DD

Design Document

Data4Help, AutomatedSOS and Track4Run

Irene Nizzoli, Isabella Piacentini, Elio Salvini
10483798 10508831 10490058

POLITECNICO DI MILANO

08/12/2018

Sommario

1 Introduction 4

1.1. Purpose 4

1.2 Scope 4

1.3 Acronyms, Abbreviations 4

1.3.1 Acronyms 4

1.3.2 Abbreviations 5

1.4 Document Structure 5

2. Architectural Design 6

2.1 Overview 6

2.2 Component View 7

2.3 Deployment view 10

2.4 Runtime View 11

2.4.1 GroupRequestCreation 11

2.4.2 MedicalHelpRequest 12

2.4.3 FollowARunner 13

2.4.4 RunnerSignIn 14

2.5 Component interfaces 15

2.6 Selected architectural styles and patterns 16

2.6.1 Architectural styles 16

2.6.2 Architectural design patterns 16

2.6.3 Implementative design pattern 17

2.7 Other design decisions 17

3 User interface design 17

4 Requirements traceability 18

6 Implementation, integration and test plan 19

6.1 Implementation plan 19

6.2 Integration and testing 20

6.2.1 Entry criteria 20

6.2.2 Elements to be integrated 20

6.2.3 Integration Testing Strategy 22

7 Effort Spent 23

7.1 23

7.2 23
7.3 23
8 References 24

1 Introduction

1.1. 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

1.2 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
- DAG: Direct Acyclic Graph

1.3.2 Abbreviations

- [Gn]: n^{th} goal
- [Rn]: n^{th} functional requirement

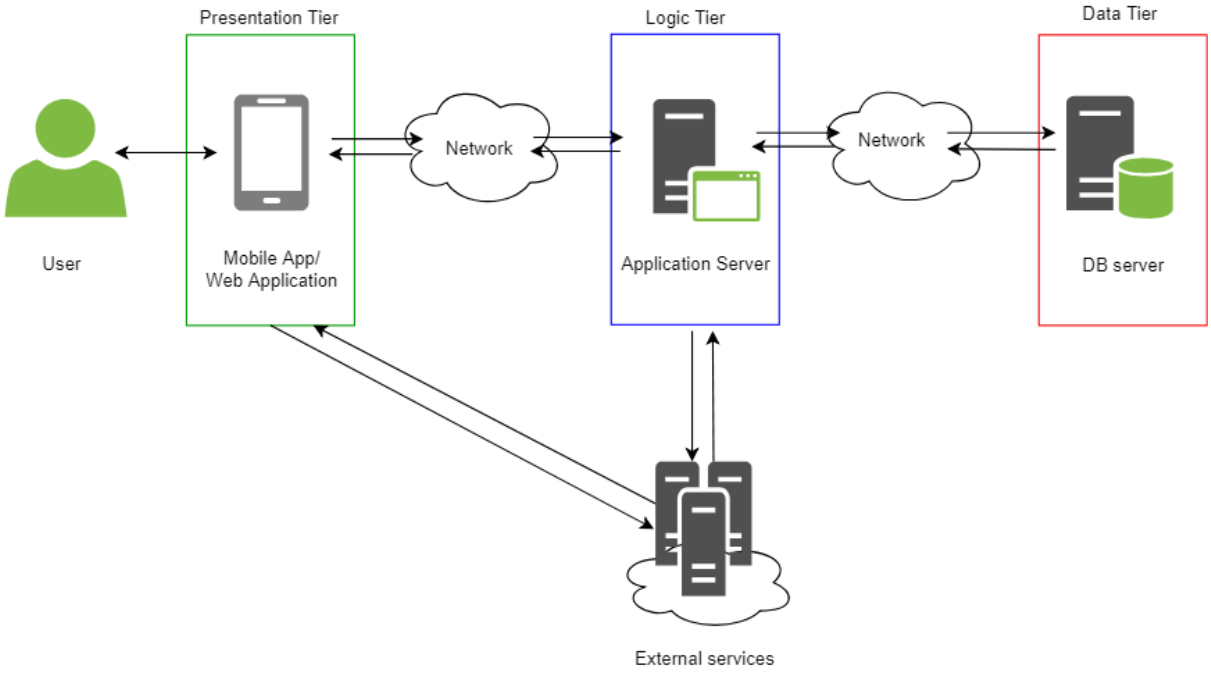
1.4 Document Structure

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 section gives 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:



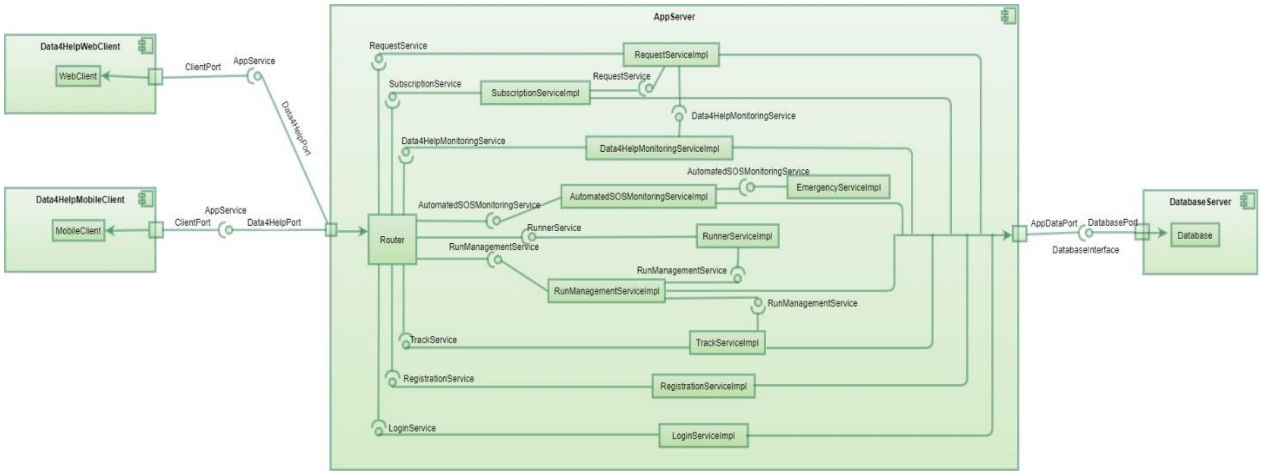
- *Presentation tier*: This is the top-most level of the application. It consists in a user interface (view) and communicates with the application services and the external services. The connection with the external services is useful because, for example, the organizer (for Track4Run) can be directly 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.

