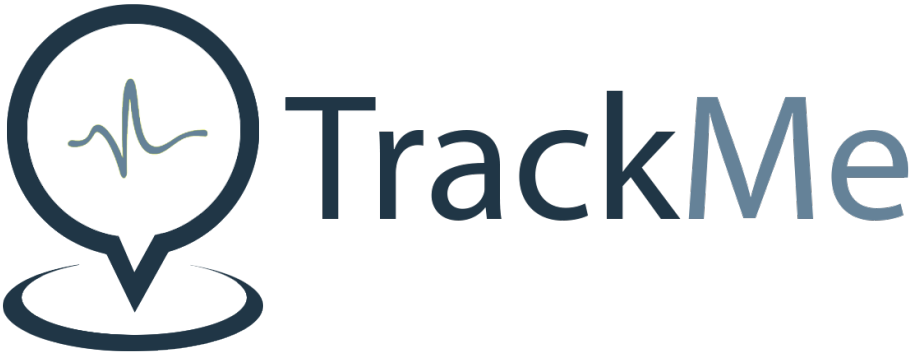
POLITECNICO
MILANO 1863

M.Sc. Computer Science and Engineering
Software Engineering 2 Project



Design Document

Gargano Jacopo Pio, Giannetti Cristian, Haag Federico

10 December 2018

GitHub Repository: https://github.com/federicohaag/GarganoGiannettiHaag

Version 1.0

Contents

Chapter 1

Introduction

1.1 Purpose

TrackMe wants to offer a service named "Data4Help" on top of which two services, named "AutomatedSOS" and "Track4Run", will be built.

Data4Help will be a system collecting data of *Users* through a *Smart wearable* connected to their smartphone. This data may be sent to *Third parties* after *User* consent. AutomatedSOS will constantly monitor *User data* to allow for immediate assistance. Track4Run will allow individuals to organize running competitions, to enroll in or watch one.

More details may be found in section 1.1 of [3].

In this document, the design of the system to be will be explained and analyzed in detail. This includes the identification of all the components and the design decisions regarding the structure and the patterns to be implemented. Moreover, implementation and testing will be discussed and planned.

1.2 Scope

TrackMe proposes to offer its system in a world in continuous technological evolution, where almost everyone owns a smartphone and is always connected to the Internet. By using Data4Help, *Users* will integrate their everyday activities with constant collection of their health data and position, through sensors of their *Smart wearable* and smartphone GPS. By adding *Third party Services* they may benefit of data monitoring and analysis with useful insights on their daily activities.

Users may enhance data collection by adding AutomatedSOS to their *Services*. This *Service*, by constantly monitoring their data, will allow for *Anomalous data*identification and immediate assistance for the *User in need*. This can

be crucial for individuals with health issues and elders living alone. People fond of running have the possibility of enrolling in a *Run* listed in Track4Run. Their data, including position and health data, will be shown to *Spectators*, who can watch the *Run*. The process of organizing running competitions will be carried out by *Organizers* through Track4Run.

More details may be found in section 1.2 of [3], including a detailed analysis of shared phenomena.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

User: registered individual of Data4Help who agreed on the acquisition and processing of their data (see *User data*).

User data: *User*'s health data and location acquired by Data4Help.

Third party: a company that is willing to access *User data* stored in TrackMe's database.

Service: application available for Data4Help *Users*, generally offered by a *Third party*.

Group data: set of *User data* acquired by Data4Help. The set of *Users* is determined by specific characteristics and constraints defined by the *Third party* requesting the data. When sent to the *Third party*, this data is anonymized.

Smart wearable: smart devices that can be worn on the body as accessories. These devices are required to have specific sensors for data acquisition. They must be connected to an external device, such as a smartphone.

Anomalous data: health data that is outside certain intervals identifying a *User* normal health condition.

User in need: registered user of AutomatedSOS in need of assistance since their health data is *anomalous*.

Run: running competition registered on Track4Run.

Organizer: person organizing a *Run*.

Spectator: person participating as spectator of a *Run*.

Participant: *User* who added Track4Run to their services, participating in a *Run*.

1.3.2 Acronyms

GPS: Global Positioning Service

GUI: Graphical User Interface

1.3.3 Abbreviations

G_n: n^{th} goal

R_n: n^{th} requirement

1.4 Revision History

1. Version 1.0 - 10^{th} December 2018

1.5 Reference Documents

- Rumbaugh, Jacobson, Booch. 1999. *The Unified Modeling Language Reference Manual*. Addison-Wesley.
- *UML-Diagrams*. https://www.uml-diagrams.org/

1.6 Document Structure

This document is divided into seven chapters.

First chapter In this chapter is summarized the project. This has been already introduced and described in RASD so here it is shown briefly only in terms of purpose and scope. Moreover, the chapter contains the definitions, acronyms and abbreviations needed to properly understand the sections following. All the documents used during the development of this project are listed at the end of the chapter.

Second chapter In this chapter is described a proposal of architecture able to deliver an implementation of the system to be described in the RASD. The architecture takes into consideration all the goals and requirements listed in the RASD. A particular attention is dedicated to the satisfaction of non functional requirements (e.g. reliability, performances). The architecture is described following a top-down approach: from overview to detailed components descriptions and interaction flows.

Third chapter In this chapter are listed and showed all the user interfaces of the system to be. The description is given with a particular attention for details and interaction between different interfaces.

Fourth chapter In this chapter requirements defined in the RASD are mapped to the design elements defined in in Chapter 2.

Fifth chapter In this chapter is presented the list of activities needed to implement the system to be, according to the architecture showed in Chapter 2 and according the user interfaces showed in Chapter 3. Activities are analyzed to identify eventual constraints that force certain activities to be executed before or after other else.

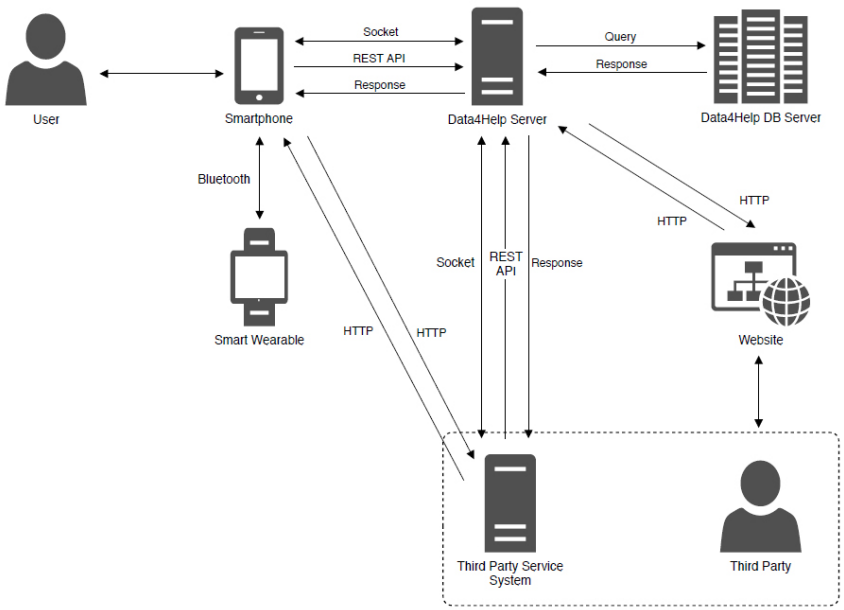
Sixth chapter Effort spent by all team members is shown as the list of all activities done during the realization of this document.

Seventh chapter References of documents that this project was developed upon.

Chapter 2

Architectural Design

2.1 Overview

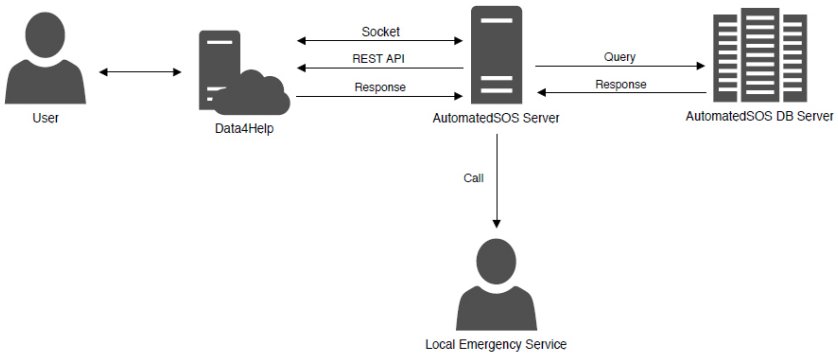


Data4Help Overview

This diagram is a general overview of the system to be.
The *User* interacts with their smartphone performing several actions, including but not limited to managing their services, sending their consent, and using a service. The smartphone is constantly connected to the Data4Help Server both calling it through its REST APIs. If requested by the third party, Data4Help can enstablish a socket connection between itself and specific users to collect

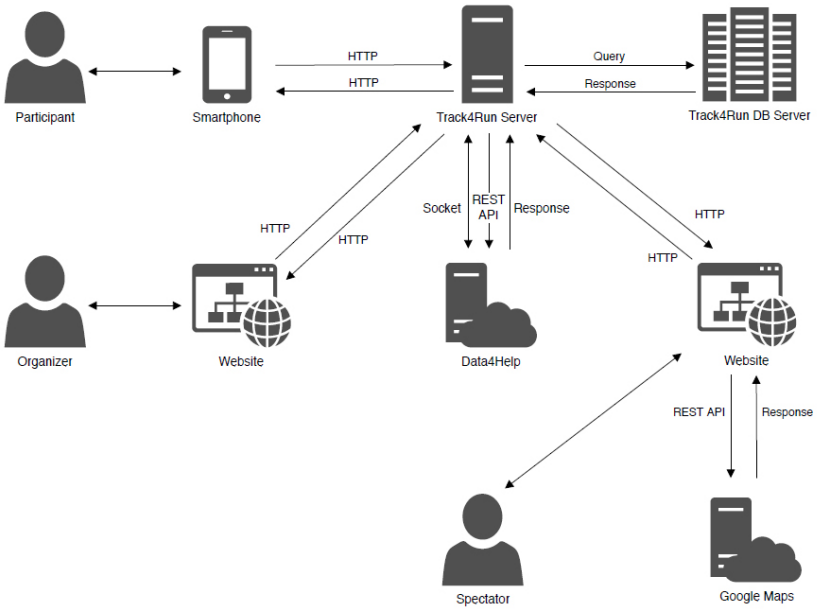
data at the maximum throughput frequency enabled by the users' smartwearables. This feature it's used for example by *Track4Run* during a *Run* or by *AutomatedSOS* when a *User* has parameters that are getting near to the alarm range. The smartphone is also connected to the *User's Smart wearable*, allowing for continuous data collection.

