POLITECNICO DI MILANO

POLITECNICO
MILANO 1863

SOFTWARE ENGINEERING 2 COURSE

The TrackMe project

Requirements Analysis and Specification Document

Kevin Mato, Antonio Mazzeo

Github: https://github.com/antoniomazzeo/MatoMazzeo

16/12/2018

Summary

1 Introduction

This section will explain which the main scopes of the TrackMe system is and will provide a general overview of all the features involved.

1.1 Purpose

This document is the RASD (Requirement Analysis and Specification Document) and its main purpose is to describe the TrackMe system, its components, constraints, functional and non-functional requirements and relationships with third party software.

This document will also provide different scenarios along with typical user use cases and a formal description of application features through alloy language.

This document is aimed at giving a detailed system specification to developers, testers and quality assurance.

1.2 Scope

The scope of the entire project is to develop a new mobile service, TrackMe, composed of a core service invisible to the end users, Data4Help, compatible with the most popular smartwatches and bands available in the market which stores and shares health and location data with third party services.

This document will particularly focus on the interaction with AutomatedSOS, a software that will be built on top of Data4Help, aimed at helping people by contacting an ambulance in case of emergency (i.e. heart beat under or above the safe parameters).

Data4Help must share and store user data only by explicit consent respecting the General Data Protection Regulation (GDPR).

AutomatedSOS must constantly monitor user's health, if the values go outside the safe intervals given by the WHO (World Health Organization), it must contact an ambulance.

This RASD (Requirement and Analysis Document) aims at describing the domain in which our system is going to work and determine the main use cases for this system.

This document will also define the primary users of the system, why the system is developed, the condition in which this is going to operate. Moreover, we will illustrate the functional and non-functional requirements for a better understanding of the system.

The **goals** achieved by this solution are:

[G1] Third parties can analyze users individually or in groups.

[G2] Third parties are constantly up-to-date on new data about the users.

[G3] Users can decide whether to share or not their location and health data with the system.

[G4] Users must receive automatically help if there is an anomaly in their health parameters.

[G5] The user can be recognized by providing a form of identification.

1.3 Definitions, Acronyms, Abbreviations

Definitions

- Platform: system/application as a whole.
- User: An end user who is currently registered to the Data4Help application and has credentials to access.
- Guest: Person not yet registered and with limited access to features.
- Framework: Reusable set of libraries or classes for a software system.
- Cross-Platform: software able to run on different platforms.
- Partner: A third-party system using Data4Help data.
- Health data: User's heart beat data.

Acronyms

- RASD: Requirements Analysis and Specification Document
- DB: Database
- DBMS: Database Management System
- OS: Operating System
- HTML: Hypertext Markup Language
- CSS: Cascading Style Sheets
- JS: JavaScript
- JSON: JavaScript Object Notation
- API: Application Programming Environment
- HTTP: Hypertext Transfer Protocol
- HTTPS: Hypertext Transfer Protocol Secure
- TCP: Transmission Control Protocol

Abbreviations

- [Gn]: n-th goal
- [Rn]: n-th functional requirement
- [Dn]: n-th domain assumption
- [UCn]: n-th use case

1.4 Revision History

Version, date and summary

Version	Date	Summary
1.0.0	11/11/2018	First Release
2.0.0	16/12/2018	Second Release added edit permission menu on the mock-up and improvement of alloy model. Corrected an error on UC4

1.5 Reference Documents

The following documents were used:

1. Assignment document: https://beep.metid.polimi.it/documents/121843524/3f744351-7378-4162-86b0-45eddaf10713
2. IEEE Std 1016tm-2009 Standard for Information Technology-System Design-Software Design Descriptions.
3. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.

1.6 Document Structure

The document is divided in the following sections:

- **Section 1:** General description, overview and goals of the TrackMe applications (Data4Help and AutomatedSOS).
- **Section 2:** Overall description of the software features and implemented functions. Description of world and shared phenomena, constraints, domain assumptions and dependencies.
- **Section 3:** Characterization of functional and not functional requirements.
- **Section 4:** Formal model of the main aspects of the application using Alloy modelling language.
- **Section 5:** Effort spent for writing the project.
- **Section 6:** References and software used for the creation of the system.

2 Overall Description

2.1 Product Perspective

Both Data4Help and AutomatedSOS will be cross-platform web application which will be able to run on every mobile device.

AutomatedSOS will gather data from Data4Help through custom API.

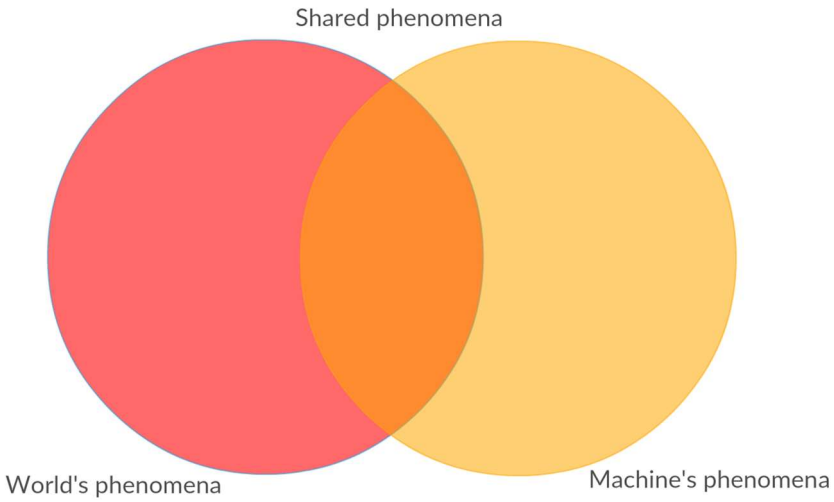


Figure 2.1.1 Relation between world and machine phenomena

World is everything that is outside the system scope and **machine** is the opposite. Therefore, **world's phenomena** are all the external events happening in the outside world and **machine's phenomena** are events related to the system. **Shared phenomena** are the intersection of the two i.e. events observable by both parts.

What can be a **world phenomenon**?

For instance:

1. The weather for a certain day is extremely hot, accompanied by a high humidity in the air.
2. The diet followed by the user is rich in unsaturated fats.
3. Emergency office getting help calls.

What can be a **machine phenomenon**?

1. The registration of a new user in the database.
2. A thread spins up to save an event, after the computation of an anomaly.
3. Database queries.

What can be a **shared phenomenon**?

1. Locating a user.
2. Notifying an ambulance with the address of a user.
3. User's health data.

2.2 Product Function

Our system should be very simple to use by everyone. The third parties should also have data about single and group of users respecting the constraints (see section 2.4).

The system should also be able to:

1. Let the user register and log in.
2. Ask users for their data on behalf of third parties.
3. Let a user choose and edit his privacy preferences.
4. Make itself easy-to-use to everyone.
5. Compute aggregations of users according third parties' requests.
6. Give an evaluation of the data.
7. Notify a third party if certain data are not available and why.
8. Interact with sharing services.
9. Send emergency help to users in danger.
10. Avoid any privacy misuse of data.
11. Understand and store location and health status of a user.
12. Can identify people as unique entities thanks to the information they provided.

2.3 User Characteristics

To use the application, no special skills are required.

The actors in the system are:

- **Guests**: People not registered yet. Guests cannot access any application functionality apart sign up.
- **Users**: People currently registered to the system, they can log in and access full service with no restriction.

The **users**, in the perspective of Data4Help, can be of two types:

- **Third parties:** They have higher privileges in order to perform queries and access data through custom APIs.
  Third parties can request an account contacting directly Data4Help providing a legit VAT identification number.
- **The normal user**: Final user that can perform normal tasks (see section 3).

From AutomatedSOS perspective we have just **normal users**.

2.4 Domain assumptions, dependencies and constraints

Constraints

1. **Hardware limitations:** The client application under study must run on every mobile device as smartphones and tablets, regardless of the OS (i.e. Android, iOS, …).
   For AutomatedSOS the user must have a smartwatch or smartband with heart beat monitor capabilities. Unfortunately, there's no standard API to gather information from such devices, so only the most famous companies' devices will be supported (Fitbit, Samsung, Garmin, Xiaomi, Sony).
2. **Interface to other applications:** As said in the previous point, the application needs to interface itself with other third-party applications to read health data from the user.
3. **Parallel operations:** The application must be able to handle not blocking parallel requests from several users grating a high reactivity.
4. **Availability limitations:** The system's uptime and availability must be around 99.999%, corresponding to a maximum downtime of 5 minutes in the whole year, since the application deals with people's lives.
5. **Privacy constraints:** The application must only handle data that the user previously accepted to share.
   Moreover, a third-party can access data of single or group of users only if:
   1. The single user accepts the third-party request.
   2. The cardinality of the group of users fitting the given criteria is greater or equal than 1000.
6. **SOS constraint:** AutomatedSOS must contact the emergency office in less than 5 second.

Domain assumptions and Dependencies

[D1] **Health-smart device**.
All the users have a device that collects health data.

[D2] **GPS devices.**
All users have a device that possesses a built-in GPS functionality.

[D3] **Connectivity**.
All the devices that take part to the system, have a stable connection.

[D4] **Veracity of the data.**
All the information about the users are correct.

[D5] **Well-rounded algorithms.**
The algorithms for anomaly detection are infallible.

[D6] **Help availability in time.**
Ambulances can be contacted anytime.

[D7] **Help Range.**
Ambulances are spread on the areas of the users of the system in the most optimized way.

[D8] **Data Mining possibilities.**
Users give all the personal information needed to be clustered in groups thanks to different objective functions.

[D9] **Privacy Regulations.**
There are strict regulations about privacy.

[D10] **Big Data oriented.**

All the information is structured.

[D11] **There are no privileged users apart third parties.**

[D12] **Every user is independent from each other.**

[D13] **Every user has several contact info.**

[D14] **Timestamps sharing.**
Every agent in the system share the same time.

[D15] **API Availability.**
Whatever will interact with the system has available API.

[D16] **Battery Duration.**
Every user keeps its devices charged continuously.

[D17] **OS Permissions Granted.**
The user will always grant to his OS's device the permission to access to all the needed services.

[D18] **Sharing services availability.**
The sharing services interacting with the system have a greater uptime than the system itself.

[D19] **Host Availability.**
The system has a greater uptime than the single clients.

[D20] **Sim credit status.**
The user has always enough credit to send at least a SMS

3 Specific Requirements

3.1 External Interfaces Requirements

User Interface

The user interface must be user-friendly to guarantee to the final user an easy way to interact with the application. The development of the front end is realized through HTML and CSS to provide a wide range of combability among devices.

