and the topics those clients are subscribed to.

When a client wants to establish a connection with a broker for the first time, it sends a *CONNECT* message to it containing a lot of useful info about this connection [41].

Among them, it is remarkable to remember the *cleanSession* field. If it is set to *false*, it states that all subscriptions of that particular client need to be stored by the broker, as well as all the missed messages sent while it was offline, for istance [41].

2) Implementation: For the sake of this project work, we implemented the Android application myHealth as a subscriber client. Then we thought about some possible third-party apps - belonging to the eHealth domain - which might connect to our application.

In order to keep things simple, we named them this way (Fig. 14):

- *FoodApp*: user can order food choosing from a list and set the correspondent calories;
- *TransportApp*: user can keep track of the distance spent every time she uses bicycle;
- *WeigthApp*: user can keep track of her weight changes;
- *WaterApp*: user can keep track of drunk water;
- *StepCounter*: user can keep track of the steps counting every time she goes out for a walk.

According to MQTT logic, all those simple apps were thought as publishers on a certain topic. Topics structure can be summarized in this way:

- FoodApp ⇒ *calorie/username*;
- TransportApp ⇒ *bici/username*;
- WeightApp ⇒ *peso/username*;
- WaterApp ⇒ *acqua/username*;
- StepCounter ⇒ *passi/username*

where *username* stands for the username of the particular user logged in.

Choosing a unique *clientId* for a particular client was easy, too. It is the concatenation of the *app name* and *username* strings, which is trivially unique for a particular broker server.

The structure of published messages is a JSON element built in the following way:

```
{
    "app": "app name",
    "value": "sent value",
    "date": "date of sent value",
    "time": "time of sent value"
}
```

Now, let us focus on libraries and third-party services adopted to implement the MQTT protocol.

As regards clients - both publishers and subscribers - we used Paho Java MQTT Client[11] library.

On the other hand, we had mainly two options for the broker server: either a private one or a public one. First we started testing the overall architecture with the help of a private Mosquitto[12] broker instance installed on Ubuntu 16.04. Eventually, we decided to go public exploiting one of the possible public brokers on the Internet, and we went for HiveMQ MQTT broker [42].

[11] https://www.eclipse.org/paho/clients/android/
[12] https://mosquitto.org/

These are the steps a generic publisher needs to follow to publish a certain message to the broker [43]:

1) connect to the broker at tcp://broker.hivemq.com:1883 with clientId as described before and with cleanSession field set to *false*;
2) as soon as connection is established, publish message in JSON format on a certain topic which follows rules presented earlier in this section.

On the other hand, these are the steps a generic subscriber needs to follow to subscribe and receive constant updates on a certain topic [43]:

1) connect to the broker at tcp://broker.hivemq.com:1883 with clientId as described before and - really important this time - with cleanSession field set to *false*;
2) as soon as connection is established, subscribe to a certain topic which follows rules presented earlier in this section;
3) listen continuously to updates coming from broker - *e.g.* with a *callback* function [44]; remember that, as documentation states [43], client library does not support any reconnect logic yet and public broker - as the website says [42] - is not 100% reliable; for this reason, consider the possibility to implement a periodic reconnection to broker.

3. The crawler: queries to Google Play. Among all described features of myHealth, let us focus on the suggested tips about Google Play apps the user might find interesting. In order to accomplish this task, myHealth queries a *crawler* to retrieve the five most important Google Play applications from a specific category. The crawler runs locally on a NodeJS[13] server with the help of the package *google-play-scraper* [45].

We deeply thought about application categories to take into account, and eventually we decided to consider the following categories [46]:

- *Food and Drink*
- *Health and Fitness*
- *Medical*
- *Lifestyle*

The main application sends a HTTP POST request to the NodeJS server specifying the app category it wants to retreive. Then the server returns a JSON array containing five results from Google Play.

One possible example is reported here:

```
{
    "title": "Samsung Health",
    "url": "https://play.google.com/store/apps
        /details?id=com.sec.android.app.
        shealth",
    "score": 4.3,
    "icon": "https://lh3.googleusercontent.com
        /wh-dPpiZeTSz6hYgoBYIEQ_vwlXVD8
        BcszygaS4RMQa3NeZeo96ZD0x96TBb9x-Wo-Y=
        w340"
}
```