The Data4Help Server is the core of the system to be. It manages all *User* requests, collects its data and sends them to the database - Data4Help DB Server. It also sends through a socket connection newly collected *User data* only to those *Services Users* are subscribed to. It allows the forwarding of *Third Party User data* requests to *Users* and supports the interaction of *Third Parties* with the system through their dedicated website. Lastly, the Third Party Service System communicates with the *User's* smartphone via HTTP. Data4Help offers the possibility of embedding the Third Party Service website into the Data4Help app to enable users using custom interfaces for every Service. Third Party has just to provide to Data4Help the address of the website and should develop it in a way that is mobile friendly.



AutomatedSOS Overview

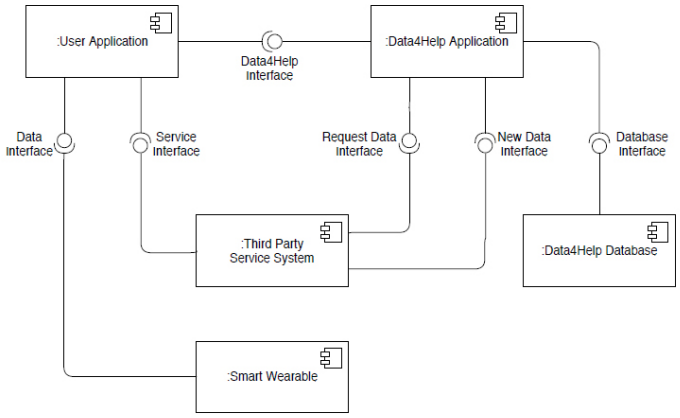
AutomatedSOS is based on a central server which receives *User data* as soon as it is produced and constantly analyzes it. The data is sent from Data4Help to AutomatedSOS via a socket connection always active. When AutomatedSOS recognizes a *User* as a *User in need*, it calls Local Emergency Services and store in the AutomatedSOS database all the relevant information, including which which Local Emergency Services are in charge of assisting a specific *User in need*.



Track4Run Overview

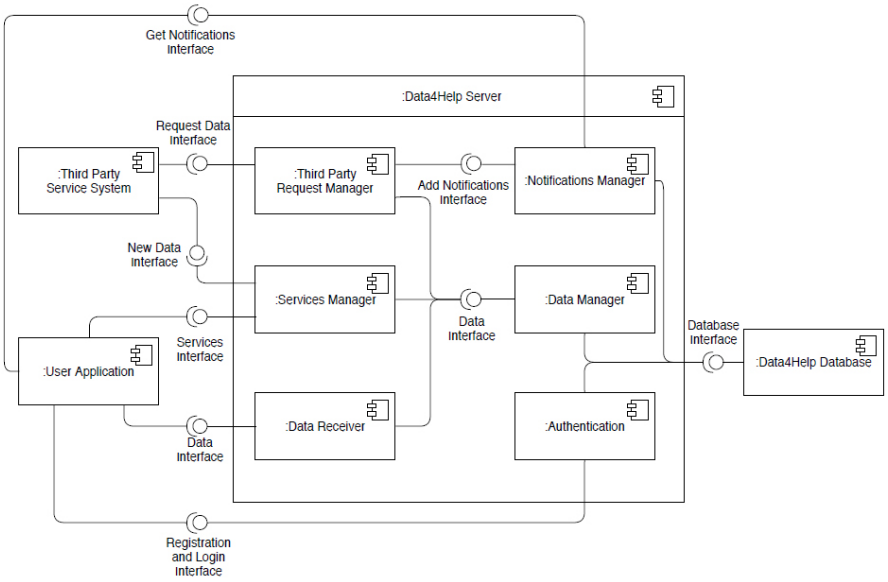
The overview of Track4Run gives a general idea of the several components that comprise this *Service*.
The Track4Run Server manages all the actions that can be performed by the authors in this context. First of all, it allows *Organizers* to create a *Run* and manage it through a dedicated website. Moreover, it communicates with Data4Help in order to allow *Users* to enroll in a *Run* as *Participants* and acquire their real time data while running. This data will be displayed in the dedicated *Spectators* website, together with the real time position of *Participants*, retrieving the real world map of the *Run* through Google Maps. All data that needs to be stored, is passed to and retrieved from a database - DB Server. This data is mainly made up of *Run* information.

2.2 Component View



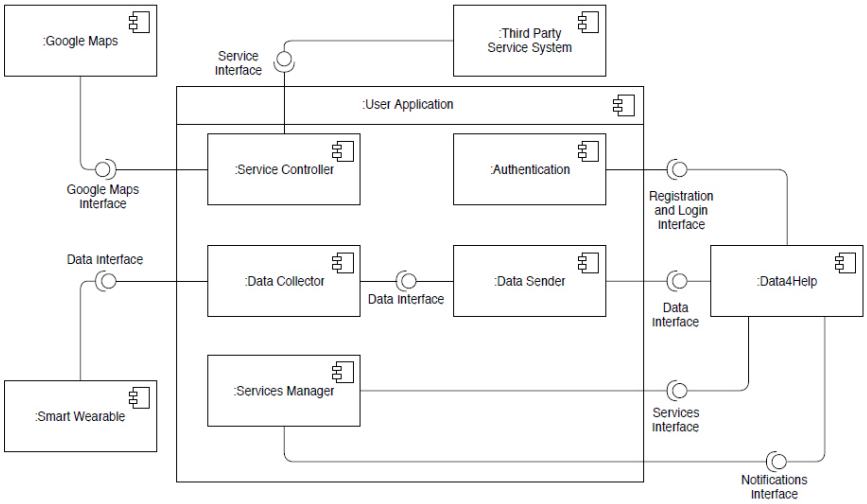
Data4Help Component Diagram Overview

A general view of the main components of the system to be is given. The core components are the Data4Help User Application and the server side Application. They communicate through three different interfaces offered by Data4Help. These will be further explained in the following two diagrams. Here they are simplified under the Data4Help Interface. The User Application receives data from the Smart Wearable through the Data Interface and pushes them to Data4Help. It finally uses the Service Interface to retrieve the *Service* website and interact with it. The server side Application provides an interface for the Third Party Service System to send data requests. It also sends newly collected data to the Third Party Service System through the New Data Interface. It finally communicates with the database through the offered Database Interface.



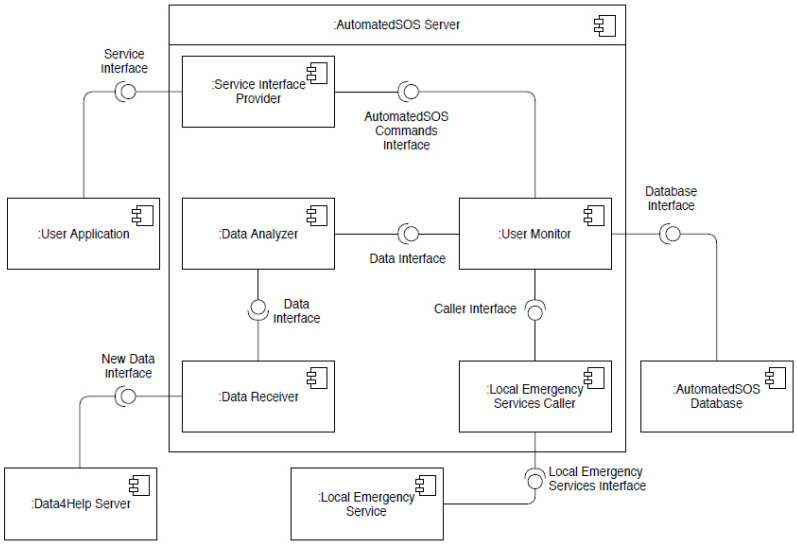
Data4Help Component Diagram

These are the internal components and interfaces of the Data4Help server side Application, together with the interfaces offered to external components. The Authentication component is needed to authenticate the *User* through the User Application. It needs to access the database where all login information is stored through the Database Interface. The Data Receiver is needed to receive data from the User Application as soon as it is collected. It then sends it over to the Data Manager through the Data Interface, whose function is to manage all *User data* stored by Data4Help. In fact, it uses the Database Interface to query it. The Data Manager has a crucial role when it comes to *Third party* requests. A *Third party* may send a request through the Request Data Interface. The request is processed by the Third Party Request Manager. It uses the Add Notifications Interface when a *User* has not yet given consent to a *Third party Service* and the *Third party* wants to acquire its data. It also uses the Data Interface of the Data Manager to retrieve *User data* and *Group data*. The interaction of this component with the others is further explained in section 2.4. Furthermore, the Notifications Manager is needed to provide notifications to the User Application. In fact, the application constantly queries the Data4Help Server for new notifications through the Notifications Interface. The Notifications Manager interacts with the database to retrieve all notifications for a given *User*. Finally, the Services Manager implements the Services Interface, which allows *Users* to add new *Services* or manage the ones they already have through the User Application. Moreover, it uses the New Data Interface offered by the Third Party Service System when it needs to send new *User data* or *Group data* to *Third parties*.