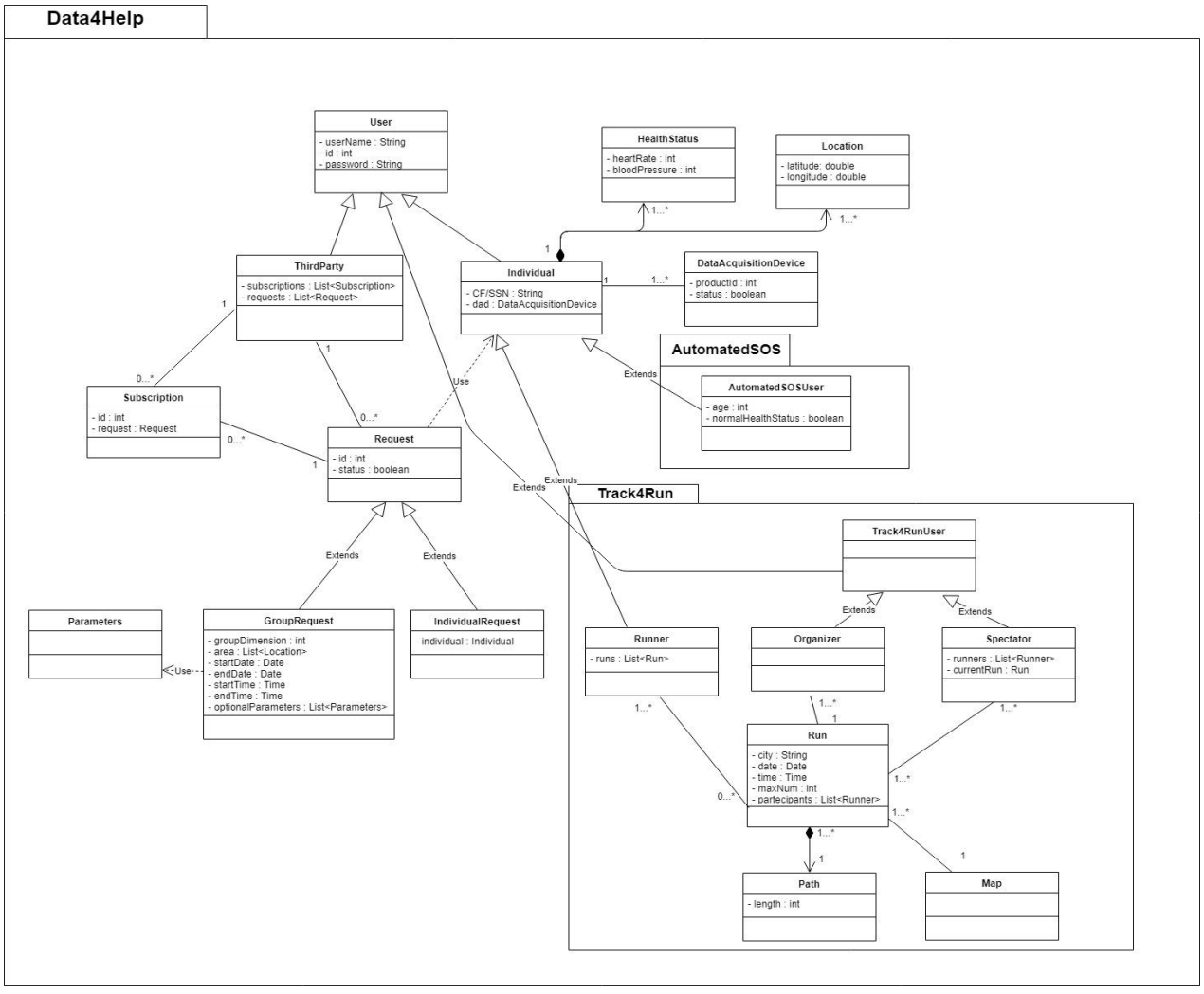


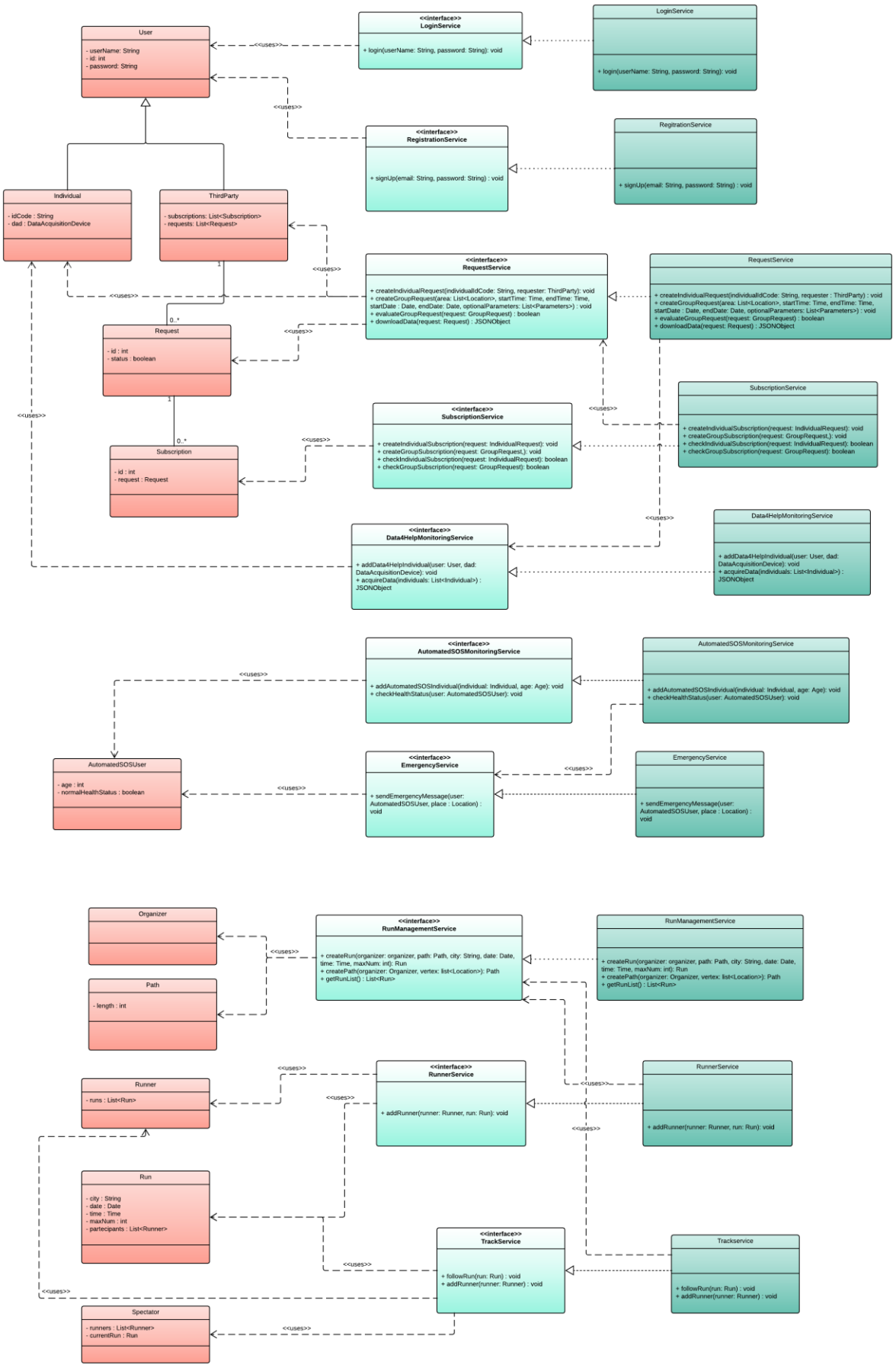
The interfaces portrayed in the diagram above, with their respective implementation identified by the suffix Impl, are the following:

- 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
- SubscriptionService: responsible for the managing of subscriptions (to download data it exploits the functionalities of the RequestService)
- AutomatedSOSMonitoringService: provides the mechanisms needed to monitor AutomatedSOSUser 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

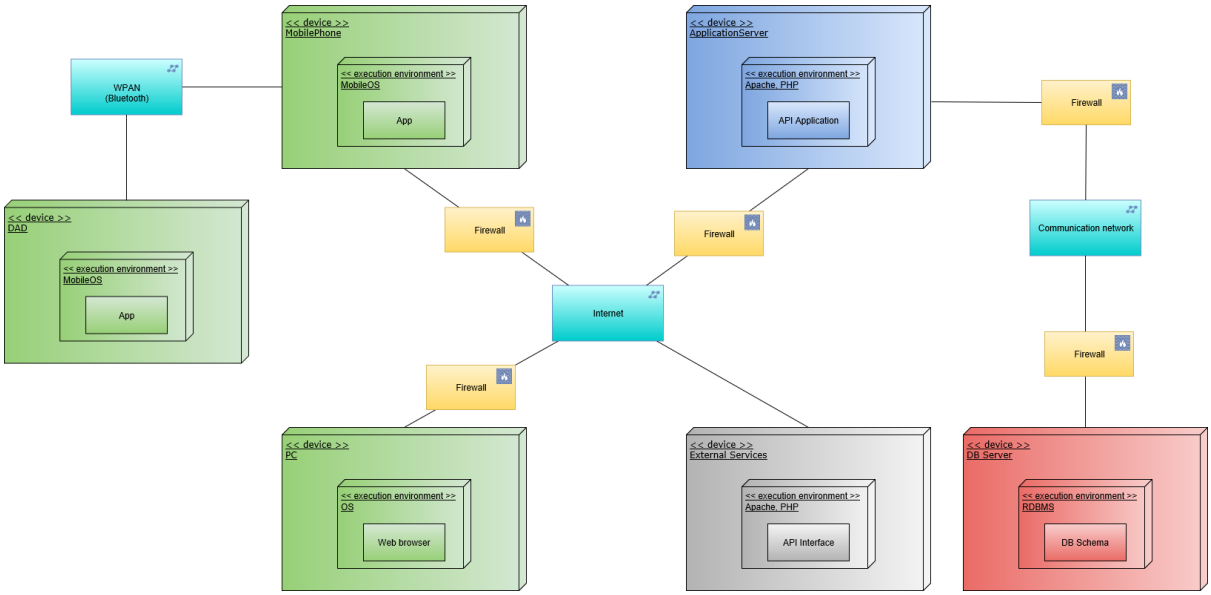
The logic of the application is composed by the interconnection of the services listed above and it works using the connection between each other to exploit different request given by the client. The presence of the router helps to distribute different task more easily inside the appServer and allows 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.