[13] https://nodejs.org/en/

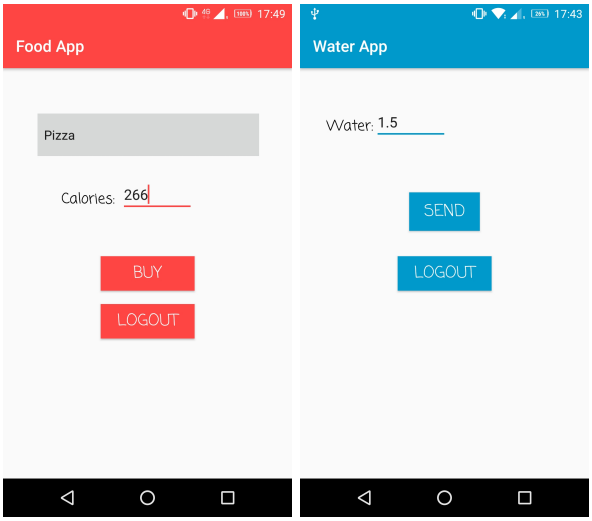

Fig. 14: Main activity of all dummy apps. The interface is quite simple since they were only used for test purposes.

B. User management

In this section we are going to see how we modelled the user and how we managed his behaviour. In the first part we will see the models that allowed us to define the user: his personal and compartmental characteristics. We used JSON files to store all this information on the app and they are unique for each user. In the second part we will explain our approach, using the information collected and memorized in the JSON files, for giving health suggest to the user and how the system evolves over time.

1. The user models. Every single user has three models associated with his account that have different purposes. These are:

- daily model;
- goal model;
- weekly model.

The daily model has the purpose to collect the data about the activity of the user coming from the app linked with the system. It has the form of a JSON like below.

```
{
    "id": "user username",
    "altezza": "user height",
    "eta": "user age",
    "sesso": "user sex",
    "peso": "user weight",
    "calorie": "calories taken during the day"
        ,
    "passi": "number of steps during the day",
    "bici": "kilometers traveled by bike
        during the day",
    "acqua": "liters of water taken during the
         day"
}
```

The goal model permits to define what is the goal standard for the user. In our case, the goal standard is to help the user to arrive to his normal weight and let him assume his minimum water requirement. This model has also the form of a JSON.

```
{
    "calorie": "user BMR",
    "acqua": "user minium water requirement",
    "pesoFormaSup": "superior limit of normal
        weight",
    "pesoFormaInf": "inferior limit of normal
        weight"
}
```

The parameter *calorie* has been calculated with the BMR, the basal metabolic rate. The BMR measures the minimum requirement of energy per day to guarantee the biological activities inside the body. The BMR has been calculated with The Mifflin St Jeor Equation [47]. This equation has been demonstrated has the highest accuracy respect to the others [48].

The parameter *acqua* represents the minimum requirement in liters of water recommended from the World Health Organization [49].

The parameters *pesoFormaSup* and *pesoFormaInf* refer to the range of normal weight based on the BMI index [50].

The weekly model is used to store the elaboration of the data collected in the day. All data present in this model are binary. It has two parameters:

- *cal*: this parameter represents the behaviour of the user about the energetic balance. If the user has to lose weight and has a negative energetic balance or if the user has to gain weight and has a positive energetic balance this value is set to 1, else it is set to 0;
- *water*: this parameter represents the behaviour of the user about the water requirement. If the user has taken more water than the minimum requirement is set to 1, else it is set to 0.

2. The tip system. The system has to give healthy tip to the user. The type of tips that the system gives to the user are two:

- daily tips;
- weekly tips.

1) Daily tips: The daily tips are recommendation about the user's daily activity, for example if the user must lose weight and he has taken more calories that the one who has spent during the day the system will notify him that he must do much movement.

We have three kind of daily tips: energy balance, weight control and water control.

For the energy balance we use the equation below:

$$EB = CalTaken - BMR - CalBicycle - CalStep \quad (1)$$

To calculate the values *CalBicycle* and *CalStep* we used the definition of Metabolic Equivalent of Task (MET) and used the following equations [51]:

$$CalBicycle = 4 * weight * time \quad (2)$$

$$CalStep = 3.6 * weight * time \quad (3)$$

In (2), coefficient 4 is the MET associated with bicycling at moderate velocity, weight is the user's weight and time is the time the user has done that activity (calculated on the average speed in city). In (3) coefficient *3.6* is the MET associated with walking, time is calculated respect to the human's average speed in walking.

For giving the tip in this topic we use (1) combined with the BMI [50]:

- if $EnergyBalance \geq 0$ and $BMI \geq 25.01 \Rightarrow$ the user needs more physical exsercise;
- if $EnergyBalance < 0$ and $BMI < 18.51 \Rightarrow$ the users needs to assume more calories;
- otherwise $\Rightarrow$ the user has a good energetic balance;

For the weight control the system gives two tips: the first one reminds the user about the difference of weight between the actual weight and the previous weight, the second gives the distance between and the normal weight of the user respect to the BMI.

The water control reminds the user about his water consumption, basically if the water taken during the day is more than the minimum recommendation present in the goal model.

