Politecnico di Milano

Master's Degree in
Computer Science and Engineering

Software Engineering 2

---

# TrackMe
# Requirements Analysis and Specification Document

---

*Authors*
Alberto Archetti
Fabio Carminati

*Reference professor*
Elisabetta Di Nitto

v.0 - November 9, 2018

# Contents

# 1 Introduction

## 1.1 Purpose

### 1.1.1 Project description

TrackMe wants to develop a software-based service that allows individual users to collect health data, called `Data4Help`. This data can be retrived from the system and visualized according to different filters by a user interface.
The system allows third parties registration. Third parties can request access to users'collected data in two ways:

**Single user data** After a third party makes a request to the system for a single user data sharing, by providing user's fiscal code, the system asks the user for authorization; if positively provided, the third party is granted access to the user's data

**Amonymous group data** Third parties can be interested in big amounts of data, but not in who are the people providing it; the system, once the request is sent by the third party, checks if the data can be effectively anonymized (it must find at least 1000 people that can provide data matching the third party request's filters) and, if positively evaluated, grants access to the anonymized data to the third party

Third parties can subscribe to new data and receive it as soon as it is collected by the system.

Another service that TrackMe wants to develop is `AutomatedSOS`, built on `Data4Help`. This service analyzes users'data and calls a SOS whenever data exceedes the basic health parameters. For this particular purpose, system performance will be a critical aspect to be taken into account, because even the slightest delay matters in critical health situations.

### 1.1.2 Goals

Here we present the goals that will be reached once the project is completed:

`G.U1` Users can collect, store and manage their health data

`G.U2` Users can choose to have their health monitored; if their health is critical, an ambulance will be dispatched

`G.T1` Third parties can ask single users for their health data sharing

`G.T2` Third parties can request access to anonymized data that comes from groups of people

`G.T3` Third parties can subscribe to new data and receive it as soon as it is produced

## 1.2 Scope

### 1.2.1 World

Our *world* is composed of two main types of actors: users and third parties. Users are interested in monitoring their health parameters and third parties are interested in developing services or researches that exploit data gathered from the users. `Data4Help` is the service that acts as a bridge between these actors'needs.
Phenomena that occur in the *world* and are related to our application domain are

- physical conditions of the users
- third parties'projects, researches and interests
- ambulances dispatched by the SOS system

These phenomena exist in the *world*, but cannot be observerd directly by our system.

### 1.2.2 Shared phenomena

In order to communicate with the *world*, our system needs to share some aspects with it. We will list the aspects controlled by the world, but observable by the machine:

`S.1` physical parameters of the users, gathered through sensors on wereable devices

`S.2` third parties requests to the system for the data they need

`S.3` users'location, acquired through GPS signals

On the other hand, the aspects that occur in the machine, but are observable by the world are

`S.4` interfaces that organize the gathered data that can be filtered according to time or type of data

`S.5` messages for the SOS system, that are sent in case of critical health of a user

`S.6` payment requests

## 1.3 Definitions

**Data** Set of values of qualitative or quantitative variables[1]; in the context of `Data4Help` we will refer to quantitative variables concerning health parameters (Section 2.2.2)

**Aggregate data** See *DataSet*

**Anonymous data** *data entry* that doesn't contain information about the user from which it was produced; a *data set* is said to be anonymized if it contains only anonymous *data entries* and its cardinality is greater or equal than 1000

**Data entry** Tuple that corresponds to the user's parameters in a particular moment

**Data set** Set of *data entries*; depending on the context, it can identify a set of entries all belonging to a single user or or a set of anonymous entries belonging to more that 1000 users; a *data set*, among all *data* that the system is storing, can be identified and constructed according to the filters of a third party request (Section 2.2.3)

**Request** Third parties can ask the system for some data sharing through requests; requests are encoded through filling a form; the system, provided that the request is satisfiable, grants the third party access to the requested data (Section 2.2.3)

**Third party** Actor interested in collecting data from a single user or from an anonymous group of users (Section 2.3)

**Threshold** Numerical values related to a particular health parameter; they act as boundaries between the domain of critical health status and normal health status

**User** Actor interested in his/her health data collecting and managing; a user can also be interested in automating SOS calls whenever his health status becomes critical (Section 2.3)

## 1.4 Acronyms and abbreviations

**API** Application Programming Interface

[1] https://en.wikipedia.org/wiki/Data

**Data** Whenever the context refers to generic groups of *data entries*, the terms *data* and *data set* are interchangeable

**System** Software product that TrackMe wants to develop; can be interchanged with *S2B*

**S2B** Software To Be

## 1.5 Revision history

| Version | Log |
|---|---|
| `v.0` | First version of RASD completed |

## 1.6 Document structure

This document uses the IEEE standards for requirement analysis documents [2] as a guideline towards a clear and logical explanation of its contents:

- Section 1 gives a brief introduction on the project to be developed, by describing its goals and the environment in which it will be released; keeps track of the revision history
- Section 2 describes the world and the shared phenomena, by defining assumptions and constraints; it identifies the main functions of the project
- Section 3, as the main part of this document, is about requirement analysis; it has also sections about interfaces of the system and software attributes
- Section 4 contains the `Alloy` model that certifies system correctness
- Section 5 contains UML diagrams that help the understanding of the system structure
- Section 6 lists the work sessions that drove this project's development, ordered by date, as the hour counter of effort spent by each group member

# 2 Overall description

## 2.1 Product perspective

`Data4Help` is a service oriented to data acquisition and data sharing. Its software nature rises the necessity to combine it with another service able to directly retrive raw data from the *world*. Today we can find for sale multiple wearables that can acquire data from users and make it readable from software side. `Data4Help` users should already own these devices in order to exploit the data acquisition functionality (Section 1.2.2: `S.1`). Once the user registered to the service, its interface will gather the last data collected and organize it in a chart view, that can be filtered by date or type and rendered by the user interface (Section 1.2.2: `S.4`; Section 3.1.1).

It is mandatory for the user's wearable or device to provide a GPS signal, if the user wants to apply to the `AutomatedSOS` service. GPS will be used to track the user's location in case of health danger and the signal will be shared to the SOS service that already exists in the *world* (Section 1.2.2: `S.3`, `S.5`). This SOS service accepts messages that contain the GPS location of the person in health danger and an emergency log that `AutomatedSOS` generates from the acquired data. The log explains the suspected health danger and the data that passed the defined thresholds.

Third party organization are interested in data gathering. In order to allow them to make requests for specific types of data, `Data4Help` provides a user interface that is in charge of composing their request, to make it understendable by the software (Section 1.2.2: `S.2`; Section 3.1.1). The interface provides all the possible options that the third party can compose in order to provide the closest data request to its needs.

## 2.2 Product functions

Here we present the major functions that our product will offer. Some of them will entirely be handled internally by our system, but for others it will rely on external services. In the latter case, we will specify that the system will not directly provide the service and we will add examples of existing systems that can collaborate with ours, in order to guarantee the feasibility of the functions.

### 2.2.1 Profile management

The system will provide a registration form at which users and third parties can apply. Once registered, they will have a uniquely identifiable account, provided

the requested information for its creation (Table 1). Once the account has been successfully created, its owner can exploit the system's functionalities.

Note that accounts for users and third parties must be distinguishable from the system perspective, as it should offer different functionalities to different account types, in order to reflect the account owners'needs.

| **Account type** | **Required information** | **Optional information** |
|---|---|---|
| User | *email, password, fiscal code, date of birth, weight, height* | *social status, address, hours of work per day, hours of sport per week* |
| Third party | *email, password, third party name, third party description* | *website, research interests* |

Table 1: Example of registration form information

### 2.2.2 Data gathering

Data gathering is exploited through physical wearables that users wear and that can communicate with our software by API calls (ex. Google Fit API [6]). Data entries can be identified with a timestamp and the owner user.

Collected data types depend on wearables. For example, the most common sensors for health parameters that we find on smart clothing [5] are pulse, body temperature, electrocardiogram, myocardial and blood oxygen.

### 2.2.3 Data sharing

`Data4Help` relies on a database for data storage. Once data has just been added to the system, only the user that produced it will have access. Data sharing is exploited by granting access to required data also to third parties, if their requests have been accepted. Third parties can retrive these data by visualizing or downloading them from their inteface.

Third party requests are encoded in the system by filling a form that contains information like Table 2. They can be accepted or rejected. In the former case the third party is granted access to the data, while in the latter it is not granted access.

### 2.2.4 Data management

Once a user collected some data, he/she can organize it by changing the view options in the user interface. These options depend on the device the user is working on, but

| Request type | Accept condition | Filters |
|---|---|---|
| Single user | Target user should accept the request through his/her account | *fiscal code of target user, from-date, to-date, data types (weight, heart rate, etc.), time granularity (seconds, minutes, hours, etc.)* |
| Anonymous | Every user should have accepted the automatic sharing of requested data | *size, from-date, to-date, data types (weight, heart rate, etc.), user characteristics (age interval, weight interval, etc.)* |

Table 2: Example of third party request form

will always provide basic filters such as time interval or data type. Once filters have been selected by the user, the graphical interface will render a chart that organizes collected data according to them (see Section 3.1.1).

### 2.2.5 Payment handling

When third parties initialize requests, they are asked for payment by the system. Payment details are defined by TrackMe policy, so they won't be discussed in this document.

Our system will only initialize payment requests as the effective payment operation will entirely be handled by an external software. This software should

- instanciate payment processes
- check if the payment is feasible and, if not, notify to our system
- handle the payment operation
- notify to our system if the operation has concluded correctly, otherwise notify the error occurred

There are plenty of payment handlers that exist in our *world* and can be paired to our system (ex. PayPal API [7]).

