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`.

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

1.3.3 Abbreviations

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**

2 Architectural Design

2.1 High Level Components and their Interactions

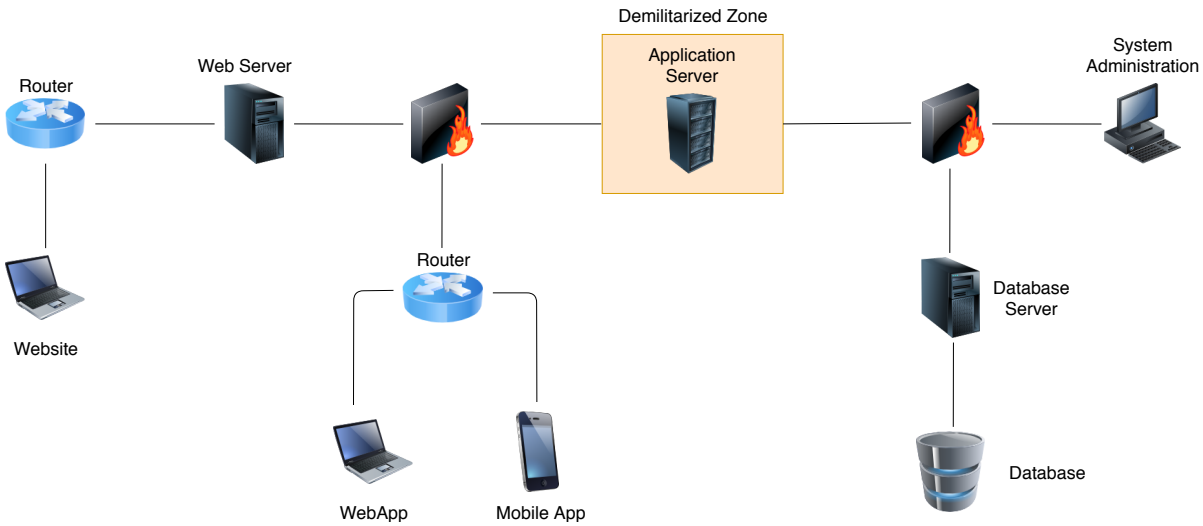


Figure 1: High Level System Structure

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.

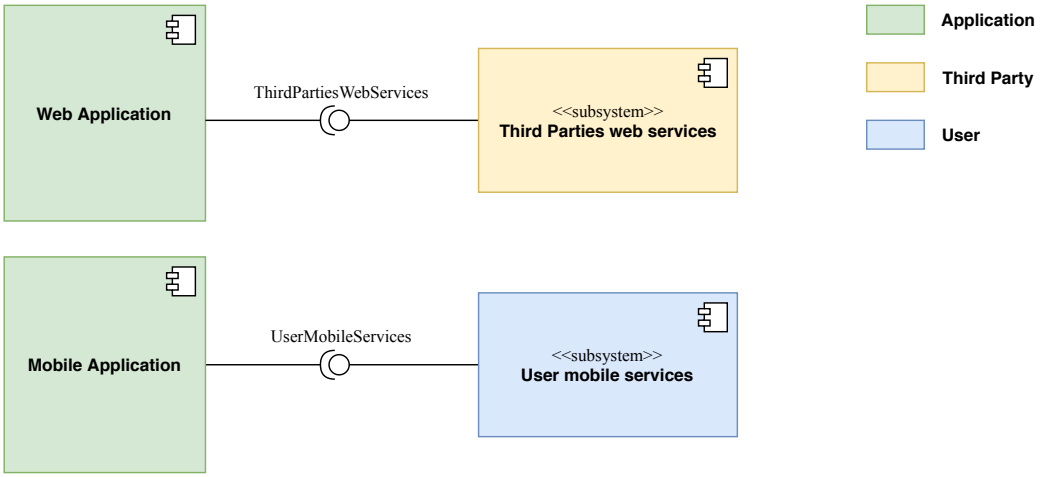
System Administration is implemented through a third-party ERP solution, hence we do not provide information about its implementation.

