POLITECNICO
MILANO 1863

TrackMe project - Argiro' Anna Sofia,
Battaglia Gabriele, Bernardo Casasole

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Argiro' Anna Sofia, Battaglia Gabriele, Bernardo Casasole
Version:	0.5
Date:	December 8, 2018
Download page:	https://github.com/BernardoCasasole/ArgiroBattagliaCasasole.git

Contents

1. Introduction

1.1 Purpose

1.2 Scope

Data4Help means to provide services to authenticated users only. Those services are addressed to both:

- Individual Users
- Third parties Users

To dispatch specific functionalities to the user they are reserved, Data4Help System avails itself of:

- a Mobile App, reserved to individual users
- a Web Page, reserved to third party users.

The mobile app, using the GPS location provided by the smartphone, allows the individual user to:

- check his own health parameters (measured by a smartwatch)
- enable and disable additional services (AutomatedSOS and Track4Run)
- give or deny authorization to every third party to access health data about himself.

Data4Helps System handles both data of the past and real time ones. The web page allows the Third-party user to:

- make requests for statistical data of the past or real time
- make a request for individual data of the past or real time (the request is forwarded to the individual user)
- organize and watch run competitions.

This factorization allows the system to be accurate in providing every user with all and only resources he has the right to access: authentication and authorization processes rely on the access control.

The necessity to use a mobile app could prevent third parties from choosing Data4Help over other services of data collection: Data4Help Web Page can be easily accessed from a browser hosted on a computer or a mobile.

1.3 Definitions

- *User*: a person, third-party or user, that has registered;
- *Individual User*: every registered person from whom the system collects data;
- *Third-Party User*:every entity registered with the purpose to request data for external use;
- *non-human Third-Party User*: a software Third-Party User that access to the offered D4H services thorugh the exposed APIs
- *Live Data*: the data on a IU produced in real time.
- *Stored Data*: the data on a IU collected so far.
- *Data Request*: a request for data made from a TPU.
- *Stored Data Request*: a data request for stored data.
- *Subscription Request*: a request for subscribing to newly generated data.

1.4 Acronyms

- API: Application Programming Interface
- TPU: Third-party User
- D4H: Data4Help
- ASOS: AutomatedSOS
- T4R: Track4Run
- UX: User experience
- REST: REpresentational State Transfer
- EENA: Eurepean Emergency Number Association
- PSAPs: Public Safety Answering Points
- NG112: Next Generation 112

1.5 Abbreviations

- Gn: n-goal
- Dn: n-Domain assumption
- Rn: n-Requirement

1.6 Revision history

- **v0.1 - 27/11/18** Document created
- **v0.2 - 30/11/18** Component view
- **v0.3 - 2/12/18** Model diagrams, User inteface and High level overview
- **v0.4 - 8/12/18** Architectural patterns, interfaces, deployement, high level architecture review
- **v0.5 - 10/12/18** Implementation, integration and testing

