POLITECNICO
MILANO 1863

Data4Help

Lorenzo, Molteni, Negri

RASD Document

November 11, 2018

Contents

1 Introduction **3**
- 1.1 Purpose . . . 3
- 1.2 Scope . . . 5
  - 1.2.1 World Phenomena . . . 5
  - 1.2.2 Shared Phenomena . . . 5
  - 1.2.3 Machine Phenomena . . . 6
- 1.3 Definitions, Acronyms, Abbreviations . . . 6
  - 1.3.1 Acronyms . . . 6
  - 1.3.2 Definitions . . . 7
  - 1.3.3 Abbreviations . . . 8
- 1.4 Revision History . . . 8
- 1.5 Reference Documents . . . 8
- 1.6 Document Structure . . . 9

2 Overall Description **10**
- 2.1 Product perspective . . . 10
  - 2.1.1 Class Diagrams . . . 13
  - 2.1.2 State Charts . . . 16
- 2.2 Product functions . . . 17
- 2.3 User characteristics . . . 20
- 2.4 Assumptions, dependencies and constraints . . . 21
  - 2.4.1 Constraints . . . 21
  - 2.4.2 Dependencies . . . 21
  - 2.4.3 Domain Assumptions . . . 21

3 Specific Requirements **23**
- 3.1 External Interface Requirements . . . 23
  - 3.1.1 User Interfaces . . . 23
  - 3.1.2 Hardware Interfaces . . . 40
  - 3.1.3 Software Interfaces . . . 40
  - 3.1.4 Communication Interfaces . . . 40
- 3.2 Functional Requirements . . . 41
  - 3.2.1 Use Case Diagrams . . . 41
  - 3.2.2 Scenarios . . . 42
  - 3.2.3 Use Cases . . . 44
  - 3.2.4 Sequence Diagrams . . . 57
  - 3.2.5 Goal mapping on requirements . . . 67
- 3.3 Performance Requirements . . . 72
- 3.4 Design Constraints . . . 73
  - 3.4.1 Standards compliance . . . 73
  - 3.4.2 Hardware limitations . . . 73
- 3.5 Software System Attributes . . . 74
  - 3.5.1 Availability . . . 74
  - 3.5.2 Reliability . . . 74

3.5.3 Security 74
3.5.4 Maintainability 74
3.5.5 Portability 74

4 Formal analysis with Alloy **75**
4.1 Overview 75
4.2 Static Model 76
4.3 Dynamic Model 78

5 Efforts **85**

1 Introduction

The following RASD aims at providing an overview of the project *Data4Help*. This document will guide the reader in understanding the purpose of the project, on which grounds it is set and in which environment it will live. Precisely, it will illustrate both informally and formally which are the goals and how these may be reached by guaranteeing certain functional and nonfunctional requirements, while certain phenomena hold in the world. The document will provide a baseline for the project's planning and estimation and will support the software's evaluation. Finally, it will be the starting point for the software's review and update. The document is addressed to all those who intend to benefit from the service and who may be involved in its development.

1.1 Purpose

Data4Help is a service that aims at keeping track of individuals' health conditions. The application's goal is achieved by allowing individuals to share their position and health data and by enabling *Third Parties* to request and monitor them. *Single User*s will enjoy a clear summary of their health status and will feel safe by being continuously monitored by *Third Parties*. *Third Parties*, on the other hand, will have direct access to users' data and will be able to use them to perform analysis for medical purposes.

Data4Help requires the user to create an account to access its services, the functionalities unlocked after registration depend on the type of account created.
If a user creates an account as a "*Single User*", he/she will be asked to provide some basic information, precisely name, last name and date of birth. Furthermore, he must provide an email with which he will be uniquely identified, a password and a fiscal code. When all the registration requirements are fulfilled, a user will have activated his personal account. However, he will only enjoy of the application's service once he allows access to his health data and location, imported from his devices. The user can finally visualize a summary of his health conditions and reject or accept *Third Party* requests asking for his data. The application also allows users to keep track of how many and which *Third Parties* are monitoring him.

On the other hand, a company will create a "*Third Party*" account and will need to provide its name, its P.IVA, a password and, as for private users, will be uniquely identified by an email.
Third Parties can use the application to collect data in two ways. Firstly, they can retrieve data of a specific individual, identified by a fiscal code or an email, provided that the user responds positively to the request. The *System* also allows *Third Parties* to subscribe to new data and receive it as soon as it is produced.
Secondarily, *Third Parties* may also retrieve data concerning groups of users

by performing group requests, provided that the filtered result satisfies certain constraints. These requests won't require the involved users' approval and will provide anonymous results.

In addition to these functionalities, *Data4Help* allows *Single User*s to activate *AutomatedSOS*. This integrated service adds up as another feature preserving the users' safety. Whenever a user's health status results critical according to certain thresholds, if *AutomatedSOS* is enabled an ambulance is immediately called and sent to his location.

Data4Help offers to its customers also an additional service: *Track4Run*. While relying on the solid architecture of *Data4Help*, this feature can be accessed by downloading its exclusive application. When a user downloads *Track4Run* he will login with his *Data4Help* account or will easily register to it at the moment. Those who will access *Track4Run* are runners, run organizers and *Spectator*s. This application is intended to enable *Third Parties* to organize runs, *Single User*s to join them and *Spectator*s to keep track of runners live during an active path. While runners will enjoy visualizing their position live in the path, organizers will be able to track all participants and, thanks to *Data4Help*'s underlying services, will access the runners' health data. *Spectator*s won't need to create any account as they will simply scroll through the list of active runs and track the runners' positions in the selected path.

From this brief description of the functionalities we may extract the following goals for *Data4Help*:

- **[G1]** Users can be identified either as *Single Users* or as *Third Parties*.
- **[G2]** *Single Users* can have their data monitored by *Third Parties* for medical purposes.
- **[G3]** *Single Users* can visualize a summary of their health status.
- **[G4]** *Third Parties* can access data of those *Single User*s who granted their permission.
- **[G5]** *Third Parties* can access data of anonymous groups of users.

With regards to *AutomatedSOS*, we may identify one goal:

- **[G6]** Whenever a user's health status becomes critical, an ambulance is sent to his location.

Finally, for *Track4Run*, we extract the following goals:

- **[G7]** Organizers can create and manage runs.
- **[G8]** Runners can enroll in an existing run.

- **[G9]** *Spectator*s can track the position of runners during a run.

1.2 Scope

Following the definition originally proposed by M. Jackson and P. Zave in 1995, we will try to clarify the scope and the application range of the *System* by identifying the so called World and Machine phenomena.
The Machine is the *System* to be developed, in this case a software, while the World, also known as environment, corresponds to the part of the real world that is affected by the *System*.
The World and the Machine are associated with different types of events which we are going to describe and detail. World and the Machine are associated with different types of events which we are going to describe and detail.

1.2.1 World Phenomena

World phenomena are events that take place in the real world and taken by themselves do not have a direct impact on the *System*.
We identify the following world phenomena:

- Heart rate of a *Single User* drops due to a stroke.
- A user has an accident.
- A user is in an extreme physical condition.
- A user changes his location.
- A user has a regular heart rate.
- A user performs some kind of physical activity.
- A user takes part in a run.

1.2.2 Shared Phenomena

Shared phenomena are world phenomena that are shared with the Machine.
These are further divided in two categories: phenomena controlled by the World and observed by the Machine and phenomena controlled by the Machine and observed by the World.
With regard to the first kind we identify the followings:

- A user inputs to the *System* all data needed for registration.
- A user's health data is collected into the *System*.
- A user sends a data request to the *System*.
- A user enables *AutomatedSOS*.

- A user joins an active run.
- A user checks-in a run he previously joined.
- A user organizes a run.

Concerning the second type:

- A user visualizes his health status.
- The *System* shows the result of a request.
- The *System* forwards a request notification to a user.
- The *System* asks a runner to share his data.
- The *System* shows to all *Spectator*s a user's position during a run.
- The *System* shows data of participants of a run to its organizer.

1.2.3 Machine Phenomena

Machine phenomena are events that entirely take place inside the *System* and cannot be observed in the real world.
We identified:

- Retrieving result data for a request.
- Storing permanently collected data.
- Encryption of sensitive data.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Acronyms

S2B	*System* to be
GDPR	General Data Protection Regulation
GPS	Global Positioning *System*
API	Application Programming Interface
FC	Fiscal Code
P.IVA	Partita IVA
BPM	Beats Per Minute
Kcal	Kilocalories
UI	User Interface
WP	World Phenomena
MP	Machine Phenomena
ID	Identifier

Table 1: Acronyms

1.3.2 Definitions

- **S2B:** the *System* to be developed. It includes both *Data4Help* and *Track4Run*.
- **Historical Data:** all data provided to the S2B by the user since his registration.
- **Active Energy:** Energy consumed by the user when not standing still, measured in Kcal.
- **Resting Energy:** Energy consumed by the user when standing still or resting, measured in Kcal.
- **Health parameter:** health feature imported from the user's device, namely weight, height, heart rate, stand hours, active and resting energy, sleeping hours, walking and running distance.
- **Health status:** the set of all health parameters, modelling the health of the user at a given time.
- **Group request:** how a *Third Party* interacts with the S2B to obtain anonymized data from a group of users.
- **Single request:** how a *Third Party* interacts with the S2B to obtain data from a *Single User*.
- **Search parameters:** parameters used to filter users in a group request, such as age, location (address or selected area), values of users' health parameters.
- **Requested data types:** health parameters that *Third Parties* want to obtain from users when submitting a request.
- **Sensitive Data:** information that is not provided to *Third Parties* when answering a multiple request to guarantee anonymity. This includes position, FC and email.
- **Time series:** cartesian plot of the evolution of a health parameter in time. X axis is a time scale, Y axis is the value of the parameter.
- **Statistical operators:** aggregated statistical operators such as Mean, Median, Percentiles, Variance, Standard Deviation and Error.
- **Fiscal code:** a 16 characters code used in Italy to uniquely identify a citizen.
- **Partita IVA:** a 11 characters code used in Italy to uniquely identify a company for value added tax purposes (VATIN in Europe).
- **Credentials:** the email and password provided during registration.

- **Joining a run:** expressing the non-binding intention to take part in a run.
- **Check-in:** binding confirmation that a user will take part in a run, agreeing to share his data

1.3.3 Abbreviations

- **(Gn):** n-th Goal
- **(Rn):** n-th Requirement
- **(Dn):** n-th Domain Assumption
- **(Cn):** n-th Constraint
- **SP1:** shared phenomena controlled by the World and observed by the Machine
- **SP2:** shared phenomena controlled by the Machine and observed by the World
- **WP:** World Phenomena
- **MP:** Machine Phenomena

1.4 Revision History

Version	Date	Changes
1.1	24/10/2018	Initial draft
1.2	10/11/2018	First deadline draft

Table 2: Revision History

1.5 Reference Documents

- ISO/IEC/IEEE 29148: https://www.iso.org/standard/45171.html
- Alloy Official Documentation: http://alloy.lcs.mit.edu/alloy/documentation.html
- Four Dark Corners of Requirements Engineering – Jackson, Zave
- Project Assignment

1.6 Document Structure

The rest of the document is organized as follows:

- **Overall Description** (Section 2) contains an in-depth description of the domain under analysis, following the Jackson-Zave approach used in the Scope paragraph in Section 1. Class Diagrams will be hereby provided to formalize all the actors involved in the *System* and their relations, while their needs and scope as Stakeholders will be analyzed later in the section. Key functional requirements will then be listed, together with all the Domain Assumptions that can be used to entail each Goal.

- In the **Specific Requirements** (Section 3) functional requirements are associated to use cases and Sequence Diagrams to clarify processes and interactions between users and the S2B. Nonfunctional requirements are also included here, together with external constraints (mainly regulations) imposed on the *System*.
  Developers can also find requirements regarding External Interfaces, with a focus on Hardware and Software Interfaces. Finally, a brief overview of the properties that the S2B will have to guarantee is provided.

- **Formal Analysis** (Section 4) uses Alloy to generate a formal model of two distinct parts of *Data4Help*, the identification and requests.

2 Overall Description

2.1 Product perspective

In this section we will explain in more detail the various World, Machine and shared phenomena. We will also provide some of the descriptions with State and Class Diagrams.

Heart rate of a Single User drops due to a stroke (WP)
In the unlikely situation in which the user is subject of a severe heart attack, his heart rate will suddenly drop under the minimum threshold. The same applies for any other medical condition that can arise such as low heart rate due to low blood pressure.

A user has an accident (WP)
The user may get involved in some kind of accident, for example getting in a car crash or being wounded as a result of a work incident. In such cases the user's biological parameters may vary indicating a critical health condition.

A user is in an extreme physical condition (WP)
If the user is performing extremely demanding physical tasks that are exceeding his body capabilities, such as running too fast or for too long, ascending a steep cliff or lifting heavy loads, his heart rate may rise too much and put him in danger.

A user changes his location (WP)
During the course of his daily routine the user usually moves between different areas and locations modifying his current position.

A user has a regular heart rate (WP)
In conditions of healthiness and well being, while no particular physical tasks are performed, the heart rate of the user normally lies in the safe interval indicated for is age range.

A user takes part in a run (WP)
A user that enjoys running or jogging may decide to participate to an organized run. An organized run usually takes place in a specific location at a given time and could allow the enrollment of both amateurs and professionals.

A user inputs to the System all data needed for registration (SP1)
During the registration phase the user must provide some mandatory informations such as email, password, FC and birthdate. These infos are sent through the *System* which takes care of validating and storing them in a new account.

A user's health data is collected into the System (SP1)
Granting access to health parameters and location is an essential step towards

the exploitation of the *System*'s potentials. This data is collected by the machine, associated with the user of origin, and stored into a database.

A user sends a group request to the System (SP1)
A *Third Party* can send a group request to the *System* in order to retrieve anonymized data about a group of users, filtered with search parameters. The group request is provided as input through the user interface and processed internally.

A user enables AutomatedSOS (SP1)
At a certain point in time a *Single User* may decide to enable *AutomatedSOS*, which is an additional feature offered by *Data4Help*. In order to do so, he has to press a button on the user interface. The *System* registers the input and enables *AutomatedSOS* functionality.

A user joins an active run (SP1)
A user who wishes to participate to a run organized by *Track4Run* can either find it by inputting the run's ID or by looking for it in the list of active runs on the *System* UI. He then press a button and sign up for the event. The *System* is alerted and adds that user to the list of participants.

A user checks-in a run he previously joined (SP1) When the time of an organized run comes, all participants confirm their presence by giving permissions to share their health and position data with the *System*. After a user grants this access, the machine starts collecting and processing its data.

A user organizes a run (SP1)
The user can organize a run by selecting the relative view on the UI and filling all the required fields. All information is received by the *System* who proceeds to create a new active run visible to all other users.

The System shows to the user his health status (SP2)
The *System* retrieves both current and historical health data related to the user and proceeds to show them in an organized and aggregated way on the user interface.

The System shows the result of a group request (SP2)
After processing a group request, if some results have been obtained from the DBMS, the *System* shows the anonymized data to the requesting party in a neat, organized and aggregated way.

The System forwards a request notification to a user (SP2)
After receiving a request from a *Third Party* the *System* checks if the user specified in the parameters exists and if so proceeds to forward a notification request to him. The notification includes the generalities of the requesting *Third Party*, the type of request (subscription or not) and the kinds of health parameters

asked.

The System asks a runner to share his data (SP2) When the time of a run comes, the *System* sends a notification to all users asking for their permission to share Position and Health data.

The System shows to all *Spectator*s a user's position during a run (SP2)
During a run the *System* constantly update its internal state with the position of all participants using the data they share. This information is then used to show to *Spectator*s where each runner is located using a map on the UI.

The System shows data of participants of a run to its organizer (SP2)
The *System* constantly shows to an organizer data about the participants enrolled in a run he created. Said data includes the number of runners, their generalities and after the run starts their position and health data.

Retrieving result data for a group request (MP)
When a group request is received, the *System* extracts the filtering parameters and forwards the query to its database. The DBMS extracts all data that satisfies the query. If the results match less than 1000 accounts they are discarded and a failure notification is generated. Otherwise the result is sent to the requesting *Third Party*. Furthermore the data is anonymized, which means that all sensible information related to the respective users is hidden.

Storing permanently collected data (MP)
All data collected from registered users is sent to an internal database. The *System* makes sure that all this data is not lost or corrupted in any way by using the functionalities provided by a DBMS.

Encryption of sensitive data (MP)
All sensitive data are encrypted by the *System* in order to guarantee the privacy of the users in case of malicious attacks or data leaks. Privacy is a big issue in this project because the Machine deals with personal data.

