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 9, 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. 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

- Event: an event organised by a user;
- Notification: a warning that advise the user of an action required;
- RESTful API: API that follow the REST paradigm;
- Third-parties: external organisations.

1.3.2 Acronyms

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

1.3.3 Abbreviations

- $[Gn]$: n-th goal
- $[Rn]$: 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

This document essentially follows the structure defined by IEEE for DD document.

2 Architectural design

2.1 Overview: High-level

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

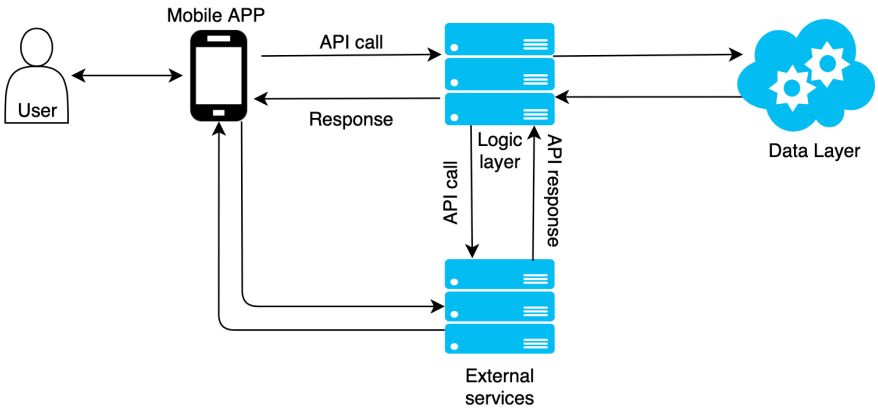

Figure 1: High level view of the system's architecture

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

- *Presentation layer*:
  - Individuals: an iOS application which contains a view of the entire system;
  - Third-parties: a web interface where manage authorisation 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 grows in a modular fashion and let us change cloud data provider following the system growth with the minimum effort.

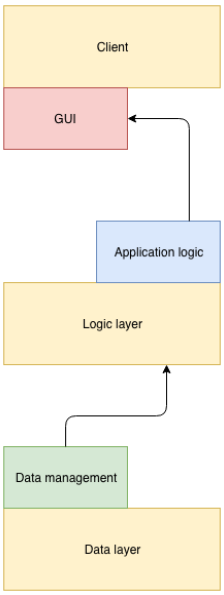

Figure 2: Distribution of application's function among the tier

2.2 Component view

We now provide a high level view of system's components. The system is composed by two client components: *Web Interface* and *Mobile Application*; they consume services offered by a set of logic components: *HealthSharing manager*, *SOS manager*, *RunEvent Manager* and *Access Manager*. Each of them will be interfaced with the *DatabaseLink Manager* which provides transparent access to the tier devoted to persistency.
We are now going to provide further details of each subsystem.

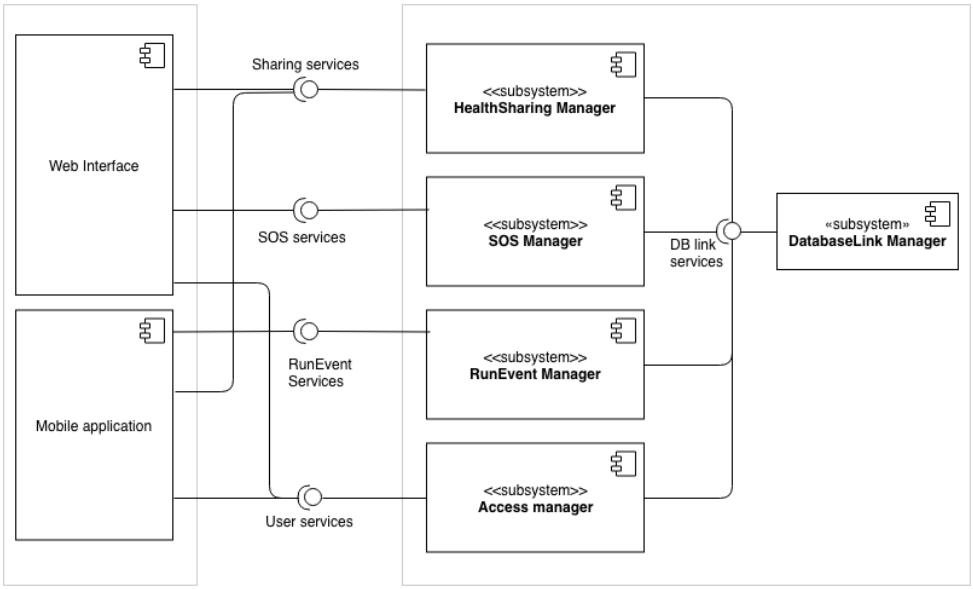

Figure 3: High level overview of system components

2.2.1 Component view of HealthSharing Manager

HealthSharing manager is composed of three main submodules:

- 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 guarantee data consistency in the whole system.
The *Data Elaboration Module* has to handle the operations devoted to data anonymisation and manage the requests of aggregated data from third-parties.

