

**TrackMe project L. Barilani, W. Bonvini,  
L. Carnaghi**



**POLITECNICO**  
MILANO 1863

# **Design Document**

---

<b>Deliverable:</b>	DD
<b>Title:</b>	Design Document
<b>Authors:</b>	Leonardo Barilani, William Bonvini, Lorenzo Carnaghi
<b>Version:</b>	1.0
<b>Date:</b>	8-November-2018
<b>Download page:</b>	<a href="https://github.com/Lockyard/BarilaniBonviniCarnaghi">https://github.com/Lockyard/BarilaniBonviniCarnaghi</a>
<b>Copyright:</b>	Copyright © 2018, Leonardo Barilani, William Bonvini, Lorenzo Carnaghi – All rights reserved

---

## Contents

<b>Table of Contents</b>	<b>3</b>
<b>1 Introduction</b>	<b>5</b>
1.1 Purpose	5
1.2 Scope	5
1.3 Definitions, Acronyms, Abbreviations	5
1.3.1 Definitions	5
1.3.2 Abbreviations	6
1.3.3 Acronyms	6
1.4 Revision history	6
1.5 Document Structure	6
<b>2 Architectural Design</b>	<b>7</b>
2.1 Overview	7
2.2 Component View	7
2.3 Deployment View	10
2.4 Runtime View	11
2.4.1 Registering a new user in TrackMe ecosystem	11
2.4.2 Storing a new raw data packet in Data4Help	11
2.4.3 Emergency detected in AutomatedSOS	12
2.4.4 Retrieving the positions of runners in a run in Track4Run	12
2.5 Component Interfaces	13
2.6 Selected architectural styles and patterns	14
2.6.1 Architectural styles	14
2.6.2 Patterns	14
2.7 Other design decisions	15
2.7.1 ER schema	15
2.7.2 AutomatedSOS optimization	15
2.7.3 Login optimization	16
<b>3 User Interface design</b>	<b>17</b>
3.1 Mockups	17
3.2 User Experience Diagrams	23
<b>4 Requirements traceability</b>	<b>26</b>
4.1 Mapping Goals to Components	26
4.2 Mapping Requirements to Components	27
<b>5 Implementation, integration and test plan</b>	<b>30</b>
5.1 Implementation plan	30
5.1.1 Overview	30
5.1.2 Major tasks	30
5.1.3 Implementation Schedule	30
5.2 Integration and test plan	31
5.2.1 Overview	31
5.2.2 Integration plan	31
5.2.3 Integration details	33
5.2.4 Integration test plan	37

<b>6</b>	<b>Effort Spent</b>	<b>38</b>
6.1	Leonardo Barilani	38
6.2	William Bonvini	38
6.3	Lorenzo Carnaghi	38

# 1 Introduction

## 1.1 Purpose

This document contains a technical review of the RASD of the projects Data4Help, AutomatedSOS and Track4Run. In details, we want to highlight the importance of these aspects:

- High and low level architecture;
- Components and their connections;
- Interaction between users and apps;
- Design patterns;
- Implementation, Integration and Testing plans.

## 1.2 Scope

Data4Help is a service mainly thought for big companies who want to make better targeted choices for their business. Since the service acquires the health status of the users, most of the companies interested in the service are supposedly going to be related to lifestyle, fitness, and generally the sport world. The position of the users is instead a very useful information for third parties related to the transportation system and tourism. Health status and position combined offer a good overview of the health situation of a population of a specific area selected by the third party. Such data can be used for pools and statistical purposes, so being attractive to health related realities: pharmaceutical industry, health organizations (state entities and non state ones). The final product will consist of 3 main softwares:

- The server-side software;
- The mobile application for standard-users;
- The desktop application for third-parties users.

There are quite a few shared phenomena that occur in the projects:

- When a third party request a query in Data4Help;
- When one of the three application gather information;
- When AutomatedSOS call an ambulance in case of an emergency;
- When a run is created in Track4Run.

## 1.3 Definitions, Acronyms, Abbreviations

### 1.3.1 Definitions

- User: a user is a person who uses Data4Help's services by having installed the application on their smartwatch and registered to the Data4Help platform.
- Third-party: any organization/company/authority who has access to query the database which stores all users' data.
- Query: A filtered search asked to the Data4Help's DB.
- Race/Run: an event organized through Track4Run service, in a specific location.

### 1.3.2 Abbreviations

- [G<sub>x</sub>]: x-th Goal
- [D<sub>x</sub>]: x-th Domain assumption
- [R<sub>x</sub>]: x-th Requirement
- [FR<sub>x</sub>]: x-th Functional Requirement

### 1.3.3 Acronyms

- TP: Third Party (TPs for plural)
- SSN: Social Security Number
- CF: Codice Fiscale
- DB: Database
- D4H: Data4Help
- D4HU: Data4Help User view
- D4HTP: Data4Help Third Party view
- ASOS: Automated SOS
- T4R: Track4Run
- ADQR: Anonymous Data Query Request
- IDR: Individual Data Request

## 1.4 Revision history

For version 2 of DD:

- Removed references to chat system from UX diagrams and mockups

## 1.5 Document Structure

**Architectural design** explains how the whole structure of the projects comes together, both in the client-server networking aspect and in the interfaces that are used in the software shared between all the devices involved in the services.

**User interfaces design** gives a first look on how the user interface of the projects should look like and shows how the user interacts with the application.

**Requirements traceability** shows how the implemented elements actually fulfill the requirements and goals requested by the RASD document.

**Implementation, Integration and Test plan** provides the relative schedules which developers have to follow in order to achieve the optimal outcome.

**Effort spent** shows for each section the effort spent by individual group member.

## 2 Architectural Design

### 2.1 Overview

The high level architecture is here presented and described. This architecture is divided in 3 tiers: presentation tier, business logic tier and data tier.

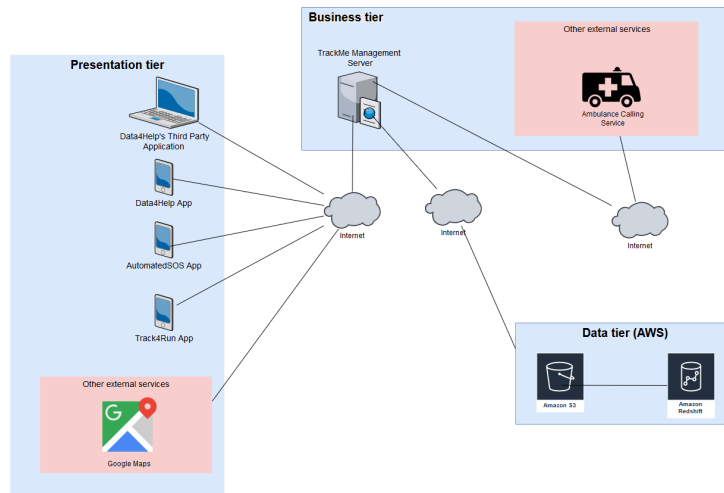


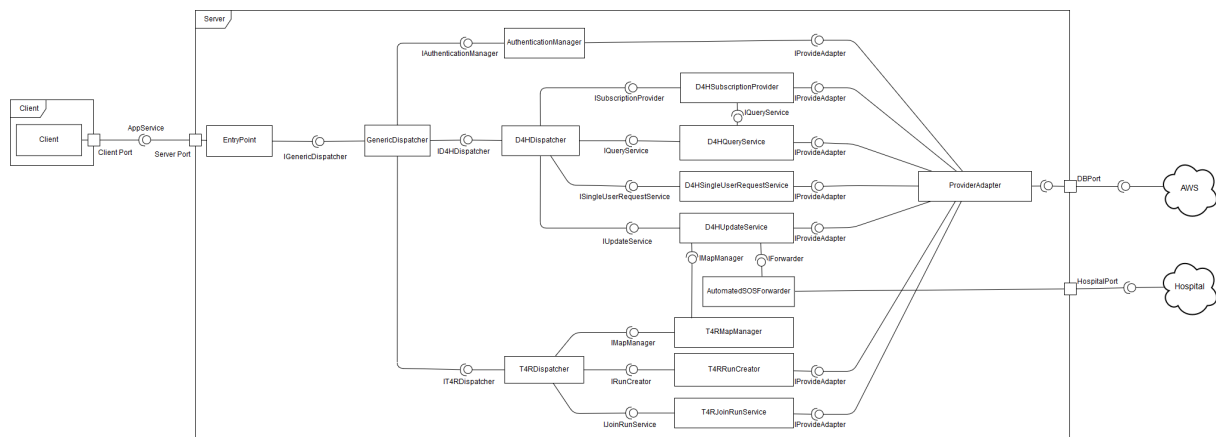
Figure 1: High Level Architecture Diagram

**Presentation Tier:** is composed by all the client applications, which have only a user interface functionality and do not contain almost any logic. This tier is connected to the Business logic tier.

**Business logic tier:** is composed by the server application and the external services excluded AWS, i.e. the hospital calling service. It contains all the core logic of the business, and thus is linked to both the presentation and the data tiers.

**Data tier:** is composed by the AWS service. It's function is to provide storage of data used by the logic tier, and hence is linked to it.

### 2.2 Component View



Component diagram for Software Engineering 2 Project "Track Me" Authors: Leonardo Barilani, William Bonvini, Lorenzo Carnaghi

Figure 2: Component View of the TrackMe Management Server

The components present in the system are the following:

- Entry Point: this component takes care of receiving all the packets from the clients. It performs some validation checks on the arriving packets. If the packet contains readable and correctly formatted data it is forwarded to Generic Dispatcher;
- Generic Dispatcher: this component takes care of properly sorting incoming data. Every packet with no exception is passed to Authentication Manager that validates the packet. After validation the Generic Dispatcher reads the type of the packet and it forwards it to the correct component. A packet can be of 3 types:
  - User packet: forwarded to Authentication Manager, this type of packet contains information about registration, login and account management of a user (changing credentials, ecc.);
  - D4H packet: forwarded to D4HDispatcher, this type of packet contains raw data from the users gathered through their smartwatches and smartphones or information about queries;
  - T4R packet: forwarded to T4RDispatcher, this type of packet contains data used to create, manage, join and spectate runs.
- Authentication Manager: this component is used to manage users and third parties credentials and validate incoming packages;
- D4H Dispatcher: this component analyzes the incoming packet and forwards it to the correct component. A packet can be of 4 types:
  - Subscription packet: forwarded to D4H Subscription Provider, this packet contains information about creating, managing and deleting subscriptions created by third parties users;
  - Query packet: forwarded to D4H Query Service, this packet contains information about executing a query;
  - Single User Request packet: forwarded to D4H Single User Request Service, this packet contains information about a third party user requesting the access to a single user's data;
  - Update packet: forwarded to D4H Update Service, this packet contains information about new raw data gathered from a user's smartwatch or similar device.
- D4H Subscription Provider: this component manages information about creating, managing and deleting subscriptions created by the third parties users. It manages them in the database and also uses the D4H Query Service to perform them;
- D4H Query Service: this component manages all the queries performed on the raw data collected. It executes them and then decides if they can be deployed or not (i.e. not returning results with less than 1000 records if the query was not about a single user);
- D4H Single User Request Service: this component manages the interaction between third parties that can request the access to a single user's data and single users that can accept or reject requests. This component memorizes the request in the database;
- D4H Update Service: this component receives all the new data gathered from users. In order to decide what to do with the new data, the component has to analyze the packet and see what services the user is using:
  - If the packet is marked with the D4H service, then the data contained in the packet is stored in the database;
  - If the packet is marked with the AutomatedSOS service, then the data is sent to the AutomatedSOS Forwarder component;



- If the packet is marked with the T4R service, then the data is sent to the T4R Map Manager.