2.1.1 Class Diagrams

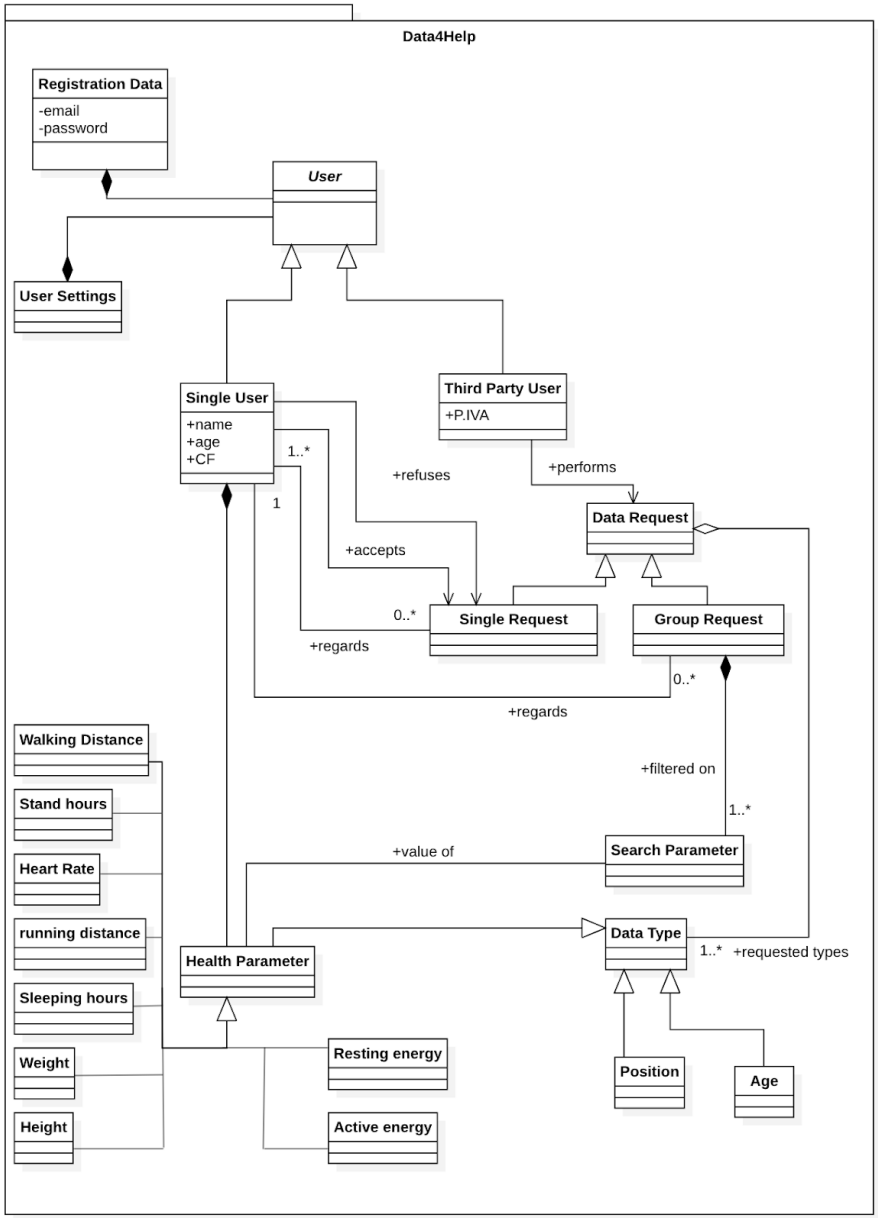


Figure 1: *Data4Help* Class Diagram

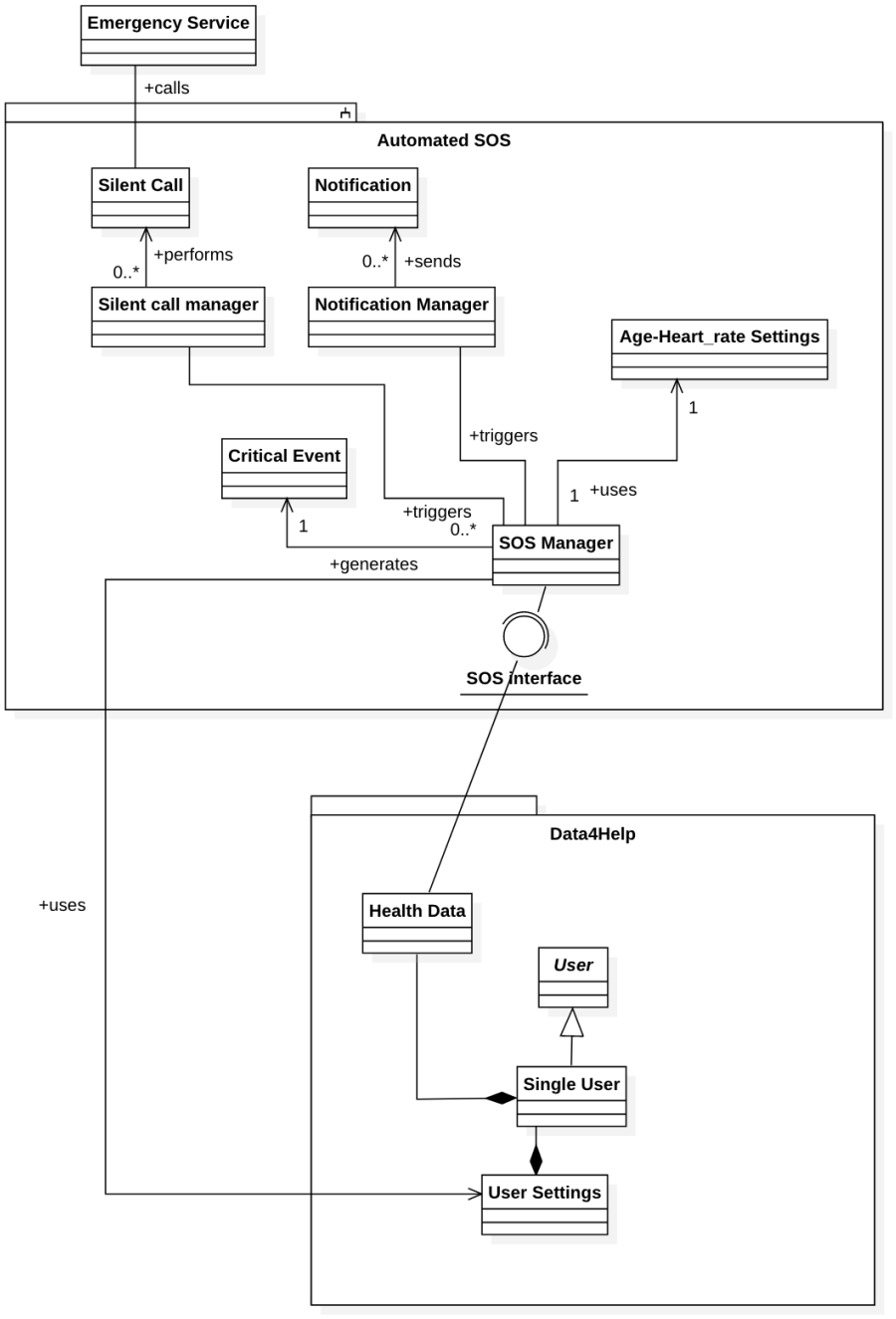


Figure 2: *AutomatedSOS* Class Diagram

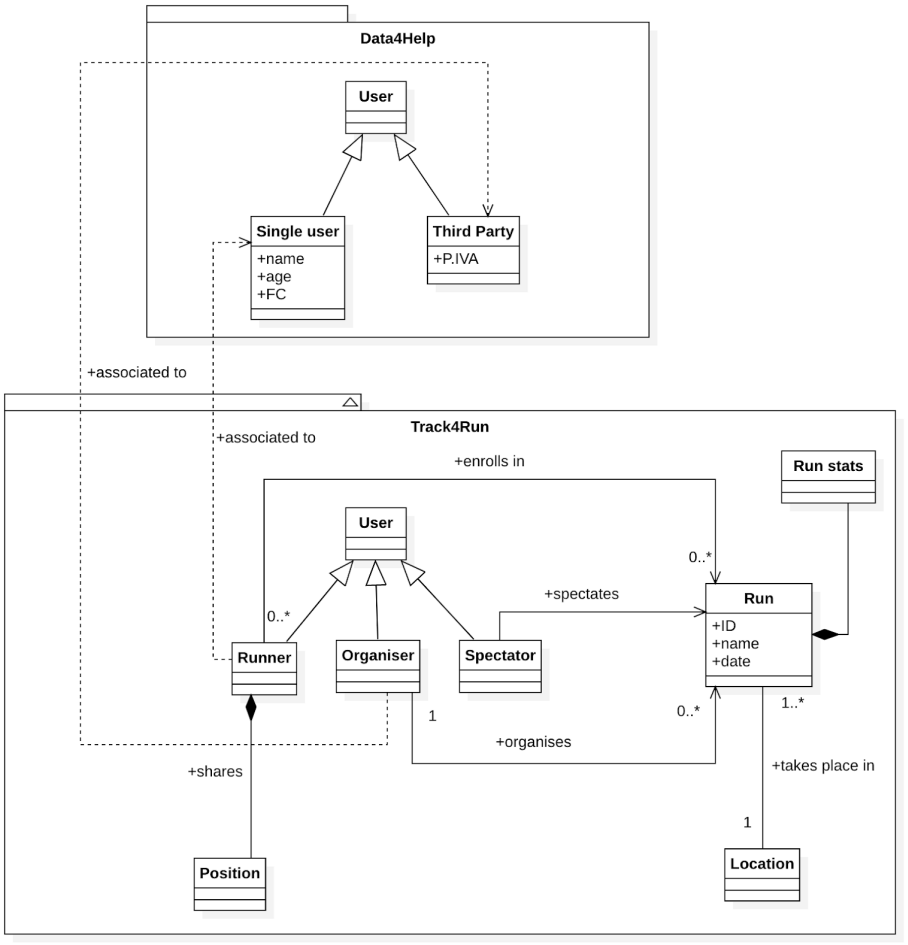


Figure 3: *Track4Run* Class Diagram

2.1.2 State Charts

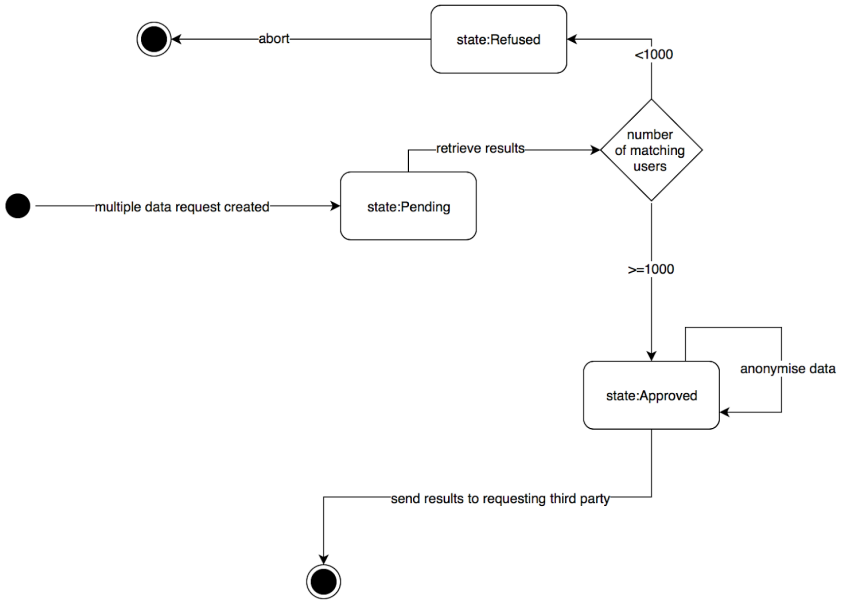


Figure 4: Group request state chart

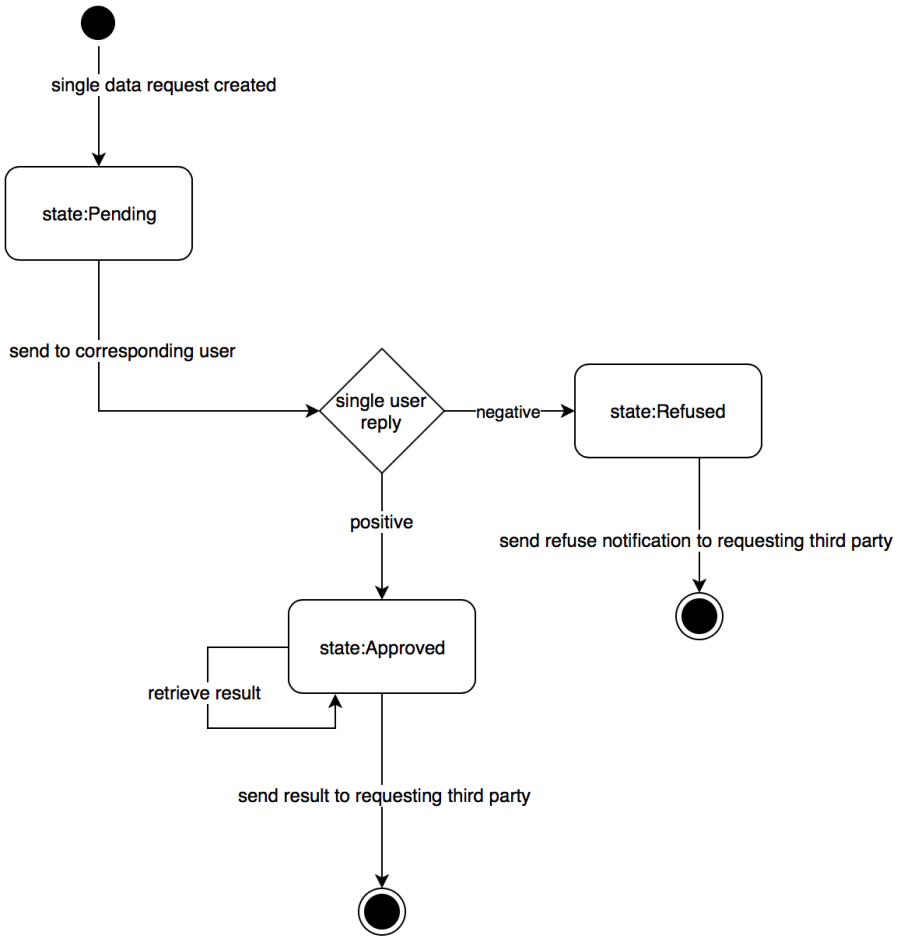


Figure 5: Single request state chart

2.2 Product functions

This section is devoted to analyzing the Value Proposition that *Data4Help* and *Track4Run* provide to its users. The presented functionalities are grouped according to the user they address.

- **Private User functions:**

  Visualize your own health status
  Data4Help enables *Single User*s to visualize their health status. This is done by retrieving all information about the user's position and health from his device, by analyzing it and by resulting into a graphical explana-

tion. Analysis performed on data consists of aggregations, statistical operations and derivations. Results are finally provided to the user through a clear interface containing significant plots, tables and charts.
Users will then refer to *Data4Help* to understand if they are either healthy for their age or might have some values out of range.

Have your health monitored
Data4Help enables *Single User*s to have their health monitored by *Third Parties* by accepting their requests and allowing them to subscribe. This feature is useful especially for people having a medical condition, so that they can be remotely monitored by their doctor. Elderly people could benefit from it as well.

AutomatedSOS
AutomatedSOS is an additional feature that, when enabled, can provide a significant help to individuals in critical situations. In fact, when the *System* observes some health parameters below certain thresholds an ambulance is immediately called to the user's location. This functionality may be activated by any *Single User* and will function only in case of working GPS.

Take part in a run
Track4Run gives runners, using a private account, the possibility to participate to an existing run that has not started yet. Runners will provide their data to the organizers of the run so they can be tracked on the path, allowing the *System* to measure key statistics such as average pace, heart rate and energy spent (if provided). This data will be shown to each runner at the end of the event, comparing him to other participants.

- **Third Party functions:**

Request individual's health data
A reason why *Third Parties* decide to register to *Data4Help* is to retrieve data about individuals, precisely concerning their position and their health status. *Data4Help* aims at allowing these companies to collect their desidered data for medical purposes.
The retrieval of data is possible by performing requests to *Single User*s, denoted with either their FC or their email. Requests contain the *Third Parties*' generalities and specify whether they require the user's data only once or want to subscribe to it, meaning that they will receive updates on the data whenever they are produced. Subscriptions must indicate a duration and can be ended by the *Single User* at any time. Requests may be customized by including a list of desired data types, chosen among the following:

- position
- heart rate
- activities (outdoor or indoor)
- nutrition
- sleep

Request are sent directly to the involved users and require their acceptance to proceed with the forwarding of data. Therefore a *Single User* can decide either to accept or reject requests of any type.

Perform group requests on anonymized data
Another feature offered by the *System* to *Third Parties* is the possibility to perform group requests and retrieve anonymous data. The requests are received by the *System* and performed onto its centralized database. Researches of this type have the following features:

- Must concern a number of individuals greater than 1000
- Do not include sensible data
- Include search parameters
- Include requested data types
- Do not need to be accepted by users

If the request is valid according to the stated conditions, the *System* will retrieve all data matching the search parameters and present it to the requesting user. *Single User*s whose data is part of the query result are not involved. Therefore they neither are mentioned, in order to preserve their anonymity, nor are asked to accept or reject the request.

Organize new runs
Track4Run gives organizers, using a *Third Party* account, the possibility to create a run and get data from participants. When creating a run, organizers have to define the path that the run will follow on a map, set the starting time and ending time of the event and the maximum number of participants. When a run is full no more runners can join it.
Throughout the event, organizers have access to live data of all participants, and can save that data after the event has finished.

- *Spectator* **functions:**

Spectate a live run
On the day of a run, any user can access *Track4Run* without creating any

account, select a run from the list of active runs and follow it live. *Spectator*s can visualize the position of the users and, at the end of the run, of the final statistics of every runner.
During the run, *Spectator*s have the option to select which runner they want to track, filtering using email and FC.

2.3 User characteristics

Data4Help addresses two types of users: *Single User*s and *Third Parties*. The first type corresponds to an individual of any age that downloads the application onto his device and allows access to his personal health data. In particular we have three main customer segments: elderly people, people with a medical condition and people who want to keep track of their health status.
The second refers to any company that provides a valid P.IVA to the *System*, and wants to obtain user data for medical purposes.

Track4Run's main targets are runners and organizers.
Runners, in particular, generally range from 15 to 60 years old people, of all social categories.
Organizers, on the other hand, must have a P.IVA, so they generally represent sport clubs or any company with medical interests.
When signing-up in the application, runners are forced to create a private account on *Data4Help*, which shares the user database with *Track4Run*, while organizers are forced to create a *Third Party* account.
Finally, *Spectator*s use the application with the sole purpose of spectating an existing run, and do not have to create an account on *Data4Help*.

We will now briefly precise which types of users the *System* encounters at different stages:

- *Guest:* a user who has downloaded the application onto his device, but has yet to create an account. No *Guest* can access the application's functionalities prior to registration.
- *Single User:* a user who has created a "*Single User*" account, but has yet to log in. Once he has inserted his credentials he may access all of the application's functionalities.
- *Third Party:* a user who has created a "*Third Party*" account, but has yet to log in. Once he has inserted his credentials he may access all of the application's functionalities.
- *Logged-in Single User:* a user that has provided the credentials to his *Single User* account and that may access all application functionalities.
- *Logged-in Third Party:* a user that has provided the credentials to his *Third Party* account and that may access all application functionalities.