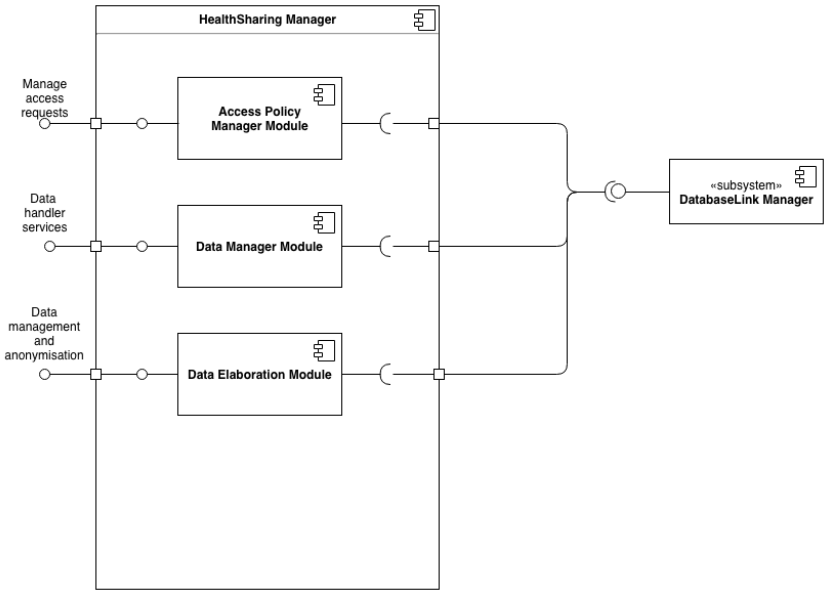

Figure 4: Specific description for HealthSharing Manager component

2.2.2 Component view of SOS Manager

The *Anomaly Detection Module* has to read and control heartbeat data from users that have activated SOS functionality in their mobile application.
In case of anomaly, the component activates the routine to handle the emergency cases.

