Politecnico di Milano
AA 2018-2019

POLITECNICO
MILANO 1863

Computer Science and Engineering
Software engineering 2

# Data4Help DD

Design Document
Version 1.0
14/11/18

Diego Piccinotti, Umberto Pietroni, Loris Rossi

Table of Contents

# 1 Introduction

## 1.1 Purpose

The purpose of this document is to further analyze the design and architectural choices for the system to be. Whereas the RASD presented a general view of the system and its features, this document will detail those concepts by showing components of the system, its run-time behavior, deployment plan and algorithm design.

The document will therefore contain a presentation of:

- Overview of the high-level architecture
- Main components and their interfaces provided one for another
- Runtime behavior
- Design patterns
- Algorithm design of the most critical parts of the application
- Implementation plan
- Integration plan
- Testing plan

## 1.2 Scope

*(An SDD shall identify the design stakeholders for the design subject.*
*An SDD shall identify the design concerns of each identified design stakeholder.*
*An SDD shall address each identified design concern.*

*Stakeholders are those people, groups, or individuals who either have the power to affect, or are affected by the endeavor you're engaged with.*
*)*

## 1.3 Definitions, Acronyms, Abbreviation

### *1.3.1 Definitions*

- **User**: individual who allows *Data4Help* to monitor his location and health status.
- **Third party**: individual or organization registered to *Data4Help* which can request users' data.
- **Data collection**: gathering of users' data through a wearable device.
- **Anonymized data**: data about more than 1000 users whose personal information has been previously removed so that they are not directly relatable to the system's users.
- **Elderly**: user who is subscribed to *AutomatedSOS* and is older than 60 years old.
- **Risk threshold**: Set of boundary health parameters defined for each elderly. If monitored values of the user's health parameters get below these boundaries, an SOS request is placed to an external ambulance provider.
- **Athlete**: user who participates in a run.
- **Run organizer**: third party who can manage runs for athletes.
- **Spectator**: non-registered individual who follows a run through a map with runners' positions.
- **Wearable**: a personal device provided with biometric sensors and GPS given to each user for free after the registration process.

### *1.3.2 Acronyms*

- **API**: Application Programming Interface
- **DB**: Database
- **DBMS**: Database Management System
- **DD**: Design Document
- **GUI**: Graphical User Interface
- **RASD**: Requirements Analysis and Specifications Document
- **GPS**: Global Positioning System
- **GSR**: Galvanic Skin Response

### *1.3.3 Abbreviations*

- **Gn**: $n^{th}$ goal.
- **Dn**: $n^{th}$ domain assumption.
- **Rn**: $n^{th}$ functional requirement.
- **Rn-NF**: $n^{th}$ nonfunctional requirement.

## 1.4 Reference Documents

- IEEE Standard on Software Design Descriptions (*IEEE Std 1016-2009*)