2.2 Component View

2.2.1 High Level Component Diagram

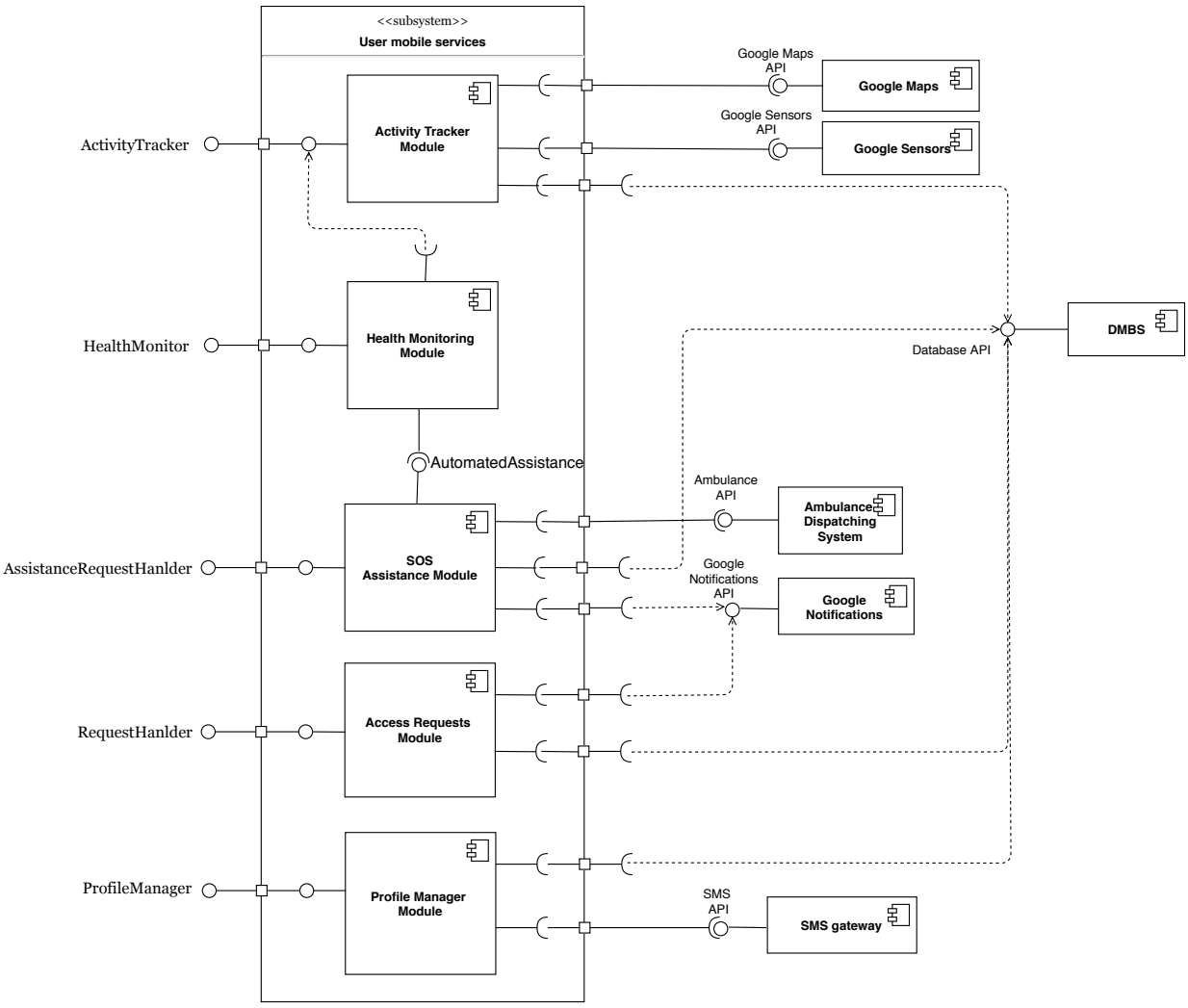
The following diagrams illustrate the system components and the interfaces through which they interact to fullfil their functionalities. A distinction can be made between Client side and Server side:

- 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.