Note that a packet can be marked with all the 3 services at the same time. This means that a packet can be copied up to 3 times.

- AutomatedSOS Forwarder: this component analyzes all the packets that it receives and if a packet is marked as "Emergency packet", it starts the procedure that calls the most suitable hospital for the client needs;
- T4R Dispatcher: this component analyzes the incoming packet and it forwards it to the correct component. A packet can be of 3 types:
  - Map packet: forwarded to T4R Map Manager, this packet is a request for an update of all the positions of the runners for a given run;
  - Create packet: forwarded to T4R Run Creator, this packet contains information about creating a new run;
  - Join packet: forwarded to T4R Join Run Service, this packet contains information about joining an existing run.
- T4R Map Manager: this component stores all the data needed to show a run in progress, i.e. the current position of all the participants. The D4H Update Service sends the position of all the users of different runs to this component, while the T4R Dispatcher requests all the positions of all the participants of a specific run;
- T4R Run Creator: this component is used to manage the creation of a new run. When a valid request is made, the new run is stored in the database;
- T4R Join Run Service: this component manages the joining process of a user to a run as a participant of a run (not as a spectator). If the request is valid, then a new participant is added to the requested run;
- Provider Adapter: this component job is to act as an intermediary between the TrackMe's components that utilize AWS and AWS itself.

## 2.3 Deployment View

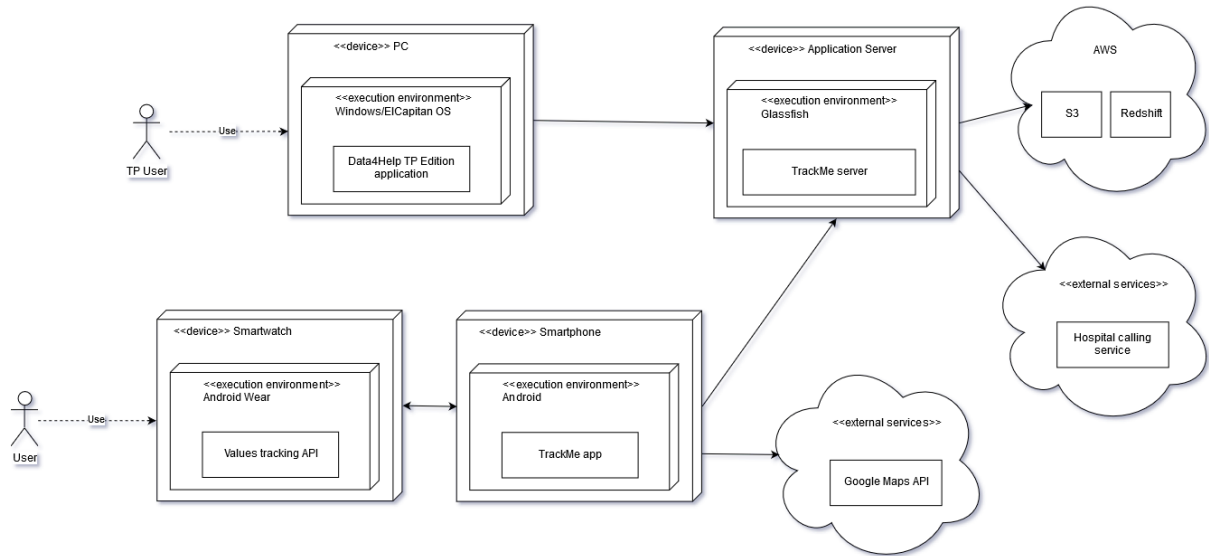


Figure 3: Deployment View

The deployment view shows the distribution of software and hardware in the TrackMe system. The components and their interactions are here described:

- Data4Help TP Edition application is the desktop application developed for third party users. This application makes queries and request to the Data4Help server, and receive updates of subcripted data, or various notification from Data4Help services.
- TrackMe app is a general TrackMe app for smartphone: Data4Help, AutomatedSOS or Track4Run. Each one is able to connect to the TrackMe server, sending updates or managing operations according to the service, and is also able to receive updates and notifications from it. Track4Run must also be able to use Google Maps API in order to display the real time map during a competition.
- Values tracking API: this is a generic smartwatch API which allows the TrackMe app to get data from the device, depending on which kind of device it is.
- TrackMe Server is the core of the TrackMe environment, as it respond to most of the action performed by the users applications. In general, it receives a request from an application, determines the service and does the operation associated to it. Also for some kind of operation it communicates with the AWS services, or the ambulance calling service.
- AWS is the chosen service for the DB deployment. All the accesses to it are made through the server, which stores Data4Help data on it, and also all the data used for the other two services.
- The hospital calling service is the service defined with the D3 in the RASD (section 2.4). The calls directed to it are made only through the TrackMe server.

## 2.4 Runtime View

### 2.4.1 Registering a new user in TrackMe ecosystem

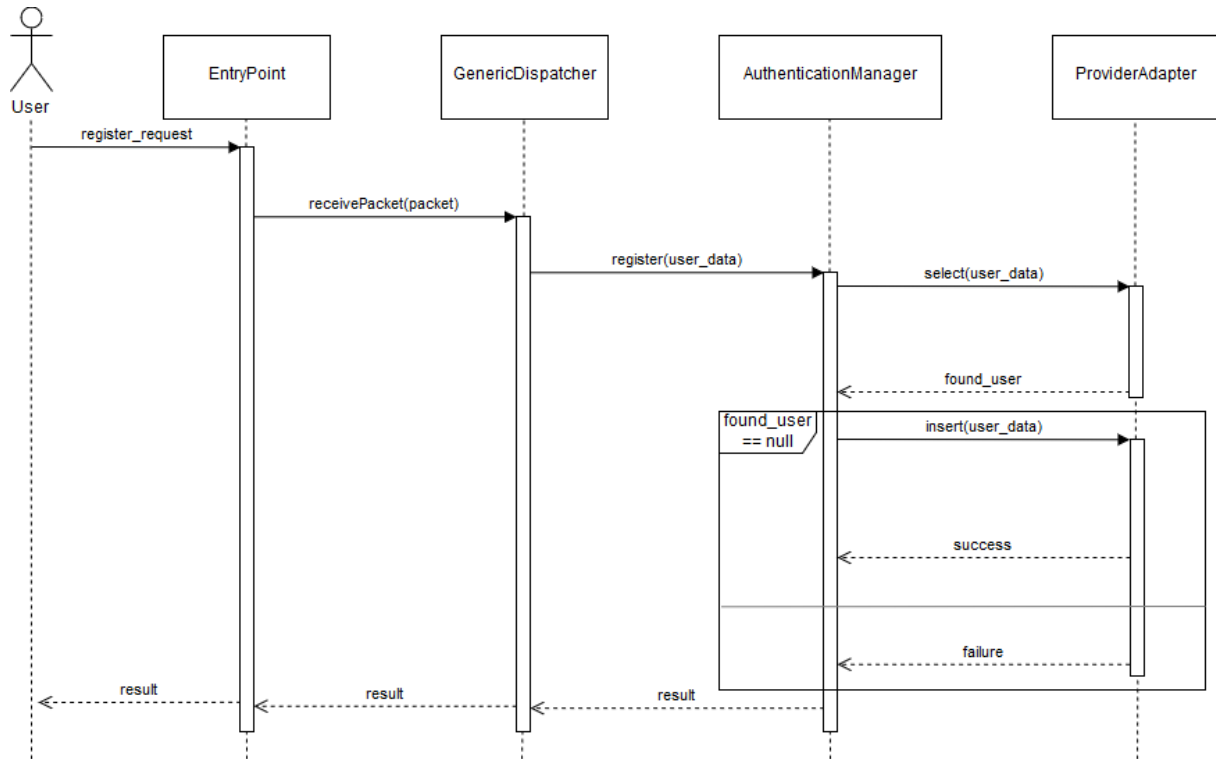


Figure 4: Registering New User

This section shows the process of registering a new user. The first thing that happens is that the incoming packet is read by the Entry Point and then passed to the Generic Dispatcher. The Generic Dispatcher reads the type of the packet and forwards it to the Authentication Manager, because no service (D4H, ASOS, nor T4R) is marked in the packet's header. The Authentication Manager finally extrapolates the actual data from the packet and proceeds creating a new user: firstly it makes a query to check if the user already exists and if it doesn't it proceeds making a query inserting a new user. These two operations are done through the Provider Adapter component.

### 2.4.2 Storing a new raw data packet in Data4Help

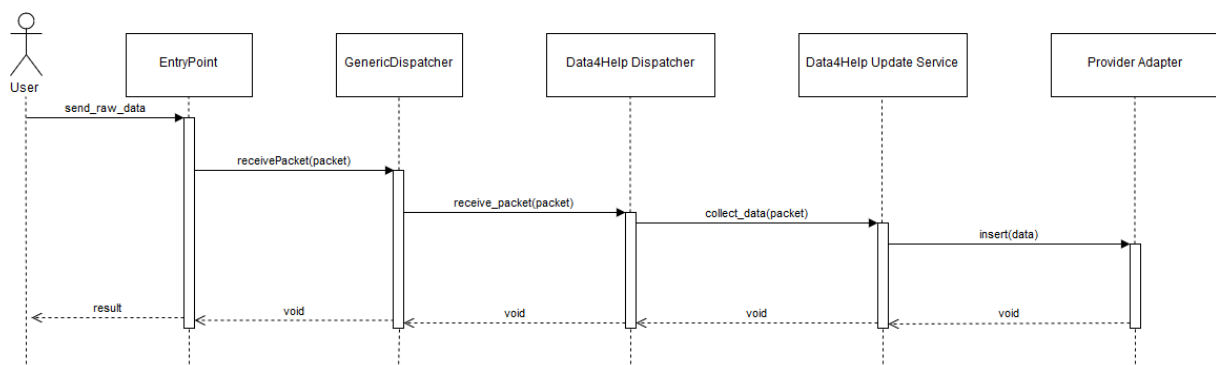


Figure 5: Storing New Data

This section shows the process of storing a new raw data packet in the database. The first thing that

happen is that the incoming packet is read by the Entry Point and then passed to the Generic Dispatcher. The Generic Dispatcher reads the type of the packet and forwards it to the Data4Help Dispatcher. The Data4Help Dispatcher finally sends it to the D4H Update Service which sends it to the proper services. In this example the only service is Data4Help, so the only component that it is used is the Provider Adapter that make the actual INSERT in the database.