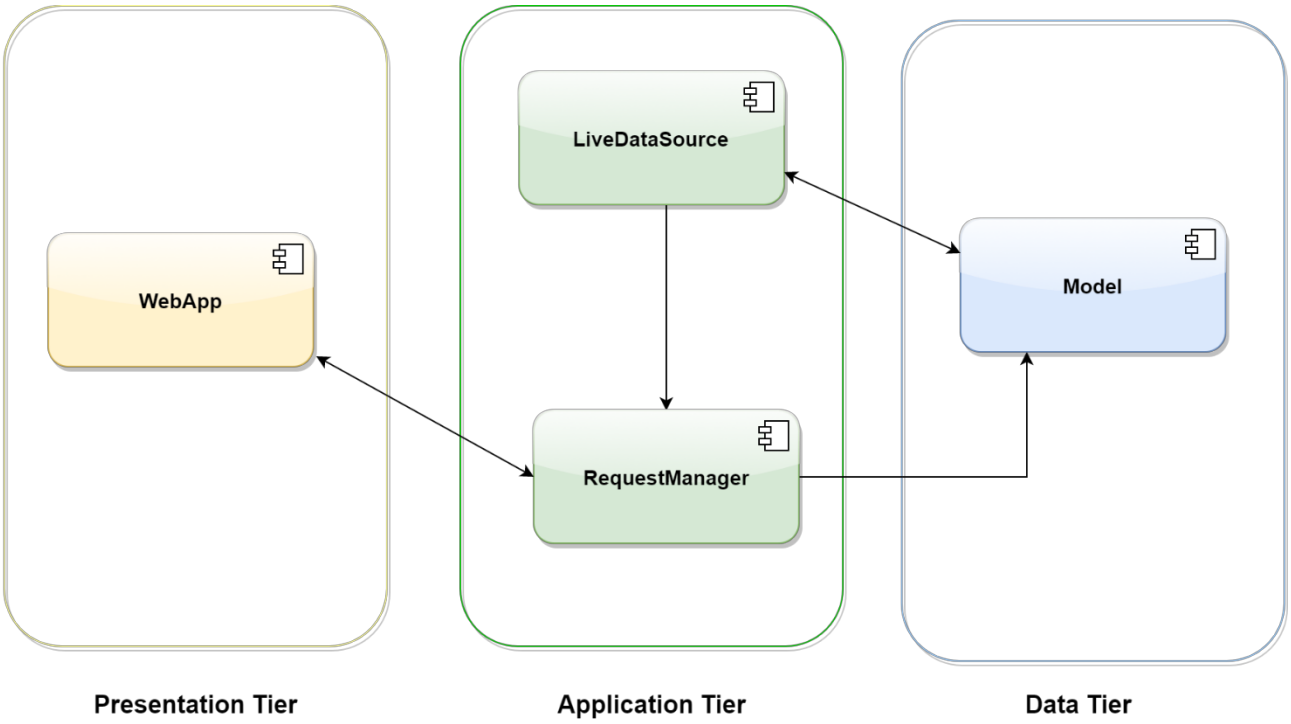
## 1.5 Document Structure

# 2 Architectural Design

## 2.1 Overview:

*(High-level components and their interaction)*

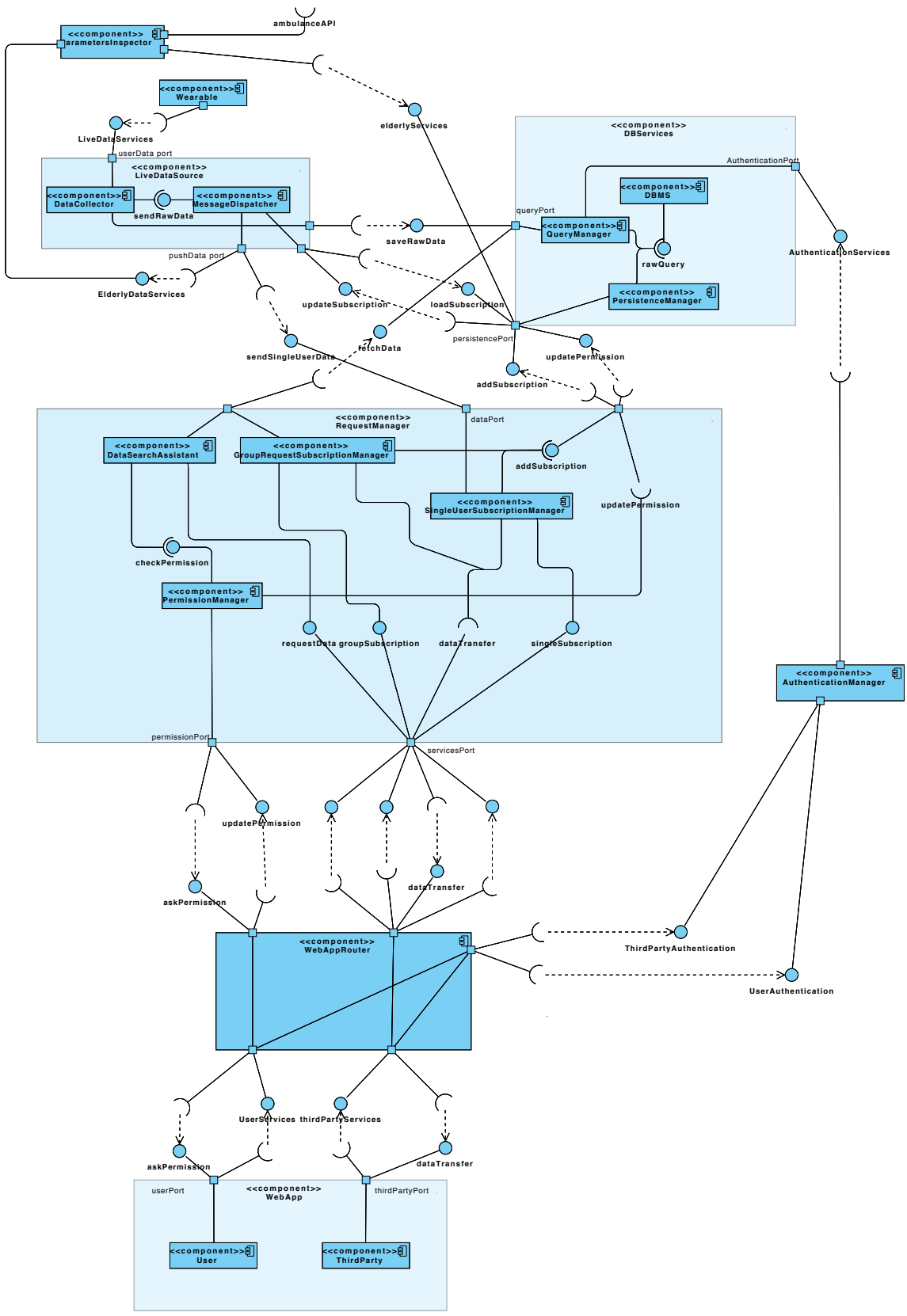
This system is based upon a three-tier architecture, divided in three "layers" of logical computing. The first one is Presentation Tier which is the front-end and consist of the user interface of the web app and all the client logic. Secondly the Application Tier which contains the functional business logic and lastly the Data Tier that is responsible for the database storage system and data access.

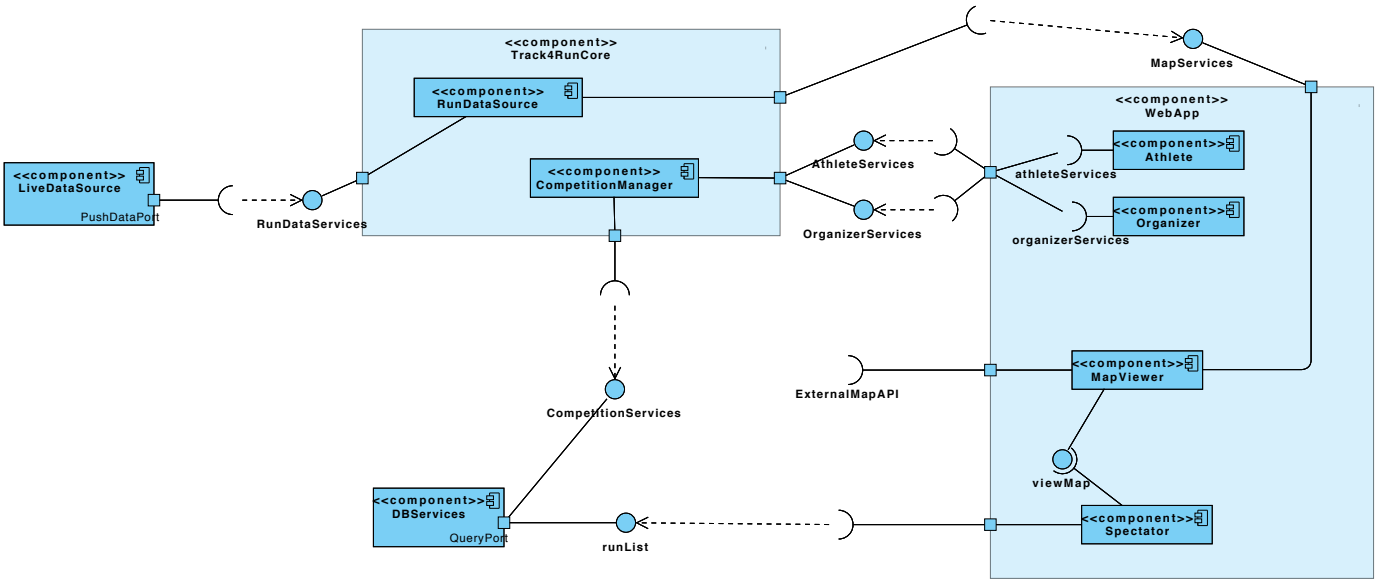



The high-level component related to the presentation layer is the WebApp component, whose task is to show the user interface and react to user input.
The data tier corresponds to the Model component that store data in the database and make queries.
The application tier is composed by two main components: LiveDataSource, which is responsible for collecting users' data and then redistributing them immediately to third parties or submit them to the Model, and RequestManager which contains all the logic related to third parties' requests and subscriptions.

Track4Run and AutomatedSOS services are built on top of this architecture.