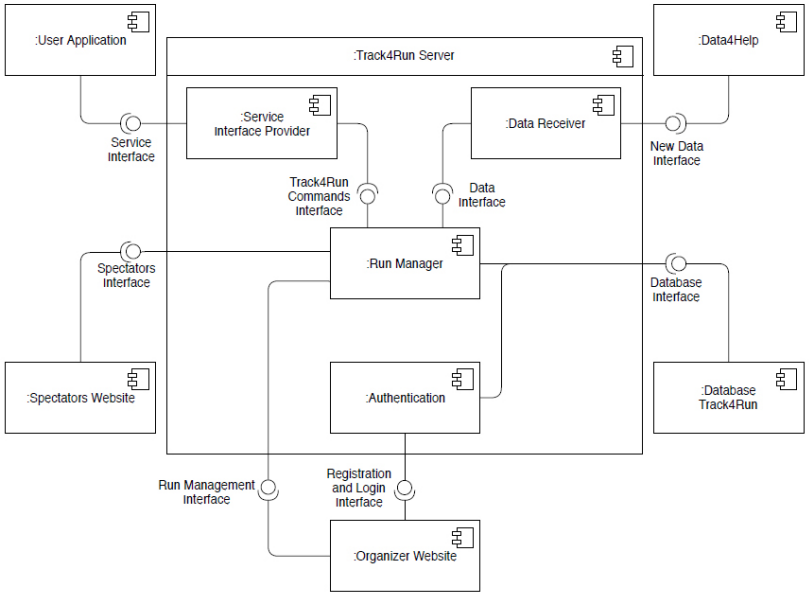
Data4Help User Application Component Diagram

The User Application is used by *Users* to interact with Data4Help and the *Services* they added. They can do this through several components and interfaces. They must be authenticated in order to use the Application. In order to do so, through the Authentication component, they can sign up and login into their Data4Help account. Moreover, the User Application will use the Registration and Login Interface offered by the server side Application to authenticate into it. The User Application collects *User data* from both the smartphone and the *Smart wearable* through the Data Interface of the Data Collector. This forwards the data to Data4Help through the Data Interface offered by the Data Sender which sends them to Data4Help. The Services Manager allows *Users* to add and manage their *Services*. This component exploits the Services Interface offered by Data4Help to accomplish its main functions. Moreover, the it is in charge of constantly asking Data4Help for new notifications through the Notifications Interface. Furthermore, the purpose of the Service Controller is to allow the usage of a *Service* on the User Application through the Service Interface. In fact, the Third Party System will offer this interface for the User Application to connect and interact with. Finally, the Service Controller connects to Google Maps to obtain maps information for *Services* like Track4Run.



AutomatedSOS Component Diagram

The main component of AutomatedSOS is the User Monitor. It offers the AutomatedSOS Commands Interface for the Service Interface Provider to forward commands received from the interaction with the User Application through the Service Interface. In particular, it allows *Users* to reactivate their data monitoring. This component also uses the Database Interface to store which Local Emergency Services are in charge of assisting a specific *User in need*. When new data is received from Data4Help, through the New Data Interface, it is passed to the Data Analyzer. The Data Analyzer compares the *User data* against certain thresholds, possibly identifying it as *Anomalous data*. When a *User* is identified as a *User in need*, the User Monitor sends a call request through the Caller Interface to the Local Emergency Services Caller, which calls the Local Emergency Services.



Track4Run Component Diagram

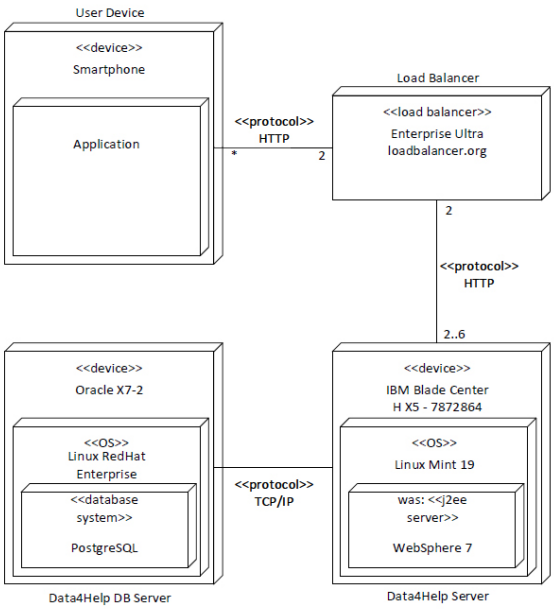
At the core of Track4Run lies the Run Manager. It is responsible for everything that concerns a *Run*. It offers several interfaces. First of all, it offers the Track4Run Commands Interface, for the Service Interface Provider to forward commands received from the interaction with the User Application through the Service Interface. In particular, it gives *Users* the possibility to enroll in a *Run* as *Participants* and to lookup *Runs*. The Run Manager also gives *Organizers* the possibility to create and manage a *Run* through the Run Management Interface from the *Organizers* dedicated website. It uses the Database Interface to store relevant data in a database. Moreover, it offers the Spectators Interface to *Spectators* to watch a *Run* through the *Spectators* dedicated website. The Authentication allows *Organizers* to sign up and login in the dedicated website through the Registration and Login interface. Finally, the Data Receiver offers the New Data Interface, as all *Services* do so as to receive new data from Data4Help. It forwards the just received data to the Run Manager through the Data Interface, which is in charge of displaying it to *Spectators* and of storing it in the database.

2.3 Deployment View

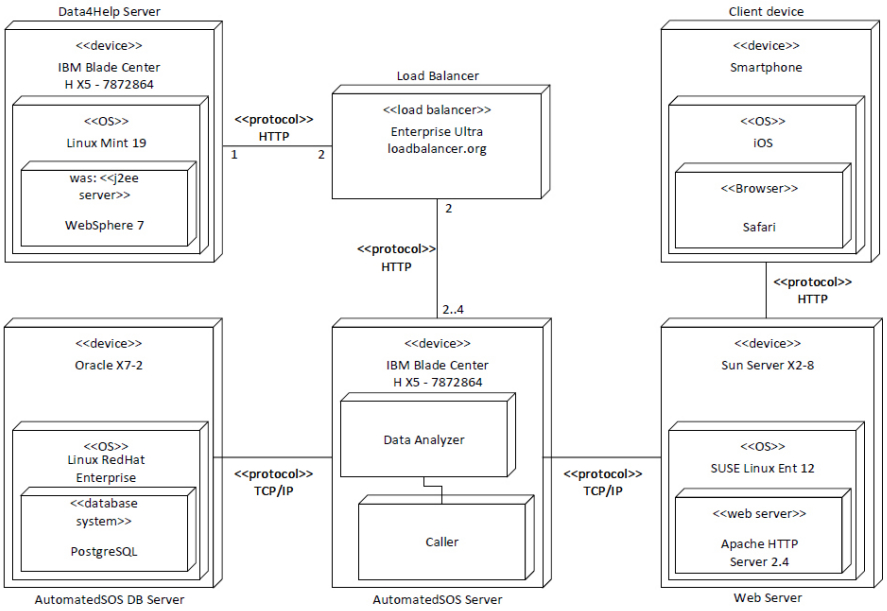
The following are deployment diagrams. Their purpose is to show how the three systems will look like from a physical point of view - both hardware and software - when they will be implemented and delivered.

All three systems will have a central server architecture split in several servers

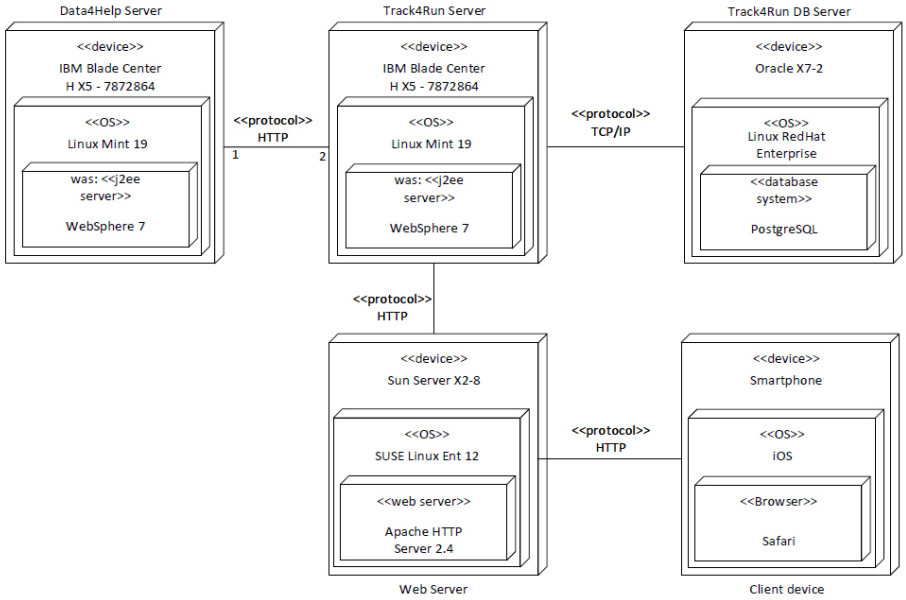
to ensure reliability and availability. A load balancer is included in all three systems to balance the calls from the clients and send them to the less busy servers. There are two load balancers for availability. All servers communicate with the relative database through a TCP/IP protocol. Track4Run has a Web Server since it needs to offer a website for *Spectators* and *Organizers*.