2) Weekly tips: At the end of the day, the system will process the data collected in the daily model and it will compile the weekly model that will be sent to the server.

In the first 6 days the system collects only the data coming from the daily activity, at the seventh day the app will elaborate the weekly models coming from the server and the one coming from the actual day and it will compute the mean of the parameters present in the weekly model. If the mean of the parameter *cal* is greater than 0.5 this means that the user behaved well in the last 7 days. The same reasoning is done for the parameter *water*.

IV. Validation

The purpose of this section is to show the validation phase of this project work and the corresponding results. Testing the application is one of the most important step in software engineering, and its main goal is to check if outputs are coherent with the expected ones.

The section is structured as follows: the first part describes data and parameters used to perform the test; the second and last part shows the obtained results.

A. Used data and parameters

To test the correct functioning of the whole system and of each component, repeated tests were carried out for one week. The tests started on 12/24/2018 and ended on 31/12/2018 for a total of 7 days.

Let us pretend the user who tested the application is Luigi Bisceglia. He is 24 years old and his height is 170 cm (Tab. I).

NAME	Luigi
SURNAME	Bisceglia
AGE	24
HEIGHT	170 cm
SEX	Male

TABLE I: User info for the test.

The user registration steps are shown in Fig. 15. After that, Luigi can login to the application (Fig. 16).

The main view of the app consists of (Fig. 17):

- the connected third-party apps;
- a graph with the latest measurements for a specific parameter (you can choose which one to see);
- calculated daily tips based on data coming from connected apps.

By clicking on the + button on the bottom right, Luigi can access an activity (Fig. 18) that allows to select new services to add to myHealth. It also shows the five most downloaded Google Play apps in the eHealth domain.

Let us suppose Luigi has selected all the available services. They are linked to five dummy apps, whose purpose is to simulate real-world eHealth applications.

Their names are:

- *FoodApp*;
- *TransportApp*;
- *WeigthApp*;
- *WaterApp*;
- *StepCounter*.

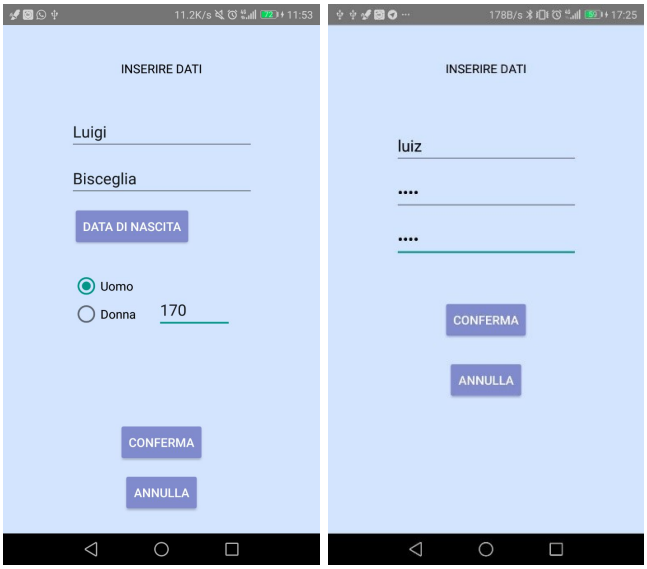

Fig. 15: Registration phase.

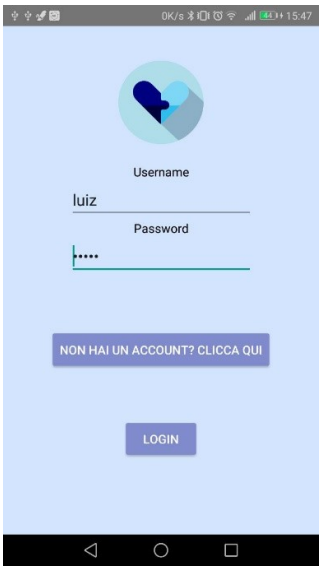

Fig. 16: After registration, the user can login to the application.

Now, test can finally begin. The only thing Luigi has to do is using these five apps as he always did.

Table II summarizes the test cases.

B. Results

No malfunctions or errors were found from the tests performed; the system seems to behave according to the requirements of the project.

What follows is a demonstration of one updating of *PASSI* and *ACQUA* occurred on day 4 in order to demonstrate that everything works fine.

Fig. 17: Screenshot of the dashboard.

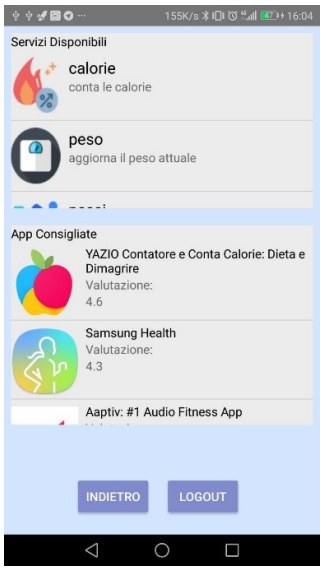

Fig. 18: User can add new services and see most downloaded Google Play apps in the eHealth domain.

