POLITECNICO
MILANO 1863

TrackMe
Software Engineering 2 Project
DD Document

Stefano Martina, Alessandro Nichelini, Francesco Peressini

A.Y. 2018/2019
Version 1.0.0

December 7, 2018

Contents

1 Introduction

1.1 Purpose

The purpose of this Design Document is to give a functional description of the Data4Help application. The focus is on how the whole system is implemented, with particular attention to the components and its architecture.
While in the RASD document the description is presented at a high level, in the DD all the relevant components and their interfaces are explained in details, accompanied with the related diagrams. Algorithms of the most important features of the application and Implementation, Integration and Test Plan are also presented in this document.

1.2 Scope

The main scope of the Data4Help project is to provide a platform where the user can consult his/her health and location data and share these data with third-parties so that the latter can increase their business value.
The application also offers two additional services: AutomatedSOS, a constant vital parameters monitoring service, thought for elderly people, to detect anomalies and, in case of emergency, call an ambulance directly to the location of the costumer, and Track4Run, a service mainly thought for runners to create and join running events and live tracking on a map participants of the runs.
Both AutomatedSOS and Track4Run exploit the features offered by Data4Help.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

1.3.2 Acronyms

- API: Application Programming Interface;
- BPM: Beats Per Minutes;
- DD: Design Document;
- RASD: Requirement Analysis Specification Document;
- JSON: JavaScript Object Notation.

1.3.3 Abbreviations

- $[Gn]$: n-th goal
- $[Dn]$: n-th domain assumption
- $[Rn]$: n-th functional requirement

1.4 Revision history

- 1.0.0 Initial version (10/12/2018)

1.5 Reference Documents

- RASD document previously delivered

1.6 Document Structure

2 Architectural design

2.1 Overview: High-level

The system is going to be implemented with a three tier architecture. Tiers are as briefly described by the following schemas.

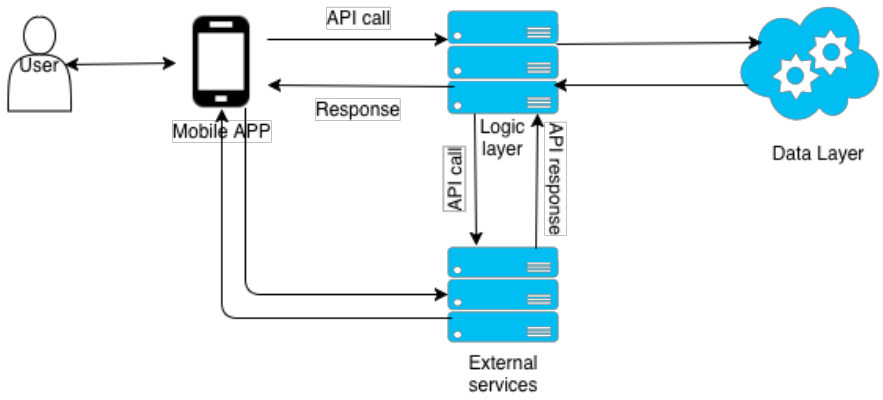

Figure 1: High level view of the system's architecture

The decision of this kind of architecture has been taken in order to build the system in the most modular possibile way. Here are described layer organisation:

- Presentation layer:
  - *Mobile clients*: users will be given with a iOS application which will be a view of the entire system.
  - *Third parties*: third parties will be given with a light web interfaces to register/manage API access and they will be authorised to communicate with the system.
- *Logic layer*: logic layer will implement all the logic of the entire system and will handle communications between clients' app and the data layer.
- *Data layer*: data layer will be implemented in third party's cloud system and will keep persistent users' data.

The idea is to keep as separate as possibile the logic layer from the data layer in order to let the system grow in a modular fashion and let us change cloud data provider as the system's dimension grow with the minimum effort.