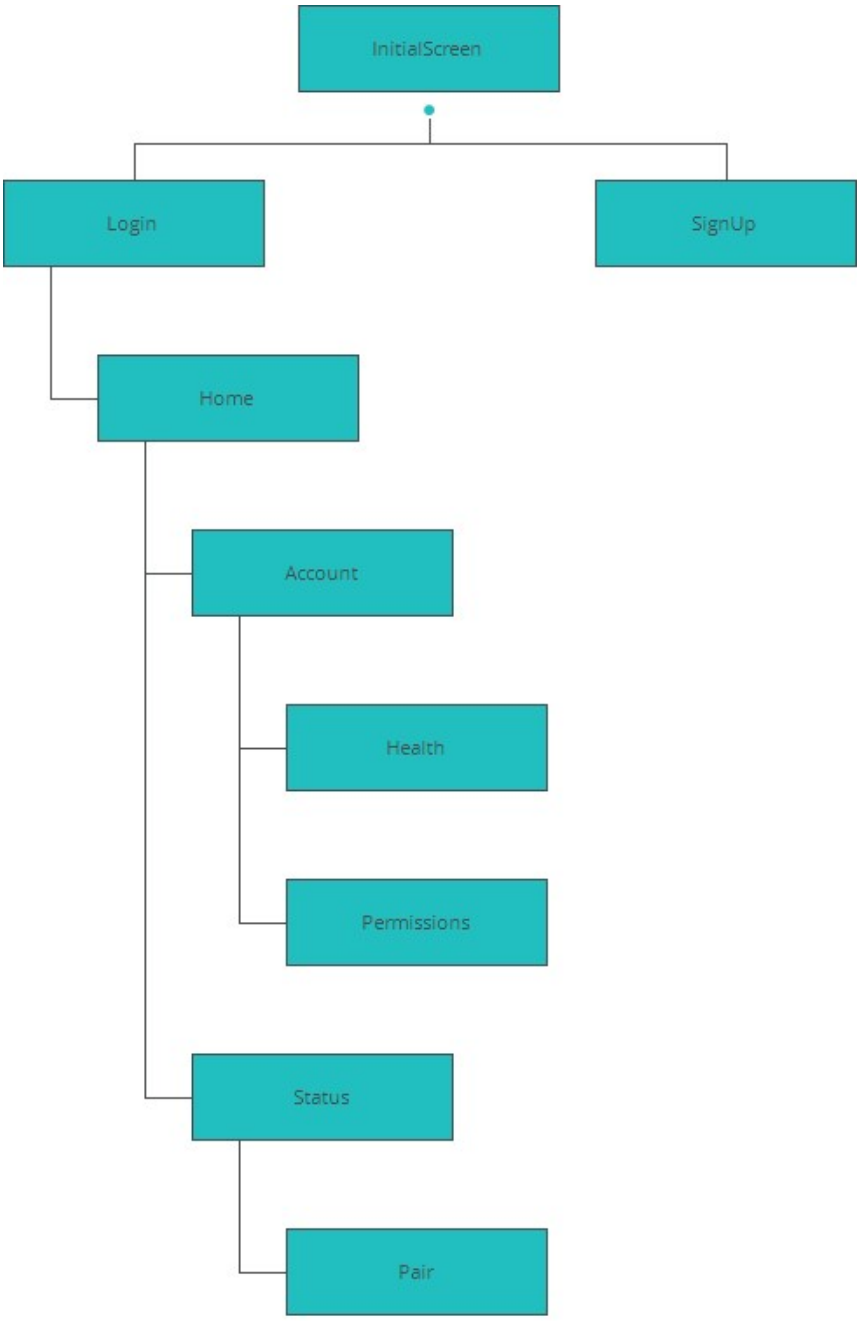


Figure 3.1.1 App Map

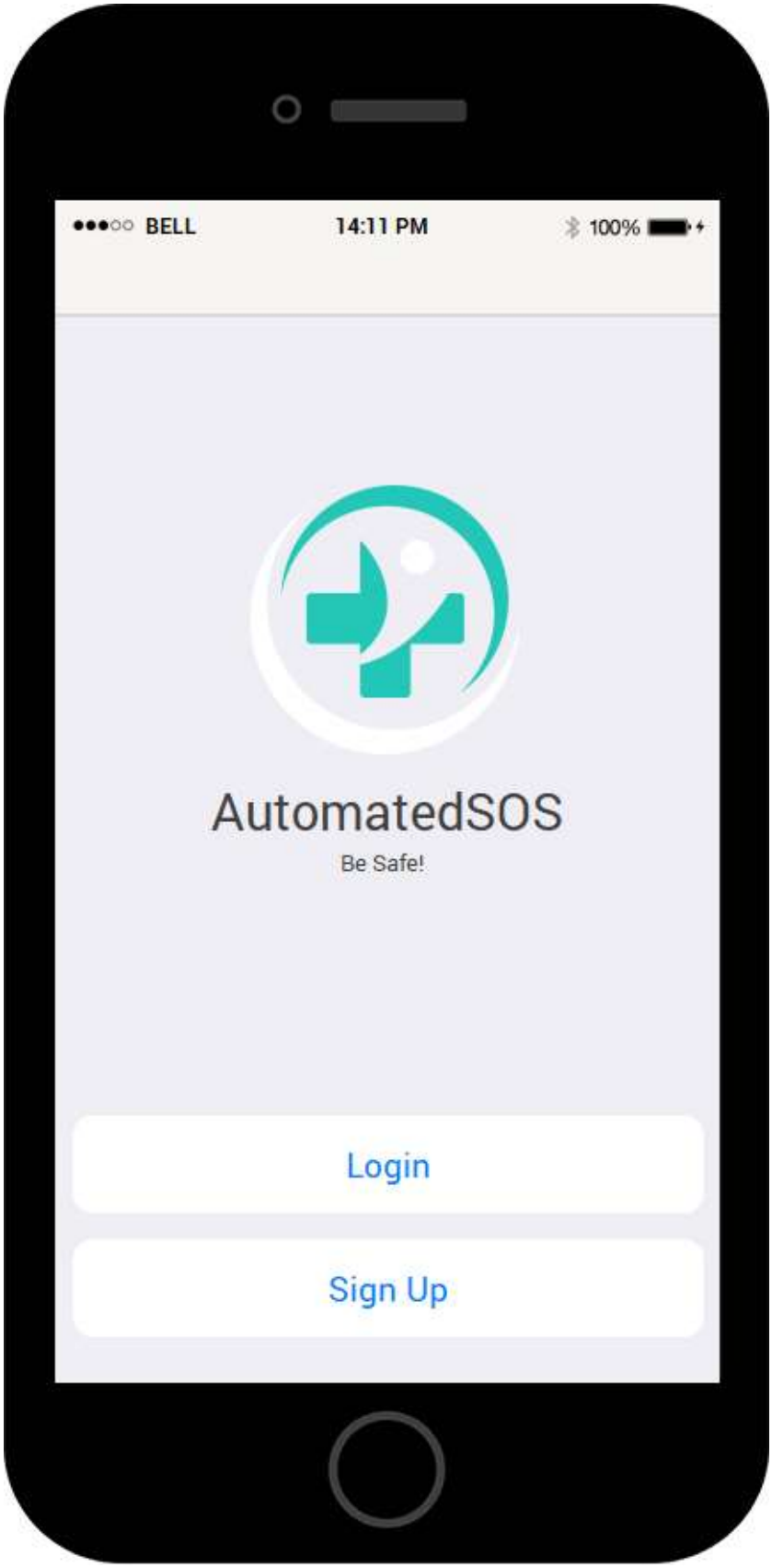


Figure 3.1.2 InitialScreen

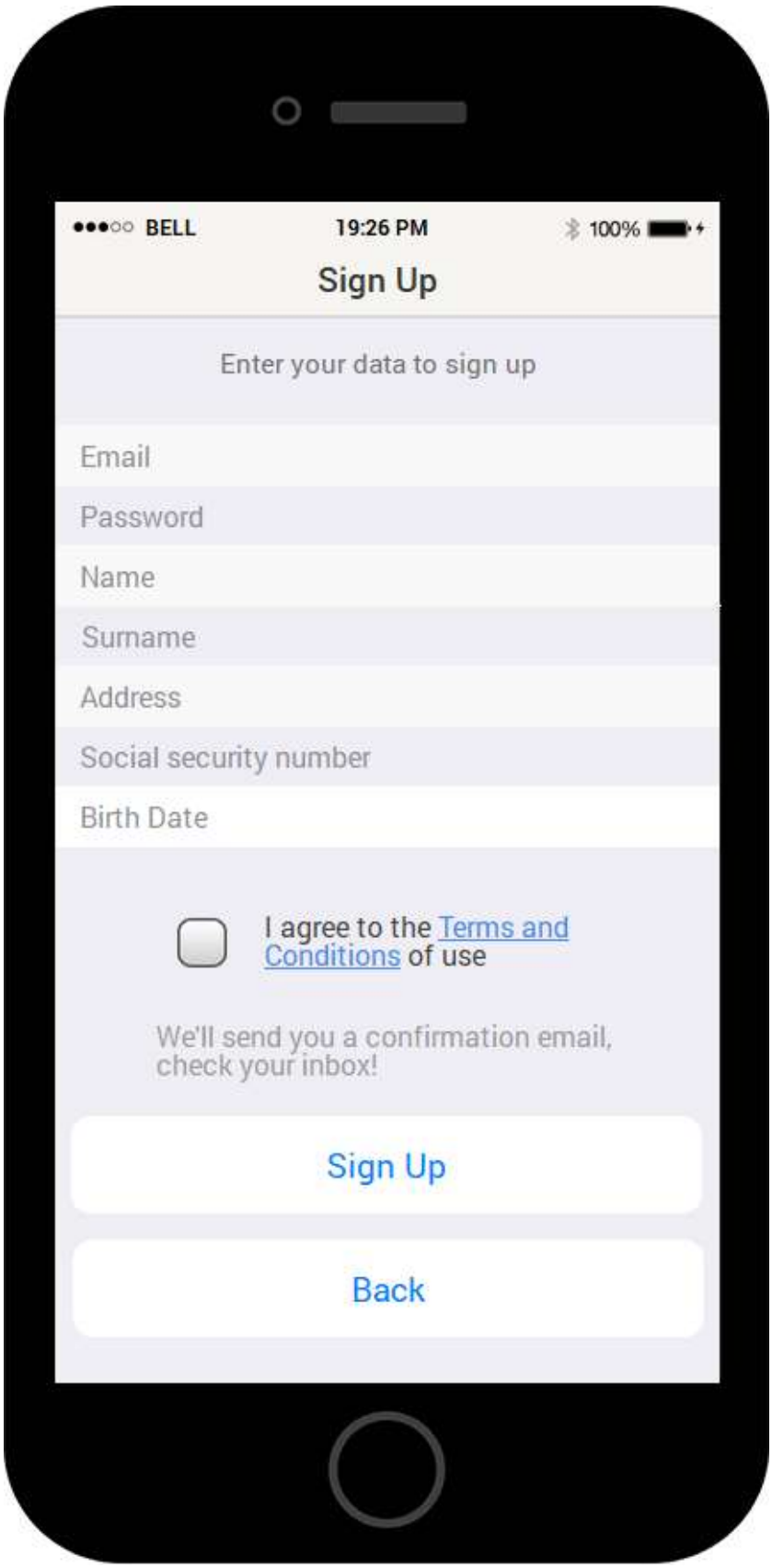


Figure 3.1.3 SignUp

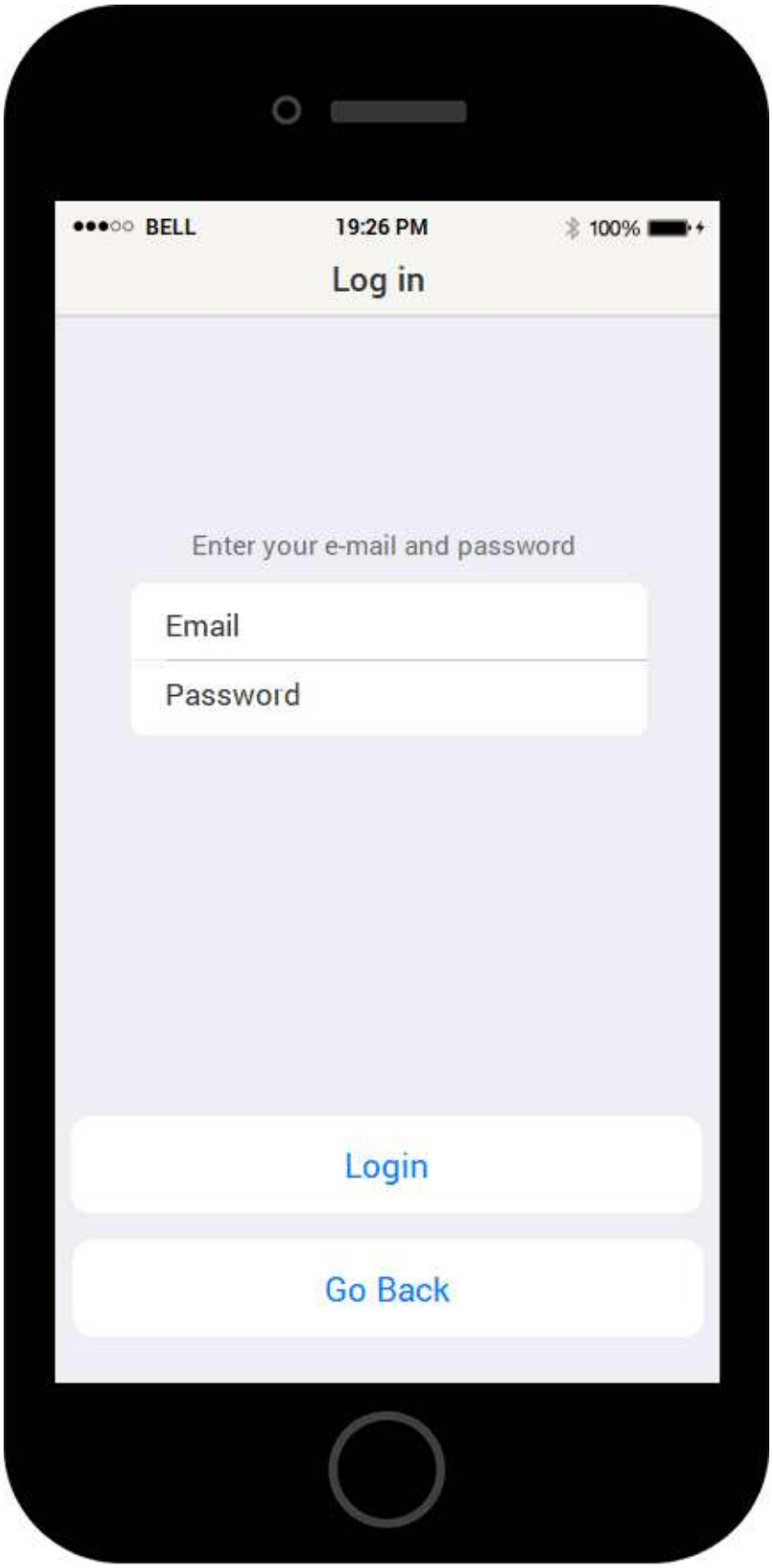


Figure 3.1.4 Login

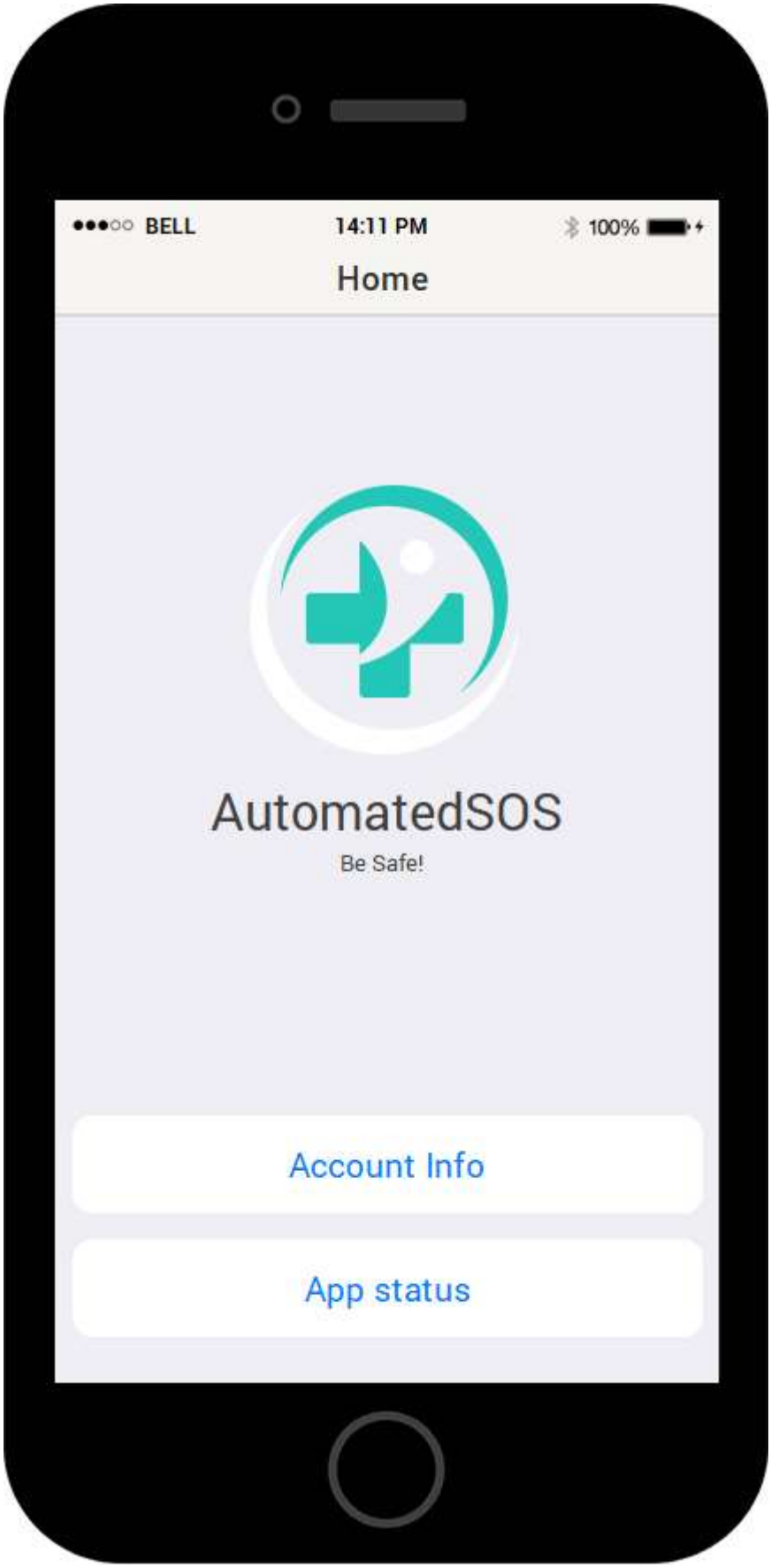


Figure 3.1.5 Home

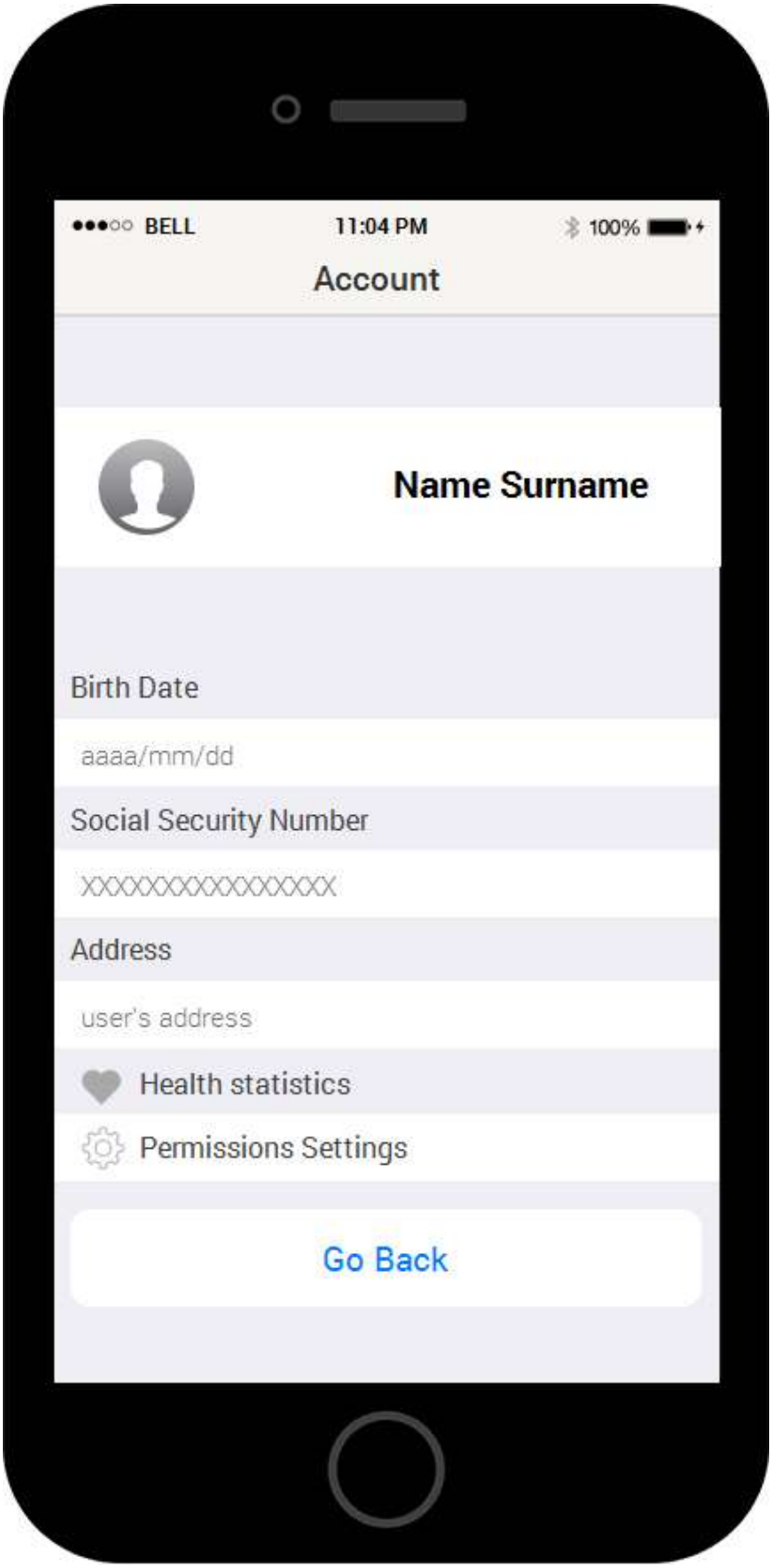


Figure 3.1.6 Account

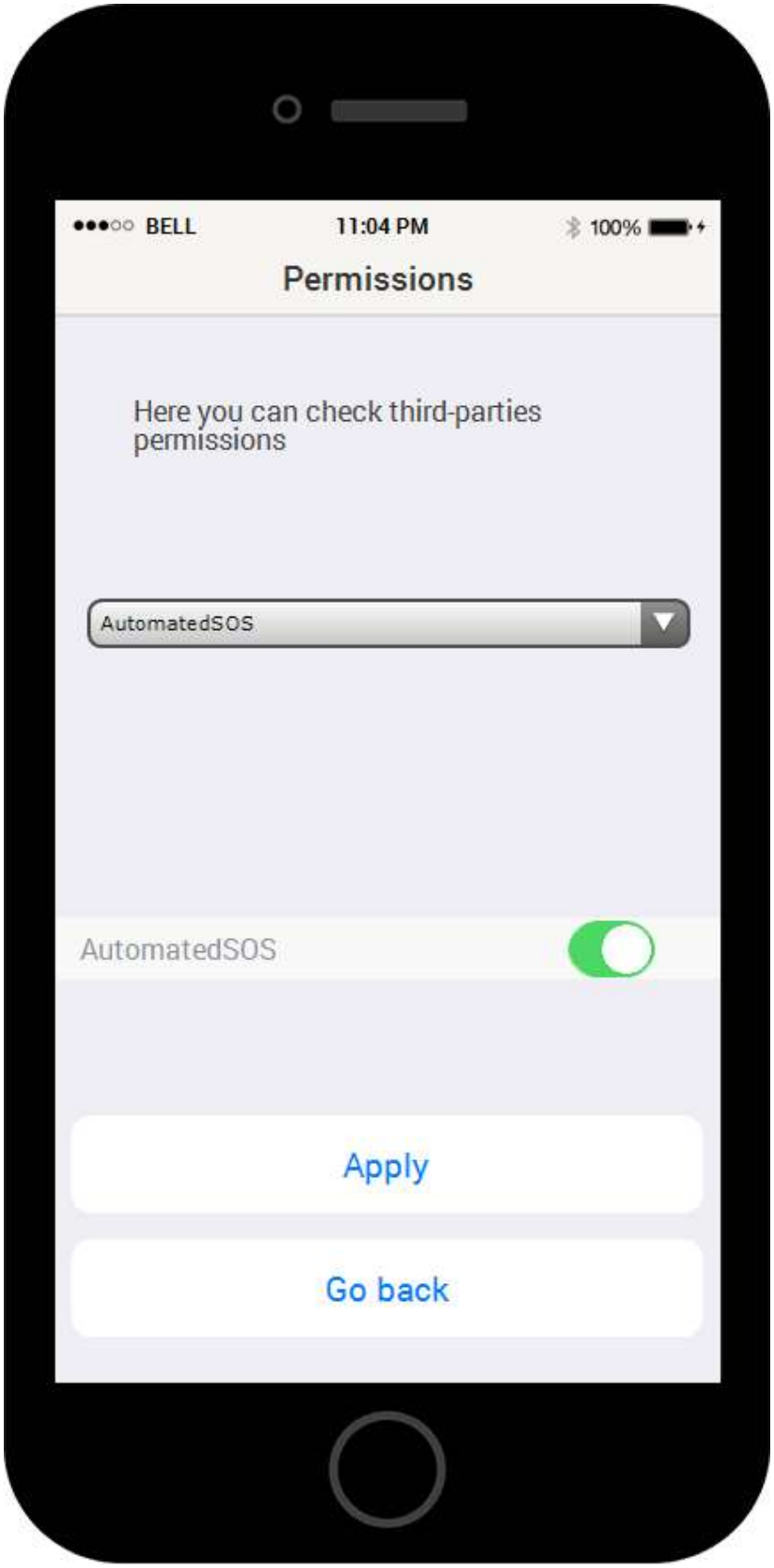


Figure 3.1.7 Permissions

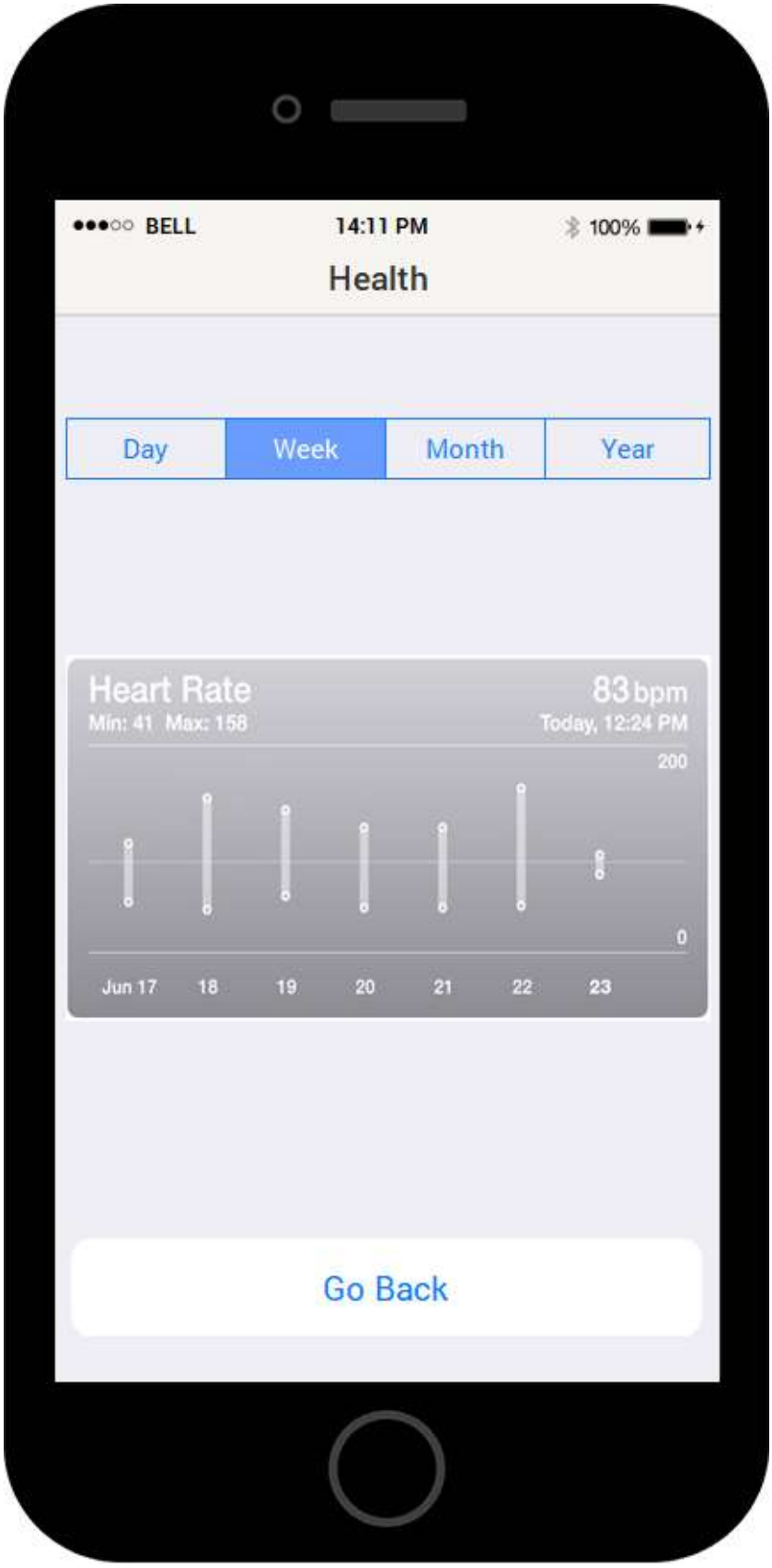


Figure 3.1.8 Health

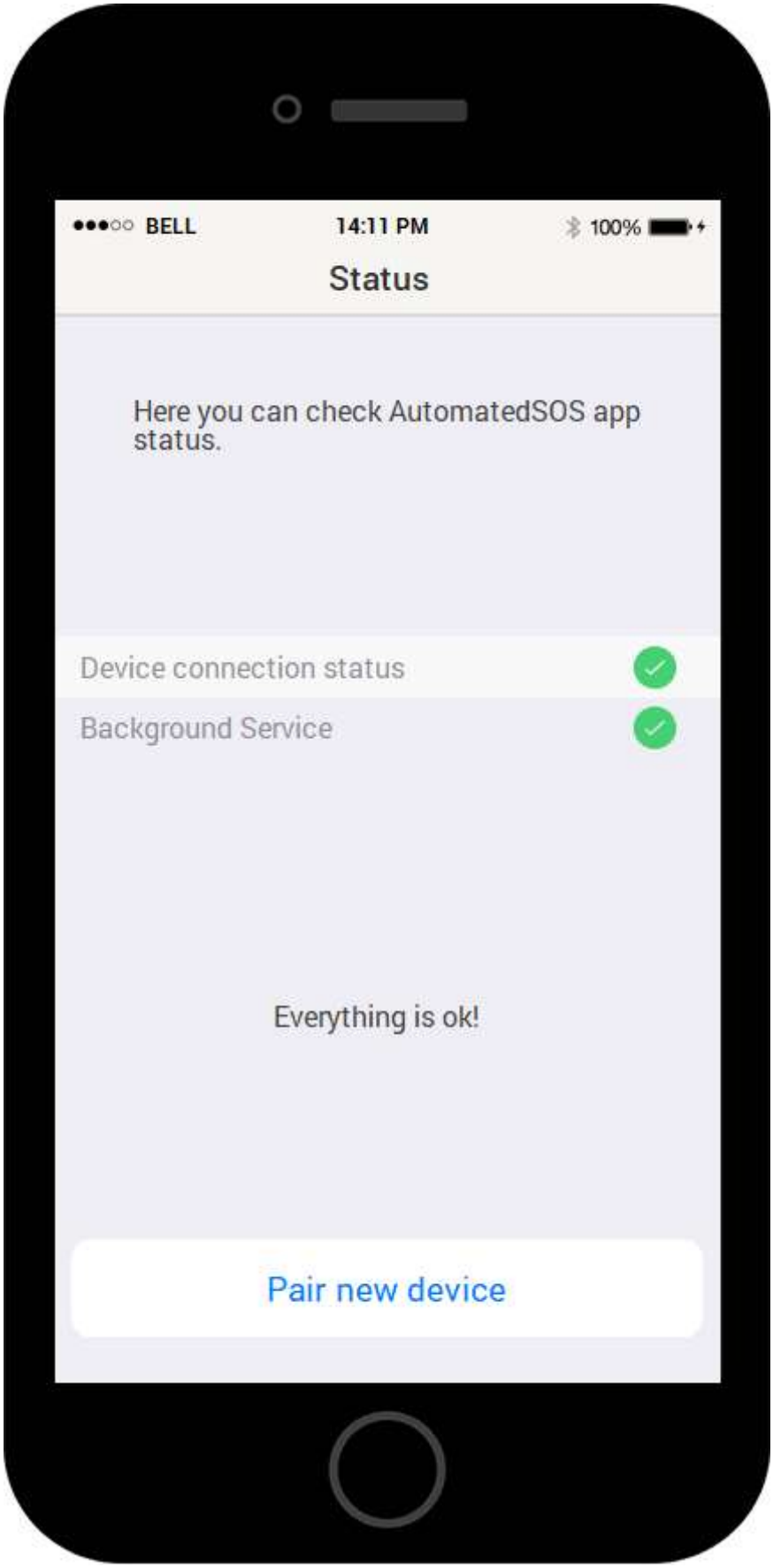


Figure 3.1.9 Status

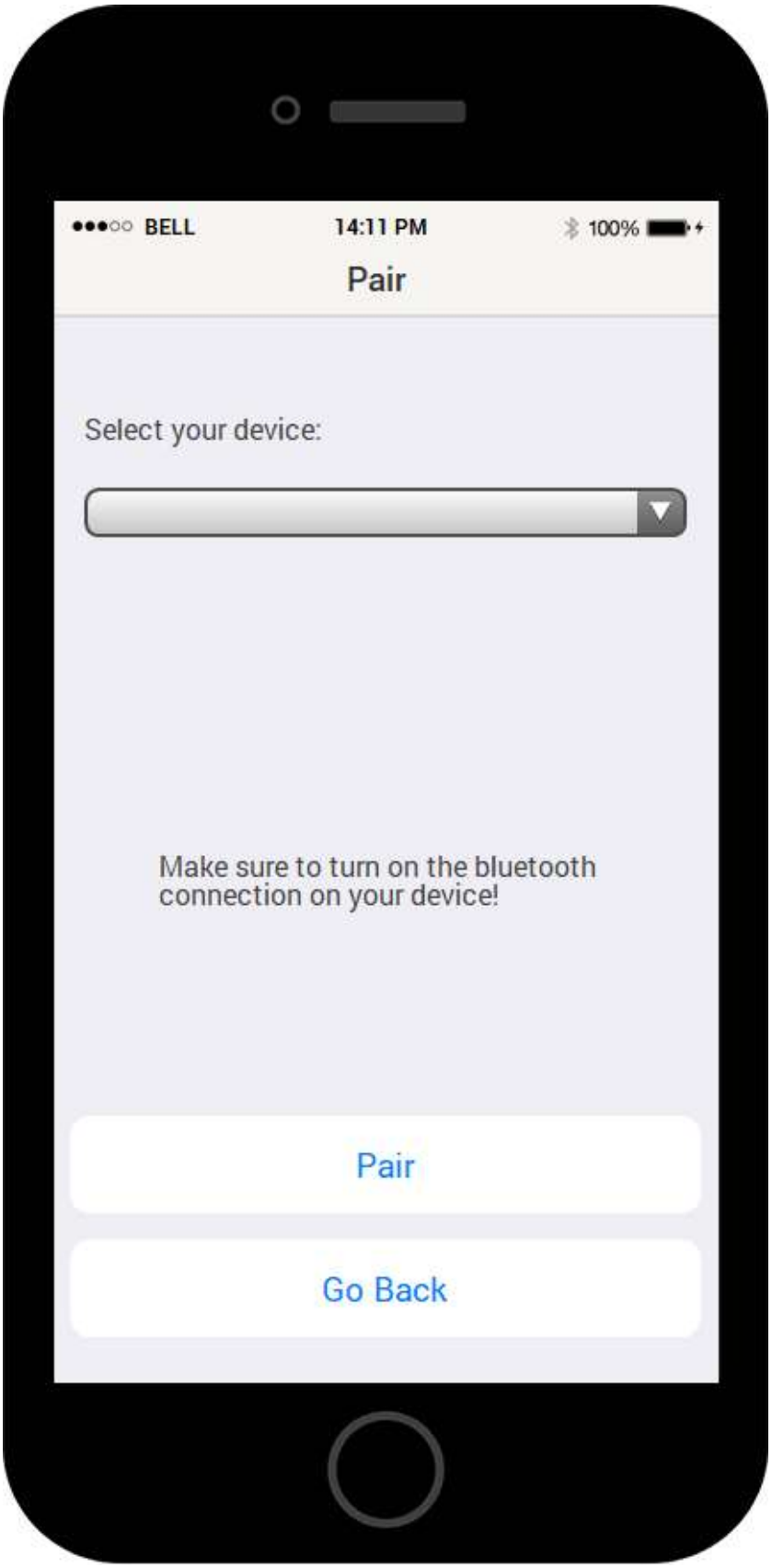


Figure 3.1.10 Pair

Fig. [3.1.2]

Welcome screen, the user can login if it was already registered.

Fig. [3.1.3]

Sign up screen, here the user can register to the system.

The user can see a detail of privacy, data shared and conditions of use clicking the underlined link.

Fig. [3.1.4]

Log in screen, the user must insert his credentials to log in.

Fig. [3.1.5]

Home screen, this is the main entry point to all app's functionalities.

Fig. [3.1.6]

Account screen, here the user can review his own data.

Fig. [3.1.7]

Permission screen, here the user can review third parties' permissions.

Fig. [3.1.8]

Health screen, here the user can review his own heart rate statistics ordered by time.

Fig. [3.1.9]

Status screen, the user can view the app status to see if the app is properly working and if the health monitoring device is connected.

Fig. [3.1.10]

Pair screen, the user can pair a new health device with the application.

N.B.

The end user, at first time access, must complete a tutorial step in which the app will guide him/her to connect to the health monitor device he/she owns.

Software Interface

The system will use the following programming, markup and representation languages:

- JavaScript
- HTML
- CSS
- Python
- SQL
- JSON

Communication Interface

Communication between client and server happens through HTTPS by TCP protocol using the standard port 443.

The system must preserve user's privacy, all the communications involving sensitive data must be held in a crypted channel.

