POLITECNICO
MILANO 1863

Software Engineering II project

Academic Year: 2018/2019

Design Document

Version 1.0 - 10/12/2018

Davide Rutigliano - 903616

Claudio Ferrante - 903417

Davide Matta - 920349

Contents

List of Figures

1 Introduction

1.1 Purpose

The Purpose of this Design Document is to provide the precise implementation details required to satisfy the requirements specified in the RASD [1]. It is assumed that the reader has read the RASD before this document.

The purpose of this paper is also to point out the main components, their interfaces and their behaviours, show the high level architecture and its interaction with architectural components and to identify style and design patterns that shall be used in deployment of the application. This Document will provide detailed explanation of each functionality through components and sequence diagrams also used to explain to programmers the overall structure of the system in a precise and implementation-oriented way.

Furthermore this document provides also an *Implementation & Test* plan in order to give to the developer a precise idea of the deployment process.

1.2 Scope

The scope of this application is to provide health monitoring services for both young and old people who needs to keep track of their personal data in order to to keep their health safe.

TrackMe allows Third-Party to access Individual's health data exploiting the functionality of *Data4help* service: Third-Parties can make individual requests that the Individuals can accept or reject, or group requests handled directly by TrackMe that approves them if it is able to properly anonymize the requested data. For sake of simplicity, we already assumed that TrackMe will accept any request for which the number of individuals whose data satisfy the request is higher than 1000. The application may also allow individual users to connect external devices such as smart-watches, specific pathology's monitoring devices or other types of external devices able to monitor health parameters with BT or NFC connection.

In addition, the system permits to enable the *Automated-SOS* service that guarantees that the GPS position of the user who has enabled *Automated SOS*, can be send to the nearest ambulance available, when user's health parameters overcome some critical thresholds. This procedure takes place thanks to an ambulance dispatcher connected to the Third-Party which has enabled automated sos service and has been selected by that individual to provide this service.

Moreover, through the *Track4Run* service, *TrackMe* allows *Organizers* to *create* a new run: a competition in which other individuals can either participate as *athletes* or watch as *spectators*. Additionally, this feature guarantees other services for keeping track of athletes progresses during the run, providing an interactive map of the run, with runners position on it.

1.3 Definition, Acronyms and Abbreviations

1.3.1 Keywords

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and

"OPTIONAL" in this document are to be interpreted as described in RFC 2119 [2].

1.3.2 Definition of Terms

This document uses several terms that might be more loosely used elsewhere. These terms are defined here as they will be used later on in this document.

Subscribed User an Individual for which one Third Party whom asked to subscribe to the data has accepted the Third Party's request

External Device a device such as a smart-watch or similar, capable of data acquisition

Run a competition organized and managed by an *Organizer* to which *Athletes* and *Spectators* can enroll as participant or watchers

Run Started a competition with at least two *Athletes* enrolled that the *Organizer* has started

Run Created a competition with any number of *Athletes* enrolled

Track A route where athletes will compete in a specific *Run*

Map A *Track* that display all *Athletes* position in a specific *Run*

Accident An event triggered when the monitored user's parameter overcome certain thresholds

TrackMe the *"system to be"*

Data4Help a data monitoring service provided by *TrackMe*

Automated-SOS an SOS service built on top of *Data4Help*

Track4Run run management service offered by *TrackMe* application

Credential as used in this document, is a combination of both username and password used by an *User* to authenticate him/herself during the Log-in phase

Personal Data users' data of different kind either for Individuals (name, surname, age, etc.) or for Third-Parties such as Organization name, number of employees or VAT number

1.3.3 Acronyms

DD Design Document

UML Unified Modelling Language

SQL Structured Query Language

MVC Model View Controller

MVP Model View Presenter

REST REpresentational State Transfer

API Application Programming Interface

RASD Requirement Analysis and Specification Document

UX User eXperience

ER Entity Relationship

DAO Data Access Object

DTO Data Transfer Object

1.3.4 Definitions

Business Logic Layer another term to intend the Application and Service Layers together

Controller a component of the application layer that is responsible of handling events from the client

Service part of the business logic, executes the core logic of a specific function or set of functions

Repository is the component that handles query methods to the database

Data Access Object sometimes called *Entity*, is the object used by Repository to map data to a specific object thus, it should encapsulate the logic for retrieving, saving and updating data in the data storage

Data Transfer Object also called *Bean* is the representation of a DAO and it is used to transfer the data between the client and server

1.3.5 Abbreviation

REQ-n Requirement number n from the Requirement Analysis document

1.4 Reference Documents

[1] Davide Matta Davide Rutigliano Claudio Ferrante. "Requirement Analysis and Specification Document v1.0". In: (2018).

[2] Scott Bradner. "Key words for use in RFCs to Indicate Requirement Levels". In: (1997).

[3] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*. Vol. 7. University of California, Irvine Doctoral dissertation, 2000.

1.5 Document Structure

- **Introduction**: this section is aimed at giving to the reader an overview of the document;
- **Architecture Design**:
  a) *High Level Components and their Interaction*: gives a global view of the components of the application and how they communicate;
  b) *Component View*: gives a detailed description of the components of the applications and includes ER diagram;
  c) *Deployment View*: shows UML deployment diagram and explanations;
  d) *Runtime View*: presents UML sequence diagrams in order to show the event flow of the different tasks of the application;
  e) *Component Interfaces*: presents the interfaces between the components through UML component diagrams;
  f) *Selected Architectural Styles and Patterns*: explain the architectural choices done during the design of the application and design patterns;
  g) *Other design decisions*;
- **User Interface Design**: this part presents mockups and user experience explained via mockups and UX diagrams;
- **Requirements Traceability**: this section aims to explain how the decisions taken in the RASD are linked to design elements in the Design Document.
- **Implementation and Test plan**: provides full information about the deployment strategy advised to developers in order to follow design decision taken in this document.

2 Architectural Design

2.1 High Level Components and their Interaction

The system is deployed on three different layers: the Client layer, the Server layer and the Database layer.

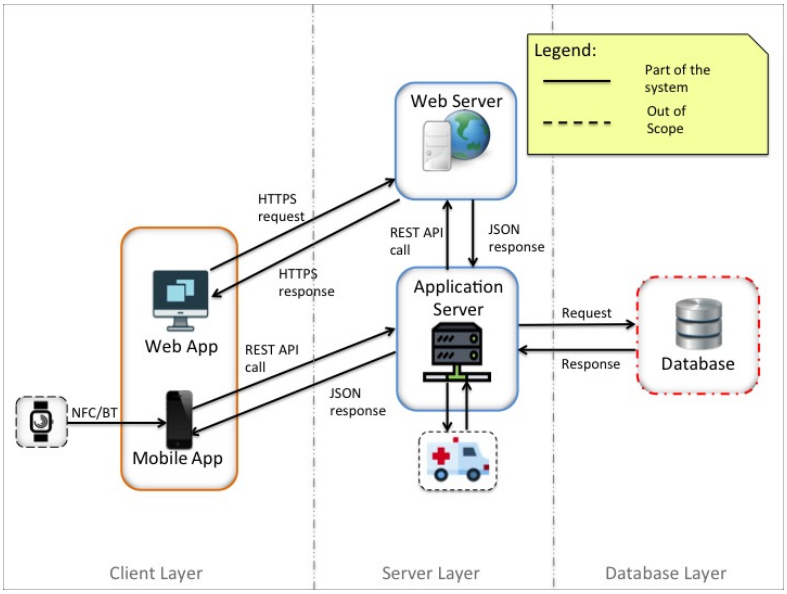


Figure 1: High Level interaction diagram

- The *Mobile Application* may be connected with an external device such as smart-watches, specific pathology's monitoring devices or other types of external devices able to monitor health parameters with BT or NFC connection;
- The *Application Server* should be able to communicate with dispatchers in order to send them individual's position and notify them in case of accident.