1.7 Document Structure

Introduction

Architectural Design

User Interface Design

Requirements Traceability

Implementation, Integration and Test plan

Effort Spent

References

2. Architectural Design

2.1 Overview

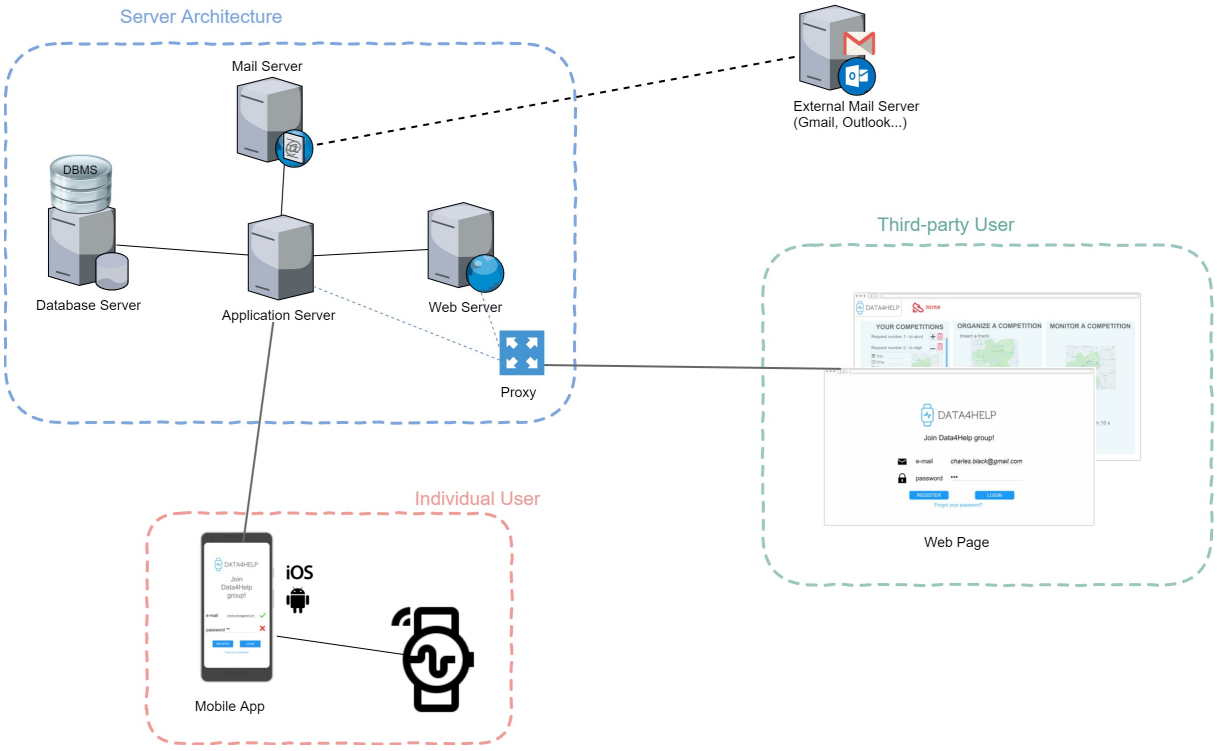
The architecture style used is a client/server structure with multiple tiers while an event-backbone will handle the dispatch of live data through the system. The presentation layer will be hosted on both client (IUs and TPUs clients) while the application server will host the logic layer and the database server the data layer. The IU client is going to be a thick client, hosting a brach of the application logic to handle better and faster the system functionalities.
Cloud server solutions, over local servers, are the ideal fit for system having variable demands and workload as Data4Help and meet the following needs:

- to avoid hardware faults: improving availability
- to enhance security
- to only pay for the exact amount of server space used
- minimization of data losses and recovery time.

Google Cloud Platform might be chosen over other cloud-server-hosting providers because of the possibility to use both SQL and NoSQL databases.