### 2.2.6 SOS handling

This function is heavily dependant on the country in which the SOS will be handled. Our system will only communicate to the external emergency service that already

exists in the *world* through automatic API calls (ex. RapidSOS Emergency system for US [8]).

Calls to SOS are handled by `AutomatedSOS` that signals users GPS position and health status feedback. Calls occur, if the user previously subscribed to `AutomatedSOS`, when his/her health parameters are above certain thresholds. The time between health danger detection and emergency call must be less than 5 seconds.

## 2.3 User characteristics

The following list contains the actors that are involved in the system functions:

**User** Person interested in the `Data4Help` **data management** service; he/she is required to register to the `Data4Help` service in order to exploit its functionalities; he/she can also apply to the `AutomatedSOS` service; every user owns an appropriate device for `Data4Help` acquisition service that can monitor at least GPS location
ex. *an athlet that wants to monitor his/her physical status during sport activity, an old person that suffers from heart diseases or a sedentary worker that wants to keep track of his health parameters in the working hours*

**Third Party** Entity interested in the `Data4Help` **data sharing** service; it is required to register to the `Data4Help` service in order to exploit its functionalities; according to the scope of its requests, it may be interested on the data of a particular user or in aggregated chunks of data
ex. *the physician of a person that suffers from heart diseases or a statistical institute*

## 2.4 Assumptions, dependencies, constraints

### 2.4.1 Domain assumptions

**World**

`D.W1` Signals processed by wearables are encoded correctly and represent the status of the *world*

`D.W2` Given certain health parameters[2], it is possible to decide if a person is in health

[2]Section 2.2.2 discusses which parameters are current wearables able to detect; we won't discuss regarding which parameters should be taken into account to determine the health status of an individual and which thresholds should be defined, as it is a topic strictly related to medical research

danger just by checking wether the parameters are above or below certain thresholds

**Existing systems**

`D.E1` In the *world* already exists a SOS system that is able to dispatch ambulances and accepts emergency calls through an API

`D.E2` In the *world* already exists a payment handler that is able to deliver money payments and accepts calls through an API

`D.E3` In the *world* already exist wearables that encode signals for health status; these encoded signals are accessible from the software side

**Legal constraints**

`D.L1` Acquired data can be sold to third parties

### 2.4.2 Dependencies

`Data4Help` relies on

- **payment handler**, to deal with payment of third parties (Section 2.2.5)
- **wearables**, to encode users'data (Section 2.2.2)

`AutomatedSOS` relies on

- **wearables**, to encode GPS signals and users'parameters
- **SOS system**, for ambulance dispatching (Section 2.2.6)

### 2.4.3 Constraints

`Data4Help` acquires data through external devices owned by other companies, so it must be legally authorized to sell the data it acquires. In this document we assume that there are no legal issues for the selling activity, as TrackMe will develop contracts with the wearables companies in order to solve this issue.

# 3 Specific requirements

## 3.1 External interface requirements

### 3.1.1 User interfaces

We will present the mockups of `Data4Help` app with `AutomatedSOS` options for users. Filters and forms fields are presented only for illustrative purposes: they may change in future releases or in the final product. These mockups are intended only to give an idea of what the graphical interface of our system will be like.