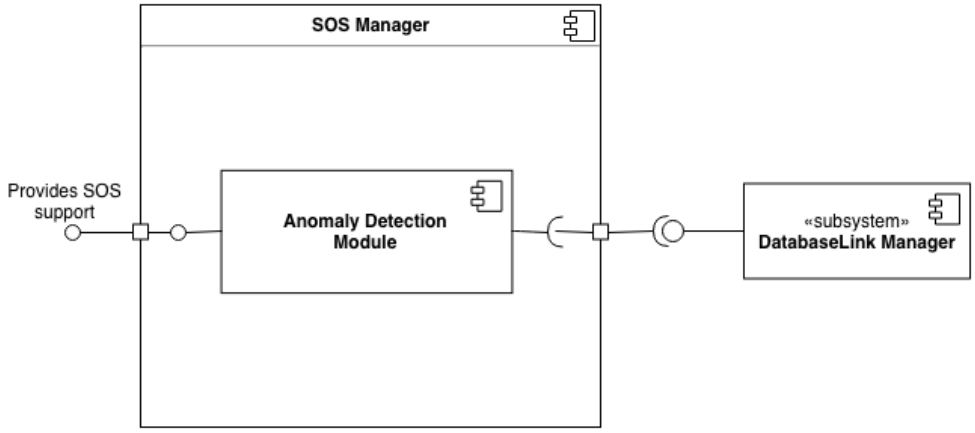

Figure 5: Specific description for SOS Manager component

2.2.3 Component view of RunEvent Manager

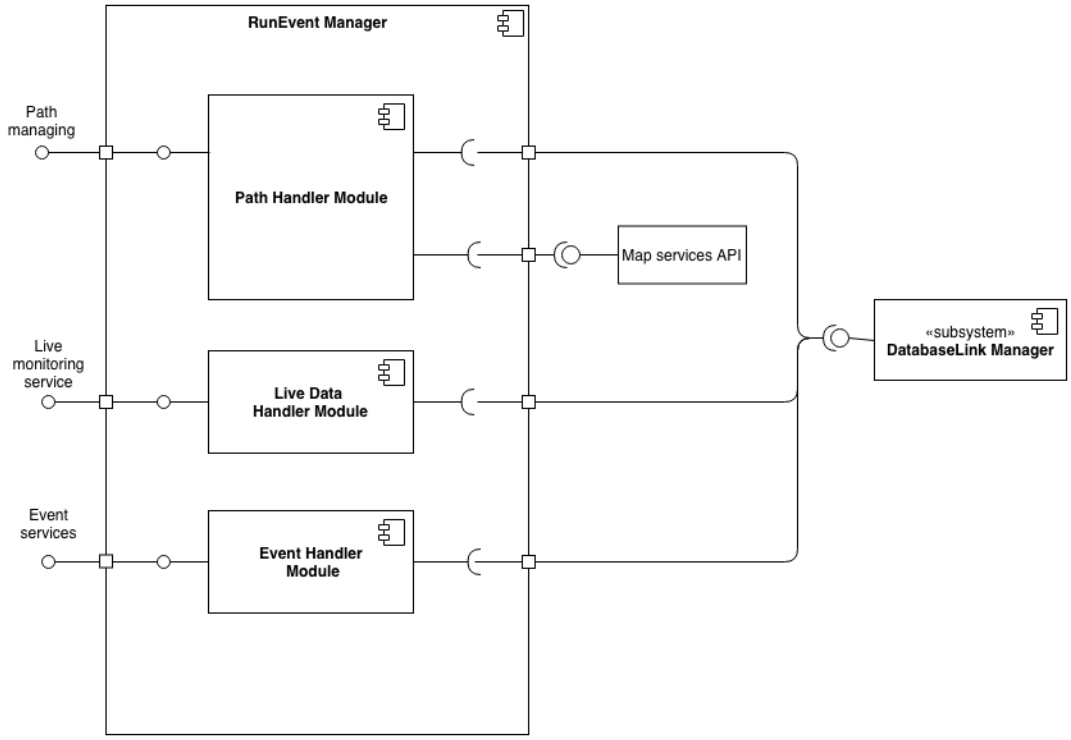

Figure 6: Specific description for RunEvent Manager

RunEvent Manager is composed of three main submodules:

- 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 map services API.
The *Live Data Handler Module* has to handle incoming live data from users and prepare and aggregate them for spectators. Moreover, it has to make them persistent for future access.
The *Event Handler Module* has to handle events creation and users' joining operations.

2.2.4 Component view of Access Manager

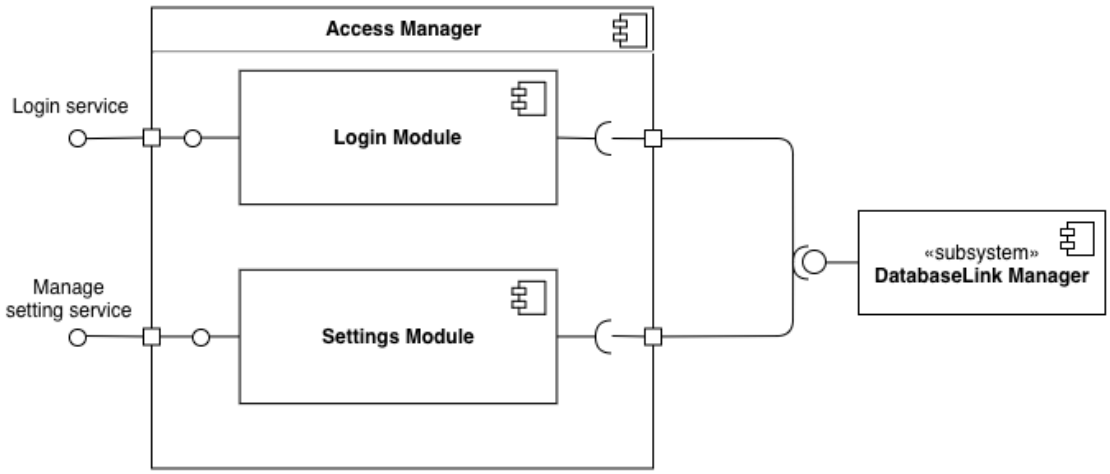

Figure 7: Specific description for Access Manager component

Access Manager is composed of two main submodules:

- Login Module;
- Settings Module.

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

2.2.5 Entity Relationship diagram

In this section we show the Entity Relationship schema of the adopted DB.