### 2.4.3 Emergency detected in AutomatedSOS

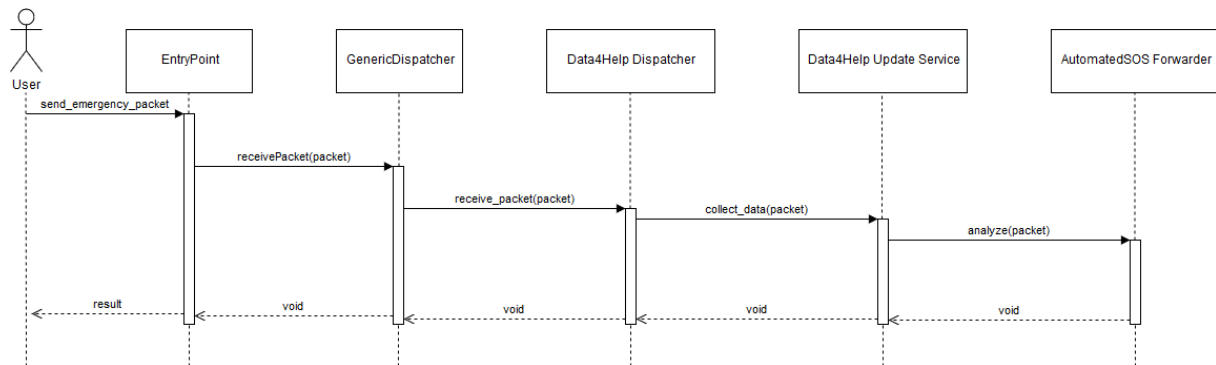


Figure 6: Asos Emergency

This section shows the process of detecting an emergency. The first thing that happen is that the incoming packet is read by the Entry Point and then passed to the Generic Dispatcher. The Generic Dispatcher reads the type of the packet and forwards it to the Data4Help Dispatcher. The Data4Help Dispatcher sends it to the Data4Help Update Service which finally sends it to the AutomatedSOS Forwarder which is warned that there is an actual emergency and start the procedure for calling the most suitable hospital.

### 2.4.4 Retrieving the positions of runners in a run in Track4Run

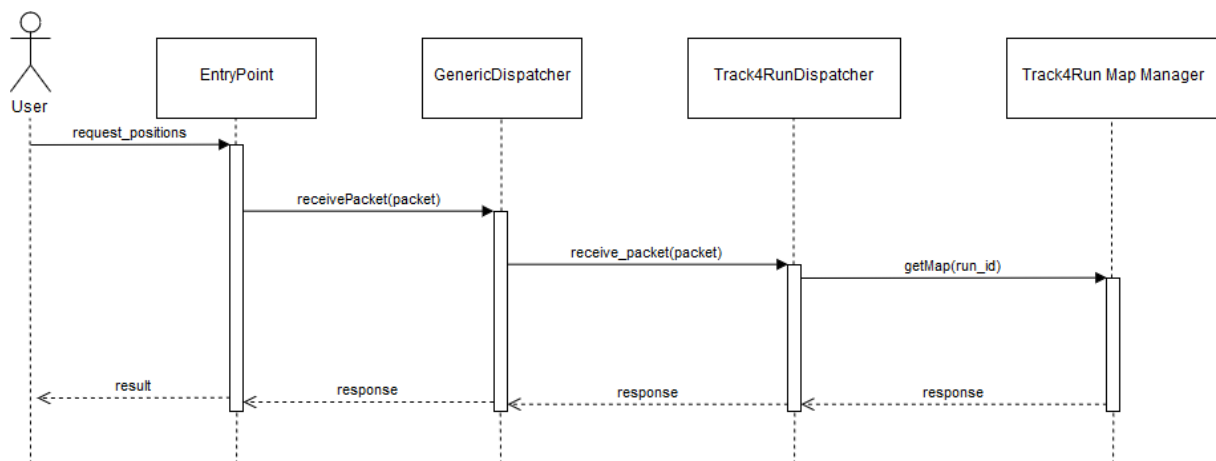
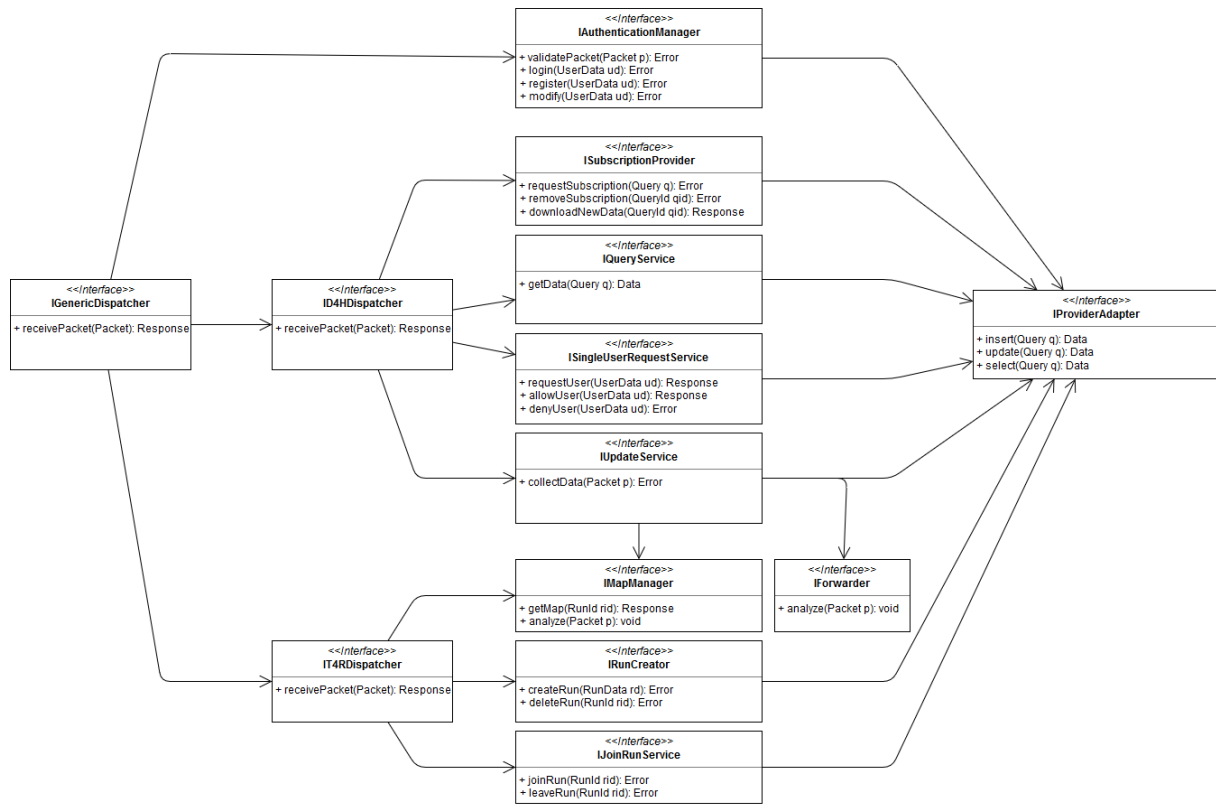


Figure 7: Retrieving Positions

This section shows the process of retrieving the positions of runners in a run. The first thing that happen is that the incoming packet is read by the Entry Point and then passed to the Generic Dispatcher. The Generic Dispatcher reads the type of the packet and forwards it to the Track4Run Dispatcher. The Track4Run Dispatcher finally sends it to the Track4Run Map Manager which packs the data with all the positions of the runners for the specified run and send it back once it is ready.

## 2.5 Component Interfaces

This paragraph shows the interfaces used by the components of the server.



Interface diagram for Software Engineering 2 Project "Track Me" Authors: Leonardo Barilani, William Bonvini, Lorenzo Carnaghi

Figure 8: Component Interfaces

## 2.6 Selected architectural styles and patterns

### 2.6.1 Architectural styles

The adopted architecture for the TrackMe environment is a client-server, 3-tier architecture, divided in: presentation tier, logic tier and data tier.

The presentation tier is formed by all the applications offered by TrackMe, both for mobile devices and PCs. It offers the access to the users to the relative service(s) they need, and does not contain any logic, resulting in a thin client. The Google Maps API are also present in this tier, because they are used only as a presentation function.

The logic tier is composed mainly by the TrackMe server application, and manages the operations performed by all users of all the three services. In this tier are also present external services, which are composed by the ambulance calling service.

The data tier is composed by the Amazon Web Services. It's used by the logic tier, or in detail by the server application, to store and use all the users' data.

The 3-tier architecture allows advantages such as decoupling, maintainability and reusability. Decoupling allows to develop independently one tier from the others, and increases maintainability. Being each tier separated, each one is more easily maintainable: for instance the smartphone apps, which work in a constantly evolving environment and often have to be updated accordingly. Finally, the logic separated from the other tiers allows to reuse components and extend itself with new services.

It's also worth noticing that the architecture adopted is mostly event-based, where users generate events and eventually receive an answer.

### 2.6.2 Patterns

**Model View Controller** The MVC pattern is naturally adapted to the TrackMe 3-tier architecture: clients applications are the View, as they only have a GUI function. The Model is represented by a part of the server, which manages the logic of the applications and partially by the DB and external services. Finally the Controller is the other part of the server, which manages and intradates the various users' inputs and requests. `GenericDispatcher`, `D4HDispatcher`, `T4RDispatcher` and `AuthenticationManager` are the components which form the Controller, the others compose instead part of the Model.

**Data access component** Data access component pattern consists in an abstraction to the access of a large amount of data, to reduce complexity and enable additional data consistency. This is present because of the use of AWS services, and also in the query service for TPs.