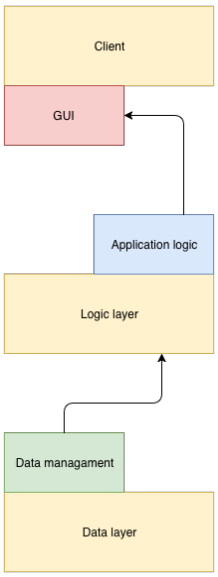

Figure 2: Distribution of application's function among the tier

2.2 Component view

We now provide a high level view of system's components. The whole system can be seen as two main client component *WebInterface* e *Mobile Application* that consumes services offered by a set of logic components: *HealthSharing manager*, *SOS manager*, *RunEvent Manager*, *Access Manager*. Each of them, will be interfaced with the *DatabaseLink Manager* which provides transparent access to the tier devoted to persistency.
We are going to provide further details of each subsystem.

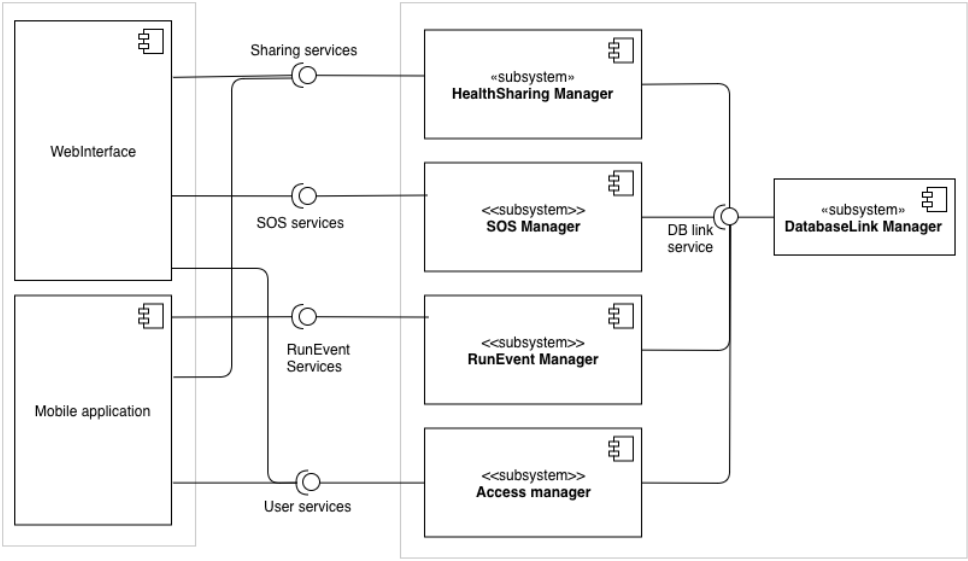

Figure 3: High level overview of system components.

2.2.1 Component view of HealthSharing Manager

HealthSharing manager is composed of two main submodule:

- Access Policy Manager Module.
- Data Manager Module.
- Data elaboration Module.

The *Access Policy Manager Module* works as a manager of all the policies associated with data sharing. It provides the list of active sharing to users and it let them manage active policies such as accepting new sharing request, or change/delete active policies. Moreover this component is also used by third parties that have to be able to subscribe to users data.
The *Data Manager Module* has to handle all the operations of retrieving/storing data between users' app and databases. It also has to guarente data consistency in the whole system.
The *Data Elaboration Module* has to handle all the operations devoted to make data anonymous and to handle request of aggregated data from third parties.