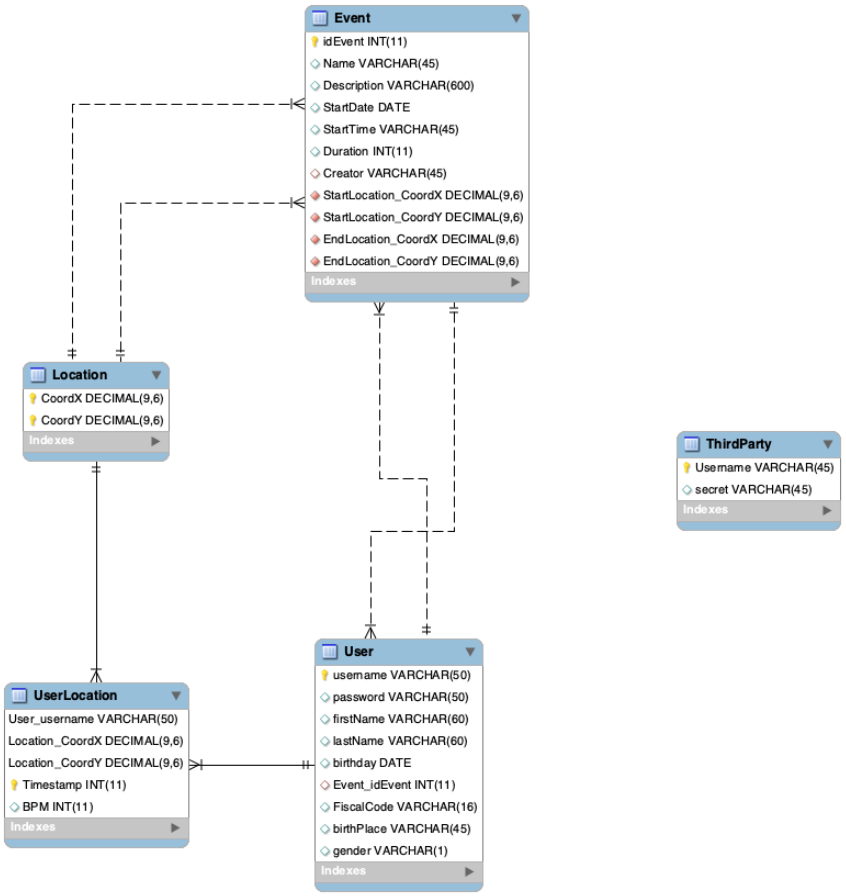

Figure 8: ER schema

2.3 Deployment view

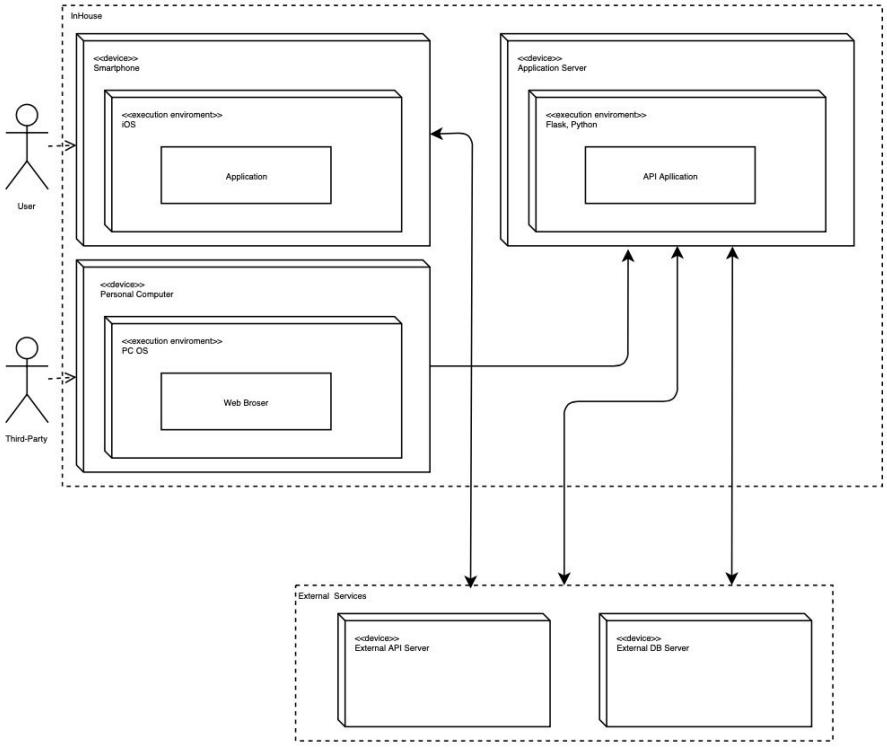

Figure 9: Deployment diagram

The Deployment diagram emphasises the nodes on which the platform runs; **InHouse** box represents the components developed inside the company while the components in the **External** box are on-demand softwares.