Data4Help Deployment Diagram



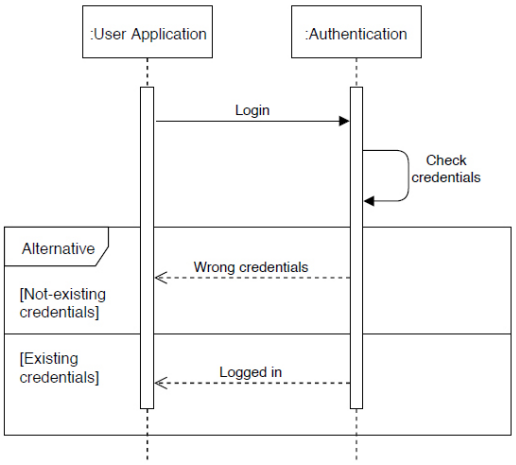
AutomatedSOS Deployment Diagram



Track4Run Deployment Diagram

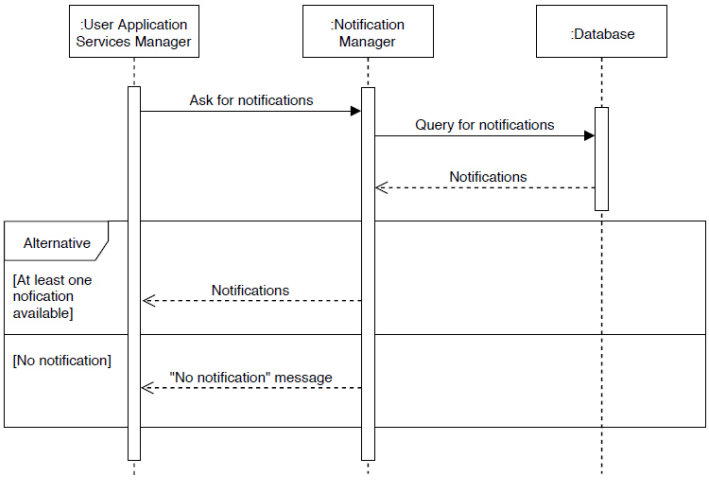
2.4 Runtime View

The following is a representation through sequence diagrams of the main functions and processes of the system to be. The actors of the diagrams are the components represented in section 2.2.



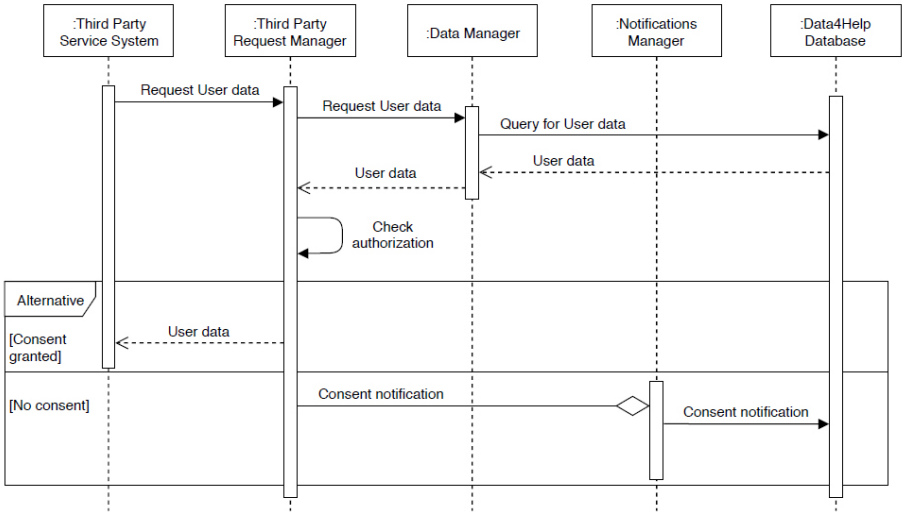
Login Sequence Diagram

In this sequence diagram the *User* login on the User Application is shown. The component involved in the process is the Authentication, which checks the credentials inserted by the *User*.



Notifications Sequence Diagram

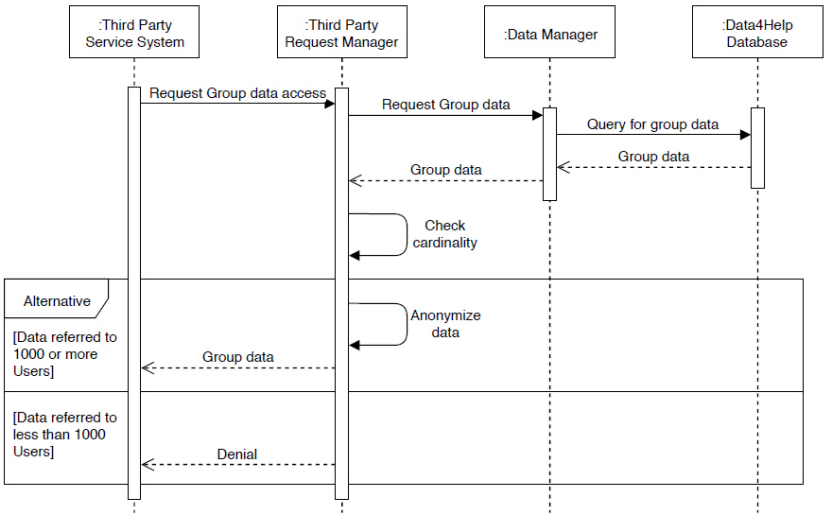
The User Application is able to receive notifications from Data4Help by querying the server side Application for new notifications. The Services Manager asks the Notifications Manager for new notifications. If there are any new notifications for the *User*, then it returns them, otherwise it tells the Services Manager that there are no new notifications.



User Data Request Sequence Diagram

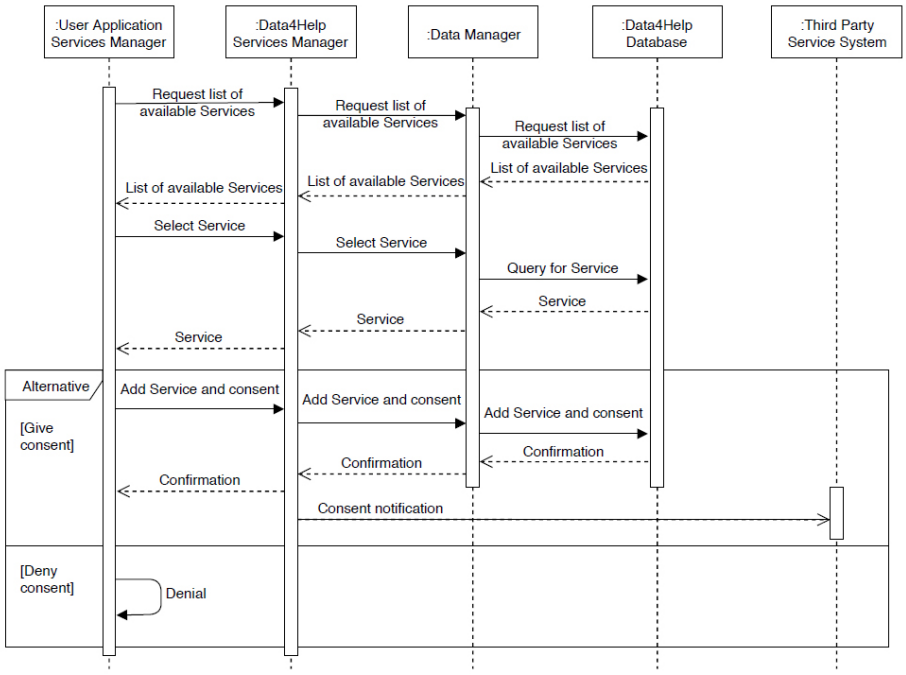
A *Third party* may request access to a specific *User data*. It sends a request to the Third Party Request Manager, which forwards it to the Data Manager.

This last component queries the database for *User data*. Assuming that the query returns non-empty *User data*, the Third Party Request Manager gets the data and checks if the *User* gave consent to the specific *Third party Service* that is requesting their data. If the consent was granted, then the *User data* is sent to the Third Party Service System. Otherwise, the Third Party Request Manager asks the Notifications Manager to add a new notification for that *User*, which is then stored in the database. The notification refers to the fact that the *Third party Service* wants to acquire their data.