User
- userName: String
- id: int
- password: String
<<interface>>
LoginService
+ login(userName: String, password: String): void
LoginService
+ login(userName: String, password: String): void
<<uses>>
<<interface>>
RegistrationService
+ signUp(email: String, password: String) : void
RegitrationService
+ signUp(email: String, password: String) : void
Individual
- idCode : String
- dad : DataAcquisitionDevice
ThirdParty
- subscriptions: List<Subscription>
- requests: List<Request>
<<interface>>
RequestService
+ createIndividualRequest(individualIdCode: String, requester: ThirdParty): void
+ createGroupRequest(area: List<Location>, startTime: Time, endTime: Time, startDate : Date, endDate: Date, optionalParameters: List<Parameters>) : void
+ evaluateGroupRequest(request: GroupRequest) : boolean
+ downloadData(request: Request) : JSONObject
RequestService
+ createIndividualRequest(individualIdCode: String, requester : ThirdParty) : void
+ createGroupRequest(area: List<Location>, startTime: Time, endTime: Time, startDate : Date, endDate: Date, optionalParameters: List<Parameters>) : void
+ evaluateGroupRequest(request: GroupRequest) : boolean
+ downloadData(request: Request) : JSONObject
Request
- id : int
- status : boolean
0..*
1
<<interface>>
SubscriptionService
+ createIndividualSubscription(request: IndividualRequest): void
+ createGroupSubscription(request: GroupRequest,): void
+ checkIndividualSubscription(request: IndividualRequest): boolean
+ checkGroupSubscription(request: GroupRequest): boolean
SubscriptionService
+ createIndividualSubscription(request: IndividualRequest): void
+ createGroupSubscription(request: GroupRequest,): void
+ checkIndividualSubscription(request: IndividualRequest): boolean
+ checkGroupSubscription(request: GroupRequest): boolean
Subscription
- id : int
- request : Request
<<interface>>
Data4HelpMonitoringService
+ addData4HelpIndividual(user: User, dad: DataAcquisitionDevice): void
+ acquireData(individuals: List<Individual>) : JSONObject
Data4HelpMonitoringService
+ addData4HelpIndividual(user: User, dad: DataAcquisitionDevice): void
+ acquireData(individuals: List<Individual>) : JSONObject
<<interface>>
AutomatedSOSMonitoringService
+ addAutomatedSOSIndividual(individual: Individual, age: Age): void
+ checkHealthStatus(user: AutomatedSOSUser): void
AutomatedSOSMonitoringService
+ addAutomatedSOSIndividual(individual: Individual, age: Age): void
+ checkHealthStatus(user: AutomatedSOSUser): void
AutomatedSOSUser
- age : int
- normalHealthStatus : boolean
<<interface>>
EmergencyService
+ sendEmergencyMessage(user: AutomatedSOSUser, place : Location) : void
EmergencyService
+ sendEmergencyMessage(user: AutomatedSOSUser, place : Location) : void
Organizer
Path
- length : int
<<interface>>
RunManagementService
+ createRun(organizer: organizer, path: Path, city: String, date: Date, time: Time, maxNum: int): Run
+ createPath(organizer: Organizer, vertex: list<Location>): Path
+ getRunList() : List<Run>
RunManagementService
+ createRun(organizer: organizer, path: Path, city: String, date: Date, time: Time, maxNum: int): Run
+ createPath(organizer: Organizer, vertex: list<Location>): Path
+ getRunList() : List<Run>
Runner
- runs : List<Run>
<<interface>>
RunnerService
+ addRunner(runner: Runner, run: Run): void
RunnerService
+ addRunner(runner: Runner, run: Run): void
Run
- city : String
- date : Date
- time : Time
- maxNum : int
- partecipants : List<Runner>
<<interface>>
TrackService
+ followRun(run: Run) : void
+ addRunner(runner: Runner) : void
Trackservice
+ followRun(run: Run) : void
+ addRunner(runner: Runner) : void
Spectator
- runners : List<Runner>
- currentRun : Run

2.3 Deployment view

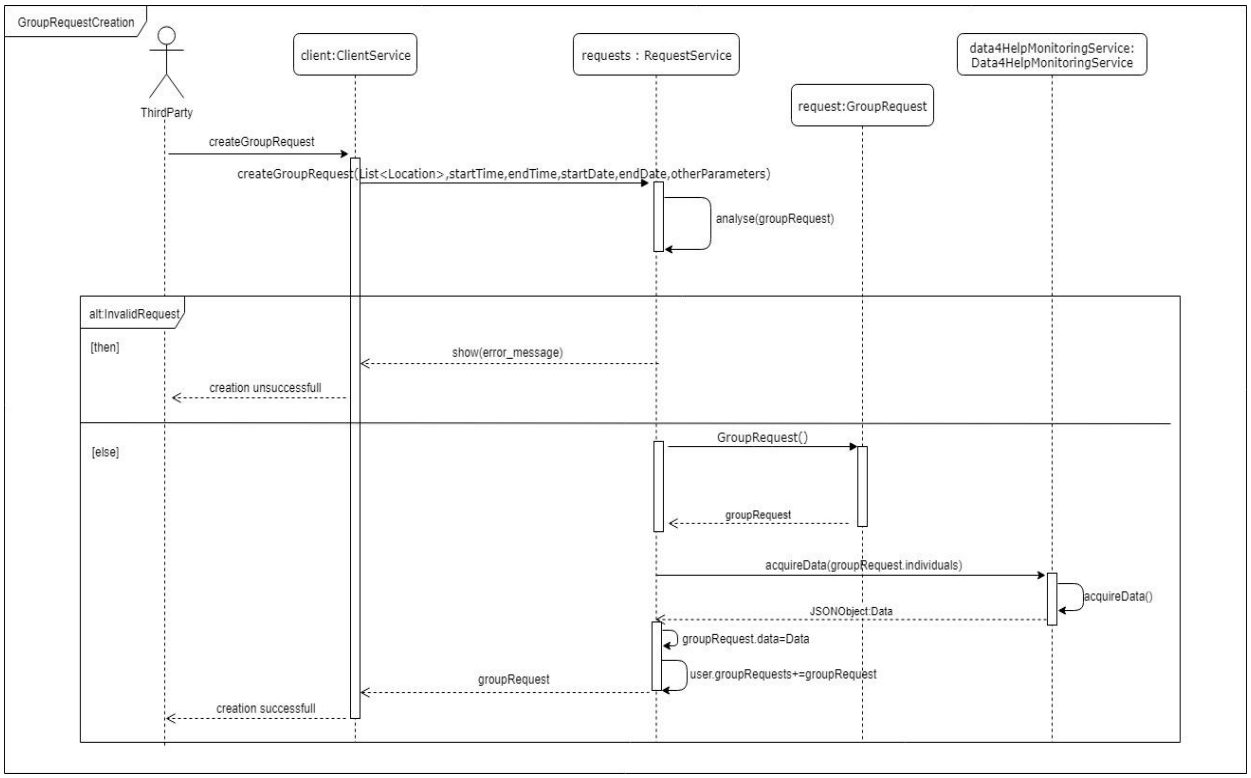


This deployment diagram shows the architecture of the system from a physical view point. Also, the distribution of software among the different hardware nodes is presented. The main nodes involved in the system are:

- *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.