- **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 API directly (e.g. maps by Google);
- **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 access point to the DB but it also communicates with the external server API;
- **External Services**: both the external services are represented as black-boxes because we don't know how they are implemented.

3 Runtime view

3.1 Data acquisition Sequence Diagram

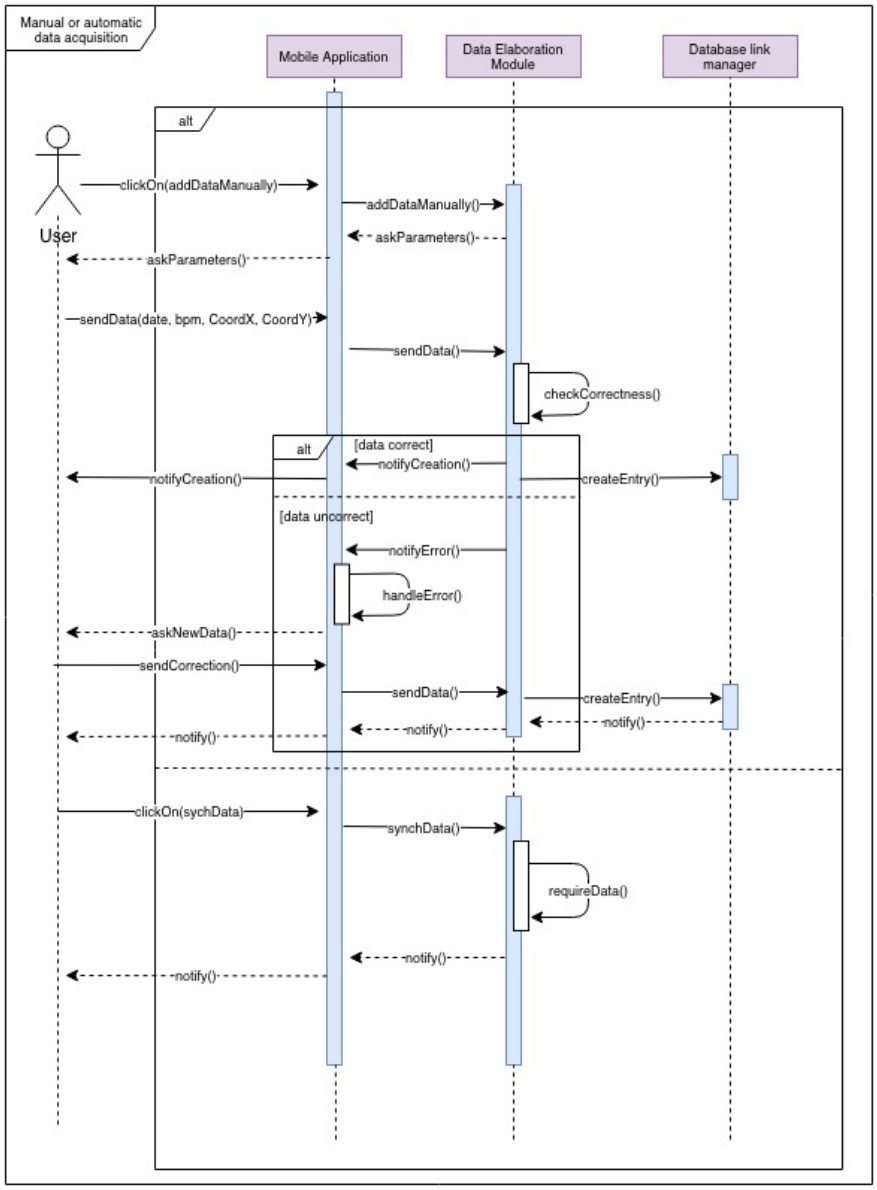

Figure 10: Manual or automatic data acquisition sequence diagram

3.2 AutomatedSOS Sequence Diagram

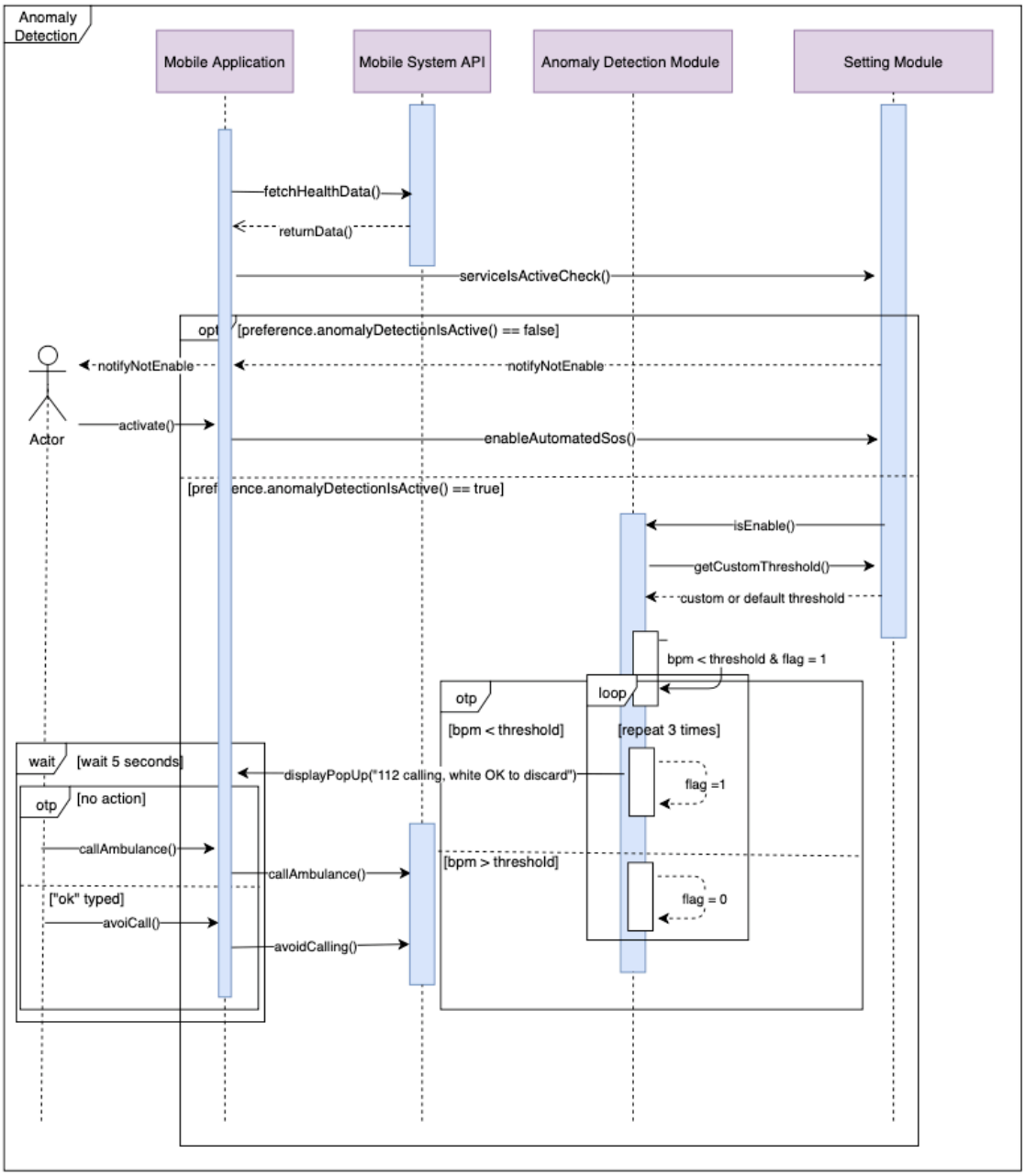

Figure 11: Automated SOS sequence diagram

The **Anomaly Detection** sequence diagram shown above represents the sequence of actions that may happen when an anomaly is detected. The Mobile application, using the Mobile Systems API, fetches the data from the smartphone. Whether the AutomatedSOS is enabled or disabled the behaviour of the platform is different. The **Setting Module** is queried and if the Anomaly Detection is disable a notification is sent to the **Mobile App component** and the user will be able to activate the service or discard it. Otherwise if it is enabled, the setting module is queried once again to obtain, if there exists, the custom threshold (otherwise the setting module component is going to return the default value). Since now the **Anomaly Detection Module** check each bpm data received until it reads three values under the threshold, in this case it display on the users mobile application a **PopUp** with written Warning! bpm under the minimum threshold. Now the user has 5 seconds to write OK in the Text Field to discard the call.