**Provider adapter** The provider adapter pattern is used to separate the logic of retrieval of data from the DB from the concrete implementation. This approach allows abstraction and more versatility and maintainability: if in the future another DB service should be adopted instead of AWS, the core of the server would not be refactored or modified. Instead only the provider adapter component would be changed.

**Elastic Queueing** Considering the asynchronous, event-based environment in which the TrackMe server must operate, a dynamic usage of resources is mandatory in order to achieve good performances. The elastic queueing pattern provides as solution a queue for incoming messages to the server, which is used, based on the number of requests, to adjust the number of components in the system and to manage resources for target components in the application.

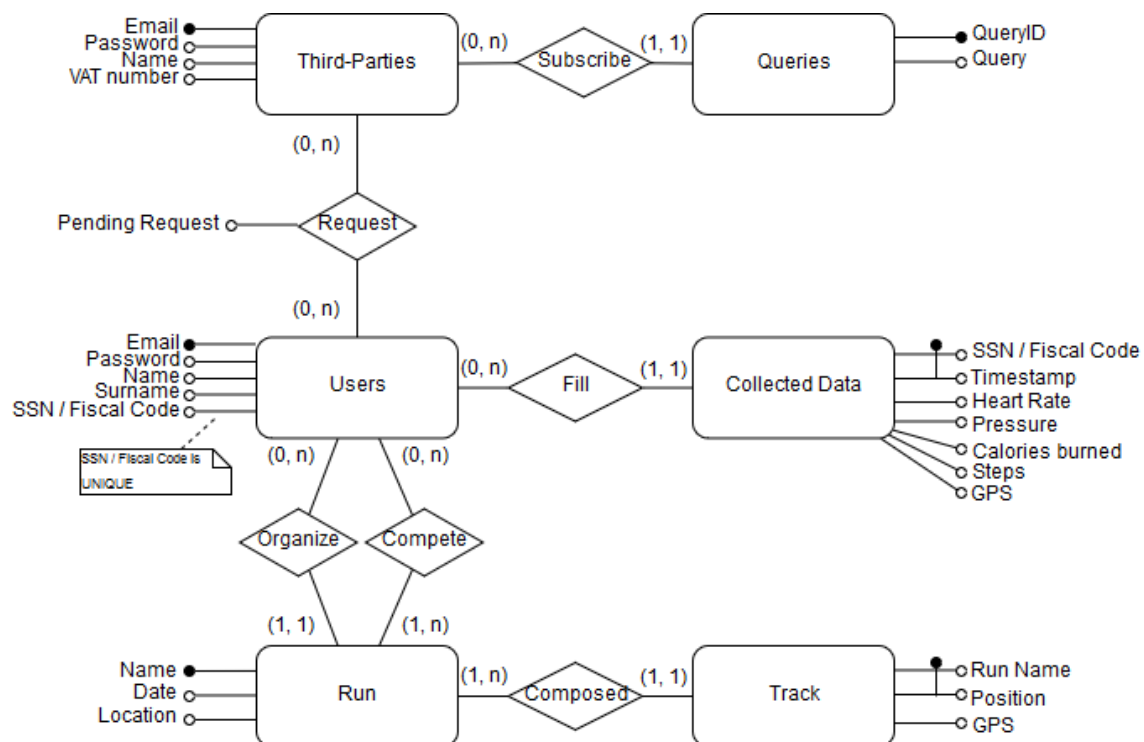
**Stateless Component** For the most part, the server application is composed by stateless component. The only exceptions are `D4HSubscriptionProvider`, which is a Singleton, and `T4RMapManager`. Other components do not need a state to operate, allowing scalability and dynamicity for resource allocation.

**Elastic component** Strictly related to the elastic queueing and stateless component patterns, the elastic component pattern provides dynamicity to the resources granted to components; a behaviour perfectly adapted to the described environment, which could need more or less resources allocated for each component depending on the situation. It's worth noticing that the elastic component pattern is consistently simplified by the fact that the scalable components are mostly stateless.

## 2.7 Other design decisions

### 2.7.1 ER schema

The following is the ER schema of the database. It shows how the users' data is organized for the authentication, the Data4Help service and the Track4Run service. Obviously the AutomatedSOS service doesn't appear because there is no data that need to be stored for a user.



ER Schema for Software Engineering 2 Project "Track Me" Authors: Leonardo Barilani, William Bonvini, Lorenzo Carnaghi

Figure 9: ER Diagram

### 2.7.2 AutomatedSOS optimization

Regarding the AutomatedSOS service there are two optimizations:

- The first optimization consists in memorizing the list of the available hospitals in the configuration files of the Automated SOS Forwarder component and not in the AWS instance. This way when an emergency occurs the component has already all the information that he needs and doesn't have to make queries that can slow down the entire process;
- The second optimization aims at lightening the amount of packets from the user to the server: typically every packet that a user sends is marked with the services that the user is subscribed to,

no matter what happens. With AutomatedSOS this is not true: a packet is marked as an AutomatedSOS packet only if the Automated SOS app detects an emergency. If the user uses only the Automated SOS' app than a packet is sent only if there is an emergency. This way we cut useless packets traffic and we lighten the load on the server.

### 2.7.3 Login optimization

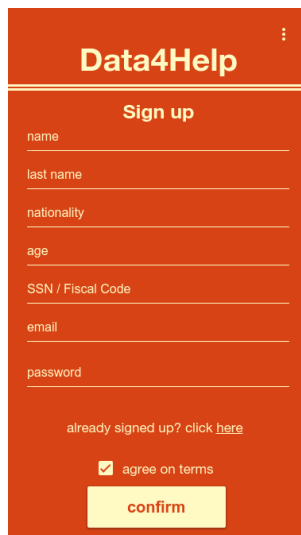
Regarding the user experience the user needs to log in just the first time, then the system provides automatic access using cookies that expire after a period of time.



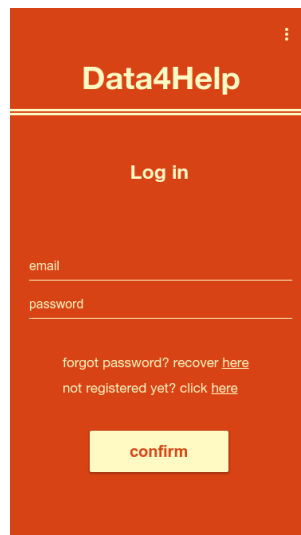
### 3 User Interface design

#### 3.1 Mockups

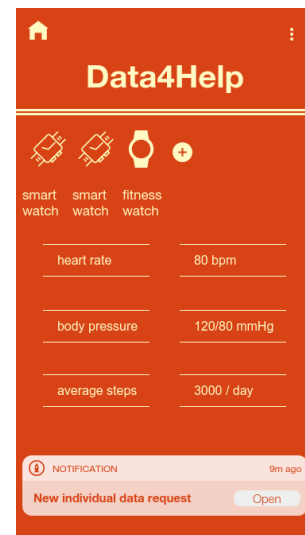
Such section describes the interaction between the third parties and the desktop application of Data4Help. The mockups have already been presented in the RASD, but in this section each of them will be described in detail.

The mockup shows a 'Sign up' form with fields for name, last name, nationality, age, SSN / Fiscal Code, email, and password. There is a checkbox for 'agree on terms' and a 'confirm' button at the bottom. A link 'already signed up? click here' is located above the checkbox.

(a) Data4Help - User - Sign Up

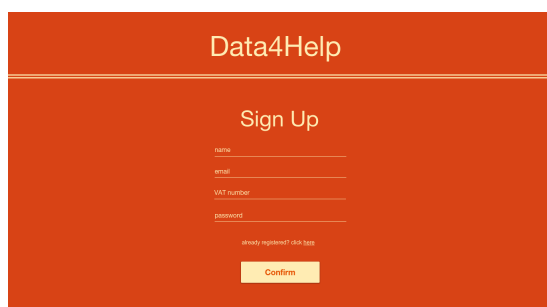
The mockup shows a 'Log in' form with fields for email and password. There are links for 'forgot password? recover here' and 'not registered yet? click here'. A 'confirm' button is at the bottom.

(b) Data4Help - User - Login

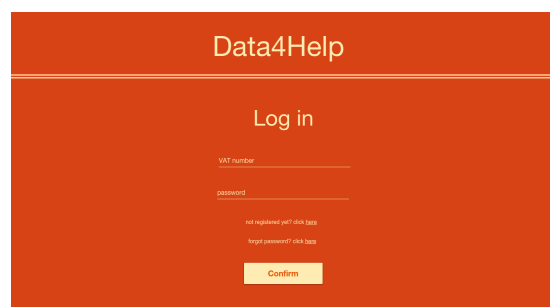
The mockup shows the user's homepage with a header bar containing icons for smart watches and a plus sign. Below the header, there are three rows of data: 'heart rate' (80 bpm), 'body pressure' (120/80 mmHg), and 'average steps' (3000 / day). At the bottom, there is a notification panel with the text 'New individual data request' and an 'Open' button.

(c) Data4Help - User - Homepage

**D4HU - Sign up, Login & Homepage** The Sign Up and Login pages are quite self-explanatory. The homepage of Data4Help's users consists in a list of the values registered by the sensors paired with the device and a list of such devices. It's given to the user the possibility to delete and add devices on his account. Moreover, on the bottom of such page is shown a notification panel, whose goal is to tell the user when a third party is willing to access his private information.

The mockup shows a 'Sign Up' form for third parties with fields for name, email, VAT number, and password. There is a 'Confirm' button at the bottom. A link 'already registered? click here' is located above the button.

(a) Data4Help - Third Party - Sign Up

The mockup shows a 'Log in' form for third parties with fields for VAT number and password. There are links for 'not registered? click here' and 'forgot password? click here'. A 'Confirm' button is at the bottom.

(b) Data4Help - Third Party - Login

**D4HTP - Sign up and Login** The sign up and login pages of the third parties' version of Data4Help are quite minimalistic. Both pages just require the VAT number of the third party and the password. Moreover is offered the chance to switch from login to sign up and viceversa by clicking the underlined text below the password field.



Figure 12: Data4Help - Third Party - Homepage

**D4HTP - Homepage** Once the user logged in, he is shown on the left the last notifications (later called "messages") he received. A notification/message is any of the following events:

- Information about new versions of the application.
- Personal message from Data4Help team's to the third party for any reason (i.e. : communicating the impossibility of satisfying a request)
- Query results' detailed information

Clicking the "show messages" button, the "messages" page gets opened. On the right is displayed the list of requests done by the TP to D4H. Clicking on any of them a file containing the results of the request gets downloaded. Clicking on the "new request" button a "new request" page gets opened.



