POLITECNICO
MILANO 1863

Department of Computer Science and Engineering

Design Document (DD)

TrackMe

- v1.0 -

Authors:

Avila, Diego	903988
Schiatti, Laura	904738
Virdi, Sukhpreet	904204

December 10th , 2018

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

TrackMe develops health-monitoring devices devoted to measure and record different parameters related to the health status of a person (i.e. body temperature, blood pressure, heart pulse rate and percentage of O2 in the blood) and also their location. TrackMe health smartwatch is synchronized with an app that gives users access to their data and stats. This document discuss more technical aspects of working of the TrackMe application.

1.2 Purpose

The purpose of this document is to give more technical details than the RASD about TrackMe application system. While the RASD presented a general view of the system and what functions the system is supposed to execute, this document aims to present the implementation of the system including components, run-time processes and deployment. It also presents in more details the implementation and integration plan, as well as the testing plan.

More precisely, the document presents:

- Overview of the high level architecture
- The main components and their interfaces provided one for another
- The runtime behaviour
- The design patterns
- Implementation plan
- Integration plan
- Testing Plan

The purpose of this document is to provide an overall guidance to the architecture of the software product.

1.3 Scope

The TrackMe environment will be composed by three systems, whose scope are defined in this section.

Data4Help is a system capable to store the physical health data of any individual using any TrackMe wearable device. All the stored data will be available to any third party company, after request and approval of its use is sent to the TrackMe wearable device user, who will be able to accept or reject the request, under no condition. Also, third party companies will be able to request data of a group of users, and Data4Help will provide that information under the solely condition of be able to anonymize it.

In addition, **AutomatedSOS** is a system that will be built on top of Data4Help, and will send requests to every elderly individual and responds back, if accepted, it adds to ASOS database in order to access its physical health parameters. ASOS is able to respond only if a city is available for health-care service in that city, before creating a request to that user. ASOS automatically assigns emergency contact according to the city of the user. It will monitor the physical health status of all of its users in order to establish their health condition based on previously defined thresholds. If any of the user's health parameters is under or above its threshold, the system will contact the associated health-care service for that user. AutomatedSOS will do its best effort to contact the health-care service, but under no circumstances will follow-up the status of the health-care response, meaning that it will not know if the health-care service answered the notification or not.

Finally, **Track4Run**, also a system built on top of Data4Help, will send requests to every user in order to access their location and let them participate in a defined running circuit. Track4Run organizer users, will be able to setup a race, define its running circuit and send invitations to all the TrackMe wearable device users. Also, the race spectators will be able to follow the participants' location using the Track4Run web site. The location of every participant will only be available during the race.

1.4 Definitions, Acronyms, Abbreviations

1.4.1 Definitions

- **Data trading**: Generate revenue from user data in a much more direct way, by selling user data to a third party.

1.4.2 Acronyms

- DD: Design Document
- D4H: Data4Help
- ASOS: AutomatedSOS
- T4R: Track4Run
- GUI: Graphical User Interface
- MVC: Model View Controller is a design pattern used for GUIs
- RASD: Requirements Analysis and Specifications Document
- JMS: Java Message Service
- DSL: Digital Subscriber Line

1.4.3 Abbreviations

- $[Gn]$: n-goal.

1.5 Revision history

It is important to keep track of the revisions made to this document:

Version	Last modified date
1.0	10^{th} December, 2018

Table 1.1: Revision history timeline

1.6 Document structure

This document is divided in seven parts, each one devoted to approach each one of the steps required to apply requirements engineering techniques.

- Chapter 1 gives an introduction of the design document. It contains the purpose and the scope of the document, as well as some abbreviation in order to provide a better understanding of the document to the reader.

- Chapter 2 deals with the architectural design of the application. It gives an overview of the architecture and it also contains the most relevant architecture views: component view, class view, deployment view, runtime view and it shows the interaction of the component interfaces. Some of the used architectural designs and designs patterns are also presented here, with an explanation of each one of them and the purpose of their usage.
- Chapter 3 refers to the mock-ups already presented in the RASD document.
- Chapter 4 explains how the requirements that have been defined in the RASD map to the design elements that are defined in this document.
- Chapter 5 presents the implementation, integration and test plan. It includes the how the different components of the application are integrated with each other, how they react, the testing strategy taken into account and analyse the risks in the application.
- Chapter 6 shows the effort spent by each group member while working on this project.
- Chapter 7 includes the reference documents.

Chapter 2

Architectural design

2.1 Overview

Application architecture design is a process which has to be executed in a defined flow. High-level components and their interaction is displayed in Figure 2.1.

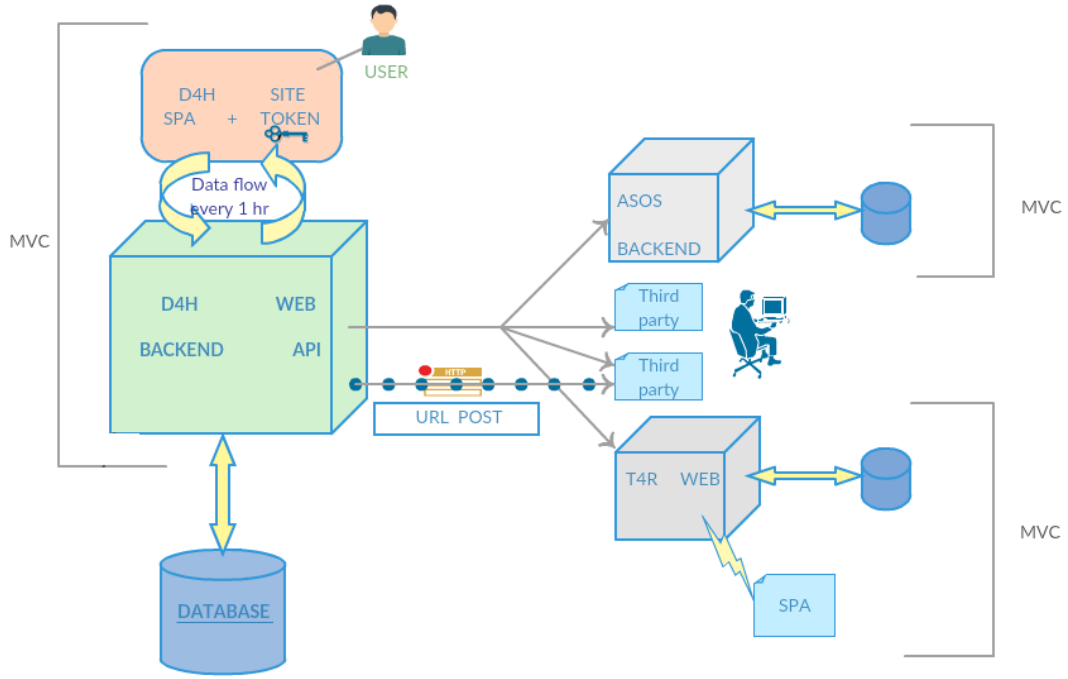

Figure 2.1: High-level architecture Diagram

The above architecture diagram depicts clearly that third party - Data4Help relation is a service oriented, when the third-party user asks for some specific data and it is an event based, when it is subscribed to the data. In addition, a CRON job runs every day at 00:00, if there is every user above age of 60 years - they receive an ASOS notification. The system uses MongoDB, that listens, watch collection and events. Apparently, all the APIs are RESTful.

When the third-party users have subscribed to the bulk data subscription, they define an update time when they want to get the refreshed data (e.g. update data in every 'X' hours).

2.2 Component view

2.2.1 Data4Help component diagram

In the Figure 2.2 it can be seen the component diagram of D4H, with all its external interfaces. It can be noticed two main components: **DataBase** and **D4HBackend**, where the former one refers to the Data4Help database, and which provides an interface used by the **DBManager** component.