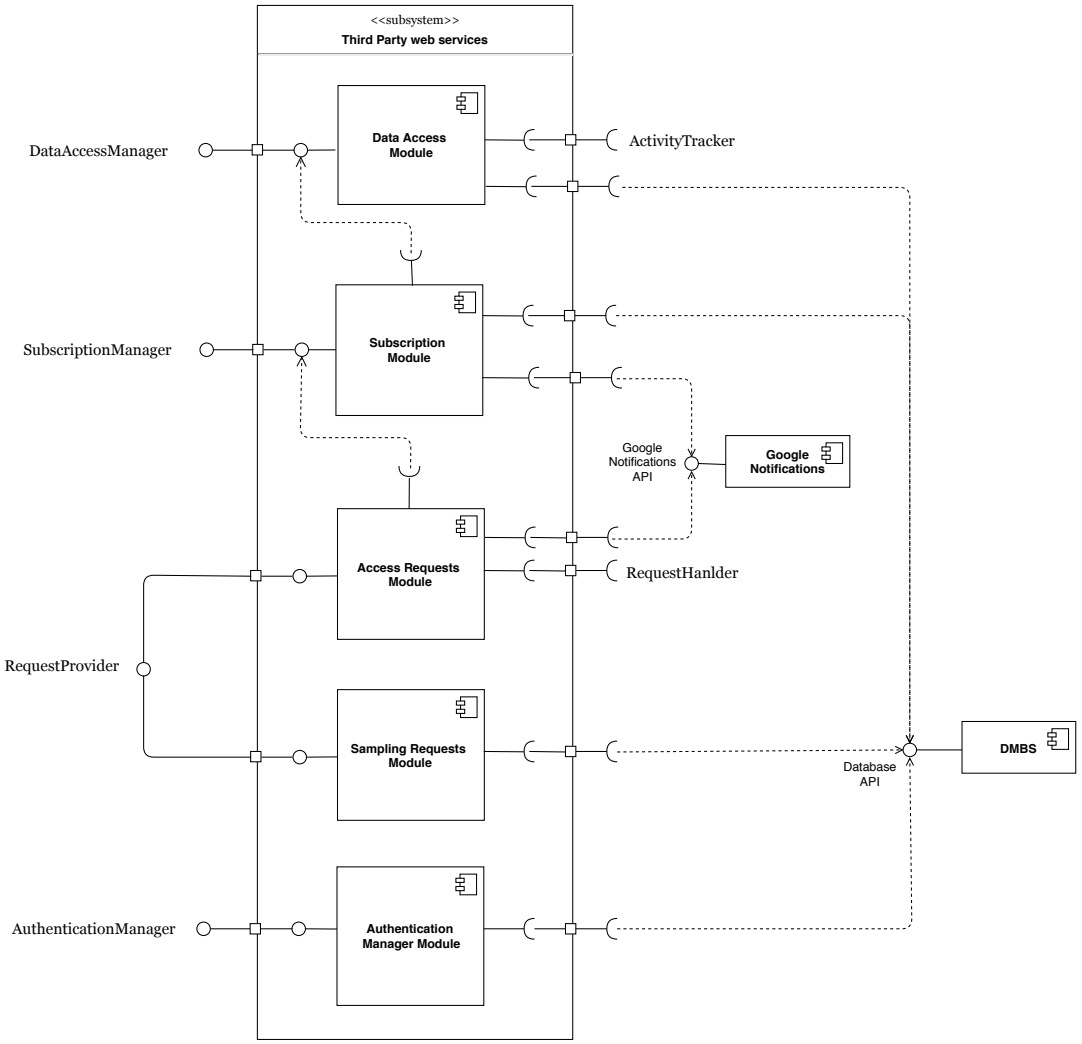
- `SOS Assistance Module`: in this module all functions dealing with assistance requests are implemented. It directy handles on-demand assistance requests and automated assistance requests performed by the Health Monitoring Module through the AutomatedAssistance interface.