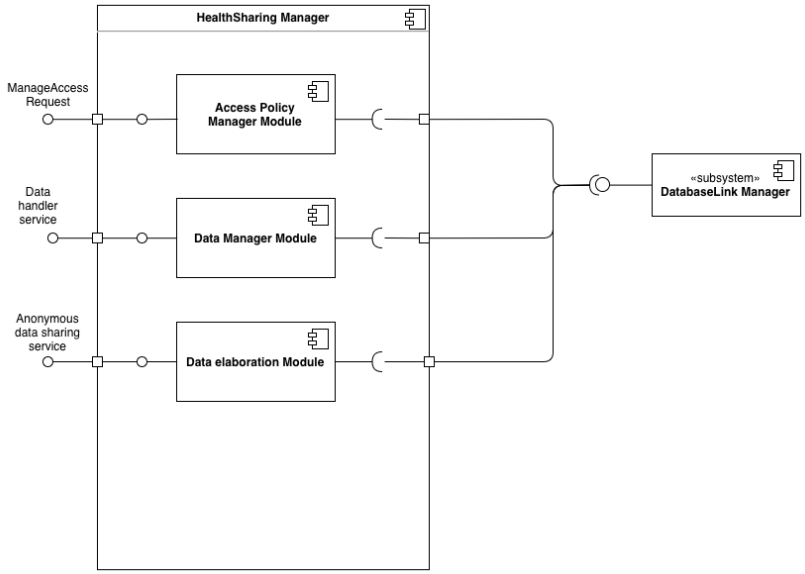

Figure 4: Specific description for HealthSharing Manager component.

2.2.2 Component view of SOS Manager

The *Anomaly Detection Module* has to live read and control heartbeat data from users that have activated SOS functionality in their mobile application.

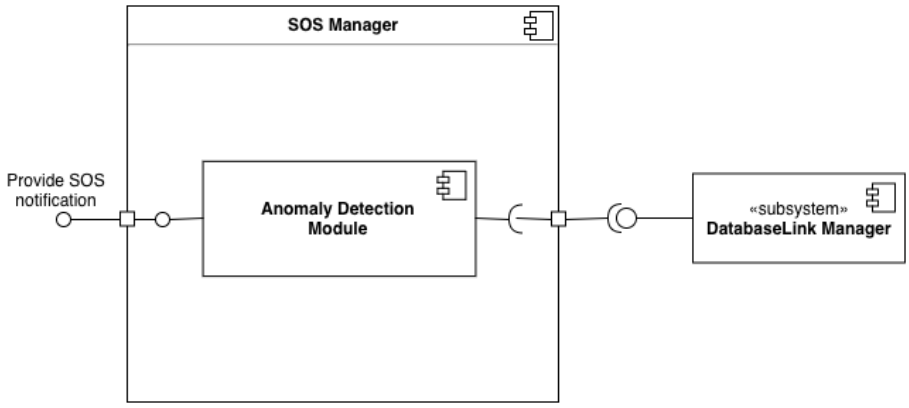

Figure 5: Specific description for HealthSharing Manager component.