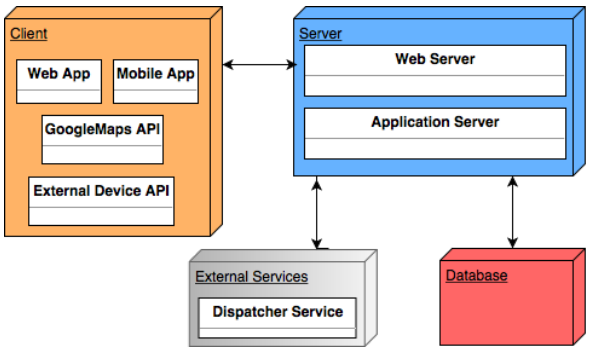


Figure 2: High Level components diagram

2.2 Component View

2.2.1 Application Layer (Server Layer)

Authenticator Controller: receives guests' registration and login requests and forward them to the lower layer and also handles users' credentials and profile data management.

Individuals Data Controller: is responsible of individuals' personal data management.

Request Controller: allows to generate individual and group requests. Furthermore it is encharged of the management of incoming request for individuals and of outgoing requests in case of third party users.

Automated-Sos Controller: allows both individual and Third-Party to enable the Sos service. In particular, it allows Individual to chose a Third-Party among the ones that have enabled the Sos service.

Run Controller: handles the creation of a run and allows both Athletes and Spectators to respectively enroll and watch run. This component also handles for organizer definition of the path of the run and, for spectators, watching that run to see athletes position and the predefined path on an interactive map.

2.2.2 Service Layer (Server Layer)

Authenticator Service: handles all the logic related to the login and registration functions. This module receives data from the *User Session Controller*.

IndividualData Service: is the first core component of *Data4Help* logic, it handles all Individual's health data and positions.

Request Service: is the second core component of *Data4Help* logic, it handles all the requests coming from the Request Controller and manages all the Users' private and public data. It validates group and individual requests and return the responses from *Individuals*.

Automated-Sos Service: monitors Individuals' data who activated the service and is able to notify the *Dispatcher* in case of accident.

Run Service: handle runs for creation, enrollment and watching.

2.2.3 Persistence Layer (Server Layer)

Health Data Repository: saves and get individuals' health data from the database.

Position Repository: saves and get individuals' position from the database.

Individual Repository: saves and get individuals' data from the database.

Third Party Repository: saves and get third parties' data from the database.

Request Repository: saves and get only request accepted by *Individuals*.

Automated-Sos Repository: saves users' enabling/disabling of sos services and further accidents data.

Dispatcher Repository: contains dispatchers' data.

Run Repository: saves and get created runs, enrollment and watching information.

2.2.4 Client Layer

Google Maps API: used by the client application to create maps through Google Maps services.

External Device API: allows connection with and data acquisition from the Individual external device from a common smart-watch to a specific pathology monitoring device.

2.2.5 External Services

Dispatcher API: used by the Sos service to communicate with ambulances dispatcher.

