POLITECNICO
MILANO 1863

# TrackMe

Design Document

Luca Conterio - 920261
Ibrahim El Shemy - 920174

A.Y. 2018/2019 - Prof. Di Nitto Elisabetta

# Contents

# 1 Introduction

## 1.1 Purpose

This document represents the `Design Document` (DD) for TrackMe software and contains a functional description of the system. We provide an overall guidance to the architecture of the system and it is therefore addressed to the software development team.

## 1.2 Scope

`TrackMe` is a company that wants to develop a software-based service allowing third parties to monitor the location and health status of individuals. Hence, the system has to be composed by two specific services:

- **Data4Help**

  This service supports the registration of individuals who agree that TrackMe acquires their data (through electronic devices such as smartwatches).

- **AutomatedSOS**

  This service is oriented to elderly people: monitoring their health status parameters, the system can send to the location of the customer an ambulance when some parameters are below certain thresholds, guaranteeing a reaction time of less than 5 seconds from the time the parameters get lower than the threshold.

## 1.3 Definitions, Acronyms and Abbreviations

### 1.3.1 Definitions

- `User Device`: any compatible device with the TrackMe application, either smartphone or smartwatch.

- `Personal Information`: the collection of infromation provided by a User during the registration process. It Incudes legal information, such as Social Security Number, name, surname, birth date, address, e-mail address, mobile number. Sometimes in the document this expression is abbreviated by `Data`, that also refers to clinical data.

- `Assistance Request`: it is a request formulated by the system and sent to the Ambulance Dispatching System in order to guarantee users the necessary assistance.
- `Individual Access Request`: sometimes referred only as `individual request` or `access request`. It refers to a singular request sent by a Third Party to a User whenever it wants to have access to User's data. It needs the User's approval.
- `Sampling Search`: it refers to a search conducted by a Third Party according to some filters, returning anonymized results for a group of people. In this document it is often called only `sampling`.
- `Mobile App`: an application that can be run by mobile devices, both smartphones and smartwatches.

### 1.3.2 Acronyms

- **DD**: Design Document
- **RASD**: Requirments Analysis and Specification
- **ERP**: Enterprise Resource Planning
- **DMZ**: Demilitarized Zone
- **RAPS**: Reliable Array of Partitioned Service
- **API**: Application Programming Interface
- **DBMS**: Database Management System
- **SMS**: Short Message Service

### 1.3.3 Abbreviations

- [Gn]: n-goal.
- [Rn]: n-requirment.
- [Rx] - [Ry]: from x-th requirement to y-th requirement.
- App: application.

## 1.4 Document Structure

- **1 Introduction**

  This section introduces the Design Document. It explains the Purpose, the Scope and the framework of the document.

- **2 Architectural Design**

  This section is focused on the main components used for this system and the relationship between them, providing information about their deployment and how they operate. It also focuses on the architectural styles and the design patterns adopted for designing the system.

- **3 User Interface Design**

  This section provides an overview on how the User Interface will look like and furthermore gives a more detailed extension using UX and BCE diagrams.

- **4 Requirments Traceability**

  This section explains how the requirements defined in the RASD map to the design elements defined in this document.

- **5 Implementation, Integration and Test Plan**

  In this section we identify the order in which we plan to implement the subcomponents of the system and the order in which we plan to integrate and test them.

# 2 Architectural Design

## 2.1 High Level Components and their Interactions

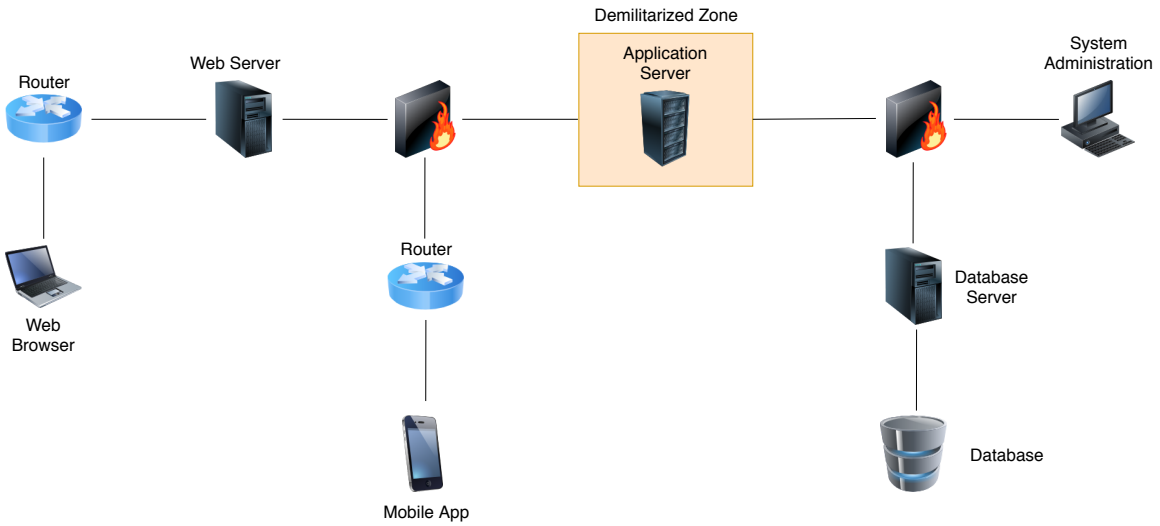



Figure 1: High Level System Structure

The architecture of our system can be streamlined into 3 logic layers:

- **Presentation Layer**
  This layer can be divided into the Client tier (that includes the Web App and the Mobile App) and the Web tier. Third parties can access the system's functionalities and Users' information to which they are subscribed to through the Web Server.

- **Application Layer**
  Users logged into the Mobile Application can access the system's functionalities and communicate with the Application Server, located in a Demilitarized Zone (DMZ), directly.

- **Data Layer**
  On top of that, we have the Database Server and the Database itself, where data about registered Users and Third Parties are stored and managed.

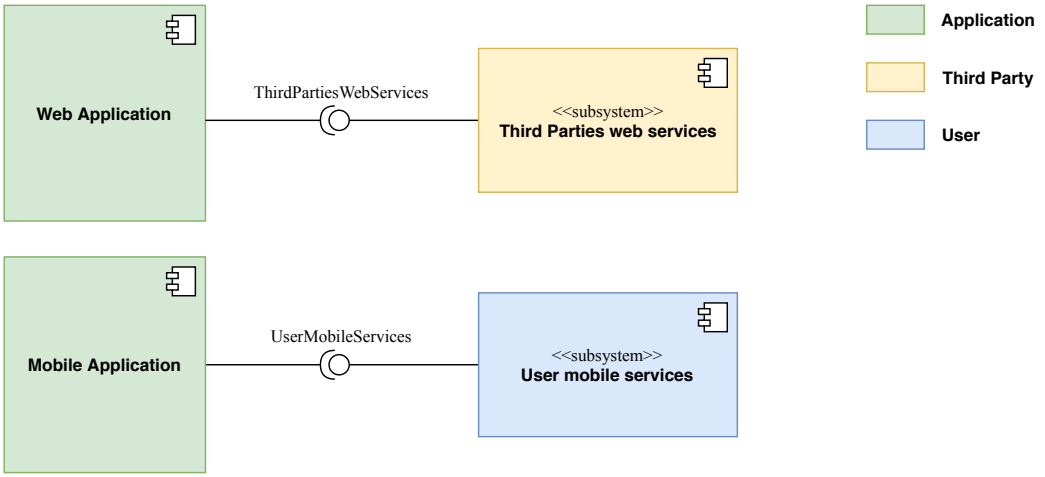
System Administration is implemented through a third-party ERP solution, hence we do not provide information about its implementation.