Figure 1: Login screen

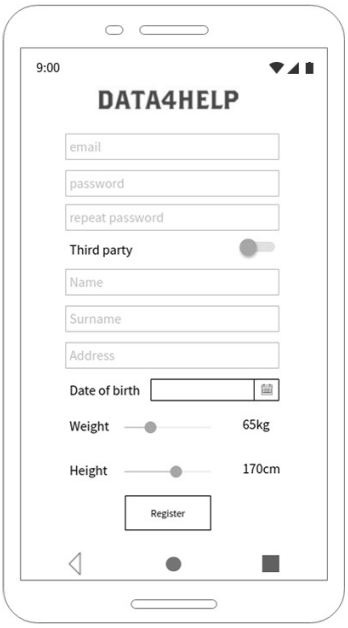


Figure 2: User registration form

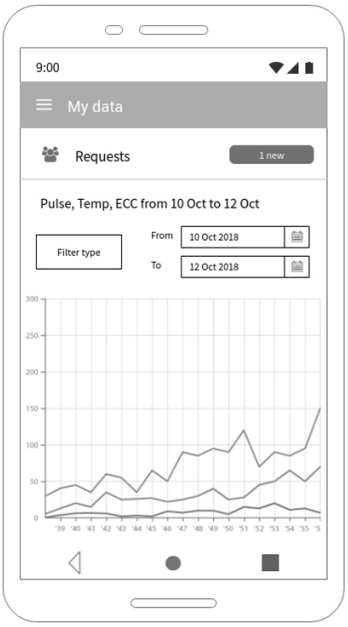


Figure 3: User default screen

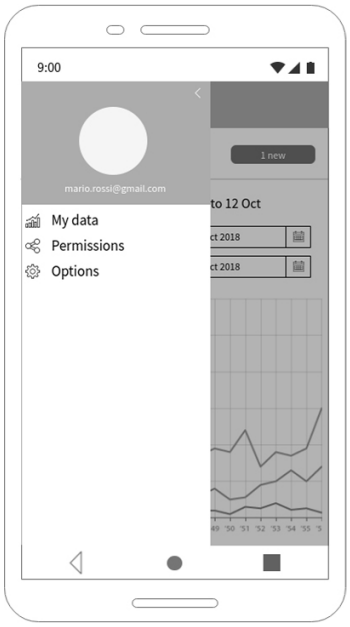


Figure 4: User sidebar

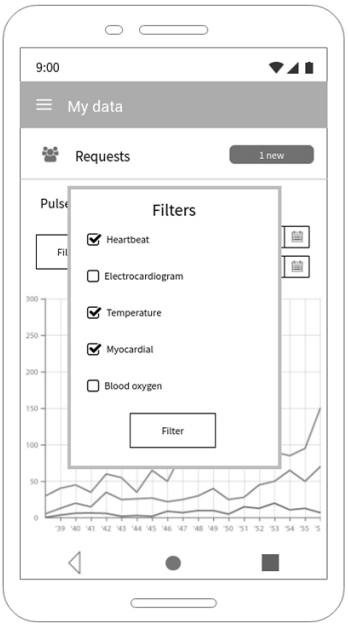


Figure 5: Graphical interface filters

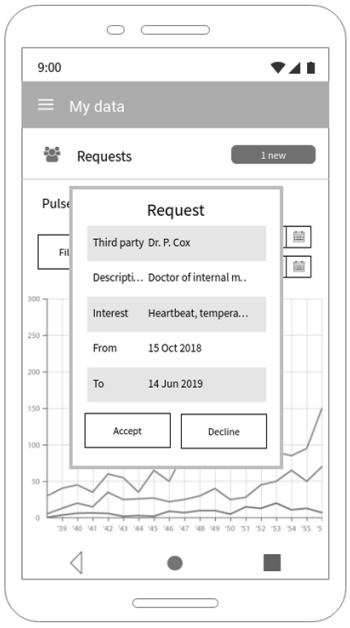


Figure 6: User request

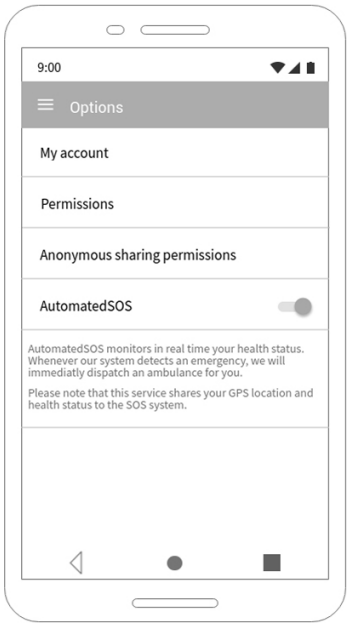


Figure 7: User options

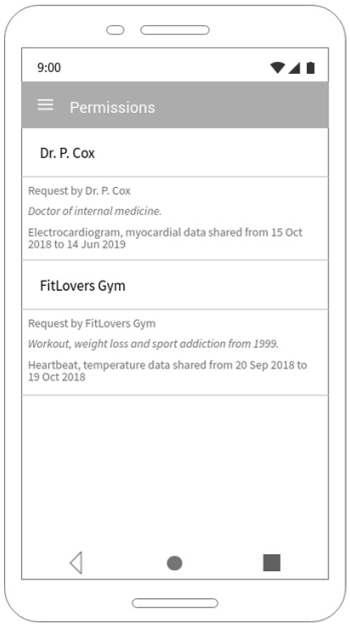


Figure 8: User permissions

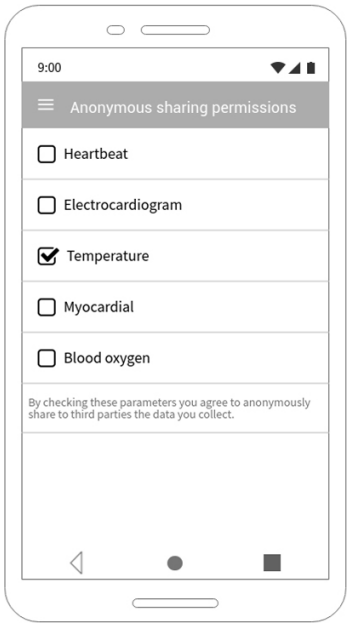


Figure 9: Anonymous sharing types

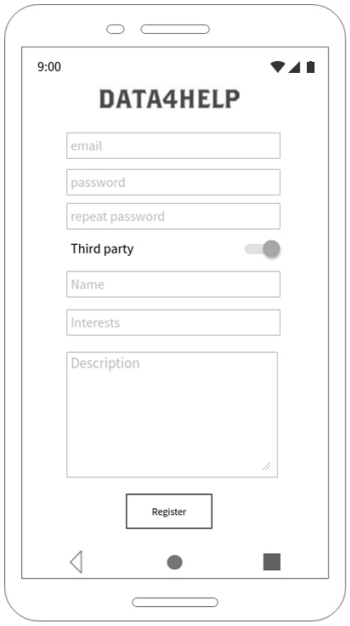


Figure 10: Third party registration

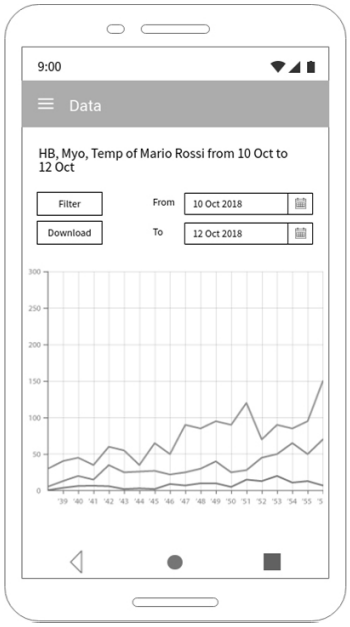


Figure 11: Third party default screen

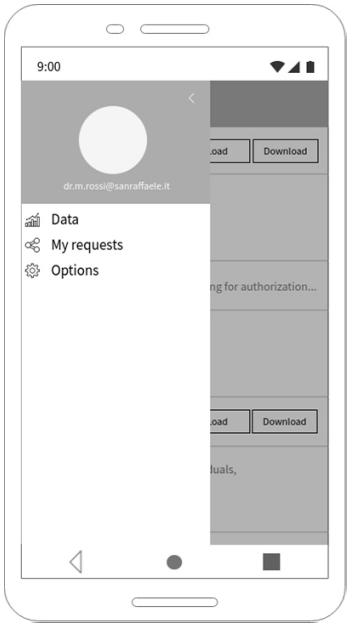


Figure 12: Third party sidebar

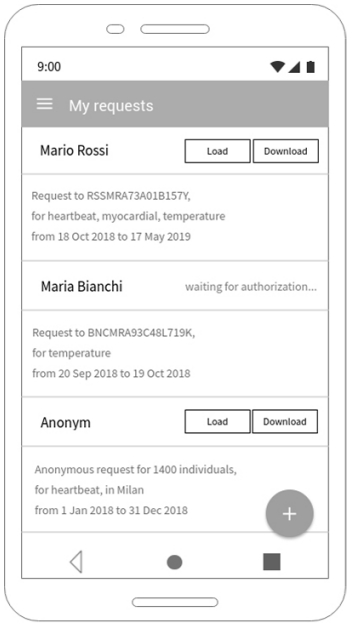


Figure 13: Third party requests

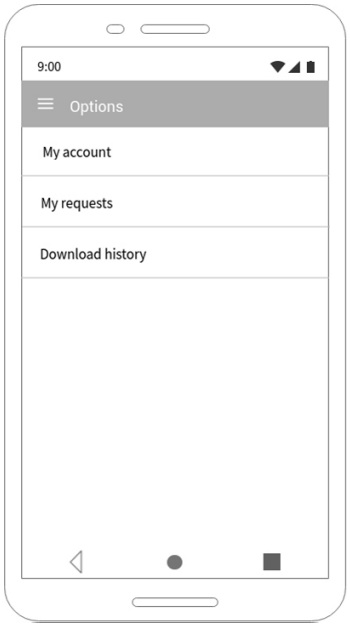


Figure 14: Third party options

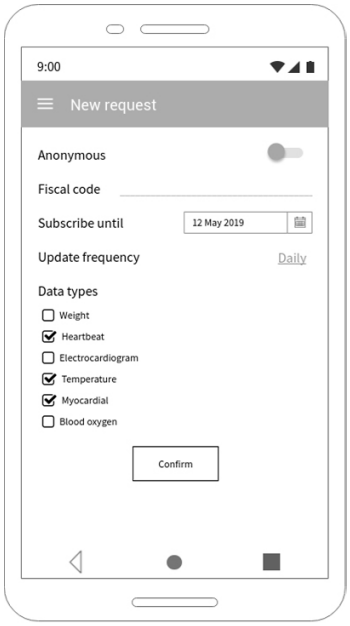


Figure 15: Single user request form

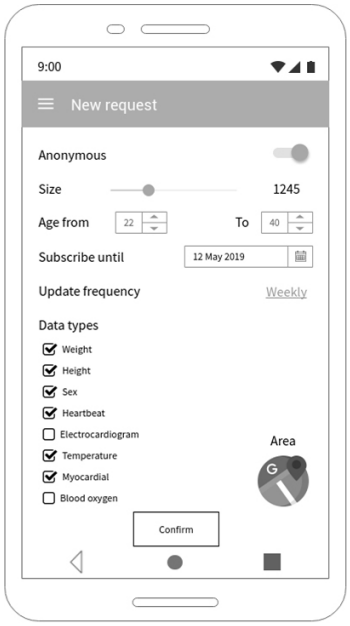


Figure 16: Anonymous request form

### 3.1.2 Software interfaces

`Data4Help` will not provide an automated software interface for user functionalities. However, for third parties that need automated access to our services, we will provide an API that communicates with messages encoded in JSON syntax. The API interface will be found in the software documentation and will be able to

- accept third party requests for access to new data
- accept third party requests for data they already have access to
- send data sets that third parties have requested, if they are allowed to have access to them

This API is used also by the app installed on third parties devices.

## 3.2 Functional requirements

**Account handling**

`R.A1` The system shall allow users registration

R.A2 The system shall allow third party registration

R.A3 The system shall distinguish between user and third party accounts

R.A4 The system shall guardantee account uniqueness by not allowing two account to have the same email

R.A6 The system shall allow users and third parties to access their account (*login*) only if they provide correct email and password

R.A7 The system shall allow users and third parties to exploit `Data4Help` and `AutomatedSOS` functionalities only if they are *logged in* their account

**Data encoding**

R.D1 The system shall encode data received through wearables'sensors and store it internally[3]

R.D2 The system shall be able to retrive data previously stored on request

R.D3 The system shall not erase data once it is stored internally

R.D4 Once the system just stored data, it shall allow access to that data only to the user account that was *logged in* while the data was collected

R.D5 The system shall be able to share the stored data to more than one account

R.D6 The system shall be able to compose groups of data entries (*aggregate data*) and anonymize them (every data entry in the group has no information about the providing user, such as fiscal code or email)

**Interfaces**

R.I1 The system shall provide a registration form for users and third parties

R.I2 The system shall provide to users and third parties an interface able to render data graphically, allowing filters like time interval or data type

R.I3 The system shall provide a request form to the third parties

**Data sharing requests**

[3]for example the system can store data into an internal database or can save it using a cloud service; the important aspect is that data must be retrivable in the future

`R.R1` The system shall allow third parties to ask for data sharing of a single user by filling the request form

`R.R2` The system shall ask the user to accept or decline every single-user request of data sharing by third parties that has him/her as target user

`R.R3` The system shall provide access to the target user data to the third party only if the user accepted the request of the third party, otherwise the system shall notify to the third party that the user declined the request

`R.R4` The system shall allow third parties to ask for data sharing of *aggregate data* by filling the request form

`R.R5` The system shall be able to check if a request for *aggregate data* by a third party can be properly anonymized (there shall be at least 1000 user data entries that fit the parameters of the request)

`R.R6` The system shall provide access to the *aggregate data* to the third party only if the the request can be properly anonymized, otherwise the system shall notify the third party that its request cannot be properly anonymized

`R.R7` The system shall provide access to a third party to newly produced data if it fits the third party request

**SOS calls**

`R.S1` The system shall provide to the user an option to apply to `AutomatedSOS`

`R.S2` If user applied to `AutomatedSOS`, the system shall monitor his/her parameters in real time by checking whether they are above or below the thresholds

`R.S3` If user applied to `AutomatedSOS` and his/her health parameters are critical (above or below thresholds), the system shall send an emergency call to the SOS system

`R.S4` When the system sends an emergency call, it shall provide to the SOS system API user's GPS location and user's health status through his/her critical parameters encoding

## 3.3 Scenarios

### 3.3.1 Stroke detection

Luke is a 65 years old man that after 40 years of work at the post office finally got retired. Because of the sedentary nature of his work he is worried that he may suffer a stroke. In order to monitor his health parameters, he buys a smartwatch that can monitor his heartbeat and downloads the `Data4Help` app. The system allows him to create an account after filling all the required information (Table 1). The system explicitly asks Luke if he wants to join the `AutomatedSOS` service and Luke accepts. The system starts checking in real time Luke's health parameters that are collected through his smartwatch.

After some days Luke's health parameters exceede thresholds because of a stroke: the system recognizes immediatly the critical situation and forwards in less than 5 seconds an emergency notification to the SOS system, providing Luke's GPS location and health status feedback.

### 3.3.2 Anonymous request

The franchise BodySlim is opening a new gym in Milan near Parco Sempione. Its most succesful and distinctive characteristic is the timetable: open 365 days per year, 8 hours per day. In order to maximize the revenue the management would like to know in which time of the day potential customers do physical activity. A reliable sample can be obtained from Data4Help, therefore BodySlim creates a third party account by giving the required information (Table 1). The franchise fills the third party request form for an anonymous request (Table 2): at least 2000 individuals of age between 20 and 60 in Parco Sempione area. As BodySlim confirms the request, the system fetches in its database users that fit BodySlim's request, composes a data set containing their data entries, anonymizes the data set and shares it with BodySlim's account. The franchise can now filter data by hour of day and identify when is Parco Sempione most frequented: these are, from its perspective, the most profitable hours in which the gym should stay opened.