Figure 13: Data4Help - Third Party - Messages

**D4HTP - Messages** On the left are shown the messages sent by the Data4Help staff to the TP. Once one of such messages gets clicked, the actual message gets shown on the right and a rounded shape appears on the left of the message opened in order to clearly show which message has been opened.



Figure 14: Data4Help - Third Party - Messages - Query Result

Queries results gets shown firstly in such page, as normal messages from Data4Help that contains the following data:

- A written summary of the request
- The day the request was made
- The number of users whose data have been used
- the document itself that contains the result of the query

As explicated in the homepage paragraph, such results gets displayed even in the homepage, the fact that they are shown in the messages section as well is a way of returning more information about the query, not merely the data file. Moreover the home symbol obviously redirects to the homepage.

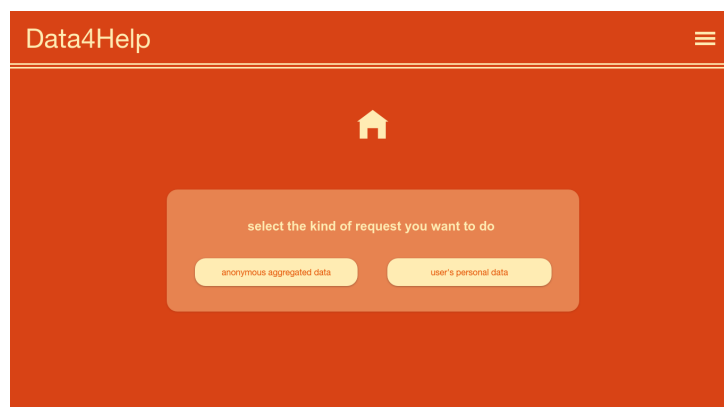


Figure 15: Data4Help - Third Party - New Query Request

**D4HTP - New Request, Anonymous Aggregated Data, User's Personal Data** the new request page consists in the selection between one of the two possible queries, their names are self-explanatory. By clicking on any of them the TP is redirected to the corresponding page.

The screenshot shows the 'Data4Help' web application interface for a 'Third Party - Aggregated Data Request'. The interface is divided into three main sections. The leftmost section, titled 'required data provided by sensors', contains a grid of buttons for selecting sensors: 'body temperature' (sensor 1), 'real time position' (sensor 2), 'heart beat' (sensor 3), 'blood pressure' (sensor 4), 'household temperature' (sensor 5), and 'steps counter' (sensor 6). The middle section contains three input fields for filters: 'specify the geographical area you are interested in...', 'insert the age range in the form XX - XX ...', and 'filter by nationality...'. The rightmost section contains a large text area for 'write here additional information about the request...' and a 'send' button. A 'subscribe' button is located in the top right corner.

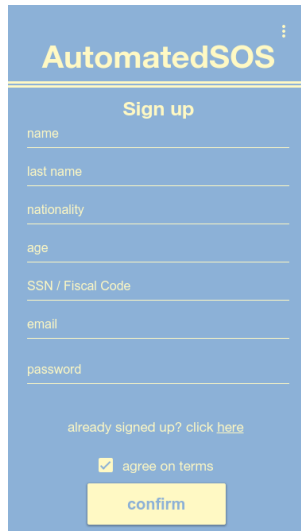
Figure 16: Data4Help - Third Party - Aggregated Data Request

The Anonymous Aggregated Data page consists in three boxes. The leftmost one shows all the available sensors from which D4H can get data, the TP will select the ones he is interested in. Proceeding clockwise the TP is asked what filters he wants to apply to the request (geographical area, age range, and nationality) and in the last box it is asked whether he wants to add any additional information. By clicking on the "subscribe" button he can subscribe to such request, specifying the period of subscription and the interval of time between one result and the next one. This latter scene consists in just those two fields. Since it's a simple functionality, it's left to the developer to design it.

The screenshot shows the 'Data4Help' web application interface for a 'Third Party - Individual Data Request'. The interface is divided into three main sections. The top section contains a text input field for 'Insert name and last name of the user whose data you want us to provide you' with a placeholder 'name of the user'. The middle section, titled 'available data provided by sensors', contains a grid of buttons for selecting sensors: 'body temperature', 'heart beat', 'real time position', and 'blood pressure'. The rightmost section contains a large text area for 'write here the request message, it will be sent to the user...' and a 'send request' button. A 'subscribe' button is located in the top right corner.

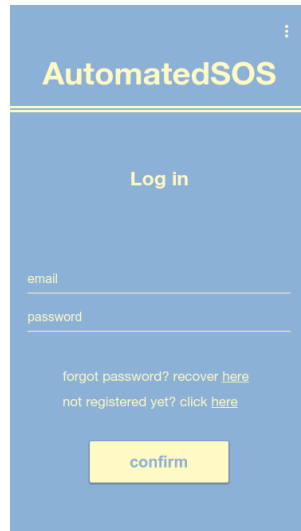
Figure 17: Data4Help - Third Party - Individual Data Request

The User's Personal Data page is similar to the latter page presented. In the "available data provided by sensors" box are shown 4 sensors, the TP can select any of such sensors to be provided to him. The subscribe button works as described earlier.



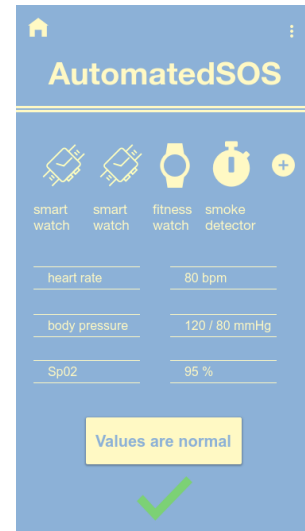
The AutomatedSOS Sign Up screen features a blue header with the title 'AutomatedSOS' and a menu icon. Below the header, the title 'Sign up' is centered. The form includes input fields for name, last name, nationality, age, SSN / Fiscal Code, email, and password. A link 'already signed up? click here' is present. At the bottom, there is a checkbox for 'agree on terms' and a yellow 'confirm' button.

(a) AutomatedSOS - Sign Up



The AutomatedSOS Login screen has a blue header with the title 'AutomatedSOS' and a menu icon. The title 'Log in' is centered. The form includes input fields for email and password. Below the password field, there are links for 'forgot password? recover here' and 'not registered yet? click here'. A yellow 'confirm' button is at the bottom.

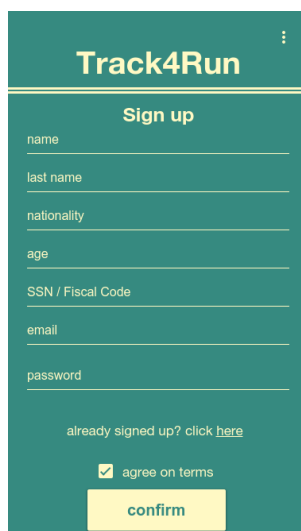
(b) AutomatedSOS - Login



The AutomatedSOS Homepage screen has a blue header with the title 'AutomatedSOS' and a home icon. Below the header, there are icons for smart watch, smart watch, fitness watch, and smoke detector. A table displays health metrics: heart rate (80 bpm), body pressure (120 / 80 mmHg), and SpO2 (95 %). A yellow box at the bottom states 'Values are normal' with a green checkmark.

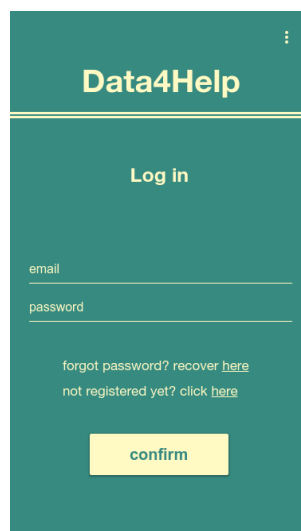
(c) AutomatedSOS - Homepage

**ASOS - Sign up, Login & Homepage** Such mockups are very similar to the ones shown for the users version of Data4Help, the only difference is in the homepage mockup, where the notification panel has been substituted with a text that tells the user whether its values are normals or not, in the latter case the user is told whether the ambulance has been called.



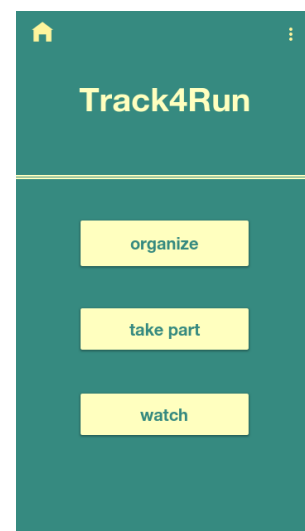
The Track4Run Sign Up screen features a teal header with the title 'Track4Run' and a menu icon. Below the header, the title 'Sign up' is centered. The form includes input fields for name, last name, nationality, age, SSN / Fiscal Code, email, and password. A link 'already signed up? click here' is present. At the bottom, there is a checkbox for 'agree on terms' and a yellow 'confirm' button.

(a) Track4Run - Sign Up



The Track4Run Login screen has a teal header with the title 'Data4Help' and a menu icon. The title 'Log in' is centered. The form includes input fields for email and password. Below the password field, there are links for 'forgot password? recover here' and 'not registered yet? click here'. A yellow 'confirm' button is at the bottom.

(b) Track4Run - Login



The Track4Run Homepage screen has a teal header with the title 'Track4Run' and a home icon. Below the header, there are three yellow buttons: 'organize', 'take part', and 'watch'.

(c) Track4Run - Homepage

**T4R - Sign up, Login & Homepage** The homepage of Track4Run gives the user the chance of choosing between three possible actions to be performed.

(a) Track4Run - Organize

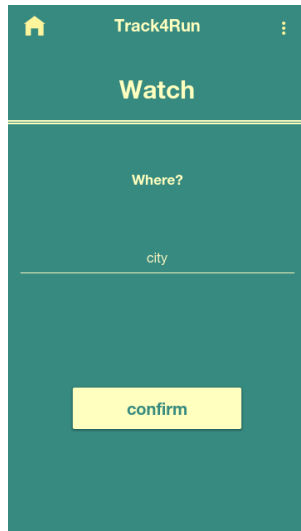
(b) Track4Run - Organize - Set Path

**T4R - Organize** When the user chooses to organize a run, he is first brought to a page in which he gets asked to fill a form with the name of such run, the date and hour of its occurrence and the city where it will be held. After clicking on the "confirm" button he gets asked to draw the path. A possible solution for such interaction is shown in the figure "Track4Run - Organize - Set Path": the user selects one out of a number of possible nodes (intersections) in the city he has selected and afterwards he clicks on the intersection where he wants to change direction or to end the run. In the second case, after having clicked on the second node he would click on the "end here" button. In the mockup the nodes are displayed in yellow, the street which is getting chosen in orange and the path temporarily selected in water green.