2.3 Component Interfaces

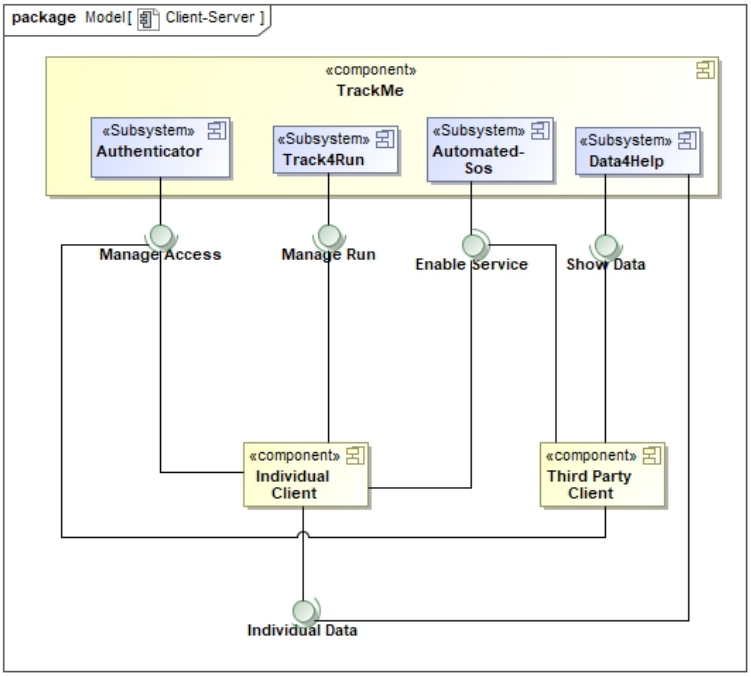


Figure 3: TrackMe Component Diagram

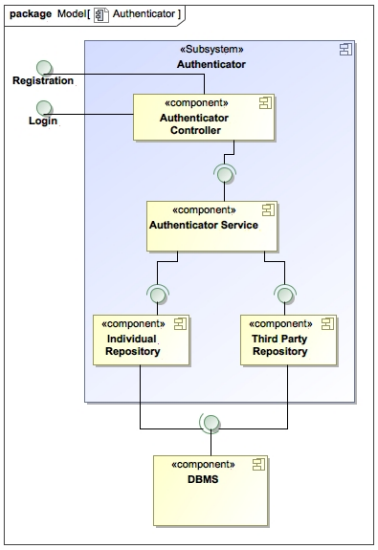


Figure 4: Authenticator Component Diagram

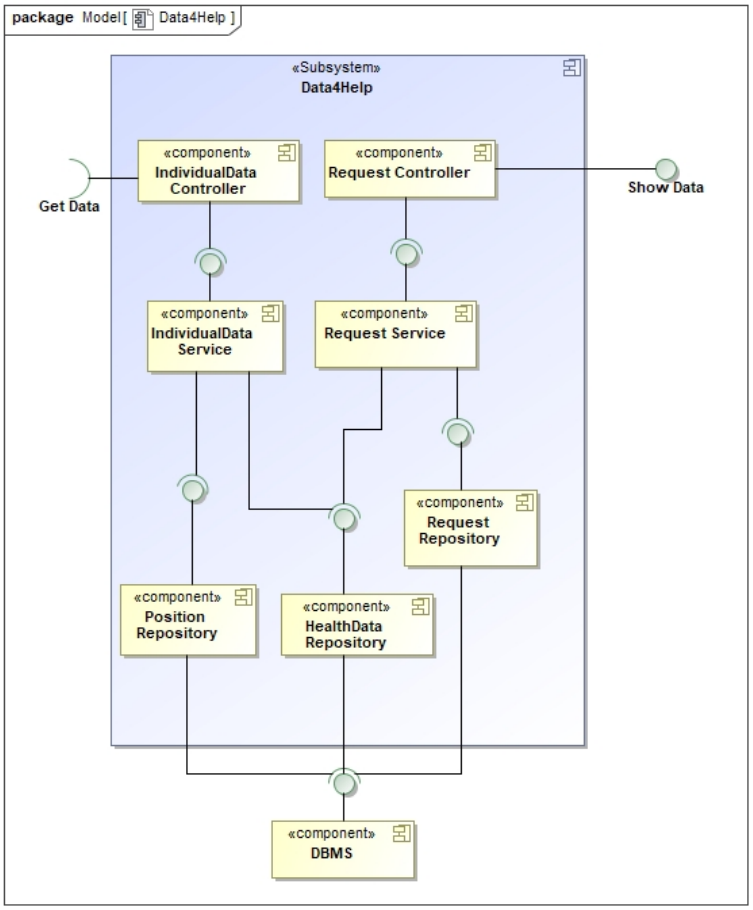


Figure 5: Data4Help Component Diagram

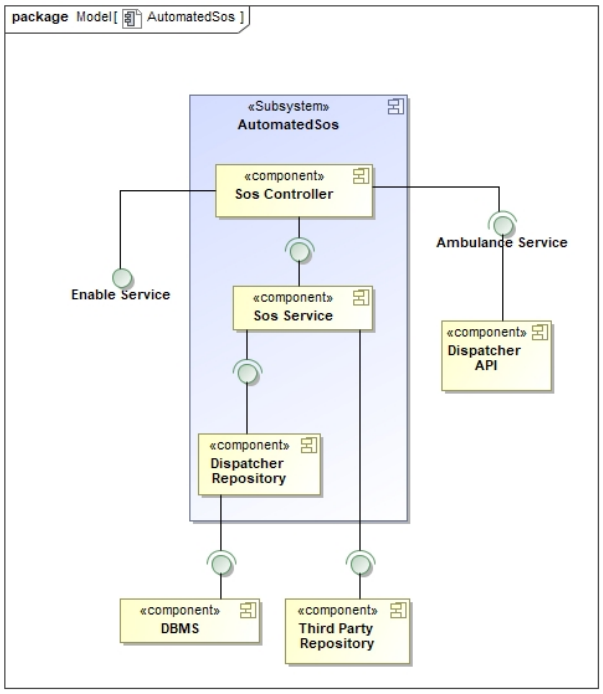


Figure 6: AutomatedSos Component Diagram

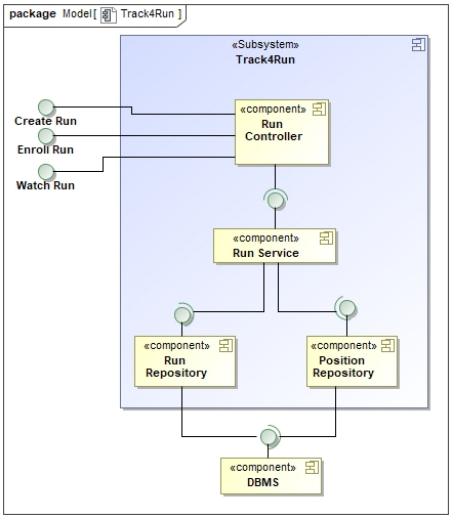