3.2 Functional Requirements

The user must be able to:

1. **[G1]** Third parties can analyze users individually or in groups.
   Since we are assured that
   - [D1]
   - [D2]
   - [D3]
   - [D4]
   - [D8]
   - [D9]
   - [D11]
   - [D12]
   - [D15]
   - [D18]

   hold, the system must:
   - [R1] Allow the third part to search by social security number or group
   - [R2] Display a summary dashboard
   - [R3] Alert the third part whether the target user accepted the request or not in case of search by SSN
   - [R4] Alert the third part whether the target user accepted the request or not in case of search by group (The third part should enlarge the search criteria in case of the constraints are not met)

2. **[G2]** Third parties are constantly up-to-date on new data about the users.
   Since we are assured that
   - [D1]
   - [D2]
   - [D3]
   - [D4]
   - [D10]
   - [D11]
   - [D14]
   - [D15]
   - [D16]
   - [D18]
   - [D19]

   hold, the system must:
   - [R5] check and save user's location and health
   - [R6] share locations and health data with third parties involved continuously

3. **[G3]** Users can decide whether to share or not their location and health data with the system.
   Since we are assured that
   - [D3]
   - [D9]
   - [D13]

   hold, the system must:
   - [R7] allow the user to accept or not to share his own data
   - [R8] alert the user whether a third part is requesting to access his own data specifically, then allow the user to accept or deny the request

4. **[G4]** Users must receive automatically help if there is an anomaly in their health parameters.
   Since we are assured that
   - [D1]
   - [D2]
   - [D3]
   - [D4]
   - [D5]
   - [D6]
   - [D7]
   - [D12]
   - [D13]
   - [D14]
   - [D15]
   - [D16]
   - [D17]
   - [D18]
   - [D19]
   - [D20]

   hold, the system must:
   - [R9] know if a communication is effective
   - [R10] recognize the occurrence of a health problem analyzing the data collected
   - [R11] Contact the emergency office in case of a health problem continuously until a confirmation arrives

5. **[G5]** The user can be recognized by providing a form of identification.