### 3.3.3 Single user request

Rose has just discovered that she's expecting a daughter. Dr. Harold, her gynecologist, needs to keep monitoring her blood ossygen, electrocardiogram and heartbeat, in order to understand if the pregnancy is proceeding well. He decides to exploit `Data4Help` third party functionalities, as he does with his other patients. He asks Rose to create an account of `Data4Help`, then logs into his account by providing

email and password. He navigates to the request form and inserts Rose's fiscal code and the data types he wants to have access to for the next nine months. After the request has been confirmed, Rose receives the notification of Dr. Harold's request to her account and accepts it.

Nine months later, Marie is born. She is beautiful and healthy, thanks to the pregnancy monitoring by Dr. Harold, exploited through the `Data4Help` service.

### 3.3.4 Subsribing to new data

Jack is a policeman that has decided to spend his holidays in a spa called BeautySPA. It will allow him to lose 15kg thanks to fitness activity and healthy diet. In order to understand if this treatment is efficient over time BeautySPA decides that Jack's weight and body temperature should be monitored. The policeman and BeautySPA create respectively a user and a third party account of `Data4Help` and the system, after checking that all the required information (Table 1) are properly filled, allows them to exploit its functionalities.

BeautySPA immediately applies for the Jack's health parameters through a third party request. The app asks to Jack the authorization whether to provide his data to the spa or not. He accepts and `Data4Help` shares his data to BeautySPA.

Unfortunately, after just one month, the policeman had a weight increase of 6kg. In order to better understand which may be the cause the spa applies for more parameters (blood oxygen and heartbeat) through another third party request. Jack receives the notification, but, unhappy about his previous treatment, declines the request. BeautySPA won't have access to Jack's blood oxygen and heartbeat parameters.

### 3.3.5 Graphical interface

Paul is studying Telecommunication Engineering at Politecnico di Milano. He would like to partecipate at PolimiRun (held in May), therefore he has trained twice a week since January. In order to monitor his progress, he downloaded the `Data4Help` app and created a user account. He allowed anonymous sharing of heartbeat and blood ossygen to third parties. The system periodically collected Paul's parameters and saved the correspondent data entries in its database.

At the end of April Paul wants to check if the PoliRun is beyond his physical capacities. After the login phase, he filters his data through the graphical interface, by limiting the rendering to blood ossygen and heartbeat weekly entries from January to April. The system retrieves the required data and renders it. Paul sees the impressive physical improvement during the four months (blood oxygen increase,

heartbeat per minute reduced) and states that the PolimiRun is absolutely realistic for him.

### 3.3.6 Multiple single user requests

FastAndEasy, a car sharing company, has a business model based on how much time its clients rent one car in a week. After five years of activity, the society has to decide whether to change this model with a kilometers-based one or keeping the old time-based one. In order to take the right decision, FastAndEasy needs to know the GPS location of all of its clients while they are driving cars for the next three months. Because of this time-limited solution it would be a waste of money to implement a new system from scratch for picking up these data. FastAndEasy decides to rely on `Data4Help` and creates a third party account. It asks through multiple third party request forms the GPS location of its clients, identified by fiscal code. Many of them accept the request, forwarded by `Data4Help`, after FastAndEasy confirmation.

After three months, FastAndEasy brings the collected data to the Board of Directors. They can now compute the possible earnings of the new model and compare them with the old ones, in order to take a decision about the business model.

## 3.4 Use cases

| ID | UC.1 |
|---|---|
| **Name** | Sign in of User to Data4Help |
| **Actors** | User |
| **Entry Condition** | Data4Help app downloaded on user's smartphone |
| **Flow of Events** | 1. User clicks on Register button of the login screen (Figure 1)<br>2. User fills the registration form (Figure 2)<br>3. User checks which data types he/she intends to anonymously share to third parties (Figure 9)<br>4. User checks whether he/she wants to apply to AutomatedSOS system<br>5. User clicks Register button of the registration form<br>6. The system creates a Data4Help account for the user |
| **Exit Condition** | User's account has been successfully created and added to the system database |
| **Exceptions** | 5.* The field email is invalid or corresponds to an email that already exists in the system database; the field fiscal code is not valid; the password fields don't correspond<br>The system notifies the issue to the user and the **Flow of Events** returns to 2, erasing invalid fields |
| **Special Requirements** | |

* The item number in the **Exceptions** section corresponds to the phase in the **Flow of Events** in which the exception can be thrown

| **ID** | `UC.2` |
|---|---|
| **Name** | Sign in of Third party to `Data4Help` |
| **Actors** | Third party |
| **Entry Condition** | `Data4Help` software downloaded on third party's device |
| **Flow of Events** | 1. Third party clicks on `Register` button of the login screen (Figure 1)<br>2. Third party fills the registration form (Figure 10)<br>3. Third party clicks `Register` button of the registration form<br>4. The system creates a `Data4Help` account for the third party |
| **Exit Condition** | Third party's account has been successfully created and added to the system database |
| **Exceptions** | 3. The field `email` is invalid or corresponds to an email that already exists in the system database; the `password` fields don't correspond<br>The system notifies the issue to the third party and the **Flow of Events** returns to 2, erasing invalid fields |
| **Special Requirements** | |

| **ID** | `UC.3` |
|---|---|
| **Name** | Log in of User in `Data4Help`* |
| **Actors** | User |
| **Entry Condition** | User has already created an account |
| **Flow of Events** | 1. User fills `email` and `password` fields of the login screen (Figure 1)<br>2. User clicks on `Login` button |
| **Exit Condition** | User is successfully logged in the system and can exploit all the system services; the graphical interfaces moves to the default screen (Figure 3) |
| **Exceptions** | 2. The system discovers that field `email` is invalid or doesn't corresponds to an email that already exists in the system database; the system discovers that field `password` doesn't correspond to the one paired with the email<br>The system notifies the issue to the user and the **Flow of Events** returns to 1 |
| **Special Requirements** | |

* The use case *Log in of Third party* in `Data4Help` is equivalent to this one, by changing *user* with *third party*

| **ID** | `UC.4` |
|---|---|
| **Name** | Call an ambulance |
| **Actors** | SOS System, User |
| **Entry Condition** | User subscribed to `AutomatedSOS`, his/her health parameters exceede thresholds |
| **Flow of Events** | 1. The system retrives the user's data entry which parameters exceeded thresholds from the database<br>2. The system composes a SOS message with the GPS location of the user and the critical parameters contained in the data entry<br>3. The system forwards to the SOS system the SOS message<br>4. The SOS system accepts the message |
| **Exit Condition** | The SOS system succesfully received the SOS message; the ambulance dispatchment is beyond the system's scope |
| **Exceptions** | 4. The SOS system is not reachable; **Flow of Events** rollbacked to 3 |
| **Special Requirements** | The system shall make the call to SOS system within 5 seconds from the moment of health danger detection |

| **ID** | `UC.5` |
|---|---|
| **Name** | Accept a single user request |
| **Actors** | Third party, User |
| **Entry Condition** | Third party and user have already created respectively a third party and a user account and they are already logged in the system |
| **Flow of Events** | 1. Third party opens the sidebar (Figure 12)<br>2. Third party clicks on `My requests` button<br>3. Third party fills the form for a single user request (Figure 15), adding user's fiscal code<br>4. Third party clicks the `Confirm` button<br>5. The system receives the request by the third party and forwards a notification to the user<br>6. User clicks on the notification and opens the request information (Figure 6)<br>7. User clicks the `Accept` button<br>8. The system receives user's response and shares requested data to the third party |
| **Exit Condition** | Third party can exploit user's requested data |
| **Exceptions** | 4. The system doesn't find any user whose fiscal code correspond to the one inserted by the third party; **Flow of Events** rollbacked to 3<br>7. User declines the request; the system notifies to the third party that is not allowed to exploit user's data |
| **Special Requirements** | |

| ID | UC.6 |
|---|---|
| **Name** | Accept an anonymous group request |
| **Actors** | Third Party |
| **Entry Condition** | Third party has already created an account and it's already logged in the system |
| **Flow of Events** | 1. Third party opens the sidebar (Figure 12)<br>2. Third party clicks on `My requests` button<br>3. Third party fills the form for an anonymous group request (Figure 16); it selects the data set size (at least 1000)<br>4. Third party clicks the `Confirm` button<br>5. The system fetches from the databse a quantity of data entries equal to the data set size selected by the third party; all of the data entries match the request filters<br>6. The system anonymizes the data entries and composes an anonymous data set<br>7. The system shares the data set to the third party |
| **Exit Condition** | Third party can exploit requested data |
| **Exceptions** | 5. The system doesn't find enough data entries that match the request filters; the system notifies to the third party that its request cannot be fulfilled |
| **Special Requirements** | |

| **ID** | `UC.7` |
|---|---|
| **Name** | Show and filter acquired data |
| **Actors** | User |
| **Entry Condition** | User has already created an account and he/she is already logged in the system |
| **Flow of Events** | 1. User opens the sidebar (Figure 4)<br>2. User clicks on `My data` button<br>3. User clicks on `Filter type` button and ticks which filter types he/she want to visualize (Figure 5)<br>4. User selects the time period of the data visualization<br>5. The system processes user's request and renders the new data chart |
| **Exit Condition** | User can see the data he/she requested according to the specified filter |
| **Exceptions** | 5. User inserted a non valid time period; the system notifies the user of the error; **Flow of Events** rollbacked to 4 |
| **Special Requirements** | |

| Raw ID | Goal ID | Req ID | Use Case ID |
|---|---|---|---|
| r1 | G.U1 | R.A1, R.A3, R.A4, R.I1 | UC.1 |
| r2 | G.T1, G.T2, G.T3 | R.A2, R.A3, R.A4, R.I1 | UC.2 |
| r3 | G.U1 | R.A3, R.A6, R.A7 | UC.3 |
| r4 | G.T1 | R.A3, R.I3, R.R1, R.R2, R.R3 | UC.4 |
| r5 | G.U2 | R.S1, R.S2, R.S3, R.S4 | UC.5 |
| r6 | G.T2 | R.A3, R.D2, R.D5, R.D6, R.I3, R.R4, R.R5, R.R6 | UC.6 |
| r7 | G.U1 | R.A3, R.A7, R.I2, R.D2 | UC.7 |

Table 3: Traceability matrix

## 3.5 Performance requirements

`R.P1` The system shall detect in **negligible time** (less than 0.1 seconds) if a data entry has values that exceed thresholds[4]

`R.P2` The system, once an emergency is detected (as in `R.P1`), shall forward an emergency call to the SOS system in **less than 5 seconds**

`R.P3` The system shall grant data updates with at most **seconds precision** to third parties (time granularity can be selected in the request form, Table 2)

## 3.6 Design constraints

### 3.6.1 Standards compliance

See Section 2.4.3 for data sharing assumptions.