On the other hand, **D4HBackend** component, contains all the components related to D4H, which are needed to provide the interfaces used by the third parties and the web site. The following interfaces are used by the web site: **SignupWeb**, **LoginWeb**, **SearchWeb**, **RequestWeb** and **SubscriptionWeb**, while the **RequestAPI** interface is used by the third parties. In this case, D4H provides the interface in order to let the third parties send requests for accessing the health status and location of the individuals to them.

Furthermore, **AuthenticatorManager** component has the responsibility of validate the different credentials, and to provide the secret codes to the third parties, to do so, it interacts with the **TokenDB** component using the **TokenConnector**.

Finally, it can be seen the different third parties related to D4H. It worth mentioning **Track4Run** and **AutomatedSOS** components which, even though they are part of the TrackMe environment, they are treated as third parties in the sense they are completely decoupled of D4H. Moreover, all third parties must provide an interface to D4H in order to let it communicate the incoming changes of the subscriptions.

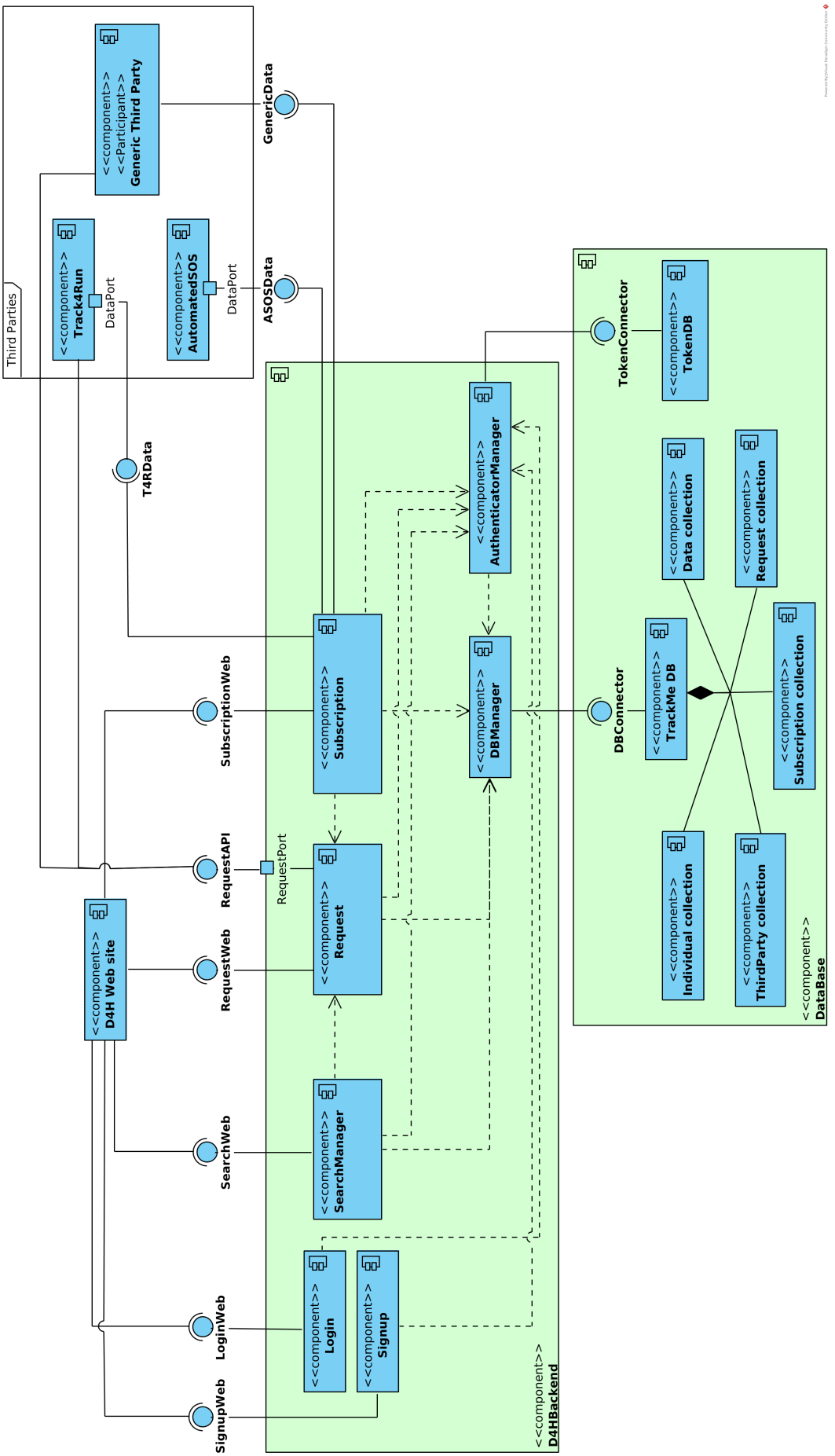

Figure 2.2: Data4Help Component Diagram

2.2.2 Track4Run component diagram

In the Figure 2.3 it can be seen the T4R components. As in the D4H component diagram, the *DataBase* component provides an interface in order to let the **DBManager** component communicate to it.

The main structure of the system is similar to the structure seen in D4H component diagram. The web site communicates with the back-end using the following interfaces: **LoginWeb**, **SignupWeb**, **UserWeb** and **NotificationWeb**. On the other hand, T4R provides the **DataAPI** interface, which is responsibly of receiving all the data of the participants, and uses the **RequestAPI** interface, in order to send the requests for accessing the individuals' location and health status.

Finally, as in D4H, the **AuthenticatorManager** component is responsible of validate the users' credentials.

