

DD

Design Document

Data4Help, AutomatedSOS and Track4Run

Irene Nizzoli, Isabella Piacentini, Elio Salvini

10483798 10508831 10490058

POLITECNICO DI MILANO

Sommario

[1 Introduction 3](#_Toc531811971)

[1.1 Purpose 3](#_Toc531811972)

[1.2 Scope 3](#_Toc531811973)

[1.3 Acronyms, Abbreviations 3](#_Toc531811974)

[1.3.1 Acronyms 3](#_Toc531811975)

[1.3.2 Abbreviations 4](#_Toc531811976)

[1.4 Document Structure 4](#_Toc531811977)

[2. Architectural Design 5](#_Toc531811978)

[2.1 Overview 5](#_Toc531811979)

[2.2 Component View 6](#_Toc531811980)

[2.3 Deployment view 9](#_Toc531811981)

[2.5 Component interfaces 10](#_Toc531811982)

[2.6 Selected architectural styles and patterns 11](#_Toc531811983)

[2.6.1 Architectural styles 11](#_Toc531811984)

[2.6.2 Architectural design patterns 11](#_Toc531811985)

[2.6.3 Implementative design pattern 12](#_Toc531811986)

# Introduction

## Purpose

The purpose of this document is to give a more detailed description of the architecture of Data4Help system. It will include the illustration of specific components and design choices that will guide the developers during implementation, integration and testing.

Overall this document outlines these elements:

* The high-level architecture
* The main components and their respective interfaces
* The runtime behaviour
* The design patterns
* The algorithm design of the most critical parts of the application
* Implementation plan
* Integration plan
* Testing plan

## Scope

The aim of TrackMe is to provide a service to either companies in need of data for business researches or individuals for more personal reasons. The main functions of Data4Help are managing requests from different users and saving and protecting a great quantity of data. A registration will be needed to provide clients a personalized experience, both by giving them the results of their requests and by showing monitored user their private health status data. The project is extended by AutomatedSOS that monitors the data of subscribed users and contacts medical services in case of need. The target of this system are elderly people who lives alone or are simply worried about their health conditions. This will need a 24/7 reliability of the application. Finally, Track4Run allows run organizers to create new races, tracks runners and show their position on the map to all possible spectators.

## 1.3 Acronyms, Abbreviations

### 1.3.1 Acronyms

* RASD: Requirement Analysis and Specification Document
* API: Application Programming Interface
* GPS: Global Positioning System
* DAD: Data Acquisition Device
* CF: “Codice Fiscale”
* SSN: Social Security Number
* DD: Design Document
* MVC: Model View Controller
* GUI: Graphical User Interface
* DB: Database
* DBMS: Database Management System