## 2.2 Component View

### 2.2.1 High Level Component Diagram

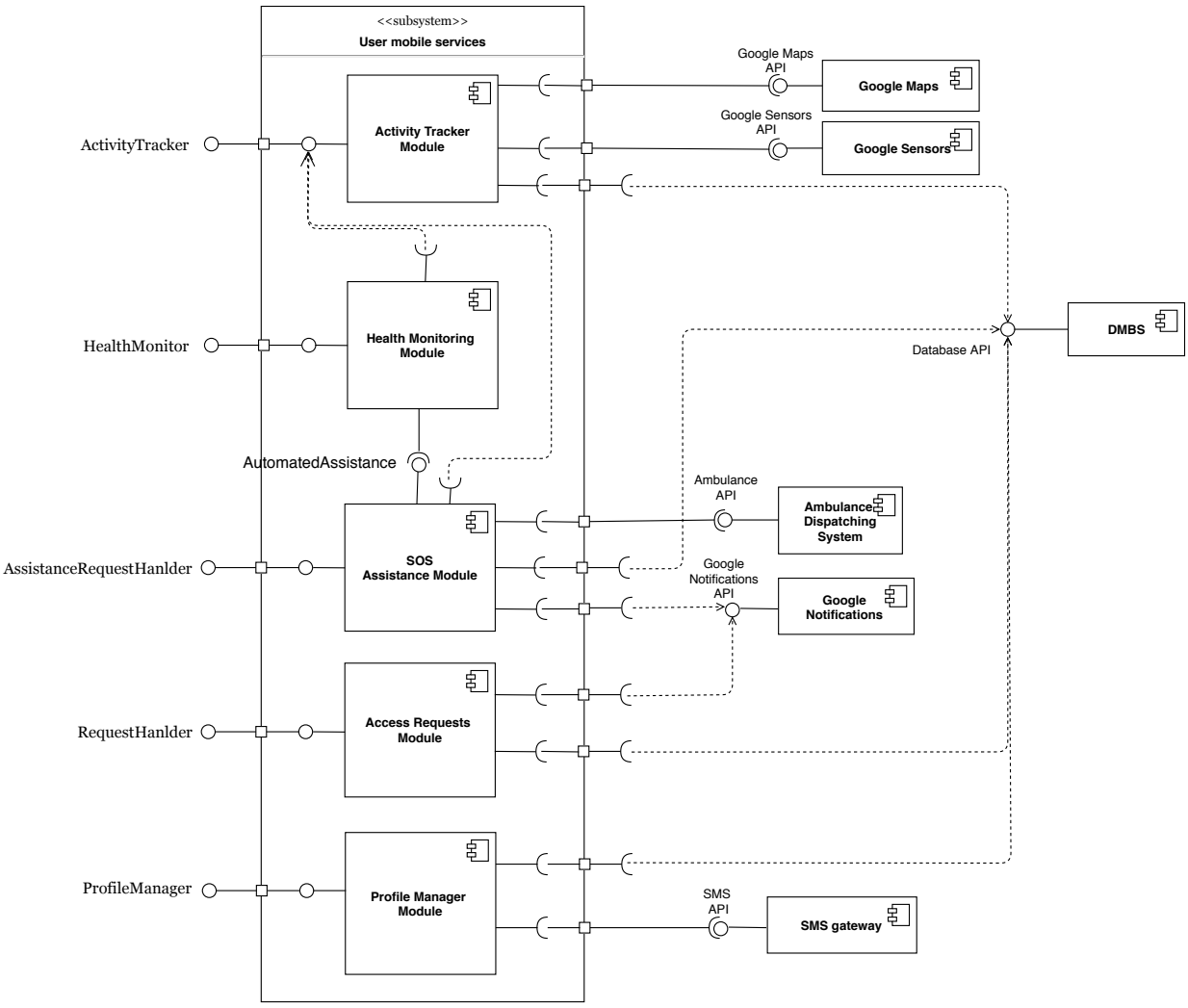
The following diagrams illustrate the system components and the interfaces through which they interact to fullfil their functionalities. A distinction can be made between Client side and Server side:

- The Client side is composed by two components, `Web Application` and `Mobile Application`, referring to the following services: `ThirdPartiesWebServices` and `UserMobileServices`.
- The Server side is composed of two main components, `Third Parties web services` that will provide functionalities aiming to fullfill third parties needs, such as sending requests to Users, `User mobile services` that will provide an interface to User in order to manage his own profile, check information about his health status, accept/reject requests, etc.



### 2.2.2 User Mobile Services Projection

User Mobile Services subsystem is composed by five components: `Activity Tracker Module`, `Health Monitoring Module`, `SOS Assistance Module`, `Access Requests Module` and `Profile Manager Module`. These components provide to the User Mobile Application the following interfaces: `ActivityTracker`, `HealthMonitor`, `AssistanceRequestHandler`, `RequestHandler` and `ProfileManager`. The components need also to communicate with `Google Maps`, `Google Sensors`, `Google Notifications`, `Ambulance Dispatching System`, `SMS gateway` and the `DBMS` to work properly and guarantee a better User experience with the Application.



For what concerns smartwatches, the server will not distinguish between messages and requests received directly by the smartwatch device (e.g. if it is equipped with GPS) and those retrieved by the sensors and forwarded to the server thanks to the connectivity of a smartphone. The User Mobile Services subsystem will handle all the mentioned situations.
The only difference is that the smartwatch interface will not provide the user the possibility to manage his/her profile, since it would not be user friendly nor comfortable.

**Module Functionalities**

- `Activity Tracker Module`: this module is devoted to everything that concerns data retrieved by user's devices, such as location, steps, covered distance, heartbeat, blood pressure and sleep.

- `Health Monitoring Module`: this component persists in an idle state as long as a user is not registered to AutomatedSOS service.
  Otherwise, it interpellates directly the Activity Tracker Module, through the ActivityTracker interface in order to check the user's health status. Moreover, thanks to the AutomatedAssistance interface it can ask for an automated assistance request.