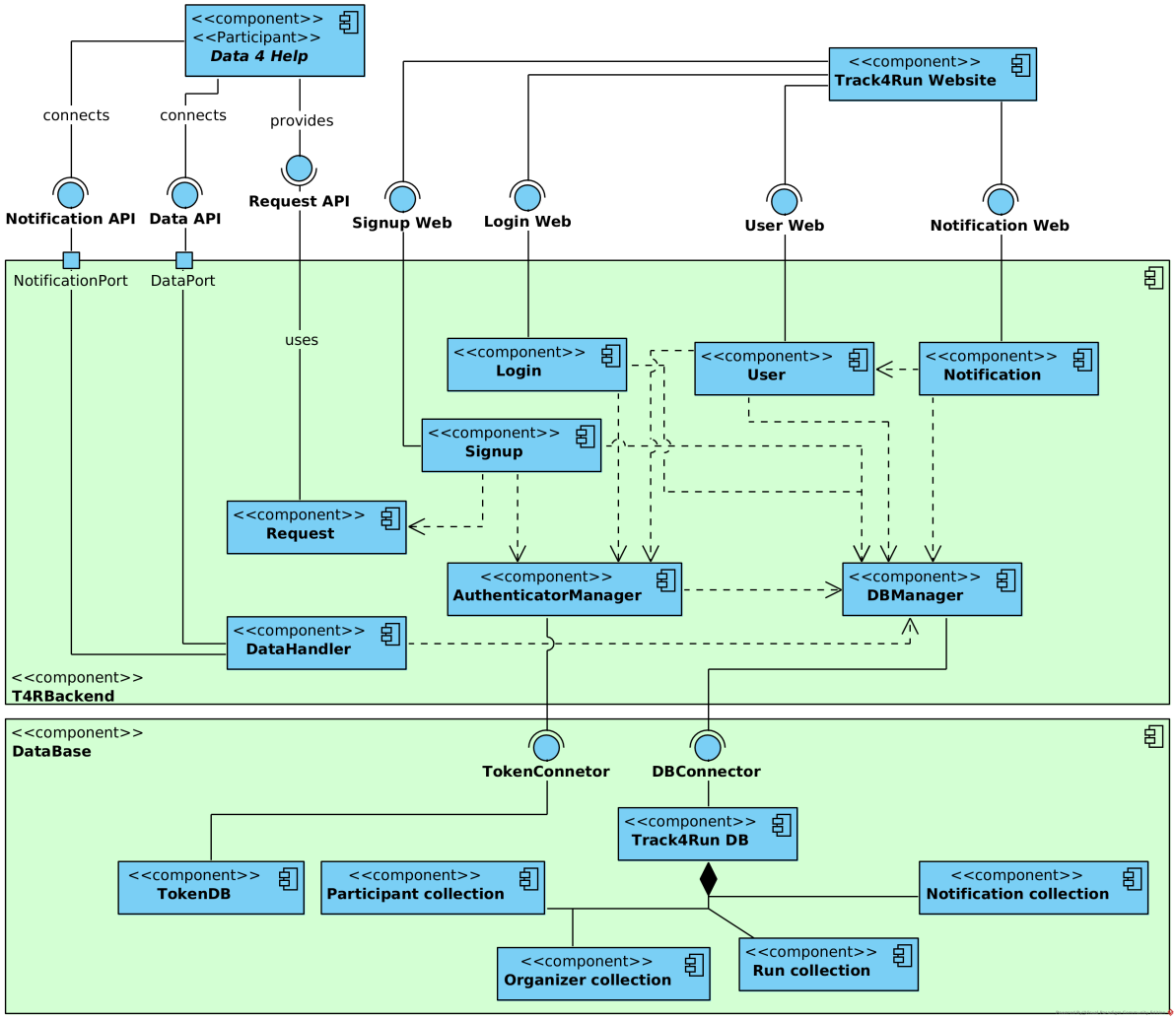

Figure 2.3: Track4Run Component Diagram

2.2.3 AutomatedSOS component diagram

In the Figure 2.4 it can be seen the ASOS components. It can be noticed that the system has a similar structure of T4R: the **ASOSBackend** component communicates with **DataBase** component through the **DBConnector** interface, and provides a **DataAPI** interface to receive the updated information of the individuals.

On the other hand, **ASOSBackend** component sends notifications to the different **Health Care Service** components using the provided **Alarm Interface**.

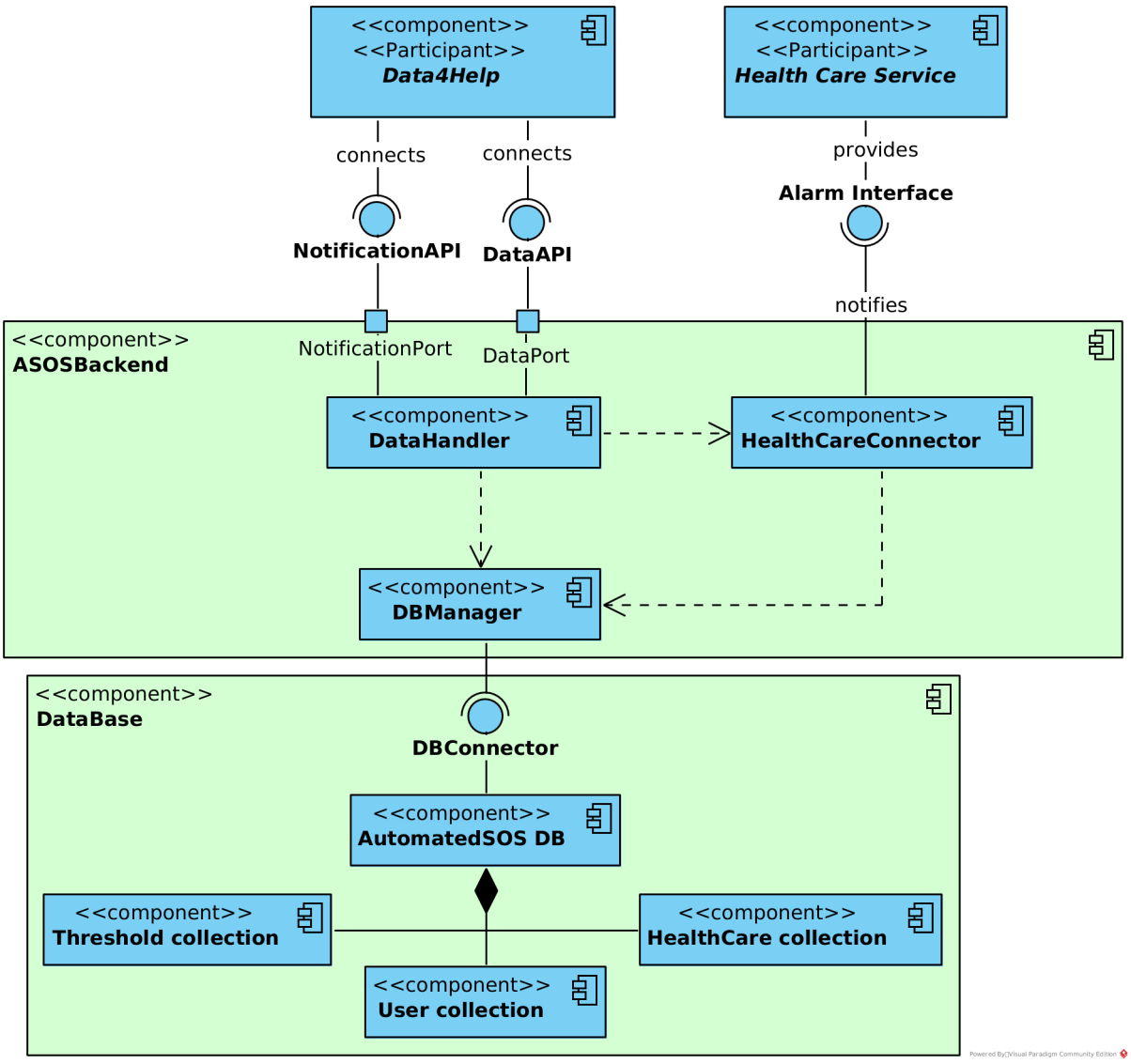

Figure 2.4: AutomatedSOS Component Diagram

2.3 Deployment view

In the current chapter the deployment diagrams of D4H, T4R and ASOS are shown. All diagrams follow the same structure and design: yellow nodes represent external nodes that interact with the system, green nodes represent hardware devices (i.e. web server, database server, etc.), pale orange nodes represent execution environments such as JVM, and blue boxes represent high level components. Finally, the protocols used to communicate between different nodes displayed as red links.

2.3.1 Data4Help deployment diagram

In the Figure 2.5 can be seen the deployment diagram of D4H. As can be seen, the **Web Server** node, is a device which contains the execution environment that holds the **D4H Backend** component. The **Web Server** node, is related to two different devices: a **MongoDB Server** which is the environment of the **TrackMe DB**, and a **Redis Server** which is the environment of the **TokenDB**.

Moreover, the **Web Server** is related to three third parties: two of them are **Track4Run** execution environment, and **AutomatedSOS** execution environment. The third third party is a **Generic Third Party Backend** node that is external to the TrackMe environment.

Finally, the deployment diagram does not show the interaction between the web-users (individuals or third parties) with the system. This interaction is inherent to a web site, and it happens between the client and the **D4H Web page** component.