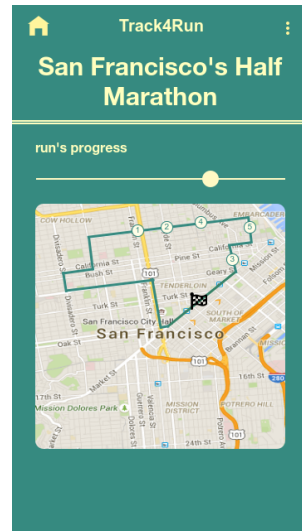
(a) Track4Run - Take Part - Step 1

(b) Track4Run - Take Part - Step 2

**T4R - Take Part** Mockups of the pages displayed after the clicking of the "Take Part" button in the homepage.



(a) Track4Run - Watch



(b) Track4Run - Watch - Map

**T4R - Watch** Mockups of the pages displayed after the clicking of the "Watch" button. the figure (b) represents a live run: the circles within the path represent the participants, identified by an id number, which is supposedly given to them in the moment of the registration to the run. The flag represents the starting point, and the orange arrows aside of the path represent the direction of travel. Any omitted detail is left to the developer to be thought. The mockups presented offer just a guideline for the development of the user interface.

### 3.2 User Experience Diagrams

The following diagrams have been written in order to offer a better understanding of the interaction between the users and the applications. It is suggested to look to such diagrams while reading the description of the corresponding mockups. It is not explicitly shown in the diagrams, but from any page it's possible to go to the homepage the developed apps.



Figure 24: UX Diagram - Data4Help - Third Party





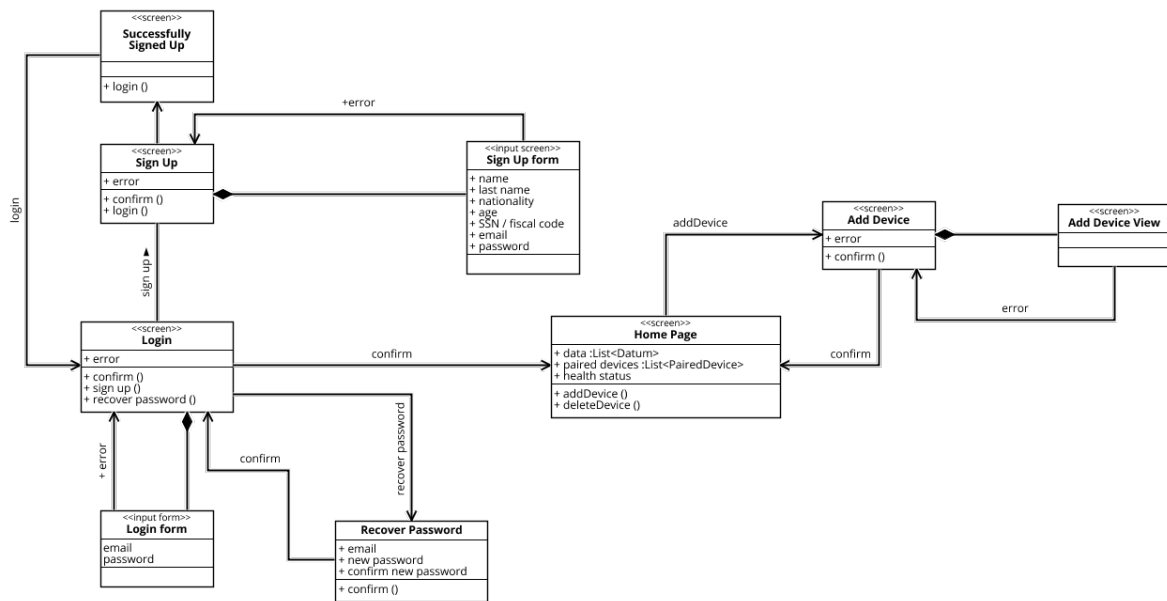


Figure 25: UX Diagram - AutomatedSOS - User

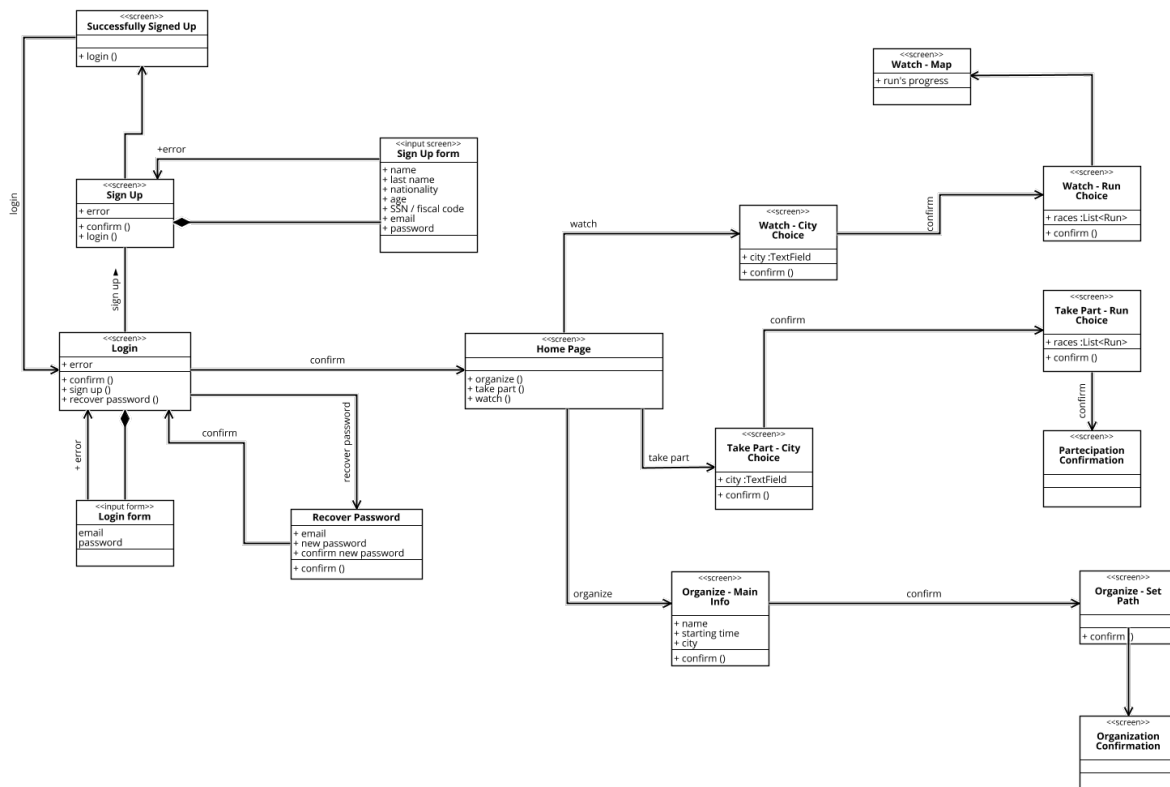


Figure 26: UX Diagram - Track4Run - User

## 4 Requirements traceability

This section consists in mapping each requirement of the project with the the components that are partially or completely responsible for the realization of such requirement. In order of better showing the mapping we firstly mapped the goals to the components, indicating what requirements have to be fulfilled by such goals (such last indication was already present in the RASD). Afterwards requirements are directly linked to components.

### 4.1 Mapping Goals to Components

- G1 - The Data4Help application gives constantly new data accordingly to users' activities.  
[R8], [R9], [R10], [R11], [R13], [R14], [FR1];
  1. AWS
  2. D4HQueryService
  3. D4HDispatcher
  4. GenericDispatcher
- G2 - Registered TPs (Third Parties) can send requests to specific users if they know their SSN or CF in order to retrieve specific data. These users can accept or refuse the proposal.  
[R2], [R3], [R8], [R10],[FR1],[FR2];
  1. D4HSingleUserRequestService
  2. D4HDispatcher
  3. GenericDispatcher
- G3 - Registered TPs have access to make queries to Data4Help's database in order to get anonymized data by filtering by one or more parameters as they need.  
[R1], [R2], [R3], [R7], [R8], [R10], [R11], [FR3];
  1. D4HQueryService
  2. AWS
  3. D4HDispatcher
  4. GenericDispatcher
- G4 - Results for TPs' queries are given only if the number of matches is greater or equal than 1000.  
[R1], [R2], [R7];
  1. D4HQueryService
- G5 - Each user of the services offered by TrackMe must be identifiable within the system in order to gather his data and communicate with him.  
[R6], [FR1], [R12], [FR4];
  1. AuthenticationManager
  2. AWS
- G6 - AutomatedSOS sends automatically an ambulance request when and only when user's values are below a specific critical threshold, specifying the user's position.  
[R5], [R6], [R9], [R10], [R11], [R14], [FR5], [FR6];
  1. D4HUpdateService

## 2. AutomatedSOSForwarder

- G7 - Users of Track4Run are able to create a run defining the precise path.  
[R1], [R6], [R7], [R14][FR7],[FR8];
  1. T4RRunCreator
  2. AWS
- G8 - Users of Track4Run are able to take part in a created run as runners.  
[R7], [R14], [FR9];
  1. AWS
  2. T4RJoinRunService
- G9 - Users of Track4Run can see real-time runners' positions in runs such runners are subscribed in.  
[R4], [R7], [R11];
  1. T4RMapManager
  2. AWS

## 4.2 Mapping Requirements to Components

- R1 - Hardware interface: This application needs a data warehouse to store all the user data and to allow high performance researchs with large amounts of data.
- AWS
- R2 - Software interface: Amazon Redshift as a data warehouse on top of Amazon S3 to store all the data in an efficient manner;
- AWS
- R3 - Software interface: Google Maps for converting stored GPS data into human readable data for the third parties clients;
- Google Maps
- R4 - Software interface: Google Maps for the Track4Run real-time map service.
- Google Maps
- R5 - Software interface :The API interface for the automatic message sent to the nearest hospital.
- Hospital APIs
- R6 - Both users and third parties will have to use HTTPS to communicate with the TrackMe server, because security and privacy is an important factor in both cases;
- EntryPoint
- R7 - Performance requirements regarding Data4Help and Track4Run: they both have to manage a large amount of people that can range from 0 (i.e. an event organized through Track4Run) to 10000 (i.e. a fair amount of user that we should expect from Data4Help) based on the current users requests.