3.3 Event Creation Sequence Diagram

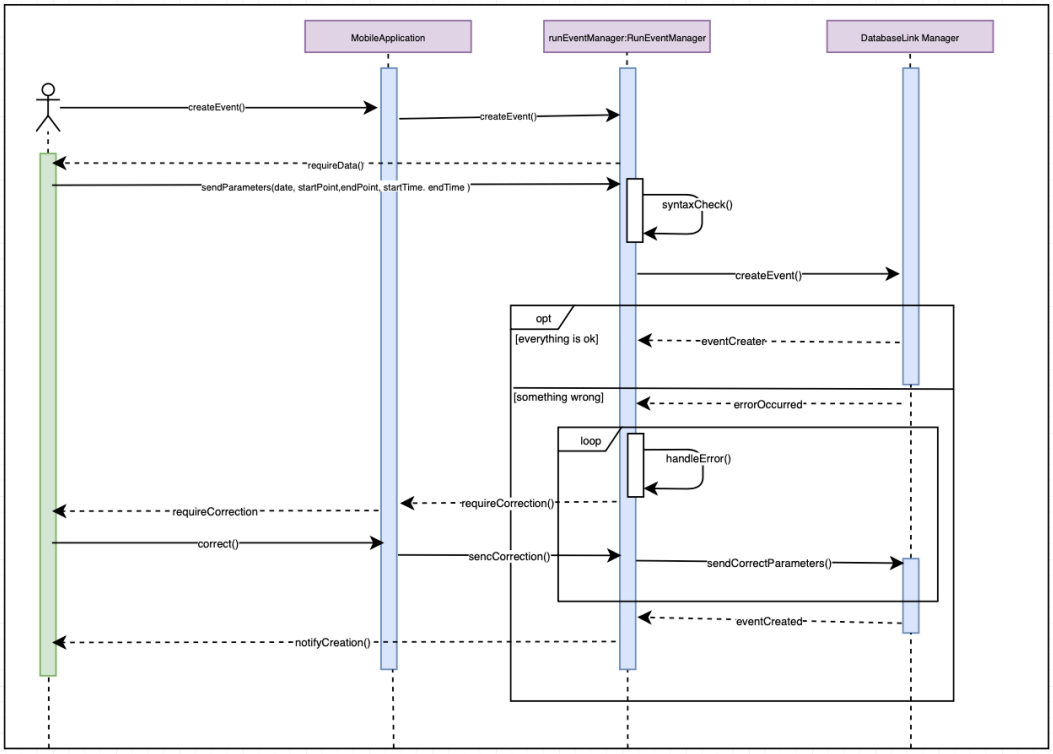

Figure 12: Event creation sequence diagram

The **Event Creation** Sequence Diagram shown above represents the sequence of actions that happen when a user tap on the button create Event. When the application controller is triggered by the tap of the user sends to the **Run Event Manager** the request. Since now the exchange is between the user and the component directly, in the sense that the control is entirely in the run event manager instead of in the mobile application component. After that the data are collected by the manager, it sends a request of creation to the **Database Link Manager** that tries to perform the insertion If everything goes correctly the event is created, otherwise the run event manager handle the error, and, if needed, it requires correction to the user directly. At the end of the process the user is notified of the occurred event creation.