## 2.2 Component View






## 2.3 Deployment View

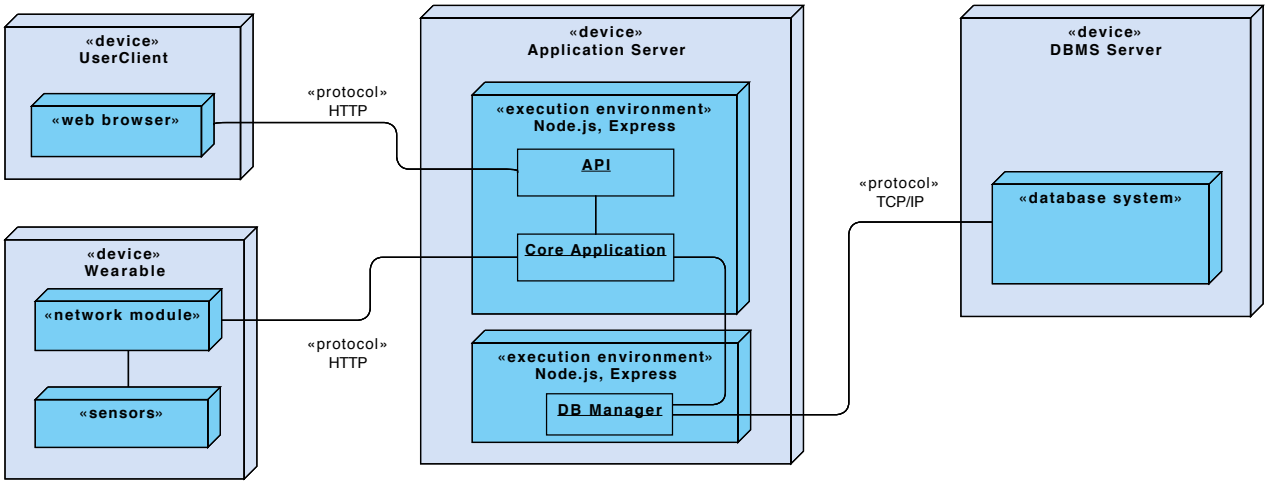
The deployment diagram shows the physical topology of the system.
We can see the three-tier architecture explicitly.

All the users can benefit from Data4Help services by accessing its web application with their own devices. The web application sends requests to the Application Server API. The Application Server handles all the business logic and the communication with the DB and sends the requested data to the client.

In the Application Server, there is a dedicated node to deploy the DB Manager. This way there is less coupling with the Application Server, and it is possible to maintain and update the two modules independently.

The DBMS Server is in charge to store all the data of the application. Deploying it in a dedicated server increases decoupling between system components.

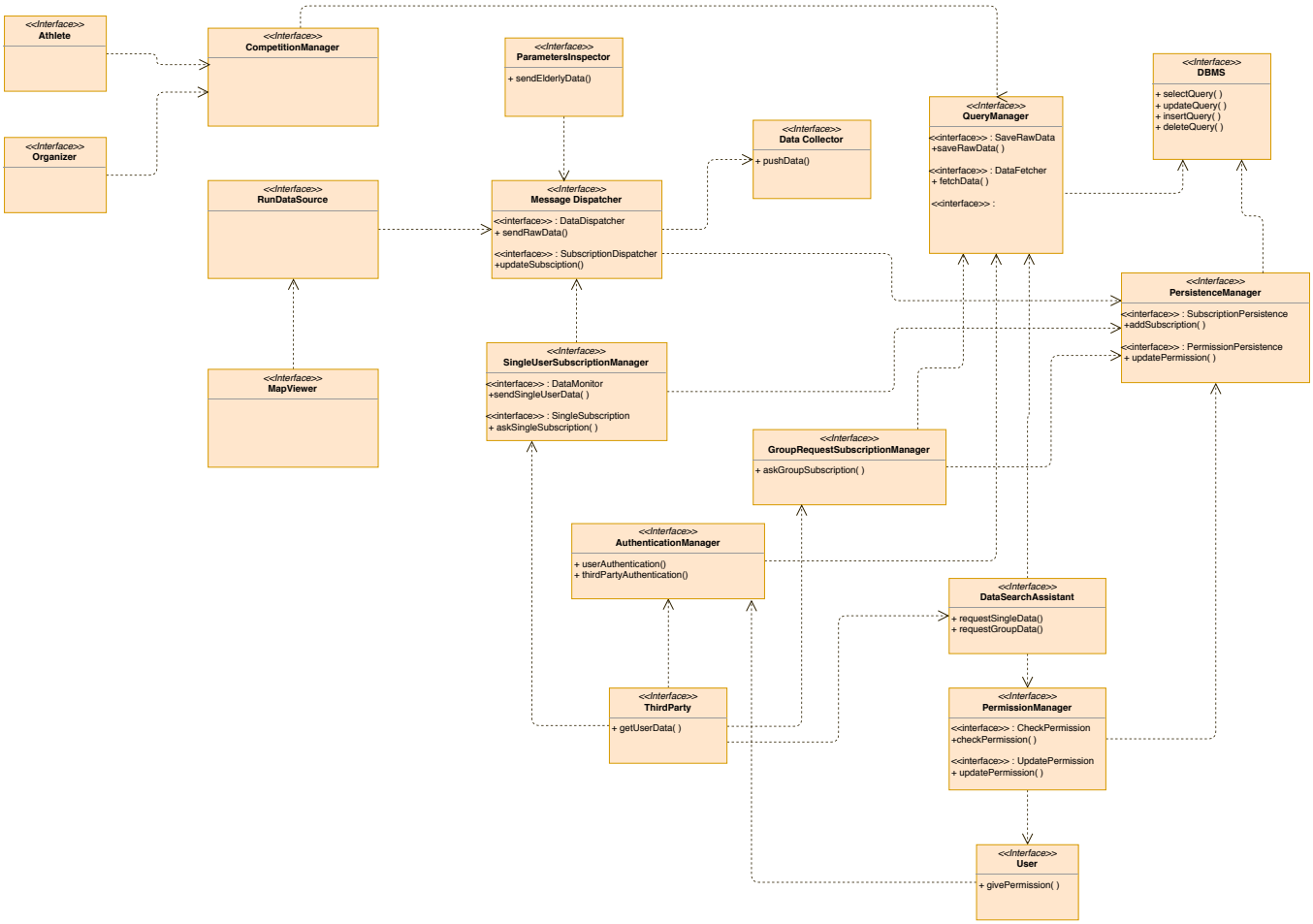
The wearable device runs a simple code that reads data from sensors, and send them to the Application Server through the wearable's network module.




## 2.4 Runtime View

*(You can use sequence diagrams to describe the way components interact to accomplish specific tasks typically related to your use cases)*

## 2.5 Component Interfaces




*(not complete, must be modified by adding new interfaces and dependencies)*

## 2.6 Selected Architectural Styles And Patterns

### *2.6.1 General architecture*

To allow an easy logical distinction of subsystems' responsibilities, for this system we propose an architecture consisting of three tiers.
This kind of architecture well suits a client-server approach mixed with a thin client design, leaving to the client the sole responsibility of presenting the data to the user and acting as an interface to the server functionalities (presentation tier).
Furthermore, the system to be relies heavily on data storage and later retrieval of this data, making a distinct database entity necessary. Encapsulating the DBMS and its related services into a logical persistence tier allows us to decouple the persistence structure from

the rest of the system and to perform easier maintenance and changes, like modifying the DB infrastructure or services, without having the application tier to be informed of the real implementation of the persistence tier but just of its interfaces.
The business logic is handled by the Application tier, which manages every interaction between the system and the external world (users' clients, wearable, external services), and communicates with the Data Persistence tier, which stores all the application data.
Considering the topological deployment of the system, the Application Server exposes an API for the client, and the DB manager inside the application server makes requests to the DBMS.