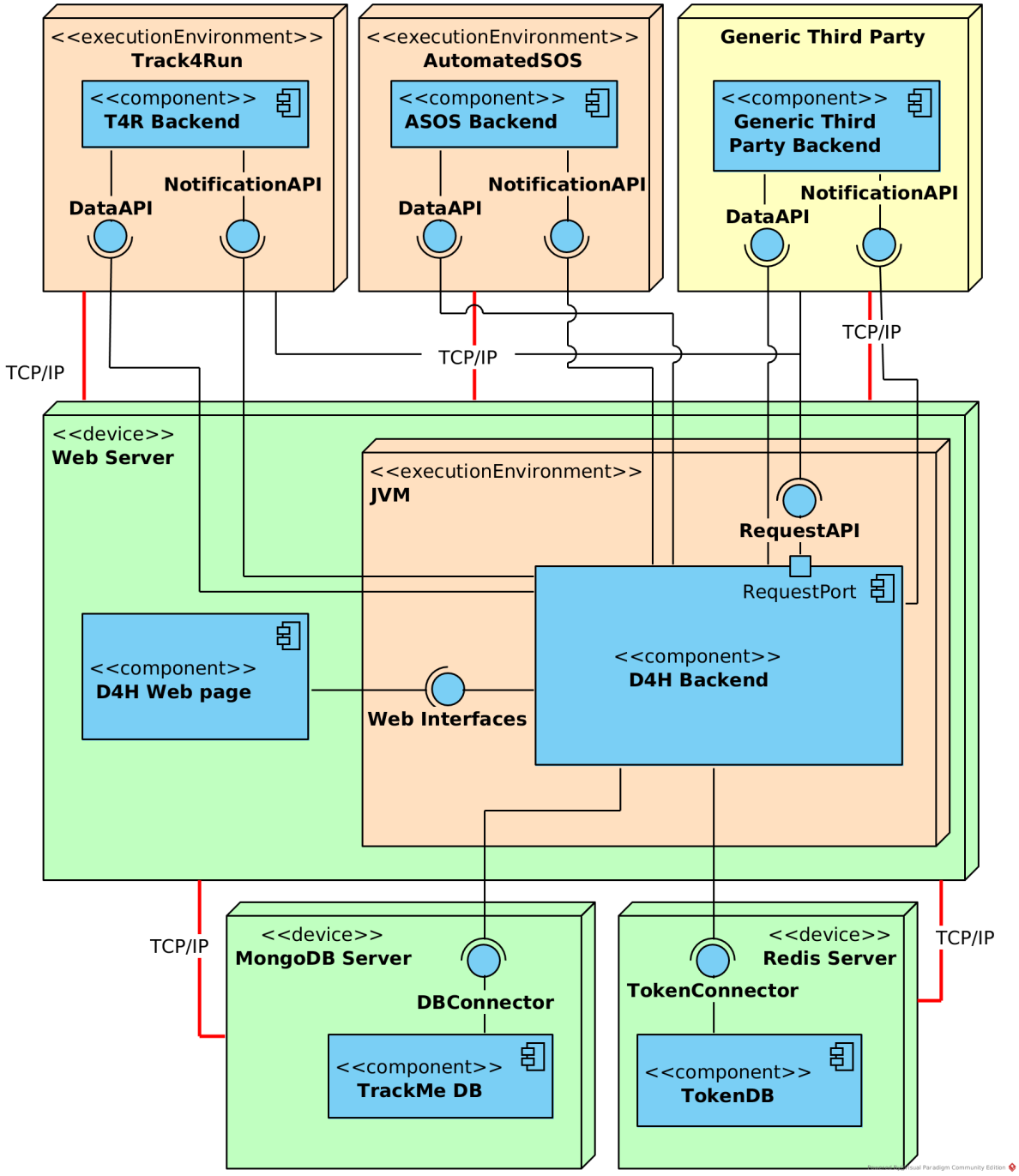

Figure 2.5: Data4Help Deployment Diagram

2.3.2 Track4Run deployment diagram

In the Figure 2.6 the T4R deployment diagram is shown. It can be seen the interaction between **Data4Help** execution environment and the **Web Server** node. This interaction happens thanks to the different interfaces provided by D4H and T4R.

Furthermore, the **Web Server** node interacts with the **MongoDB Server** node using

the TCP/IP protocol, and it works thanks to the **DBConnector** interface provided by the **T4R Database** component.

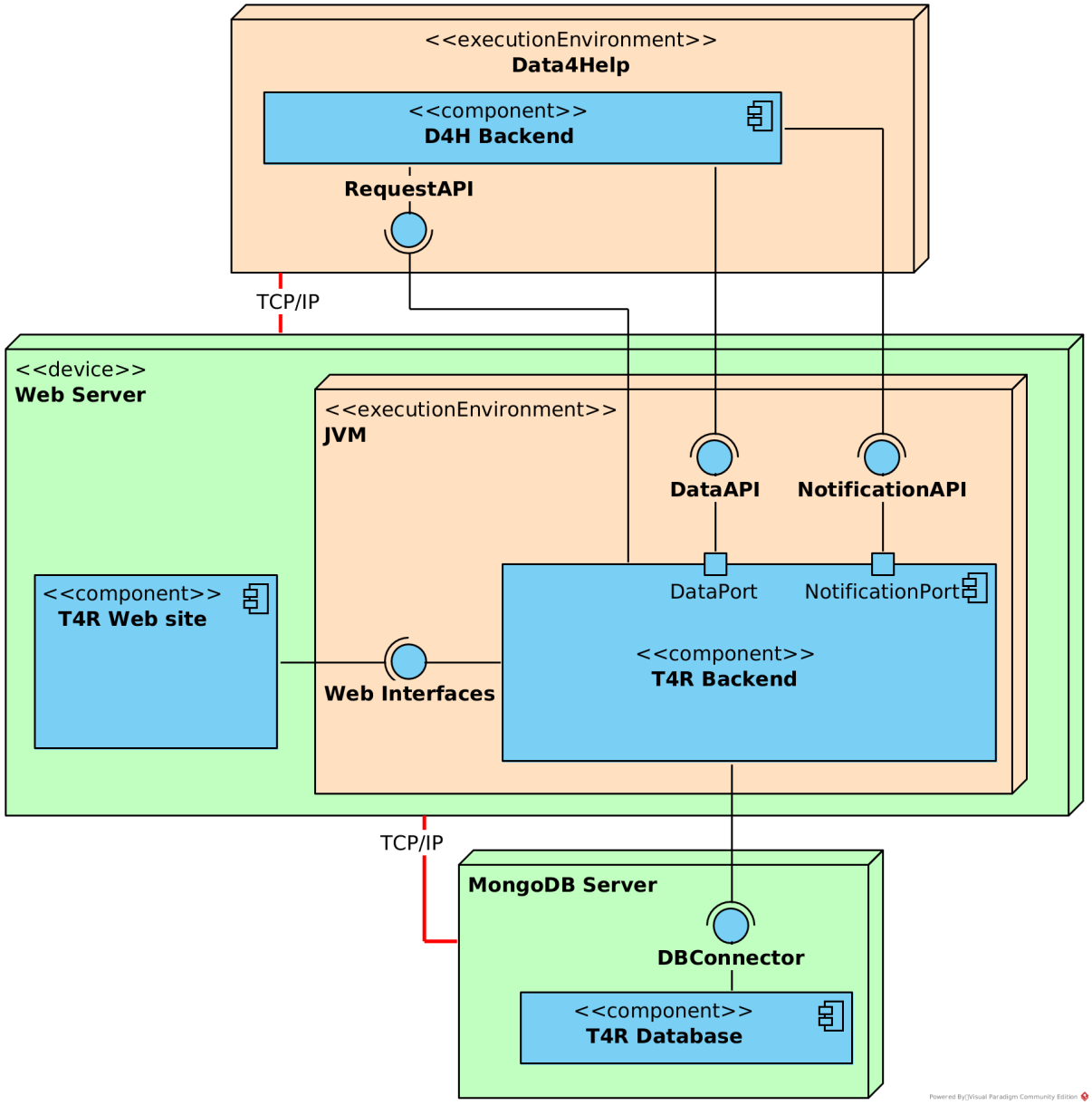

Figure 2.6: Track4Run Deployment Diagram

2.3.3 AutomatedSOS deployment diagram

In the Figure 2.7 the ASOS deployment diagram is shown. Since ASOS does not have a web site, the main node is an **Application Server** that hosts the execution environment for the **ASOS Backend** component. The **Application Server** interacts with a **MongoDB Server** node, using the TCP/IP protocol. Moreover, it interacts with an external node which represent the different **Health Care Service**s, and with the execution environment of **Data4Help**.