3.4 Component interfaces

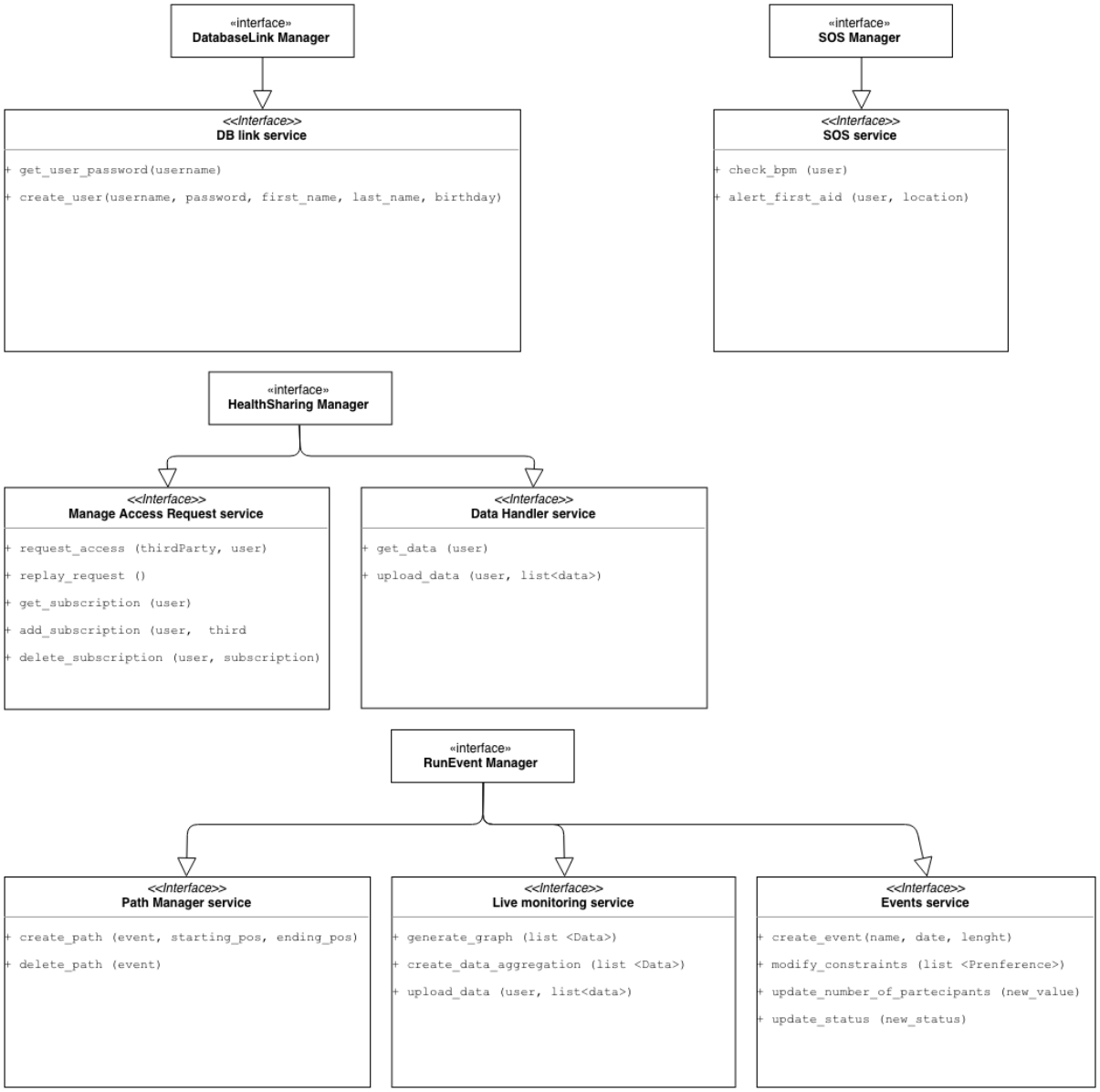

Figure 13: Components' interfaces descriptions diagram

3.4.1 API structure

The whole Data4Help infrastructure will be connected through a RESTful API system. JSON files will be used as serialisation data format.
All the API system will be implemented referring to a single endpoint (www.data4help.cloud) that TrackMe organisation 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 third-parties.

Here a complete description of each supported methods:

3.5 Selected architectural styles and patterns

Data4Help system is based on a tree tier architecture:

3.5.1 Overall Description

- **Presentation Tier** has to present the data to the user in such a way that they are meaningful for the user's perspective. It displays information obtained communicating with the other two tiers. For example, if a user performs a tap on the button "Create new run event", the presentation tier sends a request to the Logic Tier that will answer with a message requiring information to create the 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 quickest 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 sending back replies.
- **Data Tier** is composed of a DBMS that provides an interface to the DataBase externally located. It contains all the data in a Relational Schema (ER) (view Figure 8)

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

3.5.2 Design Patterns

- **Model View Controller (MVC)**

  MVC in a nutshell

  The MVC is an architectural pattern that permits to separate the presentation layer of the application from the business logic layer. It is composed by the following layers:
  - **Model**: the section that contains only the data of the application;
  - **View**: the section that contains the component needed to represent information;
  - **Controller**: the section that contains the logic of the application; it maps the user's input coming from the View to make changes to the Model section.

- **MVC in Data4Help**

  In Data4Help application the **Model** is the persistent data storage layer with the relational SQL Database (view section 2.2.5) placed externally. The **View** is the application displayed on the user's smartphone; is the only point of interaction with the client. This component contains the "UI Controller" that is the logic that handles the events on the UI. It also contains the logic of AutomatedSOS as explained in the above sections. The **Controller** is placed on the web server and is accessible with the RESTful API. It contains the logic of the application.

3.5.3 Other design decision

We decided not to let logic components directly deal with the database but to interleave the *DatabaseLink Manager* in order to build the logic layer independently from the persistent 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.

4 Algorithm Design

4.1 Anomaly Detection

One of the main critical aspects 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 reachability.
Here is the pseudocode of this 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
```

5 User interface design

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

6 Requirements traceability

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

- [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 *DBLink 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 *DBLink Service* interface in *DatabaseLink Manager* module.

7 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.

8 Effort spent

- Stefano Martina: 18.00h
- Alessandro Nichelini: 18.00h
- Francesco Peressini: 18.00h