- *Spectator:* a user that has not logged-in into *Track4Run* but is spectating an existing run.

2.4 Assumptions, dependencies and constraints

2.4.1 Constraints

- The S2B must abide to the GDPR regulation with regards to the treatment of personal data.
- The S2B must be implemented as a smartphone application.
- The S2B must be available in the Italian application stores only (based on the OS).

2.4.2 Dependencies

- The S2B will make use of the GPS services offered by users' smartphones.
- The S2B will access the APIs provided by health monitoring applications integrated by default in the smartphones.
- The S2B will access the necessary APIs to make phone calls on behalf of the user.
- The S2B will make use of internet access provided by users' smartphones.
- The S2B will make use of a map visualization service.
- The S2B will make use of a *Third Party* storage *System*.

2.4.3 Domain Assumptions

- [**D1**] Each user will create only one account.
- [**D2**] The FC and email provided during registration are valid and belong to the person who's creating the account..
- [**D3**] The P.IVA of a *Third Party* is valid and specific to that company.
- [**D4**] Data provided to the *System* is related to the person whose account was used to provide it.
- [**D5**] All health data directly provided by the user represents his real health status.
- [**D6**] Position data has an accuracy of 10 meters around the actual position.
- [**D7**] Health data has a relative error lower than 5%.

- [**D8**] Permission to access health and GPS data is always granted to the S2B.
- [**D9**] *Third Parties*, including organizers, collect data only for medical purposes.
- [**D10**] Data is stored on persistent memory.
- [**D11**] The storage *System* is fully replicated and fault-tolerant, so that a copy of a specific piece of data is always available.
- [**D12**] Permission to make calls is always granted to the S2B.
- [**D13**] Users' smartphones always have signal when needed by *AutomatedSOS*.
- [**D14**] Users' smartphones always have internet access when the S2B needs it.
- [**D15**] The path specified when creating a run is feasible.

[**D10**] and [**D11**] are crucial since the S2B will make use of a *Third Party*'s Database, and thus can't enforce its reliability.
[**D12**] and [**D13**] are needed to ensure the nonfunctional requirements of *AutomatedSOS*, otherwise the S2B should keep trying to reach an emergency number until it's successful.
[**D14**] is a strong assumption, generally the mobile application could keep pinging the server until the connection is established again.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

- **DATA4HELP INTERFACES**
  - **Logo**
    Data4Help's Logo represents a heart containing the sketch of a pulse. This simple image fully captures the purpose of the service: to collect data concerning individuals' health and allowing it to be monitored. The logo is also used as the application's icon and will be visible to whoever downloads the app onto his device.

Figure 6: *Data4Help* Logo

  - **Login or Signup**
    When a *Guest* downloads *Data4Help* for the first time, the first interface will ask the user to choose between login and signup. If the user has already created an account he will choose to login in order to access the application's functionalities. If the user instead has not created an account yet he will choose the signup option.

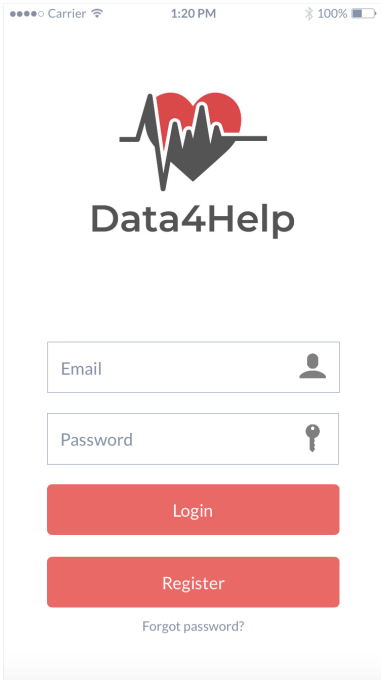


Figure 7: Login interface

– **Tutorial**
When a *Guest* decides to create a new account *Data4Help* provides him with a simple guide to the service. The user will easily swipe from one page to another and follow the tutorial on all the features it can access after registration.

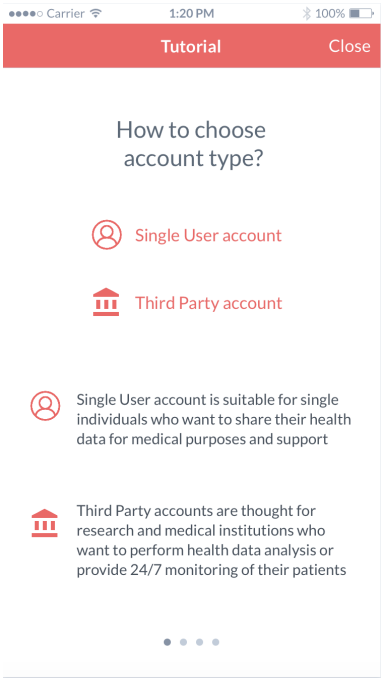
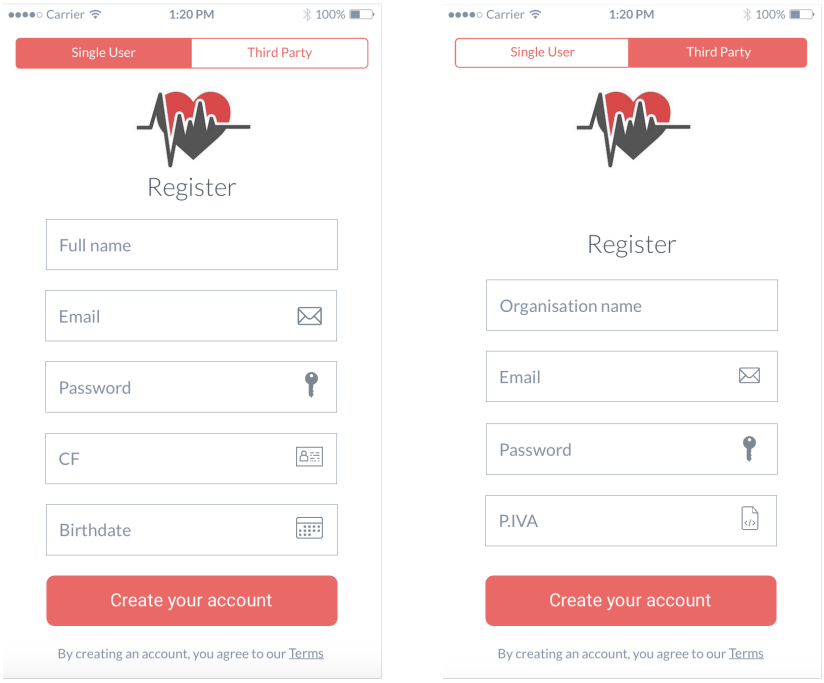


Figure 8: Tutorial interface

– **Create an account**
Following the initial guide a *Guest* will be able to create an account and register to the service. The user will be asked to select which type of account to create: *Single User* or *Third Party*. Once the type has been selected, the user will be asked to fill in a form with his general information, an email that will identify him uniquely and a password. Furthermore, according to the type of account, a user will have to provide a unique FC or P.IVA if creating respectively a *Single User* account or *Third Party* account.



(a) *Single User* registration interface (b) *Third Party* registration interface

Figure 9: Create an account

– **Single User Interface**
When a user signs in with a *Single User* account he is presented with the first of a sequence of interfaces, all handled by a Tab Bar Controller. We will now detail the features of each tab:

  ◦ **My Health tab**
  My Health tab is the default tab, the first interface presented to the *Single User*. It displays a summary of the user's health, measured thanks to the data imported from the device onto which the app is downloaded. This is the interface that a user will access in order to monitor his health status and have a clear view of how his parameters compare to the average for his age. Furthermore, plots, tables and charts will guarantee a more user friendly experience.
  Finally in this section the user will be able to activate at anytime *AutomatedSOS*. A simple click on the SOS icon will enable this feature and immediately contact an ambulance in case of critical health status.

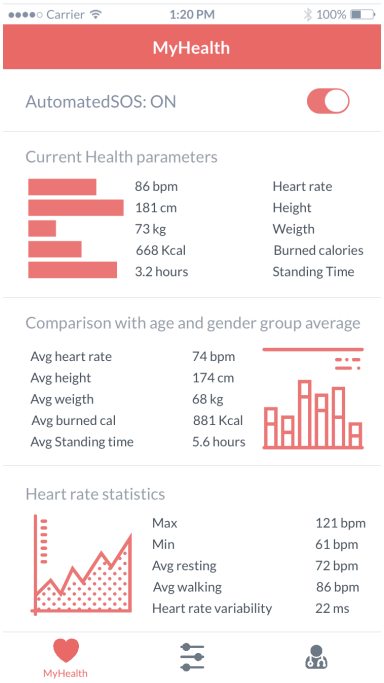


Figure 10: My Health tab

- **My Settings tab**
  My settings is the tab that allows a user to manage his personal data associated to his account. In this section a user may at any time modify his general information and password. However a user can never change the email and CF associated to its account as they are its unique identifiers.
  In this screen the user will find all the information linked to its account in a list, where each item can be modified at any time. After any modification the user will have to save its updates in order to notify the application's server.

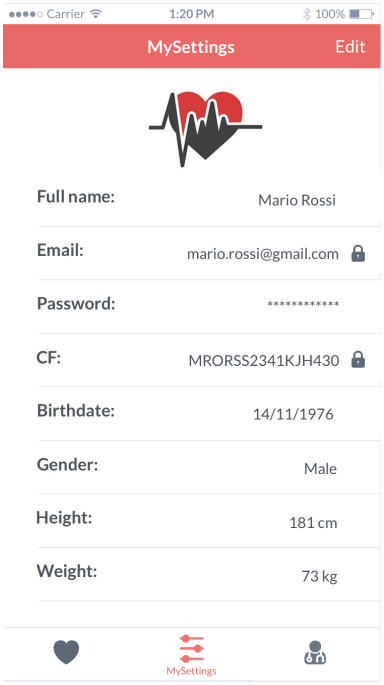


Figure 11: My Settings tab

- **My Followers tab**
  My followers is the tab that allows a user to keep track of all the *Third Parties* requesting his data. These include both those *Third Parties* that have performed single requests and those that have subscribed to the user's data. Furthermore in this section the user will also be able to accept or reject pending requests.

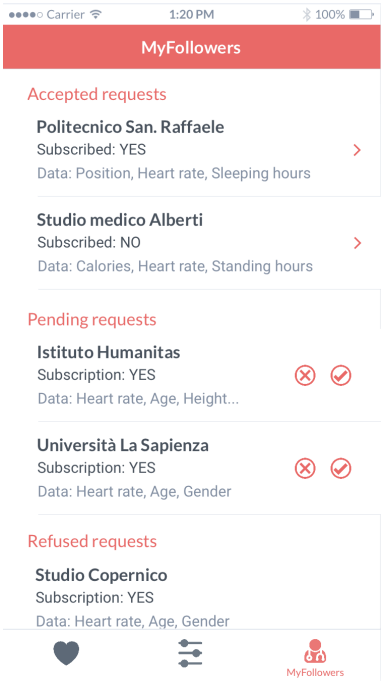
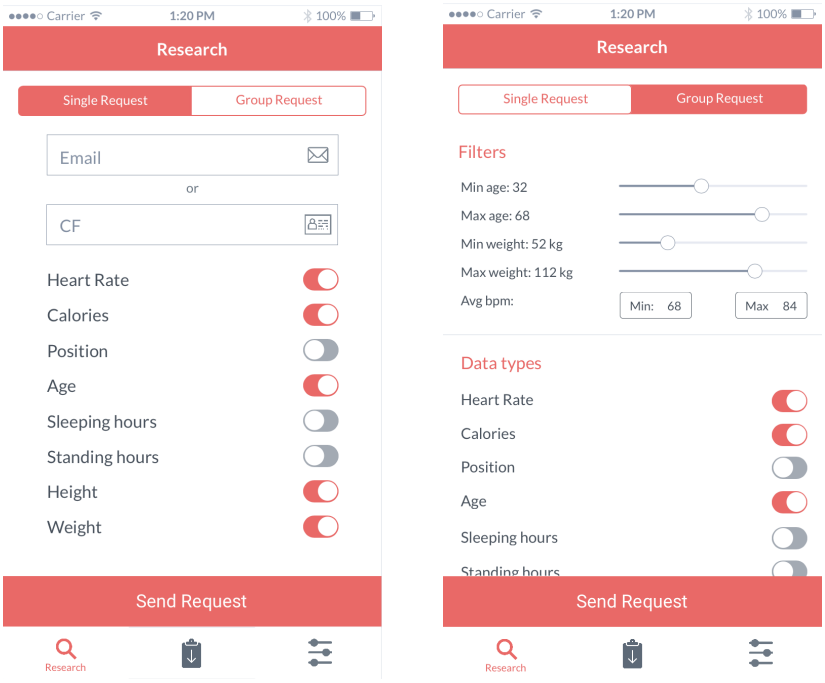


Figure 12: My Followers tab

– **Third Party Interface**

  ◦ **Research tab**
  The research tab is the default tab, the first interface presented to the *Third Party*. This is the section in which the *Third Party* can perform requests to individuals or on groups of users. The interface suggests the user to select one of the two types of requests. According to the selected type, the *Third Party* will have to fill in a form providing all the necessary parameters to perform the research.



(a) Single Request (b) Group Request

Figure 13: Research tab

- **History tab**
  In this tab the *Third Party* will be able to save all the searches it has performed on individuals or anonymous groups of users. The screen displays a list of items in chronological order, each representing an old or pending request.
  Each item in the list has the following features:
  - name: FC or email of an individual, in case of a single request, or brief description of the group of users, in case of a group request.
  - download button: enables the *Third Party* to download at any time the CSV file containing the requested data, or to have it sent by email.
  - subscription button: allows at any time to end a subscription to a certain request. In case of single requests, the subscription will be active only if the user grants his permission with a positive response.

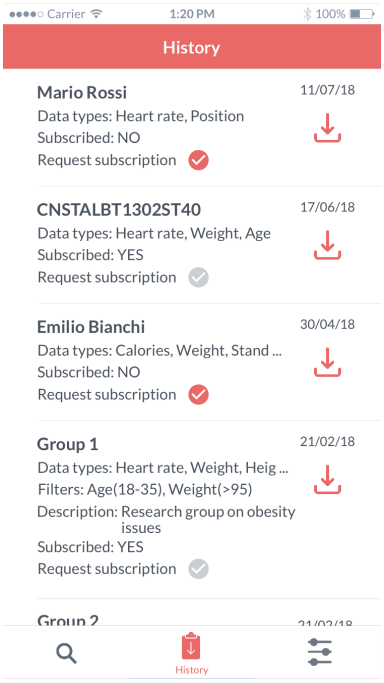


Figure 14: History tab

- **My Settings tab**
  In this section the *Third Party* is able to manage its general information associated to its account and possibly modify its password. However a *Third Party* cannot change the email and P.IVA uniquely linked to its account as they are its unique identifiers.
  In this screen the user will find all the information linked to its account in a list, where each item can be modified at any time. After any modification the user will have to save its updates in order to notify the application's server.

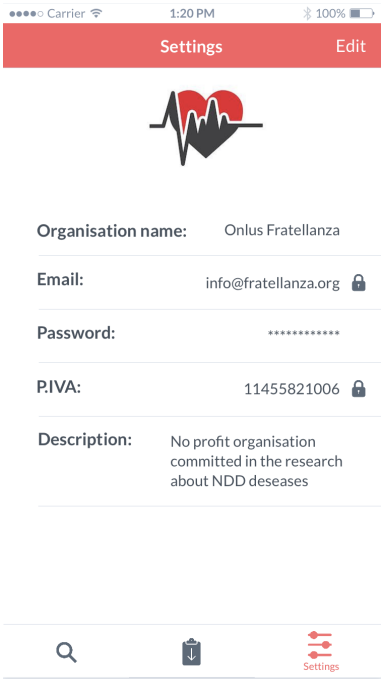


Figure 15: My Settings tab

- **TRACK4RUN INTERFACES**
  - **Logo**
    While reminiscing its underlying service's logo with the sketch of a heart pulse, *Track4Run*'s logo is iconic on its own. The image stands out for its simple but direct meaning. The symbols of a runner and the heart pulse in a red colored background immediately convey the purpose of the application: to track runners during their activity while keeping track of their health.
    The logo is also used as the application's icon and will be visible to whoever downloads the app onto his device.

Figure 16: *Track4Run* Logo

– **Login, Signup or access as** *Spectator*
Whenever a user downloads *Track4Run* for the first time, he will be immediately asked to login, sign up or access as a *Spectator*. A user can only login with his *Data4Help* account. If absent, he may create it at the moment and can therefore access all *Data4Help*'s services through its exclusive app. However, even without an account, a user will still be able to access one of *Track4Run*'s services, that look for an active run a follow its participants live on the track.

– **Single User Interfaces** When a user logs into *Track4Run* as a *Single User* he will be able to access all different features through a tab bar controller. We shall now start describing each tab's purpose:

  ◦ **My Run tab**
  The MyRun tab is the default interface, showed as soon as a user logs into his account. In this section a user can see the active run he has decided to join, check its starting and ending time and its path on a map.

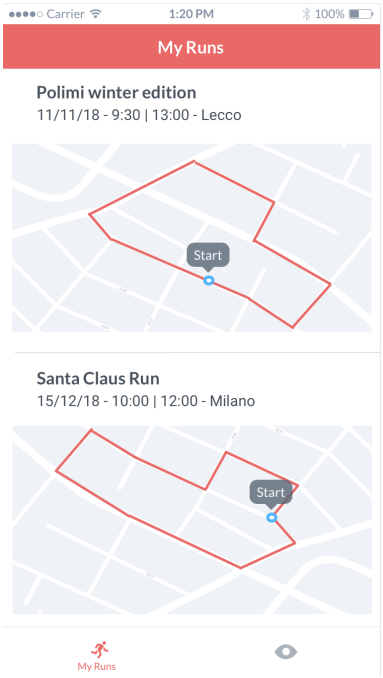


Figure 17: My run tab

- **Active Runs tab**
  Active Runs tab is an interface in which a user can scroll through a list of all active runs and decide whether to monitor it as a *Spectator* or to join it. When looking for a specific run, a user can either run through the entire list or research it by its unique identifier. Furthermore a user may also want to filter the list by location, in order to find the closest runs to his position.