### *2.6.2 Subsystems architecture*

- **Data Collector**

  The *Data Collector* can be implemented following the event-based paradigm. The *Data Collector* itself acts as the event dispatcher. It receives data from users' wearables, and then broadcasts this data to the DBMS (in order to save the data) and to the *Message Dispatcher*.

  This paradigm allows a high modularity between different components of the systems, significantly increasing decoupling and enhancing flexibility.

  Since all the data has its own timestamp, it's not relevant if the data collector dispatches data out of order.

  The weakness of this paradigm is that the *Data Collector* can be a bottleneck of the system. In order to address this problem, it is possible to set a policy for which, once a defined amount of wearable is subscribed to the data collector, a new instance of *Data Collector* is spawn, and all new wearables will subscribe to this new one.

- **Message Dispatcher**

  The *Message Dispatcher* has the same architecture of the *Data Collector*. Its role is to sends real-time data to the components that need them, specifically the *Parameters Inspector*, *SingleUserSubscriptionManager* and the *RunDataSource*. So, the message dispatcher acts as an event dispatcher, receives data from the *Data Collector* and broadcasts this data to the components mentioned above.

  The *Message Dispatcher* stores internally a list of all the *AutomatedSOS* users, single user subscriptions and the list of active runners. When there are some modifications of these data structures on the DB, the *Message Dispatcher* memory is updated accordingly. The *Message Dispatcher* can also ask for this data to the DB in case of reboot.

  The *Message Dispatcher* shall be very reliable, since the *Parameters Inspector* depends on it. For this reason, it is necessary to have parallel instances of this component in order to increase robustness to failures.

  Regarding scalability, it is possible to use a load balancer which receives data from the

*Data Collector*, and distributes this data to multiple instances of *Message Dispatcher*, accordingly to their requests load. These multiple instances shall have a shared memory, so they all know to which component they have to redirect data.

- Parameters inspector
  The *Parameters Inspector* is an event listener. It waits for users' data from the *Message Dispatcher*, and then check if this data goes beyond the owner's risk thresholds. The *Parameters Inspector* stores internally a list of all the *AutomatedSOS* users' risk thresholds, which needs to be consistent with the data on the DB.
  If some users' data goes beyond risk thresholds, the *Parameters Inspector* begins a rescue procedure, where it contacts the ambulance provider and provides the health and position data of the ill user.
  Due to the criticality of this component, it is necessary to have multiple instances in parallel, in order to increase fault tolerance.
  Also, since R1-NF requires the system-to-be to place a SOS request to the external provider within 5 seconds after detecting data that go beyond risk thresholds, it is possible to use an elastic load balancer in front of the *Parameters Inspector*. It can provision the needed resources based on the amount of data received, allowing the satisfaction of R1-NF.

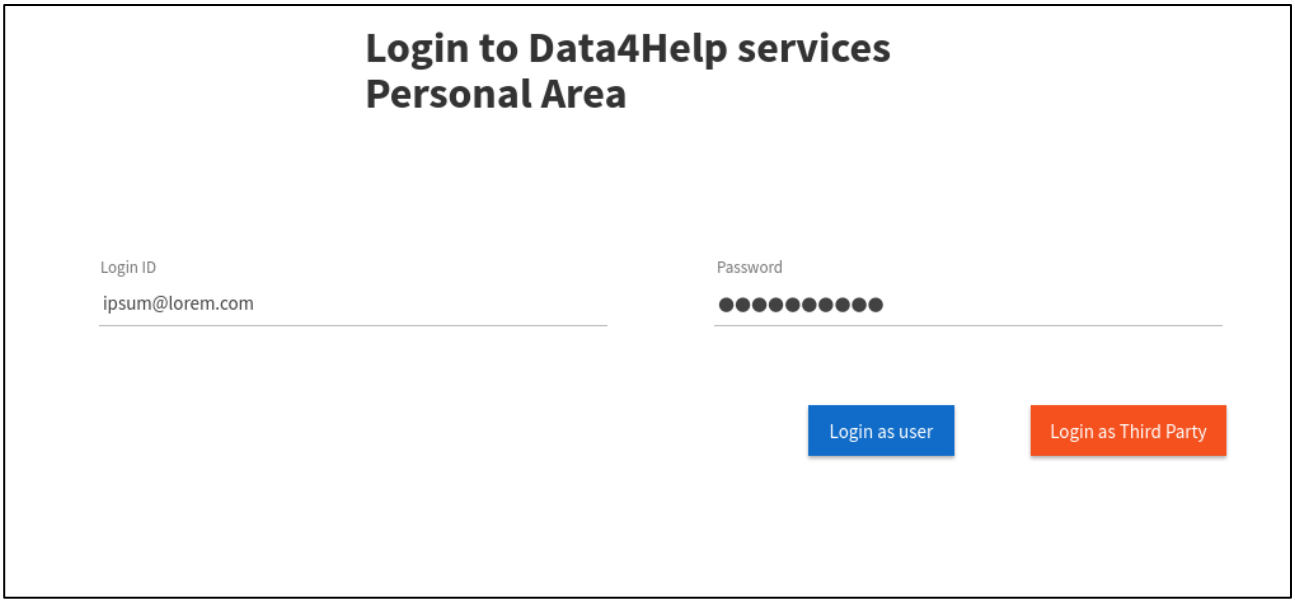
- Single and Group Subscription Manager
  These components are event listeners that wait for requests for new data subscriptions. When they receive these requests, they call addSubscription in order to add the new information to the DB.
  These components are also in charge to send to the third parties the data coming from subscriptions. The *SingleUserSubscriptionManager* receives data from the *Message Dispatcher*. Instead, the *GroupRequestSubscriptionManager* has an internal list of group requests, each one associated with an updateInterval. This component has a timeout for each updateInterval, and when a timeout expires the component fetches data from the DB and sends them to the Third Party.