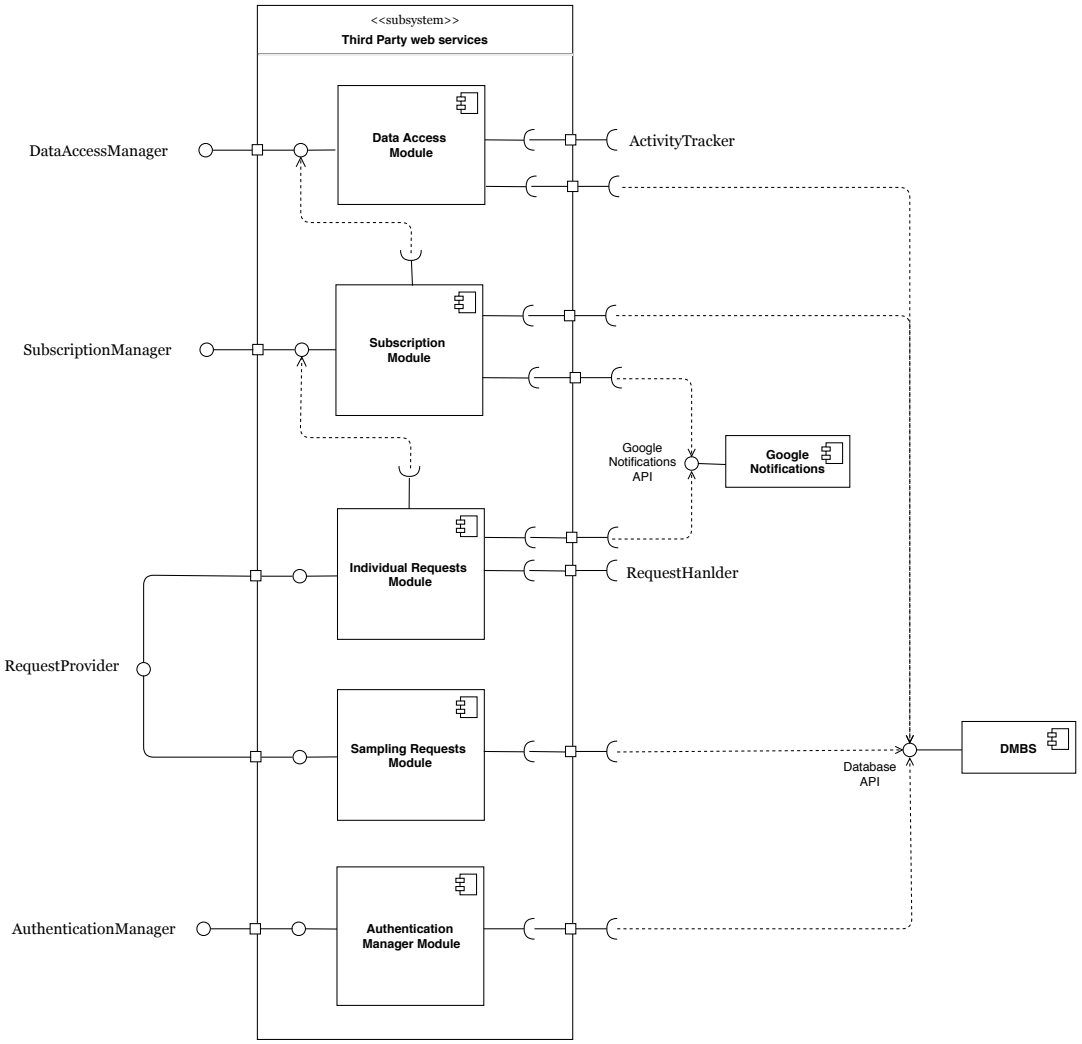
- `SOS Assistance Module`: in this module all functions dealing with assistance requests are implemented. It directy handles on-demand assistance requests and automated assistance requests performed by the Health Monitoring Module through the AutomatedAssistance interface.

- `Access Requests Module`: this component handles the access requests coming from some third parties, that want to have access to the user's data, giving the user the possibility to accept or reject them. It also manages the updates performed by the user to the permissions given to the third parties.

- `Profile Manager Module`: it provides all the functions concerning the user authentication and the update of profile informations.

### 2.2.3 Third Party Web Services Projection

Third Party Web Services subsystem is composed by five components: `Data Access Module`, `Subscription Module`, `Individual Requests Module`, `Sampling Requests Module` and `Authentication Manager Module`. These components provide to the Web Application the following interfaces: `DataAccessManager`, `SubscriptionManager`, `RequestProvider` and `AuthenticationManager`. The components need also to communicate with `Google Notifications` and the `DBMS` to work properly.



**Module Functionalities**

- `Data Access Module`: it deals with accessing to user's data, in case of the specific third party has access to him/her.
- `Subscription Module`: this component is responsible for subscribing a third party to a specific user, after the access request has been accepted. If the operation terminates successfully it checks the existence of new data retrieved by the Data Access Module and notifies the involved third party, thanks to the Notifications API.
- `Individual Requests Module`: it provides the third party with an interface that allows it to send data access request to a specific user. If a request is accepted, it also gives the possibility to the third party to subscribe to user's new data.
- `Sampling Requests Module`: similar to the Access Requests Module, provides an interface to allow the third party retrieving anonymized information of a group of users.
- `Authentication Manager Module`: this module provides all functionalities concerning with the registration and login of a third party to the system.

## 2.3 Deployment View

For the deployment of the system, we opted for a 4-tier architecture composed as follows:

- **Tier 1**
  This tier is composed by the Client (implemented as a Thin Client), that includes the Web App (run by a web browser) and the Mobile Application (run by smartphones).

- **Tier 2**
  This tier is composed by the Web Server, whose main functionality is to store, process static content and deliver web pages to the Clients. For this reason, we opted for NGINX, that guarantees better perfomances for static content processing. It can be also configured as a load balancer, in order to better handle multiple connections.

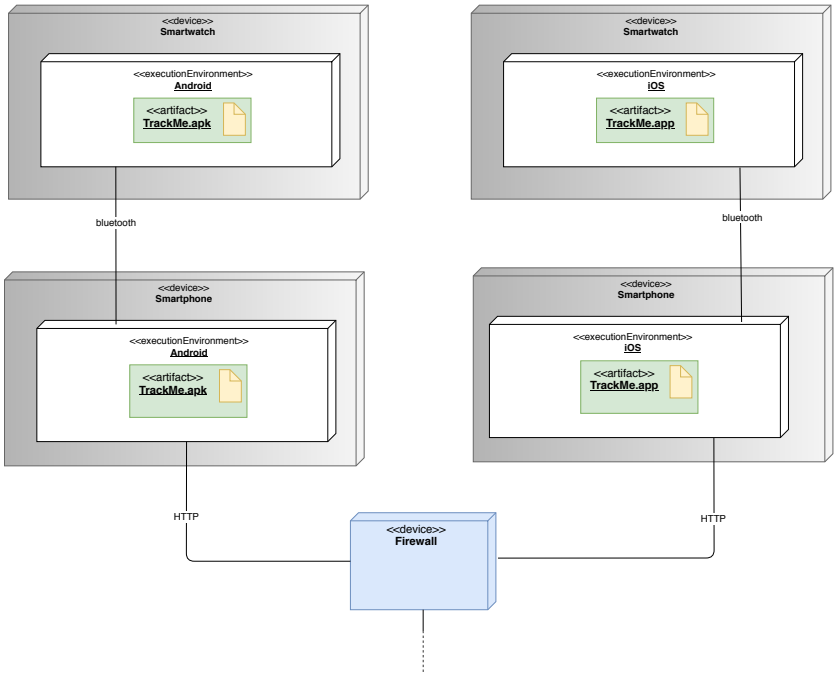



Figure 2: Deployment View for Mobile Application (Tier 1).