   Since we are assured that:
   - [D3]
   - [D4]
   - [D12]

   hold, the system must:
   - [R13] provide a sign-up functionality
   - [R14] provide the user a log in functionality, the user must be able to type a combination of id and password that matches an account

3.2.1 Scenarios

AutomatedSOS:

Scenario 1

Susan decides to download AutomatedSOS app from the store after a commercial got her eye, out of curiosity she installs it, signs up and tries it out.

She thinks that it would be a great app for her beloved grandfather John, he lives alone and he needs help in case of emergency, so she helps him installing the app and setting it right.

Scenario 2

John, Susan's grandfather, after some time using AutomatedSOS, breaks his smartwatch. He goes to the store, buys a new one and pairs the new device with the app.

Scenario 3

Mark is a good athlete, he trains really hard every day. Unfortunately, his father's family has heart attack history, even though Mark is perfectly healthy, he wants to monitor his health anyway. He hears from a close friend about AutomatedSOS, so he goes to the store, downloads and installs it.

After some time, he wants to check his health statistics, so he goes into his account information and checks it.

Scenario 4

Margaret has already AutomatedSOS app up and running.

It's a really hot day and Margaret is alone at home and her husband went to buy something to the store. Her heart rate goes suddenly outside the safe parameters, AutomatedSOS is aware of the anomaly and sends a SMS to the emergency office.

Examples of interactions with other third-party software:

Scenario 5

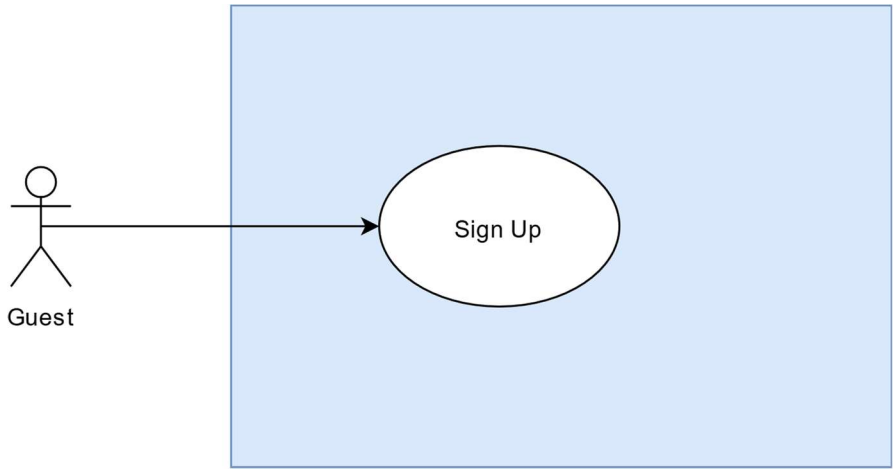
Track4Run is a third-party software connected with Data4Help, it's capable of tracking athletes participating in a run, it allows organizers to define the path for the run, participants to enrol to the run and spectators to see on a map the position of all runners during the run.

Scenario 6

LiveStatistics is another third-party software connected with Data4Help, it can track health data across the globe giving live estimations for each country.