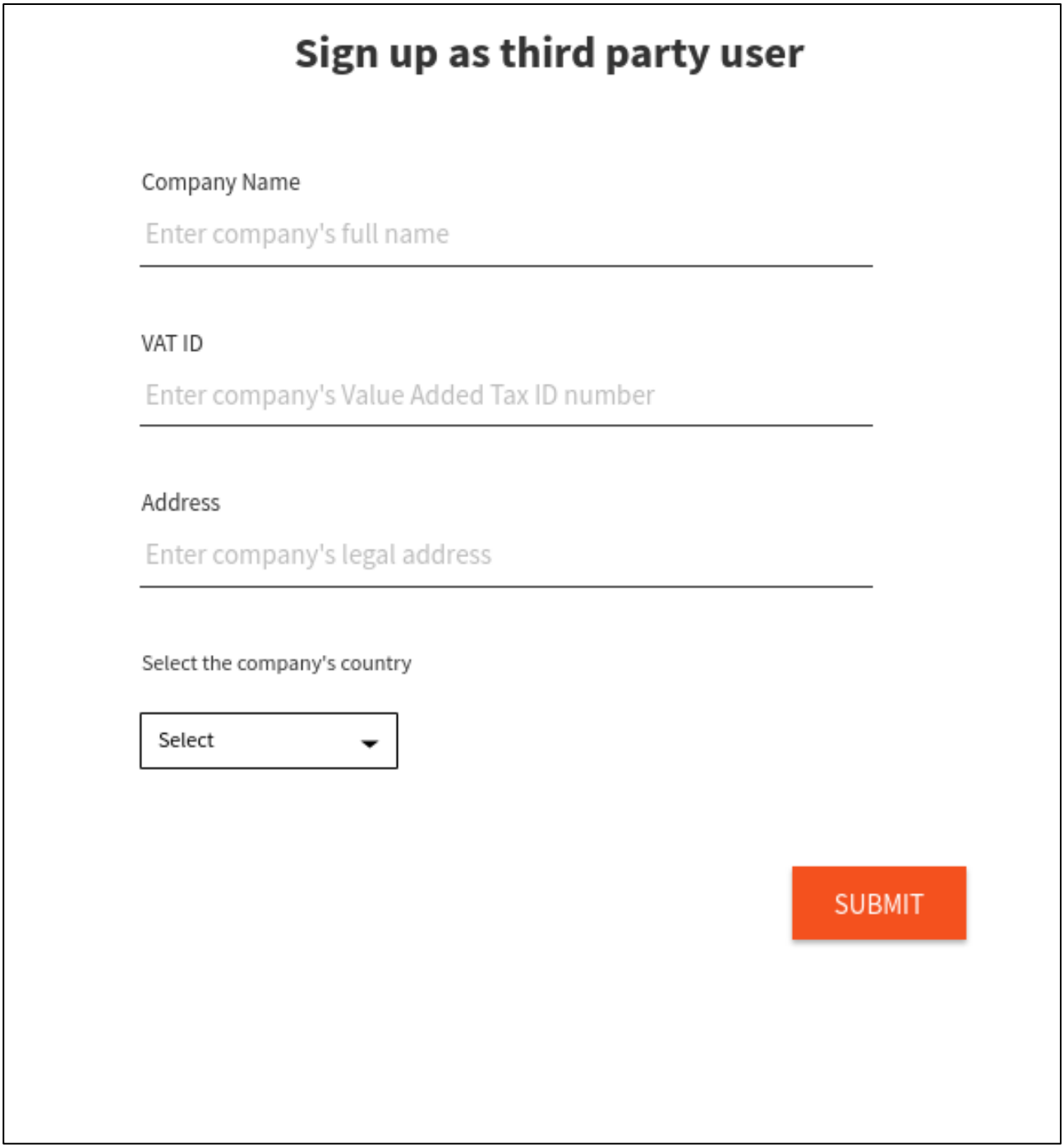
- WebApp Router
  The *WebApp Router* is the API exposed by the application server to the client. It handles all the services offered by the application.
  Since each single service is managed by the router, this component shall be always available. So there shall be multiple instances of the router in parallel, in order to limit failures.