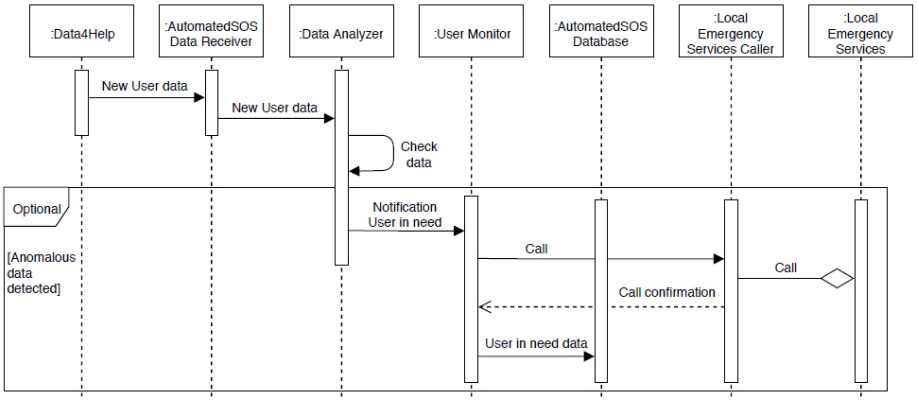
Group Data Request Sequence Diagram

When performing a *Group data* request, the *Third party* sends a request from the Third Party Service System to the Third Party Request Manager of Data4Help. This last component asks the Data Manager for group data, which queries the database for data. The *Group data* is sent back to the Third Party Request Manager, which checks the cardinality. If the data refers to more than 1000 *Users*, the data is anonymized and sent to the Third Party Service System which made the request. Otherwise, a request denial is sent back.



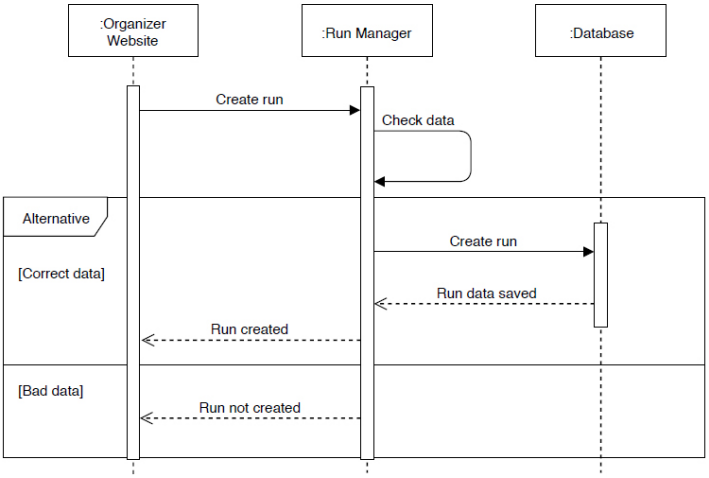
Add New Service Sequence Diagram

When a *User* wants to add a new *Service*, the User Application Services Manager sends a request for available *Services* to Data4Help. The request is received by the Services Manager that forwards it to the Data Manager, which queries the database. The list of *Services* is sent back to the User Application. The *User* then selects a *Service* and its details are returned to the User Application by Data4Help. If the *User* decides to add the *Service* and give consent to their data acquisition by the Third party, then it sends the choice to the Services Manager. This last one forwards the information to the Data Manager, which stores the consent in the database, together with adding the *Service* to the list of *Services* for the specific *User*. Moreover, the Services Manager sends a consent notification to the Third Party Service System. A confirmation is sent back to the User Application.



User In Need Assistance Sequence Diagram

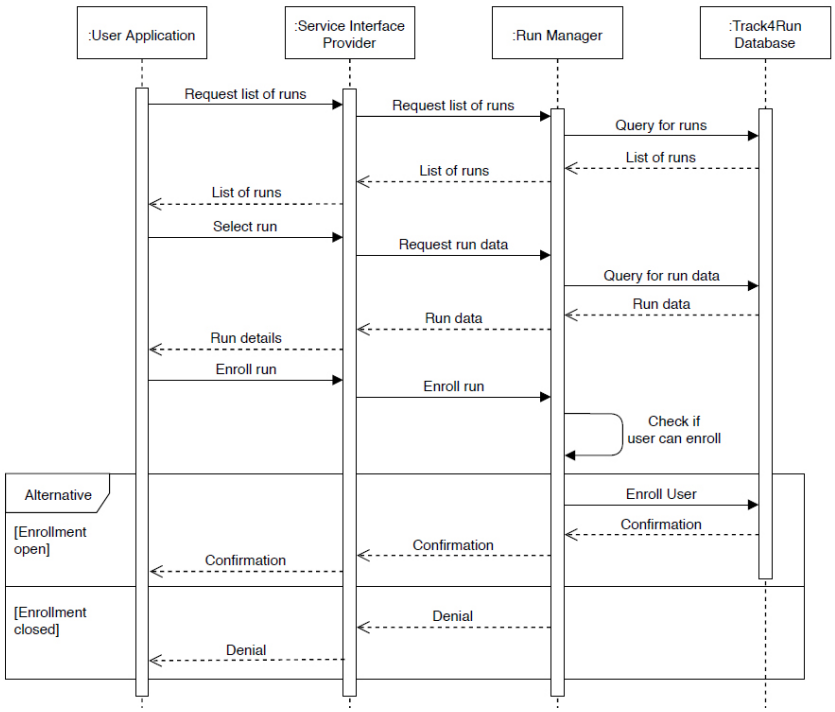
When new *User data* is collected by Data4Help, if the *User* is subscribed to AutomatedSOS, the data will be sent to AutomatedSOS for analysis. In particular, the Data Receiver will receive the data, and will pass it over to the Data Analyzer which compares it against certain thresholds. If the data is identified as *Anomalous data*, then the Data Analyzer notifies the User Monitor that the *User* is in need of assistance. The User Monitor sends a call request to the Local Emergency Services Caller, which calls the Local Emergency Services. This component sends a call confirmation back to the User Monitor and finally the User Monitor stores the *User in need* data in the database.



Run Organization Sequence Diagram

When they want to organize a *Run*, *Organizers* login into their dedicated website and send a *Run* creation request to the Run Manager of Track4Run.

This last one checks the *Run* data is correct and if it is, it creates the *Run*.



Participant Enrollment Sequence Diagram

In this diagram the *Participant* enrollment process is shown in details. A *User* who wants to enroll in a *Run* needs to first have all the available *Runs*. This is done by querying the Service Interface Provider through the Service Interface. The Service Interface Provider forwards the User Application requests to the Run Manager, which queries the database. It returns the list of available *Runs* to the User Application. Assuming that the list is not empty, the *User* will express their selection and the Data4Help components will return the *Run* details. The *User* may then select to enroll in the selected *Run*. The Run Manager is in charge of checking if the *User* can enroll or not - the latter may happen because the enrollment is closed. If the *User* can enroll in the *Run*, the Run Manager saves the enrollment in the database and sends a confirmation to the User Application. Otherwise, an enrollment denial is sent back.

2.5 Component Interfaces

The following is a representation of the component interfaces that are offered by the system to be. The interfaces are both of external and of internal components so as to provide a clearer explanation of the main methods they

offer.

The naming convention adopted is of the form "Component_InterfaceName", where "Component" is the component it is offered by and "InterfaceName" is the name of the interface. The first interface - "dataInterface" - is used and offered by many components and it is needed to send and receive data. When the component name is "service", it means that the interface must be offered by all the *Third party Services*.

The majority of the following interfaces must include a way to check that who is asking to access any kind of data is authorized. For example, the interface through which an user can retrieve the services to which is subscribed to must check that who is sending the request is really the user who is declaring to be. Interfaces that have this need are marked in the following list with a comment "requires auth". From an implementation point of view, a possible solution could be providing always the answer encrypted with an RSA algorithm: the encryption is done with a key and the user, to read data, must use the other key that was generated in his mobile phone during the sign up process.

```
//Generic Interfaces

public interface dataInterface{
   boolean receiveData(Data data); // requires auth
}

public interface service_newDataInterface{
   boolean receiveData(Data data); // requires auth
}

public interface service_serviceInterface{
   boolean executeCommand(String command);
   //command varies from Service to Service (e.g. it may be an API call)
}


//Data4Help Interfaces
public interface d4h_getNotificationsInterface{
   List<String> getNotifications(String username);
}

public interface d4h_addNotificationsInterface{
   boolean addNotifications (List<String> notifications, String username)
}

public interface d4h_registerAndLoginInterface{
   boolean isUsernameValid(String username);
   boolean isPasswordValid(String password);
   boolean login(String username, String password);
```

```
    boolean userExists(String username);
    boolean signUp(String fullname, Date dateOfBirth, boolean gender,
        String ssn, String email, String password);
    //gender: 1 for Male, 0 for Female
}

public interface d4h_servicesInterface{
    List<Service> lookupServices(String namelike);
    List<Service> getAllServices();
    List<Service> getUserServices(String username); // requires auth
    Service getServiceByID(int serviceID);
    boolean addServiceToUserServices(int serviceID, String username); //
        requires auth
    boolean deleteServiceFromUserServices(int serviceID, String
        username); // requires auth
}

public interface d4h_requestDataInterface{
    Data getUserData(UserDataRequest request); // requires auth
    List<AnonimousData> getGroupData(GroupDataRequest request); //
        requires auth
}


//AutomatedSOS Interfaces

public interface asos_automatedSOSCommandsInterface{
    boolean reactivateMonitoring(String username); // requires auth
    boolean subscribeUser(User user);
    boolean removeUser(User user);
}

public interface asos_callerInterface{
    boolean call(String username, Data userData);
}


//Track4Run Interfaces

public interface t4r_automatedSOSCommandsInterface{
    boolean enroll(User user, Run run); // requires auth
    boolean removeUser(User user, Run run);
    List<Run> getAvailableRuns();
}

public interface t4r_spectatorsInterface{
    Run getRunByID(String ID);
    List<Run> getAllRuns();
}
```

```
public interface t4r_runManagementInterface{
   boolean createRun(String name, Date date, DateTime time, Position
       start, int maxNumberOfParticipants, Date enrollmentClosureDate);
   boolean setPath(List<Position> path);
}

public interface t4r_registerAndLoginInterface{
   boolean isUsernameValid(String username);
   boolean isPasswordValid(String password);
   boolean login(String username, String password);
   boolean organizerExists(String username);
   boolean signUp(String fullname, Date dateOfBirth, String email,
       String password);
}

public interface db_queriesInterface{
   getUserByUsername(String username);
   insertUser(String fullname, Date dateOfBirth, boolean gender, String
       ssn, String email, String password);
   deleteUser(String username);
   insertData(Data data);
   getUserData(String username);
   deleteAllUserData(String username);
   insertServiceForUserServices(String serviceID, String username);
   deleteServiceFromUserServices(String serviceID, String username);
   getAllUserServices(String username);
   getAllServices();
   insertNotification(String notification);
   //TODO: capire con prof. DiNitto se fare tutta la lista come qui o
       fare solo doQuery(String query)
}
```

2.6 Selected Architectural Styles

Three Tier Architecture

For all the three systems to be developed, the chosen architecture is a three tier one. In fact, Data4Help, AutomatedSOS and Track4Run, all need three main components: one or more clients, a server and a database, using the remote presentation pattern.