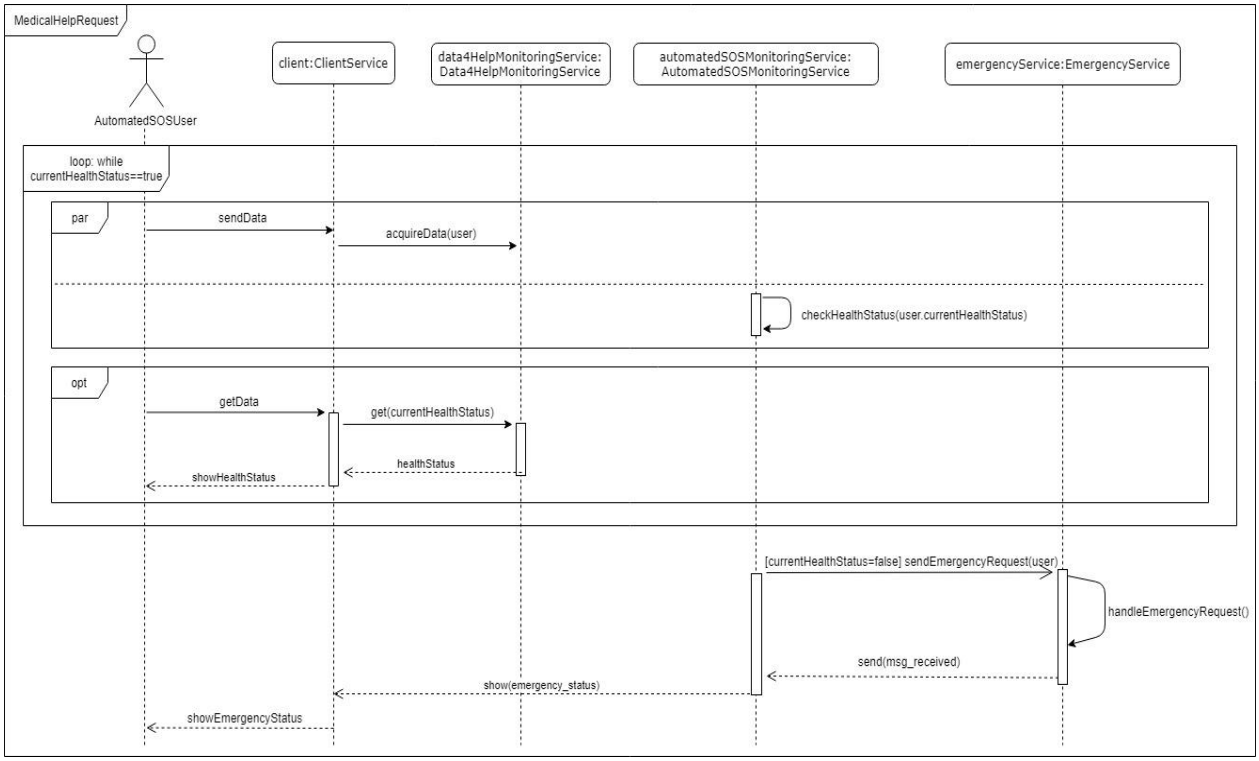
2.4 Runtime View

2.4.1 GroupRequestCreation



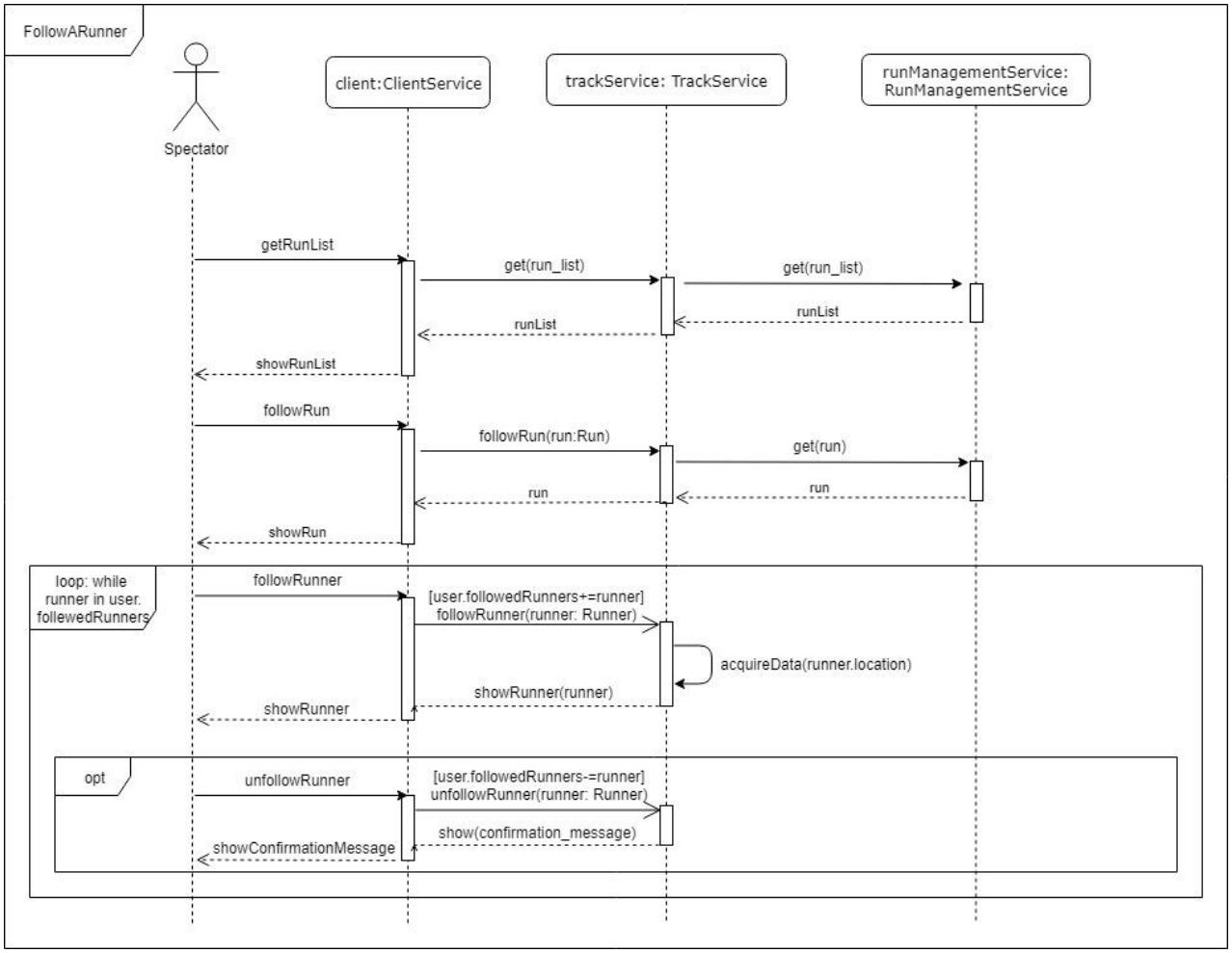
In this sequence is shown the process to create a group request. First the client connects with the RequestService to add a new request by giving it the parameters needed: the locations, time, date and other limitations like age or gender. Then the RequestService examine if the parameters given by the user ensure the privacy to the individuals concerned by checking if the number is higher than 1000. If the number is not reached the request is denied and an error message is shown to the user. Otherwise the RequestService create a new GroupRequest objects and asks Data4HelpMonitoringService to retrieve the data needed to answer the request of the client. After receiving the data, they are sent back to the client, the user is shown a confirmation message and it is given permission to download the date whenever he/she needs to.