- `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`, `Access 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.
- `Access 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.

- **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.

- **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.

As mentioned briefly in the RASD (sections 3.9.2, 3.9.5), our system has to rely on a RAPS 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 precisly, 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.

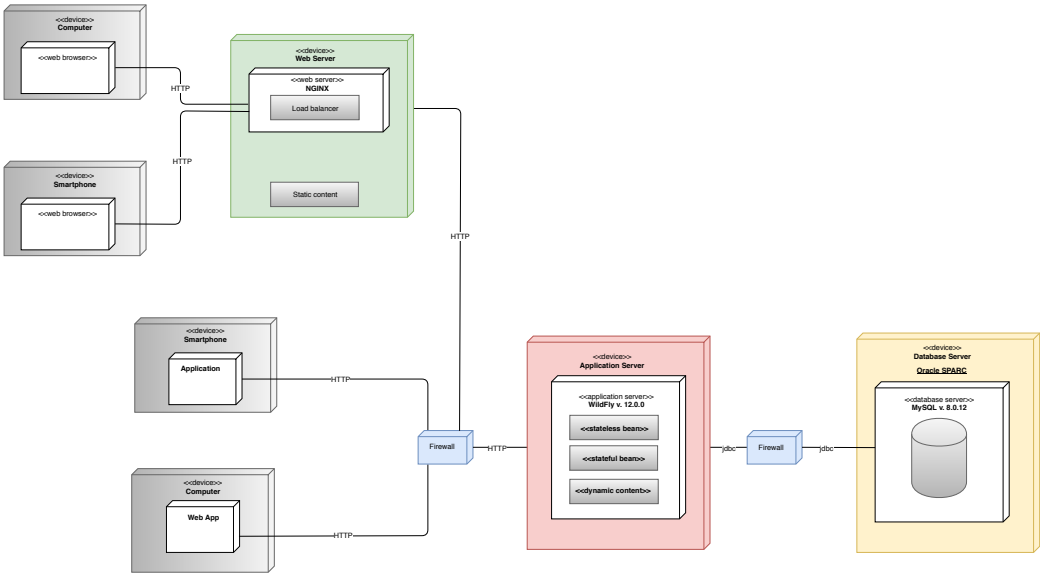


Figure 2: Deployment View Diagram

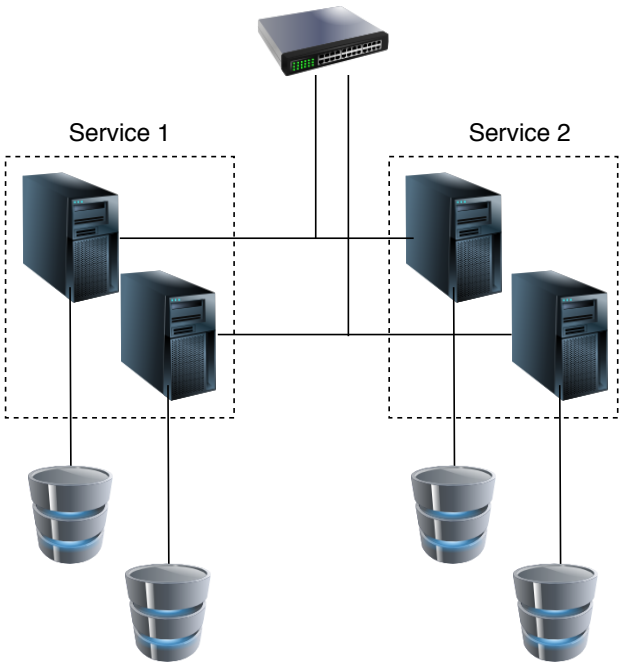


Figure 3: RAPS Architecture

2.4 Component Interfaces

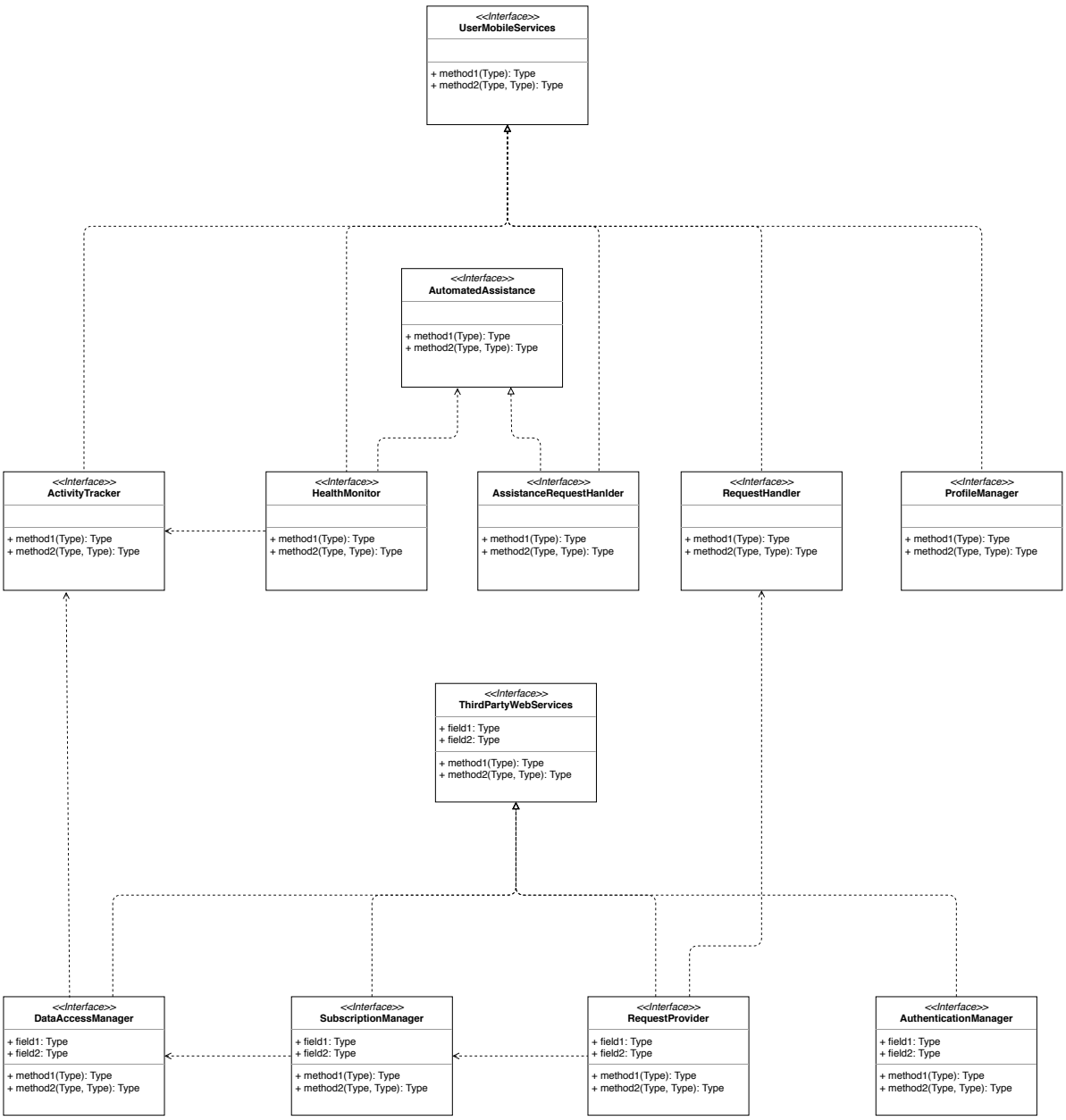


Figure 4: Component interfaces and their interactions.

2.5 Runtime View

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).

- **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.

- **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`, `UserMobileServices` and `UserSmartwatchServices` 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.

2.7 Other Design Decisions

3 User Interface Design

4 Requirements Traceability

5 Implementation, Integration and Test Plan

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
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 and Design Patterns	2.5
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

7.2 Tools

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