Figure 7: Track4Run Component Diagram

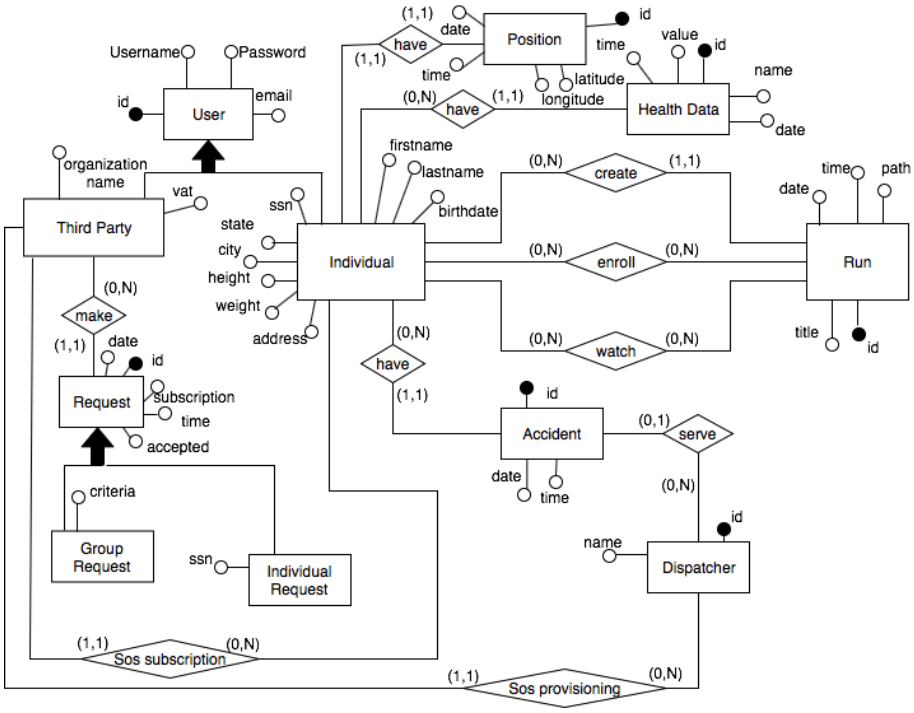


Figure 8: ER Diagram

2.4 Deployment View

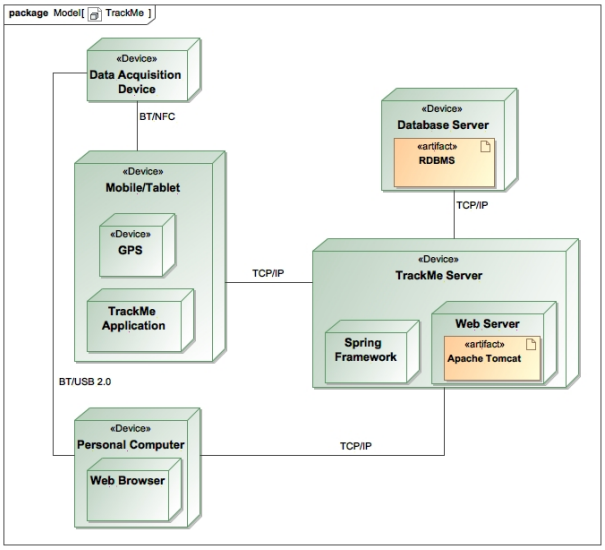


Figure 9: TrackMe Deployment Diagram

2.5 Class Diagram

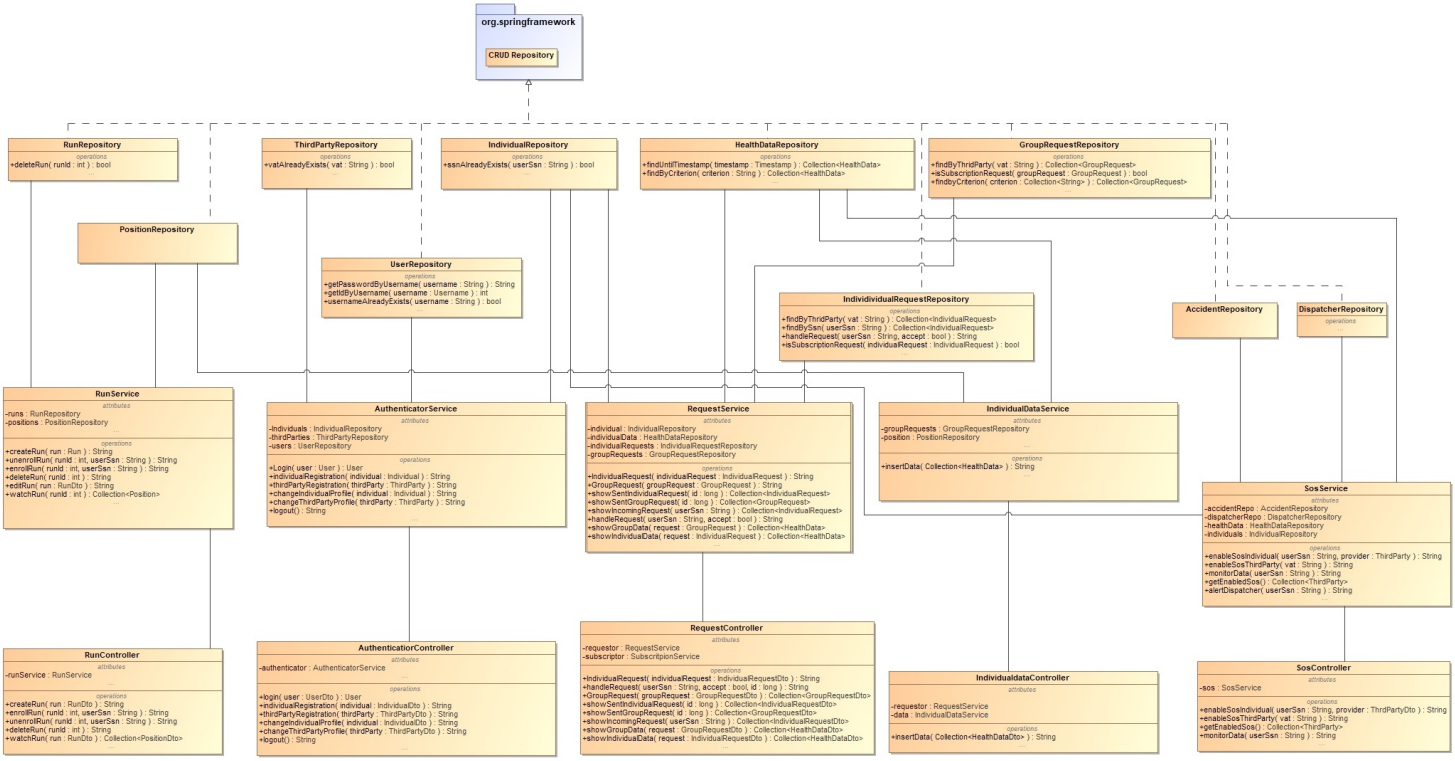


Figure 10: Class Diagram

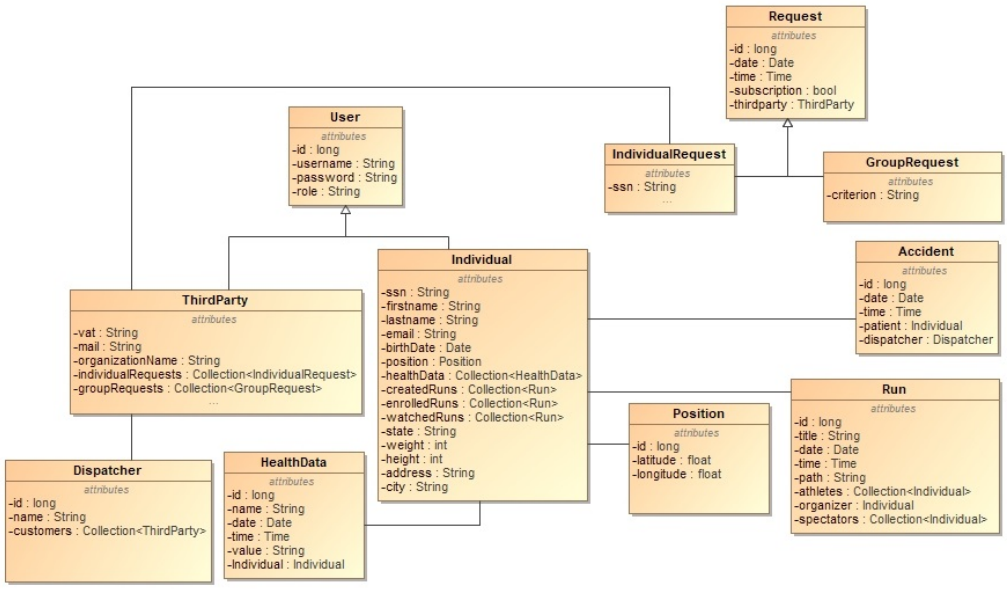


Figure 11: Class Diagram (Entities)