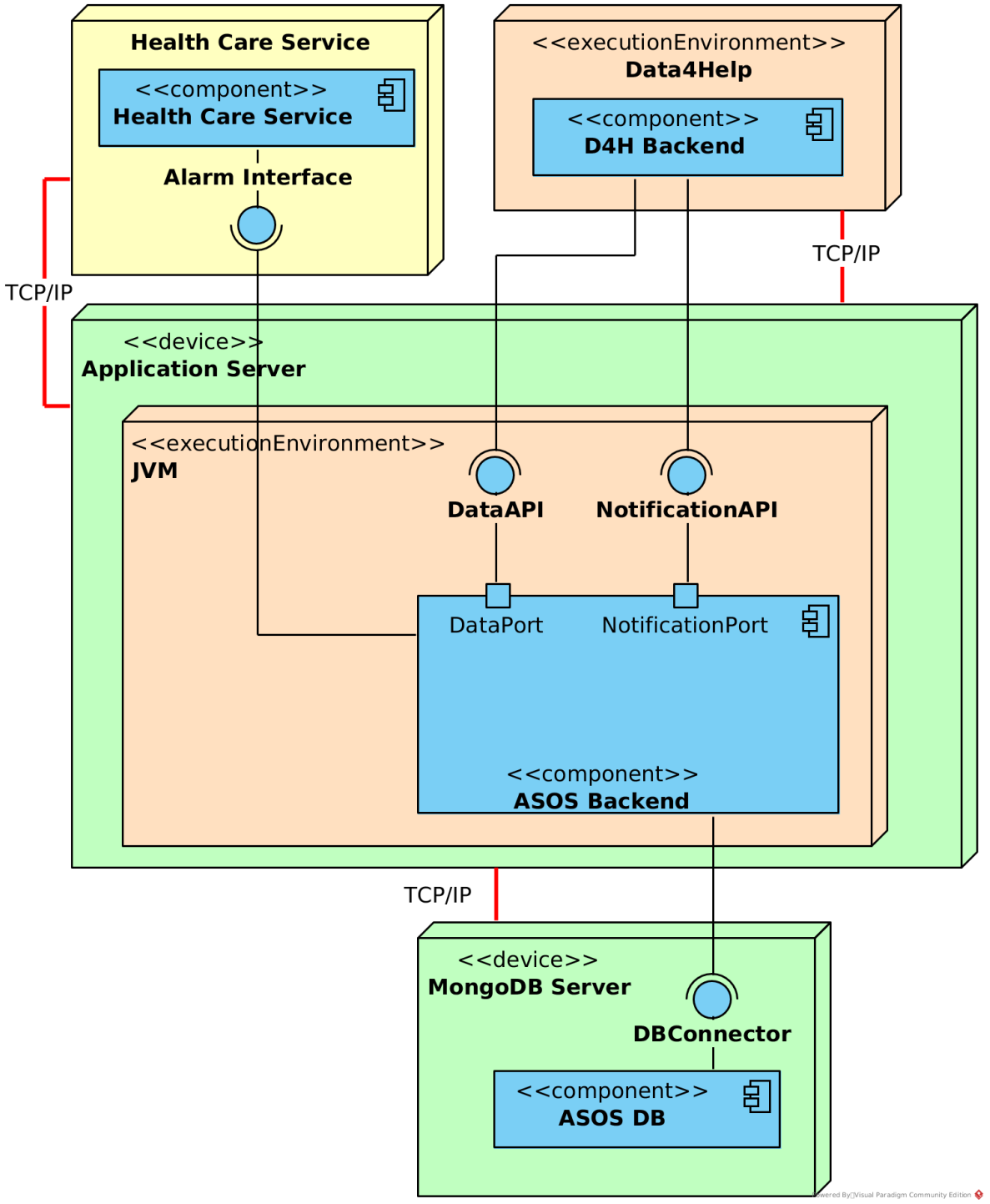

Figure 2.7: AutomatedSOS Deployment Diagram

2.4 Runtime view

You can use sequence diagrams to describe the way components interact to accomplish specific tasks typically related to your use cases

2.5 Selected architectural styles and patterns

Please explain which styles/patterns you used, why, and how

2.6 Other design decisions

Chapter 3

User interface design

The following mock-ups represent a basic idea of what the web application will look like after the first release. As when new customers visits the home page of TrackMe, they can read about the work that the company does, what services it offers, what benefits users can achieve by joining the community. In addition, they can buy the wearables and get more information, how to use them, what is TrackMe, more details about D4H, ASOS and T4R is provided here. The full home page can be seen in Figure 3.1.

Moreover, in the Figure 3.2 and Figure 3.3 can be seen the information page for D4H and ASOS respectively.

