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

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

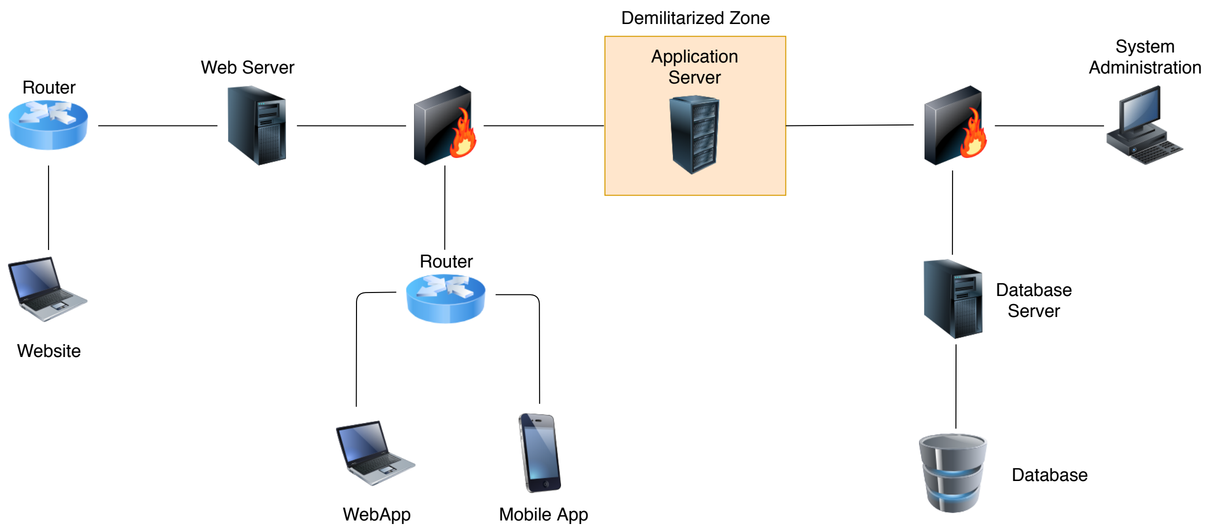


Figure 1: High Level System Structure

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.

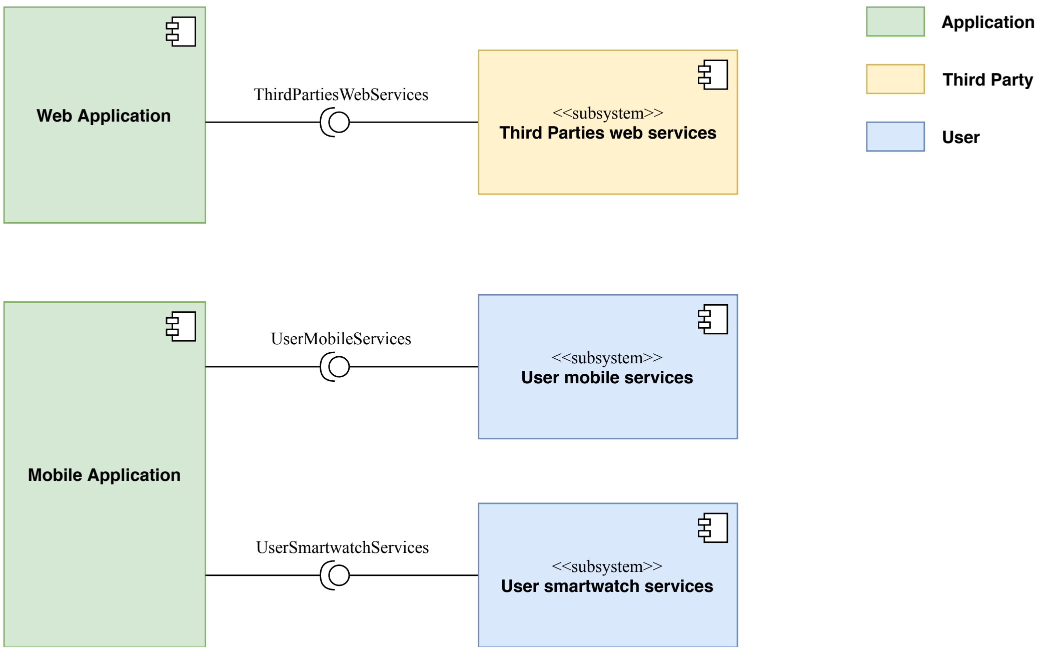
System Administration is implemented through a third-party ERP solution, hence we do not provide information about its implementation.