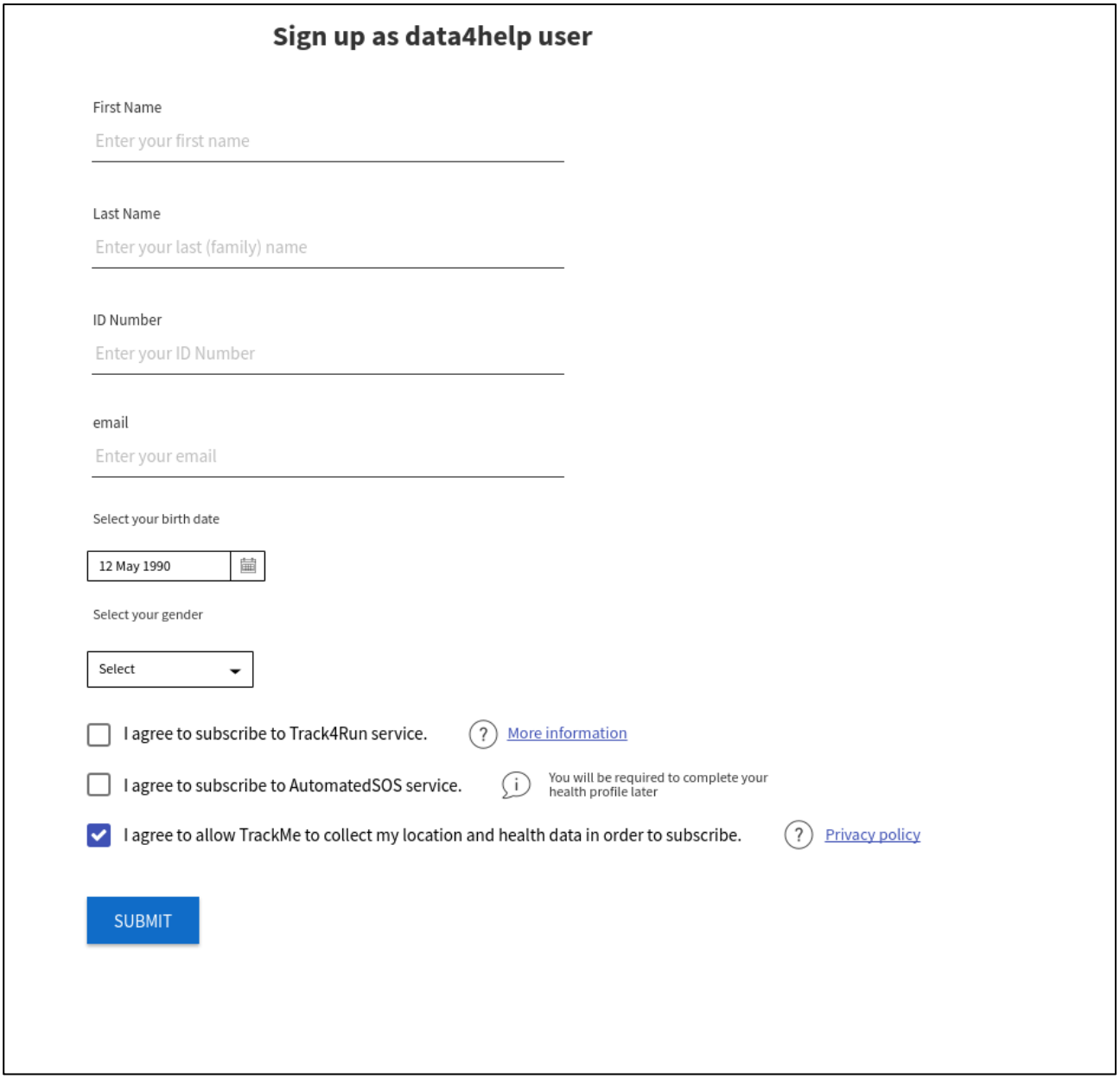
## 2.7 Other Design Decisions

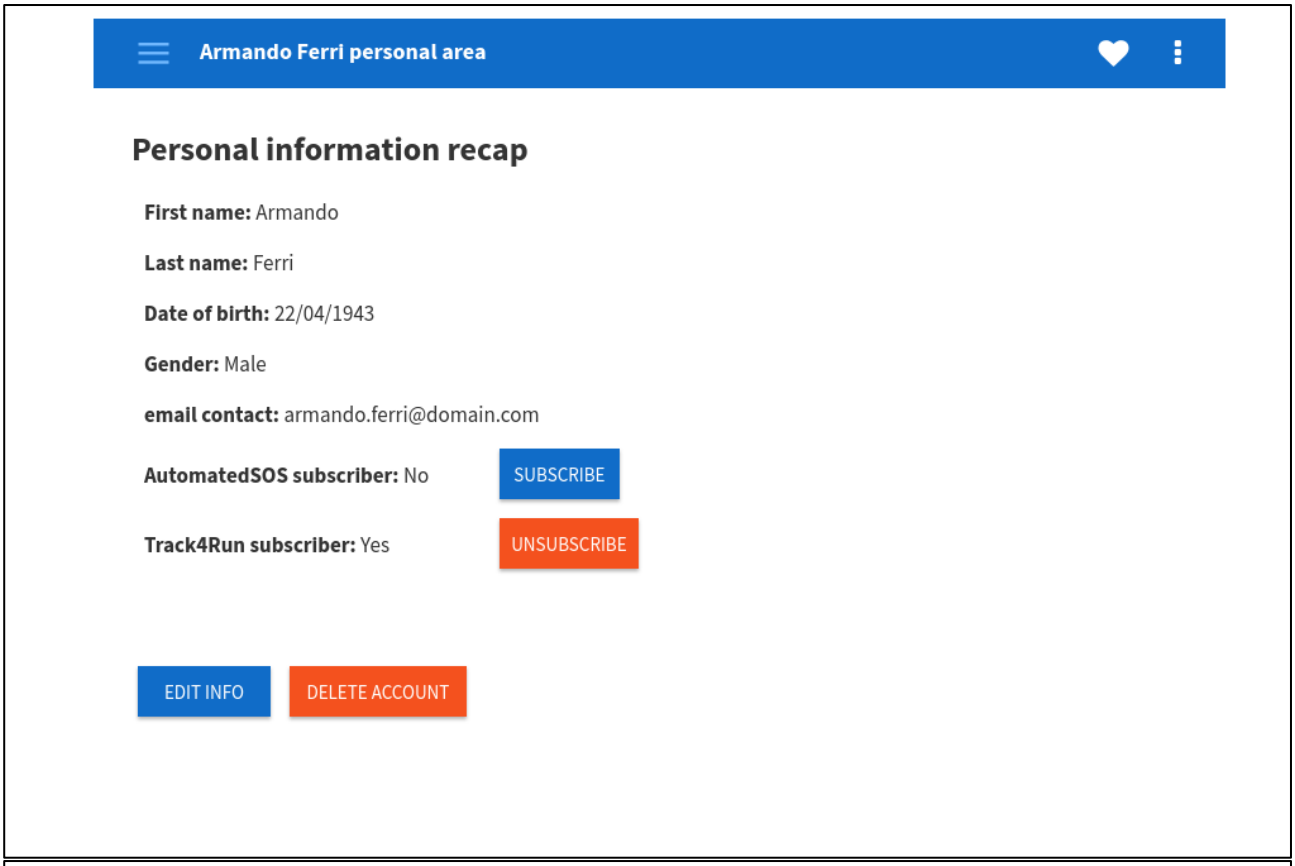
# 3 User Interface Design

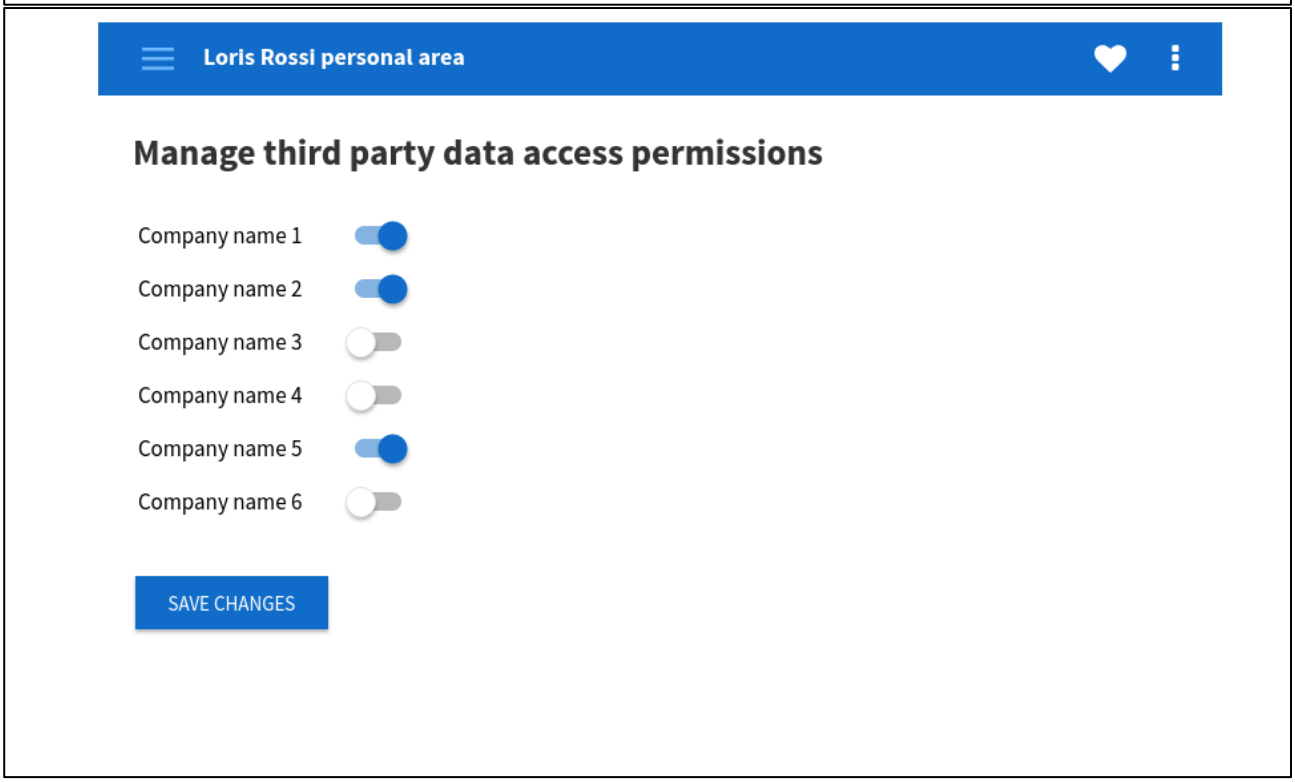
*(Provide an overview on how the user interface(s) of your system will look like; if you have included this part in the RASD, you can simply refer to what you have already done, possibly, providing here some extensions if applicable.)*