2.6 Runtime View

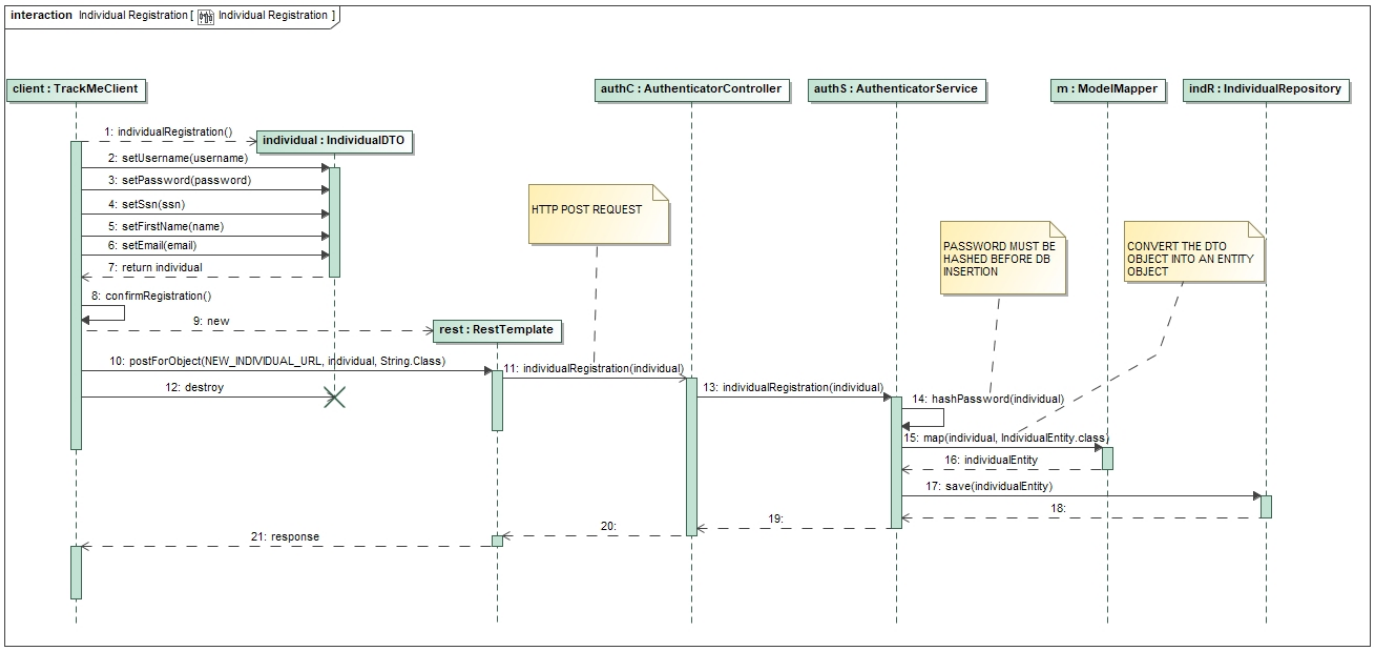


Figure 12: Individual Registration Sequence Diagram

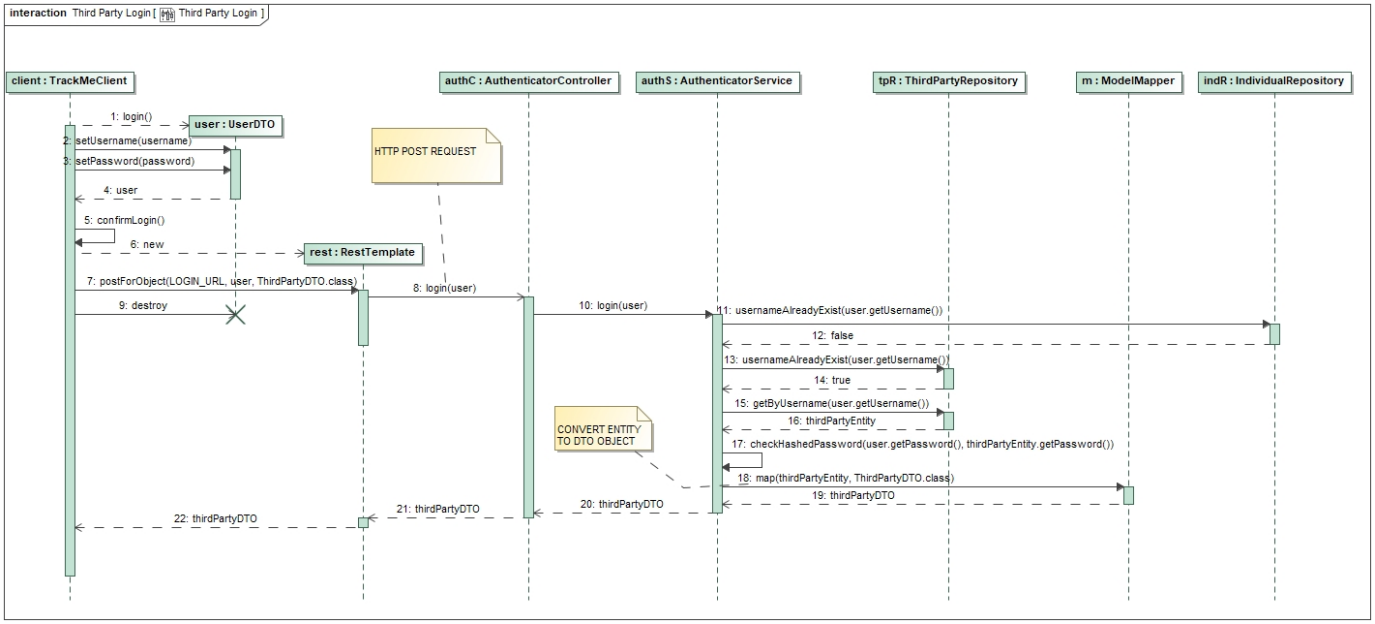


Figure 13: Third Party Login Sequence Diagram

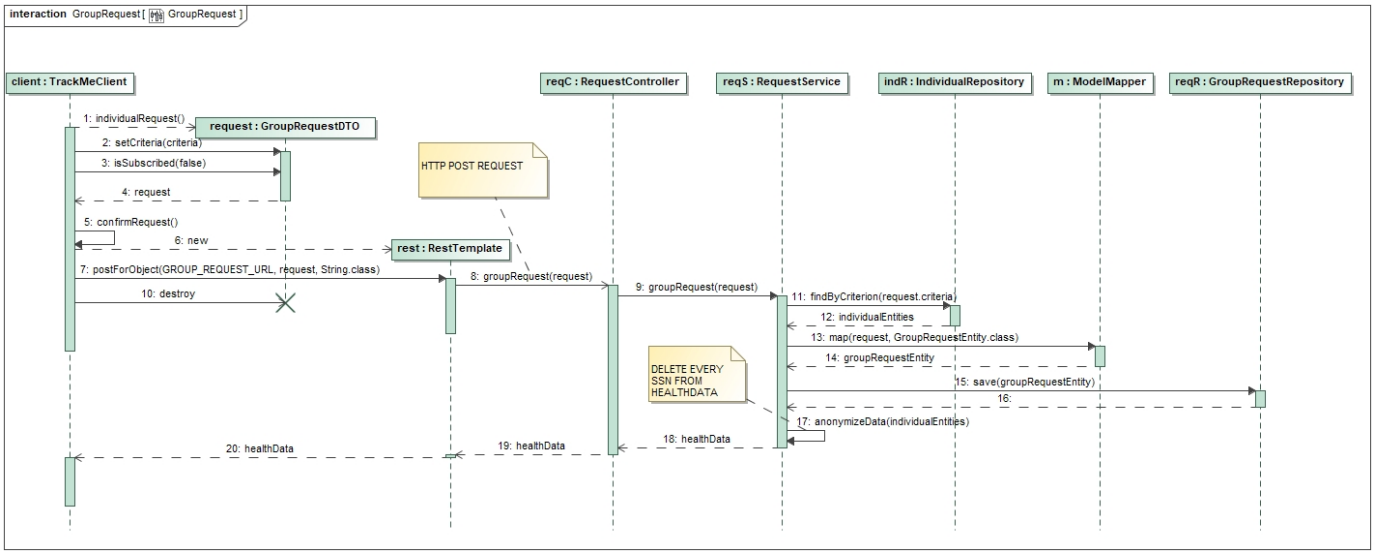


Figure 14: Group Request Sequence Diagram

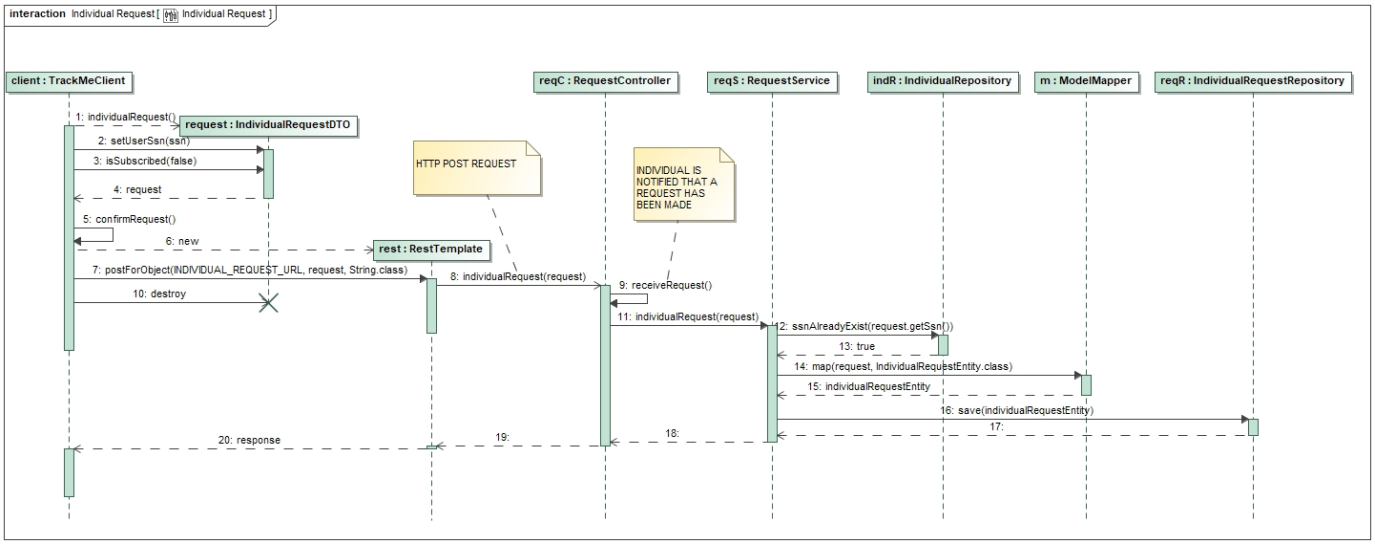


Figure 15: Individual Request Sequence Diagram

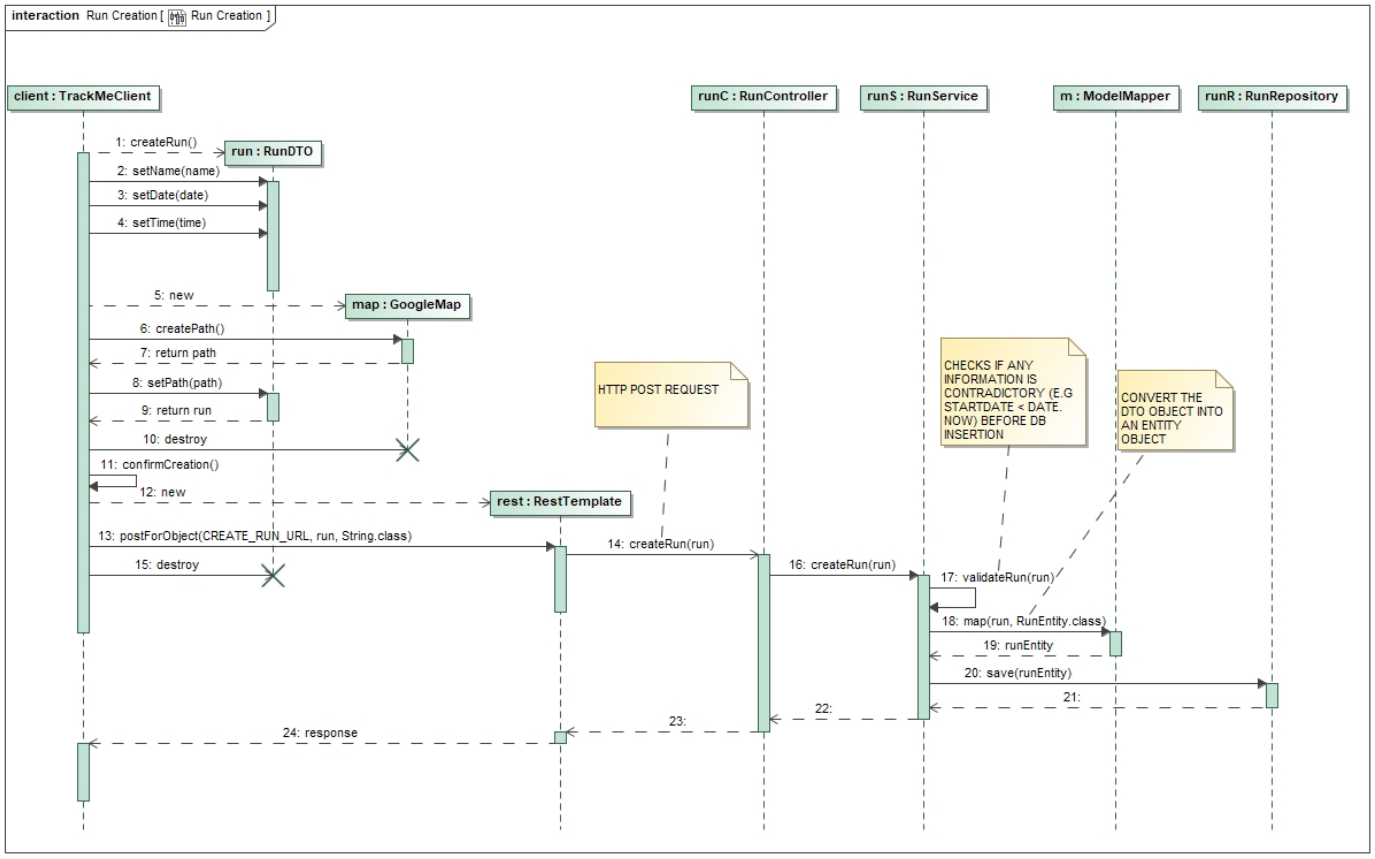


Figure 16: Create Run Sequence Diagram

2.7 Selected Architectural Styles and Patterns

2.7.1 Architecture

RESTful Service: to ensure a better scalability and reliability the best decision is to use a REST architecture [3]. Also performance is an important factor, and this is achieved due to requests following the simple HTTP protocols. Statelessness ensure that is also very simple to add parallel servers and make a distributed network: scalability is improved because the server doesn't have to store the state (no session or sticky session). Clients need to provide every information necessary in each request: the server doesn't have to manage resources across requests.