- GenericDispatcher
- D4HDispatcher
- D4HUpdateService
- AWS
- T4RDispatcher
- T4RMapManager
- T4RJoinRunService
- T4RRunCreator

R8 - Internally the data that is collected through Data4Help can be stored in any way that is comfortable for the developers to work with, but when it is delivered to the third parties it must use a standard format. This means that:

R8.1 - the TP can choose of receiving the results of its requests in the form of a summary in a PDF file.

R8.2 - the TP can choose of receiving the results of its request in a database file, in such a way that the third parties can manipulate the data the way they like.

R8.3 - The TP can request both the formats above.

- D4HQueryService

R9 - Bandwidth may be a constraint, because developers need to pay attention to send only the essential data from the apps to the server, because in an hypothetical situation the application may drain too much mobile data from the users' smartphone. To achieve this is possible to set a reasonable amount of time to be waited between each single user's update of his/her values to the server. On the contrary, AutomatedSOS doesn't have any bandwidth limitation.

- Data4Help user's app

R10 - Reliability is the main concern for AutomatedSOS, because of the nature of the service itself, and surely it is important in the development of the Data4Help module. Secondly, reliability is not the main concern in Track4Run. The whole system should at least be 99.999% reliable.

- GenericDispatcher
- D4HDispatcher
- AutomatedSOSForwarder
- D4HUpdateService

R11 - Availability: AutomatedSOS needs a six 9s (99.9999%) availability, due to the high importance availability plays for this service, while Data4Help and Track4Run need at least a five 9s (99.999%) availability.

- GenericDispatcher
- D4HDispatcher
- AutomatedSOSForwarder
- D4HUpdateService

R12 - Security: Anonymization must be granted giving the fact that we are working with very sensible data. Since AutomatedSOS and Track4Run talk with Data4Help, this means that the communication between the modules must be protected as well.

- AuthenticationManager
- D4HQueryService

R13 - Maintainability is a big concern for all the three services. Given the fact that the application is mainly developed for Android and iOS, it must be easily maintainable, because of the frequent updates of both the operating systems. This translates into the decoupling of the components of the system.

- No particular component is responsible for this requirement: such responsibility has to be found in the way the components interact with one another.

R14 - Portability: The application that is developed for Data4Help users must work for all iOS' and Android's updates by the date of launch of the app. The same can be applied to Track4Run and AutomatedSOS because it's not easy to foresee how smartwatches' and smartphones' market will change in the future. The desktop application of Data4Help instead has not the same portability requirements, because, being developed for the Windows operating system, we can rely on his well established legacy of portability; No component of the server has such responsibility. It's duty of the client's apps developers to make the services portable from one OS to the other.

- Client

## 5 Implementation, integration and test plan

### 5.1 Implementation plan

#### 5.1.1 Overview

Being the TrackMe environment divided in components and subcomponents, the implementation plan will be performed by ordering and completing one or more subcomponent at a time, based on the dependencies between themselves. First the major tasks will be identified, then a more precise implementation plan will be presented.

#### 5.1.2 Major tasks

The most complex component is the server application, thus should be one of the most important concern. Considering the component view (sec 2.2), one can see that the general flow of interface usage is from left to right, with some parallel subcomponent in terms of interface usage. Considering this, the implementation of the server application is divided in 4 groups of subcomponents:

1. the ones which communicate directly with the DB
2. the ones which use interfaces of the first group or that have no other dependencies
3. the ones which use interfaces of the second group
4. the dispatchers of incoming packets and the entry point

Another major tasks which emerges is the protocol of the format of the packets. Such protocol should be completed as soon as possible, because by defining early the protocol, the various TrackMe applications can be developed in parallel with the server. This is possible because these applications don't contain almost any logic (and because of this, these components are not further analyzed in this section), and as prerequisite to develop them they need only to know how to send and receive information from the TrackMe server.

#### 5.1.3 Implementation Schedule

The components and subcomponents of the system are grouped and ordered in the following way:

1. ProviderAdapter and protocol of packets
2. D4HQueryService, D4HSingleUserRequestService, AutomatedSOSForwarder, T4RMapManager, T4RRunCreator, T4RJoinRunService
3. D4HSubscriptionProvider, D4HUpdateService, AuthenticationManager
4. D4HDispatcher, T4RDispatcher
5. GenericDispatcher, EntryPoint

Concerning the client applications, they can be developed independently after group 1 has been completed, in particular after the packet protocol has been defined. They do not require any particular order of developing, being independent each one from the other, and can be developed in parallel or sequentially depending on the available resources.

The protocol can be of any type or structure, such as a JSON, but should be kept in mind that given a possible growing amount of users, it should be defined paying attention to be as efficient as possible for a packet both on space usage, for a faster and lighter internet connection and time employed to extract information from it.

## 5.2 Integration and test plan

### 5.2.1 Overview

The integration and test plans will be actuated in parallel with the implementation plan. In particular, a bottom-up approach will be adopted. Such approach has been chosen given the relative simplicity of the project. Given the fact that in the TrackMe server environment each subcomponent in general relies on the service(s) provided by the implementation of the interface(s) required, and that each subcomponent has one functionality (or, if there are more than one, they are strictly related), the integration between each two subcomponents will be done only after the full completion of the one offering the interface.

It is assumed that the AWS will be already available from the beginning of development, also because its adapter is the first subcomponent to be implemented on the server application. The same goes for the hospital calling service, which will be used to implement the AutomatedSOSForwarder.

### 5.2.2 Integration plan

The integration plan will proceed by grouping the subcomponents of the server application into several categories, accordingly to what has been described in the implementation plan. The categories are hence the following:

1. Database adapter
2. Components interfacing only the DB adapter
3. Components interfacing components of the second category
4. Service dispatchers
5. High level dispatcher
6. EntryPoint
7. TrackMe client applications

The integration is further described, specifying which components have to be integrated to which other(s):

**Database adapter** is composed only by the ProviderAdapter component, and must be integrated only with the AWS external component. The specific integration is:

- ProviderAdapter, AWS

**Components interfacing only the DB adapter:** this group is constituted by all the subcomponents which directly use uniquely the ProviderAdapter (which is the DB adapter), and must be integrated with it. The full list of these subcomponents and their integration is:

- AuthenticationManager, ProviderAdapter
- D4HQueryService, ProviderAdapter
- D4HSingleUserRequestService, ProviderAdapter
- T4RRunCreator, ProviderAdapter
- T4RJoinRunService, ProviderAdapter

**Components interfacing components of the second category:** here are included all subcomponents which have one or more dependencies with subcomponents of the previous group(s). Therefore they need to be integrated after the completion of the previously mentioned group. The full integration list is:

- D4HSubscriptionProvider, ProviderAdapter
- D4HSubscriptionProvider, D4HQueryService
- D4HUpdateService, ProviderAdapter
- D4HUpdateService, AutomatedSOSForwarder
- D4HUpdateService, T4RMapManager

**Service dispatchers** are the dispatchers of the specific service: Data4Help and Track4Run (being AutomatedSOS exploiting Data4Help services). They need to be integrated with each subcomponent of their service, as they must channel the packet to the correct subcomponent. The full integration list is:

- D4HDispatcher, D4HSubscriptionProvider
- D4HDispatcher, D4HQueryService
- D4HDispatcher, D4HSingleUserRequestService
- D4HDispatcher, D4HUpdateService
- T4RDispatcher, T4RMapManager
- T4RDispatcher, T4RRunCreator
- T4RDispatcher, T4RJoinRunService

**High level dispatcher** is only composed by the subcomponent GenericDispatcher, which must be integrated with the AuthenticationManager for identification purposes and the previously mentioned service dispatchers, to which has to send packets according to the service. The full integration list is:

- GenericDispatcher, Authentication Manager
- GenericDispatcher, D4HDispatcher
- GenericDispatcher, T4RDispatcher

**EntryPoint** is the last subcomponent to be integrated, and must only be integrated with the high level dispatcher. The precise integration is:

- EntryPoint, GenericDispatcher

**TrackMe client applications:** finally, being the server application completely integrated at this point, each application can be integrated with the server one independently. In terms of high level components, the integration list is:

- Data4Help app, server application
- Data4Help TP Edition application, server application
- AutomatedSOS app, server application
- Track4Run app, server application



### 5.2.3 Integration details

In this section the previously mentioned groups of integration are further analyzed and graphically represented.

Each dashed arrow represent the dependency and the verse of the integration. If multiple arrows point out from a component, that component will be integrated with two or more components separately. If multiple arrows point to a component, that component will be used in integration by multiple components in the same group. In each group the order of integration is not relevant.

**Database adapter:** this group is composed by the ProviderAdapter component alone, and has the top priority in development because the subsequent subcomponents will require to interface themselves to the AWS database.



Figure 27: Integration group 1

**Components interfacing only the DB adapter:** all the subcomponent in this group need to interface themselves directly to the DB (which has been abstracted through the ProviderAdapter), and do not require any other interface, thus are next in the dependencies hierarchy. All of them interfaces themselves to the same subcomponent, the ProviderAdapter:

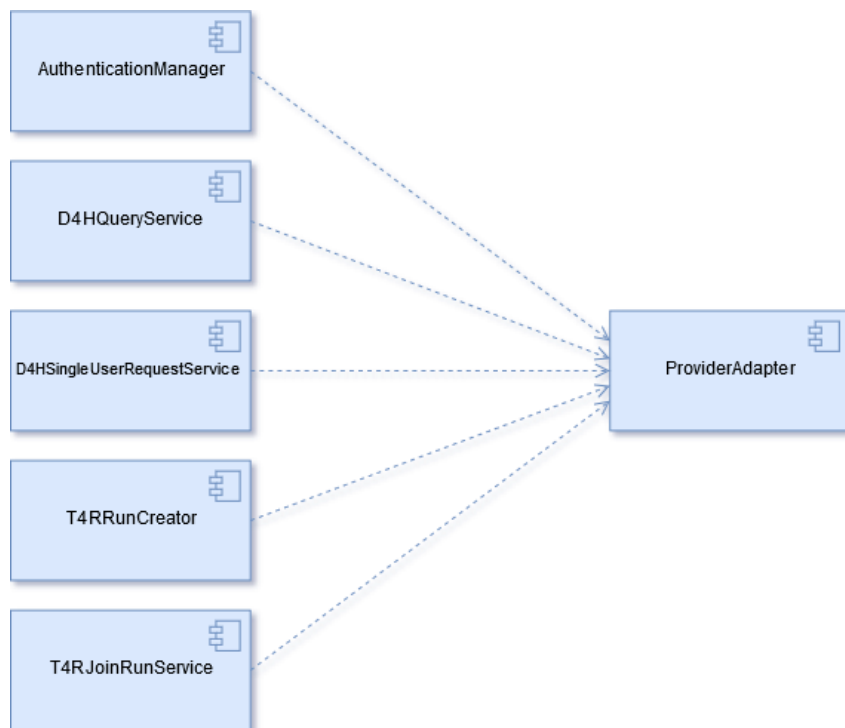


Figure 28: Integration group 2

**Components interfacing components of the second category:** these are the component next in the hierarchy, which use components of group 1 and/or 2. These are services interfacing both the Provider-Adapter and which rely also on one or more subcomponents already developed.