### 1.3.2 Abbreviations

* [Gn]: nth goal
* [Rn]: nth functional requirement

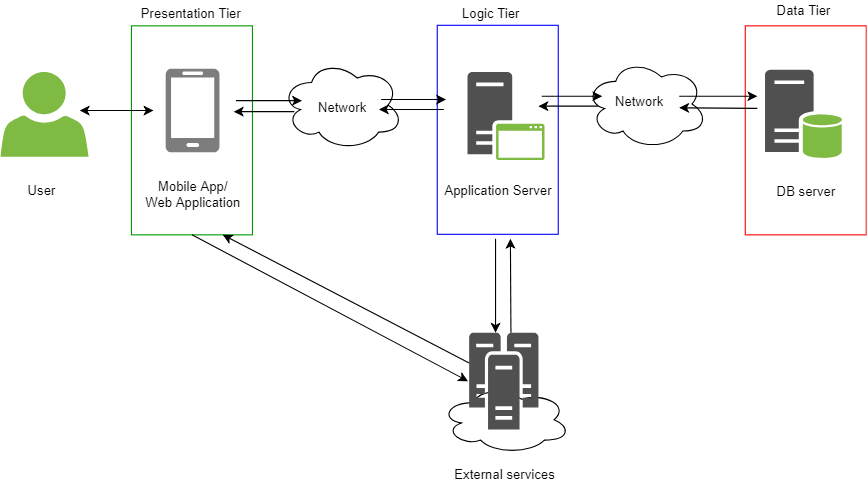
### 1.4 Document Structure

1. **Introduction:** this chapter contains the purpose and the scope of the design document. There’s also a list of the acronyms and the abbreviation that will be used in the document in order to make it more comprehensible.
2. **Architectural Design:** this sectiongives a general idea of the architecture including the three most important views: component, deployment and runtime. The interaction of the component interfaces and some architectural styles and patterns are also contained here.
3. **User Interface Design:** this chapter presents a reference to the mock-ups previously presented in the RASD document.
4. **Requirements traceability:** clarifies how the requirements that have been defined in the RASD map to the design elements that are defined in this document.
5. **Implementation, integration and test plan:** reveal the order in which it is intended to implement the subcomponents of the system and the order in which it is planned to integrate such subcomponents and test the integration.
6. **Effort spent:** shows the number of hours each member of the group spent for every chapter of the document.
7. **References:** presents the external documents used in the construction of the DD document.

# 2. Architectural Design

## 2.1 Overview

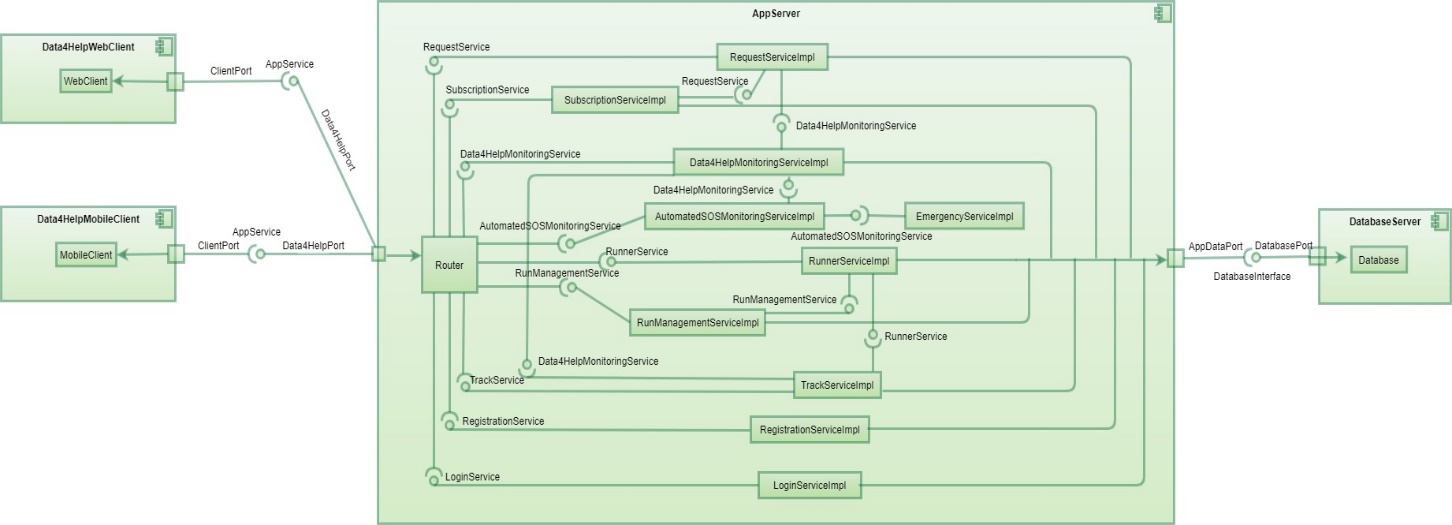
There are three separated layers in the flow in which the process of the application architecture design is executed:



* *Presentation tier*: This is the top-most level of the application. It consists in a user interface (view) and communicates with the application services and the external services. The connection with the external services is useful because, for example, the organizer (for Track4Run) can be directly provided with the map by the mobile app/web application.
* *Logic tier*: This layer controls the functionality of the application. The logic of the application is stored on the server side.
* *Data tier*: The data tier comprises of the database/data storage system and data access layer. Here the information is stored persistently and can be retrieved at any time.

## 2.2 Component View

In the following diagram is represented the interface structure of the system both inside the WebClient, the MobileClient, where only a small part of the application is present, and the AppServer. Using a browser to use TrackMe application the client has access to all the features of the mobile application besides AutomatedSOS functionalities. The external services are not shown because their implementation it is not relevant for our application.