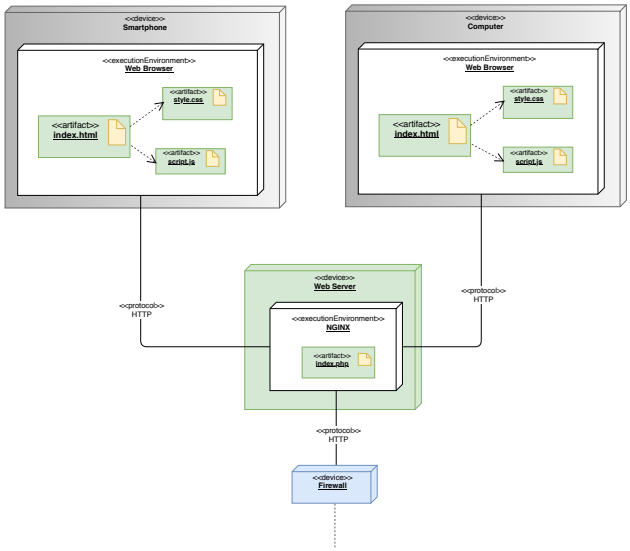



Figure 3: Deployment View for Web Browser clients and Web Server (Tier 1 and 2).

- **Tier 3**
  This tier corresponds to the Application Server, that provides both facilities to create web applications and a server environment to run them. We opted for WildFly (v. 12.0.0 or recent versions - previously JBoss) since it fully implements all Java EE specifications.

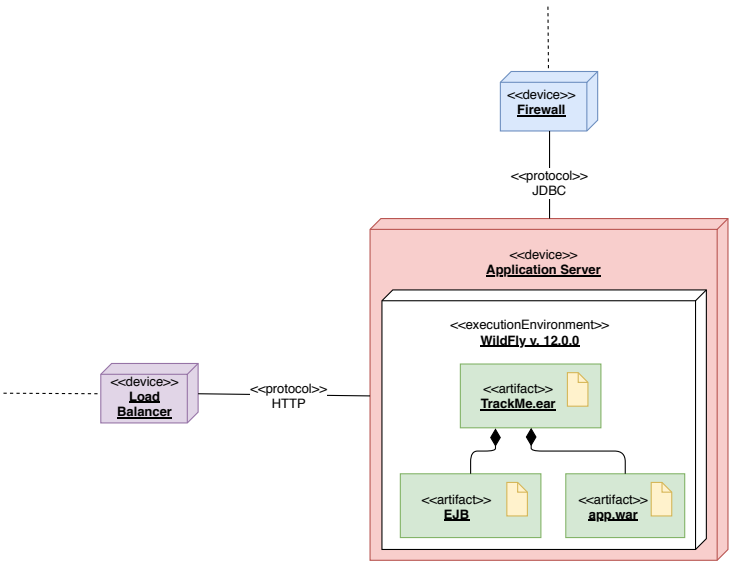



Figure 4: Deployment View for Application Server (Tier 3).

- **Tier 4**
  This tier corresponds to the Database Server on which the DBMS is running. We opted for MySQL (v. 8.0.12) since it is one of the most secure and reliable database management system used in popular web applications and guarantees scalability and high performance.

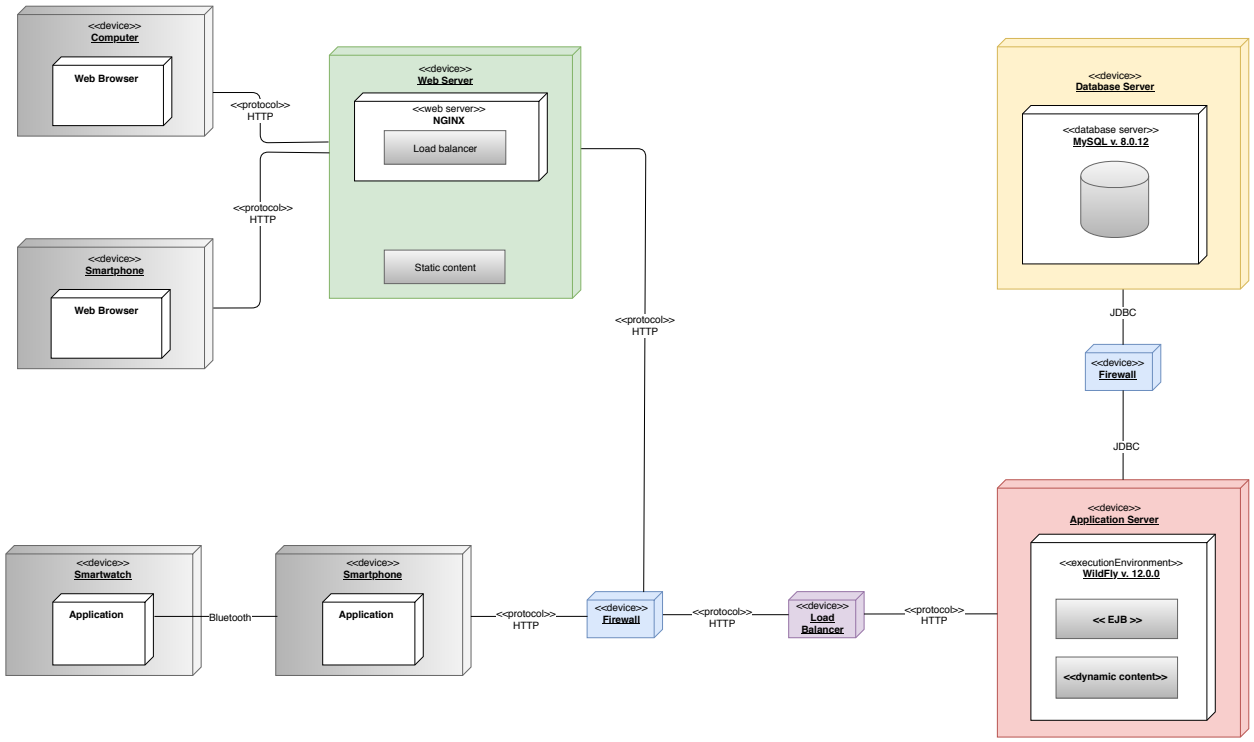



Figure 5: Deployment View Diagram (entire system).