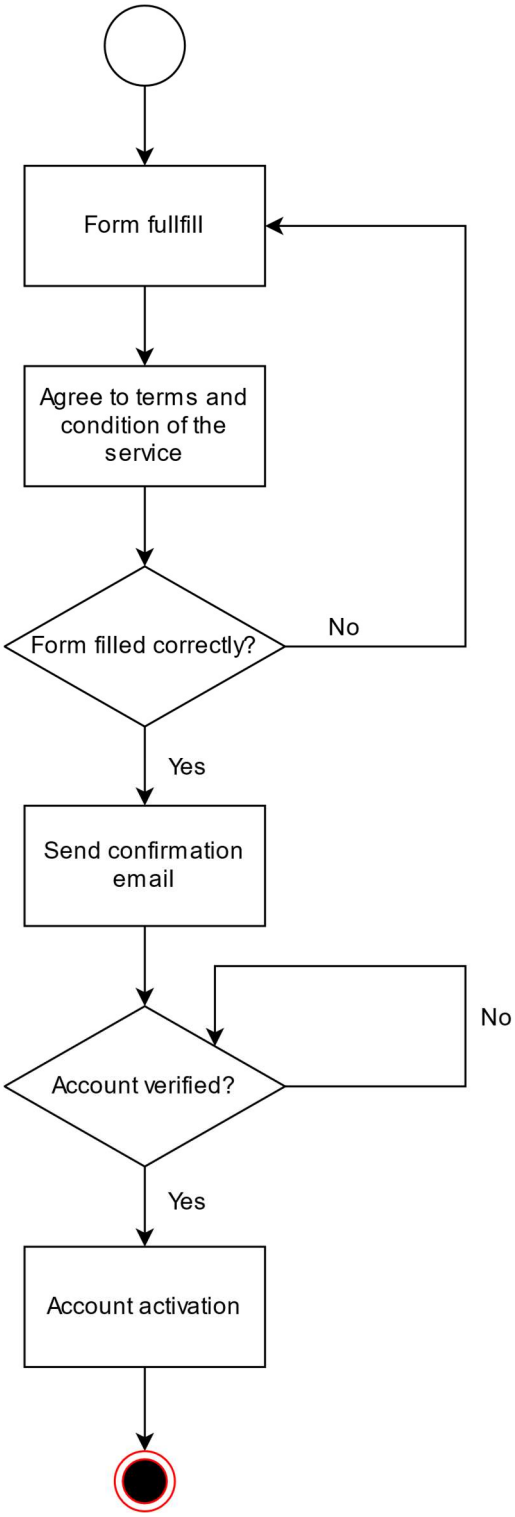
3.2.2 Use Case

[UC1] Sign Up:



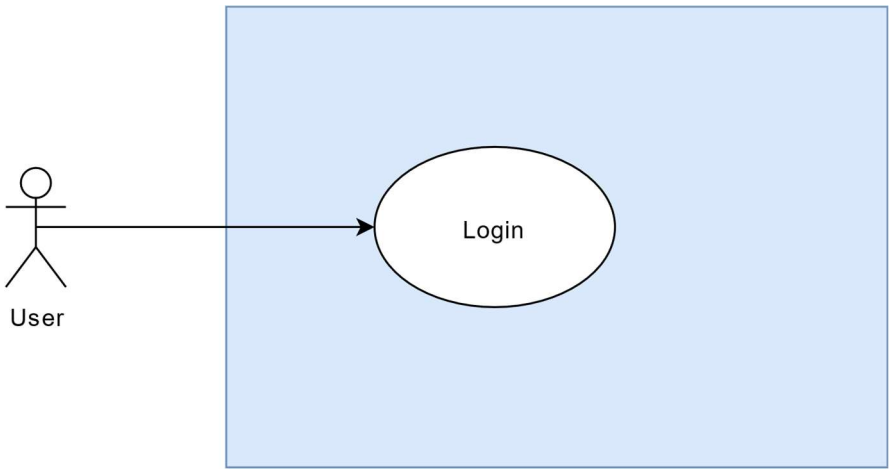
[UC1] Sign Up

Name	Sign Up
Actor	Guest
Entry Condition	The actor hasn't got any account on the system
Goal	G3, G5
Event Flow	1. The guest downloads and installs the application 2. The guest opens the application 3. The guest clicks the "Sign Up" button (See Fig.3.1.2) 4. The guest fills the provided form (See Fig.3.1.3) and click the "Sign Up" button 5. The guest opens the confirmation email and clicks on the attached link
Exit condition	The actor becomes a user of the system and can use the application
Exceptions	Some form fields get incorrectly filled or left empty, the system notifies the user on the problem and highlights what's wrong, the user needs to fill or correct the selected items. If the user provides a wrong email address, he/she must redo the sign-up process from the beginning.



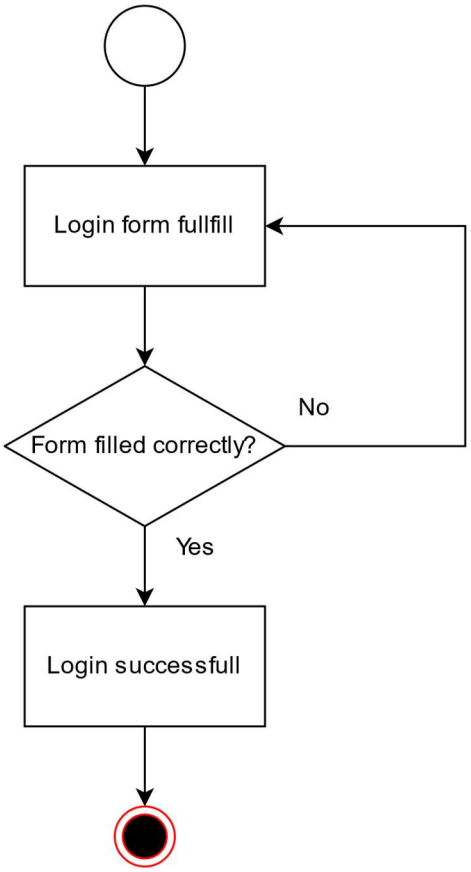
Activity Diagram [UC1]

UC[2] Login:



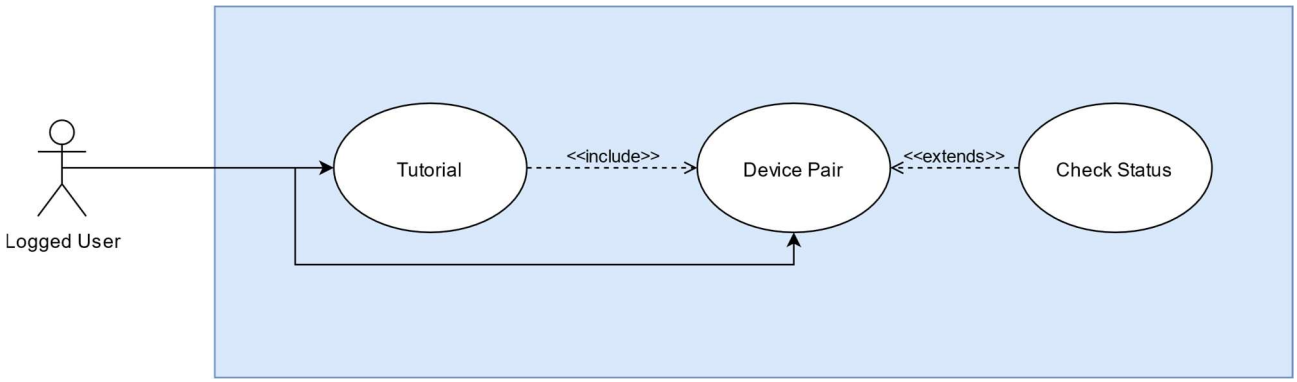
[UC2] Login

Name	Login
Actor	User
Entry Condition	The actor must be registered
Goal	G5
Event Flow	1. The user fills up the login form with their credentials 2. The system checks the credential given
Exit condition	The user is logged in and he ca use the functionalities of the application. We will call **Logged User** the user that has performed this use case.
Exceptions	1. Some form fields get incorrectly filled or left empty, the system notifies the user on the problem and highlights what's wrong, the user needs to fill or correct the selected items. 2. Some credentials provided are wrong, the user will be alerted with a "wrong credentials" message.



Activity Diagram [UC2]

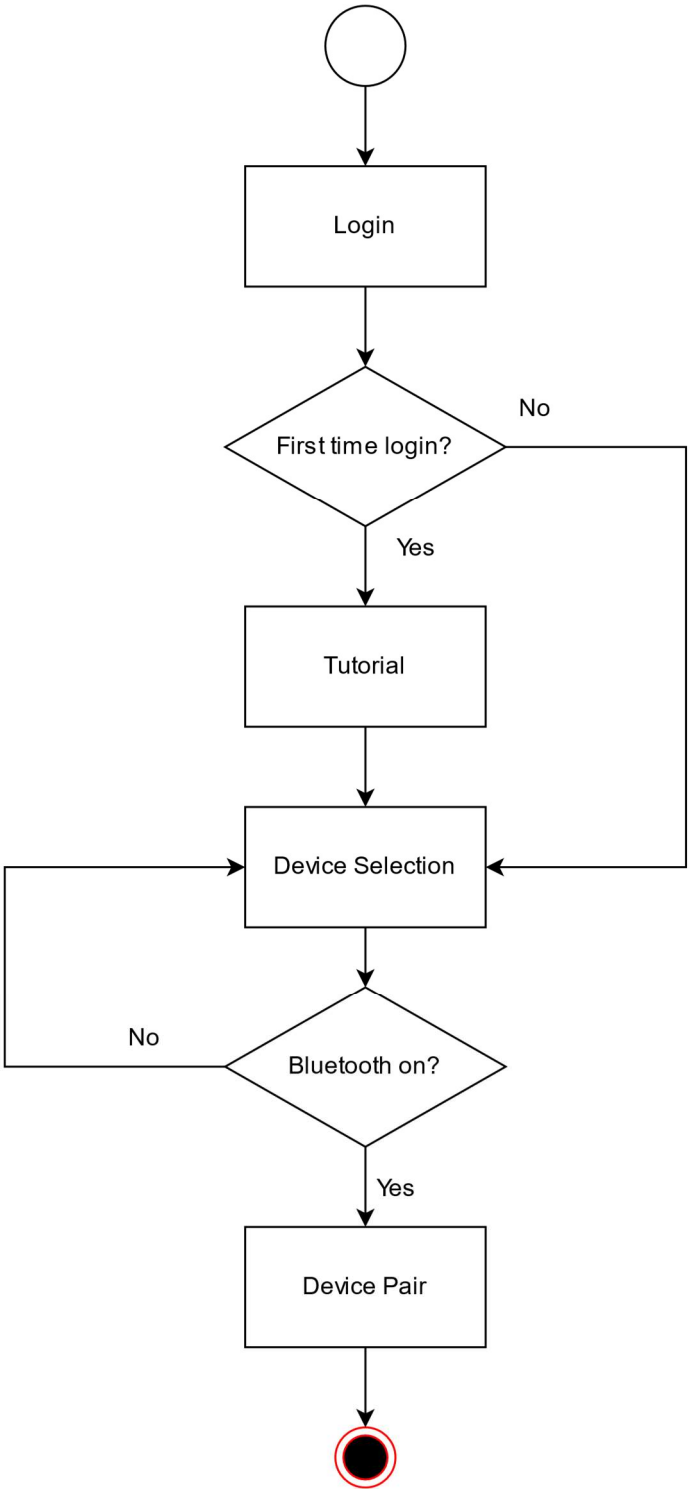
UC[3] Device Pair:



[UC3] Device Pair

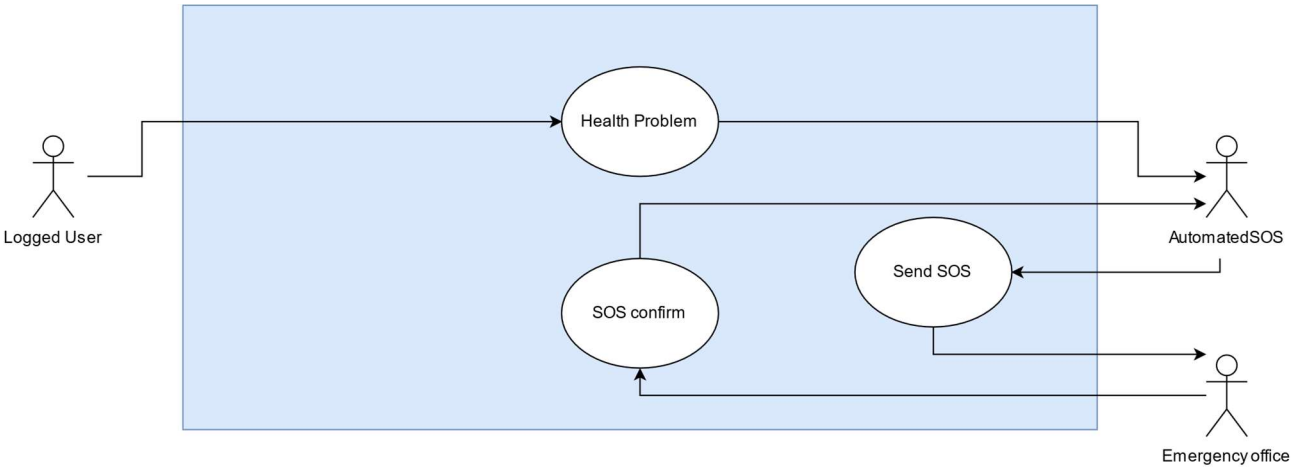
Name	Tutorial
Actor	Logged User
Entry Condition	The actor must be logged in for the first time
Goal	G1
Event Flow	1. The user gets prompted with information on the app and functionalities along with a description of the menu 2. The user gets guided through the pair device process described in the next table
Exit condition	The user gets guided until the end of pairing process
Exceptions	1. App crashes 2. The user doesn't complete the tutorial process In both the exceptions the system provides the tutorial process again

Name	Device Pair
Actor	Logged User
Entry Condition	The actor must be logged in
Goal	
Event Flow	1. The user selects his own device from the drop-down menu (see Fig.3.1.9) 2. The user clicks the "Pair" button (see Fig.3.1.9)
Exit condition	The user connects successfully a new device with the application
Exceptions	1. Bluetooth turned off, the user must activate it and open again the drop-down menu



Activity Diagram [UC3]

[UC4] Emergency:

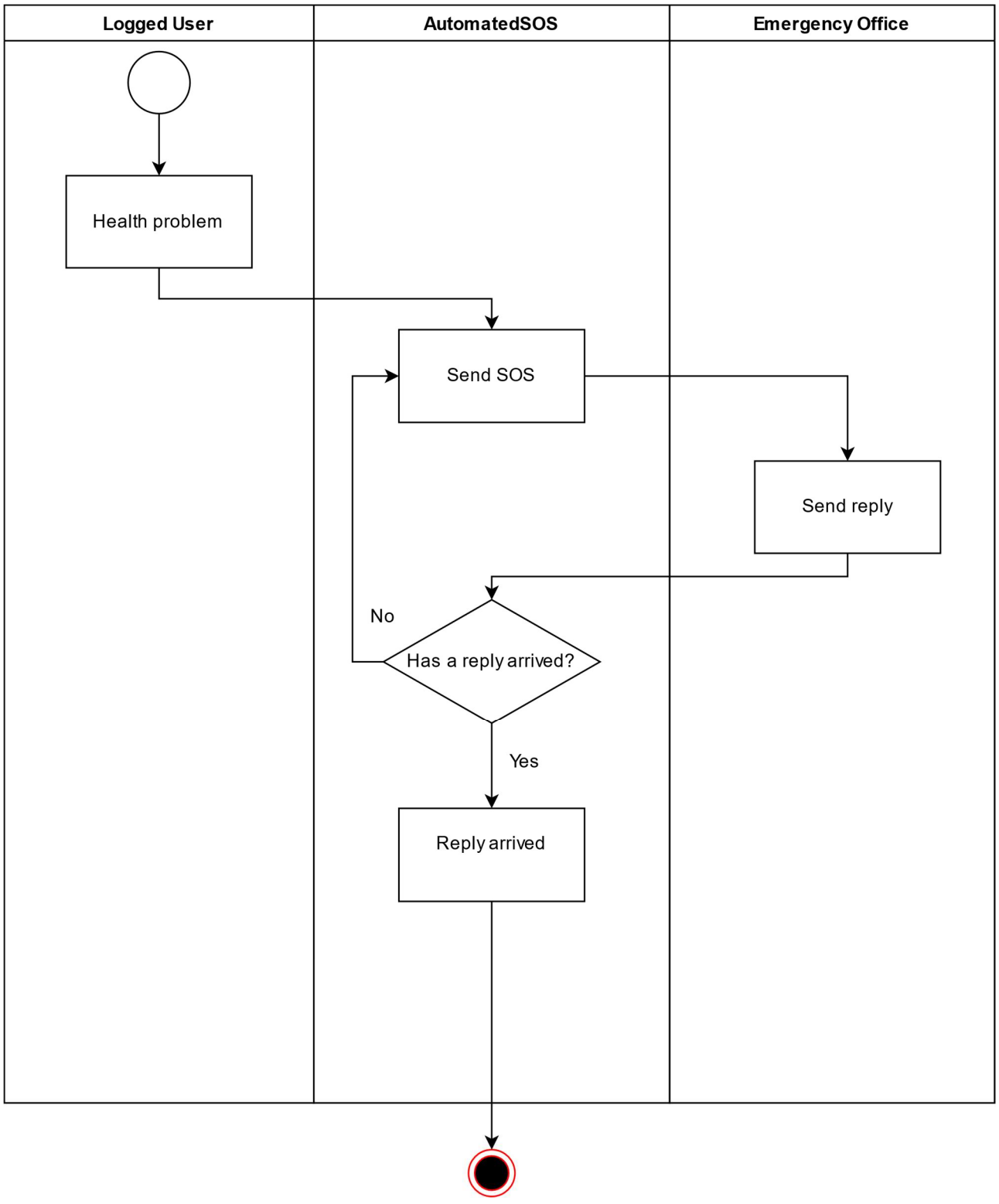


[UC4] Emergency

Name	Health problem
Actor	Logged User
Entry Condition	The user has a health problem
Goal	G1, G2, G4
Event Flow	1. The user has a health problem 2. The problem gets identified and tracked by Data4Help
Exit condition	Data4help identifies the problem and notifies AutomatedSOS
Exceptions	1. The health device isn't working properly The user must be notified if the device isn't working properly

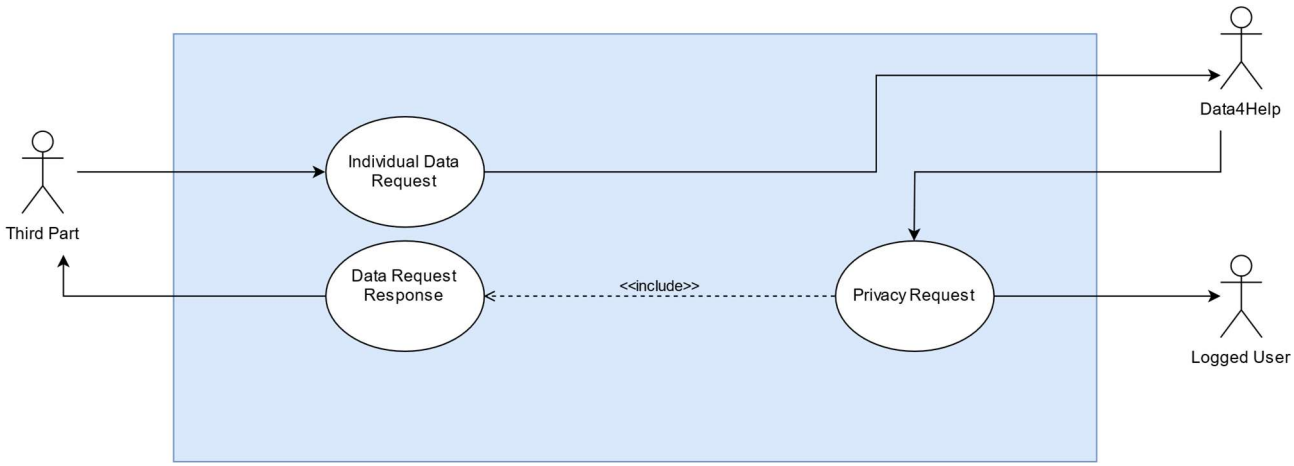
Name	Send SOS
Actor	AutomatedSOS
Entry Condition	AutomatedSOS identifies a health problem
Goal	G1, G2, G4
Event Flow	1. AutomatedSOS identifies a health problem 2. AutomatedSOS sends a SMS to the emergency office requiring assistance at the user location
Exit condition	AutomatedSOS identifies the problem
Exceptions	1. The smartphone is unable to send a SMS The system must continuously try to send the SMS until it succeeds

Name	SOS confirm
Actor	Emergency office
Entry Condition	A SOS arrives at the emergency office
Goal	G1, G2, G4
Event Flow	1. A SOS arrives at the emergency office 2. The emergency office sends a reply to acknowledge the dispatch of an ambulance
Exit condition	A reply message is sent to AutomatedSOS
Exceptions	We imply that the emergency office is efficient and has no problem sending a reply message



Activity Diagram [UC4]

[UC5] Third-party individual request:

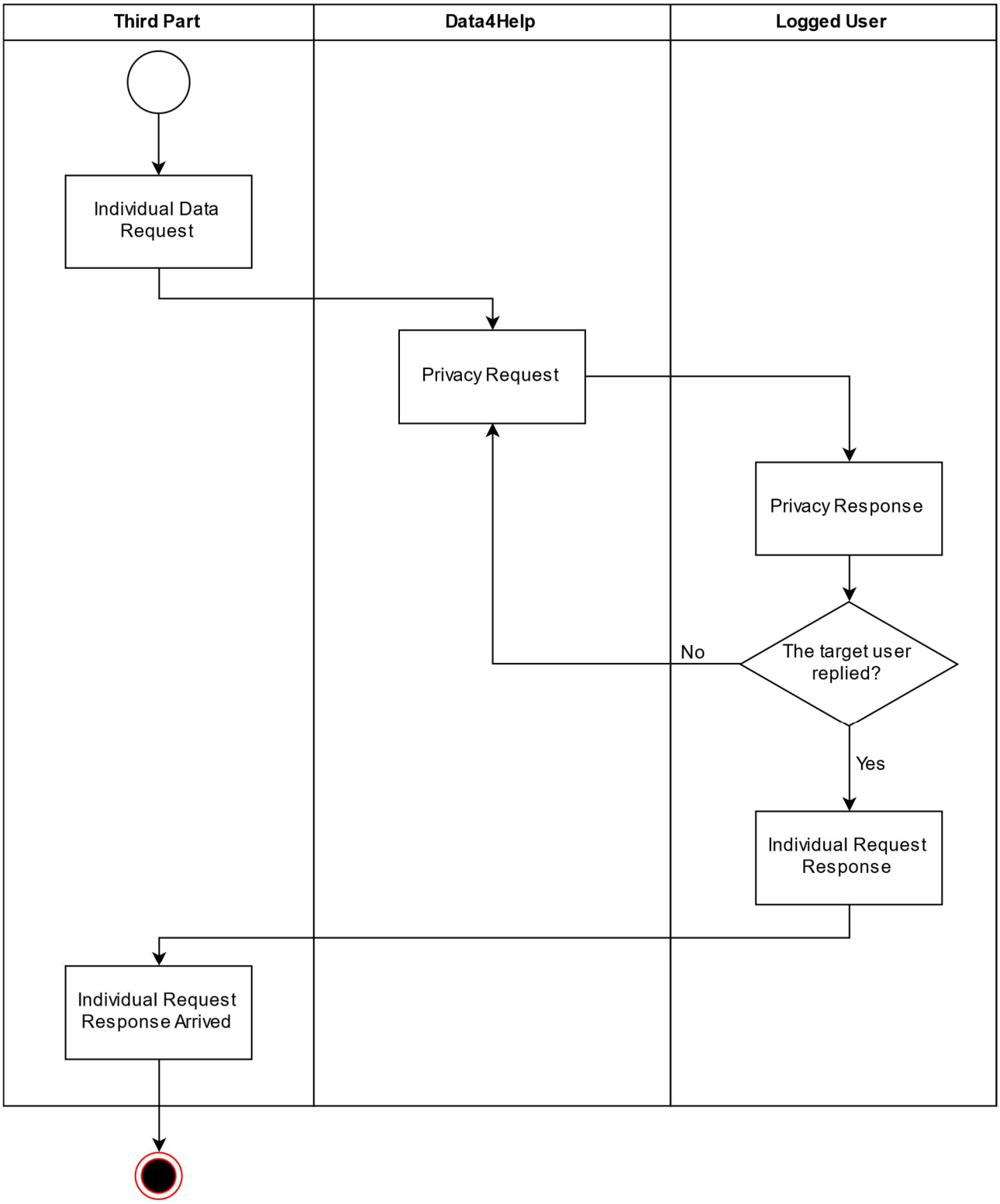


[UC5] Third-party individual request

Name	Individual Data Request
Actor	Third Part
Entry Condition	The third part must have been added into the system
Goal	
Event Flow	3. The third part sends a request 4. The third part waits a response
Exit condition	The third part sends a request and the system receives it
Exceptions	2. Data4Help isn't reachable 3. Third part system crashed In both exceptions the third part continues to send the same request until it gets a response

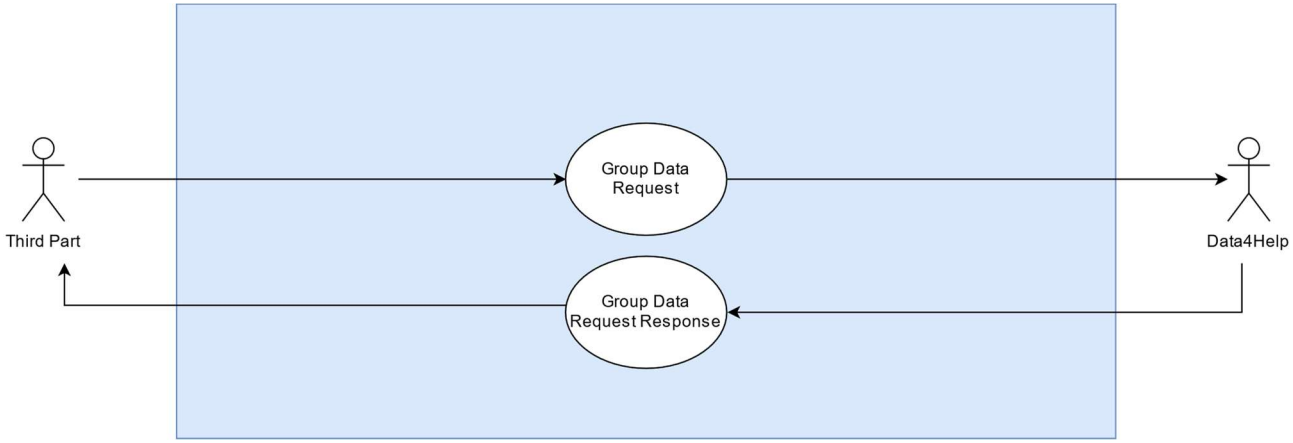
Name	Privacy Request
Actor	Data4Help
Entry Condition	A data request arrives to the system
Goal	
Event Flow	1. The system sends a Privacy request to the target user 2. The target user accepts or denies the request
Exit condition	The user accepts or denies the request
Exceptions	1. The user ignores the request, the system continues to notify the user. After 5 attempts the system considers the request denied

Name	Data Request Response
Actor	Data4Help
Entry Condition	The user replied to the privacy request
Goal	
Event Flow	1. The system replies to the third part according the target response
Exit condition	The third part gets a response
Exceptions	1. The system crashed 2. Third-party system isn't reachable In both exceptions the system continues to send the same response until it arrives



Activity Diagram [UC5]