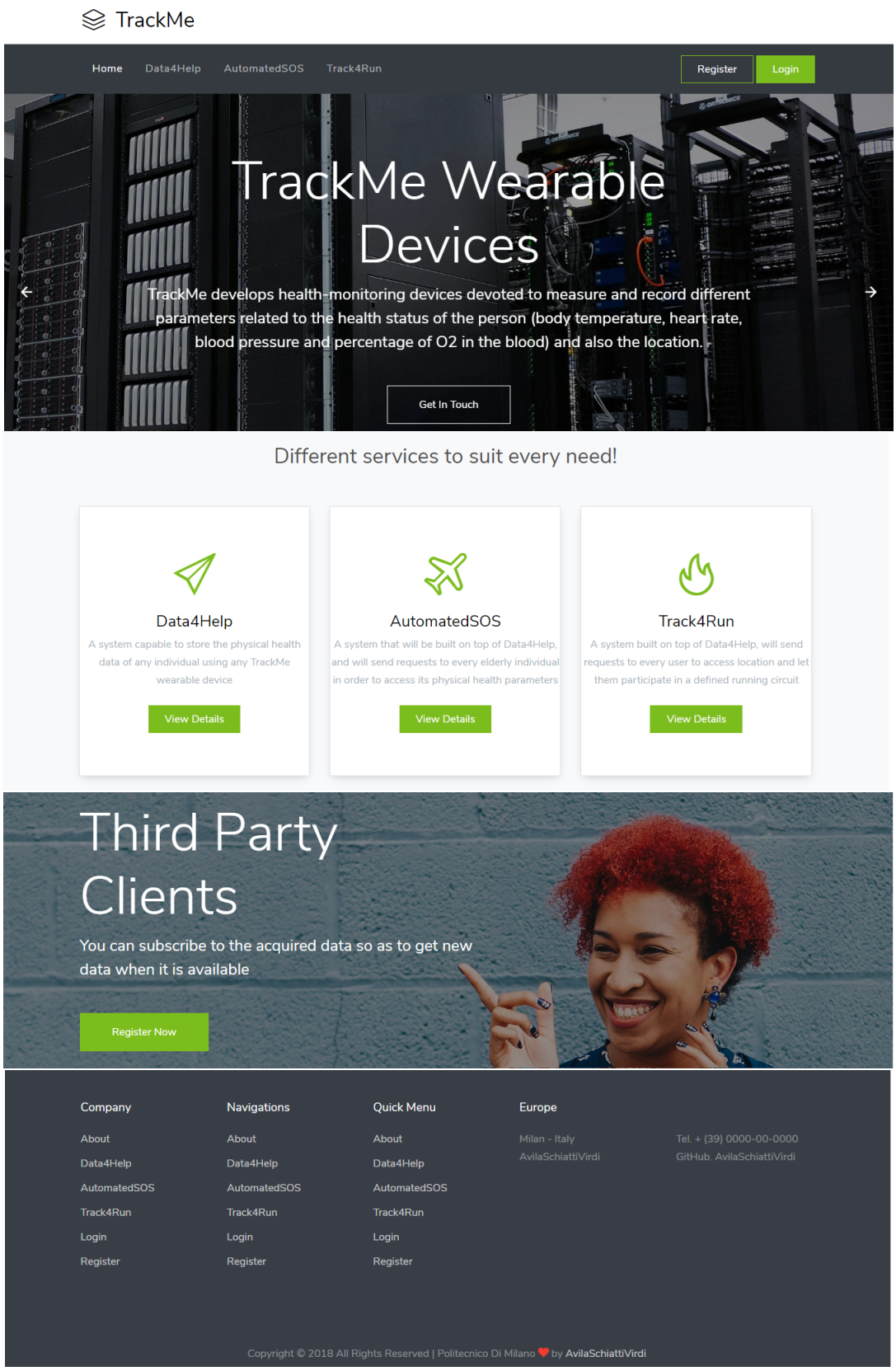

Figure 3.1: TrackMe's Home Page

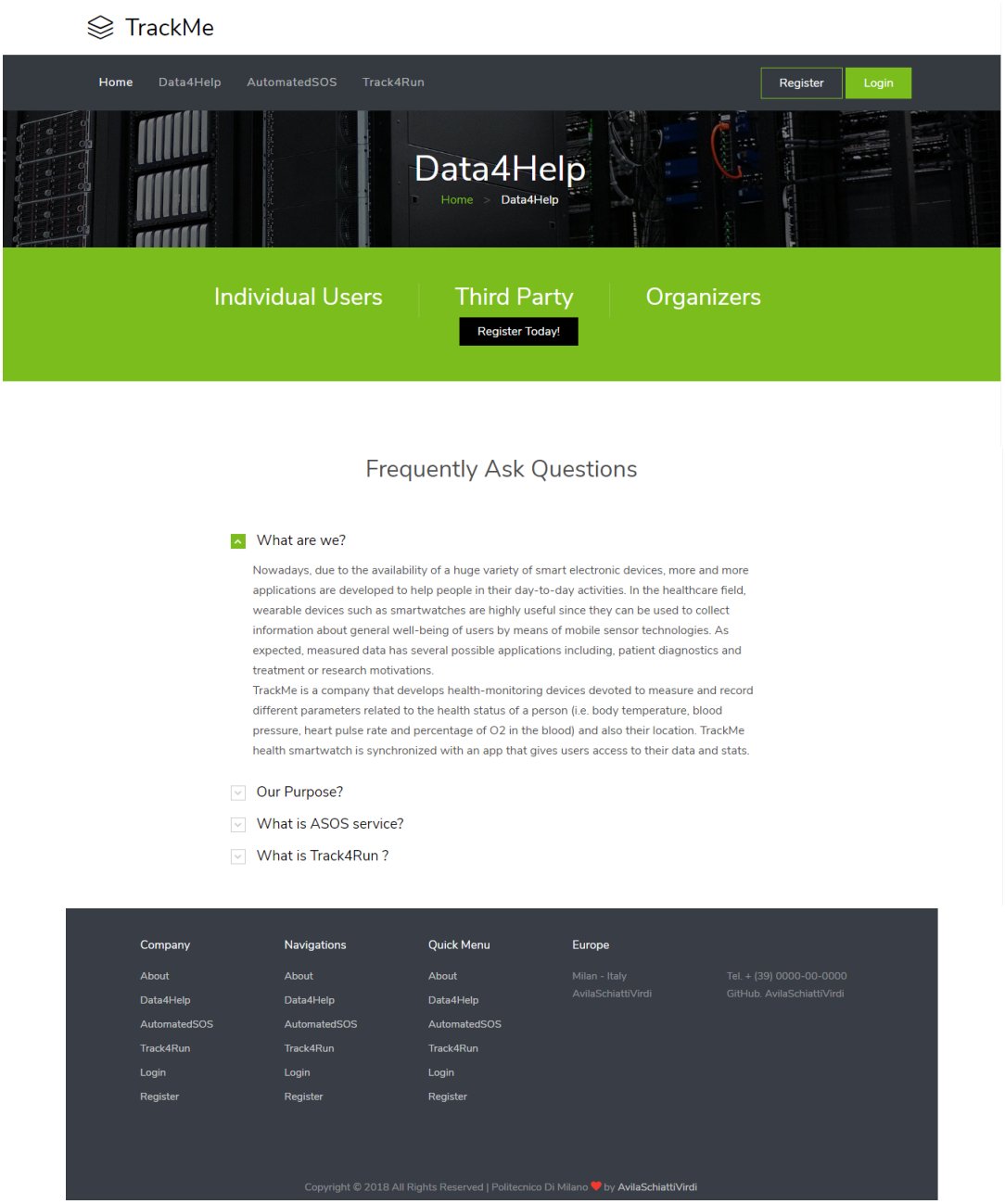

Figure 3.2: Data4Help Information Page

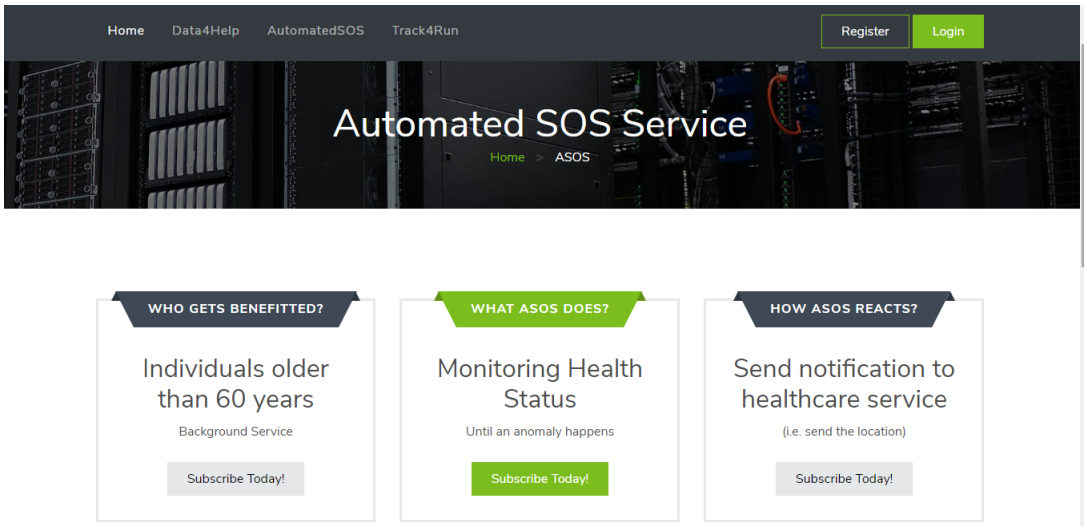

Figure 3.3: ASOS Information Page

In the Figure 3.4, can be seen the web page through which users can Login into the system (if already registered).