### 3.6.2 Hardware limitations

The smartphone app version of `Data4Help` requires Android/iOS operating system, GPS location and 3G/4G connectivity to the smartphone on which it is installed, in order to exploit its basic functionalities. Moreover it is necessary that each user has his/her own wearables, able to collect data types that they want to monitor (Section 2.2.2).

## 3.7 Software system attributes

### 3.7.1 Reliability

The system shall guarantee reliability $R = 1 - F$ ($F$ = probability of failure) as close as possible to 1. This is a very important aspect for `AutomatedSOS`, that needs fast and reliable communication with the SOS system. A possible way to decrease $F$ is to open two channels of communication that rely on different technologies with the SOS system: one via internet and the other via SMS. In this way we can decrease the overall $F$ to $F_{internet} \cdot F_{SMS}$, assuming that they are independent.

There are two major situations which may result in an *internal* system failure: components failure or unexecpected increase of communication traffic. At this level

[4] this situation directly corresponds to user's health in danger, according to the domain assumptions

of abstraction is not strictly required, but in both of these cases a distributed architecture is the optimal solution: components redundancy and load balancing minimize the probability of these failures.

### 3.7.2 Availability

The system is expected to be available 99.99% of the time. In case of failure (Section 3.7.1) it shall be able to recover within 1 second, otherwise `AutomatedSOS` won't be able to respect the requirement of Section 3.5.

### 3.7.3 Security

The system shall store securely account details and data collected into its database. Payment history shall also be protected, but it's not a critical aspect of security issues, because the system won't store any payment information of its clients (payments are handled externally, Section 2.2.5).

Another security issue regards anonimity of users in anonymous data sets, requested by third parties through anonymous group requests. The system shall not allow third parties to retrieve any personal information about the users whose data entries compose anonymous data sets.

### 3.7.4 Maintainability

`Data4Help` has to collect data through wearables devices (Section 2.2.2). Data types addition and management should be implemented in a flexible and extensible way, because wearables technologies change rapidly and our software should adapt in the least invasive way possible.

### 3.7.5 Portability

The system shall run on most iOS/Android devices, in order to simplify communication with wearables, as they often interact with smartphone apps.

A Java implementation would guarantee compatibility also with most of desktop operating systems for better data visualization.

# 4 Formal analysis using Alloy

```
open util/integer
open util/boolean


/** *** DESCRIPTION ***
 *
 * In this Alloy model we will represent Data4Help
 * and AutomatedSOS services in a simplified way:
 *  - Data4Help: we will describe third party
 *    requests according to some assumptions
 *    (single user requests always target ALL
 *    user data in the system; anonymous group
 *    requests have size, but no filter available)
 *  - AutomatedSOS: we will assume that every user
 *    is subscribed to AutomatedSOS and we will test
 *    if SOS calls are always performed correctly
 */


/** *** SIGNATURES *** */

/**
 * Actors of the system
 */
sig User, ThirdParty {}

/**
 * A DataEntry corresponds to one wearable
 * acquisition
 */
sig DataEntry {
  value : one Int
}

/**
 * Every DataEntry has the same data type,
 * therefore a single threshold is needed
 */
one sig Threshold {
  value : one Int
}

/**
 * Simplified single user request:
```

```
 * third parties ask for ALL target's data
 */
sig SRequest {
  target   : one User,       // target user
  accepted : one Bool,       // user response
  shared   : set DataEntry   // data shared
} {
  // no data outside the system
  all s : System | shared in User.(s.data)
  // decline implies no data sharing
  accepted = False ⟹ shared = none
  // accept implies data sharing
  all s : System | accepted = True
    ⟹ shared = target.(s.data)
}

/**
 * Simplified anonymous group request:
 * filters are not allowed
 */
sig ARequest {
  size   : one Int,         // # of desired d.e.
  shared : set DataEntry    // data shared
} {
  all s : System |
    // no data outside the system
    (shared in User.(s.data)) and
    // enough data implies sharing
    (#s.data ≥ size ⟹ #shared = size) and
    // not enough data implies no data sharing
    (#s.data < size ⟹ shared = none)
}

/**
 * S2B with database that contains data entries
 * and requests
 */
sig System {
  // single user requests
  srequests : ThirdParty ->set SRequest,
  // anonymous group requests
  arequests : ThirdParty ->set ARequest,
  // all acquired data entries
  data  : User ->set DataEntry,
  // emergency calls
  calls : set DataEntry
```

```
} {
  // no data entry outside the system
  User.data = DataEntry
  // no req. outside the system
  ThirdParty.srequests = SRequest
  ThirdParty.arequests = ARequest
  // no two applicants for same req.
  all disj tp, tp' : ThirdParty |
    tp.srequests & tp'.srequests = none
  all disj tp, tp' : ThirdParty |
    tp.arequests & tp'.arequests = none
  // no same data entry for two users
  all disj u, u' : User |
    u.data & u'.data = none
  // no calls outside the system
  all d : calls | d in User.data
  all d : User.data |
    // above threshold implies emergency
    (d.value > Threshold.value ⟹ d in calls) and
    // below threshold implies no emergency
    (d.value ≤ Threshold.value ⟹ d not in calls)
}


/** *** PREDICATES *** */

/**
 * User u acquires a new data entry d,
 * s is the old system,
 * s' is the new system
 */
pred acquire[u : User, d : DataEntry, s, s' : System] {
  // no unrequired changes
  s'.srequests = s.srequests
  s'.arequests = s.arequests
  // add entry
  s'.data = s.data + u->d
  // eventually call emergency
  d.value > Threshold.value ⟹
    s'.calls = s.calls + d
  d.value ≤ Threshold.value ⟹
    s'.calls = s.calls
}

/**
 * Third party tp makes a single user request r
```

```
 * to user u; u gives b as response (true
 * means accept, while false means decline),
 * s is the old system,
 * s' is the new system
 */
pred makeSRequest[s, s' : System, tp : ThirdParty,
                  r : SRequest, u : User, b : Bool] {
  // no unrequired changes
  s'.data = s.data
  s'.calls = s.calls
  s'.arequests = s.arequests
  // compose request
  r.accepted = b
  r.target = u
  // add request
  s'.srequests = s.srequests + tp->r
}

/**
 * Third party tp makes an anonymous group request r
 * expected to have n entries,
 * s is the old system,
 * s' is the new system
 */
pred makeARequest[s, s' : System, tp : ThirdParty,
                  r : ARequest, n : Int] {
  // no unrequired changes
  s'.data = s.data
  s'.calls = s.calls
  s'.srequests = s.srequests
  // compose request
  r.size = n
  // add request
  s'.arequests = s.arequests + tp->r
}


/** *** ASSERTIONS *** */

/**
 * Calls are always performed
 */
assert callOnEmergency {
  all s, s' : System, d : DataEntry, u : User |
      // above threshold implies emergency call
      (acquire[u, d, s, s'] and
```

```
        d.value > Threshold.value
         ⟹ d in s'.calls) and
       // below threshold implies no emergency call
       (acquire[u, d, s, s'] and
        d.value ≤ Threshold.value
         ⟹ d not in s'.calls)
}

/**
 * Single user request access is correct
 */
assert verifyAccessForSRequest {
  all s, s' : System, r : SRequest, tp : ThirdParty |
    // accept implies sharing
    (makeSRequest[s, s', tp, r, r.target, True]
      ⟹ r.shared = r.target.(s.data)) and
    // decline implies no sharing
    (makeSRequest[s, s', tp, r, r.target, False]
      ⟹ r.shared = none)
}

/**
 * Anymous group request access is correct
 */
assert verifyAccessForARequest {
  all s, s' : System, r : ARequest,
  tp : ThirdParty, n : Int |
    // enough data implies sharing
    (n ≤ #s.data ⟹
      (makeARequest[s, s', tp, r, n]
        ⟹ #r.shared = n)) and
    // not enough data implies no sharing
    (n > #s.data ⟹
      (makeARequest[s, s', tp, r, n]
        ⟹ r.shared = none))
}


/** *** COMMANDS *** */
run acquire
run makeSRequest
run makeARequest
check callOnEmergency
check verifyAccessForSRequest
check verifyAccessForARequest
```