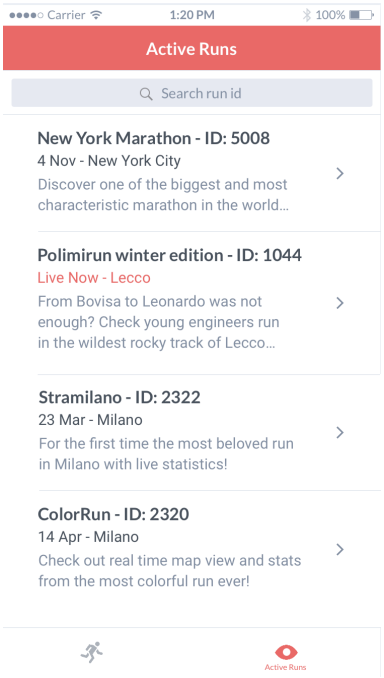


Figure 18: Active runs tab

– **Third Party Interfaces** Also *Third Parties* will access all *Track4Run*'s features by navigating through a tab bar controller. We shall briefly describe each tab's features:

  - **New Run tab**
    New Run tab is the default tab in a *Third Party*'s interface. In this section a *Third Party* can organize a new run. A form where to fill in the run's detail is immediately presented to the user. In order to create a new run the user will have to provide a name for it, a starting and ending time and finally will have to select the desired path onto a map. After having submitted the new run, the *Third Party* will receive the run's unique identifier and the newly created will appear in the My Runs tab.

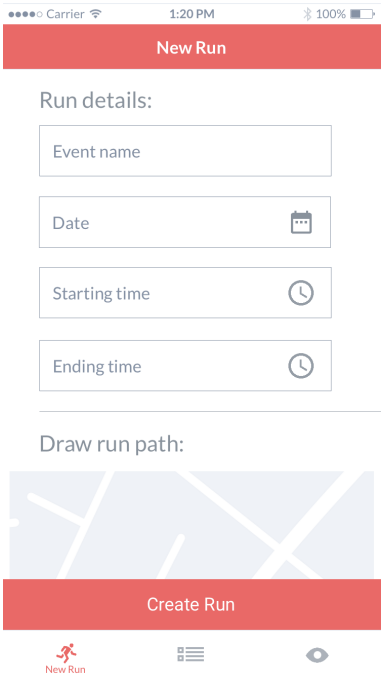


Figure 19: New run tab

- **My Runs tab**
  This section keeps a history of all runs a *Third Party* has organised, both past and active ones. As soon as *Third Party* creates a new run this will immediately appear on top of the list. For all runs, the organiser will be able to see who has participated or will participate to it and will be able to download a file containing all the runners' health data. The types of downloaded data will depend on those accepted to be shared by each participant.

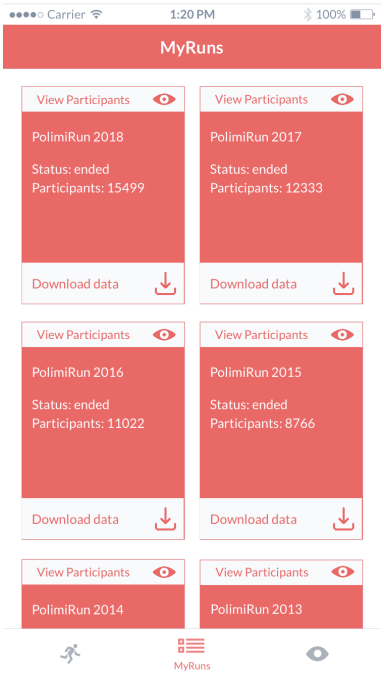


Figure 20: My runs tab

- **Active Runs tab**
  In this section a *Third Party* can check all active runs at the moment and will be able to monitor one as a *Spectator*. To look for a specific run the user can either scroll through the entire list or enter a specific run's ID. Furthermore the user can also filter the list my entering a location.

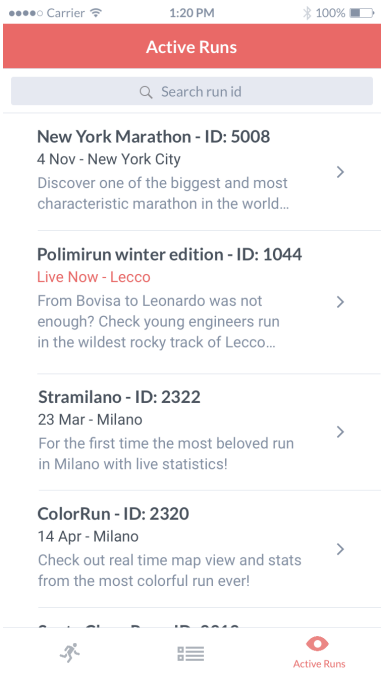


Figure 21: Active runs tab

– *Spectator* **Interfaces**

  - **Research by identifier or in a list**
    Whenever a user access the application as a *Spectator*, that is without logging in, he will be presented with a unique interface with a list of all active runs. The user can scroll through the entire list of active runs and select one to spectate. Furthermore a run can be researched by entering its unique ID or by filtering the list with a location.

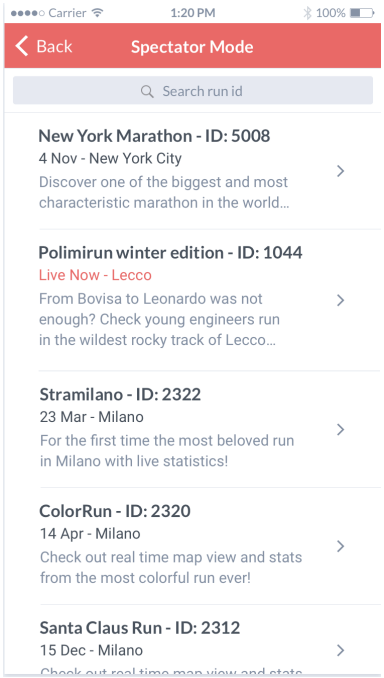


Figure 22: Research run tab

- **Selected run**
  After having selected a run, the *Spectator* will see its path and its participants' locations live on the track. This view will be visible for the whole duration of the run.

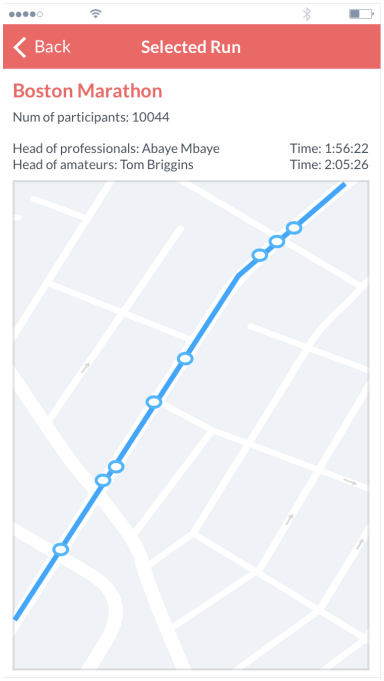


Figure 23: Selected run tab

3.1.2 Hardware Interfaces

The *System* doesn't offer any Hardware Interfaces.

3.1.3 Software Interfaces

- **Operating System:** As mobile applications, the main software interfaces would be offered to the OS on which they run, in particular:
  - iOS
  - Android
  - Android Wear
  - WatchOS
- **Web Server APIs:** Functioning as endpoints offered to both *Data4Help* and *Track4Run* mobile applications to access the Web Server. These APIs are public and can be used by *Third Parties* to directly access the Web Server (authentication has to be provided with each request).

3.1.4 Communication Interfaces

- **HTTPS Protocol:** to safely communicate through the internet with the Web Server and the DBMS.

3.2 Functional Requirements

3.2.1 Use Case Diagrams

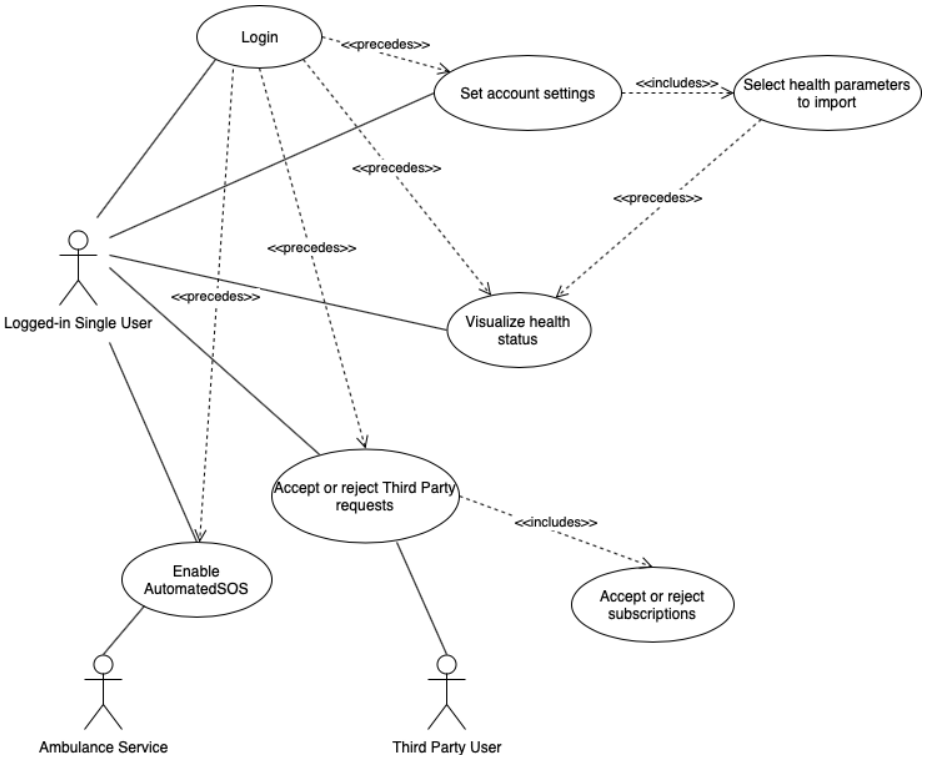


Figure 24: *Single User* Use Case Diagram

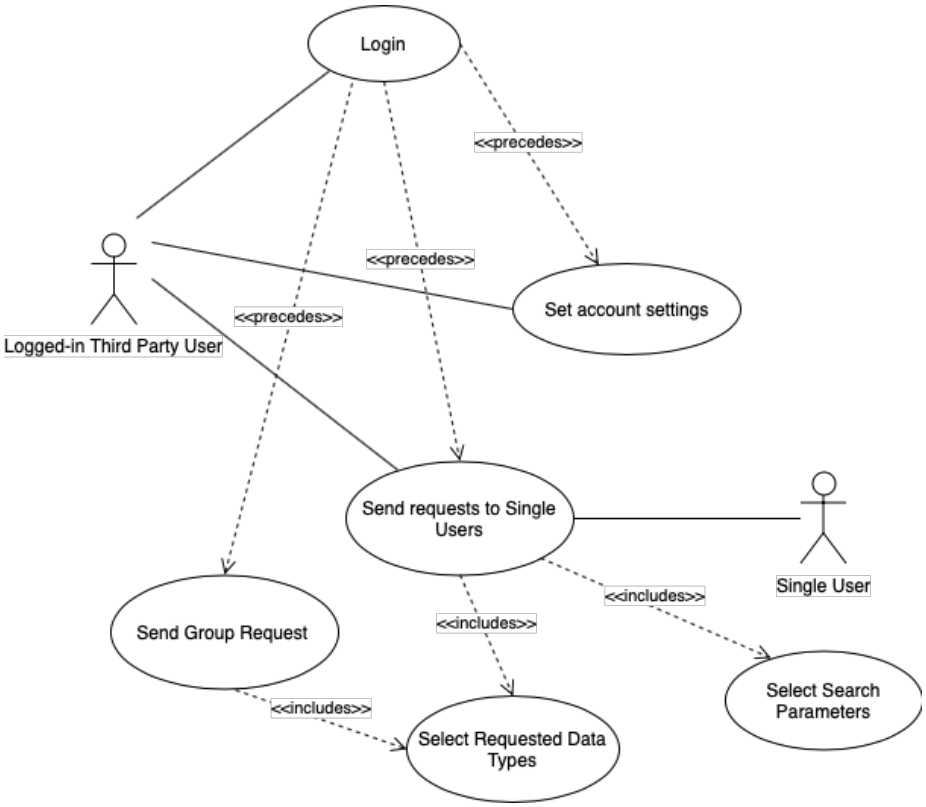


Figure 25: *Third Party* User Use Case Diagram

3.2.2 Scenarios

- **Scenario 1**
  Antonio is a 65 years old retired trader who enjoys a comfy and quiet life. He jogs in the park every day to keep up his good shape. One morning he is running as usual when he starts feeling a bit of chest pain. John does not pay much attention and he attributes it to the fatigue derived from running. Suddenly he feels a strong pang at the heart level. He is having a severe heart attack due to excessive stress and fatigue. His heart completely stops as he falls to the ground. There is no-one around to call for help. Luckily Antonio has activated the Automated SOS feature from the *Data4Help* app installed on his smartphone. The app is collecting data from his smartwatch and as soon as it notices the sudden heart rate drop it performs a silent call to the emergency number providing Antonio's position. The emergency team arrives just in time and is able to save Antonio with a cardiac massage.

- **Scenario 2**
  The San Raffaele Hospital wants to conduct a study on the average hours

slept by young adults living in Milan compared to the those of the rest of the Lombardy's towns. The Hospital downloads *Data4Help* and creates an account as a *Third Party* User. Researchers at the hospital access the application through this account and decide Polto perform group requests to retrieve data concerning sleeping hours. For the purpose of their studies, they decide to use the filtering feature provided by the application to select a group of users aged between 25 and 40 years old and living in the Municipality of Milan. In order to compare this result to the hours slept in average by all young adults in Lombardy, the researches perform the same group request but filter the result on the whole region. Thanks to the data retrieved through *Data4Help*, the group discovers that young adults in Milan sleep an average of 1.5 hours less than their neighbors and finally publishes its report.

- **Scenario 3**
  Politecnico di Milano wants to organize a new Polimirun open only to students, for free. The organizers decide to make use of the brand new *Track4Run* application released by three of its alumnis. They sign-up with an Organizer account, create an event for the 11th November 2018 in Lecco, and send the identification code to all students by email. More than 5000 students want to take part in the event, so they sign up with a Runner account and join the run after searching it with the identification code received. Politecnico can now get rid of the sensors they placed on runner ids, as *Track4Run* allows the organizers to keep track of the path of each runner and provides in-depth statistics not only regarding running pace, but also heart rate for instance. Last but not least, all supportive parents at home can now follow and cheer for their sons in real- time by joining the event as *Spectator*s, without having the need to sign-up on the service.

- **Scenario 4**
  Federico was diagnosed with a severe cardiac disease for which he has been suffering for several years. Bob's doctor decides to try a new a new medical cure that, alongside with a more healthy lifestyle, might improve Federico's everyday life. In order to follow his patient's progresses and recovery, Federico's doctor decides to download *Data4Help* and suggests his patient do the same. While the doctor will create a *Third Party* account, Federico will create a *Single User* one. Thanks to this service Federico's doctor is able to subscribe to Federico's health data, monitor his heart rate on a daily basis and verify if he is increasing his active energy. All this gathered data will help him establish whether the cure shows any results and will allow Federico to feel safer when moving around. Furthermore Federico can also activate *AutomatedSOS*, so, whenever his disease threatens him the most, *Data4Help* will immediately discover his critical condition and call an ambulance.

3.2.3 Use Cases

ID	UC1
Description	A *Guest* creates a *Single User* account for the application
Actors	*Guest*
Preconditions	• The *Guest* has downloaded the app onto his mobile device • The *Guest* doesn't already have an account
Flow of events	1. The *Guest* opens the application 2. The *System* asks the user to register or to login 3. The *Guest* selects the register button 4. The *Guest* swipes through the pages of the tutorial 5. The *System* asks which type of account the *Guest* wants to create 6. The *Guest* selects the button corresponding to the creation of a Single User account 7. The *System* presents a form to fill with the user's personal information, including FC and credentials (email and password) 8. The user fills in the form 9. The *System* checks the validity of the input form 10. The *Mailing System* sends the user an email to confirm its account 11. The *User* receives the email and selects the URL to confirm its account
Postconditions	• The *System* has stored a new *Single User* account associated to the requesting *Guest* • The *User* can access the application's functionalities by logging in

Exceptions	• The *Guest* inputs an email that is already associated to an account. • The *Guest* inputs a password that doesn't respect security constraints. • The *Guest* inputs a FC that is already associated to an account. For all the above: The *System* shows the user an error message and the flow of events restarts from point 7. • The *Guest* selects the login button and provides credentials that are not associated to any account The *System* shows the user an error message and the flow of events restarts from point 2.

Table 3: Create a *Single User* account Use Case

ID	UC2
Description	A *Guest* creates a *Third Party* account for the application
Actors	*Guest*
Preconditions	• The *Guest* has downloaded the app onto his mobile device • The *Guest* doesn't already have an account

Flow of events	1. The *Guest* opens the application 2. The *System* asks the user to register or to login 3. The *Guest* selects the register button 4. The *Guest* swipes through the pages of the tutorial 5. The *System* asks which type of account the *Guest* wants to create 6. The *Guest* selects the button corresponding to the creation of a *Third Party* account 7. The *System* presents a form to fill with the user's personal information, including P.IVA and credentials (email and password) 8. The user fills in the form 9. The *System* checks the validity of the input form 10. The *Mailing System* sends the user an email to confirm its account 11. The *User* receives the email and selects the URL to confirm its account
Postconditions	• The *System* has stored a new *Third Party* account associated to the requesting *Guest* • The *User* can access the application's functionalities by logging in

Exceptions	• The *Guest* inputs an email that is already associated to an account. • The *Guest* inputs a password that doesn't respect security constraints. • The *Guest* inputs a P.IVA that is already associated to an account. For all the above: The *User Account Manager* shows the user an error message and the flow of events restarts from point 7. • The *Guest* selects the login button and provides credentials that are not associated to any account The *System* shows the user an error message and the flow of events restarts from point 2.