DAY	PASSI	CALORIE [cal]	ACQUA [litres]	BICI [km]	PESO [kg]
1	4000	2000	2	0	80
2	4500	2500	1.5	0	80
3	5000	1800	1	1	81
4	5060	2000	2.5	0	81
5	6500	2300	2	1	81
6	6000	2300	1.3	0	82
7	8000	2100	1.7	0	82

TABLE II: Test cases for 7 days.

Before these two updatings, these were the initial values for *PASSI* and *ACQUA*:

$$PASSI = 4960$$

$$ACQUA = 1l$$

On day 4, Luigi decides to go out for a walk. After that, he records this event using *StepCounter* app on his smartphone (Fig. 19). It is worth remembering these are simple dummy apps, with the only purpose of testing. In a real-world scenario he probably would have used a smartwatch connected to his smartphone to take note of the steps counting.

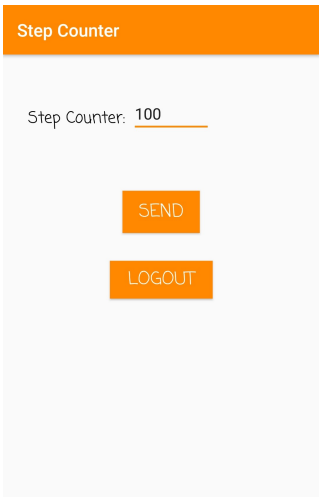

Fig. 19: Luigi sends the PASSI updating with *StepCounter* app.

Immediately, the updating is sent to myHealth thanks to MQTT protocol. Then, our application displays the new value on the dashboard (Fig. 20) and sends it to the server to update the user model.

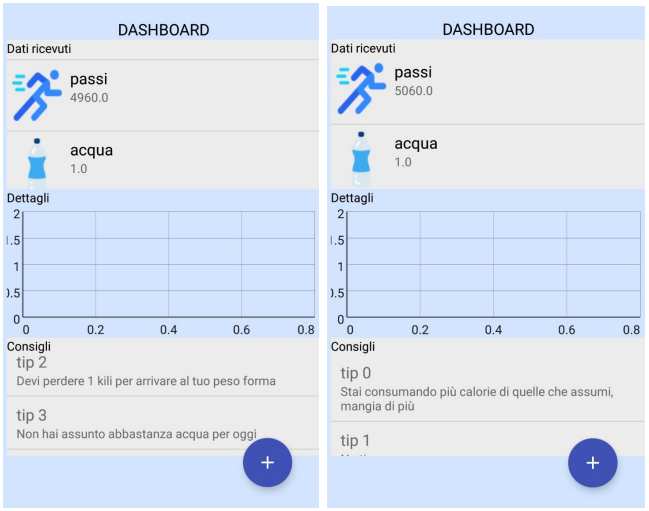

Fig. 20: Before and after the PASSI updating.

At the end of day 4, Luigi also decides to register the amount of water he has drunk during the whole day. Using *WaterApp* application, he records the value (Fig. 21). What happens here is exactly the same as before, and the results are shown in Fig. 22.

Considering again Fig. 22, it is important to point out the health tip shown at the bottom part of the screen, which states *Non hai assunto abbastanza acqua per oggi*. This is coherent

with our *gold standard* since we found out men should drink at least 2.9 litres of water per day [49], while Luigi only had a total of 2.5 litres (Fig. 22).

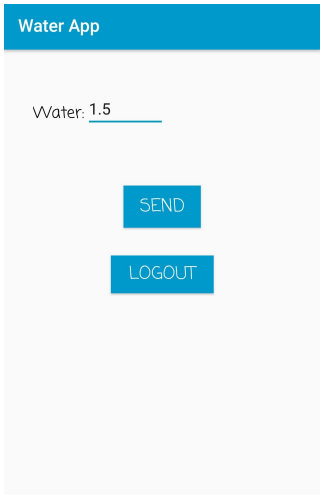

Fig. 21: Luigi sends the ACQUA updating with *WaterApp* app.