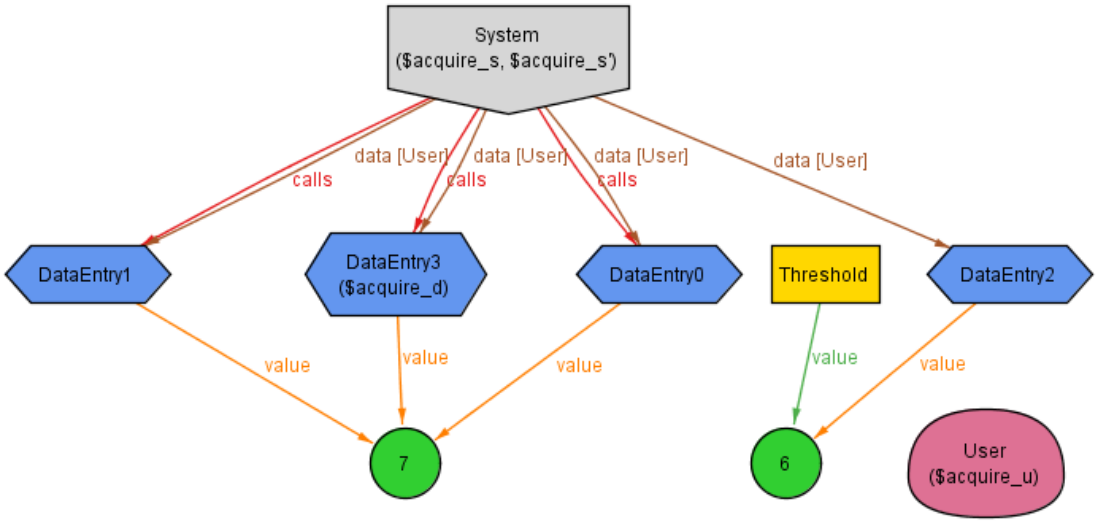


Figure 17: Acquisition of a new data entry that may trigger a SOS call

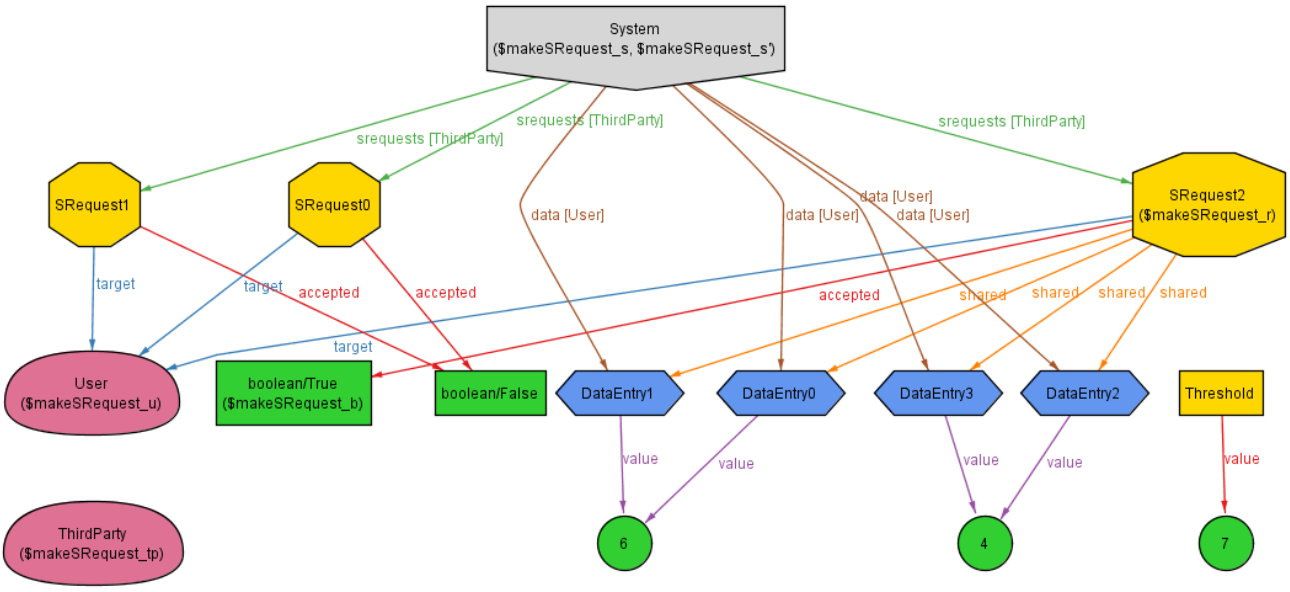


Figure 18: Single user requests that may be accepted or refused by users

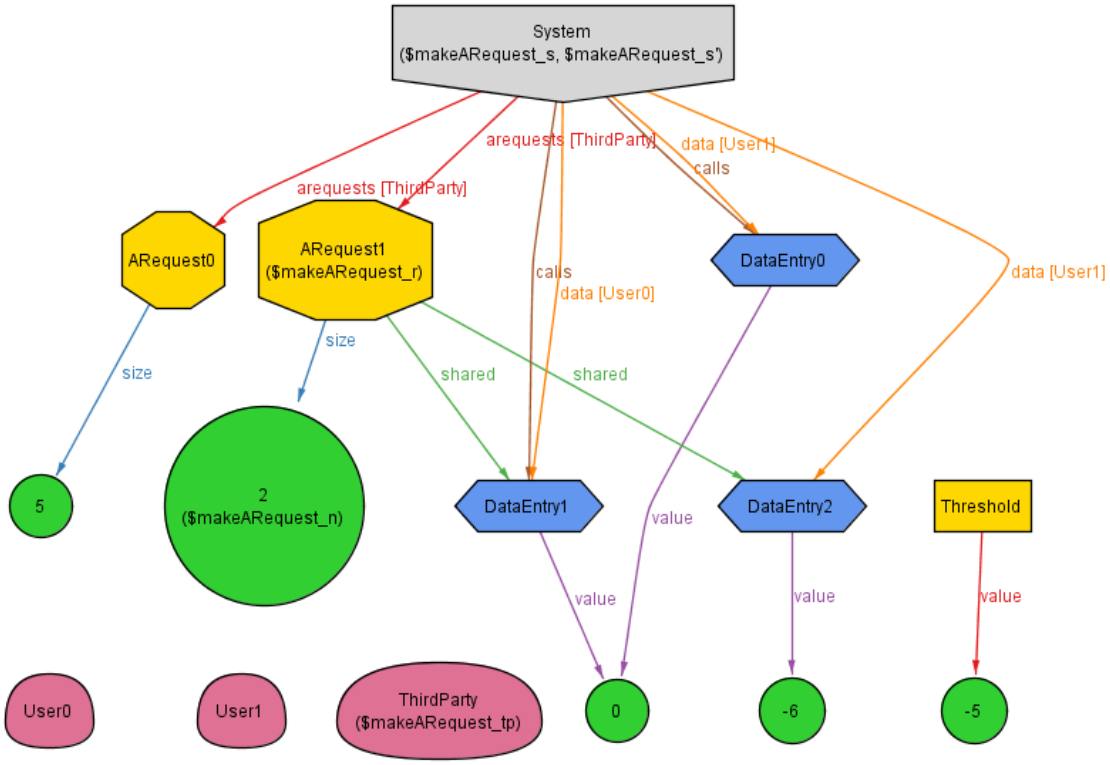


Figure 19: Anonymous group requests that may be accepted or refused by the system

```
Executing "Run acquire"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=2 Symmetry=20
   4820 vars. 265 primary vars. 12059 clauses. 25ms.
   Instance found. Predicate is consistent. 19ms.

Executing "Run makeSRequest"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=2 Symmetry=20
   4630 vars. 270 primary vars. 11325 clauses. 22ms.
   Instance found. Predicate is consistent. 23ms.

Executing "Run makeARequest"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=2 Symmetry=20
   4714 vars. 281 primary vars. 11547 clauses. 16ms.
   Instance found. Predicate is consistent. 27ms.

Executing "Check callOnEmergency"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=2 Symmetry=20
   4831 vars. 265 primary vars. 12148 clauses. 20ms.
   No counterexample found. Assertion may be valid. 8ms.

Executing "Check verifyAccessForSRequest"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=2 Symmetry=20
   4630 vars. 265 primary vars. 11372 clauses. 15ms.
   No counterexample found. Assertion may be valid. 4ms.

Executing "Check verifyAccessForARequest"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=2 Symmetry=20
   4981 vars. 281 primary vars. 12473 clauses. 16ms.
   No counterexample found. Assertion may be valid. 22ms.
```