In the remote presentation pattern the client is only responsible of providing a GUI to the user for them to interact with the server. Both Data4Help and *Third party Services*, in fact, need their users to interact with the application. For instance, a *User* may add a new *Service* or they may enroll in a *Run*. First of all, the decision of not having any logic on the client side allows to manage the processes only on the server side. This is crucial in order not to have a heavy impact on the user smartphone performance. Moreover, this decision also has an effect on maintainability, since the application will not need any updates

if a logic change is required in the future: only the server will be updated. Furthermore, by doing this, if *Third party Services* already have a GUI in the form of a website, they do not need to implement a different GUI for the user to use: the *Service* website will be displayed on the client application. This decision also allows a platform independence for *Third party Services* GUIs. In fact, the GUI they offer is accessible from any operating system, since it is a website, with no need to have platform specific GUIs.

The server is the core of the system. For all three systems, it manages all the processes and the logic. It queries the database and it does not need any GUI. All users querying the server have a dedicated application, in the form of a website or of a smartphone application. For instance, the *Organizers* manage *Runs* from the *Organizers* website.

The database is where all information is stored. The decision of not having a local database for the client application is due to the fact that there is no need of having it since all data can be retrieved through the network passing from the server.

Communication between Components

This paragraph makes references to elements shown in the diagrams of section 2.1.

Data4Help is based on constant data exchange from the client application to the server. In order to allow this continuous communication, a socket connection is established between client and server. This allows for a continuous data stream. Therefore, Data4Help can receive data as soon as it is collected by the client application. This kind of connection is present also between Data4Help and the Third party Service Systems for them to receive new data as soon as it is produced as long as they are subscribed to it.

Data4Help Server offers REST APIs to both the client application and the Third Party Service System. The client application will use the REST APIs to interact with the server. *Third parties* will receive the documentation relative to the exposed APIs for them to communicate with Data4Help. By doing this, *Third parties* may develop their system, having in mind what operations they can do on the server. Moreover, there is no need of a particular communication protocol, as REST APIs are just basic HTTP calls exploiting the main operations: GET, POST, PUT, DELETE.

The communication between the client application and the Third Party System Service is done through HTTP, as already explained in the previous paragraph.

Track4Run Users

Track4Run has three kinds of users: *Spectators*, *Organizers* and *Participants*. These three all have different needs and interactions with the system, reason why they are kept separate. Each one has a different way to interact with the system:

- **Spectators** are the simplest kind of user. They only need a dedicated

GUI where they can watch a *Run*.

- **Organizers** require a more complex GUI to interact with the system, allowing for registration and *Run* management.
- **Participants** are Data4Help *Users*, who, after adding Track4Run to their *Services*, may enroll in a *Run*.

For these reasons, *Spectators* and *Organizers* do not need to be Data4Help *Users*: in fact, their data collection is not needed. By doing this, the system is kept simple by not requiring *Organizers* to register to Data4Help and *Spectators* not to register at all.

Data Interface

When it comes to sending data from a component to another component, then it is the receiving component that offers the interface to receive data. This choice is due to the fact that as soon as data is produced, it must be sent over to the receiving component, assuming it is always ready for data receipt. If it were the opposite - where the interface would be offered by the sending component - then the receiving component would have to constantly query the sending component for new data. This solution would have as a consequence several useless queries. On the other hand, all the calls to the data interface are useful.

Notifications

The pattern used to implement the notifications system is the opposite of the one discussed in the previous paragraph. In fact, it is the client application that constantly queries the server for new notifications. As soon as they are produced, the server will return them. This choice is driven by the fact that the client application is not always connected to the Internet, therefore, in an opposite scenario, all the calls from the server to the client application, when this last one is offline, would be useless as they will not have any answer. By implementing the suggest solution, all calls will be useful, even if many may still return nothing.

Third Party Services Interfaces

With the proposed design, there are only two different interfaces that need to be implemented and offered by the *Third party Services*. These are the Service Interface and the New Data Interface (see sections 2.2 and 2.5 for more details). *Third party Services* need to implement the two interfaces, which are analyzed in details:

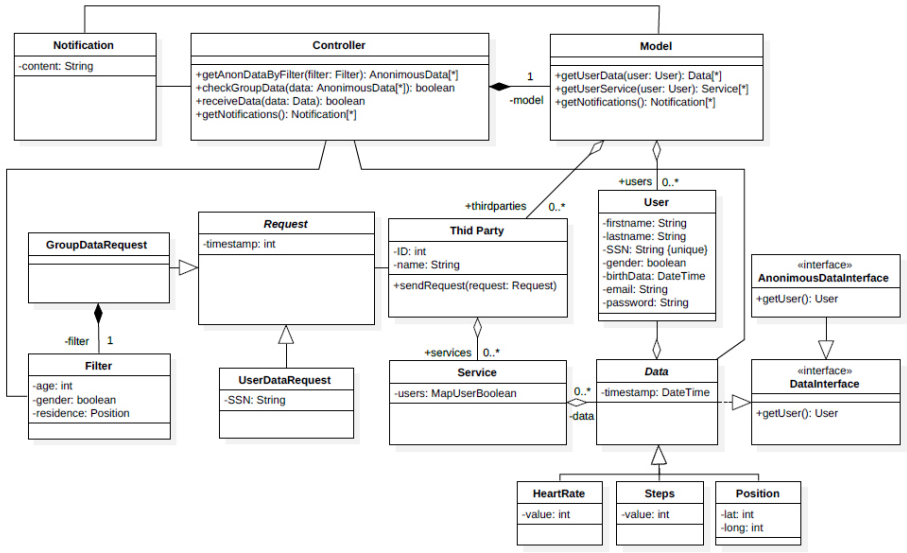
- **Service Interface** must respect the documentation about it provided by TrackMe in order to allow the User Application to retrieve the GUI offered by the *Service*. In particular, the GUI must be a website for the *User* to interact with. The User Application will call methods of the interface, therefore this must be *Service*-independent. A solution to this lies in the

executeCommand(String command) method, where the User Application will pass a string containing the command to be executed to the *Service* offering the Service Interface.

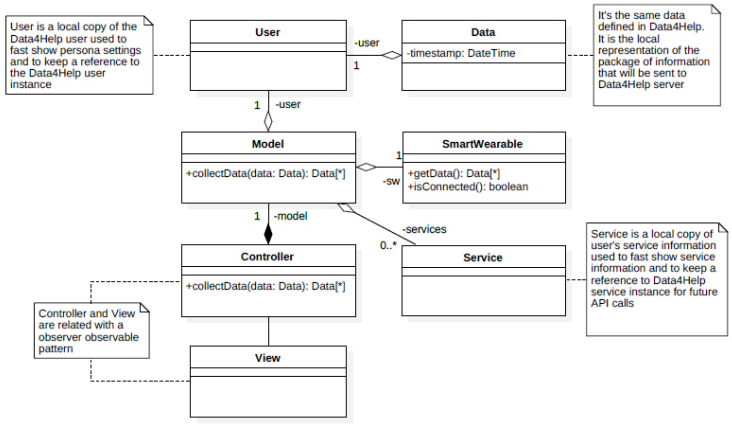
- **New Data Interface** must also respect the documentation about it provided by TrackMe. This interface is crucial for *Third party Services* to receive new data. In fact, when new data is collected, it is sent to the subscribed *Third party Services* right away. In order to have a *Service*-independent way of sending new data, a common interface must be shared among *Services*.