2.1.1 High level components and basic interactions

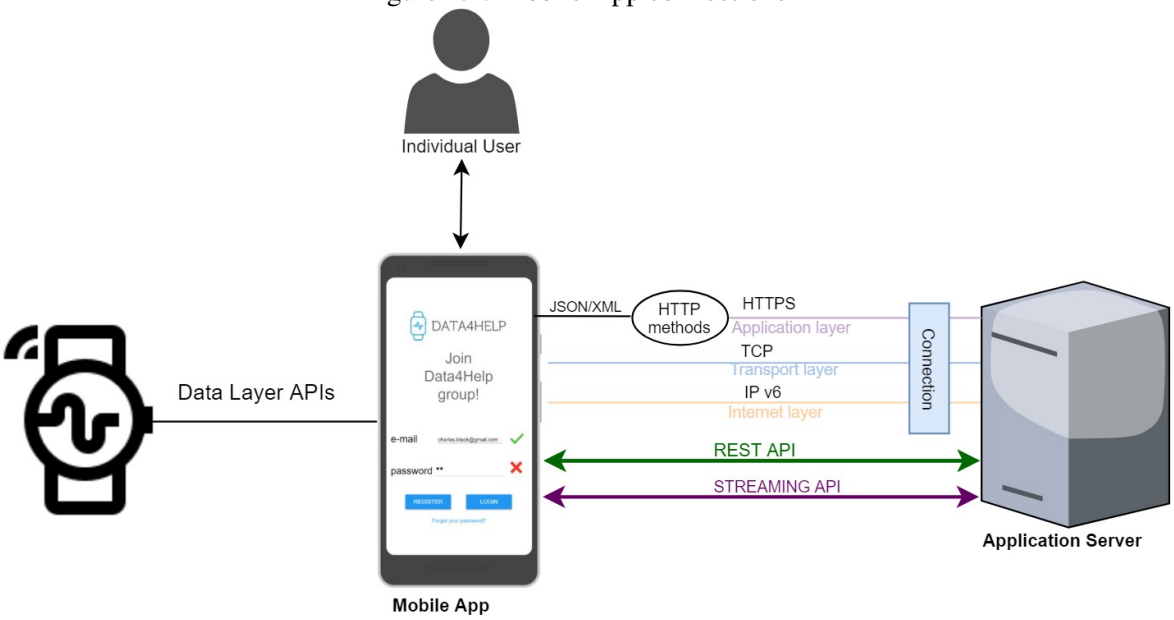
Figure 2.1: High level components' elements



The overall structure, at high level, is made of three main components and their interaction. The red component refers to the tools the individual user needs to interface with Data4Help System, it communicates with the Application Server that is part of the blue component charged with the Server Architecture. This is composed by a Database Server that includes the DBMS, a Mail Server which means to exchange SMTP messages with other Mail Servers (external to the system), an Application Server communicating with any other element in the Server Architecture, a Web Server and a Proxy (meant to dispatch requests to Application and Web Servers). The proxy links the Server Architecture with the green component charged with the interaction with the third party user that takes place through Data4Help Web Page.