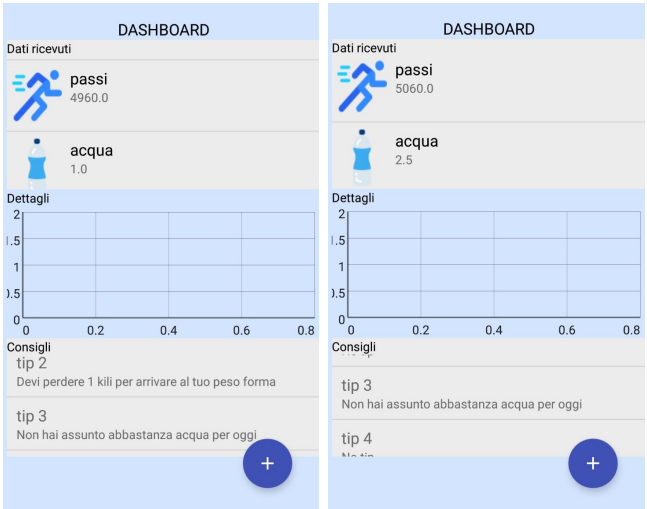

Fig. 22: Before and after the ACQUA updating.

V. Related work

The purpose of this section is to present some interesting works found in literature in the domain of eHealth and self-adaptation. Some of them were truly inspiring to design and implement our project.

The current section is structured as follows: the first part deals with examples taken from eHealth, while the second part is about examples taken from self-adaptation.

A. eHealth

To begin with, Google, in partner with pharmaceutical company Novartis, has developed smart contact lenses for people who suffer from diabetes. Smart contact lenses take the tears in a persons eye and measure the glucose levels [52].

A wearable ECG monitoring and alerting system called *iHeart* alerts the patient if it detects an abnormal hearth behaviour [53].

It is quite important to keep a costant eye on blood pressure, since when it is high - *i.e.* hypertension - it can cause serious problems like heart attacks and strokes. In [54] a noninvasive approach to monitor these parameters is proposed.

Asthma and other pulmonary diseases can be continously monitored by a wearable stethoscope described in [55].

B. Self-Adaptation

The work presented in [56] was dramatically inspirational for our project, especially for the design of the user model. It is an interesting approach to complex adaptive mobile applications modelling and implementation, able to dynamically change according to changed behavioural properties, state and/or context variables and user's preference. To this purpose, the authors designed a metamodel made up of an action repository (AR). The AR is structured as a set of logical propositions - pre conditions and post conditions of an action. Beyond the AR, their metamodel is composed by a set of sensor, effectors, a user model and an algorithm that finds a set of possible actions that can be executed.

In [57], it was proposed a mobile middleware system on top of Android which eases the development and maintenance of adaptive, context-aware mobile applications, implementing a Instant Messenger app with an adaptive multimodal user interface.

The mobile application works this way (Fig. 23): it is a Instant Messenger app which adapts its interface accordingly to user's movements. For istance, when the user is moving fast, is switches from text-only to text-to-speech mode.

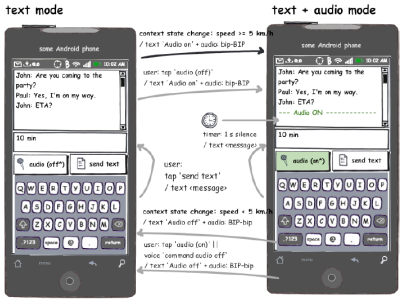

Fig. 23: The mobile application switches from text-only to text-to-speech mode [57].

VI. Conclusion and future works

In this report we proposed myHealth, an Android Self-Adaptive application in the domain of eHealth.

Starting from what had already been done and fully explained in literature, we took ispiration from that but decided to implement something different and innovative in some ways.

Our intention was not to create yet another eHealth Android application - there are plenty of them on Google Play and they perform quite well.

We actually wanted to build a *collector* for user's health parameters coming from those already existing apps - *e.g.* calories she eats, water she drinks, steps counting, spent distance on a bike and measured weight - and use them to build a user model and give suggestions about health.

As a matter of facts, our application needed to actually live inside user's environment, like a personal trainer that silently looks at your daily habits to keep you healthy.

In order to accomplish these tasks, we adopted MQTT protocol to let the third-party apps communicate with the main app, and we made myHealth self-adaptive.

What is more, the application is really simple to use - the user only has to connect the already existing applications on the smarphone to myHealth - with a user-friendly graphical interface.

The tests carried out during the whole development of the application proved that the overall system respects the defined requirements, behaving in line with the planned ones.

Despite all the positive features of myHealth, there is still something to work on in the future.

First of all, we need to improve the interfaces to expose them to external applications, for example by creating APIs for those apps that want to become part of the myHealth ecosystem and advertise to existing realities in the health domain the possibility of integrating these APIs.

As far as the server is concerned, we adopted a free but also limited solution only for testing purposes, but we should consider a more robust hosting platform to ensure security and scalability to our system.

In addition to this, the display of suggested real-world applications should be improved as well. In fact, now the application continously loops on the Google Play categories we decided to retrieve from the crawler. Due to a lack of time, we were not able to implement a logic which could suit the user.