Table 4: Create a *Third Party* Use Case

ID	UC3
Description	A *Single User* logs in
Actors	*Single User*
Preconditions	• The *Single User* has already created a *Single User* account
Flow of events	1. The *Single User* opens the application 2. The *System* asks the user to login or register 3. The *Single User* selects the login button 4. The *System* asks the user to input its credentials (email and password) 5. The *Single User* inputs its credentials 6. The *System* checks the provided credentials
Postconditions	• The *System* shows the *Single User*'s main interface

Exceptions	• The *Single User* inputs the wrong credentials. The *System* shows the user an error message and the flow of events restarts from point 4.

Table 5: Log in a *Single User* Account

ID	UC4
Description	A *Third Party* logs in
Actors	*Third Party User*
Preconditions	• The *Third Party* has already created a *Third Party* account
Flow of events	1. The *Third Party* opens the application 2. The *System* asks the user to login or register 3. The *Third Party* selects the login button 4. The *System* asks the user to input its credentials (email and password) 5. The *Third Party* inputs its credentials 6. The *System* checks the provided credentials
Postconditions	• The *System* shows the *Third Party*'s main interface
Exceptions	• The *Third Party* inputs the wrong credentials. The *System* shows the user an error message and the flow of events restarts from point 4.

Table 6: Log in a *Third Party* Account

ID	UC5

Description	A *Single User* selects which health data to import
Actors	*Logged-in Single User*
Preconditions	• The *Single User* has logged into his account
Flow of events	1. The *Single User* selects the My Settings tab in the application 2. The *System* shows all the information related to the user's account and enables modifications 3. The user selects the types of health data to import from the list displayed in the interface
Postconditions	• The *System* updates the user's account settings • The *System* imports data from the user's device corresponding to the new selected types (if any) • The *System* does not import anymore the removed types (if any)
Exceptions	• The User has no available data related to the selected type The *System* does not retrieve any data corresponding to that type

Table 7: Select types of health data to import

ID	UC6
Description	A *Single User* accepts a request from a *Third Party*
Actors	*Logged-in Single User*, *Third Party*
Preconditions	• The *Single User* has logged into his account

Flow of events	1. The *System* signals a new *Third Party* request with a notification on the My Follower tab icon 2. The Logged-in *Single User* selects the My Followers tab 3. The *System* in this tab shows the user a *Third Party* request 4. The user reads the description of the *Third Party* sending the request 5. The *System* asks the user to accept or reject the request 6. The user accepts the request 7. The *System* signs the request as accepted and forwards the response to the sender
Postconditions	• The *Third Party* sending the request is added to the list of the user's followers • The *System* retrieves the accepting *Single User*'s data and forwards it to the requesting *Third Party*
Exceptions	• There is no available data related to the *Single User*'s account corresponding to the parameters input by the *Third Party* The *System* notifies to the *Third Party* that there is no such data available.

Table 8: Accept a *Third Party* Request

ID	UC7
Description	A *Single User* rejects a request from a *Third Party*
Actors	*Logged-in Single User*, *Third Party*
Preconditions	• The *Single User* has logged into his account

Flow of events	1. The *System* signals a new *Third Party* request with a notification on the My Follower tab icon 2. The Logged-in *Single User* selects the My Followers tab 3. The *System* in this tab shows the user a *Third Party* request 4. The user reads the description of the *Third Party* sending the request 5. The *System* asks the user to accept or reject the request 6. The user rejects the request 7. The *System* signs the request as rejected and forwards the response to the sender
Postconditions	• The *Third Party* receives the notification from the *System* signaling the user's rejection
Exceptions	

Table 9: Reject a *Third Party* Request

ID	UC8
Description	A *Third Party* sends a request to a *Single User*
Actors	*Logged-in Third Party* , *Single User*
Preconditions	• The *Third Party* has logged into his account

Flow of events	1. The *Third Party* opens the Search tab in the application 2. The *System* asks the user to select the type of research: single request or group request 3. The *Third Party* selects the request to a *Single User* 4. They *System* asks the user to select the requested data types and the *Single User*'s ID 5. The user inputs the ID (FC or email) and the requested data types 6. The user submits the request 7. The *System* checks if the input ID is valid and if any requested data type is available 8. The *System* forwards the request to the addressed *Single User*
Postconditions	• The *System* forwards the *Single User*'s response to the *Third Party* (acceptance or rejection)
Exceptions	• The input ID is not associated to any account • All requested data types of the *Single User* are not available. For all the above: The *System* notifies the *Third Party* with an error message and the flow of events restarts from point 2.

Table 10: Request Data from a *Single User*

ID	UC9
Description	A *Third Party* submits a Group Request
Actors	*Logged-in Third Party*
Preconditions	• The *Third Party* has logged into his account

Flow of events	1. The *Third Party* opens the Search tab in the application 2. The *System* asks the user to select the type of research: single request or group request. 3. The *Third Party* selects group request 4. The *System* asks the user to select the search parameters and the selected data types 5. The user inputs the search parameters 6. The user submits the request 7. The *System* analyses the search parameters and checks the corresponding data in the DB
Postconditions	• The *System* sends the anonymous data matching the search parameters to the *Third Party* • The *Third Party* can download the researched data
Exceptions	• The matching data belongs to a group of not more than 1000 users. The *System* notifies the user that the matching data corresponds to a group that is too small and therefore cannot be retrieved. The flow of events restarts from point 2. • There is no data in the db matching the search parameters. The *System* notifies the user that there is no available data matching the request. The flow of events restarts from point 2.

Table 11: Submit a Group Request

ID	UC10
Description	A Logged-in *Single User* enables *AutomatedSOS*
Actors	*Logged-in Single User*

Preconditions	• The user has logged into his account
Flow of events	1. The Logged-in *Single User* selects the My Health tab in the application 2. The user selects the *AutomatedSOS* button to enable it 3. The *System* asks the user permission to access his calls 4. The user grants access to his calls
Postconditions	• When the user's health parameters are critical, the *System* calls an ambulance to the user's location
Exceptions	• The user does not grant permission to make phone calls. The *System* displays an error and disables *AutomatedSOS*.

Table 12: Enable *AutomatedSOS*

ID	UC11
Description	A *Third Party* organizes a run
Actors	*Logged-in Third Party*
Preconditions	• The user has logged into his account in *Track4Run*

Flow of events	1. The user selects the New Run tab 2. The *System* asks the user to define the run's details: name, starting and ending times, maximum participants 3. The user inputs the run's details 4. The *System* shows the user a map on which to define the path 5. The user selects a path on the map 6. The user submits the new run
Postconditions	• The new run is added to the list in the Active Runs tab • The new run is visible to all *Single User*s and *Spectator*s
Exceptions	• The organizer selects a non feasible path on the map. The *System* asks the user to select another path.

Table 13: Organize a run

ID	UC12
Description	A *Single User* joins an active run
Actors	*Logged-in Single User*
Preconditions	• The user has logged into his account on *Track4Run*

Flow of events	1. The user selects the Active Runs tab 2. The *System* shows the user a list of all active runs 3. The user selects a run to join 4. On the day of the run, the *System* sends the user an automatic request to share his health data and position with the Organizer. 5. The User accepts the request
Postconditions	• The User can track his position live on the run's map in the My Run section • The User's position is visible to all *Spectator*s
Exceptions	• The user doesn't accept the request to share his data. The *System* shows an error message to the user saying he cannot participate to the run. The flow of events restarts from point 2.

Table 14: Join an active run

ID	UC13
Description	A *Spectator* looks for a specific run to spectate
Actors	*Spectator*
Preconditions	• The *Spectator* has downloaded the application *Track4Run*

Flow of events	1. The User opens *Track4Run* 2. The *System* asks the user to login, register or access the service as a *Spectator* 3. The User selects to access as a *Spectator* 4. The *System* asks the User to find a run by its identifier or to search in the list of active runs 5. The User inserts the run's identifier 6. The User submits its research
Postconditions	• The user can spectate the run and see its participants' live positions
Exceptions	• The identifier input by the user is not associated to any existing run The *System* shows an error message to the user. The flow of events restarts from point 4.

Table 15: Spectate a run

3.2.4 Sequence Diagrams

The first Sequence Diagrams models the creation of a new account on *Data4Help*. The two actors involved are a guest user and the *System*. We assume the guest to be a generic user as the same procedure is applicable to both users wanting to create a Third Party account and a *Single User* account. Furthermore, the same sequence of events occurs when registering through *Track4Run*.

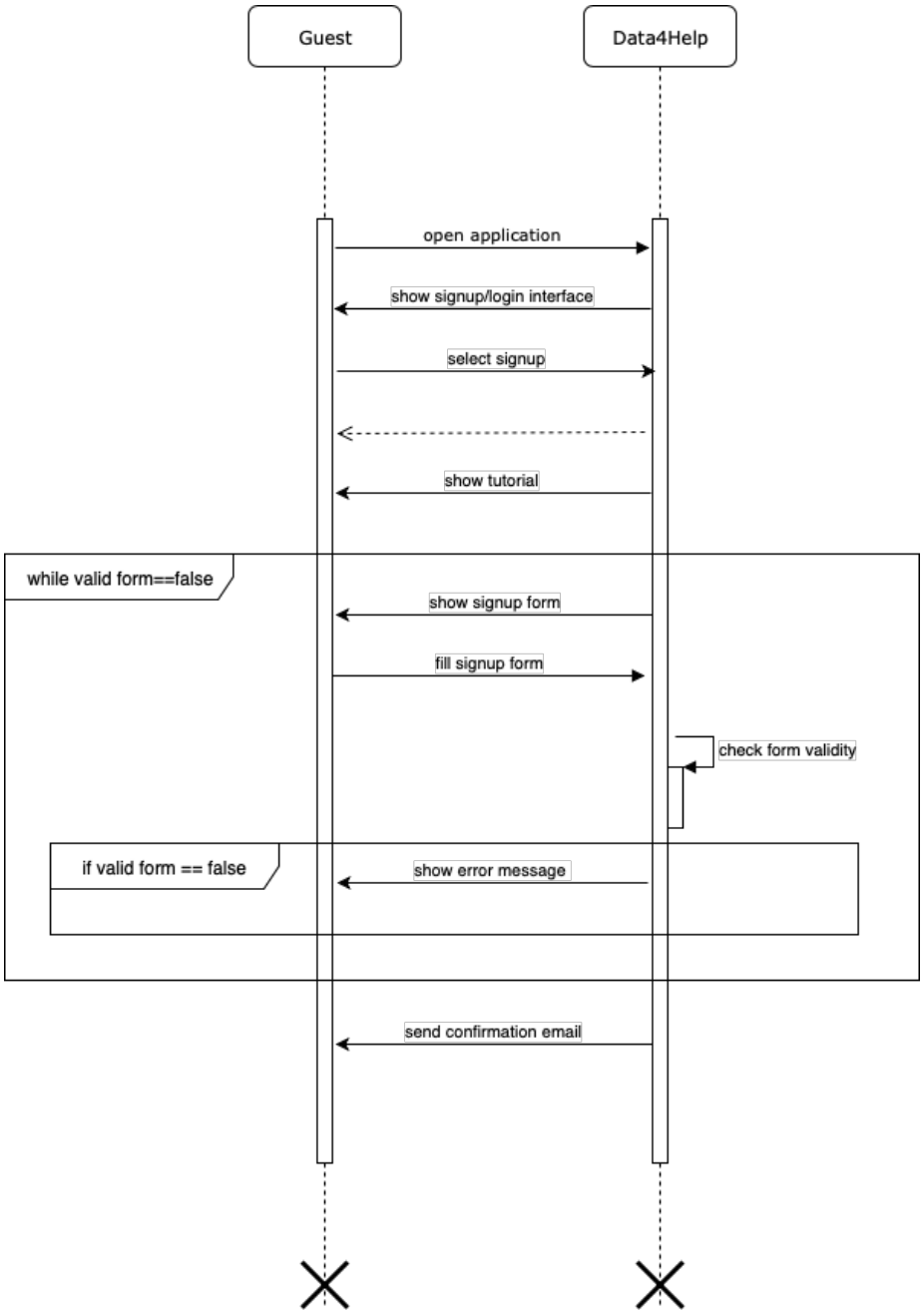


Figure 26: Create an account

The following diagram describes the sequence of events to be performed in order for a user to log into his account. As in the previous diagram, we shall

consider a generic user since this procedure is the same for logging into a *Third Party* account or a *Single User* account.

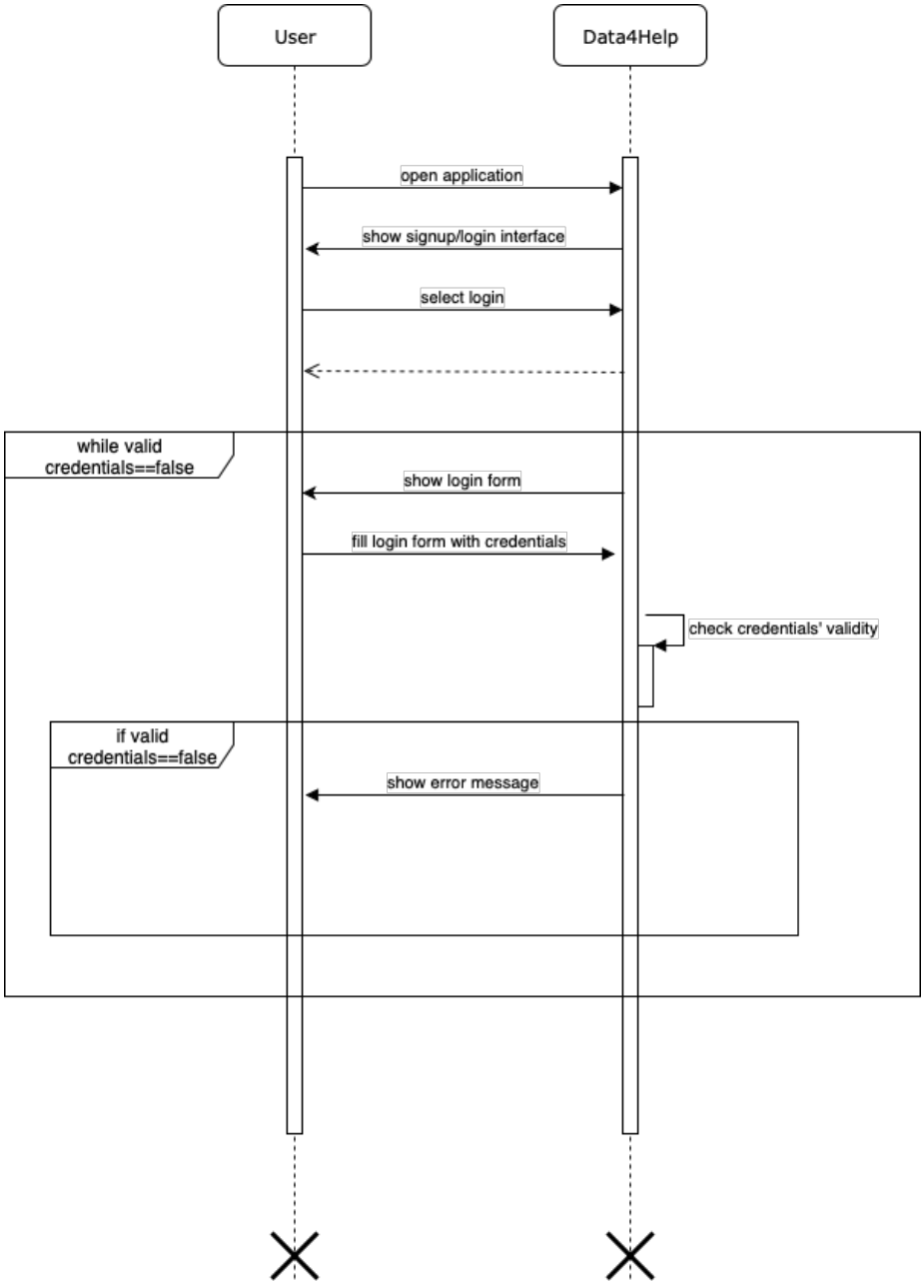


Figure 27: Log in

The following diagram describes the procedure that occurs whenever a *Third Party* wants to send a request to a *Single User*. The actor involved in this scenario are a *Third Party*, a *Single User* and the *System*.