2.1.2 Interaction between Server Architecture and Individual User

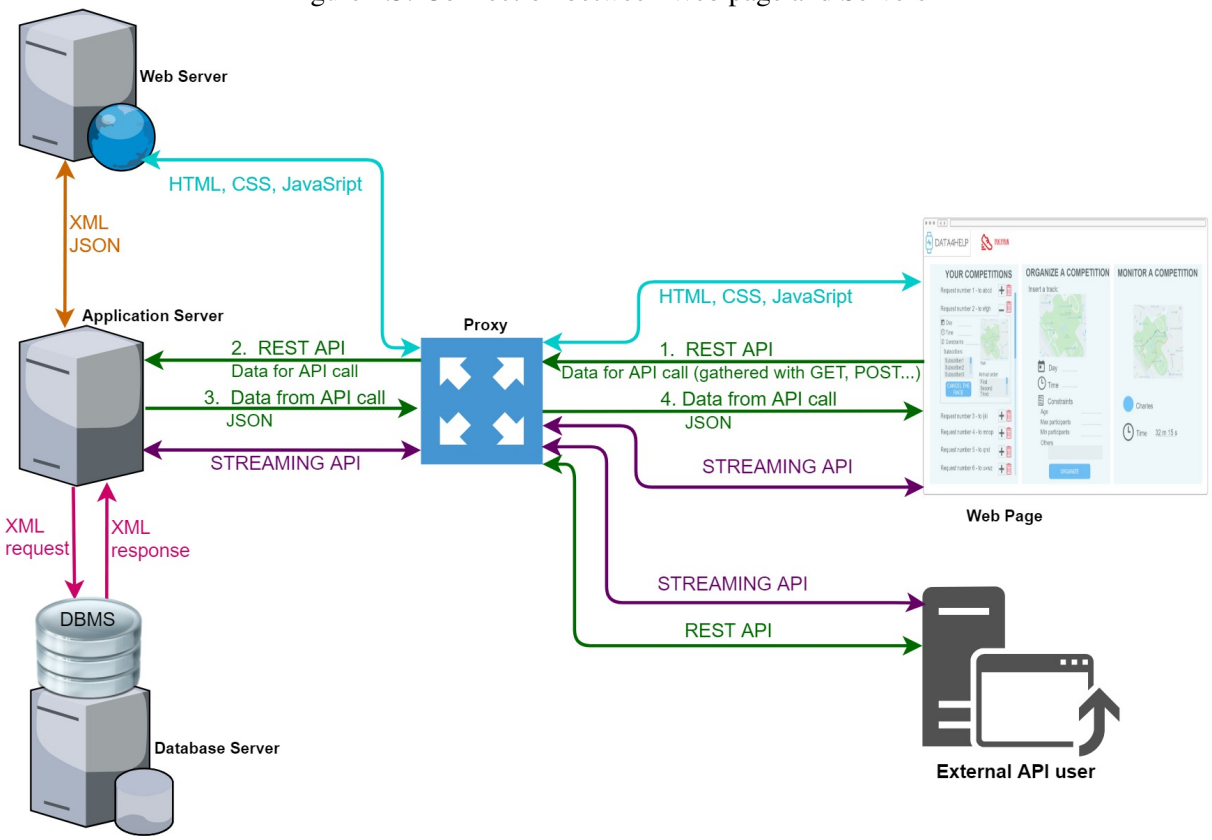
Figure 2.2: Mobile App connections



The Mobile App receives data from the smartwatch, exchanges informations with the Individual User and communicates with the Application Server: at different levels are specified protocols that are supposed to be used.

2.1.3 Interaction between Server Architecture and Third-Party User

Figure 2.3: Connection between Web page and Servers



The browser hosting the Web Page needs to communicate both with the Web Server and the Application Server. The Web Server can easily handle and exchange HTML, CSS and JavaScript files with the client; the Application Server manages methods like GET, POST receiving a REST API call and forwarding data in JSON format. Data to forward are provided by the Database Server which includes the DBMS: a request in XML is sent by the Application Server, the DBMS processes the request and extracts data from the database that are sent back to the Application Server in a XML file. To establish a communication channel between the Application Server and the Web Server is not a necessity, however it provides an alternative to REST API: developers are up to decide to implement them both or to keep the REST API alone.

2.2 Component view

The system is divided in four subsystem:

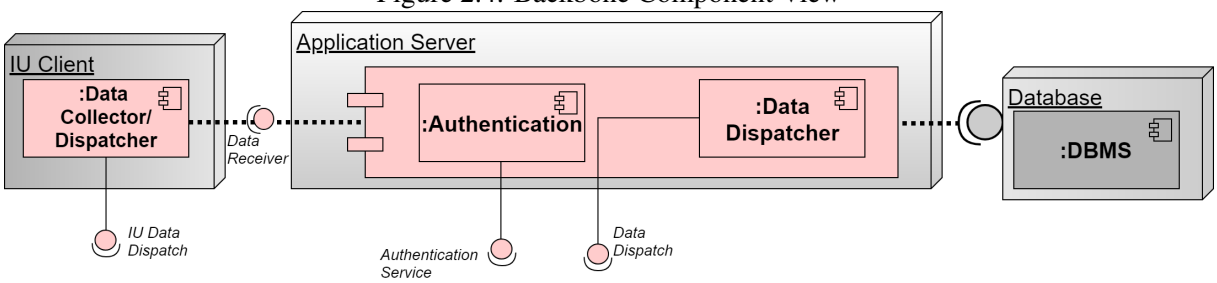
- **Backbone**
- **Data4Help**
- **AutomatedSOS**
- **Track4Run**

The Backbone is the core of the system: all other subsystems interact with it and don't interact with each other. The backbone provides interfaces for authentication and to receive live data published form the

Backbone with a event-based paradigm.
The last three are divided, on the Application server, in a router that provide an interface gathering all the subsystem functionalities, and a module, containing all other components of the subsystem, which uses the exposed method of the DBMS to be able to work indipendently.
On the IU and TPU clients the view component represent the presentation layer of the system, which Users can access directly.
The relation between the components and the model il further defined in figure 2.10.