One possible solution might be to query the crawler specifying an application category the user seems to be interested in.

As regards MQTT protocol, we used a public but not so much reliable broker only for test purposes. In a real-world scenario it would be better to adopt a private broker, *i.e.* a broker only used for this application but still accessible on the Internet.

Finally, let us consider the self-adaptive and user model features once again.

The user management in this project is simple since its main purpose was not to create the best user management, but to create a self-adaptive system that could suit user's behaviour.

This simplicity comes with advantages and disadvantages of course. The principal advantage of this kind of management is its easy formulation and implementation because it works with just few parameters. On the other hand, the main disadvantages are two and can be summarized in one sentence: *simplicity brings inaccuracy*.

To evaluate the energy balance, we used the BMR. The BMR is a relatively simple parameter to calculate and the Mifflin St Jeor equation is the most accurate in the scientific literature, but the BMR is not the best way to calculate the energy consumption of a person. This is because - as definition states - it is the energy consumption needed for metabolic functions like breathing, blood circulation, digestion, nervous system activities and it does not take into account energy consumption for daily activities; as a matter of facts, all the energetic balance are underestimated with respect to the real values for a person's energy consumption.

The small quantity of information received by the app is the second disadvantage. The calculus of energetic consumption for activities like bicycling or walking is done with the MET parameter associated with those activities and requires the time spent doing an activity; however, our system does not consider this time, so we have to estimate it with the average speed for that kind of activity, but we cannot say that this measure is the most accurate one.

In the future, we should use a more precise model that considers all the parameters that are missing, like the parameters that influence the total metabolism; what is more, having a bigger number of parameters coming from the third-party apps would surely increase the accuracy of the system.

REFERENCES

[1] Cisco Visual Networking Index (VNI) Global Mobile Data Traffic Forecast (2015 to 2020), [Online]. Available: https://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html?CAMPAIGN=MobileVNI2015&COUNTRY_SITE=us&POSITION=PR&REFERRING_SITE=SocialMedia&CREATIVE=PR+to+VNI+page.

[2] AppBrain. Android and Google Play statistics, [Online]. Available: https://www.appbrain.com/stats.

[3] World Health Organization Official website. Physical Inactivity: A Global Public Health Problem, [Online]. Available: http://www.who.int/dietphysicalactivity/factsheet_inactivity/en/.

[4] J. P. Higgins, "Smartphone applications for patients' health and fitness," *The American Journal of Medicine*, pp. 11–19, 2016.

[5] R. de Lemos, G. Giese, H. Müller *et al.*, "Software engineering for self-adaptive systems: a second research roadmap," *Software engineering for self-adaptive systems II*, pp. 1–32, 2013.

[6] C. Pagliari, D. Sloan, P. Gregor, F. Sullivan, D. Detmer, J. P. Kahan, W. Oortwijn, and S. MacGillivray, "What is ehealth (4): A scoping exercise to map the field," *J Med Internet Res*, vol. 7, no. 1, p. e9, Mar 2005.

[7] G. Eysenbach, "What is e-health?" *J Med Internet Res*, vol. 3, no. 2, p. e20, Jun 2001.

[8] The Fifty-eighth World Health Assembly. Resolutions and Decisions., [Online]. Available: http://apps.who.int/iris/bitstream/handle/10665/20378/WHA58_28-en.pdf;jsessionid=C8637296A3BF59F18F6D3E2CA67528FF?sequence=1.

[9] R. S. H. Istepanian, E. Jovanov, and Y. T. Zhang, "Guest editorial introduction to the special section on m-health: Beyond seamless mobility and global wireless health-care connectivity," *IEEE Transactions on Information Technology in Biomedicine*, vol. 8, no. 4, pp. 405–414, Dec 2004.

[10] S. H. Almotiri, M. A. Khan, and M. A. Alghamdi, "Mobile health (m-health) system in the context of iot," *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pp. 39–42, Aug 2016.

[11] H. Jee, "Review of researches on smartphone applications for physical activity promotion in healthy adults," *Journal of Exercise Rehabilitation*, vol. 13, no. 1, pp. 3–11, 2017.