2.2.3 Component view of RunEventManager

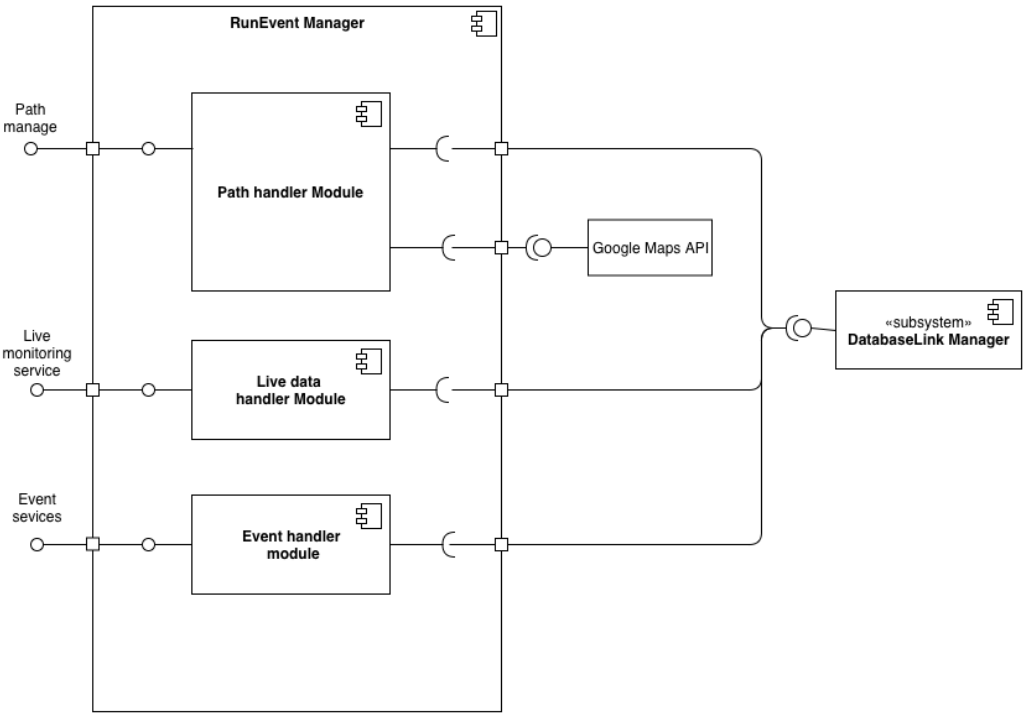

Figure 6: Specific description for RunEvent manager.

RunEvent manager is composed of three main submodule:

- PathHandler module.
- Live data handler module.
- Event handler module.

The *PathHandler module* has to offer services for creation and paths managing to users. In order to to that, it has to communicate with external Google Maps API.

The *Live data handler module* has to handle incoming live data from users and prepare and aggregated versione for spectators. Moreover it has to make them persistent for future access.

The *Event handler module* has to handle events creation, users' joining operations and spectators.

2.2.4 Component view of access manager

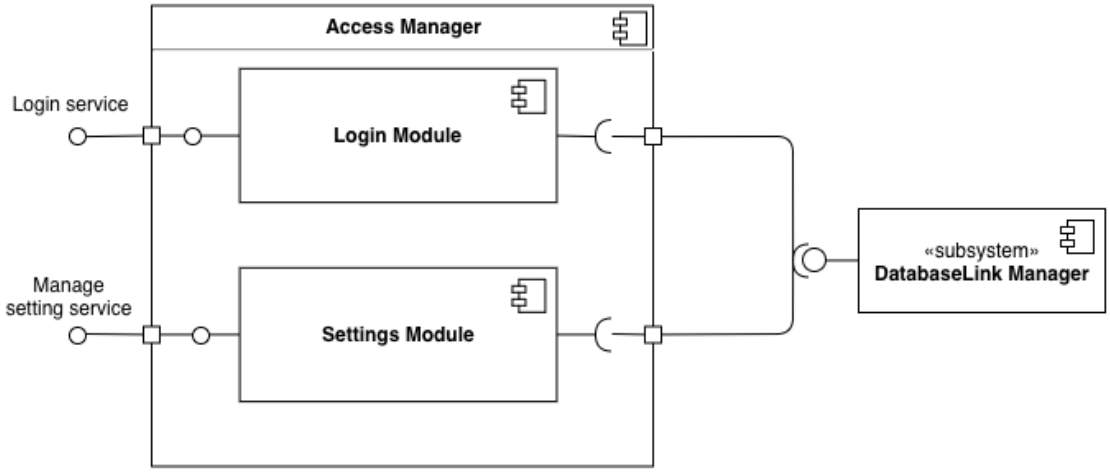

Figure 7: Specific description for Access Manager component.

Access Manager is composed of two main submodule:

- Login Module
- Setting Module

The *Login Module* offers all the function devoted to login for both users and third parties.
The *Setting Module* offers services devoted to let users to change and manage their preferences.