2.2.1 Backbone

Figure 2.4: Backbone Component View



This is the backbone of the system: collects the data on the device, keep it syncronized though the system, stores it onto the database and provide the functionalities to receive live data; Furthermore provide functionality concernig authentication.

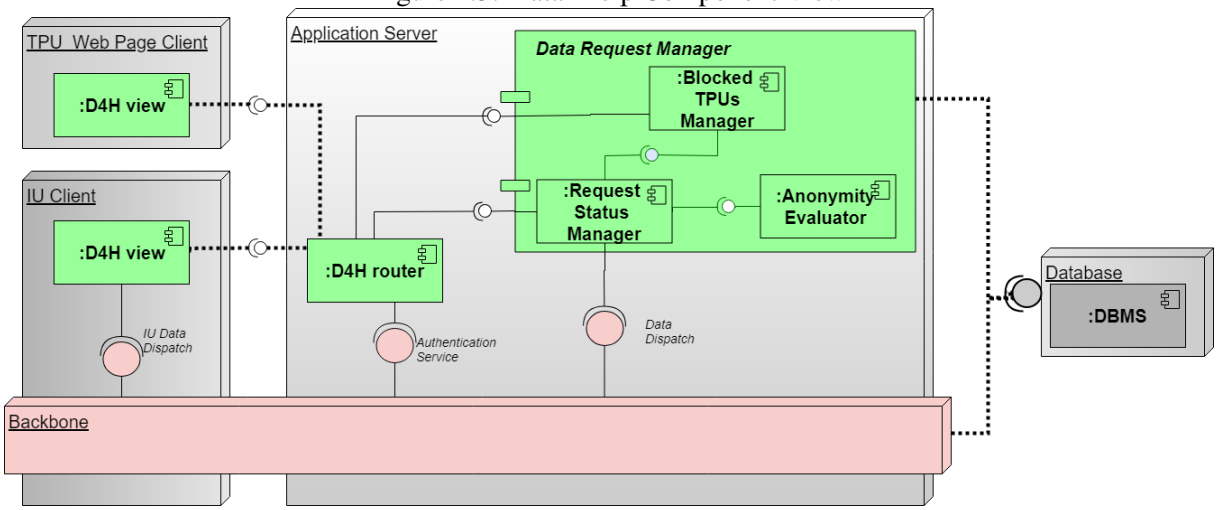
Data collector/dispatcher Allow subscribtion from other components on the IU client and publishes/dispatches the collected live data of the Individual User logged in from the device.

Autenthication Offers services related to User authentication and the functionalities to handle their info.

Data Dispatcher Allow subscribtion from other components on the application server and publishes/dispatches the collected live data of all Users and it stores it onto the database.