[12] J. Fanning, S. P. Mullen, and E. McAuley, "Increasing physical activity with mobile devices: A meta-analysis," *J Med Internet Res*, vol. 14, no. 6, p. e161, Nov 2012. [Online]. Available: http://www.jmir.org/2012/6/e161/

[13] L. G. Glynn, P. S. Hayes, M. Casey, F. Glynn, A. Alvarez-Iglesias, J. Newell, G. ÓLaighin, D. Heaney, M. O'Donnell, and A. W. Murphy, "Effectiveness of a smartphone application to promote physical activity in primary care: the smart move randomised controlled trial," *British Journal of General Practice*, vol. 64, no. 624, pp. e384–e391, 2014. [Online]. Available: https://bjgp.org/content/64/624/e384

[14] M. C. Carter, V. J. Burley, C. Nykjaer, and J. E. Cade, "Adherence to a smartphone application for weight loss compared to website and paper diary: Pilot randomized controlled trial," *J Med Internet Res*, vol. 15, no. 4, p. e32, Apr 2013. [Online]. Available: http://www.jmir.org/2013/4/e32/

[15] M. N. K. Boulos and S. P. Yang, "Exergames for health and fitness: the roles of gps and geosocial apps," *International Journal of Health Geographics*, vol. 12, no. 1, p. 18, Apr 2013. [Online]. Available: https://doi.org/10.1186/1476-072X-12-18

[16] E. M. Wohlers, J. R. Sirard, C. M. Barden, and J. K. Moon, "Smart phones are useful for food intake and physical activity surveys," pp. 5183–5186, Sep. 2009.

[17] H. Ashrafian, T. Toma, L. Harling, K. Kerr, T. Athanasiou, and A. Darzi, "Social networking strategies that aim to reduce obesity have achieved significant although modest results," *Health Affairs*, vol. 33, no. 9, pp. 1641–1647, 2014, pMID: 25201670. [Online]. Available: https://doi.org/10.1377/hlthaff.2014.0370

[18] J. I. Recio-Rodríguez, C. Martín-Cantera, N. González-Viejo, A. Gómez-Arranz, M. S. Arietaleanizbeascoa, Y. Schmolling-Guinovart, J. A. Maderuelo-Fernandez, D. Pérez-Arechaederra, E. Rodriguez-Sanchez, M. A. Gómez-Marcos, L. García-Ortiz, and on behalf the EVIDENT Group, "Effectiveness of a smartphone application for improving healthy lifestyles, a randomized clinical trial (evident ii): study protocol," *BMC Public Health*, vol. 14, no. 1, p. 254, Mar 2014. [Online]. Available: https://doi.org/10.1186/1471-2458-14-254

[19] Y. Zheng, X. Ding, C. C. Y. Poon, B. P. L. Lo, H. Zhang, X. Zhou, G. Yang, N. Zhao, and Y. Zhang, "Unobtrusive sensing and wearable devices for health informatics," *IEEE Transactions on Biomedical Engineering*, vol. 61, no. 5, pp. 1538–1554, May 2014.

[20] A. Middelweerd, J. S. Mollee, C. N. van der Wal, J. Brug, and S. J. te Velde, "Apps to promote physical activity among adults: a review and content analysis," *International Journal of Behavioral Nutrition and Physical Activity*, vol. 11, no. 1, p. 97, Jul 2014. [Online]. Available: https://doi.org/10.1186/s12966-014-0097-9

[21] What to look for in a fitness app, available online at: http://health.usnews.com/health-news/health-wellness/articles/2014/07/18/what-tolook-for-in-a-fitness-app.

[22] AppBrain Official website. Android Statistics. Top Categories, [Online]. Available: https://www.appbrain.com/stats/android-market-app-categories/.

[23] Google Play Store Official website. Health and Fitness. Samsung Health, [Online]. Available: https://play.google.com/store/apps/details?id=com.sec.android.app.shealth&hl=it/.

[24] Samsung Official website. Samsung Health Smartphone App, [Online]. Available: https://www.samsung.com/global/galaxy/apps/samsung-health/.

[25] Google Play Store Official website. Health and Fitness. Fitbit, [Online]. Available: https://play.google.com/store/apps/details?id=com.fitbit.FitbitMobile/.

[26] Fitbit Official website, [Online]. Available: https://www.fitbit.com/it/home/.

[27] Just Eat Official website, [Online]. Available: https://www.justeat.it/.

[28] Google Play Store Official website. Food and Drink. Just Eat, [Online]. Available: https://play.google.com/store/apps/details?id=com.justeat.app.it&hl=it/.

[29] Google Play Store Official website. Medical. Sanità km zero - Ricette, [Online]. Available: https://play.google.com/store/apps/details?id=it.regioneveneto.sanitakmzero&hl=it.

[30] Sanità km zero - Ricette Official website, [Online]. Available: https://www.sanitakmzero.it/dev/app-sanita-km-zero-ricette/.

[31] A. G. Ganek and T. A. Corbi, "The dawning of the autonomic computing era," *IBM Syst. J.*, vol. 42, no. 1, pp. 5–18, Jan. 2003. [Online]. Available: http://dx.doi.org/10.1147/sj.421.0005

[32] B. Nuseibeh and S. Easterbrook, "Requirements engineering: A roadmap," pp. 35–46, 2000. [Online]. Available: http://doi.acm.org/10.1145/336512.336523

[33] Z. Yang, Z. Li, Z. Jin, and Y. Chen, "A systematic literature review of requirements modeling and analysis for self-adaptive systems," pp. 55–71, 2014. [Online]. Available: https://doi.org/10.1007/978-3-319-05843-6_5

[34] M. M. Lehman, "Software's future: managing evolution," *IEEE Software*, vol. 15, no. 1, pp. 40–44, Jan 1998.