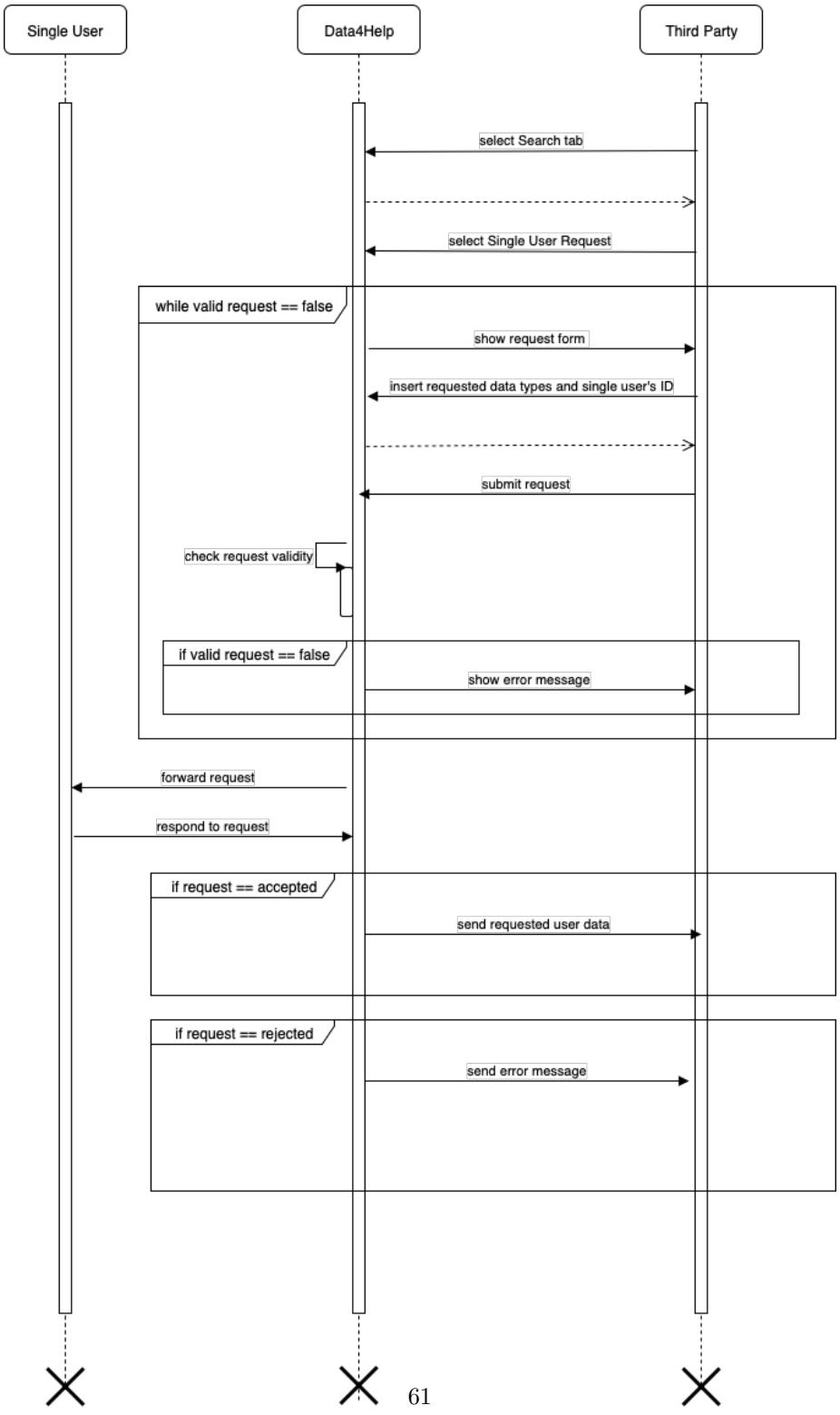


Figure 28: Send a *Single Request*

In this diagram we describe the sequence of procedures that are activated when a *Third Party* sends a group request to the *System*. In this case no single users are involved, thus resulting in a two-actor scenario: a *Third Party* and the *System*.

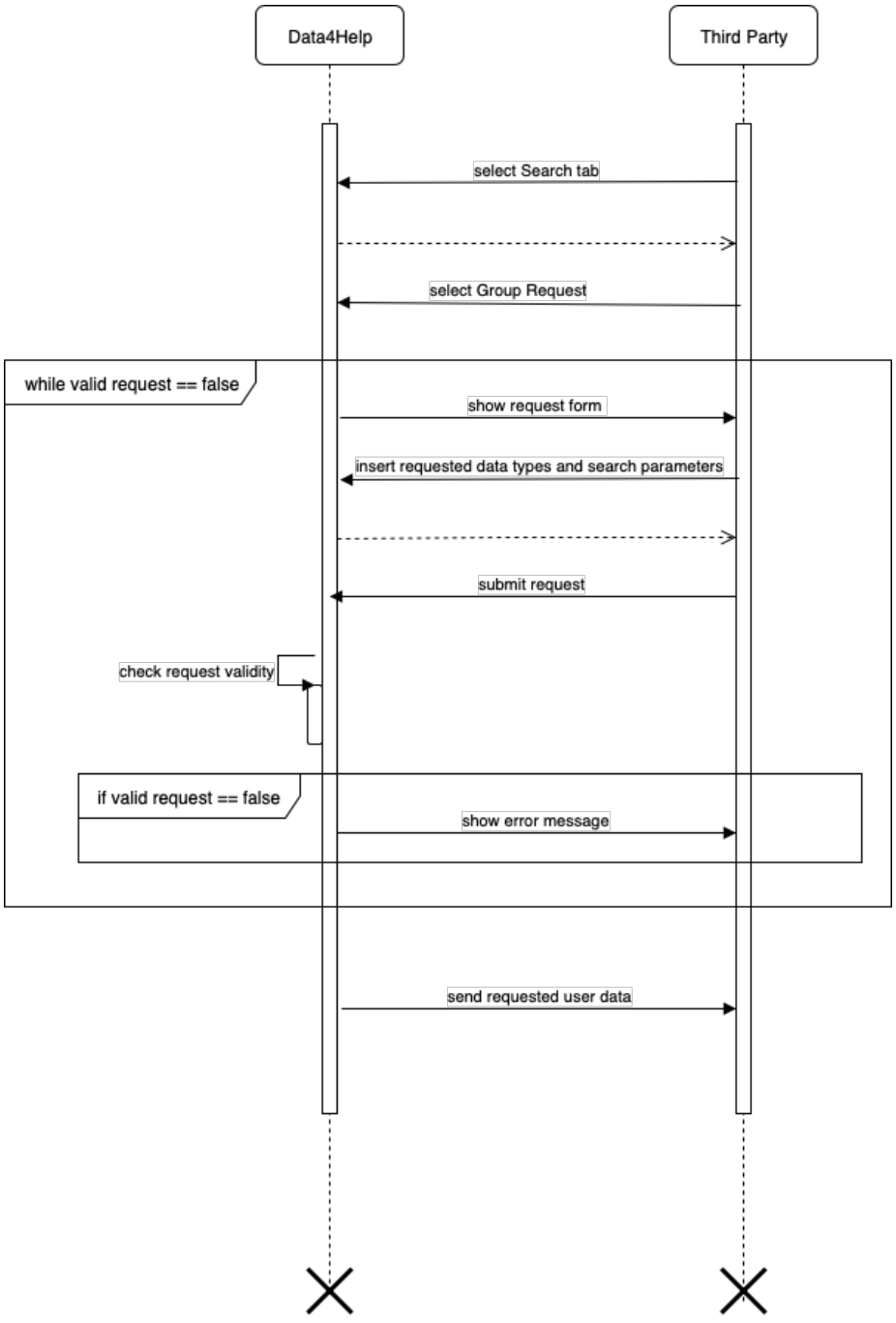


Figure 29: Send a *Group Request*

The following diagram describes the sequence of events to be performed in order for a single user to enable *AutomatedSOS*. While modeling this feature activation, we also describe how it works when critical parameters are observed by the *System*. Therefore in this case our actors are a *Logged-in Single User*, the *System* and the *Ambulance Service*.

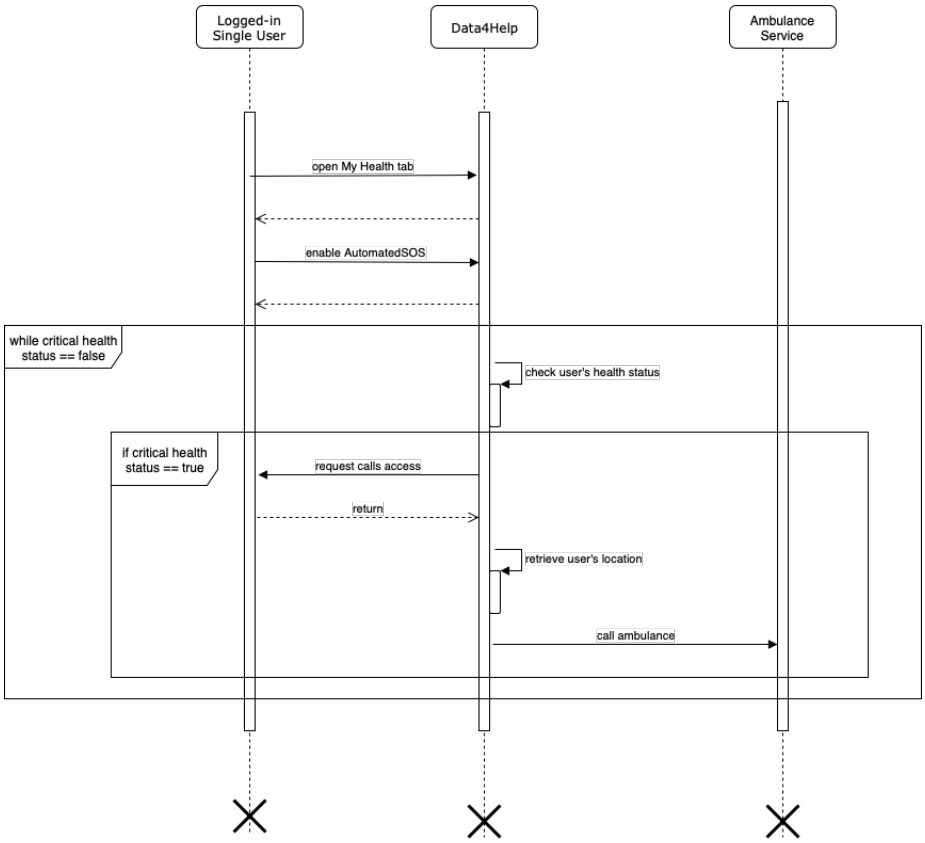


Figure 30: Enable *AutomatedSOS*

The next diagram describes the sequence of events that occur whenever a *Third Party* organizes a new run in *Track4Run*. The *Third Party* and the *System* are the only actors involved.

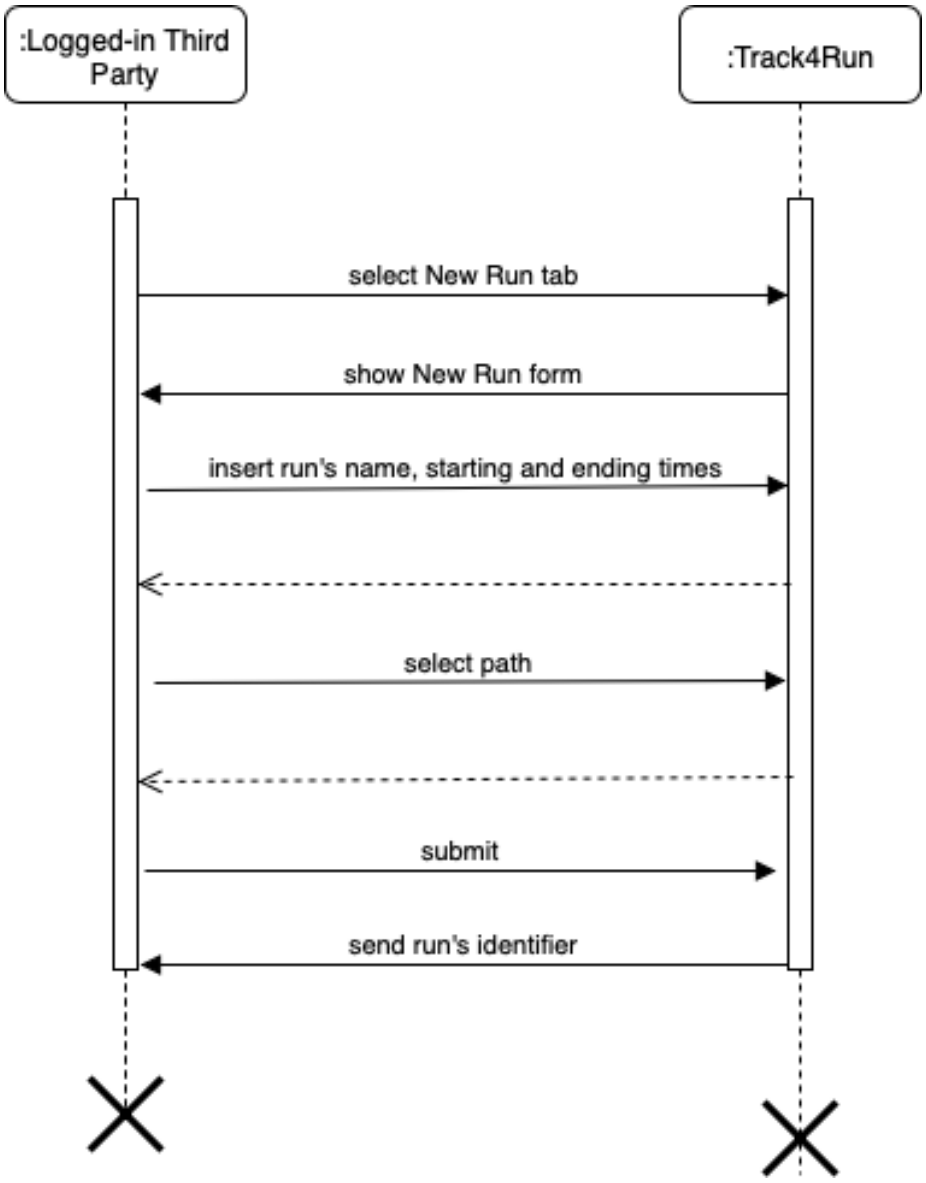


Figure 31: Organize a run

The following digram represents the sequence of events that occur when a *Single User* decides to join an existing run in *Track4Run*. The actors involved in the scenario are only the *Single User* and the *System*.

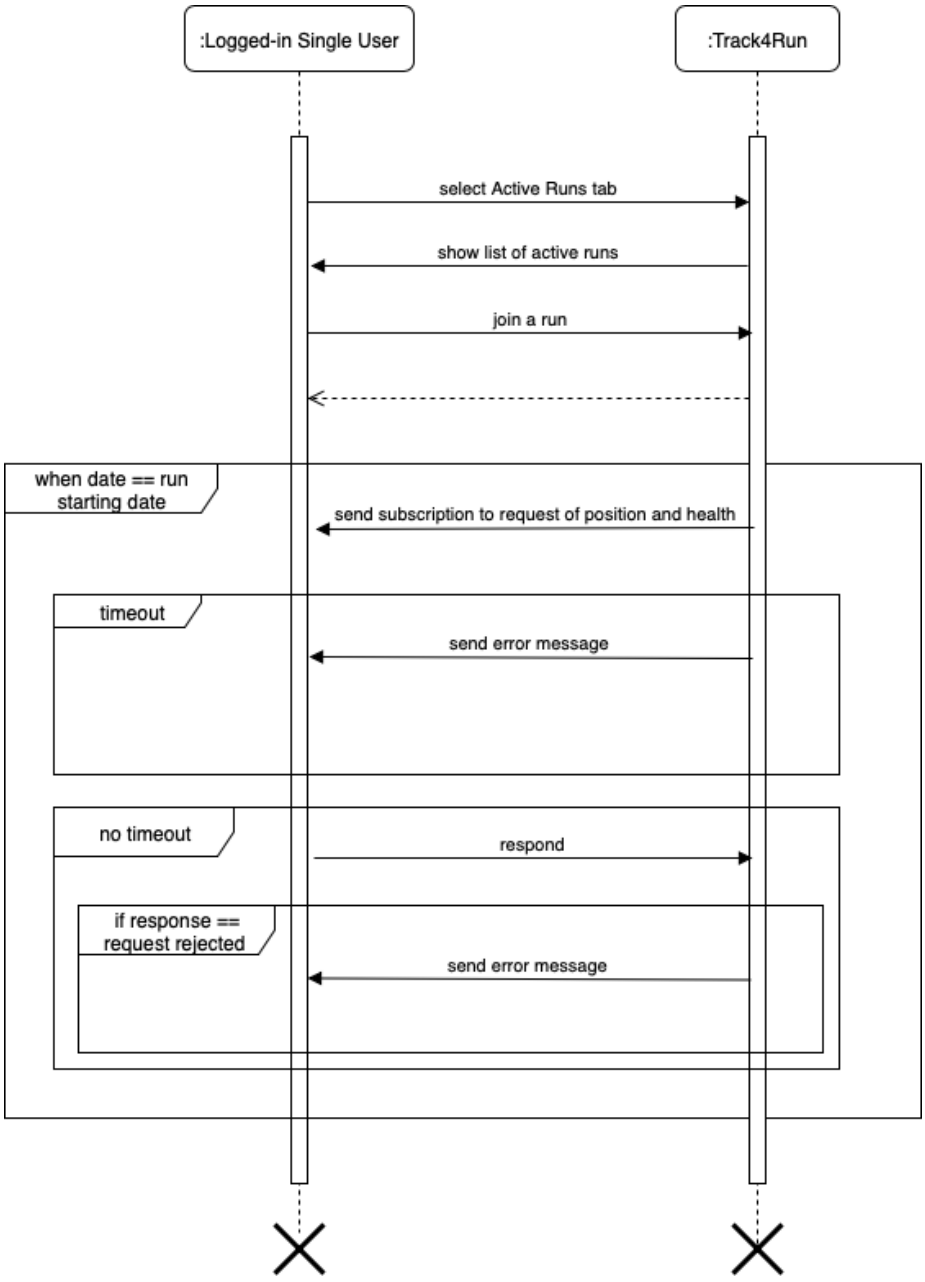


Figure 32: Join a run

The final diagram shows the sequence of events that occur when a *Spectator* uses *Track4Run* to spectate an existing run. The *Spectator* and the *System* are the only actors involved in the scenario.