Three Tier Architecture: as showed previously in this document, the layered architecture pattern allows to separate the presentation from the business logic and to realize a thin client with only the user interfaces and input forms and a more complex server that handles all the software services. The overall architecture is structured as a three tier architecture: the *Client Layer*, the *Server Layer* and the *Database Layer*.

Layered Server: The main component of the system is the Server Layer, that it's also deployed on three different layers: the **Application Layer**, the **Service Layer** and the **Persistence Layer**. These layers are respectively mapped on the three main components of the software application: controllers, services and repositories.

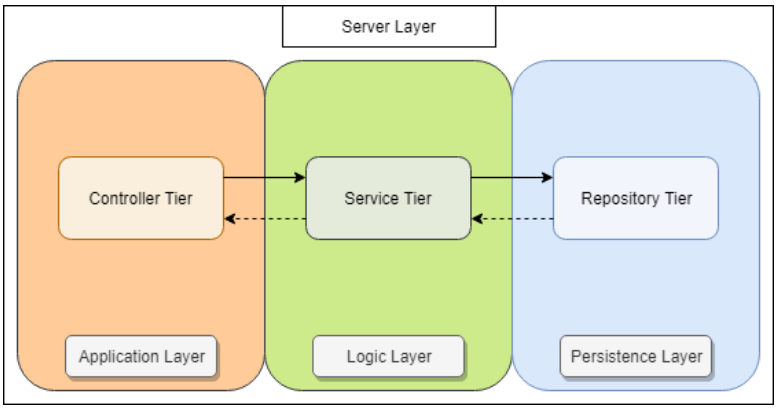


Figure 17: Server Layer diagram

The **Application Layer** communicates directly with client browser and mobile application, while the **Service Layer** manages all the application logic: the latter is invoked by the Application Layer.
Lastly the **Persistence Layer**: here the application has all the database management logic and is responsible of the communication with the DBMS. This layer is invoked by the Logic Layer.
This was done to add **separation of concerns**: each layer is specialized in a specific activity.

Model View Presenter (MVP): in this architecture there are three entities: the *Model* which essentially maintains all the data, the *View* that is responsible of showing the model's data and routes user commands to the presenter, and finally the *Presenter* that is responsible of handling the data access from *Model* and to format it to display in the view. This architecture is used in the *Client Layer*.

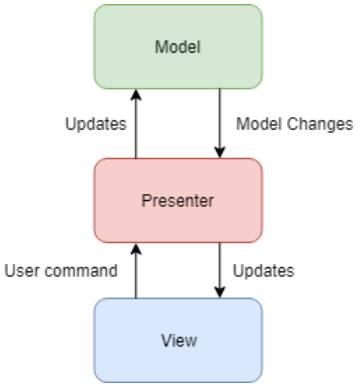


Figure 18: MVP Architecture

2.7.2 Design Patterns

Publisher-Subscriber: this is the most important pattern that is used in the application. It is used for almost anything from handling *Data4Help* requests to send athlete positions to spectators through *Track4Run*.

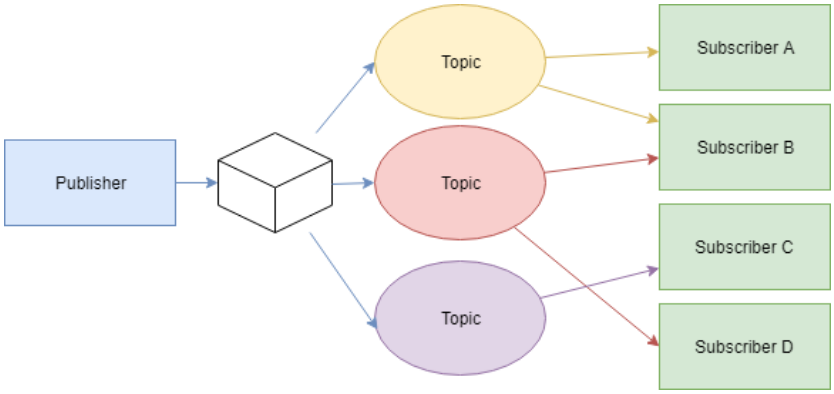


Figure 19: Pub/Sub Pattern

This is necessary for performance reasons: *TrackMe* is a data-intensive real-time system, so *polling* is not a viable strategy in this case. *WebSockets* are clearly the best solution and this is why this pattern shall be used. Suppose a scenario where a Third Party wants to subscribe to the data of an Individual named Pippo. When Pippo registers to the application, *TrackMe* creates a *topic* associated with him. The Third Party becomes

a subscriber of that topic: every time Pippo (Publisher) upload his data, the associated topic is updated and all Third Parties subscribed to him (Subscriber) are notified.

2.8 Other Design Decisions

2.8.1 Client Localization

The management of the external device is a client only operation that will allow the application to get Individual's personal health data from the smart device and send them to the server in a lightweight format (e.g. JSON).

The client map of the run is *local* too: the run organizer will be able (through Google Maps API) to draw a path on an interactive map and send it to the server which will be able to decode the run path and store it as a string; also spectators will be able to use a map to see athletes positions and again, the map will be local too. The Server will provide only athlete positions with the associated track.

2.8.2 Client Native Application

In order to allow the user to exploit all the functions of *TrackMe*, the mobile app should be developed for native *Android/iOS*. This is necessary for performance reasons. *TrackMe* is a real-time system that relies heavily on data transfer where performance must be maximized; an **hybrid application** was considered a bad choice for the following reasons:

- Slower than native application;
- Have to work with a wrapper dependent on a third party platform.

2.8.3 Client Web Application

In addition, a web application may be further developed in order to allow customers to use *Data4Help* features also from their Personal Computers; *Automated-Sos* and *Track4Run* services will instead be available only on mobile devices.

2.8.4 Programming Languages and Frameworks

In this design document we assumed the developer is using Java as programming language and in particular is deploying the application server through Spring Framework. The rationale for this choice is that this framework fits very well the architectural style of the proposed software solution, has proprietary implementation of the Model View Controller design patter and of several APIs and is very suitable for both provisioning and consuming RESTful services.

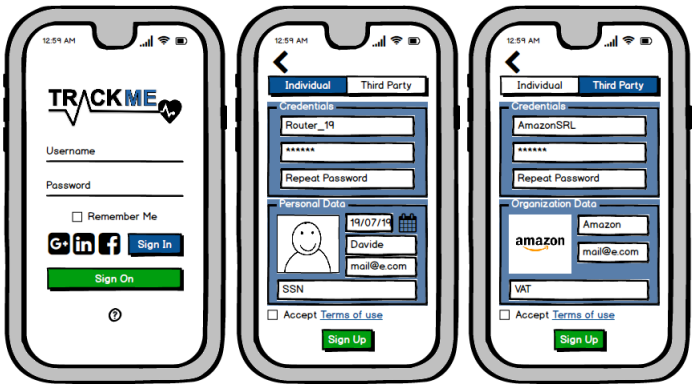
The choice of using Java and Spring is not a full restriction on the deployment process. In a decision to use a different programming language or to not use Spring Framework would occur, due to the proprietary implementation of several classes, such as the persistence API or the Spring MVC ones, some adaptions to the class diagram may be needed.