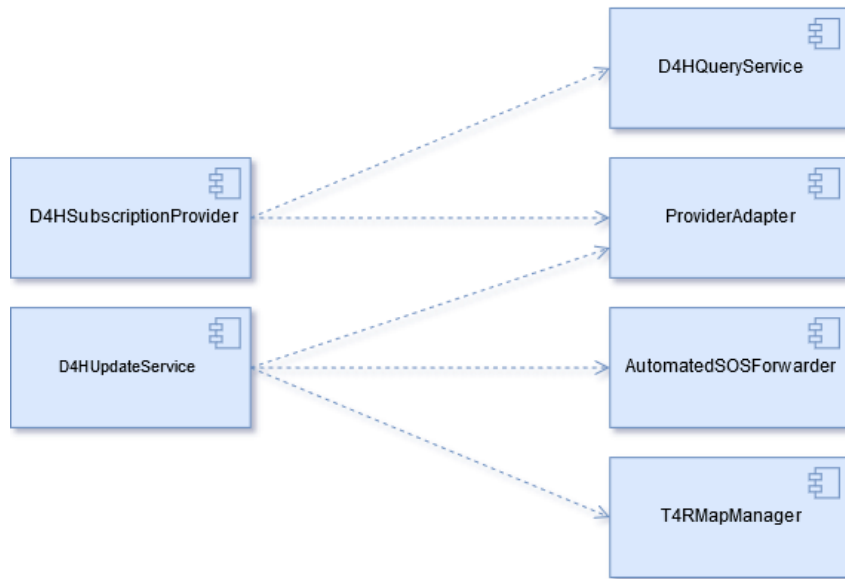


Figure 29: Integration group 3

**Service dispatchers:** the service dispatchers are the two lower-level dispatcher designated to forward packets of a service to the specific task of a service. Therefore they must rely on such tasks, implemented by the previous groups' subcomponents. They must be integrated in parallel with multiple components.

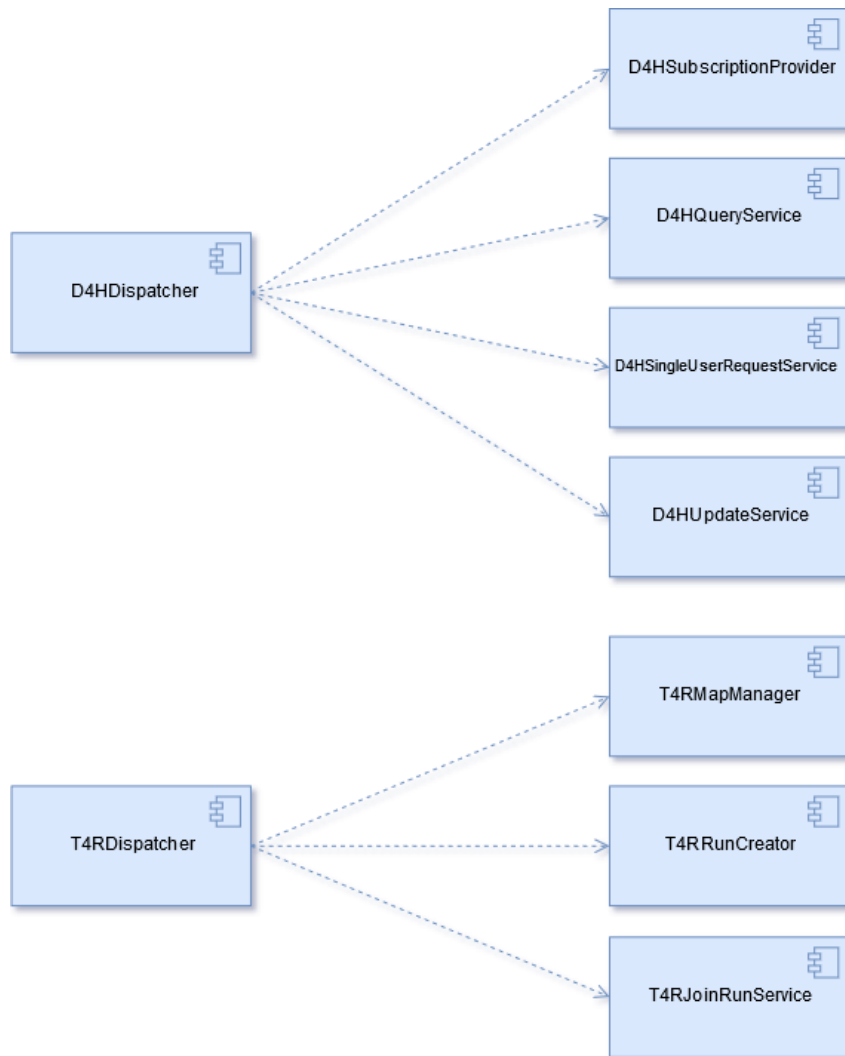


Figure 30: Integration group 4

**High level dispatcher:** the high level dispatcher group is composed by the GenericDispatcher alone, as previously said, and must be integrated with the service dispatchers and the AuthenticationManager. The integration step is hence composed by integrations of the same component with 3 higher (in the hierarchy) components.

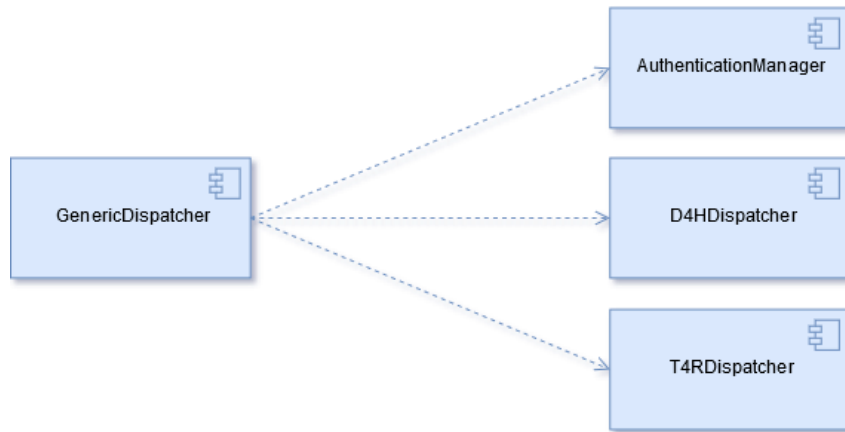


Figure 31: Integration group 5

**EntryPoint:** the entry point is the last subcomponent to be integrated, and has to be integrated only with the GenericDispatcher, as its task is to analyze incoming packets and forward them to the dispatcher if complete and correct.



Figure 32: Integration group 6

**TrackMe client applications:** this last group is formed by higher level component, which are the various TrackMe client applications. At this point the server application is fully integrated and the applications require to be integrated with it. This is the last step of integration.

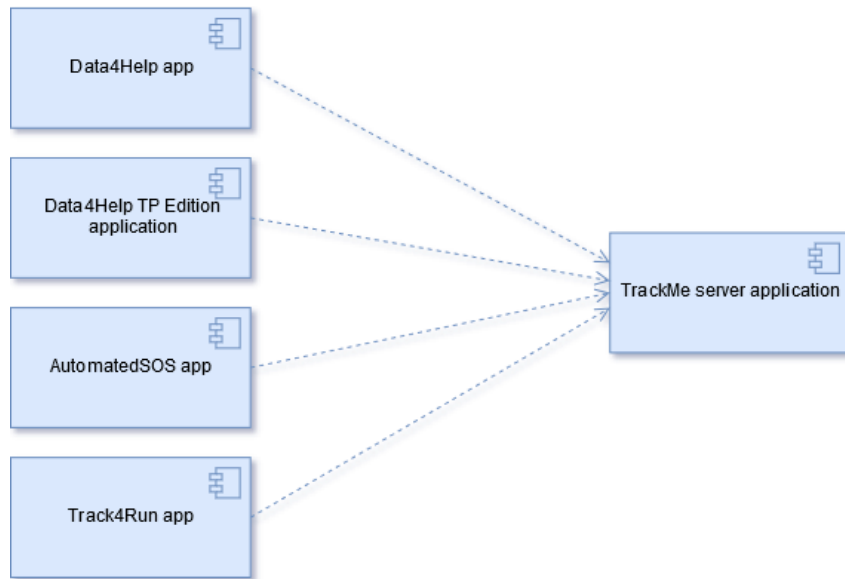


Figure 33: Integration group 7

#### 5.2.4 Integration test plan

The adopted integration test strategy is a bottom-up approach, given the already mentioned structure of the system. For each group of implementation, the single units will be tested through the creation of drivers, which have to test the most possible and most critical cases. In this sense, while building the system, a white-box approach will be used for each single subcomponent. This means that each subcomponent will have a driver to test its functionalities. Such driver will be replaced when integrated to the actual subcomponent. This approach allows to find more easily errors and critical task concerning single components.

Is advised that the integration between subcomponents is done only after a full test is performed successfully on the subcomponent offering the interface. In this way is assumed that the subcomponent on which the new subcomponent relies is fully operative: testing can be hence done on the newer subcomponent with a new driver.

Is also to notice that, having been grouped up the server application subcomponents in the integration plan, the testing on each component in a group can be done independently, as subcomponents in the same group never depend one on another.

When the server application will be completed, a black-box test will be required. This will be done for different reasons:

- A global test is required to acknowledge the correct integration of the whole system.
- A black-box approach allows to eventually find different kinds of problems which could not emerge in white-box testing for single units, such as bottlenecks in the system.
- A stress test is required to test the elasticity of the server application, which has to be flexible and allocate dynamically resources based on the number of the requests received.

## 6 Effort Spent

### 6.1 Leonardo Barilani

Section	Hours
Introduction	2
Architectural design	19
User Interface design	1
Requirements traceability	3
Implementation, integration and test plan	2

### 6.2 William Bonvini

Section	Hours
Introduction	2
Architectural design	7
User Interface design	15
Requirements traceability	9
Implementation, integration and test plan	2

### 6.3 Lorenzo Carnaghi

Section	Hours
Introduction	2
Architectural design	11
User Interface design	2
Requirements traceability	2
Implementation, integration and test plan	8