2.7 Other Design Decisions

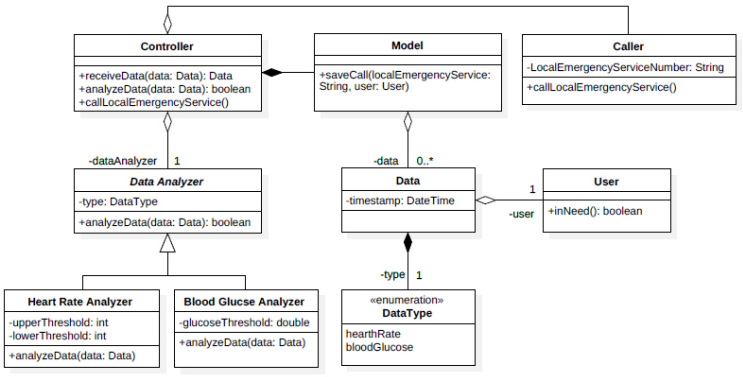
2.7.1 Class Diagrams



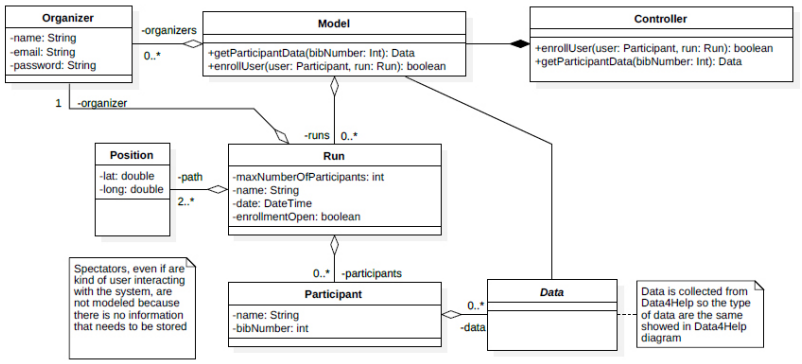
Data4Help Class Diagram



User Application Class Diagram



AutomatedSOS Class Diagram



Track4Run Class Diagram

2.7.2 Interfaces Use

Description - scrivere che stiamo mappando interfacce alle classi e che le classi si possono vedere nel diagramma di sopra

Data4Help

- Data4Help Controller
  - Add Notifications Interface
  - Data Interface
  - New Data Interface
- Data4Help Model
  - Database Interface
- ThirdParty
  - Request Data Interface

User Application

- User Application Controller
  - Data Interface
  - Get Notifications Interface
  - Registration and Login Interface
- User Application Service
  - Services Interface
  - Service Interface (AutomatedSOS)

  - Google Maps Interface
- SmartWearable
  - Data Interface

AutomatedSOS

- AutomatedSOS Controller
  - Caller Interface
  - Data Interface
  - AutomatedSOS Commands Interface
  - Local Emergency Services Interface
- AutomatedSOS Model
  - AutomatedSOS Database Interface
- DataAnalyzer
  - Data Interface

Track4Run

- Track4Run Controller
  - New Data Interface
  - Track4Run Commands Interface
  - Data Interface
- Track4Run Model
  - Database Interface

Chapter 3

User Interface Design

All relevant mockups of the *User* interface are included in section 3.1.1 of [3].

Interacting with the User Application

The following provides a description on how to move through interfaces when interacting with the User Application so as to give a clear idea of the navigation flows.

- *Data4Help Welcome Page* is shown when the User Application is opened.
- *Data4Help Welcome Page* redirects to Data4Help Login Page as soon as the User Application is loaded.
- *Data4Help Login Page* redirects to Data4Help Sign Up Page if the *User* taps on Sign Up, or redirects to Data4Help Menu if the *User* fills in the form and taps on Login.
- *Data4Help Sign Up Page*redirects to *Data4Help Login Page* once the *User* fills in the form and taps on Create Account (assuming inserted data is correct). If data is not correct, the interface does not change and an alert is shown.
- *Data4Help Menu* redirects to:
  - *User Services Page* if the *User* taps on My Services.
  - *Services Discovery Page* if the *User* taps on Discover.
  - *Data4Help Login Page* if the *User* taps on Logout.

  My Account, My Devices, Settings, Terms and Conditions redirect to specific pages that are not listed in [3] because they are simple and negligible interfaces.
- *User Services Page* redirects to *User Subscribed Service Page* when the *User* taps on the icon of a *Service*.

  - If the *Service* is AutomatedSOS and the monitoring was deactivated due to a previous alarm since data was detected as *Anomalous*, the interface shown is *AutomatesSOSPage*.

- *Services Discovery Page* redirects to *Add Service Page* when the *User* taps on the icon of a *Service*.
- *Organizer Registration Page* is shown when an individual connects to the *Organizers* dedicated website and wants to register as an *Organizer*.
- *New Run Creation Page* is shown when an *Organizer* asks to create a new *Run*.
- *Runs Discovery Page* is shown when a *User* taps on Track4Run in their application.
- *Spectators Run Discovery Page* is shown when an individual connects to the *Spectators* dedicated website. It redirects to *Watch Real Time Run Page* when the *Spectator* taps on the icon of a *Run*.

Chapter 4

Requirements Traceability

The following is a way to ensure that all requirements defined in section 3.2.5 of [3] are satisfied by the components defined in section 2.2 of this document. Each component is needed in order to fulfill one or more requirements, therefore all components defined have a specific purpose. Moreover, where applicable, references to sequence diagrams are included so as to provide a graphical representation on how components fulfill each requirement.

Data4Help

R_1 Unregistered individuals and companies must not be able to use Data4Help.

- Authentication (User App and Data4Help)
- Login Sequence Diagram

R_2 At sign up, *User* must provide: first name, last name, SSN, gender, date of birth, email and password.

- Authentication (User App and Data4Help)

R_3 At sign up, *Third party* must provide a company name.

- Authentication (Data4Help)

R_4 At sign up, *User* must accept *Terms and conditions*, including the *Privacy statement*.

- Authentication (User App and Data4Help)

R_5 At sign up, *Third party* must accept *Terms and conditions*.

- Authentication (Data4Help)

R_6 Identify a *User* by their identifier.

- Authentication (User App and Data4Help)

  - Login Sequence Diagram

R_7 Query the database for a *User* by their identifier.

  - Authentication (User App and Data4Help)

R_8 Receive *User Data*.

  - Authentication (User App and Data4Help)
  - Data Collector (User App)
  - Data Sender (User App)
  - Data Receiver

R_9 Validate *User Data*.

  - Data Receiver

R_{10} Authenticate *User Data*.

  - Data Receiver

R_{11} Store collected *User Data* in a database.

  - Data Receiver
  - Data Manager

R_{12} Retrieve specific *User Data* by database querying based on *User* identification.

  - Data Manager
  - User Data Request Sequence Diagram

R_{13} Receive *Third party* data access request.

  - Third Party Request Manager
  - Group Data Request Sequence Diagram

R_{14} Validate *Third party* data access request.

  - Third Party Request Manager

R_{15} Authenticate *Third party* data access request.

  - Third Party Request Manager

R_{16} Forward *User Data* access request to the specific *User*.

  - Third Party Request Manager
  - Notifications Manager
  - Services Manager (User App)

- User Data Request Sequence Diagram

R_{17} Receive *User* consent approval or denial.

- Services Manager (User App and Data4Help)
- Add New Service Sequence Diagram

R_{18} Check if a specific *User* gave consent to a specific *Service*.

- Third Party Request Manager
- Data Manager

R_{19} Send specific *User Data* to the requesting *Third party*.

- Third Party Request Manager
- Data Manager
- User Data Request Sequence Diagram

R_{20} Not send specific *User Data* to the requesting *Third party* if the specific *User* denied consent.

- Third Party Request Manager
- Data Manager
- User Data Request Sequence Diagram

R_{21} *Third party* must be able to set specific constraints to define a group of *Users*: age, gender, residence.

- Third Party Request Manager

R_{22} Check how many *Users* requested *Group Data* refers to.

- Third Party Request Manager
- Group Data Request Sequence Diagram

R_{23} Properly anonymize *Group Data*.

- Third Party Request Manager
- Group Data Request Sequence Diagram

R_{24} Send *Group Data* to the requesting *Third party*.

- Third Party Request Manager
- Group Data Request Sequence Diagram

R_{25} Not send *Group Data* if the group it refers to is made up of less than 1000 *Users*.

- Third Party Request Manager

- Group Data Request Sequence Diagram

R_{26} Receive *Third party* subscription request.

- Third Party Request Manager

R_{27} Validate *Third party* subscription request.

- Third Party Request Manager

R_{28} Authenticate *Third party* subscription request.

- Third Party Request Manager

R_{29} Automatically send new data to subscribed authorized *Third parties* as soon as they are produced.

- Data Receiver
- Data Manager
- Services Manager (Data4Help)
- Third Party Request Manager

R_{30} Allow *Users* to subscribe to *Services*.

- Services Manager (Data4Help)
- Data Manager
- Add New Service Sequence Diagram

R_{31} Allow *Users* to unsubscribe from *Services*.

- Services Manager (Data4Help)
- Data Manager

R_{32} Send a specific *User* all their data stored, if requested by them.

- Services Manager (Data4Help)
- Data Manager

R_{33} Delete a *User* specific data, if requested by them.

- Services Manager (Data4Help)
- Data Manager

R_{34} Allow *Users* to request all their data stored by TrackMe at any time.

- Services Manager (User App and Data4Help)

R_{35} Allow *Users* to request the deletion of all their data stored by TrackMe at any time.

- Services Manager (User App and Data4Help)

AutomatedSOS

R_{36} Receive *User Data* from Data4Help.

- Data Receiver (AutomatedSOS)
- User In Need Assistance Sequence Diagram

R_{37} Compare *User Data* against certain thresholds.

- Data Analyzer
- User In Need Assistance Sequence Diagram

R_{38} Call local emergency services providing necessary *User Data* of *User in need*.

- User Monitor
- Local Emergency Services Caller
- User In Need Assistance Sequence Diagram

R_{39} *User* must be able to reactivate AutomatedSOS monitoring.

- Service Controller (User App)
- Service Interface Provider
- User Monitor

Track4Run

R_{40} Receive *User Data* from Data4Help.

- Data Receiver

R_{41} At sign up, *Organizers* must provide: first name, last name, email and password.

- Authentication

R_{42} At sign up, *Organizers* must accept *Terms and conditions*.