3 Component View

3.1 High Level Component Diagram

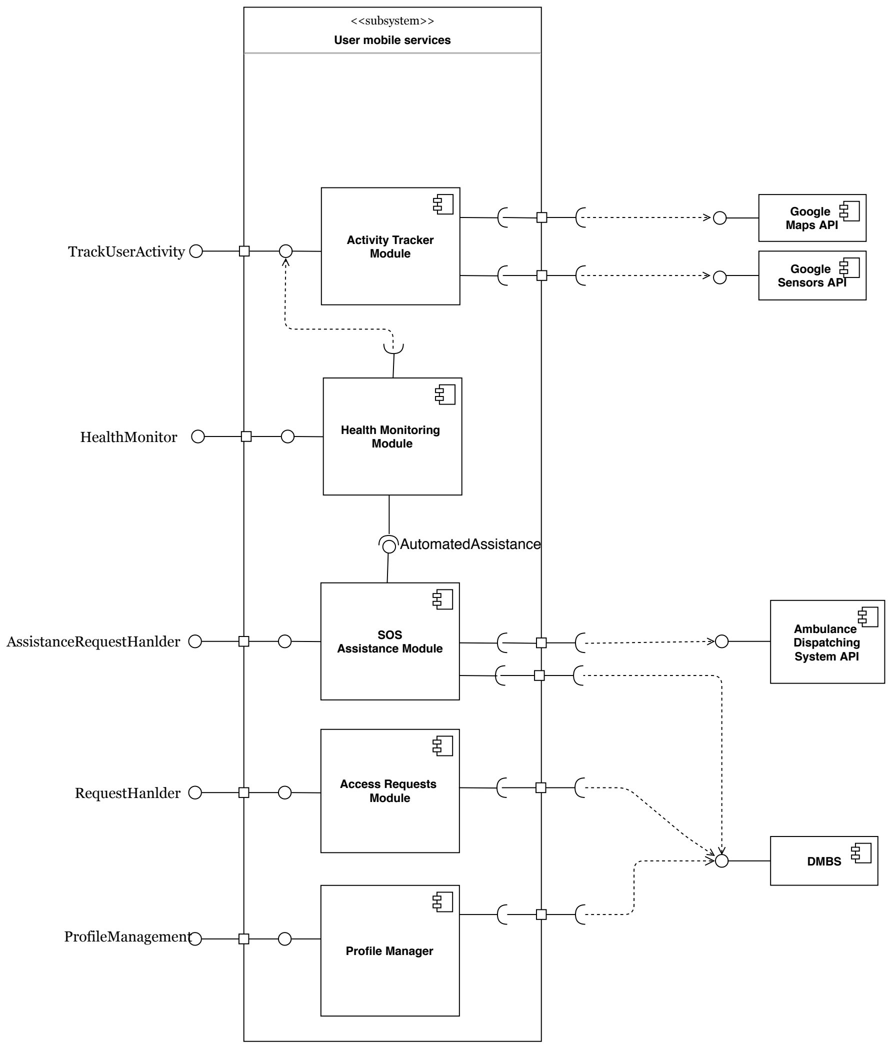
The following diagrams illustrate the system components and the interfaces through which they interact to fullfil their functionalities. A distinction can be made between Client side and Server side:

- The Client side is composed by two components, `Web Application` and `Mobile Application`, referring to the following services: `ThirdPartiesWebServices`, `UserMobileServices` and `UserSmartwatchServices`.
- The Server side is composed of three 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. and `User smartwatch services` that will provide the User a similar interface to the one provides by the mobile services.