2.4.2 MedicalHelpRequest



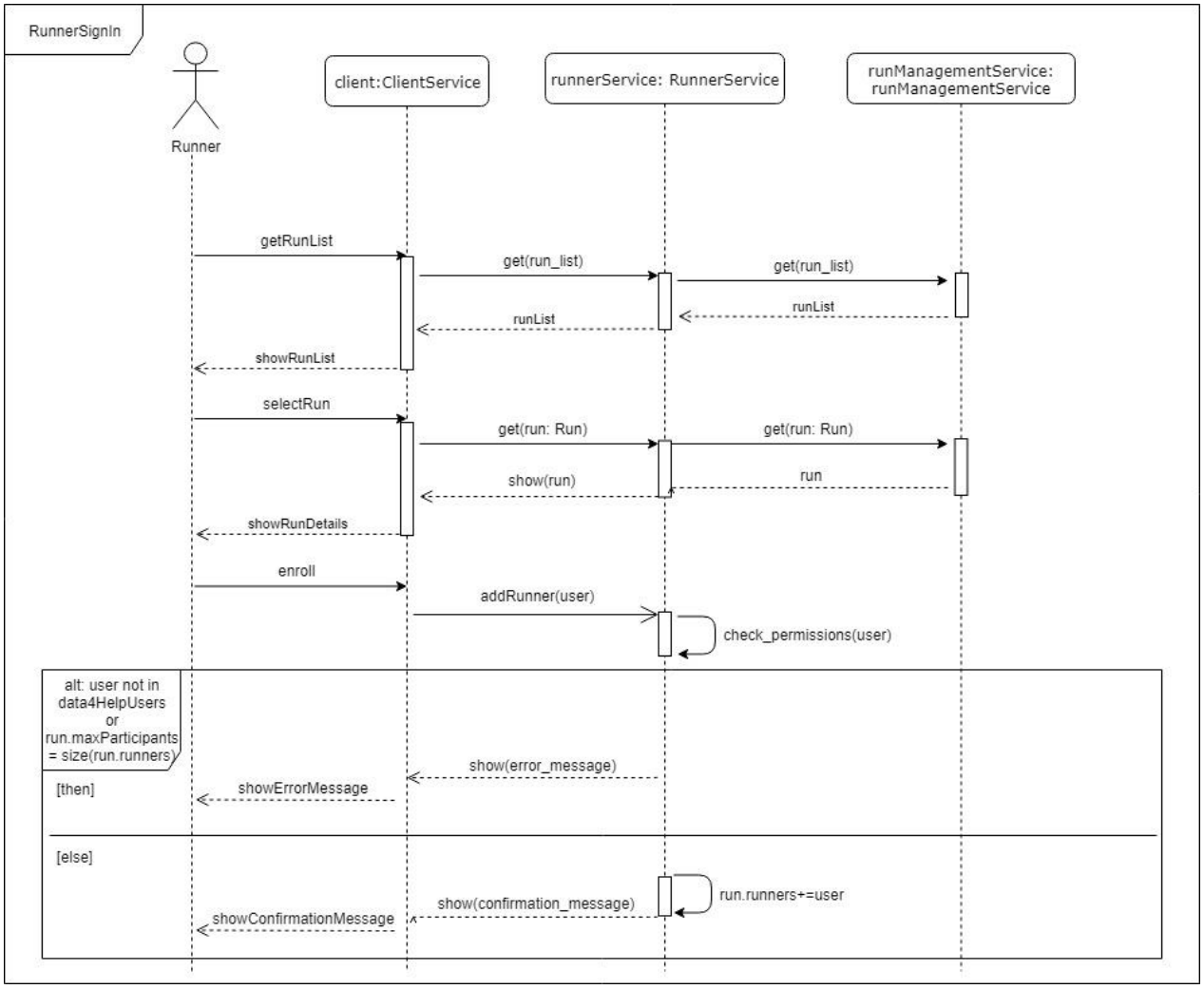
In the diagram above is described the process concerning medical help requests in case of emergency. The monitoring of an AutomatedSOSUser works by exploiting the data saved inside the database that were previously retrieved by Data4HelpMonitoringService from the user. The user may want to check his/her health condition and ask the application for an update. Data4HelpMonitoringService retrieves the last data received from the user's device and show them to the AutomatedSOSUser (this process can happen for all users who activated the monitoring beforehand). In the unfortunate case of an emergency situation, the AutomatedSOSMonitoringService check calls the EmergencyService to send a message to the external medical services including the current health status of the individual in need of help, their location, name and cf/ssn. When the emergency service receives a confirmation from the external parties it forwards it to the AutomatedSOSMonitoringService. At the same time an emergency status screen is shown to the user.

2.4.3 FollowARunner



This sequence diagram describes the operations needed to follow a runner inside a race. First of all, the user asks for a list of the ongoing runs and the TrackService retrieves it by connecting to the RunManagementService and sends it back to the spectator. Then the user starts to follow a precise race and TracksService fulfills the request giving back to the client the run that needs to be shown asking for data to the RunManagementService as well. Then the spectator can choose to follow a particular runner by searching for is name. When the TrackService receives the request, it exploits Data4HelpMonitoringService to get the data he needs the whole time the runner is selected. When the user wants to stop following the runner TrackMe deletes it from the list of runners the user is currently following and stops asking Data4Help for data. Then it sends a confirmation message to the spectator.