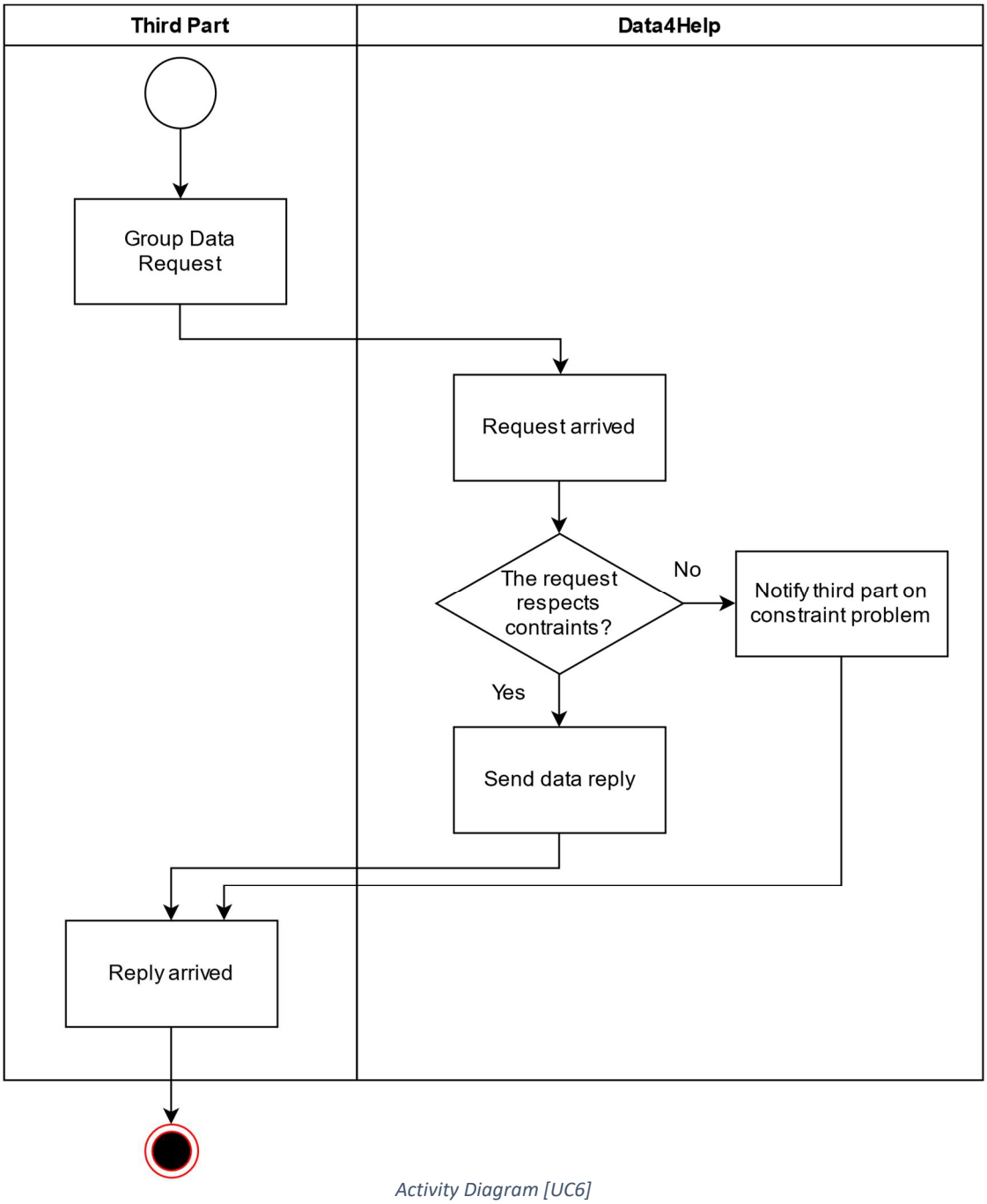
[UC6] Third-party group request:



[UC6] Third-party group request

Name	Group Data Request
Actor	Third Part
Entry Condition	The third part must have been added into the system
Goal	
Event Flow	1. The third part sends a request 2. The third part waits a response
Exit condition	The third part sends a request and the system receives it
Exceptions	1. Data4Help isn't reachable 2. Third part system crashed In both exceptions the third part continues to send the same request until it gets a response

Name	Group Data Request Response
Actor	Data4Help
Entry Condition	A data request arrives to the system
Goal	
Event Flow	1. The system replies to the third part respecting the cardinality constraint (See constraint 5, section 2.4)
Exit condition	The third part gets a response
Exceptions	1. The system crashed 2. Third-party system isn't reachable In both exceptions the system continues to send the same response until it arrives



Activity Diagram [UC6]

3.3 Performance Requirements

As the system is meant to help a large number of people, situated in big enough cities, the requests that it will have to satisfy will be at least large as much as the number of users and the majority of these requests will be likely to be sent simultaneously.
Since a part of the system deals with the life of people, that is called AutomatedSOS, the system must guarantee a high reliability and a good fault tolerance.
Therefore, the system must fulfil any request within half of the largest time of reaction possible, so that it will be 2 seconds. The number of users satisfied should be 20000 but it is expected a load of parallel 2000 every time.

3.4 Design Constraints

3.4.1 Standards Compliance

- The application requests all the device permission that are needed to grant the functionality and the reachability of the goals.
- The application can be used by elderly people, that can have vision and hearing limitations, that means that usability and functionalities must be preserved no matter what kind of dimension or orientation has the device that displays and interacts with the user.
- The app preserves user's choices and runs independently from its state ('app is open' or 'app is closed') until the device has charge. If it is open must be able to collect the decisions of the user.

3.4.2 Hardware limitations

The known hardware limitations until today are related to the fact that our application can interact with only some of the brands that produce smartwatches, and with smartphones that can provide:

- GPS
- 2G/3G/4G Connection
- Android or iOS associable with smartwatches and similar.

3.4.3 Other Constraints

- **Legislation**

  All the user's data must be treated as state law declare. Regulations on minor and regulations on data recording and distribution must be respected.

3.5 Software System Attributes

1. **Reliability**
   The application must be available 24/7. Small concessions can be tolerated for Data4Help but none for AutomatedSOS.

2. **Availability**
   AutomatedSOS will be required to have an availability of 99.999% corresponding to a total downtime of 5 minutes during the whole year. To reach this objective the system will be based on a cloud system, so that if a machine is not reachable others will be. Data4Help will have an availability of 99% which is equal to 3.65 days in a year. Data4help will have a solution similar to AutomatedSOS.

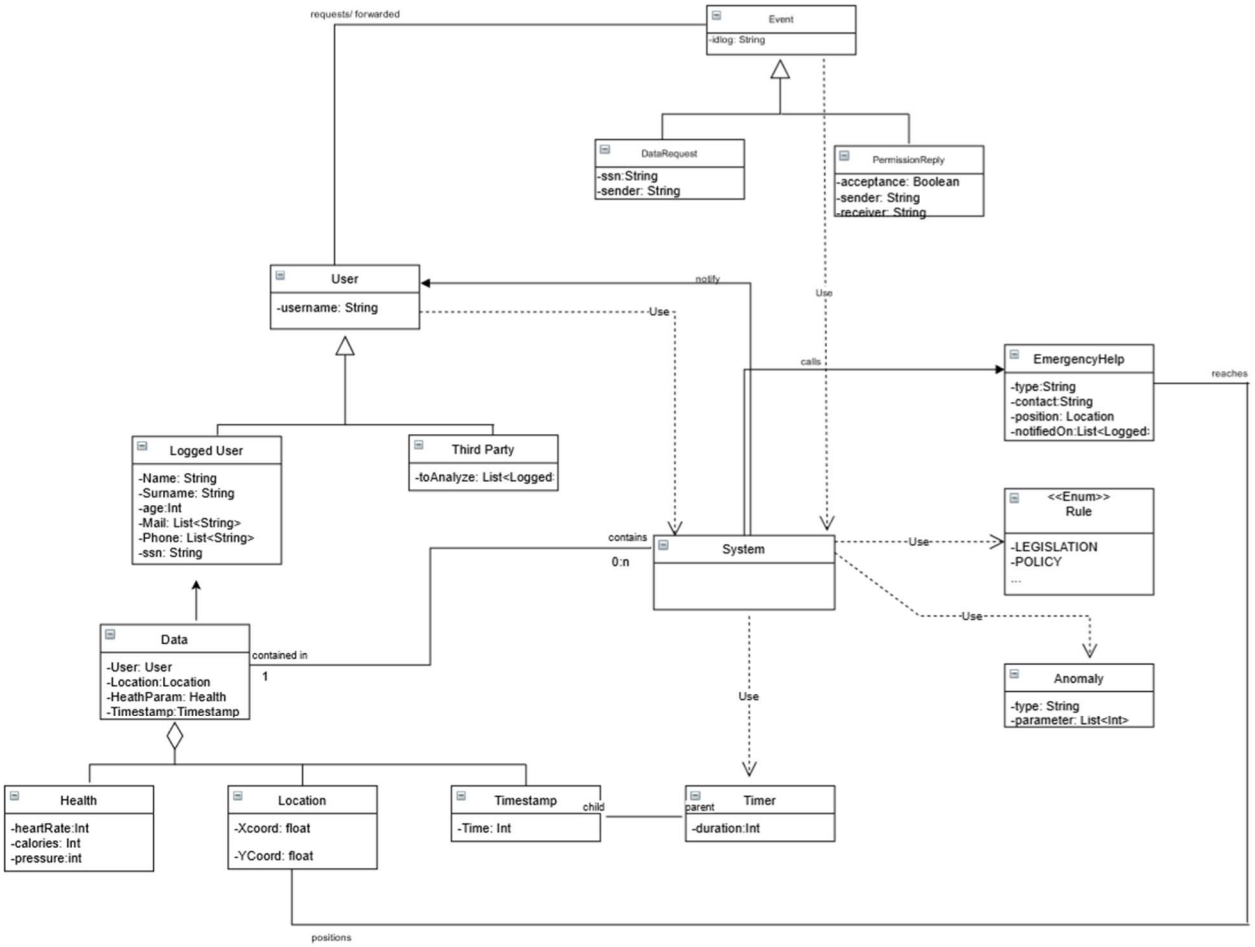
3. **Security**
   The access to the system is guaranteed after a secure login.
   The system must be able to maintain all the sensitive information safe and private.
   To fulfil this goal the system should encrypt all the communication between every server and every client in the network, to avoid leaks of sensitive data. The data should be persistent, hence several back-up routines will be run to avoid any loss and to make any system recovery easy. Only logged user will be able to visualize information and interact with the system.
   The users will be able to modify, visualize and interact with data and tools that permissions let them access.

4. **Maintainability**
   The system should be developed with techniques and behaviours that increase its modularity and extensibility. For such purpose the project must be built on languages that enhance these capabilities.
   The system should also be tested intensively to make it predictable and not leave space to unexpected behaviours.

5. **Portability**
   The system application logic is not run on the client but totally on the server, this makes the clients code fully portable. This is a key point because this feature maximizes also the possibility for collecting new user by making the app easier to install to the current devices of possible future clients.

6. **Scalability**
   The system should be scalable horizontally in order to maximize the possibility of augmenting workloads.

4 Formal Analysis using Alloy



The system that we are trying to describe is really complex and so is the world that surrounds it, so what is modelled and represented is just a subset of the problems regarding the formalization of the problem.

In the Class Diagram it has been decided to explain relations between the users which have some defined attributes with a certain cardinality, like the unicity of the SSN code. It was necessary to define the nature and the content of the data, how it is related to the user ad how it can define the flow of events depending on the comparison of anomalies. The data can't be tangible but are still what is valuable about the system. The time is a key point of the system since it's what determines the performance and makes trustworthy the system itself; it wasn't modelled because of its rich features and not suitable for a RASD.

In the Class Diagram the will of the users are expressed through events. The rules are a mix of legislation and policies and the help for emergency are indeed real actors that take participation in case of an emergency.

Most of the attributes, instances of an entity and hierarchical relations have been rewritten to let them be modelled in Alloy.

But why Alloy?

The Alloy model is useful, aside to give a visualization of the generated world, to understand how the entities operating inside the application world interact with each other and what types of constraints and facts exist between them. The model is self contained on purpose in order to not abuse of the formalization.

What was modelled is then:

- Every relation between the entities with regard to the cardinalities.
- The possibility for a user to decide whether to share or not his information.
- The possibility of asking to a user for his data through the ssn.
- The flow of events that happens immediately after the creation of new data, whether it is an anomaly or not.
- It is verified one of the main core characteristics of the system which is the guarantee of privacy for the user that have expressed a preference.
- It is verified that whenever there is a new emergency an available ambulance has the notification

```
open util/integer
open util/boolean

abstract sig  User{
		username: one Username
}
sig ThirdParty extends User{
		analyze: set LoggedUser,
		dataToRead: set Data
}

sig LoggedUser extends User{
		ssn: one SSN,
		name: one Name,
		surname: one Surname,
		mail: one Mail,
		phone: one Phone,
		preference: one PRIVACY_PREFERENCE,
		hisData: set Data,
		isOld: one Bool
}

sig Data{
		user: one LoggedUser,
		location: one Location,
		health: one Health,
		timestamp: one Timestamp
}
sig Location{
		Xcoord: one Int,
		Ycoord: one Int
}
{ Xcoord >= -3 and Xcoord <= 3 and Ycoord >= -6 and Ycoord <= 6 }
// coordX >= -90 and coordX <= 90 and coordY >= -180 and coordY <= 180
```

```
some sig EmergencyHelp{
		notification: one Location,
		contact:  one Contact,
		available: one Bool,
		location: one Location
}
sig Emergency{
		u: one loggedUser,
		d: one Data,
		a: one Anomaly,
		h: one EmergencyHelp
}

sig Health{
		heartrate: one Int,
		pressure: one Int

}{heartrate>=0 and heartrate<=6 and pressure>=0 and pressure<=4}
//instead of heartrate>=60 and heartrate<=200 and pressure>=60 and pressure<=100

sig Timestamp{
		timestamp: one Int
}{timestamp>=0 and timestamp<=6}

abstract sig Event{
}
sig DataReq extends Event{
		respondedBy: one ReqReply,
		ssn: one SSN,
		sender: one ThirdParty
}
sig ReqReply extends Event{
		responseTo: one DataReq,
		acceptance: one Bool,
		sender: one loggedUser,
		receiver: one ThirdParty
}
```

```
sig SSN{
		owner: one loggedUser
}
sig Username{}
sig Name{}
sig Surname{}
sig Contact{}
sig Mail{}
sig Phone extends Contact{}
sig ID{}
//////////////////RULES__AND__ANOMALIES////////////////////////////////
abstract sig Rule{}

sig PRIVACY_PREFERENCE extends Rule{
		whoCan:  set ThirdParty
}
sig POLICY extends Rule{
		misuse: one Int
}{misuse>0 and misuse<=5}

-------------------------------------------------------------------------
one sig Anomaly{
		condition: some Health,
		recordedBy: some Emergency
}

//////////////////////PREDICATES_AND_SHOW///////////
pred result(u: User){}
//////////////////USER////////////////////////
fact nickunique{
		no disj first, second : User | first.username=second.username
}

fact unitime{
		no disj t1,t2: Timestamp | t1.timestamp=t2.timestamp
}
/////////////////////////////////////////////////
```

```
fact ssnunique{
		no disj first, second : loggedUser | first.ssn=second.ssn
}
fact ssnowner{
		all s: SSN | all u: loggedUser | u=s.owner iff u.ssn=s
}

fact mailunique{
		no disj u1, u2 : loggedUser | u1.mail=u2.mail
}

fact phoneunique{
		no disj u1, u2 : loggedUser | u1.phone=u2.phone
}

fact locunique{
		no disj loc1, loc2: Location | loc1.Xcoord=loc2.Xcoord and  loc1.Ycoord=loc2.Ycoord
}
fact prefunique{
				no disj u1, u2 : loggedUser | u1.preference=u2.preference
}
//user knows which his data
fact userToData{
		all u:loggedUser |( all d:Data | d.user=u iff d in u.hisData )
}

//and back
fact dataToUser{
		all d1, d2 : Data |  (d1.timestamp=d2.timestamp and d1.location=d2.location and d1.user=d2.user and d1.health=d2.health )
						implies d1=d2
}
//every user must produce some data
fact onePerUser{
		all u:loggedUser | one d:Data | d in u.hisData iff d.user=u
}
```