3 User Interface Design

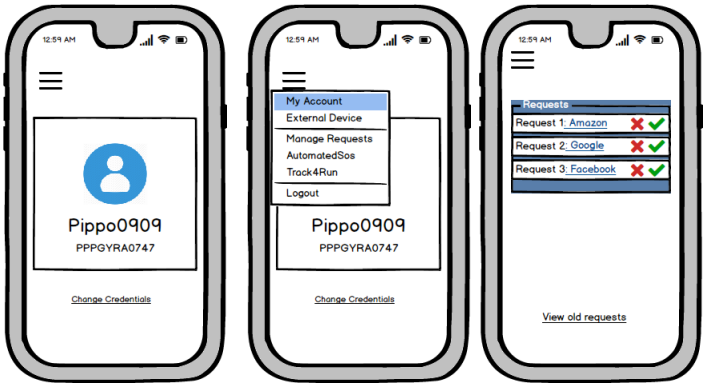
As specified in RASD there are two different user interfaces, one for mobile application and one for web browsers. In this specification we assume the web version of the application will be the same as the mobile application with reduced functions.

3.1 Mockups

The guest who access to the main page can sign-up to the application providing his credentials. Once clicked on the *Sign On* button the guest can decide whether registers as an Individual or as a Third-Party, swiping left and right to choose the wright option. Once registered, on the initial page, an user can sign-in in the application inserting Username and Password in the relative boxes and clicking on the *Sign In* button.



Once inside the application, the user is redirected to the home page, where he can see his account data. He can press the top-left menu and navigate from there. Here the user can access to services provided by *Data4help* and can enable *Automated-SOS* service too. Individual can also access to *Track4Run* and *External Devices* section. The Individuals can handle incoming request by accepting or rejecting them on the Manage Request page.



Third-parties can choose either individual or group request and can enable the option to subscribe to new data.

The Third- Party, form the Data4Help page, can also reach the view data section: here she can see and plot the data of sent requests. Furthermore, swiping left and right, it can move between individual and group data.

Once in the *Truck4Run* section, the individual can create, watch, enroll, unroll, start or delete a run. In the create run page, he can set all the characteristic of the run. Furthermore, he can access to an interactive map to set up the path of the run. In the section watch run there's instead a map with all the position of athletes.

3.2 User Experience

In this specification we include only mobile users' experience because for web application clients the experience will be the same except for the fact that all *Automated-Sos* and most of *Track4Run* function will be unavailable for web browsers.

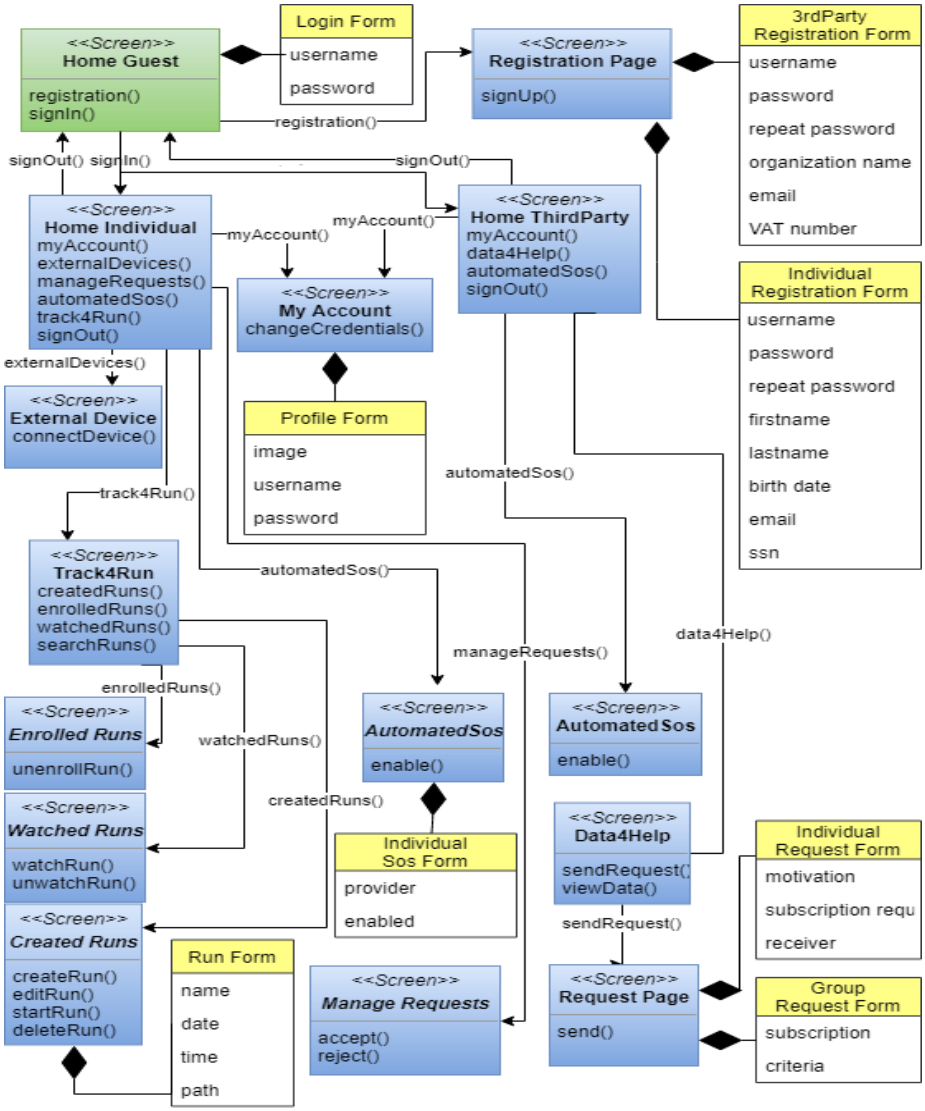


Figure 20: User eXperience

4 Requirements Traceability

REQ-1 : A customer not signed-on must be able to begin the Individual's registration process to TrackMe providing a username, a password and its organization data.

- Authenticator Controller
- Authenticator Service
- User Repository
- Individual Repository

REQ-2 : A customer not signed-on must be able to begin the Third Party's registration process to TrackMe providing a username, a password and its organization data.

- Authenticator Controller
- Authenticator Service
- User Repository
- Third Party Repository

REQ-3 : The system must provide a log-in interface for already registered users, not previously signed into the application.

- Authenticator Controller
- Authenticator Service
- User Repository

REQ-4 : TrackMe must provide to users the possibility to change their username.

- Authenticator Controller
- Authenticator Service
- User Repository

REQ-5 : TrackMe must provide to registered users the possibility to change their password.

- Authenticator Controller
- Authenticator Service
- User Repository

REQ-6 : The system must provide the possibility to the user of logging out.

- Authenticator Controller
- Authenticator Service

REQ-7 : The system must provide the possibility to change Individual's personal data.

- Authenticator Controller
- Authenticator Service
- Individual Repository

REQ-8 : The system must provide the possibility to change Third Party's organization data.

- Authenticator Controller
- Authenticator Service
- Third Party Repository

REQ-9 : Data4help must allow the Third-Parties to send a request to a particular individual, provided his SSN/FC.

- Request Controller
- Request Service
- Individual Repository
- Individual Request Repository

REQ-10 : Data4help must allow the Third-Parties to generated a request for a group of individuals.

- Request Controller
- Request Service
- Individual Repository
- Group Request Repository

REQ-11 : The system must be able to anonymize users' requested data.

- Request Controller
- Request Service
- Health Data Repository
- Group Request Repository

REQ-12 : Data4help must allow the Third-Parties to choose if subscribe to new data associated with a particular Individual.

- Request Controller
- Request Service
- Individual Request Repository
- Group Request Repository

REQ-13 : The system must be able to periodically query the database in order to get new data as soon as they are produced.

- Request Controller
- Request Service
- Individual Request Repository
- Group Request Repository
- Health Data Repository

REQ-14 : Data4help must allow the Third Party to see a list of sent requests and related Individual's data.