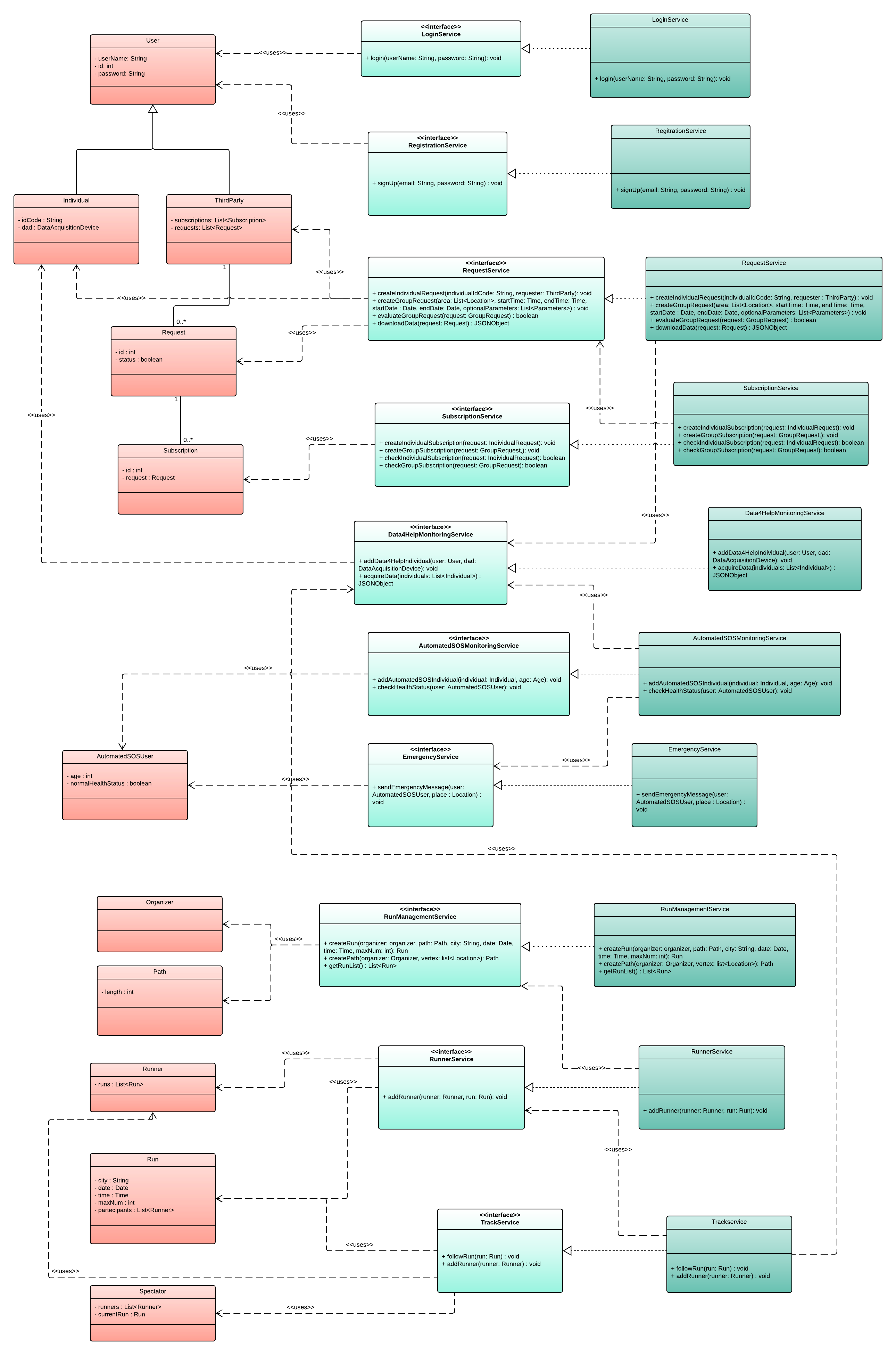
The interfaces portrayed in the diagram above, with their respective implementation identified by the suffix Impl, are the following:

* RegistrationService: allows users to create new personal accounts
* LoginService: deals with the authentication of the users
* Data4HelpMonitoringService: manages the data received from the devices by storing and recovering them from the Database when needed
* RequestService: provides functionalities regarding the creation of new requests or the approval phase
* SubscriptionRequest: responsible for the managing of subscriptions
* AutomatedSOSMonitoringService: provides the mechanisms needed to monitor AutomatedSOSUsers and to continuously check their health status conditions
* EmergencyService: manages the connection with external medical service when needed and awaits confirmations of message received
* RunManagementService: allows organizers to add a new run inside the application, listing place, date, time and other details of the race
* RunnerService: provides functionalities about the registration of a new runner to a specific run among the ones listed inside the database
* TrackService: deals with the mechanism that shows participants of a specific run on a map inside the application to the spectator who make a request for it

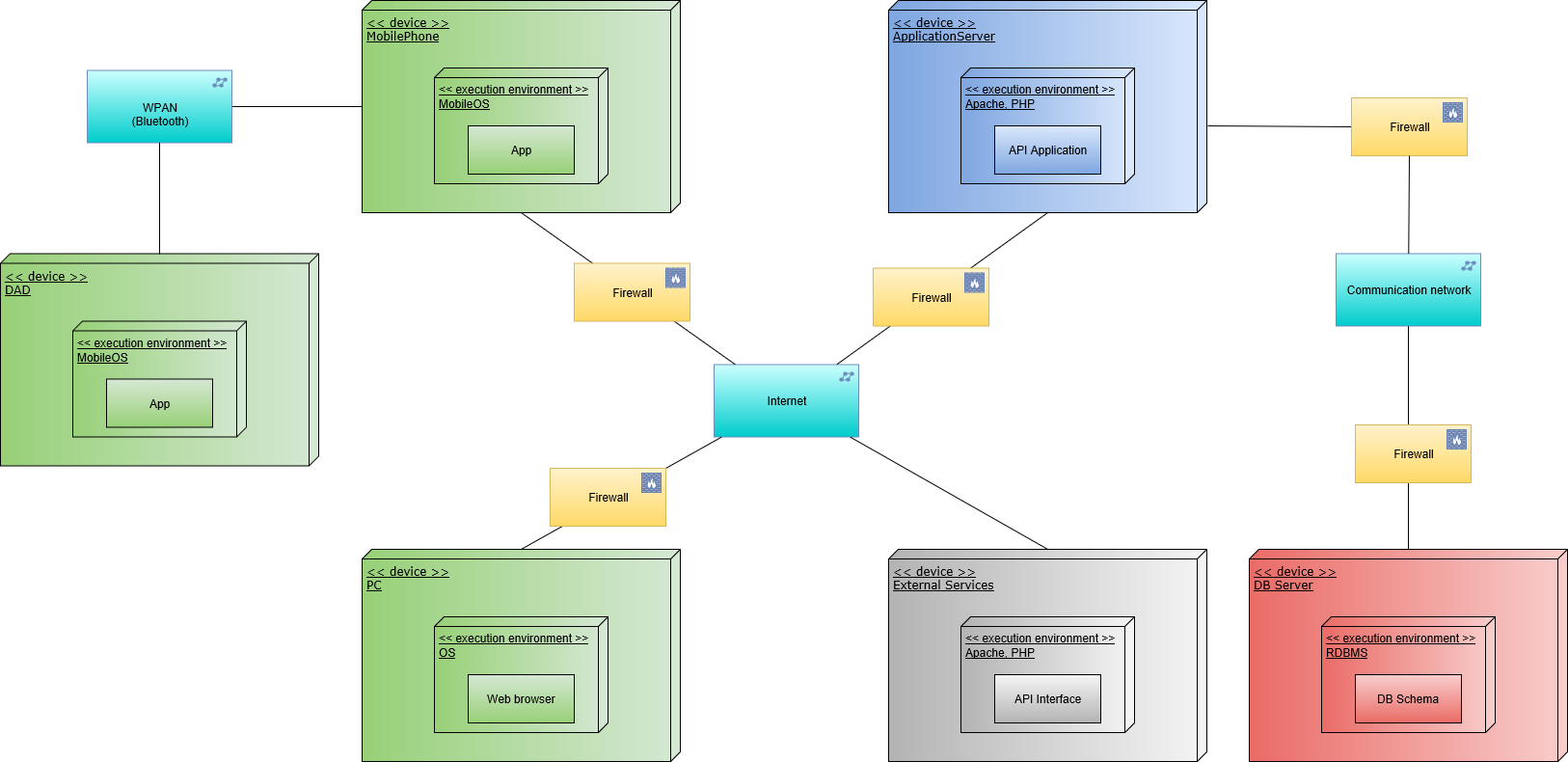
The logic of the application is composed by the interconnection of the services listed above and it works using the connection between each other to exploit different request given by the client. The presence of the router helps to distribute different task more easily inside the appServer and allows a better growth in the future of the application because adding new interfaces will be simpler.