## 2.4 Component Interfaces

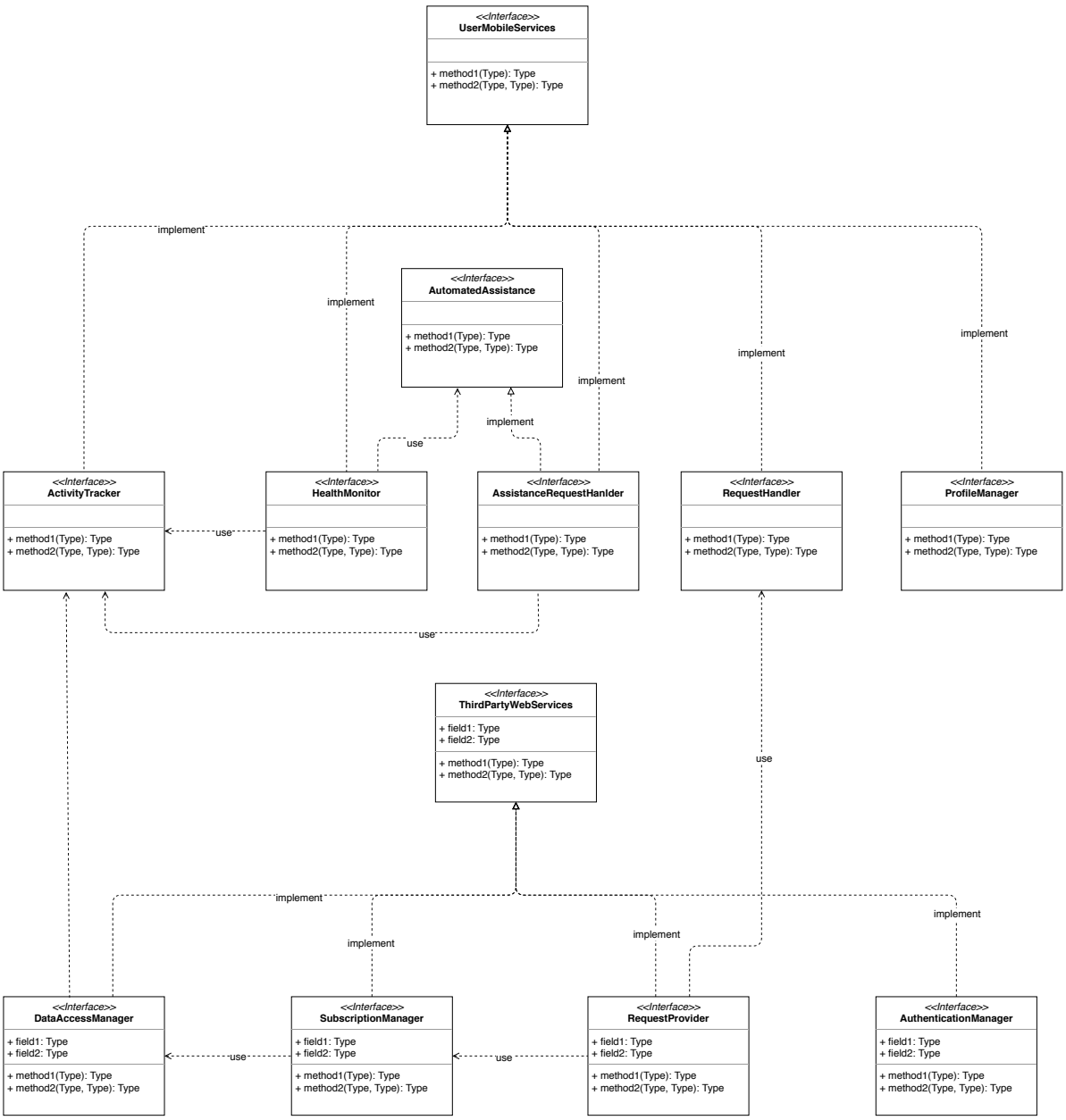



Figure 6: Component interfaces and their interactions.

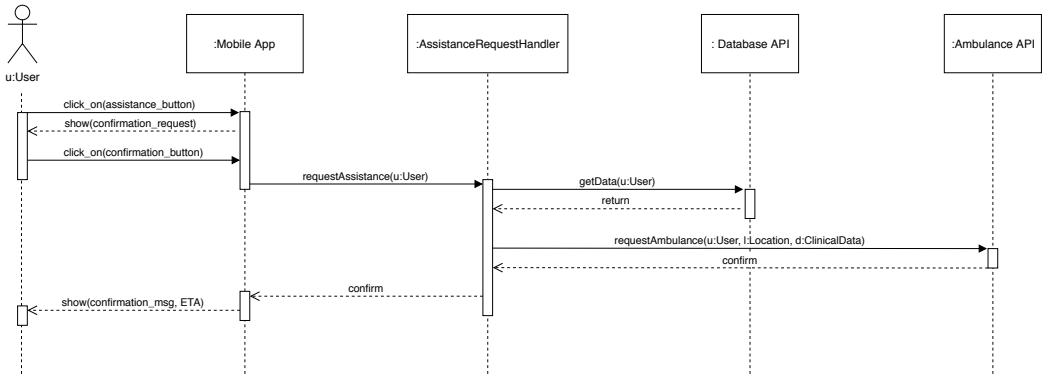
### 2.4.1 External Interfaces

TrackMe will make use of some Application Programming Interfaces to simplify the implementation, since these components are largely used and compatible with the majority of the devices currently on the market:

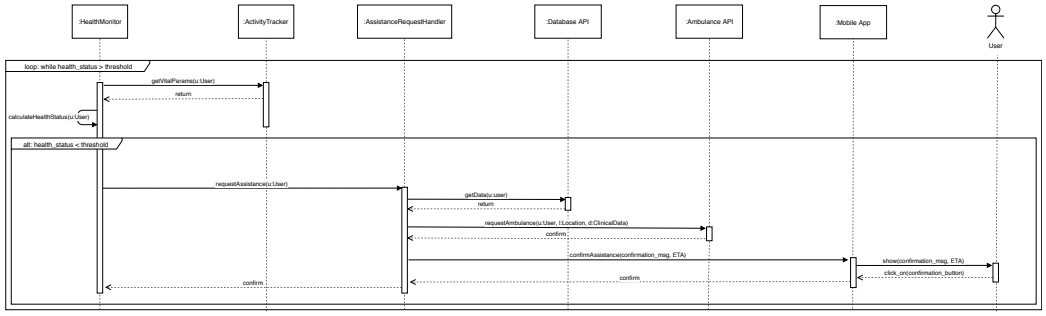
- `Google Maps API` to have a visual representation of user location and to get it in critical moments, such as AutomatedSOS requests to the Ambulance Dispatching System.
- `Google Fit Sensors API` to read raw sensor data in real time. It allows listing the available sensors on the device on which te application is running and registering a listener on specific sensors to retrieve data automatically. —¿ CoreMotion e HealthKit per iOS.
- `Google Notifications API`: this lets applications send information to a user even if the app is idle or in the background. —¿ e per iOS ?
- `Ambulance Dispatching System API` to perform assistance requests. Actually it could be an automatic phone call to the ambulance contact center, during which an algorithm communicates all relevant data to the interlocutor (e.g. name, age, location etc.).
- `Database API` to support mutiple database servers easily, to provide a structured interface for the dynamic construction of queries and to enforce security checks.
- `SMS API` to enable code to send short messages via a SMS Gateway.