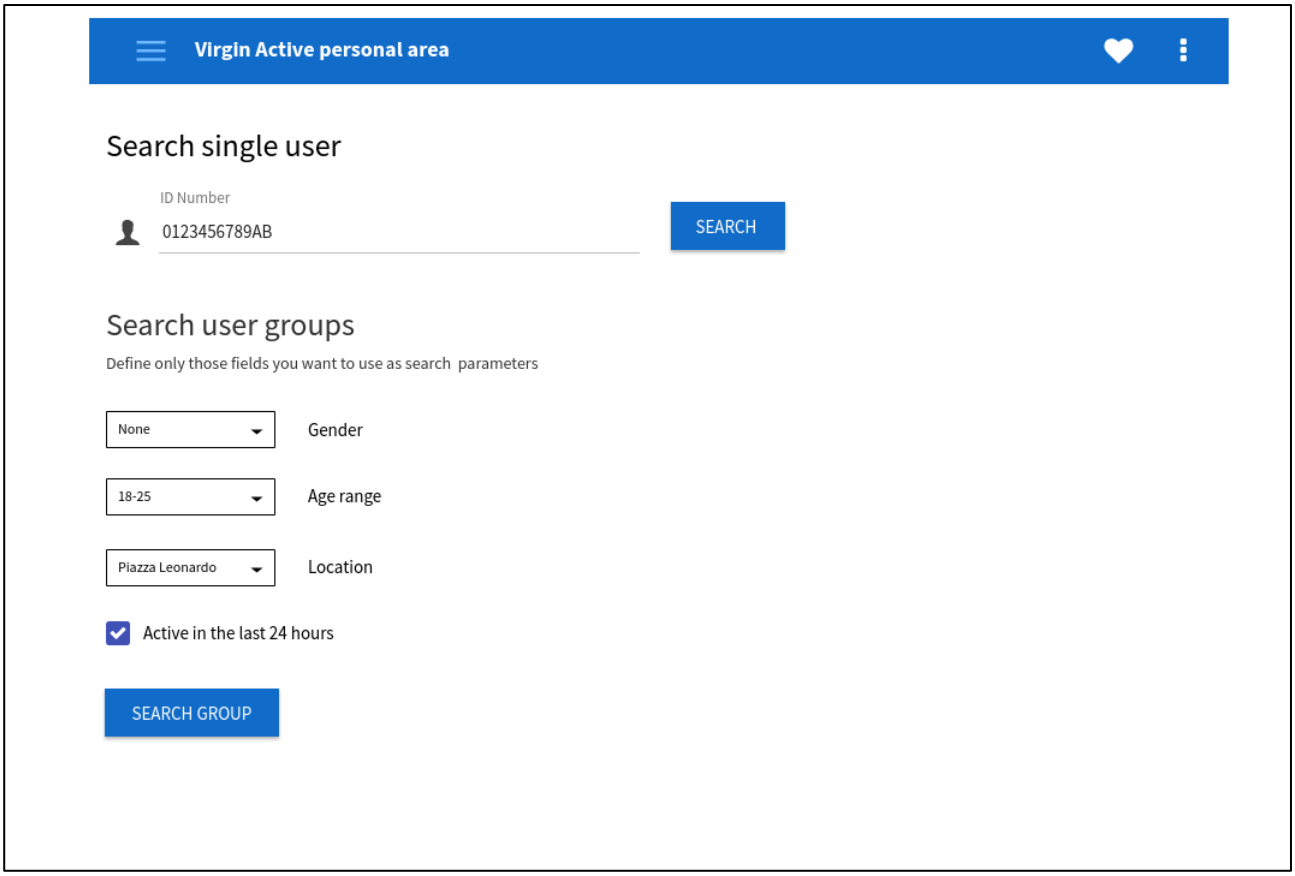

Login to Data4Help services
Personal Area
Login ID
ipsum@lorem.com
Password
Login as user
Login as Third Party


Sign up as third party user
Company Name
Enter company's full name
VAT ID
Enter company's Value Added Tax ID number
Address
Enter company's legal address
Select the company's country
Select
SUBMIT


Sign up as data4help user
First Name
Enter your first name
Last Name
Enter your last (family) name
ID Number
Enter your ID Number
email
Enter your email
Select your birth date
12 May 1990
Select your gender
Select
I agree to subscribe to Track4Run service.
More information
I agree to subscribe to AutomatedSOS service.
You will be required to complete your health profile later
I agree to allow TrackMe to collect my location and health data in order to subscribe.
Privacy policy
SUBMIT


Armando Ferri personal area
Personal information recap
First name: Armando
Last name: Ferri
Date of birth: 22/04/1943
Gender: Male
email contact: armando.ferri@domain.com
AutomatedSOS subscriber: No
SUBSCRIBE
Track4Run subscriber: Yes
UNSUBSCRIBE
EDIT INFO
DELETE ACCOUNT


Loris Rossi personal area
Manage third party data access permissions
Company name 1
Company name 2
Company name 3
Company name 4
Company name 5
Company name 6
SAVE CHANGES


Virgin Active personal area
Search single user
ID Number
0123456789AB
SEARCH
Search user groups
Define only those fields you want to use as search parameters
None
Gender
18-25
Age range
Piazza Leonardo
Location
Active in the last 24 hours
SEARCH GROUP

# 4 Requirements Traceability

*(Explain how the requirements you have defined in the RASD map to the design elements that you have defined in this document.)*

**G1:** The user can be recognized by providing a form of identification.

- *User* **[R1, R3, R4]**
- *AuthenticationManager* **[R1, R2]**
- *QueryManager* **[R2]**

[R1]: The system shall allow registration of individuals through an email and a password.
[R2]: The system shall guarantee the unicity of email addresses.
[R3]: The system shall ask users to provide their personal data (birthdate, gender, residency address, ID number).
[R4]: The system shall ask the user to agree to a policy that specifies that, by registering, users agree that TrackMe acquires their data.

**G2**: Allow third parties to monitor data about location and health status of individuals.

- *DataCollector* **[R5]**
- *MessageDispatcher* **[R5, R7]**
- *QueryManager* **[R5, R6, R30]**
- *Wearable* **[R5]**
- *ThirdParty* **[R6, R30]**
- *AuthenticationManager* **[R6, R30]**
- *GroupRequestSubscriptionManager* **[R7]**
- *SingleUserSubscriptionManager* **[R7]**
- *PersistenceManager* **[R7]**

[R5]: The system shall store past position and health data of every single user.
[R6]: The system shall support the registration of third parties and provide them with a unique identifier (ID).
[R30]: The system shall support the login of third parties with their ID and password.
[R7]: Third parties shall be allowed to subscribe to new data and the system shall send data as soon as they are produced.