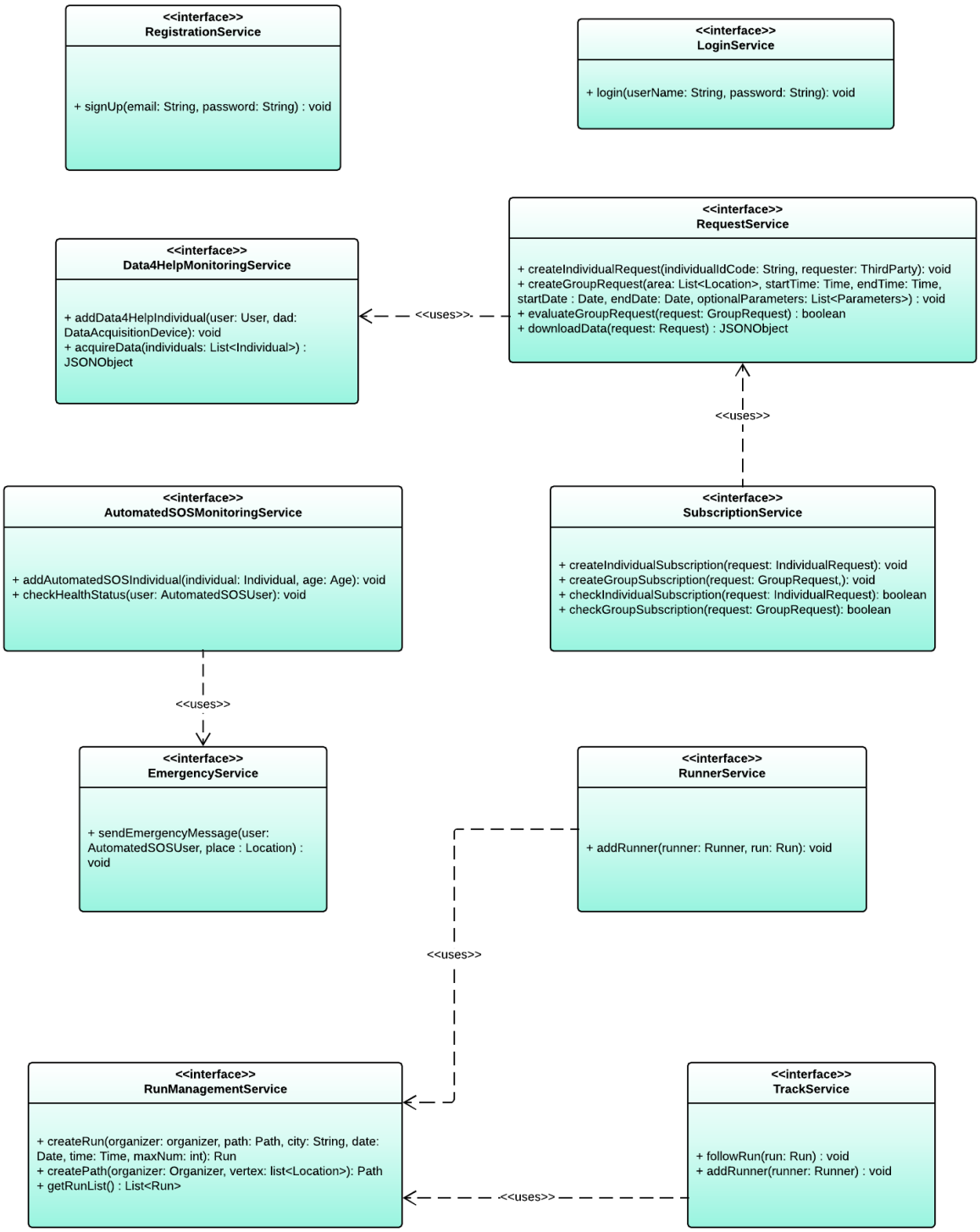
2.4.4 RunnerSignIn



The diagram above describes the process to sign up inside a run. At first the user asks for a list of the ongoing runs and the RunnerService retrieves it from the RunManagementService and sends it back to the runner. Then the user selects a run in which he/she would like to participate, and former service gets it from the latter service and shows the details to the user. At this point the runner asks enroll in the race and the RunnerService checks if the user has active his/her data4Help account and if the actual number of participants in the run has not reached the maximum threshold. If both the conditions are met, then the runner is added to the participants list and a confirmation message is shown to the user. Otherwise the user will receive an error message that briefly describes the problem that has occurred.

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.

2.7 Other design decisions

Since the external street map service is of a crucial importance for correct provision of the Track4Run service, it should integrate the mobile application and the web application.

3 User interface design

The mock-ups for this application are presented in the section 3.1.1 of the RASD.

4 Requirements traceability

During the definition of the design choices we kept a close connection with the requirements and the goals already presented in the RASD. Here is listed how we mapped them with the components illustrated in this document:

[G1]. Third parties can request access to data of some specific individuals (Requirements [R1], [R2], [R13])
- RegistrationService
- LoginService
- Data4HelpMonitoringService
- RequestService

[G2]. Third parties are allowed to request access to anonymized data of groups of individuals (Requirements [R1], [R2], [R13])
- RegistrationService
- LoginService
- Data4HelpMonitoringService
- RequestService

[G3]. Third parties can access the data of specific individuals (Requirements [R1], [R3], [R4], [R5])
- Data4HelpMonitoringService
- RequestService

[G4]. Third parties are able to access anonymized data of groups of individuals (Requirements [R5], [R6])
- Data4HelpMonitoringService
- RequestService

[G5]. Individuals are given the choice to accept or refuse third parties' requests to access their data (Requirements [R1], [R3], [R4])
- Data4HelpMonitoringService
- RequestService

[G6]. To subscribed users is guaranteed anonymity (Requirements [R6])
- RequestService

[G7]. Subscribed costumers receive immediate help when they are displaying serious medical conditions (Requirements [R7], [R8])
- AutomatedSOSMonitoringService
- EmergencyService

[G8]. Spectators are able to know the position of the athletes on the path of the run (Requirements [R9], [R10], [R11])

- RunManagementService
- RunnerService
- TrackService

6 Implementation, integration and test plan

6.1 Implementation plan

The implementation of the system will proceed at the same time of the testing; when a new functionality is implemented it is tested as soon as possible.

The components of Data4Help, AutomatedSOS and Track4Run systems will be implemented giving precedence to the most important ones, and then focusing on those components whose functionalities are less crucial. This because since the central components offer their functionalities to many others, if an error during the implementation occurs, it's better to find it as soon as possible in order to limit the work to be undone.

The criteria used to decide if a component is more important than one other is linked to the complexity of the component itself and its links with other components, principally a component is more significant if its functionalities are used by many other components, and it doesn't depend on others.