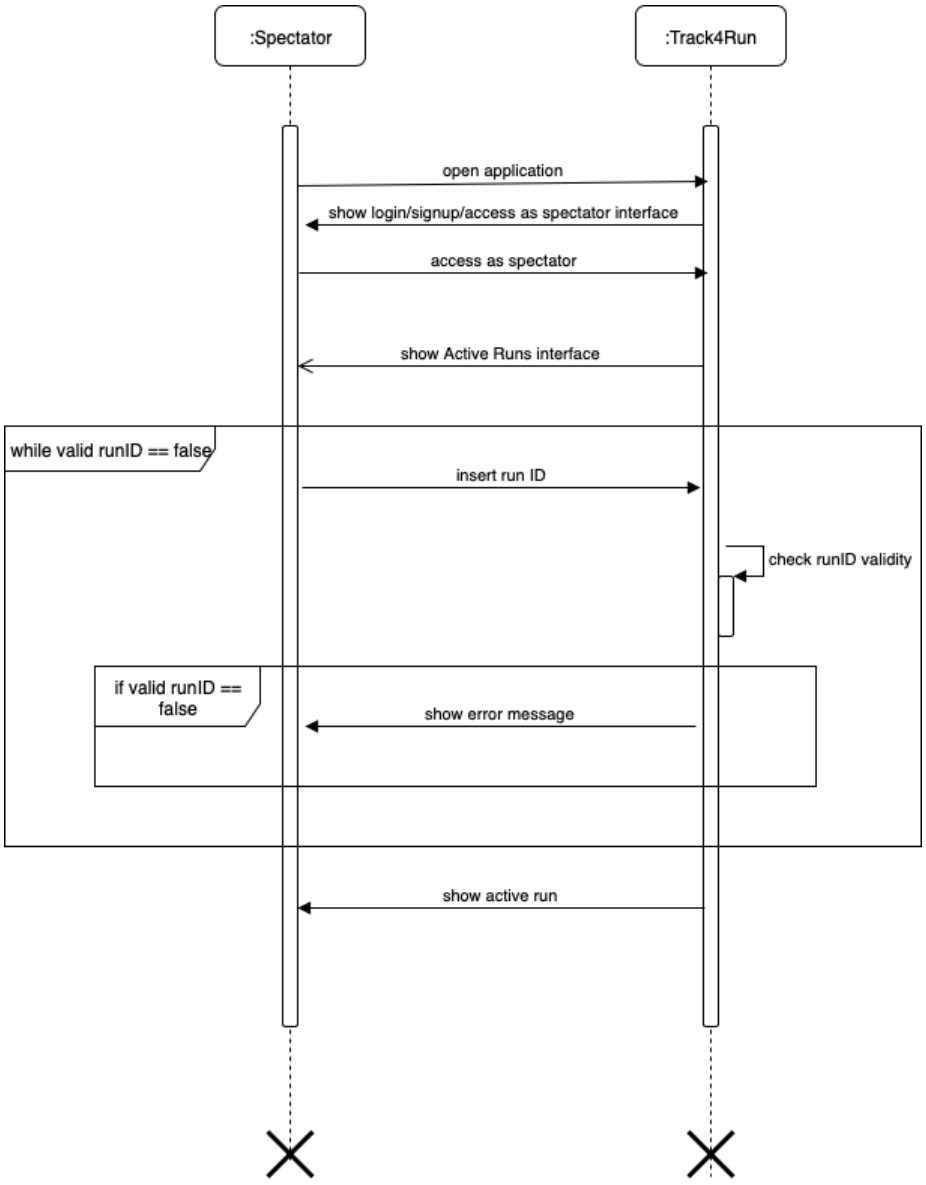


Figure 33: Spectate a run a

3.2.5 Goal mapping on requirements

- **[G1]** Users can be identified either as *Single Users* or as *Third Parties*.
  - **[R1]** The S2B allows users to create either a *Single User* or a *Third Party* account.

  - [**R2**] A *Single User* account can be created if and only if the user provides his Fiscal Code.
  - [**R3**] A *Third Party* account can be created if and only if a valid P.IVA is provided.
  - [**R4**] All users can create an account if and only if they provide a unique email and a password.
  - [**R5**] To access the service users must log in with their account credentials.
  - [**D1**] Each user will create only one account.
  - [**D2**] The FC and email provided during registration are valid and belong to the person who's creating the account.
  - [**D3**] The P.IVA and email of a *Third Party* are valid and specific to that company.

- [**G2**] *Single Users* can have their data monitored by *Third Parties* for medical purposes.
  - [**R6**] The S2B allows users to manually insert Health data.
  - [**R7**] The S2B automatically imports new data whenever the application is opened.
  - [**R8**] When the application is open, the S2B continuously imports data in background.
  - [**R9**] The S2B binds collected data only to the user's account that imported it.
  - [**R10**] *Third Parties* can submit a request to access data of a *Single User*.
  - [**R11**] Requests concerning *Single User*s must specify either the email or the FC of the desired user.
  - [**R12**] Requests concerning *Single User*s are forwarded only to the specified user.
  - [**R13**] A user can accept or refuse requests forwarded to him.
  - [**R14**] *Third Parties* can access user's data if and only if their request is accepted by said user.
  - [**D4**] Data provided to the *System* is related to the person whose account was used to provide it.
  - [**D5**] All health data directly provided by the user represents his real health status.
  - [**D7**] Health data has a relative error lower than 5
  - [**D8**] Permission to access health and GPS data is always granted to the S2B.

  - [**D9**] *Third Parties* collect data only for medical purposes.

- [**G3**] *Single Users* can visualize a summary of their health status
  - [**R6**] The S2B allows users to manually insert Health data.
  - [**R7**]The S2B automatically imports new data whenever the application is opened.
  - [**R8**]When the application is open, the S2B continuously import data in background.
  - [**R9**] The S2B binds collected data only to the user's account that imported it.
  - [**R15**] The S2B allows *Single User*s to visualize their historical data using Time Series.
  - [**R16**] The S2B allows *Single User*s to visualize their historical data using aggregated statistical operators.
  - [**R17**]The S2B allows *Single User*s to visualize their historical data over multiple timespans.
  - [**D4**] Data provided to the *System* is related to the person whose account was used to provide it.
  - [**D5**] All health data directly provided by the user represents his real health status.
  - [**D7**] Health data has a relative error lower than 5%.
  - [**D8**]Permission to access health and GPS data is always granted to the S2B.
  - [**D14**] Users' smartphones always have internet access when the S2B needs it.

- [**G4**] *Third Parties* can access data of those *Single User*s who granted their permission.
  - [**R10**] *Third Parties* can submit a request to access data of a *Single User*.
  - [**R11**] Single Requests must specify either the email or the FC of the desired user.
  - [**R12**] Single Requests are forwarded only to the specified user.
  - [**R13**]A user can accept or refuse requests forwarded to him.
  - [**R14**] *Third Parties* can access user's data if and only if their request is accepted by said user.
  - [**R18**] Single requests must specify the requested data types.
  - [**R19**] A request to a *Single User* must specify whether or not the *Third Party* is subscribing to that request of data.

  - **[R20]** Subscriptions to requests must specify a duration.
  - **[R21]** Subscriptions to requests can be ended by both the *Third Party* and the *Single User* at any time.
  - **[R22]** If none of the requested data types of the *Single User* is available, the *Third Party* receives an error message.
  - **[R23]** *Third Parties* can access only the requested data types that are available.
  - **[R24]** *Third Parties* can download all data obtained through requests on their devices or have it sent by email.
  - **[D8]** Permission to access health and GPS data is always granted to the S2B.
  - **[D9]** *Third Parties* collect data only for medical purposes.
  - **[D10]** Data is stored on persistent memory.
  - **[D11]**The storage *System* is fully replicated and fault-tolerant, so that a copy of a specific piece of data is always available

- **[G5]** *Third Parties* can access data of anonymous groups of users.
  - **[R25]** *Third Parties* can submit a group request to access data of groups of users.
  - **[R26]** Group requests must include at least one search parameter.
  - **[R27]** Group requests must specify the requested data types.
  - **[R28]** Group request results are provided if and only if the number of users matching the search parameters is higher than 1000.
  - **[R29]** Group request results include only the data retrieved by the *System* matching the search parameters.
  - **[R30]** Sensitive data is excluded from group request results.
  - **[R31]** All group requests must specify whether the *Third Party* is subscribing to that request of data.
  - **[R20]** Subscriptions to requests must specify a duration.
  - **[R32]** Subscriptions to requests can be ended by the *Third Party* at any time.
  - **[R24]** *Third Parties* can download all data obtained through requests on their devices or have it sent by email.
  - **[D8]** Permission to access health and GPS data is always granted to the S2B.
  - **[D9]** *Third Parties* collect data only for medical purposes.
  - **[D10]** Data is stored on persistent memory.
  - **[D11]** The storage *System* is fully replicated and fault-tolerant, so that a copy of a specific piece of data is always available.

- **[G6]** Whenever a user's health status becomes critical, an ambulance is sent to his location.
  - **[R33]** Only private users can choose whether or not to enable *AutomatedSOS*.
  - **[R34]** *AutomatedSOS* can be enabled only if the user grants permission to make emergency phone calls.
  - **[R35]** If *AutomatedSOS* is enabled and the *System* detects that a user's heart rate is below or above the critical threshold for his age, an ambulance is called.
  - **[D4]** Data provided to the *System* is related to the person whose account was used to provide it.
  - **[D5]** All health data directly provided by the user represents his real health status.
  - **[D6]**Position data has an accuracy of 10 meters around the actual position.
  - **[D7]** Health data has a relative error lower than 5
  - **[D8]**Permission to access health and GPS data is always granted to the S2B.
  - **[D12]** Permission to make calls is always granted to the S2B.
  - **[D13]** Users' smartphones always have signal when needed by *AutomatedSOS*.
- **[G7]** Organizers can create and manage runs.
  - **[R36]** The S2B allows organisers to create a *Third Party* account using *Data4Help*.
  - **[R37]** Only *Third Parties* can create a run.
  - **[R38]** When creating a run, the organiser must specify its the path, duration and maximum participants.
  - **[R39]** On the day of a run the S2B sends a Single Request to every user who has registered to that run.
  - **[R40]**The request is sent on behalf of the organiser using its email and P.IVA.
  - **[R41]** The request sent is with a subscription that lasts until the end of the run.
  - **[R42]**The request sent by the S2B has as requested data types the position and all available health parameters of the user.
  - **[R43]**The S2B provides a list of all existing runs visible to everyone using *Track4Run*.
  - **[D15]** The path specified when creating a run is feasible.

- [**G8**] Runners can enroll in an existing run.
  - [**R44**] The S2B allows runners to create a *Single User* account using *Data4Help*.
  - [**R45**] Only users with a *Single User* account can join an existing run.
  - [**R43**] The S2B provides a list of all existing runs visible to everyone using *Track4Run*.
  - [**R46**]A user can only join a run in the list provided by the S2B.
  - [**R47**] A user can't check-in for a run if he doesn't accept the request received by the organiser of such run.
  - [**R48**] A user can't accept the request if he doesn't have at least his position available as requestable data type.
  - [**R49**] If a user fails to check-in in the run, the S2B removes him from it.
  - [**D8**] Permission to access health and GPS data is always granted to the S2B.
- [**G9**] Organizers can create and manage runs.
  - [**R50**] An existing run can be spectated without logging in the *System*.
  - [**R43**] The S2B provides a list of all existing runs visible to everyone using *Track4Run*.
  - [**R51**] An existing run can be spectated by selecting it in the list of existing runs.
  - [**R52**] *Spectator*s can see the live position of all runners participating in the run they are spectating.
  - [**D6**]Position data has an accuracy of 10 meters around the actual position.
  - [**D14**] Users' smartphones always have internet access when the S2B needs it.

3.3 Performance Requirements

In this section we will specify some static and dynamic numerical requirements attributed to the *System* or to the interaction between the human user and the application that highlight the performance reachable by the machine.
Regarding static numerical requirements the *System* can support up to 1 million registered users. This limitation is not posed by the front-end of the *System*, but rather by the back-end part, specifically the DB.
Data exchanged is only textual and for each user the *System* provides enough

space to store up to 10Mb of data. If the stored information exceeds this limitation then the oldest pieces will be overwritten.
All the requests that need a direct confirmation from *Single User*s will have a response time of at most 3 days. If at the end of this period the request is still pending it will be automatically denied.
Requests about groups of users shall be processed in less than 5 seconds.
Requests about subscribed *Single User*s shall be processed in less than 1 second.
The *AutomatedSOS* feature does not need to communicate with the back-end; also its role is much more critical given the function it provides and therefore the response time of this subSystem will be of 0.5 seconds starting from the moment it observes critical health parameters for the first time.
Finally, speaking of dynamical numerical requirements the *System* as a whole should not be subject to peak workload conditions; data is retrieved from the end user at a fixed rate and the data requests from *Third Parties* only represent a minimum percentage of the data exchanged and processed between the *System*'s front-end and back-end.

3.4 Design Constraints

3.4.1 Standards compliance

Our *System* will be compliant with standard D0-178C. This means that all software testing will be driven from requirements, thus enabling to link each single requirement the test case used to verify if it has been met.
In our case we consider the S2B as a level D risk level *System* in regard to D0-178C. Therefore we can trace *System* and high level requirements to the test cases, test procedures and test results.
The data format used to track all the variables is the following:

- BPM for heart rate, which is the most common unit of measurement that can be found in the medical field.
- Standard *SI* measurement units for height and weight, namely meters and kilograms.
- Standard longitude and latitude measures for the position.

3.4.2 Hardware limitations

To be able to perform as intended, the application should run in a hardware environment that guarantees the following specifications:

- 100Mb of mass memory
- 20Mb/s wireless internet connection
- 2Gb of Ram

3.5 Software System Attributes

3.5.1 Availability

Given the integration of *AutomatedSOS* in *Data4Help* and assuming that at least one user has the service activated, the Availability should be high. This has to hold for all users, since it is not possible to adapt it to each of them. Therefore the target value is 99%.
Track4Run ideally requires a lower availability, but since part of the back-end is shared with *Data4Help*, the target value should stay the same.

3.5.2 Reliability

The Reliability of *Data4Help*, like the Availability, should be high enough to guarantee the nonfunctional requirements of *AutomatedSOS*. To prevent downtime, one of the main goals of architecture design must be ensuring graceful degradation of the *System*.
This holds for *Track4Run* as well, since downtime during a run would be a severe service disruption.

3.5.3 Security

Security is a key feature in a *System* that holds sensitive data. Thus, the S2B must:

1. As previously stated, use HTTPS to safely communicate with the Server and DBMS.
2. Hash the passwords so that they are not stored in clear in the DB.
3. Encrypt sensitive data before storing it to reduce the effects of data leaks.
4. Prevent *Third Parties* from accessing sensitive data with group requests.

3.5.4 Maintainability

Good software engineering practices must be followed to reduce coupling, avoid code duplication and ensure that modules are self-sustainable. These three practices will guarantee Maintainability.

3.5.5 Portability

The implementation of the S2B as a mobile application ensures itself portability with regards to the user, since a porting from Android to iOS, and *vice versa*, would be quite natural. For what concerns the back-end part, it should be OS independent and easily reused with other hosts.

4 Formal analysis with Alloy

4.1 Overview

We decided to use two alloy models that focus on two different aspects of the S2B.

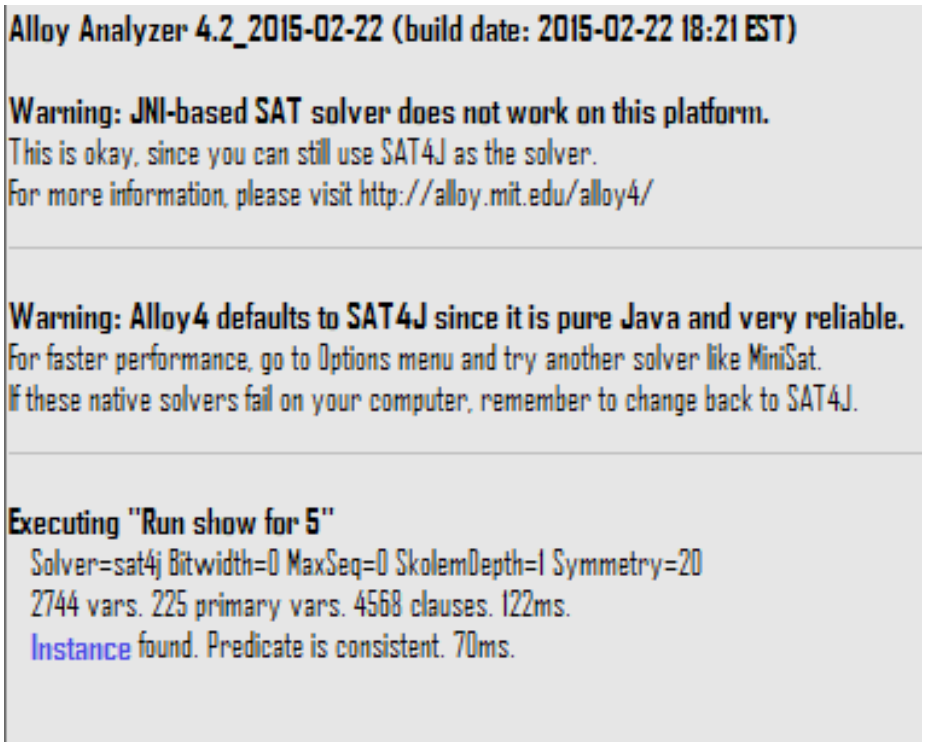
The first one is a very simple static model that formalizes **[G1]** and the requirements used to entail it. The purpose of this first model is to create a possible view of the *System* in its lifecycle, when some accounts have already been created, and to check that requirements regarding uniqueness are satisfied. This model doesn't describe all the components and actors, it just serves as a first-look on the main entities used in the second model.

The second one is a more in-depth dynamic model that formalizes the main aspects of **[G4]** and **[G5]**. This model is derived from the first one by relaxing some constraints on uniqueness in order to represent dynamicity when creating both single and group requests.
This model is intended to be used only through its predicates, and not with a generic show to create a possible world.

In order to make generated worlds easier to understand, all relations that were not relevant for that specific world have been removed from the graphs.