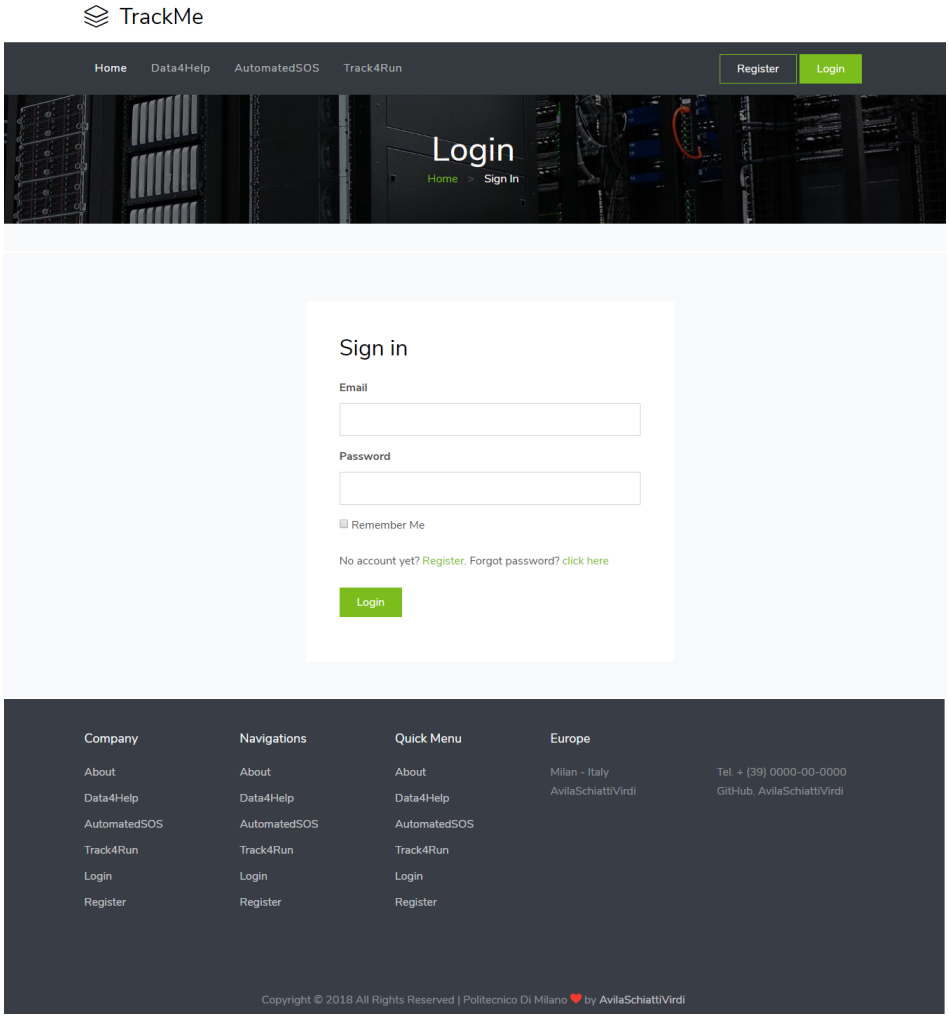

Figure 3.4: Login Page

And if not already registered into the system, they can register themselves through the Register web-page (Figure 3.5). There are separate register forms for the Individual users and for Third Party users.

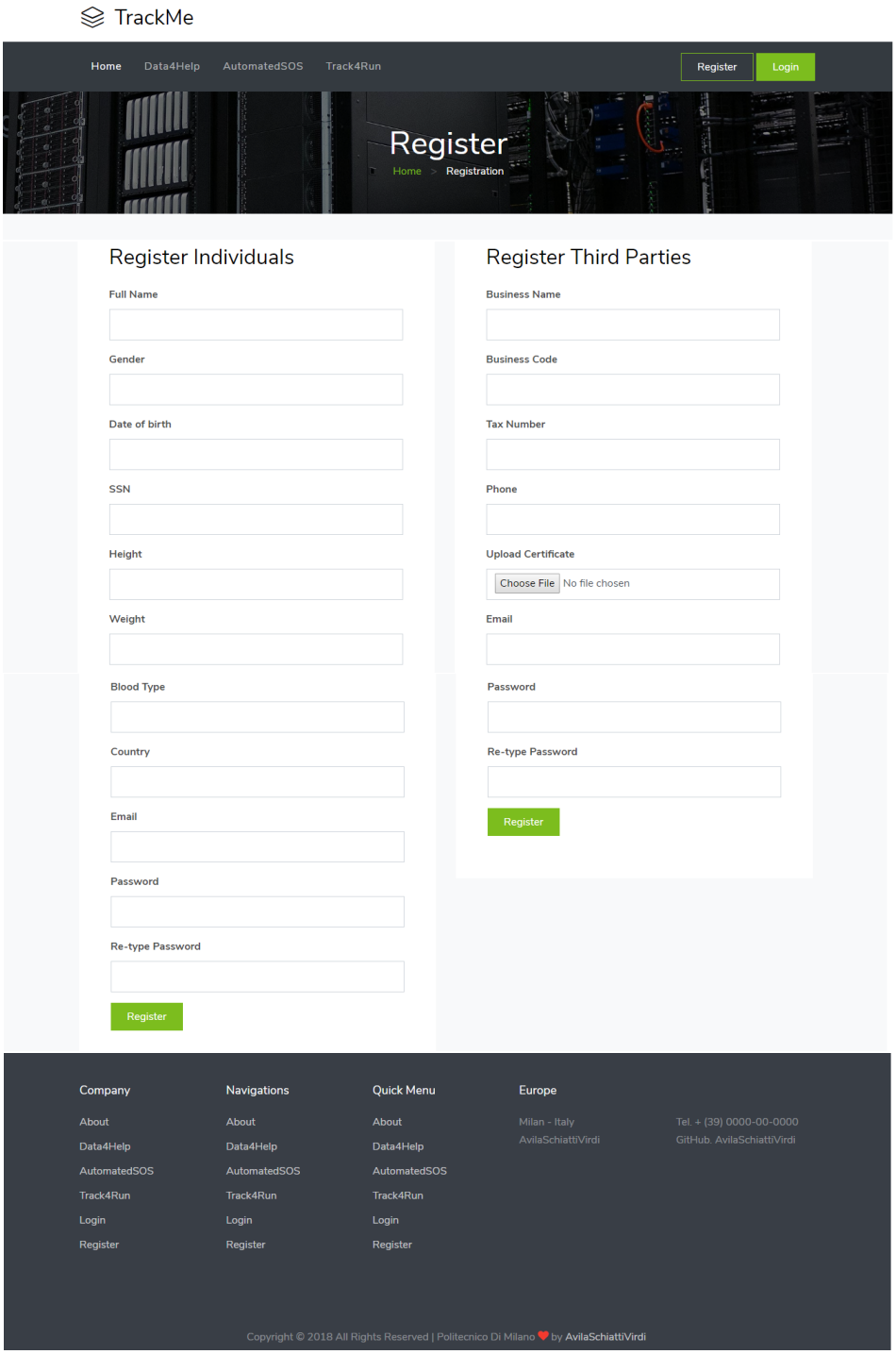

Figure 3.5: Registration Page

Chapter 4

Requirements traceability

Explain how the requirements you have defined in the RASD map to the design elements that you have defined in this document.

Chapter 5

Implementation, integration and test plan

5.1 Requirements of implementation and testing

In order to start implementation process of the project, RASD and DD must be finished and available for all the teams of the project, so they will be informed about requirements and adopted design choices. A good analysis of each document is also required to be sure that implemented system will be secure and reliable.

Development Tools The Java programming language will be used to develop our system because it suits very well for communication among different platforms and devices. In addition, Java Enterprise Edition will be used to deploy our business functionalities and to build a reliable and powerful web tier to let users communicate with main server using a standard web browser or a Java application. Java Enterprise Edition offers also an enhanced version of JMS services useful for communication between client and server. Spark Framework will be used as it is built for productivity. Spark Framework is a simple and expressive Java/Kotlin web framework DSL built for rapid development.