[35] M. M. Lehman and L. A. Belady, Eds., *Program Evolution: Processes of Software Change*. San Diego, CA, USA: Academic Press Professional, Inc., 1985.

[36] T. Mens, *Introduction and Roadmap: History and Challenges of Software Evolution*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. [Online]. Available: https://doi.org/10.1007/978-3-540-76440-3_1

[37] B. A. Kitchenham, G. H. Travassos, A. von Mayrhauser, F. Niessink, N. F. Schneidewind, J. Singer, S. Takada, R. Vehvilainen, and H. Yang, "Towards an ontology of software maintenance," *Journal of Software Maintenance: Research and Practice*, vol. 11, no. 6, pp. 365–389, 1999. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291096-908X%28199911/12%2911%3A6%3C365%3A%3AAID-SMR200%3E3.0.CO%3B2-W

[38] MQTT Official website, [Online]. Available: http://mqtt.org/.

[39] HiveMQ Official website. MQTT Essentials Part 2: Publish and Subscribe, [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/.

[40] HiveMQ Official website. MQTT Essentials Part 5: MQTT Topics and Best Practices, [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/.

[41] HiveMQ Official website. MQTT Essentials Part 3: Client, Broker and Connection Establishment, [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/.

[42] HiveMQ Public Broker Dashboard, [Online]. Available: http://www.mqtt-dashboard.com/.

[43] MQTT Client Library Encyclopedia - Paho Android Service, [Online]. Available: https://www.hivemq.com/blog/mqtt-client-library-enyclopedia-paho-android-service/.

[44] Android MQTT client, [Online]. Available: http://androidkt.com/android-mqtt/.

[45] Google-Play-Scraper npm package, [Online]. Available: https://www.npmjs.com/package/google-play-scraper/.

[46] Google-Play-Scraper Categories, [Online]. Available: https://github.com/facundoolano/google-play-scraper/blob/dev/lib/constants.js#L58.

[47] M. D. Mifflin, S. T. St Jeor, L. A. Hill, B. J. Scott, S. A. Daugherty, and Y. O. Koh, "A new predictive equation for resting energy expenditure in healthy individuals," *The American Journal of Clinical Nutrition*, vol. 51, no. 2, pp. 241–247, 1990. [Online]. Available: http://dx.doi.org/10.1093/ajcn/51.2.241

[48] D. Frankenfield *et al.*, "Comparison of predictive equations for resting metabolic rate in healthy nonobese and obese adults: A systematic review," *Journal of the Academy of Nutrition and Dietetics*, vol. 105, pp. 775–789.

[49] G. Howard and J. Bartram, Domestic Water Quantity, Service Level and Health, [Online]. Available: https://www.who.int/water_sanitation_health/diseases/WSH03.02.pdf.

[50] Wikipedia. Indice di massa corporea, [Online]. Available: https://it.wikipedia.org/wiki/Indice_di_massa_corporea.

[51] Wikipedia. Metabolic Equivalent, [Online]. Available: https://en.wikipedia.org/wiki/Metabolic_equivalent.

[52] Top 15 Wearables in Healthcare, [Online]. Available: https://www.wearable-technologies.com/2015/04/wearables-in-healthcare/.

[53] H. Watanabe, M. Kawarasaki, A. Sato, and K. Yoshida, "Wearable ecg monitoring and alerting system associated with smartphone: Iheart," *Int. J. E-Health Med. Commun.*, vol. 4, no. 4, pp. 1–15, Oct. 2013. [Online]. Available: http://dx.doi.org/10.4018/ijehmc.2013100101

[54] C. C. Y. Poon, Y. M. Wong, and Y. Zhang, "M-health: The development of cuff-less and wearable blood pressure meters for use in body sensor networks," pp. 1–2, July 2006.

[55] K. J. Cho and H. H. Asada, "Wireless, battery-less stethoscope for wearable health monitoring," pp. 187–188, April 2002.

[56] T. D. Noia, E. D. Sciascio, F. M. Donini, M. Mongiello, and F. Nocera, "Formal model for user-centred adaptive mobile devices," *IET Software*, vol. 11, no. 4, pp. 156–164, 2017.
[57] L. David, M. Endler, S. D. J. Barbosa, and J. V. Filho, "Middleware support for context-aware mobile applications with adaptive multimodal user interfaces," pp. 106–111, July 2011.