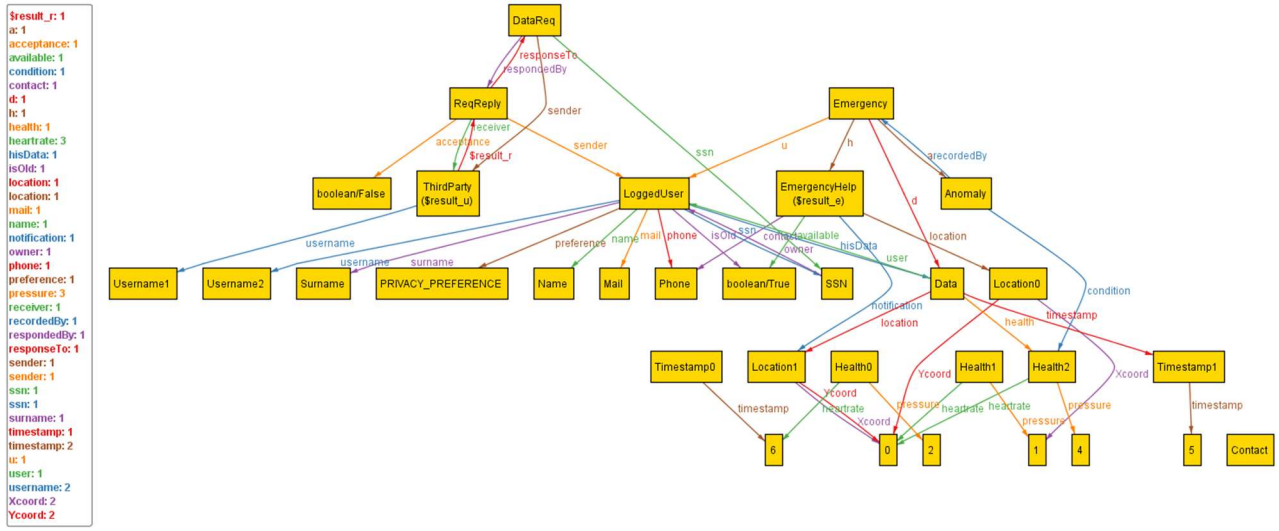
```
fact uniquedata{
		no disj d1, d2 : Data | d1.timestamp=d2.timestamp and d1.location=d2.location
}

fact uniquedata_withReagardToAUser{
		all d1, d2 : Data |  d1.timestamp=d2.timestamp and d1.location=d2.location implies
									(d1.user=d2.user and d1.health=d2.health )
}
//a third party can see all the data of those he can follow
fact dataToThird{
		all t:ThirdParty |( all d:Data | d.user in t.analyze iff d in t.dataToRead )
}

fact unique_healthcondition{
		no disj h1,h2: Health | h1.heartrate=h2.heartrate and h1.pressure=h2.pressure
}

fact uniqueEmefirst{
		no disj e1, e2: EmergencyHelp | e1.location=e2.location
}

fact uniqueEmesecond{
		no disj  e1, e2: EmergencyHelp | e1.contact=e2.contact
}

fact unique_reply{
		no disj r1,r2: ReqReply | r1.responseTo=r2.responseTo
}
fact unique_request{
		no disj d1,d2: DataReq | d1.respondedBy=d2.respondedBy
}
fact link{
		all r:ReqReply |( all d:DataReq | r.responseTo = d iff d.respondedBy=r)
}

fact linking{
all r:ReqReply |( all d:DataReq | r.responseTo = d implies (r.sender.ssn = d.ssn and d.sender=r.receiver))
}
```

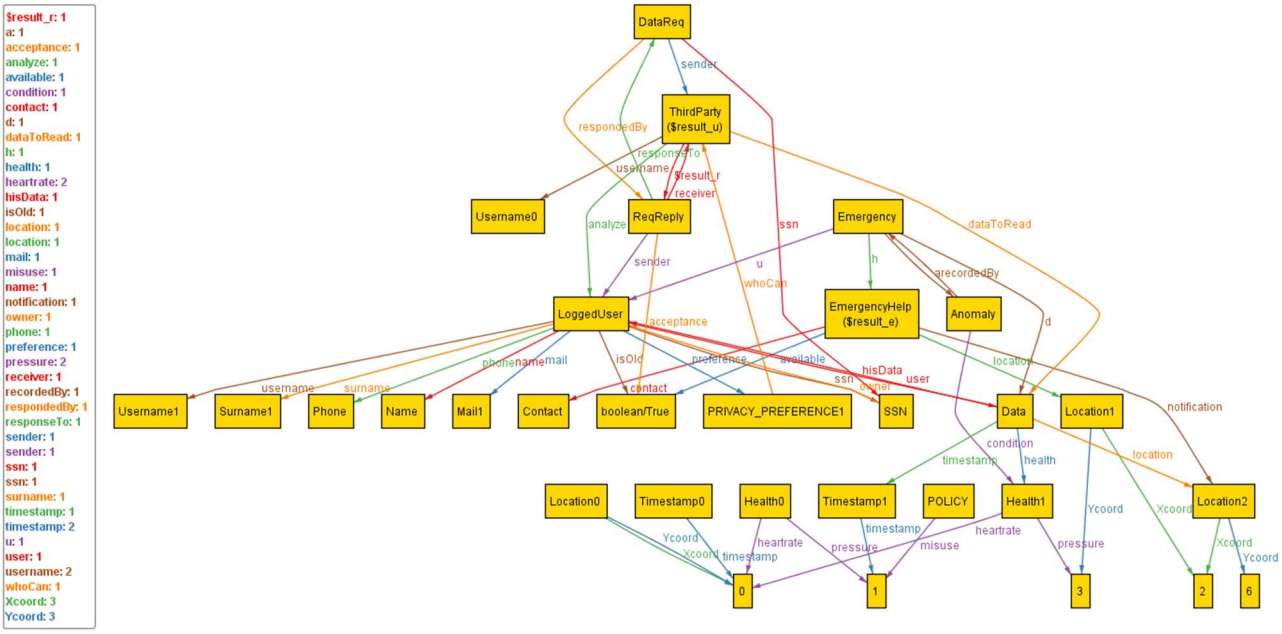
```
//a third party must do some request to get the data
fact onePerThird{
		one d: DataReq| one t:ThirdParty | d.sender=t
}
////////////////////////////////////////////////////////////////////////////
//whenever there an anomaly and data that match this is recorded and matched with an emergency
fact notified{
		all em: EmergencyHelp| all da: Data | all aa: Anomaly | all e: Emergency |
					da.health in aa.condition implies ( e.u = da.user and e.d=da and e.a=aa and e.h=em)
}
//if a reply is positive the third can see the data of a user, and the user can be seen
fact if_accepted_analyze_and_can_see{
		all r: ReqReply, t:ThirdParty, u:LoggedUser |
				some d:DataReq | r.acceptance=boolean/True
iff u in t.analyze and t in u.preference.whoCan and  r.receiver=t and r.sender=u and d.sender=t and d.ssn=u.ssn

}

fact monoreply{
		all r1, r2: ReqReply| r1.sender=r2.sender and r1.receiver=r2.receiver
					iff( r1.acceptance=r2.acceptance and r1.responseTo=r2.responseTo)}
fact monoreply_identity{
		no disj r1, r2: ReqReply| r1.sender=r2.sender and r1.receiver=r2.receiver
}
fact monorequest_identity{
		no disj r1, r2: DataReq| r1.sender=r2.sender and r1.ssn=r2.ssn
}

fact onlyThoseWhoAsked{
		all t: ThirdParty| all u: LoggedUser | all d:DataReq |  (u in t.analyze implies t in u.preference.whoCan) implies d.sender=t
}

//se non è nella reply non può nemmeno esserci la possibilità di osservare
fact possibility{
		all t: ThirdParty | some r :ReqReply | #t.analyze>0 iff (r.receiver=t and r.acceptance=boolean/True)
}
```

```
fact alwaysOneavailable{
		some e : EmergencyHelp  | e.available=boolean/True
}
fact availTakesEme{
		all e : EmergencyHelp | all eme: Emergency | e=eme.h and eme.h.available=boolean/True
}
fact EmeHasdiffAnomaly{
		no disj eme1, eme2: Emergency |  eme1.a=eme2.a
}
fact EmeHasdiffHelp{
		no disj eme1, eme2: Emergency |  eme1.h=eme2.h
}
fact EmeHelpIsNotExactlyWhereIsEmergency{
		all eme: EmergencyHelp |  eme.location!=eme.notification
}
fact ambulanceDiffPatient{//better distribution of the ambulances
		no disj amb1, amb2: EmergencyHelp| amb1.available=boolean/True and amb2.available=boolean/True implies
							amb1.notification!=amb2.notification
}
fact recordingAnomaly{
		all e: Emergency | all aa: Anomaly | e.a=aa iff aa.recordedBy=e
}
fact healthAnomalysavedBeforeChecked{
		all l: LoggedUser, e: Emergency| e.u=l iff e.a.condition in e.u.hisData.health
}
fact ifTwoUsersHaveSameHealth{
		//possono avere la stessa condizione ma luoghi diversi o tempo diverso
}
fact notificationIsDataLocation{
		all e: Emergency | e.d in e.u.hisData and e.h.notification in e.u.hisData.location
							and e.a.condition=e.d.health
}
```

```
//predicate that spawns a new data and conccets it to the system, almost like a notification of new data
pred new(d: Data, u:loggedUser, h:Health, l:Location,t:Timestamp,req: ReqReply,th:ThirdParty,dr:DataReq){
                d.user=u
                d.health=h
                d.location=l
                d.timestamp=t
                req.acceptance=boolean/True
                th=req.receiver
                req.sender=u
                dr.sender=th
                dr.respondedBy=req
                req.responseTo=dr
                th.analyze = th.analyze + u
                u.hisData=u.hisData+ d
}
pred newEmergency(e: Emergency, an: Anomaly, help: EmergencyHelp, logged: loggedUser, data: Data){
                e.u=logged
                e.h=help
                e.d=data
                e.a=an
}

assert newDataSeen2{
one third:ThirdParty, users: loggedUser |
                some data: Data, ll:Location, heal: Health, tt:Timestamp,req:ReqReply,dataReq: DataReq |
                                (#DataReq>0 and new[data, users, heal, ll,tt,req,third,dataReq] )implies
                                                data in third.dataToRead and third in users.preference.whoCan
}

assert newEme{
all logged: loggedUser | some eme:Emergency, anom: Anomaly | some help: EmergencyHelp, data: Data |
                        newEmergency[eme, anom,help,logged, data] implies help.notification=data.location
}


check newDataSeen2 for 2
check newEme for 2
run result for 3
```

```
Executing "Check newDataSeen2 for 2"
  Solver=sat4j Bitwidth=4 MaxSeq=2 SkolemDepth=1 Symmetry=20
  6557 vars. 368 primary vars. 17349 clauses. 502ms.
  No counterexample found. Assertion may be valid. 101ms.


Executing "Check newEme for 2"
  Solver=sat4j Bitwidth=4 MaxSeq=2 SkolemDepth=1 Symmetry=20
  4998 vars. 370 primary vars. 12289 clauses. 155ms.
  No counterexample found. Assertion may be valid. 38ms.


Executing "Run result for 3"
  Solver=sat4j Bitwidth=4 MaxSeq=3 SkolemDepth=1 Symmetry=20
  9962 vars. 642 primary vars. 23920 clauses. 172ms.
  Instance found. Predicate is consistent. 214ms.
```





5 Effort Spent

While working on the project, we always met to discuss main topics to provide more consistence.

Then we worked on each part on our own and finally we have met to check that everything was in place.

Name	Effort
Kevin Mato	15h group and 25h alone
Antonio Mazzeo	15h group and 25h alone

6 References

1. The revised document of the assignment:
   https://beep.metid.polimi.it/documents/121843524/3f744351-7378-4162-86b0-45eddaf10713

2. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.

3. IEEE Std 1016tm-2009 Standard for Information Technology-System Design-Software Design Descriptions.

4. Documentation about alloy:
   http://alloy.mit.edu/alloy/