Figure 20: Model validation

# 5 Requirement level UML diagrams

## 5.1 Class diagram

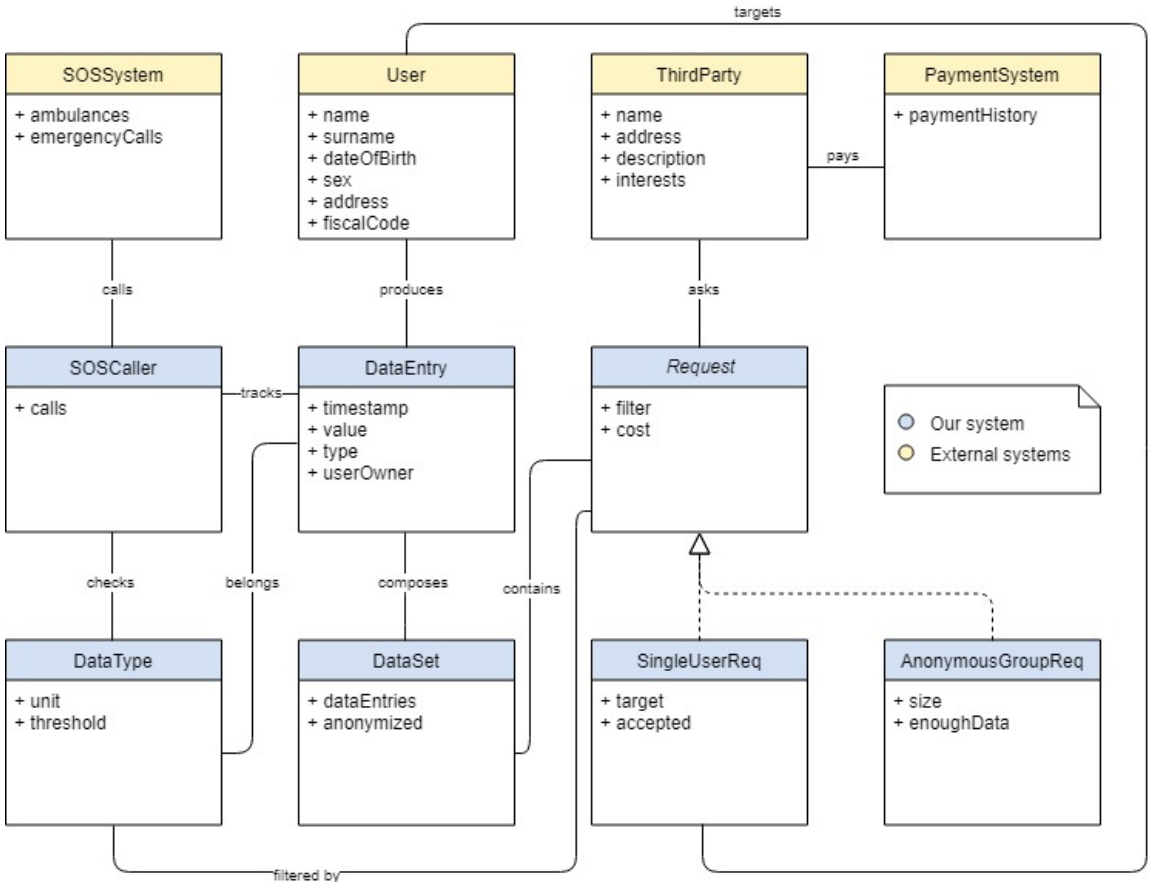


Figure 21: Shows the main machine-world interaction entities

## 5.2 State diagrams

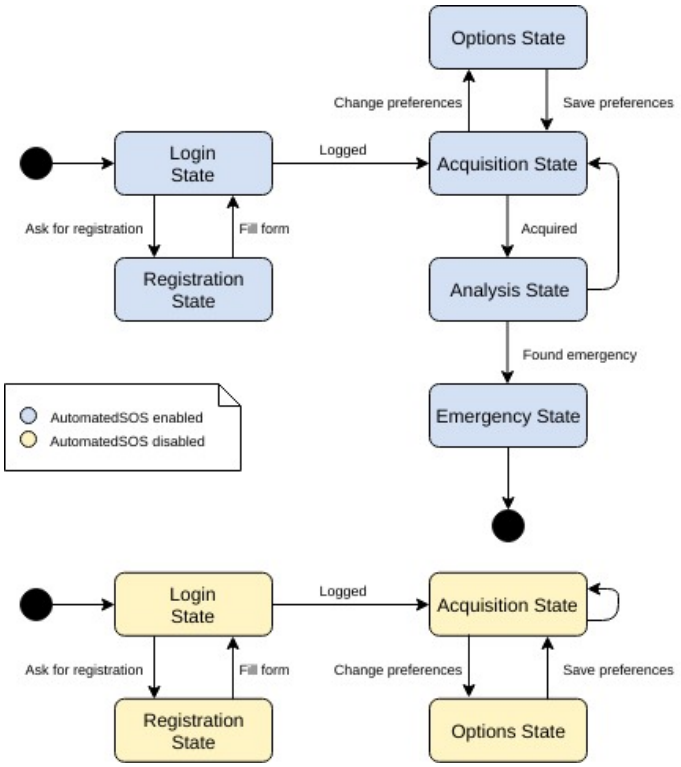


Figure 22: Shows the system state while logged in a user account; `AutomatedSOS` may be enabled

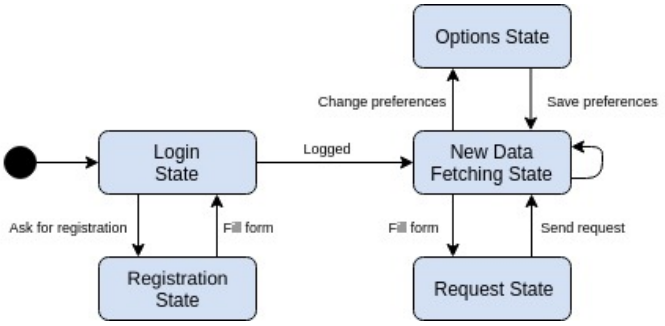


Figure 23: Shows the system state while logged in a third party account; new data may be produced by the system and forwarded to the third party