## 2.5 Runtime View

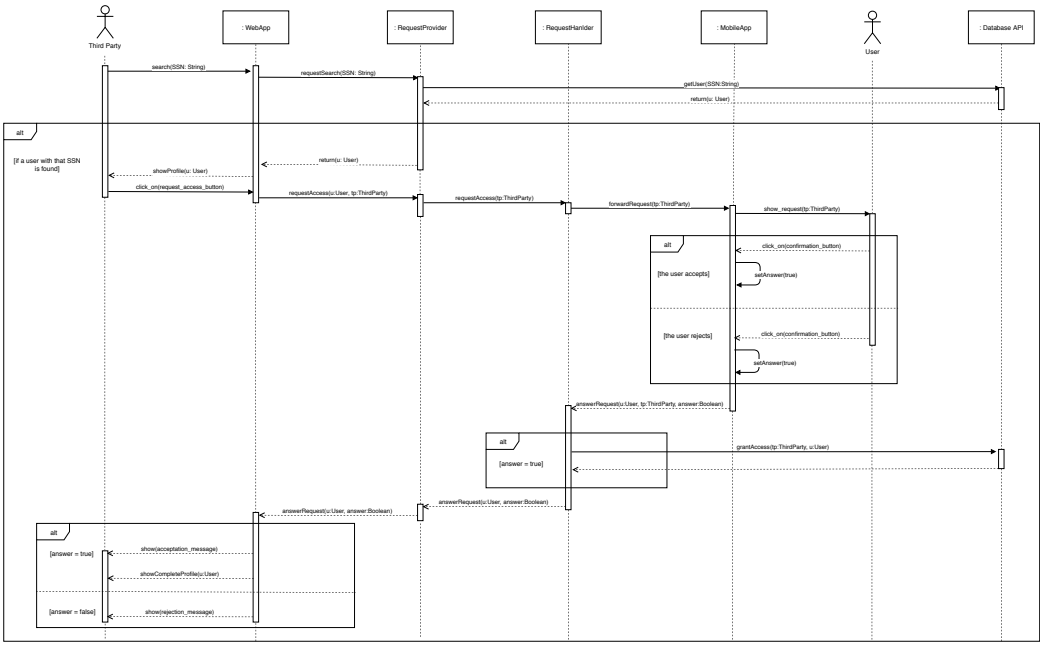
### 2.5.1 On-Demand Assistance Request



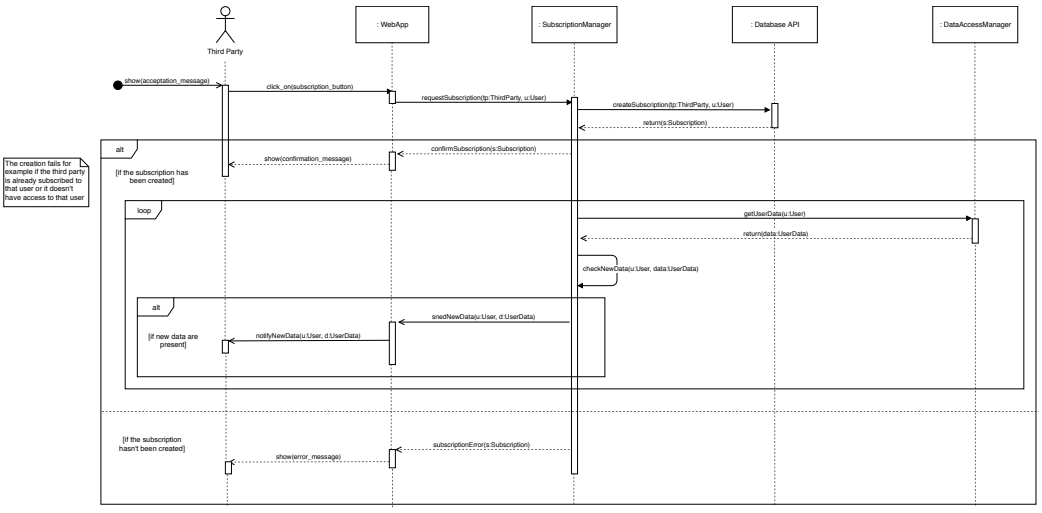

### 2.5.2 Automated Assistance Request



### 2.5.3 Data Access Request




### 2.5.4 Subscription to User's Data



## 2.6 Architectural Styles and Design Patterns

### 2.6.1 Design Patterns

In order to better formalize our architecture and make it as flexible as possible and speed up the devolpment process of our system we used different design patterns:

- **Model View Controller**
  The majority of Mobile and Web applications rely on this pattern. In fact, these applications retrieve data from a data collector (generally a Database) and update the User Interface, according to the input provided and the validity of the requested operation is checked by the Controller.
  Thus, the main key of this pattern is to create a separation between the User Interface (View), the data (Model) and the response and validity checking of the User's input (Controller).

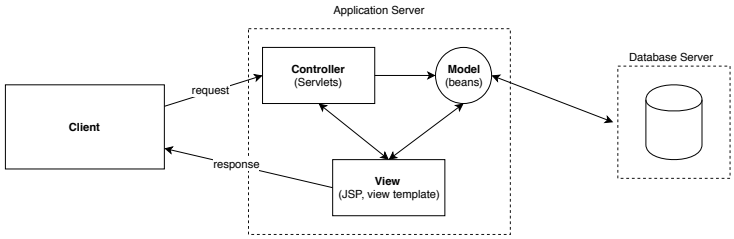



Figure 7: Distributed MVC in the context of Java EE.

- **Observer and Observable**
  In this pattern, there are many observers (objects) which are observing a particular subject (observable object). Observers are basically interested and want to be notified when there is a change made inside that subject. So, they register themselves to that subject. When they lose interest in the subject they simply unregister from the subject. In our system, the intent of this pattern is to let the User execute some query through the UI and after searching the Database, the result is reflected back in the UI. In most of the cases we segregate the UI with the Database. If a change occurs in the database, the UI should be notified so that it can update its display according to the change. When a change is occurring an event is notified and the changes are applied and updated for the User.
  When referring to a distributed system, this pattern is called Pub-Sub (**Publish-Subscribe**).

- **Visitor Pattern**
  This pattern's main purpose is to abstract functionalities that can be applied to an aggregate hierarchy of 'element' objects. In particular, we adopted this pattern in order to handle events that notify changes to the User Interface (View component). This way, we create a double dispatching mechanism, and the Controller is able to recognize the event type and associate correctly the requested operation.

- **Façade Pattern**
  This design pattern supports loose coupling. In fact we emphasize the abstraction and hide complex details by exposing a single interface instead of multiple ones. According to our system we expose to the client `ThirdPartyWebServices` and `UserMobileServices` that are in turn extended respectively by all the interfaces provided by the subsystems Third Party Web Services, User Mobile Services and User Smartwatch Services (see section 3, Component View).

- **Proxy Pattern**
  It provides a surrogate for another object to control access to it. In our system architecture it can be useful to interface the Application Server with the Database Server. For example, whenever a user or a third party requests some data that is unchanged, the proxy can answer the query without involving access to the database, providing a better response time. In our case, this is useful to improve service response time, very important in critical services that involve users' health.

### 2.6.2 RAPS Architecture

As mentioned briefly in the RASD (sections 3.9.2, 3.9.5), our system has to rely on a RAPS (textttReliable Array of Partitioned Service) architecture, in order to prevent unavailability of some functionalities in case of breakdowns. This architecture consists in a partitioned and redundant structure: servers are cloned to achieve this objective. More precisely, multiple services are divided on different machines and each machine can access in turn to a copy of the stored data. Such architecture guarantees better availability and scalability and provides a high rate of maintainability: in case of breakdowns, it is sufficient to work on the damaged machine, without interfering with the other machines' tasks, while the specific service can still be perfomed. In addition, if the system is willing to expand some services, it is sufficient invest on the specific partition associated to that service.