4.2 Static Model

```
open util/boolean

----------Signatures----------
sig Email {}
sig FC {}
sig PIVA {}

abstract sig Account {
	email: one Email
}

sig PrivateAccount extends Account {
	fc: one FC
}

sig ThirdPartyAccount extends Account {
	piva: one PIVA,
	accessedUsers: set PrivateAccount, //users that approved a single request at least once
	subscriptions: set PrivateAccount //users to whom the third party is subscribed
}



----------Facts----------
fact uniquenessAndExistentiality {
	no disj a1, a2: Account | a1.email = a2.email //there are no accounts with the same email
	no disj a1, a2: PrivateAccount | a1.fc = a2.fc //there are no private accounts with the same FC
	no disj u1,u2: ThirdPartyAccount | u1.piva = u2.piva //there are no third parties with the same PIVA
	no e: Email | no a: Account | a.email = e //all emails are linked to at least one account
	no f: FC | no p: PrivateAccount | p.fc = f //all fc are linked to at least one private account
	no p: PIVA | no t: ThirdPartyAccount | t.piva = p //all piva are linked to at least one third party
}

fact subIfAccessed {
	all p: PrivateAccount | all t: ThirdPartyAccount | p in t.subscriptions implies p in t.accessedUsers //a third party can be subscribed to a user he only accessed at least once
}



----------Predicates----------
pred show {}

run show for 5
```



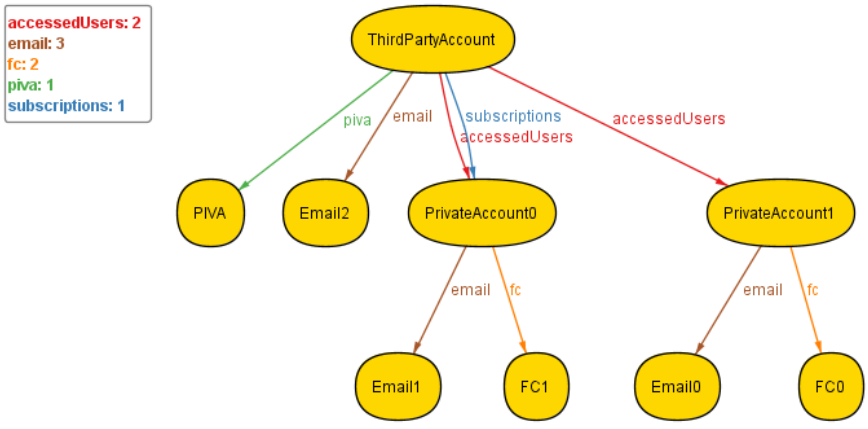


Figure 34: Alloy Static Model

4.3 Dynamic Model

As stated in the Overview, uniqueness constraints have been dropped in order to represent an entity before and after an event has occurred, in particular we're considering as events the submission, approval and refusal of a request.

All requests must specify the email of the sender, and have a status (Pending, Declined, Approved) and a boolean indicating whether the sender is subscribing to the Private Account or to the type of group request in general. The request data type is omitted since it's not relevant in this model.
SingleRequests must also include the P.IVA of the sender, and only one between the FC and email of the receiver.
GroupRequests must include the search parameter used to target a group of individuals. To make things easier, the only possible type of condition we modeled is, for example, "All the users who have a height of 160cm". The request is approved by the *System* only if at least 3 existing private users have a height of exactly 160cm, in this case we're not modeling user evolution in time, so we assume that the number of user is fixed.

The health parameters taken into consideration are Age, Heart Rate and Height, all represented as integers with a positive value.

As an addition to the first model, all Private Accounts must have an Age, Height and the last recorded value for the Heart Rate. Also, each account keeps track of the pending, approved and declined single requests, which are disjoint one another.

```
sig PrivateAccount extends Account {
	fc: one FC,
	age: one Age,
	height: one Height, //in cm
	bpm: one BPM, //heart rate
	pendingRequests: set SingleRequest,
	approvedRequests: set SingleRequest,
	declinedRequests: set SingleRequest
} {
	//all lists are disjoint one another
	#(pendingRequests & approvedRequests) = 0
	#(approvedRequests & declinedRequests) = 0
	#(pendingRequests & declinedRequests) = 0
}

sig ThirdPartyAccount extends Account {
	piva: one PIVA,
	accessedUsers: set PrivateAccount, //users that approved a request at least once
	subscriptions: set PrivateAccount //single users to whom the third party is subscribed
}



----------Facts----------
fact existentiality {
	no e: Email | no a: Account | a.email = e //all emails are linked to at least one account
	no f: FC | no p: PrivateAccount | p.fc = f //all fc are linked to at least one private account
	no p: PIVA | no t: ThirdPartyAccount | t.piva = p //all piva are linked to at least one third party
	no a1: PrivateAccount, a2: ThirdPartyAccount | a1.email = a2.email //the same email can't be used by both a private account and third party, emails should be unique but this constraint is relaxed in this model
	no a: Age | no p: PrivateAccount | p.age = a //all ages are linked to at least one account
	no h: Height | no p: PrivateAccount | p.height = h //all heights are linked to at least one account
	no b: BPM | no p: PrivateAccount | p.bpm = b //all heart rates are linked to at least one account
	no disj a1,a2 : Age | a1.value = a2.value //value of two different age entities must be different as well
}

fact reqProperties {
	//all requests contain the email of the sender, and single requests also contain the PIVA of the sender
	//the sender is not unqiue in order to represent dynamicity, since there will be an entity before and after a request has been approved/declined, and those entity have the same email and PIVA
	all r: Request | some a: ThirdPartyAccount |
		(a.email = r.fromEmail and (r in SingleRequest implies a.piva = r.fromPIVA))
}

fact singleRequestProperties {
	//all single requests contain either the email or fc of the receiver, who is unique
	all r: SingleRequest | some a: PrivateAccount | (a.email = r.toEmail or r.toFC = a.fc)

	//if a request is in one of the user's list, then that user is the receiver of that request
	all r: SingleRequest | all p:PrivateAccount | r in (p.pendingRequests + p.approvedRequests + p.declinedRequests) implies (p.email = r.toEmail or r.toFC = p.fc)

	all r: SingleRequest |
		(r.status in Approved iff one p: PrivateAccount | r in p.approvedRequests) and //if a request is approved, then there is exactly one account that has it in the approved list
		(r.status in Declined iff one p: PrivateAccount | r in p.declinedRequests) and //if a request is declined, then there is exactly one account that has it in the declined list
		((one p: PrivateAccount | r in p.pendingRequests) implies r.status in Pending) //if a request is in the pending list of one account, then it's pending (but not viceversa, it can be pending but not submitted yet)
}
```

```
open util/boolean
open util/integer

----------Signatures----------
sig Email {}
sig FC {}
sig PIVA {}

abstract sig Status {}
one sig Approved extends Status {}
//for a single request, pending means that the request is created but not in the pending list of
//the target user, or that it is in the pending list of the user, so waiting for him to approve it
//for a group request, pending means that the system has yet to evaluate it
one sig Pending extends Status {}
one sig Declined extends Status {}

//health parameter, in this case used as a search parameter and not as a requested data type
abstract sig Parameter {
value: one Int
}{value>0}

sig Age extends Parameter {}
sig Height extends Parameter {}
sig BPM extends Parameter {}

abstract sig Request {
	fromEmail: one Email, //email of the third party
	status: one Status,
	subscribing: one Bool
}

//instance of a single request, a request can be pending, and then can be accepted or declined. For each of these phases, there is a different instance of request
sig SingleRequest extends Request {
	toEmail: lone Email,
	toFC: lone FC,
	fromPIVA: one PIVA, //for single requests the target user should know the PIVA of the third party as well
} {
	#toEmail = 1 iff #toFC = 0 //the request can be done either through the email or the fc, not both of them
}

//group requests don't have to be accepted by a user, so the S2B doesn't need the email and fc of the target, neither the PIVA of the third party since it knows the email already
sig GroupRequest extends Request {
	condition: one Parameter, //condition expressed in the anonymous request, eg: "Data of all the people whose age is 30"
}

abstract sig Account {
	email: one Email
}
```

```
//a group request is approved only if there exist at least 3 private account that match the condition
fact groupRequestProperties {
	all r: GroupRequest | r.status in Approved implies
		let x = {p: PrivateAccount |
			(r.condition in Age and p.age.value=r.condition.value) or
			(r.condition in Height and p.bpm.value=r.condition.value) or
			(r.condition in BPM and p.bpm.value=r.condition.value)} |
		#x > 3 //this is a restriction, the actual value should be 1000
}

//a request is approved or declined only if it was previously pending
fact pendingRequests {
	all r: Request | r.status in (Approved + Declined) implies
		(one r': Request | r'.status in Pending and
			((r in GroupRequest and groupReqUnchanged[r,r']) or
			(r in SingleRequest and singleReqUnchanged[r,r'])))
}

fact accessAndSubscriptions {
	//a third party has access to a private account's data if and only if one of his request was accepted
	all t: ThirdPartyAccount | all p: PrivateAccount |
		p in t.accessedUsers iff  (some r: SingleRequest |
				(r.fromPIVA = t.piva and r.fromEmail = t.email) and
				r in p.approvedRequests)

	//a third party is subscribed to a private account if and only if his request of subscription was accepted
	all t: ThirdPartyAccount | all p: PrivateAccount |
		p in t.subscriptions iff (some r: SingleRequest |
				(r.fromPIVA = t.piva and r.fromEmail = t.email) and
				(r.subscribing in True) and
				(r in p.approvedRequests))
}



----------Predicates----------
//predicate used to check that two group requests haven't changed
pred groupReqUnchanged[r,r': GroupRequest] {
	r'.condition = r.condition
	r'.fromEmail = r.fromEmail
	r'.subscribing = r.subscribing
}

//predicate used to check that two single requests haven't changed, apart from the status
pred singleReqUnchanged[r,r': SingleRequest] {
	r'.toEmail = r.toEmail
	r'.fromEmail = r.fromEmail
	r'.fromPIVA = r.fromPIVA
	r'.toFC = r.toFC
	r'.subscribing = r.subscribing
}
```

```
//predicate used to check that two private accounts generalities haven't changed
pred accUnchanged[a,a': PrivateAccount] {
	a'.fc = a.fc
	a'.email = a.email
}

//accept a group request, thus defining a world in which the conditions expressed in groupRequestConditions hold
pred acceptGroupRequest[r,r': GroupRequest] {
	//precondition
	r.status in Pending

	//postcondition
	groupReqUnchanged[r,r']
	r'.status in Approved
}

//decline a group request, thus defining a world in which the conditions expressed in groupRequestConditions do not hold
pred declineGroupRequest[r,r': GroupRequest] {
	//precondition
	r.status in Pending

	groupReqUnchanged[r,r']
	r'.status in Declined
}

//add a pending single request
pred addSingleRequest[r: SingleRequest, a,a': PrivateAccount] {
	//precondition
	r not in a.pendingRequests

	//postcondition
	a'.fc = a.fc
	a'.email = a.email
	a'.pendingRequests = a.pendingRequests + r
	a'.approvedRequests = a.approvedRequests
	a'.declinedRequests = a.declinedRequests
}

//accept a pending single request
pred acceptSingleRequest[r,r': SingleRequest, a,a': PrivateAccount] {
	//precondition
	r in a.pendingRequests
	r' not in (a.approvedRequests + a.declinedRequests + a.pendingRequests)

	//postcondition
	singleReqUnchanged[r,r']
	accUnchanged[a,a']
	a'.pendingRequests = a.pendingRequests - r
	a'.approvedRequests = a.approvedRequests + r'
	a'.declinedRequests = a.declinedRequests
}
```

```
//decline a pending single request
pred declineSingleRequest[r,r': SingleRequest, a,a': PrivateAccount] {
	//precondition
	r in a.pendingRequests
	r' not in (a.approvedRequests + a.declinedRequests + a.pendingRequests)

	//postcondition
	singleReqUnchanged[r,r']
	accUnchanged[a,a']
	a'.pendingRequests = a.pendingRequests - r
	a'.approvedRequests = a.approvedRequests
	a'.declinedRequests = a.declinedRequests +r'
}

//add a pending request and then accept it
pred showSingleAccept[r,r': SingleRequest, a,a',a'': PrivateAccount, t: ThirdPartyAccount] {
	addSingleRequest[r, a,a']
	acceptSingleRequest[r,r',a',a'']
}

//add a pending request and then decline it
pred showSingleDecline[r,r': SingleRequest, a,a',a'': PrivateAccount, t: ThirdPartyAccount] {
	addSingleRequest[r, a,a']
	declineSingleRequest[r,r',a',a'']
}



----------Assertions----------
//a third party is subscribed to a user if he also accessed that user, at least once
assert subscriptionIfAccess {
	all t: ThirdPartyAccount | all p: t.subscriptions | p in t.accessedUsers
}



check subscriptionIfAccess
run declineGroupRequest for 4 but 0 SingleRequest, 1 ThirdPartyAccount, 2 GroupRequest
run acceptGroupRequest for 5 but 0 SingleRequest, 1 ThirdPartyAccount, 2 GroupRequest
run showSingleAccept for 4 but 0 GroupRequest, 1 ThirdPartyAccount,  2 SingleRequest
```

```
Alloy Analyzer 4.2_2015-02-22 (build date: 2015-02-22 18:21 EST)

Warning: JNI-based SAT solver does not work on this platform.
This is okay, since you can still use SAT4J as the solver.
For more information, please visit http://alloy.mit.edu/alloy4/

Warning: Alloy4 defaults to SAT4J since it is pure Java and very reliable.
For faster performance, go to Options menu and try another solver like MiniSat.
If these native solvers fail on your computer, remember to change back to SAT4J.

Executing "Check subscriptionIfAccess"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   5717 vars. 315 primary vars. 13998 clauses. 227ms.
   No counterexample found. Assertion may be valid. 86ms.

Executing "Run declineGroupRequest for 4 but 0 SingleRequest, 1 ThirdPartyAccount, 2 GroupRequest"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   5762 vars. 360 primary vars. 13723 clauses. 75ms.
   Instance found. Predicate is consistent. 81ms.

Executing "Run acceptGroupRequest for 5 but 0 SingleRequest, 1 ThirdPartyAccount, 2 GroupRequest"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   8153 vars. 516 primary vars. 19205 clauses. 141ms.
   Instance found. Predicate is consistent. 71ms.

Executing "Run showSingleAccept for 4 but 0 GroupRequest, 1 ThirdPartyAccount, 2 SingleRequest"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   6056 vars. 424 primary vars. 12926 clauses. 66ms.
   Instance found. Predicate is consistent. 86ms.

4 commands were executed. The results are:
   #1: No counterexample found. subscriptionIfAccess may be valid.
   #2: Instance found. declineGroupRequest is consistent.
   #3: Instance found. acceptGroupRequest is consistent.
   #4: Instance found. showSingleAccept is consistent.
```

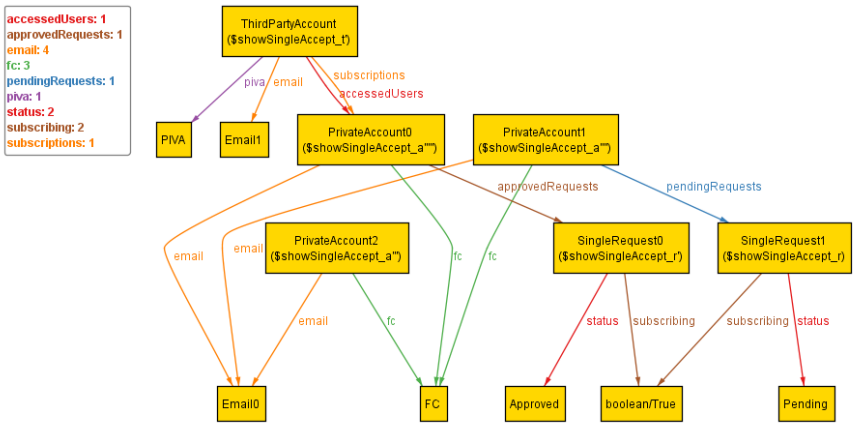


Figure 35: Approved Single Request

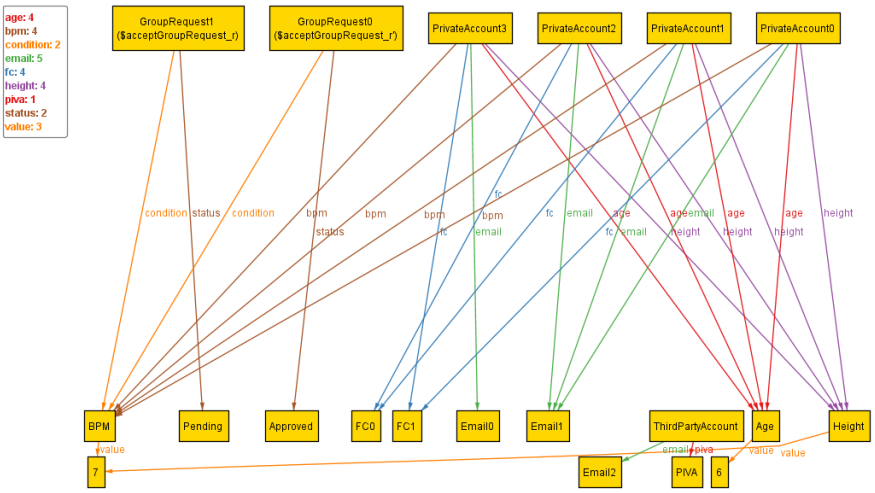


Figure 36: Approved Group Request

5 Efforts

In the following tables we will summarize the effort spent by each member of the team on the *RASD* Document.

Date	Task	Hours
12/10/18	Git setup	2
19/10/18	Pre-analysis	2

24/10/18	Complete Part 1 and organize following work	2
25/10/18	Product Functionalities and User Characteristics	2
29/10/18	Software, Hardware, Communication Interfaces	1
30/10/18	User Interfaces	2
03/11/18	Use Cases	4
04/11/18	Latex setup	2
05/11/18	Review Meeting	3
06/11/18	Use Case Diagrams, Scenarios	1,5
08/11/18	Track4Run Use Cases	2
09/11/18	Goals, Requirements and Domain Assumptions Revision	3
10/11/18	Copy on Latex	5
11/11/18	Final Revision	3
		Total
		36,5

Table 16: Virginia Negri's effort