The two diagrams below help giving a more detailed view of the UML constructed before and showing more features of the connection among services.





## 2.3 Deployment view

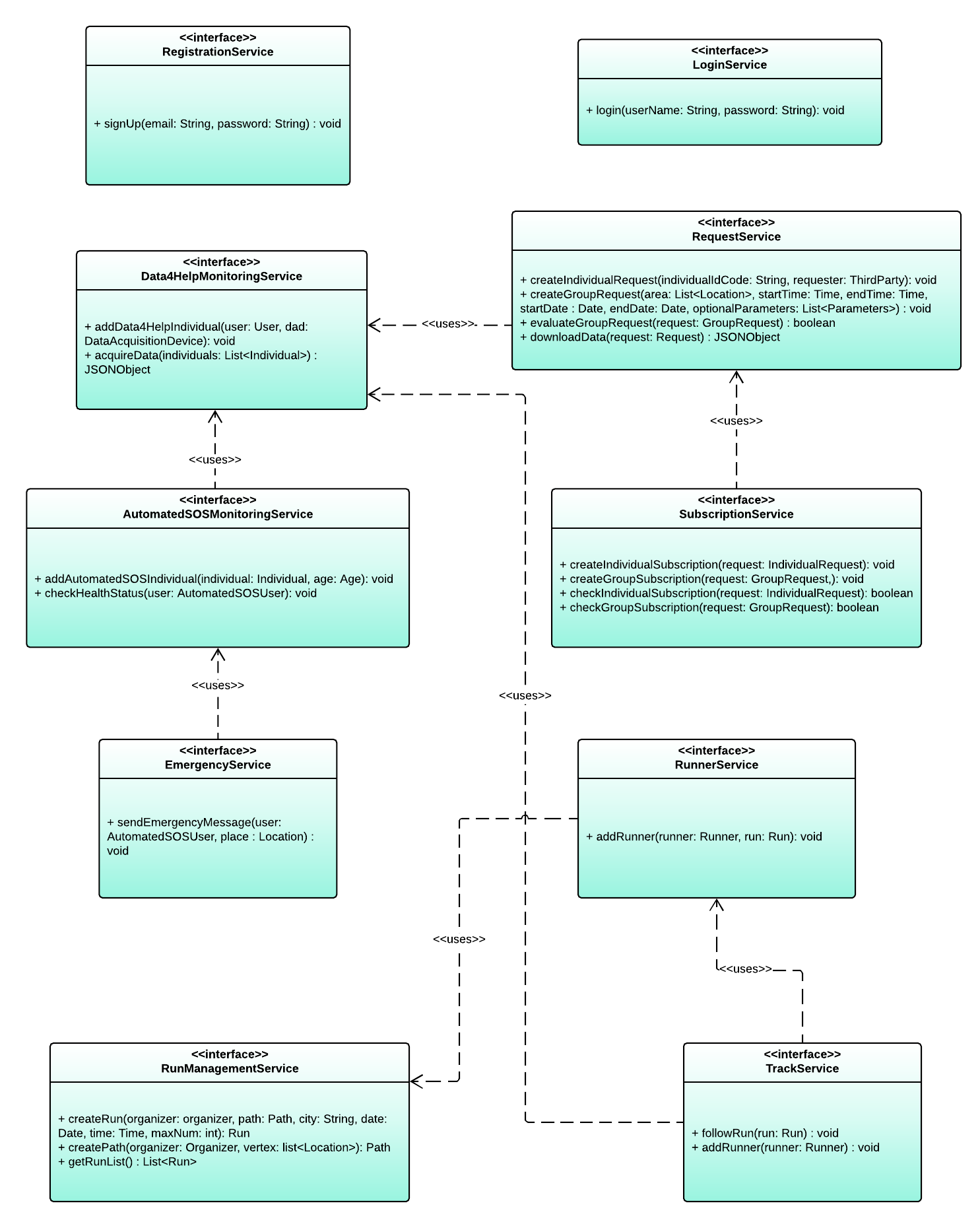


This deployment diagram shows the architecture of the system from a physical view point. Also, the distribution of software among the different hardware nodes is presented. The main nodes involved in the system are:

* *DAD*, this device acquires health status data from user and send them to the user’s mobile phone using Bluetooth channel.
* *Client nodes*:
  + *Mobile phone*, users can access to Data4Help services through a mobile application that communicates with TrackMe’s server or directly with other external services (such as the street map service for Track4Run). This node is responsible of the forwarding of acquired DAD’s data to Data4help server.
  + *PC* or other devices able to access to web services, Data4Help services are also available through its website, accessible via web browser. Even this node, as the mobile phone one, can access to external services.
* *Application server*, the application logic belongs to this node. The application server’s communication with clients is based on the client-server pattern. This node also communicates with the system’s DB that stores all users’ data. The application server exploits external services API to provide a complete service to its clients.
* *DB server*, this node’s aim is to store all users’ data, from general account information to health status and location data. This node is directly accessed only by the application server.
* *External services*, client nodes and application server take advantage of this services to provide all AutomatedSOS and Track4Run functionalities.

## 2.5 Component interfaces

The diagram below portrays the relations among the interfaces presented before in paragraph 2.2. This will help give a clearer understanding of the association between different components and their interfaces.



## 2.6 Selected architectural styles and patterns

### 2.6.1 Architectural styles

Data4Help, AutomatedSOS and Track4Run systems are based on the following architectural styles:

* *Client-server*, Data4Help system is based on the client-server architectural style to manage its communication with users. Clients send requests to server who sends back responses providing the wanted service.