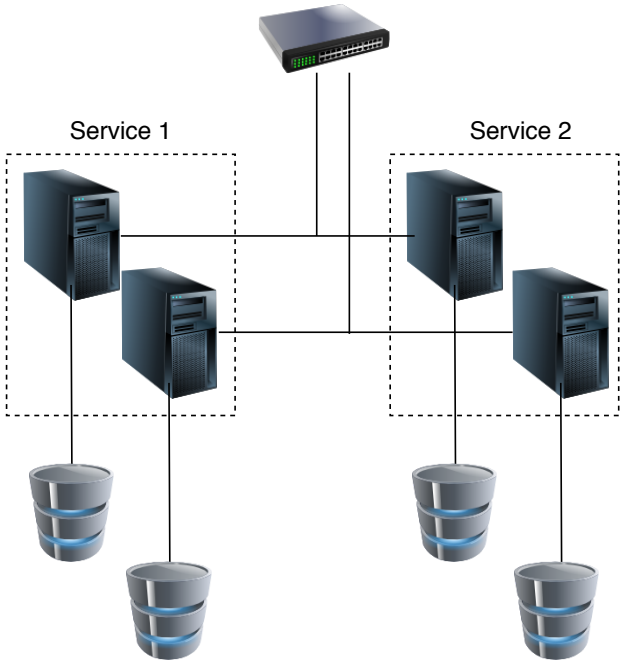



Figure 8: RAPS Architecture

### 2.6.3 Transactions Commit Protocol

Since our system is likely to use more than one single node for the Database Server, that communicate with one or more Application Server nodes, a non-blocking distributed protocol for transaction should be selected. An example is the Three-Phase Commit Protocol (3PC): the protagonists are a

Coordinator ()Transaction Manager, TM) and several Participants/Cohorts (Resource Manager, RM). Thanks to a third phase (with respect to the simple Two-Phase Commit Protocol, 2PC), each participant can become a coordinator in case of failure of the TM itself. This allows 3PC to been non-blocking, while 2PC would block in case of failure of both the coordinator or a participant. This protocol also ensures that whenever a transaction is holding some resources, it will release the lock after a timeout, trying to avoid possible deadlocks (see section 7.1 Reference Documents for more details).

## 2.7 Other Design Decisions

For what concerns the Mobile App, we decided not provide a cross-platform web based application, using some framework/library such as React.js, even though it can save time since the code can be reused for different platform.
Due to the fact that our application must interface with some wearable devices, we prefer providing native applications, both for Android and iOS. Doing so the developed applications can guarantee a better performance during their execution and a better user experience: a user who is used to, for example, the iOS graphical interface, will feel more comfortable using an app with the same type of GUI components.
Furthermore, native applications guarantee their optimal integration the one with the other and with the entire ecosystem (referring for example to an Apple Watch paired to an iPhone device).

# 3 User Interface Design

As already mentioned in the textttRequirements Analysis and Specification Document (RASD) the idea is to provide a Mobile App for individual users, both for smartphone and for smartwatch, while TrackMe just have a Web Interfaces for third parties. This choice has been driven by the idea that it would be more comfortable for third party companies, since their employees can access the system through their office laptop or desktop PC.

For a render of how we thought the user experience should look like, we refer to the specification document in which some mockups have been presented (section 3.1.1, page 13).

# 4 Requirements Traceability

**[G1]: Allow visitors to easily register in the system and later to login**

- Requirements:
  - [R1] - [R5]
- Components:
  - Profile Manager Module
  - Authentication Manager Module

**[G2]: Allow Users to share personal information/health parameters**

- Requirements:
  - [R6] - [R8]
- Components:
  - Profile Manager Module
  - Activity Tracker Module

**[G3]: Allow Third Parties to access information shared by the Users**

- [G3.1]: Allow Third Parties to access information of specific individuals
  - Requirements:
    - [R9] - [R11]
  - Components:
    - Access Requests Module
    - Individual Requests Module
    - Data Access Module
- [G3.2]: Allow Third Parties to access anonymized information and parameters of groups of individuals
  - Requirements:
    - [R12] - [R14]
  - Components:
    - Sampling Requests Module

- [G3.3]: Allow Third Parties to subscribe to new information of a specific individual and to receive it
  - Requirements:
    * [R15] - [R16]
  - Components:
    * Subscription Module
    * Data Access Module

**[G4]: Guarantee users an assistance service when necessary**

- [G4.1]: Guarantee the elderly users to receive an immediate assistance by an ambulance in case of high risk disease
  - Requirements:
    * [R17] - [R19]
  - Components:
    * Profile Manager Module
    * Activity Tracker Module
    * Health Monitoring Module
    * SOS Assistance Module

- [G4.2]: Allow users to request on-demand ambulance assistance
  - Requirements:
    * [R18] - [R20]
  - Components:
    * Activity Tracker Module
    * SOS Assistance Module

**[G5]: Guarantee the preservation of the privacy of the Users**

  - Requirements:
    * [R5]
    * [R14]
    * [R21] - [R24]
  - Components:

* Profile Manager Module
* Authentication Manager Module
* Access Requests Module
* Sampling Requests Module

# 5 Implementation, Integration and Test Plan

## 5.1 Implementation Plan

The implementation of our system will be done component by component and module by module, according to a bottom-up approach that will facilitate a high deployment coverage in early phases. The order in which our implementation will be carried out, takes in account different factors such as the complexity of the modules and how they are interconnected with one another and we must take in consideration the possibility of discovering flaws and bugs in our designed system that will require to be fixed as soon as possible in order to prevent further additional costs and difficulties in debugging.

Hence, we can group the system components as follows:

- **Model**: This component is the first one that must be implemented since all the parts of the Application Server will be using its elements and it allows services to communicate with the DBMS and thus, it is a very important and crucial component for the whole system.

  The following two components can be identified as the most crucial of the system and their implementation is required to be accomplished as soon as possible since they enclose all the functionalities required both for Users and for Third Parties, and for this reason their fulfillment will require a significative amount of time.

- **User Mobile Services**: This macro-component, along with the `Third Party Web Services` component is the most complex and most important of the system, due to its functionalities of providing all the services necessary for the User. The functionalities provided are achieved by means of different internal modules, identified as follows: `ActivityTrackerModuel`, `HealthMonitoringModule`, `SOSAssistanceModule`, `AccessRequestsModule` and `ProfileManagerModule`.

- **Third Party Web Services**: Likewise, this macro-component is very complex and its functionalities provide all the services necessary for Third Parties. The functionalities provided are achieved by means of different internal modules, identified as follows: `DataAccessModule`, `SubscriptionModule`, `IndividualRequestsModule`, `SamplingRequestsModule` and `AuthenticationManagerModule`.

- **Client and Router**: These components are used to redirect requests and offer the User a way to communicate with the modules and services which will be carrying out the main functions of the system.

## 5.2 Integration and Testing

In the following sections we are going to illustrate how external components will be integrated with the ones implemented for the system and what testing strategy will be adopted.

### 5.2.1 Entry Criteria

The integration of components and its testing should be carried out as soon as possible, although some conditions must be respected before beginning the verification activity.