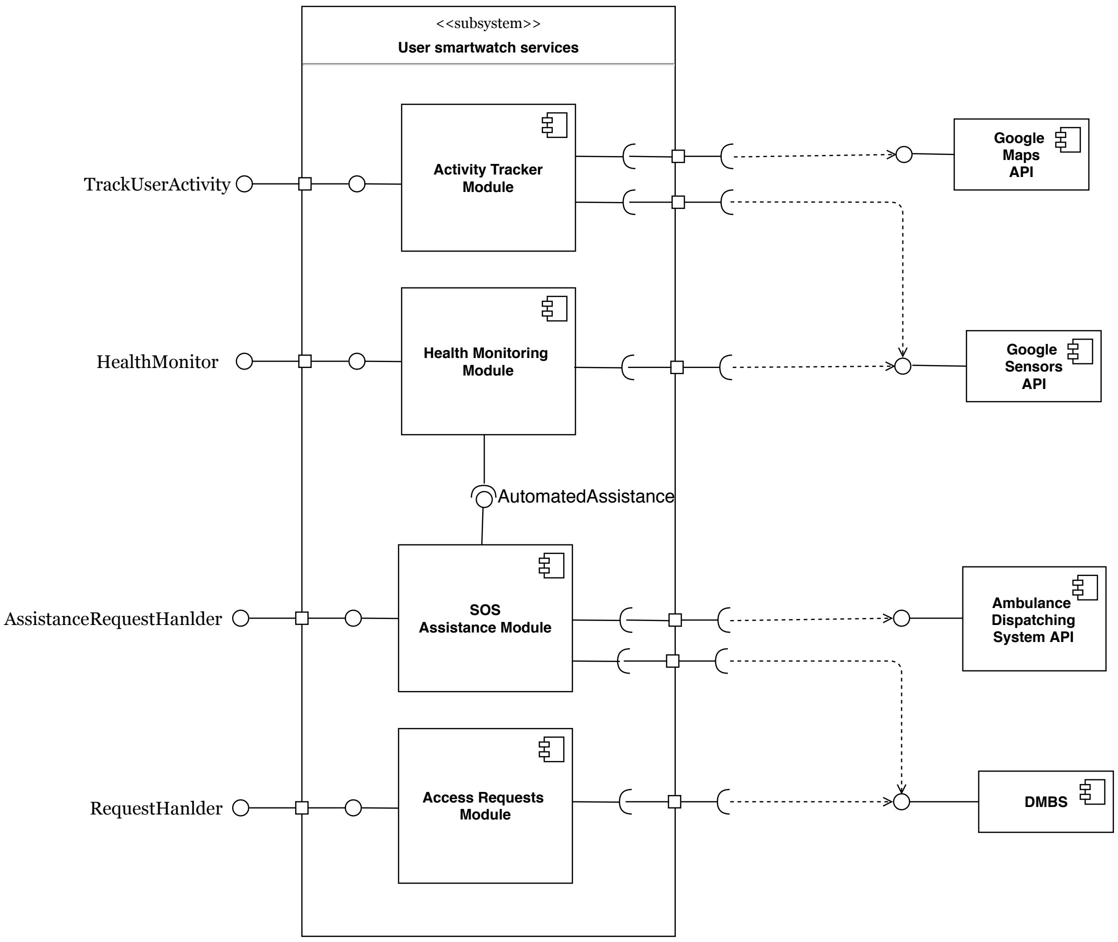
3.2 User Mobile Services Projection

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



3.3 User Smartwatch Services Projection

User smartwatch services subsystem is similiar to the User mobile services system with a minor difference; in fact it is composed by four components: `Activity Tracker Module`, `Health Monitoring Module`, `SOS Assistance Module` and `Access Requests Module`. These components provide to the User Mobile Application the following interfaces: `TrackUserActivity`, `HealthMonitor`, `AssistanceRequestHandler` and `RequestHandler`. The components need also to communicate with `Google Maps API`, `Google Sensors API`, `Ambulance Dispatching System API` and the `DBMS` to work properly and guarantee a better User experience with the Application.
We decided not to provide the User an interface to manage his profile from smartwatch, since it is not user friendly nor comfortable.