2.3 Deployment view

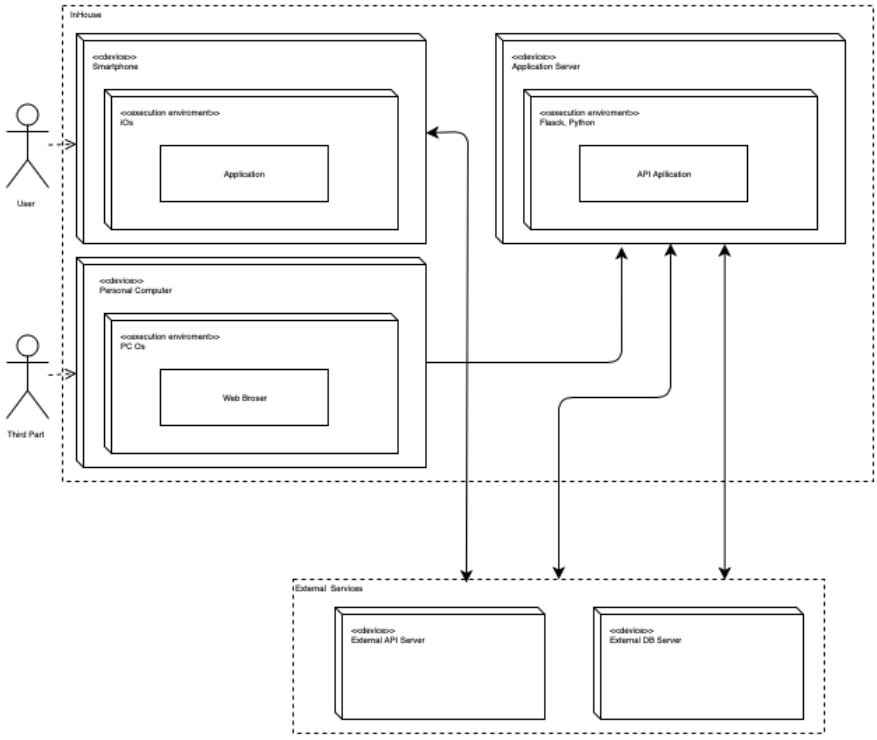

Figure 8: Deployment Dyagram

The Deployment diagram emphasises the nodes on which the platform runs: **InHouse** box represent the components developed inside the company. The components in the **External box** represent the on demand software.

- **Smartphone** : the application deployed on the iOs Smartphone used by the user. Users are able to retrieve data from the application server and, also, from the external server API directly (e.g. maps by Google Maps)
- **Web Browser API Interface** : A web page for Third Parties registration. It provides a secret to do a request to the API Server.
- **Application Server** : the main logic core of the application. Its the only one access point to the DB but also it communicates with the external server API
- **External Service** : Both the external services are represented as a black-box because we dont know how they are implemented.