On the base of what has just been stated, a possible order of implementation of the different modules of the system is the following one:

1. **Model**, fundamental module which provides the elementary functionalities to provide the system's services. It is also the unique component that has direct access to the DBMS to store and extract data. The order of implementation of the Model's component is quite similar to the same used in the next point for the service interfaces.
2. **Service interfaces and implementation**:
    a. *Data4HelpMonitoringService, RequestService* and *RunManagementService* should be implemented first since they offer key functionalities and as a group of components they don't depend on others.
    b. *AutomatedSOSMonitoringService, EmergencyService* and *TrackService,* this group of components should follow because even it is less fundamental then the previous one, it is still important because it establish a bridge between the system and the external services.
    c. *RunnerService, SubscriptionService, LoginService* and *RegistrationService*, will be the last service interface components to be implemented because they depend on other services and are less complex than the previous ones.
3. **Router and Client-Server connectivity**. After that the system is able to provide its services its necessary to focus on the connectivity. Testing of this part will be advantaged if system's services will be already available since this reduces the necessity of mocking not existing components. In this phase also deals with the implementation of the Router which drives client's requests to the correct service interface.

4. **Views (Client side)**. Last part of the implementation concerns the mobile and web application Views. These component's scope it's only to show users the result of the request services through a GUI, for this reason they could be implemented at the end.

6.2 Integration and testing

6.2.1 Entry criteria

In order to start with the integration of the components and the testing there are some conditions that must be satisfied. The most important one, to have a meaningful integration, is that the components should contain as a minimum the methods that involve one another (at least the creation of the said methods). Another one is that the operations implemented in the components should pass the unit tests to be sure about the fact that the modules are working correctly individually and if a problem is detected it should concern the integration.

If the external services, such as MapExternalService, MedicalExternalService and the DBMS and its server, are not available and ready the integration and testing should continue. However, at the end the system will be integrated with the external services (as soon as they're available).

The most important operations of every component should be completed in order to have significant integration and testing.

6.2.2 Elements to be integrated

The integration of the different components of the application can be divided into the groups listed here:

- *Integration of components with DBMS*
  In this section, we cover the integration of the services of the application that uses the external server with the database. The specific integrations are:

  - RegistrationService, DBMS
  - LoginService, DBMS
  - Data4HelpMonitoringService, DBMS
  - AutomatedSOSMonitoringService, DBMS
  - RunManagementService, DBMS
  - TrackService, DBMS

- *Integration of components with the (other) external services*
  This group contain the integration of every part of the application with the external service they need (the external services are all already existing and correctly functioning). They are listed here:

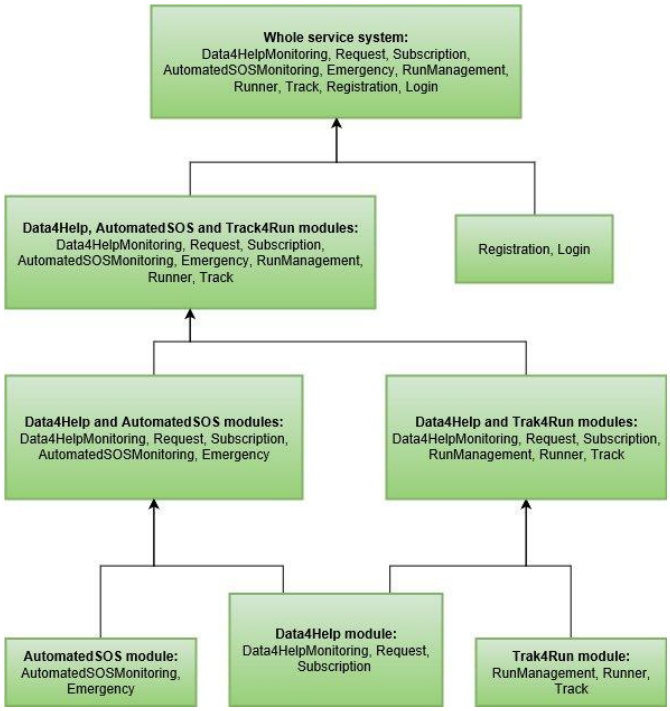
  - EmergencyService, MedicalExternalService
  - RunManagementService, MapExternalService
  - TrackService, MapExternalService

- *Integration of the components of the application server*
  In this group, we include the integration among the parts of the application server. They are the following ones:

  - Data4HelpMonitoringService, RequestService
  - RequestService, SubscriptionService
  - AutomatedSOSMonitoringService, EmergencyService
  - TrackService, RunManagementService
  - RunManagementService, RunnerService

- *Integration of the client, the application server and router*
  In this section we deal with the integration between the client and the application server, that is fundamental to ensure the correct possibility of the user to send requests (through the mobile application) to the server. The router is also integrated with them to ensure the forwarding to the accurate services.

6.2.2.1 Integration plan diagram



This DAG shows the precedencies that should be followed to integrate the system's service modules.

The module at the base of an arrow should be integrated before the module at the top of the arrow.

A similar plan could be adopted to integrate model's components.

The integration of entire modules should be carried on after the integration of couple of components described in the previous paragraph.

6.2.3 Integration Testing Strategy

For Data4Help system, including AutomatedSOS and Track4Run, bottom-up is the choice regarding the integration testing strategy. This decision means that each module at lower levels is tested individually and then with higher modules that rely on the previous ones until all modules are tested. This way is easier to find errors and this choice helps speeding up the operation because it is not needed to wait for all the components to be completed before starting the process. It should be noted that the integration testing should be able to continue even if the external services are not available right away. It is to be noted that it should be given particular attention to Track4Run because its functionalities are linked to the use of external services (the use of the map is very important to this part of the application). For this motivation in this particular case it should be better to wait for the external services to be available to start the integration testing.

7 Effort Spent

7.1

Irene Nizzoli	
Purpose, Scope	3,5
Definition, Document Structure	2
Component and Deployment View	4
Runtime View, Component Interfaces	5
Architectural style, Design patterns	2,5
Requirement Traceability	4,5
Implementation Plan	3
Integration testing	4

7.2

Isabella Piacentini	
Purpose, Scope	3
Definition, Document Structure	2,5
Component and Deployment View	5
Runtime View, Component Interfaces	6
Architectural style, Design patterns	2,5
Requirement Traceability	3
Implementation Plan	2,5
Integration testing	4

7.3

Elio Salvini	
Purpose, Scope	2
Definition, Document Structure	2,5
Component and Deployment View	4
Runtime View, Component Interfaces	5
Architectural style, Design patterns	6
Requirement Traceability	2
Implementation Plan	4
Integration testing	3

8 References

- Specification document “Mandatory Project Assignment AY 2018-2019”
- Slides – “Design”, “Architecture and Design in Practice”