- Request Controller

- Request Service
- Individual Request Repository
- Health Data Repository

REQ-15 : Data4help must allow the Individual to see a list of the received requests from Third Parties.

- Request Controller
- Request Service
- Individual Request Repository

REQ-16 : Data4help must allow the Individual to accept or reject a request from the list.

- Request Controller
- Request Service
- Individual Request Repository

REQ-17 : Data4help must allow the Individual to connect an external device through BT or NFC.

- External Device API

REQ-18 : The system must allow the Third-Party to activate Automated-SOS service.

- Sos Controller
- Sos Service
- Third Party Repository
- Dispatcher Repository

REQ-19 : The system must be able to assign an ambulance dispatcher to the SOS service.

- Sos Controller
- Sos Service
- Dispatcher Repository

REQ-20 : The system must allow the Individual to activate Automated-SOS service.

- Sos Controller
- Sos Service
- Third Party Repository

REQ-21 : The system should allow the Individual to choose between third parties who has enabled the service.

- Sos Controller
- Sos Service
- Individual Repository
- Third Party Repository

REQ-22 : The system must be able to get the Individual position.

- Sos Controller
- Sos Service
- Individual Repository
- Position Repository

REQ-23 : The system must be able to send the position to the nearest ambulance.

- Sos Controller
- Sos Service
- Sos Subscription Repository
- Individual Repository
- Dispatcher Repository

REQ-24 : Track4Run must allow the organizer to create a run with a date, time, duration and a path.

- Run Controller
- Run Service
- Run Repository

REQ-25 : The system must provide an interface that allows the user to define a path on an interactive map.

- Google Maps API

REQ-26 : Track4Run must allow the organizer to start a run previously created.

- Run Controller
- Run Service
- Run Repository

REQ-27 : Track4Run must allow the organizer to delete a run previously created.

- Run Controller
- Run Service
- Run Repository

REQ-28 : Track4Run must allow the Athlete to enroll to an already existing run.

- Run Controller
- Run Service
- Run Repository

REQ-29 : Track4Run must allow the Athlete to unroll a run.

- Run Controller
- Run Service

- Run Repository

REQ-30 : Track4Run must allow the Spectator to see the position of the Athletes on a map during a run.

- Run Controller
- Run Service
- Run Repository
- Position Repository
- Individual Repository

5 Implementation, Integration & Test Plan

With respect to specific application services, the implementation plan to be followed should be to develop first *Data4Help* functions and then *Automated-Sos* and *Track4Run* facilities, which are built on top of *Data4Help* service.

Authentication subsystem instead can be implemented independently on the order on which other components are; an implementation strategy for this module is to deploy it in parallel with Data4Help subsystem.

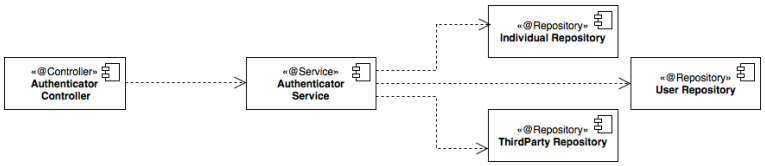
5.1 Test Strategy

Despite this document follows a *Top-Down* style starting from an high level point of view to a detailed components and operations description, we advise an *I&T* plan following the *Bottom-Up* approach, in which modules at lowest layer will be tested first and upper layers will follow then.

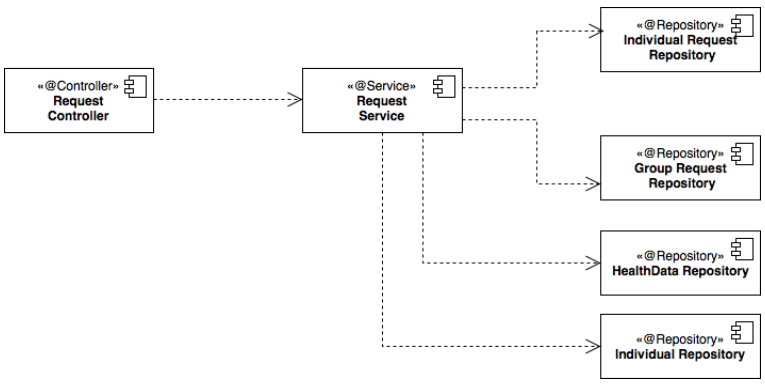
The unit testing is focused on testing all the software components' features, then the test will proceed to the integration phase: integration of two components will start only after components executing that operations pass the unit test to ensure that the components themselves work fine and that any failure is related to the integration itself.

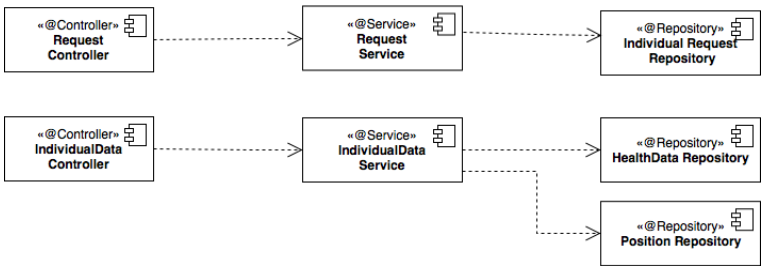
Components to be tested are listed below; integration test will flow from right to left:

a) Integration between Authenticator Controller and Service:

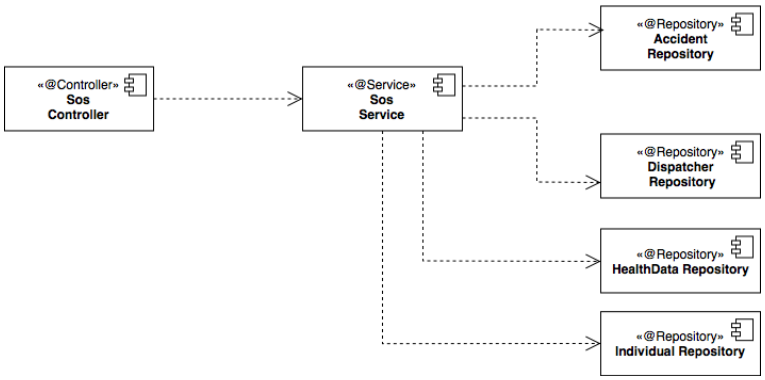


b) Integration between components of *Data4Help* Subsystem:

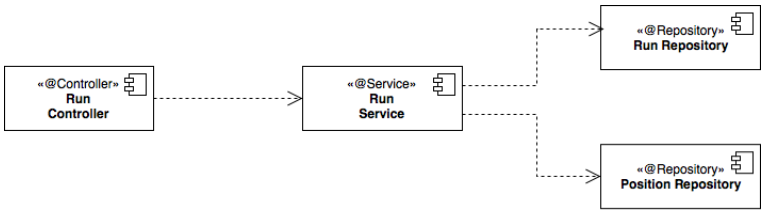




c) Integration between components of *AutomatedSos* Subsystem:



d) Integration between components of *Track4Run* Subsystem:



5.2 Entry Criteria

5.2.1 Unit Test

The *Unit Test* phase will start only after:

- the proposed system architecture has been designed and reviewed;
- testable code for units to be tested is available;
- the test environment has been properly configured and is ready to run.

5.2.2 Integration Test

The *Integration Test* phase will start only when:

- Unit Test phase is completed;
- each component has gone through and passed the unit test phase before the integration phase;
- all priority bugs found in the unit test phase are fixed and marked as solved;
- the test environment has been properly configured and is ready to run.

5.3 Exit Criteria

5.3.1 Unit Test

The *Unit Test* phase will end only after:

- all unit tests are successfully executed;
- priority bugs are fixed.

5.3.2 Integration Test

The *Integration Test* phase will end only after:

- integration system has passed all functional requirements;
- integration system has passed all performance requirements;
- all high severity or top priority bug has been fixed;
- high risk identified area has been taken up and tested.

If *I&T* integration exit criteria are satisfied, the test phase can move on to the System Testing in which Load, Performance and Stress tests may be useful to highlight some Quality of Service leaking of the system.

6 Effort Spent

- **Davide Rutigliano: 60h**
- **Davide Matta: 60h**
- **Claudio Ferrante: 60h**