2.4 Runtime view

2.5 Component interfaces

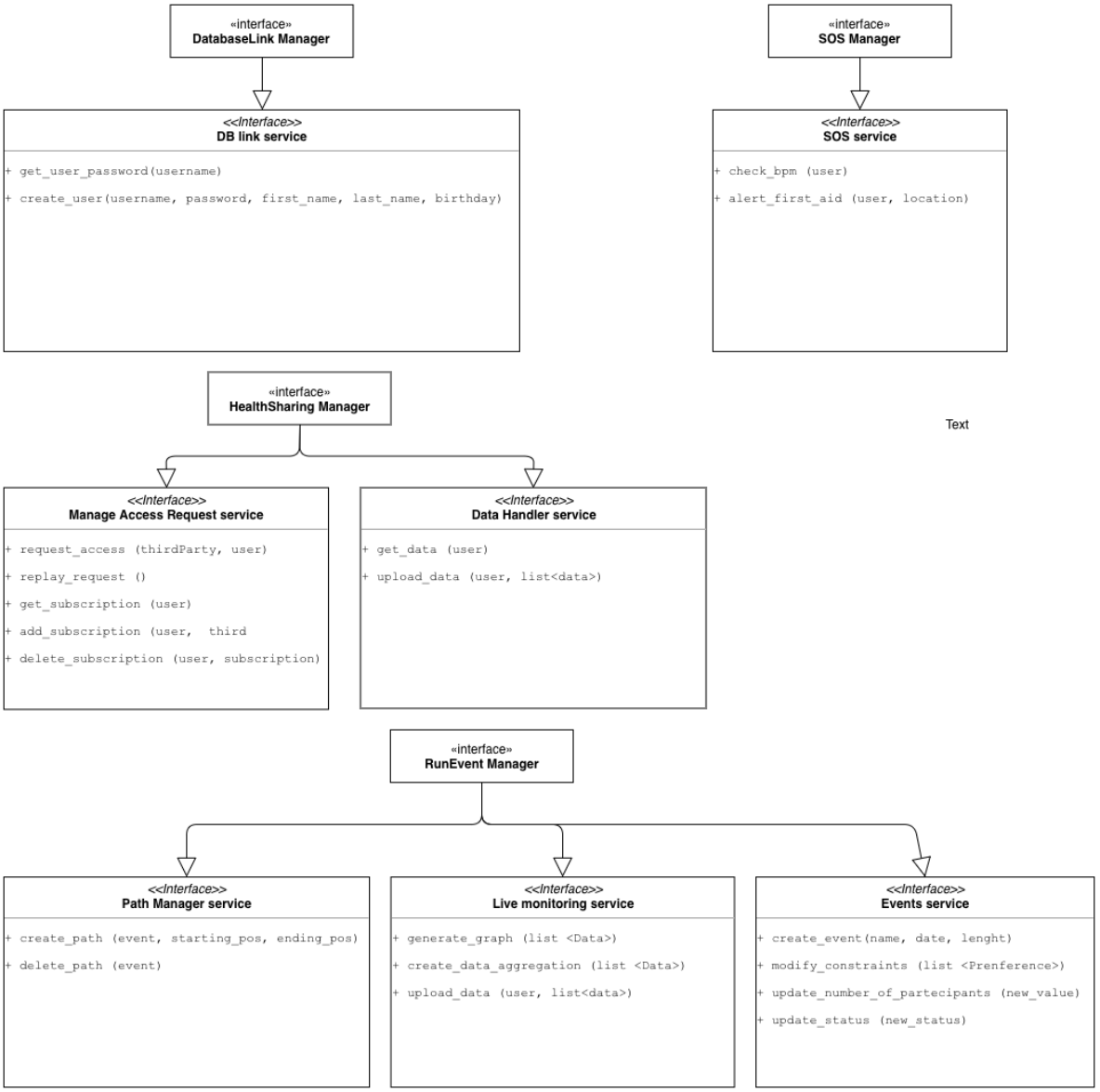

Figure 9: Components' interfaces descriptions diagram

2.5.1 API structure

The whole TrackMe infrastructure will be connected trough a RESTful API system. JSON will be used as serialisation data format.
All the api system will be implemented referring to a single endpoint www.data4help.cloud that TrackMe will buy for the implementation of this project.
Users' applications and third parties will refer to different subdomain:

- ***www.data4help.cloud/api/users***: it will be the specific endpoint for the application that serves users.
- ***www.data4help.cloud/api/thirdparties***: it will be the specific endpoint for thirdparties.

Here a complete description of each supported methods follows:

- method 1:
- method 2:

2.6 Selected architectural styles and patterns

Data4Help system is based on a tree tier architecture:

2.6.1 Overall Description

- **Presentation Tier** has to present the data to the user in such a way that they are meaningful for the users perspective. It displays information to the user obtained communicating with the other two tiers. For example, if a user performs a tap on the button create a new run event the communication tier send a query to the Logic Tier that answers with a message requiring for information to create an Event and so on. To be quick to react in case of Anomaly detection, instead of putting the logic of continuous checking of the vital parameters into the Logic Tiers (as the entire logic of the application) the Presentation Tier (the application) contains also this component. Doing this we will be able to speedy react and call the ambulance in the shortest way possible. We are also able to react in narrow situation as absence of field coperture or not working internet network.