**G3**: Allow third parties to access data relative to specific users.

- *ThirdParty* **[R8]**
- *DataSearchAssistant* **[R8, R9]**
- *QueryManager* **[R8, R9]**

- *PermissionManager* **[R9]**
- *User* **[R9]**

[R30]
[R7]
[R8]: Third parties shall be able to request and receive a specific user's data by providing user's ID number.
[R9]: Upon every data collection request, the system shall check if the permission to access that data has already been granted by the user, otherwise it shall ask them

**G4**: Allow third parties to access anonymized data of groups of users.

- *ThirdParty* **[R10]**
- *DataSearchAssistant* **[R10]**
- *QueryManager* **[R10, R11]**
- *GroupRequestSubscriptionManager* **[R12]**
- *PersistenceManager* **[R12]**
- *MessageDispatcher* **[R12]**

[R30]
[R7]
[R10]: The system shall allow third parties to search for the desired group of individuals.
[R11]: The system shall accept a group data request only if the data can be properly anonymized. Anonymization is considered proper if the number of people involved in the request is greater than 1000.
[R12]: The system shall allow third parties to subscribe to periodic updates relative to a certain group of individuals provided that the data contained in each update can be properly anonymized.

**G5**: Allow third parties to offer a personalized and non-intrusive SOS service to elderly people so that an ambulance arrives to the location of the customer in case of emergency.

- *User* **[R31, R13]**
- *AuthenticationManager* **[R31, R13]**
- *ParametersInspector* **[R14, R15, R16, R17]**
- *MessageDispatcher* **[R14]**

[R31] The system shall support the login of user with their email and password.
[R13]: The system, in order to allow Data4Help users to subscribe to AutomatedSOS, shall ask them to input their risk thresholds.

[R14]: Frequently enough, health parameters of each user are monitored by the system and compared against their threshold to detect risk situations.
[R15]: If a user's parameters get below risk thresholds the system shall contact the ambulance provider and require the dispatch of an ambulance.
[R16]: After an emergency request has been placed the system shall send data about the user and keep sending them regularly to the ambulance provider.
[R17]: The system shall allow the ambulance provider to notify when the user has been picked up, in order to stop data transmission.

**G6**: Allow athletes to enroll in a run.

- *User* **[R18]**
- *AuthenticationManager* **[R18]**
- *Athlete* **[R19, R20]**
- *CompetitionManager* **[R19, R20, R21]**
- *QueryManager* **[R19, R21]**

[R18]: The system shall allow athletes to register to the system.
[R31]
[R19]: The system shall allow athletes to check a list of available runs.
[R20]: The system shall provide the ability to enroll to the desired run only to registered athletes.
[R21]: The system shall allow enrolling only if the user has already agreed to share publicly his location for the duration of the run.

**G7**: Allow organizers to manage runs.

- *Organizer* **[R22, R23, R24, R25, R26]**
- *AuthenticationManager* **[R22]**
- *CompetitionManager* **[R23, R24, R25, R26]**
- *QueryManager* **[R22, R23, R24, R25, R26]**

[R30]
[R22] The system shall allow organizers to register to Track4Run platform.
[R23] The system shall allow organizers to create and delete races.
[R24] The system shall allow organizers to add a path for the run.
[R25] The system shall allow organizers to check a participants list.
[R26] The system shall allow organizers to add or remove participants before the start of the race.

**G8**: Allow spectators to see on a map the position of all runners during the run.

- *Spectator* **[R27, R28]**
- *QueryManager* **[R27]**
- *MapViewer* **[R28, R29]**
- *RunDataSource* **[R28, R29]**

[R27] The system shall provide a public list of live runs and allow the spectator to select one of them.
[R28] The system shall allow the spectator to see a map of the desired run, with live participants' position.
[R29] The system shall update the positions of the runners on the map as soon as new data is received.

# 5 Implementation, Integration and Test Plan

*(Identify here the order in which you plan to implement the subcomponents of your system and the order in which you plan to integrate such subcomponents and test the integration.)*

## 5.1 Implementation Plan

The implementation of this system shall be conducted in a bottom-up way by first implementing the low-level components which are used by other components and then going upwards in the system levels. However, the low-level components inside a higher-level component, for instance *DataCollector* and *MessageDispatcher* inside *LiveDataSource*, will be implemented in parallel and will be tested concurrently following incremental integration and testing approach (this will be better explained in the following paragraphs).

Following this approach, the components order of implementation shall be:

1. *DBServices (DBMS, QueryManager, PersistenceManager)*
2. *LiveDataSource (DataCollector, MessageDispatcher), Wearable*
3. *RequestManager (DataSearchAssistant, GroupRequestSubscriptionManager, SingleUserSubscriptionManager, PermissionManager)*
4. *WebApp (User and ThirdParty), WebAppRouter, AuthenticationManager*
5. *ParametersInspector*
6. *Track4RunCore, Track4Run WebApp (MapViewer, Athlete, Organizer, Spectator)*

... point 2 – *RunDataSource* of Track4Run can also be implemented earlier in this stage since it requires a slight modification in *LiveDataSource* where the *MessageDispatcher* shall push runners' data into this component.

... point3 – since *RequestManager*'s subcomponents don't interact with each other, it's not necessary to implement them in parallel.
For instance, one choice could be to implement first *SingleUserSubscriptionManager* and then test and integrate it with *DBServices* and *LiveDataSource* before starting implementing other components. Only *DataSearchAssistant* requires some functionalities of another component of *RequestManager*, *PermissionManager* and for this reason it would be better to integrate them incrementally

# 6 Effort Spent

*(In this section you will include information about the number of hours each group member has worked for this document.)*

## 6.1 Piccinotti Diego

| Description of the task | Hours |
|---|---|
| Purpose, Scope, Definition | 1 |
| High-level Components and their Interaction | 3 |
| Component View | 11.5 |
| Deployment View | |
| Runtime View - Sequence Diagrams | |
| Selected Architectural Styles and Patterns | 1 |
| Component Interfaces | |
| Algorithm Design | |
| User Interface Design | 3 |
| Requirements Traceability | |
| Implementation, Integration and Test Plan | |

## 6.2 Pietroni Umberto

| Description of the task | Hours |
|---|---|
| Purpose, Scope, Definition | 1 |
| High-level Components and their Interaction | 4 |
| Component View | 5 |
| Deployment View | |
| Runtime View - Sequence Diagrams | |
| Selected Architectural Styles and Patterns | |
| Component Interfaces | 2 |
| Algorithm Design | |
| User Interface Design | |
| Requirements Traceability | 2 |
| Implementation, Integration and Test Plan | 1 |

## 6.3 Rossi Loris

| Description of the task | Hours |
|---|---|
| Purpose, Scope, Definition | 1 |
| High-level Components and their Interaction | 3 |
| Component View | 4.5 |
| Deployment View | 2 |
| Runtime View - Sequence Diagrams | |
| Selected Architectural Styles and Patterns | 3 |
| Component Interfaces | 1 |
| Algorithm Design | |
| User Interface Design | |
| Requirements Traceability | |
| Implementation, Integration and Test Plan | |

# 7 References