- Authentication

R_{43} Allow *Organizers* to create a *Run*, defining: name, path, date, maximum number of *Participants* and enrollment closure date.

- Run Manager
- Run Organization Sequence Diagram

R_{44} Provide *Users* enrollment for an existing *Run*.

- Run Manager

  - Service Interface Provider
  - Service Controller (User App)
  - Participants Enrollment Sequence Diagram

R_{45} Prevent a *User* from enrolling in a *Run* if the maximum number of *Participants* was already reached.

  - Run Manager
  - Participants Enrollment Sequence Diagram

R_{46} Prevent a *User* from enrolling in a *Run* if it already started or finished.

  - Run Manager
  - Participants Enrollment Sequence Diagram

R_{47} Prevent a *User* from enrolling in a *Run* if enrollment is closed.

  - Run Manager
  - Participants Enrollment Sequence Diagram

R_{48} Show a *Run* by displaying the position of *Participants* on a map.

  - Run Manager

R_{49} Identify a *Run* by its identifier.

  - Run Manager
  - Participants Enrollment Sequence Diagram

R_{50} Query the database for a *Run* given its identifier.

  - Run Manager
  - Participants Enrollment Sequence Diagram

Chapter 5

Implementation, Integration and Test Plan

Inserire quì una introduzione

5.1 Components List

The following list includes all the components that must be implemented. The list is product based so components are separated in three parts: *Data4Help*, *AutomatedSOS*, *Track4Run*.

Data4Help

- Data4Help App
  - Service controller
  - Authentication
  - Data sender
  - Data collector
  - Services manager
- Data4Help Server
  - Request manager
  - Notifications manager
  - Services manager
  - Data manager
  - Data receiver
  - Authentication

- Data4help Database
- Data4help Third parties Website

Track4Run

- Track4Run Organizers website
- Track4Run Spectators website
- Track4Run Server
  - Run manager
  - Service interface provider (website)
  - Data receiver
  - Authentication
- Track4Run Database

AutomatedSOS

- AutomatedSOS Server
  - Service interface provider (website)
  - Data analyzer
  - User monitor
  - Data receiver
  - Local emergency services caller
- AutomatedSOS Database

5.2 Components dependencies

The following list of components reflects the logical or architectural dependencies between the components listed in previous list.

Data4Help

- Data4Help App: Service controller
  - No components dependencies
- Data4Help App: Authentication
  - Data4Help Server: Authentication
- Data4Help App: Data sender

  - Data4Help Server: Data receiver
- Data4Help App: Data collector
  - No components dependencies
- Data4Help App: Services manager
  - Data4Help Server: Notifications manager
  - Data4Help Server: Services manager
- Data4Help Server: Request manager
  - Data4Help Server: Notifications manager
  - Data4Help Server: Data manager
- Data4Help Server: Notifications manager
  - Data4Help Server: Data4Help Database
- Data4Help Server: Services manager
  - Data4Help Server: Data manager
- Data4Help Server: Data manager
  - Data4Help Database
- Data4Help Server: Data receiver
  - Data4Help Server: Data manager
- Data4Help Server: Authentication
  - Data4Help Database
- Data4help Database
  - No components dependencies

AutomatedSOS It depends entirely on *Data4Help*. The following listed dependencies are relative only to the ones between *AutomatedSOS* components.

- AutomatedSOS Server: Service interface provider(website)
  - AutomatedSOS Server: User monitor
- AutomatedSOS Server: Data analyzer
  - AutomatedSOS Server: Data receiver
- AutomatedSOS Server: User monitor

  - AutomatedSOS Database
  - Local emergency services caller
- AutomatedSOS Server: Data receiver
  - AutomatedSOS Server: Data analyzer
- AutomatedSOS Server: Local emergency services caller
  - No components dependencies
- AutomatedSOS Database
  - No components dependencies

Track4Run It depends entirely on *Data4Help*. The following listed dependencies are relative only to the ones between *Track4Run* components.

- Track4Run Organizers website
  - Track4Run Server: Authentication
  - Track4Run Server: Run manager
- Track4Run Spectators website
  - Track4Run Server: Run manager
- Track4Run Server: Run manager
  - Track4Run Database
- Track4Run Server: Service interface provider (website)
  - Track4Run Server: Run manager
- Track4Run Server: Data receiver
  - Track4Run Server: Run manager
- Track4Run Server: Authentication
  - Track4Run Database
- Track4run Database
  - No components dependencies

5.3 Implementation Plan

All the components shown in Inserisci reference have well specified interfaces so the implementation of them can proceed in parallel without any particular implementation dependency. Obviously this means that at the end of the implementation will be needed an important effort in integrating all the components.

In short words: the implementation can proceed completally in parallel in developing all the components, only team size or costs constraints may influence in the plan moving some activities before/after others.

Due to the fact that we don't know the team size neither eventual cost constraints, no Gantt Chart is included.

5.4 Integration Plan

The following list, divided by product, shows which integrations must be performed. To declare a product completed, all relative integrations must be performed. All services (included AutomatedSOS and Track4Run) depend on Data4Help so in these cases, in addiction to complete all integrations as stated before, another fundamental condition is that Data4Help is completed and fully working [*].

The integrations to be done are shown in a plain list because the order of integration is not important

Data4Help Server

- Integration between Data4Help Server: Request manager and Data4Help Server: Data manager
- Integration between Data4Help Server: Notifications manager and Data4Help Server: Data4Help Database
- Integration between Data4Help Server: Services manager and Data4Help Server: Data manager
- Integration between Data4Help Server: Data manager and Data4Help Database
- Integration between Data4Help Server: Data receiver and Data4Help Server: Data manager
- Integration between Data4Help Server: Authentication and Data4Help Database

Data4Help App

- Integration between Data4Help App: Authentication and Data4Help Server: Authentication
- Integration between Data4Help App: Data sender and Data4Help Server: Data receiver
- Integration between Data4Help App: Services manager and Data4Help Server: Notifications manager
- Integration between Data4Help App: Services manager and Data4Help Server: Services manager

AutomatedSOS

- Integration between AutomatedSOS Server: Service interface provider (website) and AutomatedSOS Server: User monitor
- Integration between AutomatedSOS Server: Data analyzer and AutomatedSOS Server: Data receiver
- Integration between AutomatedSOS Server: User monitor and AutomatedSOS Database
- Integration between AutomatedSOS Server: User monitor and Local emergency services caller
- Integration between AutomatedSOS Server: Data receiver and AutomatedSOS Server: Data analyzer
- *Integration between AutomatedSOS Server: Data receiver and Data4Help

Track4Run

- Integration between Track4Run Organizers website and Track4Run Server: Authentication
- Integration between Track4Run Organizers website and Track4Run Server: Run manager
- Integration between Track4Run Spectators website and Track4Run Server: Run manager
- Integration between Track4Run Server: Run manager and Track4Run Database
- Integration between Track4Run Server: Service interface provider (website) and Track4Run Server: Run manager
- Integration between Track4Run Server: Data receiver and Track4Run Server: Run manager

- Integration between Track4Run Server: Authentication and Track4Run Database
- *Integration between Track4Run Server: Data receiver and Data4Help

5.5 Testing Plan

5.5.1 Unit Testing

It must be performed during the development of each component and finalized at the end of each component. It is recommended to create enough tests to reach a coverage of 90% using some tools like SonarQube or JUnit verificare quest'ultima parte di sonarqube e junit.

5.5.2 Integration testing

Must be performed on each case listed in the previous section. To execute integration testing all the components to be integrated must be completed. The testing can be executed in parallel with respect to main implementation: this means that during the integration testing, the implementation of new components can go on.

Chapter 6

Effort Spent

Gargano Jacopo Pio Total hours of work: xxh

- 3h DD review homework
- 2h Purpose, Scope
- 3h Overview and Component diagrams
- 2h Deployment diagrams
- 2h Component diagram revision
- 2h Requirements traceability
- 2h Component interfaces
- 2h Class diagrams
- 2h Overview and Component diagrams explanation
- 2h Overview and Component revision
- 3h Sequence diagrams explanation
- 1h Requirements traceability revision
- 2h Component interfaces completion
- 1h UML revision
- 2h Architectural Styles and Patterns
- 1h Meeting with Professor

Giannetti Cristian Total hours of work: xxh

- 3h DD review homework
- 1h Creating subfiles structure in Latex
- 3h Overview, Component Diagrams
- 2h Deployment diagrams
- 2h Component diagram revision
- 2h Requirements traceability
- 2h Component diagrams
- 5h Component and Overview diagrams
- 2h Diagrams review
- 2h Component and deployment diagrams
- 2h Diagrams review
- 4h Sequence diagrams
- 1h Requirements traceability revision
- 2h Component interfaces completion
- 1h UML revision
- 2h Interface use and Effort spent
- 1h Meeting with Professor

Haag Federico Total hours of work: xxh

- 3h DD review homework
- 3h Overview, Component diagrams
- 2h Deployment Diagram
- 2h Sketch sequence diagrams
- 1h Review of sketched Sequence diagrams
- 3h Component and Overview revision
- 4h UML Class diagram
- 2h User interface design
- 1h General revision

- 2h Implementation Plan
- 2h Integration and Testing Plan
- 1h UML revision
- 1h Meeting with Professor

Chapter 7

References

1 E. Di Nitto. *Lecture Slides*. Politecnico di Milano.

2 E. Di Nitto. *Mandatory Project Assignment AY 2018-2019*. Politecnico di Milano.

3 J. Gargano, C. Giannetti, F. Haag. *Requirement Analysis and Specification Document*. Politecnico di Milano.