- **Logic Tier** has the entire logic needed for the application (excluding the Anomaly detection component as described above). It receives the requests from the Presentation Tier and, if needed, queries the Data tier before then sending the reply.

- **Data Tier** is composed of a DBMS that provide an interface to the DataBase externally located. It contains all the data in a Relational Schema (ER) (view fig XXX)

  The separation between the different layers allow us to upgrade or replace a single tier without affecting the others

2.6.2 Design Patterns

- **Model View Controller (MVC)**
  MVC in a nutshell
  The MVC is a pattern that describe how to separate the components of the project in such a way to maintain high level of separation.
  - **Model** is the section that contains only the data of the application
  - **View** is the section that contains the component needed to represent information
  - **Controller** is the section that contains the logic of the application

  MVC in Data4Help
  In Data4Help the **Model** is the persistent data storage with the Entity-Relation SQL Database (view section 2.6.1) placed externally.

The **View** is the application displayed on the smartphone, is the only point of interaction with the client. This component contains the "UI Controller" that is the logic to handle the tap on the UI. It also contains the logic of AutomatedSOS because is required to be very reactive in case of emergency
The **Controller** is placed on the web server and is accessible as a RESTful endpoint. It contains the logic of the application

- **Altri pattern??**

2.6.3 Other design decision

- We decided not to let logic components to directly deal with the database but to interleave the *Database Link Manager* in order to build the logic layer independently from the persistency layer. Developers will be able to change the way the system interacts with the database or to change database's interfaces without having to change all the components.

3 Algorithm Design

3.1 Anomaly Detection

One of the main critical aspect of the platform is Anomaly detection. As described in section 2.6 the logic of this component is inside the application, to avoid issues like network congestion or not server not available. Here is a pseudocode of the component:

```
Threshold = UserPreference Threshold or default value
Emergency = 0

Loop on data feed receiving
    Check if ( BPMReceived < Threshold )
        -- "means that the smartwatch recorded a value under the threshold"

        Check if ( Emergency = 0 )
            Emergency ++

        Check if (Emergency = 3)
            -- "means that three BPM values are under threshold"
            call the ambulance number except if the user taps on 'Its not an emergency'

    Else
        Check if( Emergency != 0 )
            -- "means that the BPM goes under threshold for brief interval"
            Emergency = 0

        continue the data recording
```

4 User interface design

User interface mockups for Data4Help application were presented in the RASD document, section 3.1.1 - User interfaces.

5 Requirements traceability

Here we map each requirement described in RASD document to the system components that will provide services to satisfy it.

G1 Users can be recognised by their credentials.

- R1, R2 provided by *Login service* interface in *Access manager*

G2 Allow users to keep track of their health data.

- R3,R6 provided by *Data handler service* interface in *Health Sharing Manager*.

G3 Allow users to have access to an overview of their data, including health parameters and performed activities.

- R10 provided by *DB link service* interface in *DatabaseLink manager* module.

G4 Allow users to manage their data access policy.

- R4,R5 provided by *ManageAccess Request service* interface in *Health Sharing Manager*

G5 Allow users to monitor their performances during workouts.

G6 Each time vital signs go below a threshold value, first aid services have to be notified.

- R6 provided by *Data handler service* interface in *Health Sharing Manager* module.
- R7 provided by *SOS service interface* interface in *SOS manager* module.

G7 Allow users to organise running events.

G8 Allow third parties to access data.

R8 The machine has to be able to communicate with third parties. [TO CLARIFY]

- R9 provided by *Anonymous data sharing service* interface in *HealthSharing Manager*
- R10 provided by *DB link service* interface in *DatabaseLink manager* module.

6 Implementation, integration and test plan

This section describes how to proceed in implementing Data4Help application system. Since Data4Help's main goal is to handle and share data of its users, the role of a database is crucial for all the components presented in the application and the first thing to be implemented in the global system. It can be assumed that an instance of MySQL, offered by an external provider, is already been created and ready to be used by the application.

7 Effort spent

8 References