3.4 Deployment View

3.4.1 Overall Architecture

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.

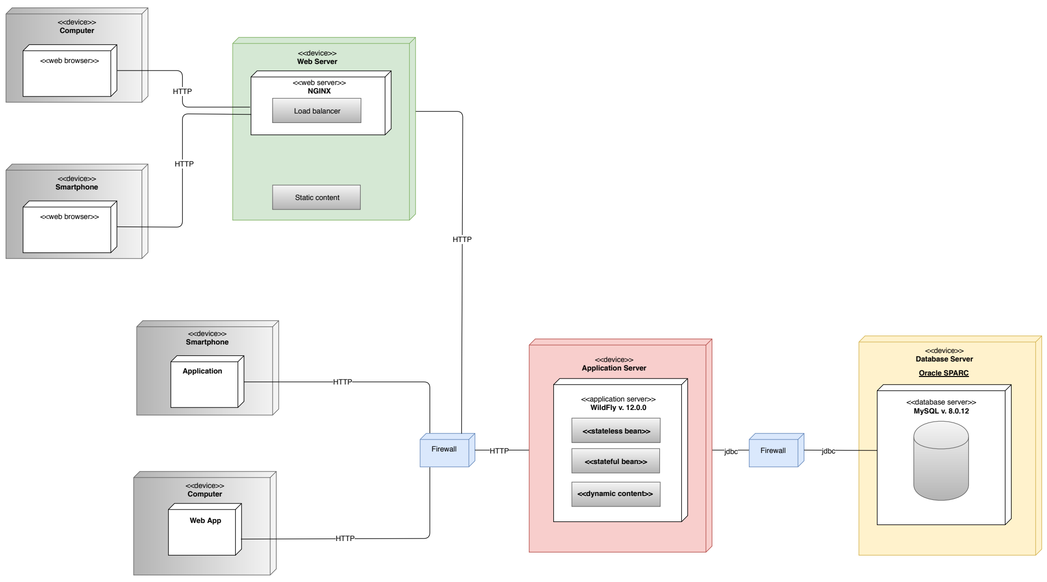


Figure 2: Deployment View Diagram

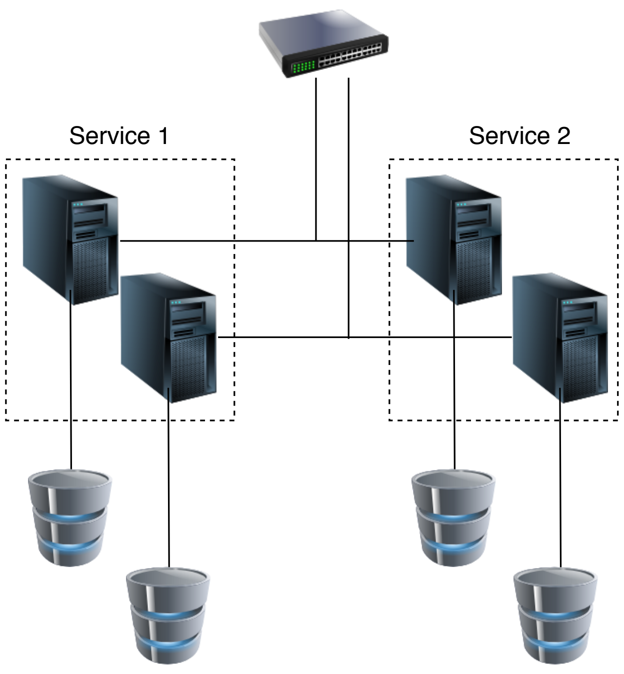


Figure 3: RAPS Architecture

3.4.2 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.

3.5 Runtime View

4 Effort Spent

- Luca Conterio

Day	Subject	Hours
19/11	High Level Components	1.30
22/11	Deployment View	2.30

- Ibrahim El Shemy

Day	Subject	Hours
19/11	Introduction of the document	1.30
22/11	Deployment View	2.30

5 References and Used Tools

5.1 Reference Documents

- Specification Document ”Mandatory Project Assignment A.Y. 2018/2019”

5.2 Tools

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