- In the very first place, the `Requirements Analysis and Specification Document` and the `Desing Document` must be complete and available to all the people involved in the development of the system in order to make them aware about requirments and adopted design choices.
- External components and API must be available and serviceable even before the beginning of the implementation process.
- The modules which are being integrated in the system are supposed to have at least the operations concerning on another created, if not carried out completely.
- The developed operations should, in the first place, pass unit tests to make sure that the components are working properly, and in case of failure, it will be guaranteed that a misfunctioning is located in the integration process itself and it will be consequently much easier to target the problem and solve it.

Furthermore, in order to test the integration of two generic components, the main featues of both of them should have been developed (partially, if not completely) and the related test units should have been performed. More clearly, there are minimum coverage percentages of completion of each component with its functionalities that have to be reached:

- 70% of the Activity Tracker Module
- 70% of the Health Monitoring Module

- 80% of the SOS Assistance Module
- 80% of the Access Requests Module
- 60% of the Profile Manager Module
- 80% of the Data Access Module
- 60% of the Subscription Module
- 80% of the Individual Requests Module
- 60% of the Sampling Requests Module
- 60% of the Authentication Manager Module

The different percentage values reflect the minimum number of features which need to be implemented in order to perform meaningful integration tests.

### 5.2.2 Elements to be Integrated

The system is composed by the components previously described and will be integrated as follows:

- `Integration of the components with the DBMS`
- `Integration of the components with external services`
- `Integration of the Client side with the Application Server`
- `Integration of the components of the Application Server`

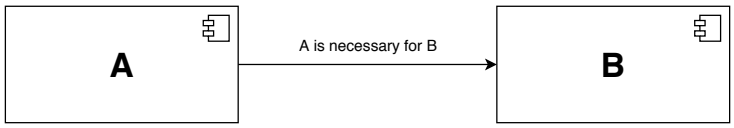
- **Integration of the components with the DBMS**
  The required integrations are the following:
  - `ActivityTrackerModule, DBMS`
  - `SOSAssistanceModule, DBMS`
  - `AccessRequestModule, DBMS`
  - `ProfileManagerModule, DBMS`
  - `DataAccessModule, DBMS`
  - `SubscriptionModule, DBMS`
  - `SamplingRequestsModule, DBMS`
  - `AutheticationManagerMoudle, DBMS`
- `Integration of the components with external services`
  The required integrations are the following:
  - `ActivityTrackerModule, MapsService`
  - `ActivityTrackerModule, SensorsService`
  - `SOSAssistanceModule, AmbulanceDispatchingSystem`
  - `SOSAssistanceModule, NotificationsService`
  - `AccessRequestModule, NotificationsService`
  - `ProfileManagerModule, SMSAPIService`
  - `IndividualRequestsModule, NotificationsService`
- `Integration of the Client side with the Application Server`
  This integration is accomplished to enable the sending of requests between the Server side and the App used by the User.
- `Integration of the components of the Application Server`
  The required integrations are the following:
  - `HealthMonitoringModule, ActivityTrackerModule`
  - `SOSAssistanceModule, ActivityTrackerModule`
  - `SOSAssistanceModule, HealthMonitoringSystem`
  - `SubscriptionModule, DataAccessModule`
  - `IndividualRequestsModule, SubscriptionModule`

### 5.2.3 Integration Testing Strategy

Given that the entire development process follows a bottom-up approach, we opted to use a similar approach for the testing phase: in the very first place we will start by integrating those components that do not depend on other ones in order to work properly, or those who depend on already developed components (such as external API's). After all the single components are tested, we will end up by the subsystems the previous components take part of. This allows us to begin to test a component or a subsystem as soon as it's finished (or near completion), which in turn gives us immediate feedback and allows us to (partially) parallelize the development phase and the testing phase.

## 5.3 Sequence of Component/Function Integration

In this section of the document the order of integration of the components and subsystems of TrackMe system will be described. As a notation, an arrow going from component A to component B means that A is necessary for B to function, so it must have already been implemented before performing the integration.




### 5.3.1 Software Integration sequence

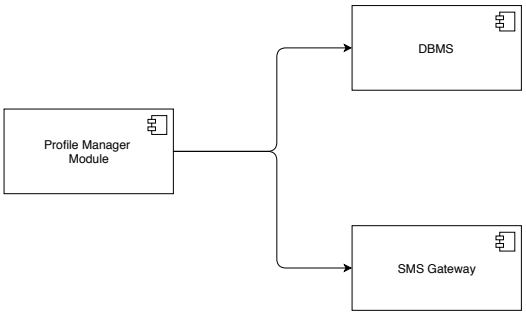



Figure 9: Profile Manager

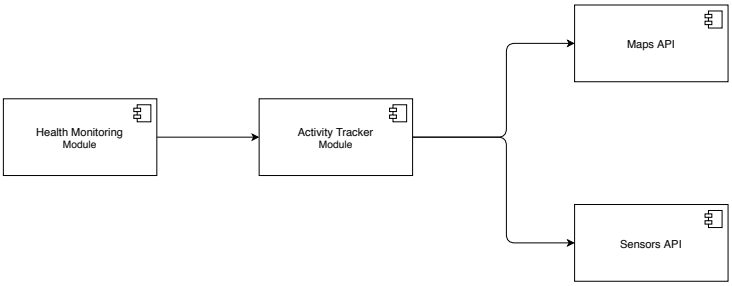



Figure 10: Activity Tracker

# 6 Effort Spent

- Luca Conterio

| Day | Subject | Hours |
|---|---|---|
| 19/11/2018 | High Level Components | 1.5 |
| 22/11/2018 | Component View | 1.5 |
| 25/11/2018 | Component View | 3 |
| 26/11/2018 | Deployment View | 2.5 |
| 27/11/2018 | Deployment View | 2 |
| 29/11/2018 | Component Interfaces | 1.5 |
| 02/12/2018 | Runtime View | 2 |
| 04/12/2018 | Runtime View | 2 |
| 06/12/2018 | Requirements Traceability | 1 |
| Total | | |

- Ibrahim El Shemy

| Day | Subject | Hours |
|---|---|---|
| 19/11/2018 | Introduction of the document | 1.5 |
| 22/11/2018 | Component View | 2.5 |
| 25/11/2018 | Component View | 3 |
| 26/11/2018 | Deployment View | 2.5 |
| 27/11/2018 | Deployment View and Design Patterns | 2 |
| 04/12/2018 | Implementation, Integration and Test Plan | 1 |
| 05/12/2018 | Implementation, Integration and Test Plan | 3 |
| Total | | |

# 7 References and Used Tools

## 7.1 Reference Documents

- Specification Document "Mandatory Project Assignment A.Y. 2018/2019"
- "Appunti di Sistemi Informativi per il Settore dell'Informazione" A.Y. 2017/2018
- "Java Design Patterns", Vaskaran Sarcar
- Software Engineering 2 course slides
- Data Bases 2 course slides
- "https://developer.jboss.org/wiki/Three-phaseCommitProtocol"

## 7.2 Tools

- **Draw.io**: `https://www.draw.io/`
- **TeXStudio**: `http://www.textstudio.org/`
- **Github**: `https://github.com/`