## 5.3 Use case diagrams

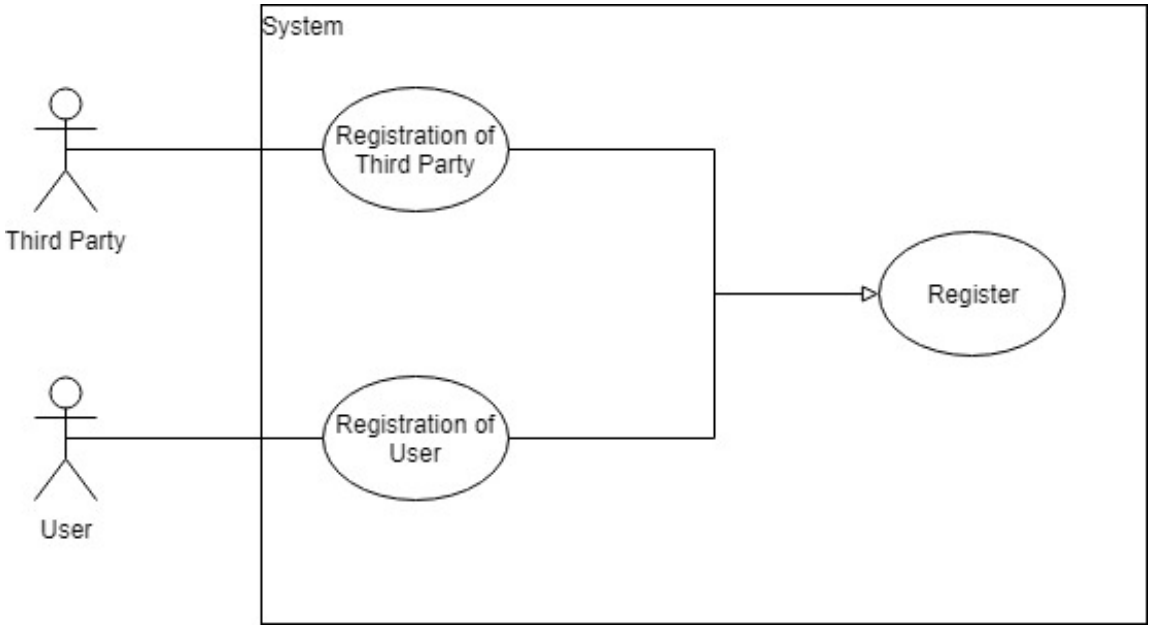


Figure 24: Shows registration use case

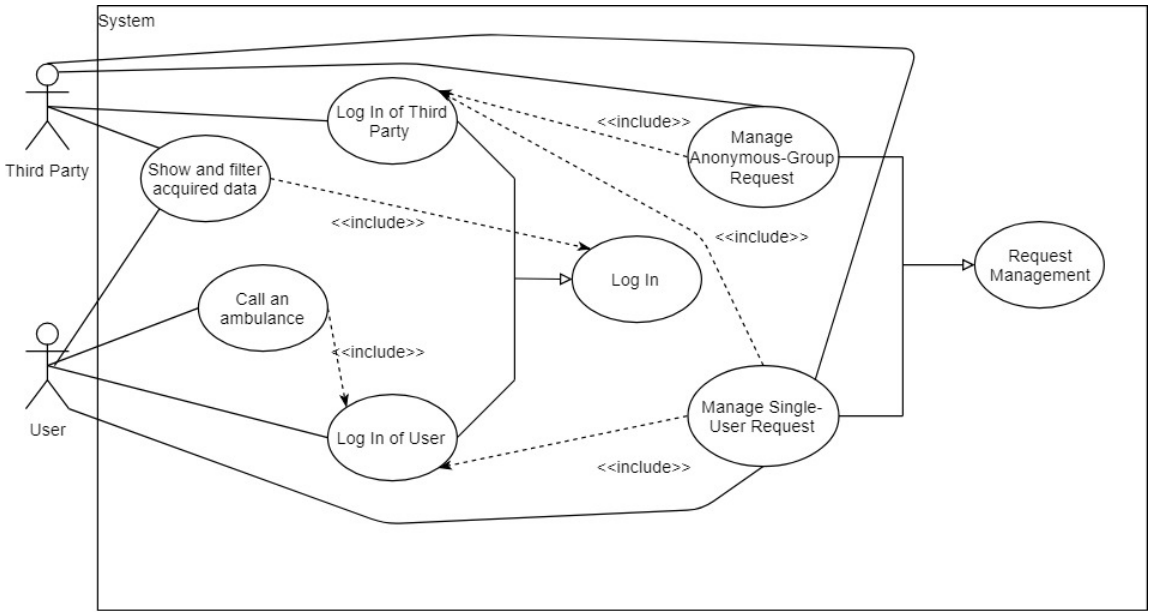


Figure 25: Shows application use cases after registration

## 5.4 Sequence diagrams

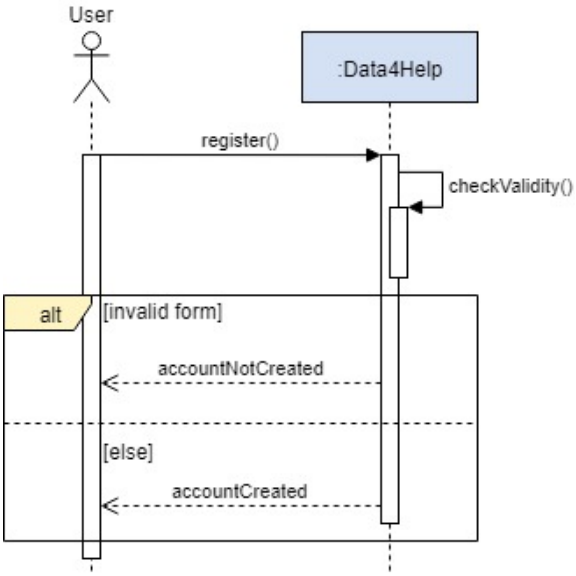


Figure 26: User registration

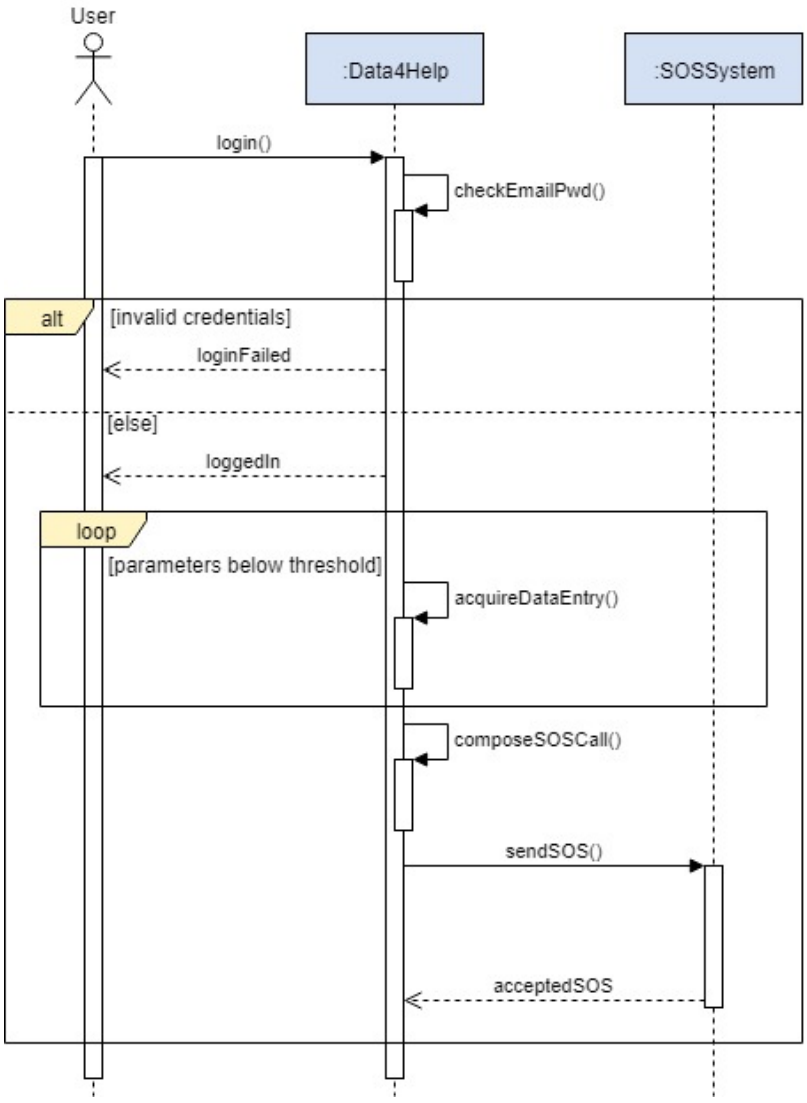


Figure 27: Shows the SOS calling procedure

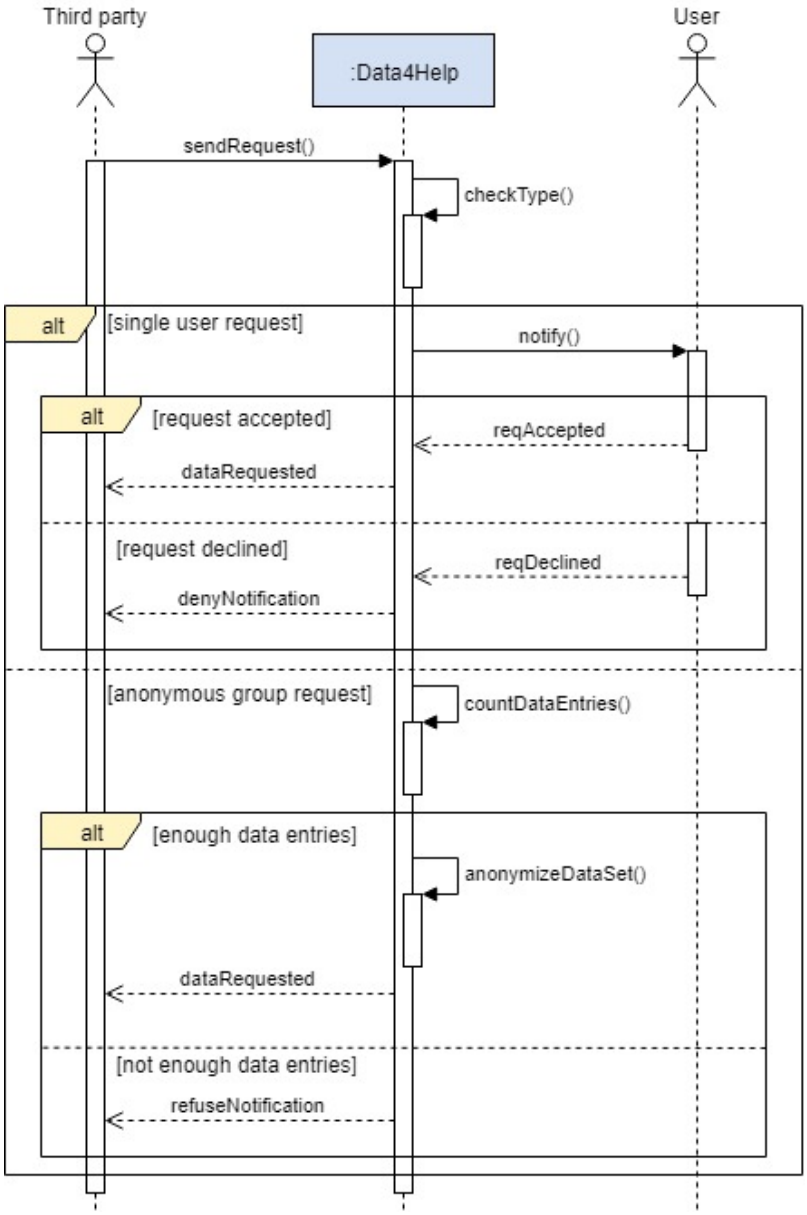


Figure 28: Shows the third party request procedure

# 6 Effort spent

| Date | Archetti Alberto | Carminati Fabio | Activity |
|---|---|---|---|
| 20/10/2018 | 3 | 2 | Introduction sketch |
| 21/10/2018 | 3 | 2 | Introduction update, requirement analysis, scenarios |
| 23/10/2018 | 3 | 1 | Goals and definitions |
| 23/10/2018 | | 3 | Scenarios and use cases |
| 26/10/2018 | 2 | | Goals revision, product perspective |
| 26/10/2018 | | 4 | Scenario |
| 27/10/2018 | 3 | | Product functions, dependencies and constraints |
| 27/10/2018 | | 3 | Interfaces and constraints |
| 28/10/2018 | 4 | 1 | Requirements, UML |
| 28/10/2018 | | 4 | Use cases |
| 29/10/2018 | 1 | 1 | UML |
| 31/10/2018 | 2 | | Interface mockup |
| 31/10/2018 | | 3 | Software system attributes |
| 1/11/2018 | 5 | 1 | Section 1, 2, 3 revision mockups |
| 2/11/2018 | 2 | 4 | Section 3 revision |
| 03/11/2018 | 3 | 4 | Section 3 revision, UML |
| 04/11/2018 | 2 | 1 | Alloy |
| 05/11/2018 | 3 | 3 | Alloy, section 2 and 3 revision |
| **Total** | 38 | 37 | |

# References


[1] Mandatory Project Assignment AY 2018-2019

[2] IEEE 830-1993 - IEEE Recommended Practice for Software Requirements Specifications

[3] ISO/IEC/IEEE 29148 - Systems and software engineering — Life cycle processes — Requirements engineering

[4] Collection and Processing of Data from Wrist Wearable Devices in Heterogeneous and Multiple-User Scenarios
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5038811/

[5] Wearable Devices in Medical Internet of Things: Scientific Research and Commercially Available Devices
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5334130/

[6] Google Fit API
https://developers.google.com/fit/overview

[7] PayPal API
https://developer.paypal.com/docs/

[8] RapidSOS Emergency API
https://info.rapidsos.com/blog/product-spotlight-rapidsos-emergency-api

[9] Slides of the course by Prof. Di Nitto
https://beep.metid.polimi.it/

[10] LaTeX templates
http://www.latextemplates.com/

[11] Alloy LaTeX Highlighting
https://github.com/Angtrim/alloy-latex-highlighting

[12] Draw.io
https://www.draw.io/