This architectural style is the most suitable for our context where it’s necessary a mediator who manages the exchange of data between the different kinds of users. Server’s role isn’t only limited to data transfer, but it also processes data and stores them in a DB. In this way clients can be lighted of much effort, so that they’re focused only on presentation tasks (thin client). Also, this style grants a better level of security among clients (anonymity of individual must be guaranteed).

* *Three-tier*, this design choice allows to split system’s tasks in three groups (tiers), and to assign them to different devices. One of the main advantages is that each tier’s tasks is executed by the proper component (for example, the storage of data is assigned to a DB).

### 2.6.2 Architectural design patterns

This system is designed taking advantage of the following design pattern (from an architectural view point):

* *Model View Controller (MVC)*, one of the most used design patterns, it divides the application components in three groups. Model’s components realise the service itself, directly operating with the stored data. View’s components are responsible for the presentation of the results produced by the Model. From a more general view point, the View is the first interface of dialog between the user and the application system; it takes user’s commands, it send them to the application server and it also shows the responses to the user. The Controller’s role is to mediate the communication between Model and View, and to do this it exploits the service interfaces defined in the previous paragraphs. View’s components are client side, while Model and Controller are server side. Direct communication between Model and View can happen for some simple operations, like the update’s ones showed in the observer pattern.

The use of this pattern makes faster the development process, and it also allows to provide multiple Views, since different of them are needed for the mobile application and the web version.

* *Façade*, this pattern provides clients with a simpler interface to access to Data4Help’s services. A client may have the necessity to exploit different functionalities of different service interfaces to accomplish a user request; this pattern simplifies client’s interaction with the whole system providing a unique interface (*router*) which drives each client’s request to the correct interfaces.

### 2.6.3 Implementative design pattern

In this part some useful design pattern for the implementation are listed:

* *Singleton*, in order to guarantee coherence of the area shown to spectators of Track4Run, the class representing the map in the model should be singleton.
* *Observer*, the constant monitoring of individuals could be performed using the design pattern observer. This allows the observer class to know every change of the observable object automatically.