Testing Tools In order to achieve all the testing purposes we will use different testing tools:

- For unit testing of each component we will use *JUnit* framework. It's a good framework offered by Oracle itself so it's well test and supported. It can test each functionality of component into an isolated environment. It can be used also to test some little interaction between components.
- *Mockito* tool will be used as it is a No expect-run-verify technique. Mockito is a mocking framework that tastes really good. It lets you write beautiful tests with a clean and simple API. Mockito doesn't give you hangover because the tests are very readable and they produce clean verification errors.
- *Arquillian* tools may be used during integration testing: it offers different features that tests very well Java Enterprise components and containers. It also provides

features for debugging servers functionalities.

- *Grinder* software may be used to make load tests onWeb Server.
- *HammerDB* may be used to test functionalities and performance of Database server.

5.2 Implementation Strategy

Implementation process will follow the bottom-up strategy: we plan to implement first single component isolated. Once at least 60% of a component will be developed, its unit testing phase can start.
Using *Drivers* for tests allows different developer teams to complete its own work. Using this method we can be sure that component will be more reliable, so it will be more easy to reuse them into different part of the system.

The implementation of the *TrackMe* system will be done module by module and component by component. The order in which it is carried out depends on a number of factors like the complexity of the modules and services, the dependence of other modules on the component being implemented and to the system as a whole, and it should also take into account the possibility of discovering flaws with the proposed design. The later should be dealt in a way that, if such an unfortunate event does happen, the flaws should be found and corrected as soon as possible, to limit the cost of the change of design. Identify here the order in which system plan to implement the subcomponents of your system and the order in which system plan to integrate such subcomponents and test the integration.

5.2.1 Implementation order

Due to modularity of system components and to the adopted strategy there are no constraints on implementation order of components. But due to some critical aspects of a few components we want to prioritize implementation of those components. The most critical components of the system that could take more than others to be implemented may be:

1. Data4HelpWebService, Track4RunWebService
2. LoginService and SignupService
3. SearchManager, AuthenticationManager, DBManager, DataHandler
4. RequestService, SubscriptionService, NotificationService

5. HealthCareService, HealthCareConnector, AutomatedSOSDB

6. UserService, TokenDB, Track4RunDB

(note that by specifying the names of interfaces of components, we are also considering the concrete implementations, in whichever number they exist)

The Data4HelpWebService is proposed to be the first component that is implemented because all parts of the application server will be using some element of it and its role in allowing some service to communicate with the DBMS in the DBManager component is crucial to the whole application. The component releasing process should follow the integration order that will be described into the next section.

5.3 Integration and Testing

5.3.1 Entry criteria

The integration of components and its testing should start as soon as possible, but before they can commence, some conditions must be met. First of all, the external services and their APIs that are going to be used in the application should be available and ready. This applies to the already mentioned services, to the DBMS and the server on which it will be running on.

Next, the modules which are being integrated should have at least the operations concerning one another created, if not completed completely. The operations that have been developed should pass the unit tests in order to be sure that the components are working fine on their own and that if an integration test fails, the problem lies in the in the integration itself.

5.3.2 Integration test strategy

The main goal of integration process is to avoid as much errors as possible at each step of the process, so the system will incrementally integrate components as soon as they are completely developed and released.

Bottom-up design will be adopted for most of the integration process: at the beginning only components that have less bindings to other components or which can work without other component will be integrated. In this way we can obtain feedbacks about system

functionalities as soon as components are released and in addition we can parallelize integration of different subsections of the system.

For the most critical components or for more complex system parts we will use instead *Critical Modules strategy*: components that fit very well for *Critical Modules* strategy are those in Data4Help subsystem, because the AutomatedSOS and Track4Run will use the Data4Help system and are the most frequent interaction performed into our system. For this reason *Bottom-Up* strategy will be applied only once Data4Help subsystem will be fully integrated using *Critical Modules*.

5.3.3 Integration order

In this section, the list and order of every integration that is performed is shown. As already stated, the integration will be performed from the bottom-up.

It should be noted that there will be no explicit integration of the Data4HelpWebService, Track4RunWebService with any of the other components. This is because the nature of the component, the extent of the usage and dependency of other components on it and the implementation plan, that clearly states that the Data4HelpWebService will be the first part that is implemented, mean that the integration itself is already being done during the implementation phase of the depending components and its correctness will inherently be tested by the unit tests of each component.

1. As written in the introduction, at the beginning we will integrate the *Data4Help* subsystem, composed by these components:

   - Data4HelpWebSite
   - LoginService and SignupService
   - RequestService and NotificationService
   - SubscriptionService

   Integration tests will check cases like addition of new user to database, accessing the system with correct credentials, the consistency of making request, the reachability of all notifications to accept/reject request in the dashboard, the modification of subscription chosen by the user (if he wants any).

2. Then we could start to implement part of the *AutomatedSOS* subsystem:

   - HealthCareService (*External component*)

- HealthCareConnector
- AutomatedSOSDB
- ThresholdCollection, UserCollection, HealthCareCollection

This integration step will check the communication of the Data4Help system with external sources HealthCareService when the vital signs are out of the normal range or the ThresholdCollection.

3. Then we could start to implement part of the *Track4Run* subsystem:

   - Track4RunWebService
   - Track4RunDB
   - RunCollection
   - TokenService, OrganizerCollection and ParticipantCollection

   This integration step will check the communication of the Data4Help system with Track4RunWebService and when RunCollection is modified, check the integration with the existing requestService and notificationService of Data4HelpWebService.

4. After that, other components can be joined together into small subsystems to test their interactions:

   - *AuthenticationManager* and *LoginService* : their interactions must not brake the consistency of Account information and they have to check only authorized users can access information
   - *SearchManager, RequestService* and *SubscriptionService* : system will check consistency between search into a request and subscription of the same user
   - *DataHandler, DBManager* and *HealthCareConnector* : system will continuously check latest data and verify with the ThresholdCollection, if varies, notifies HealthCareService through HealthCareConnector of the same user
   - *RunCollection* and *ParticipantCollection* : system will check consistency between run created and allows users participated in the same run

5. After that we can start to integrate user interactions:

   - *UserInterface* and *RequestService* : tests sending request for specific or anonymized search through different user interfaces
   - *UserInterface* and *NotificationService* : tests will check asynchronous communication between user and server for the dispatch of event messages

- *UserInterface* and *RunCollection* : tests will check asynchronous communication between user and runs available for the live view on the map of participants
- *UserInterface* and *HealthCareService* External Resource : will tests the communication with external services of User Interface and Health-Care Service and the synchronization of Data with threshold values when user's data is out of range, notifies using external alarm interfaces

Chapter 6

Effort spent

Team Work	
Task	**Hours**
Planning Architecture	3
Choosing Pattern	2
Algorithm Decisions	8
Checking document	4
Total	17

Table 6.1: Time spent by all team members

Individual Work					
Diego Avila		**Laura Schiatti**		**Sukhpreet Kaur**	
Task	**Hours**	**Task**	**Hours**	**Task**	**Hours**
X	X	Layout	X	X	X
Total	X	**Total**	X	**Total**	X

Table 6.2: Time spent by each team member

Chapter 7

References

- Requirement Analysis and Specification Document: AA 2017-2018.pdf". Version 1.0 - 26.10.2017

- Henriksen, A., Haugen Mikalsen, M., Woldaregay, A. Z., Muzny, M., Hartvigsen, G., Hopstock, L. A., Grimsgaard, S. (2018)
  Using Fitness Trackers and Smartwatches to Measure Physical Activity in Research: Analysis of Consumer Wrist-Worn Wearables. Journal of medical Internet research, 20(3), e110. doi:10.2196/jmir.9157.
  Retrieved from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5887043/

- IEEE. (1993). IEEE Recommended Practice for Software Requirements Specifications (IEEE 830-1993).
  Retrieved from https://standards.ieee.org/standard/830-1993.html

- Sloane, A. M. (2009). Software Abstractions: Logic, Language, and Analysis by Jackson Daniel, The MIT Press, 2006, 366pp, ISBN 978-0262101141.