2.2.2 Data4Help

Figure 2.5: Data4Help Component View

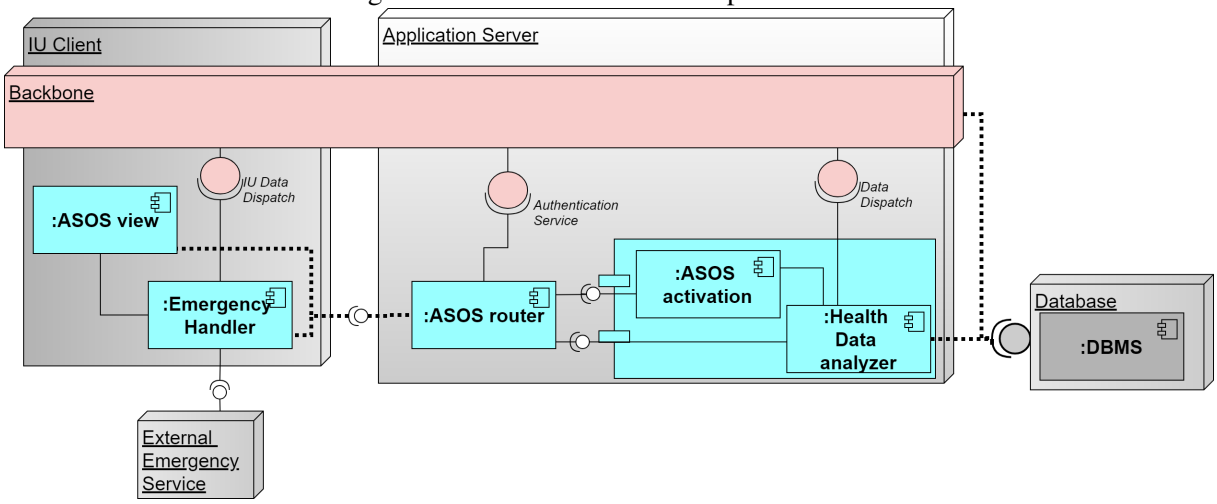


D4H router Validate the requests received from the client and dispatch them to the corresponding module or component.

Data Request Manager Provides functionality to create, approve, deny requests, block users and provide the relative data; Anonymity Evaluator is responsible to check anonymity constraints.

2.2.3 AutomatedSOS

Figure 2.6: AutomatedSOS Component View



ASOS router Validate the requests received from the client and dispatch them to the corresponding module or component.

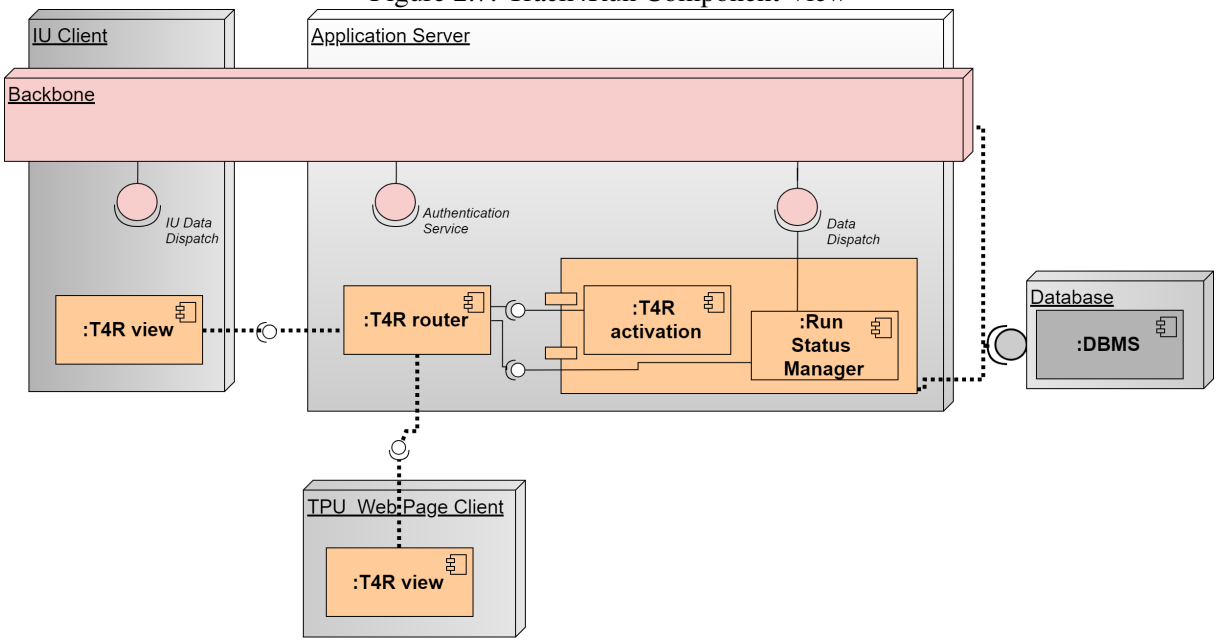
ASOS Activation Offers the functionality for the activation and deactivation of the ASOS service.

Health Data analyzer Offers functionality to extrapolate the critical health parameters for every Individual User;

Emergency Handler Responsible to handle critical health conditions based on the data published by the *Data collector/dispatcher*; the method for contacting the emergency service is further explained in section 2.7.1.

2.2.4 Track4Run

Figure 2.7: Track4Run Component View



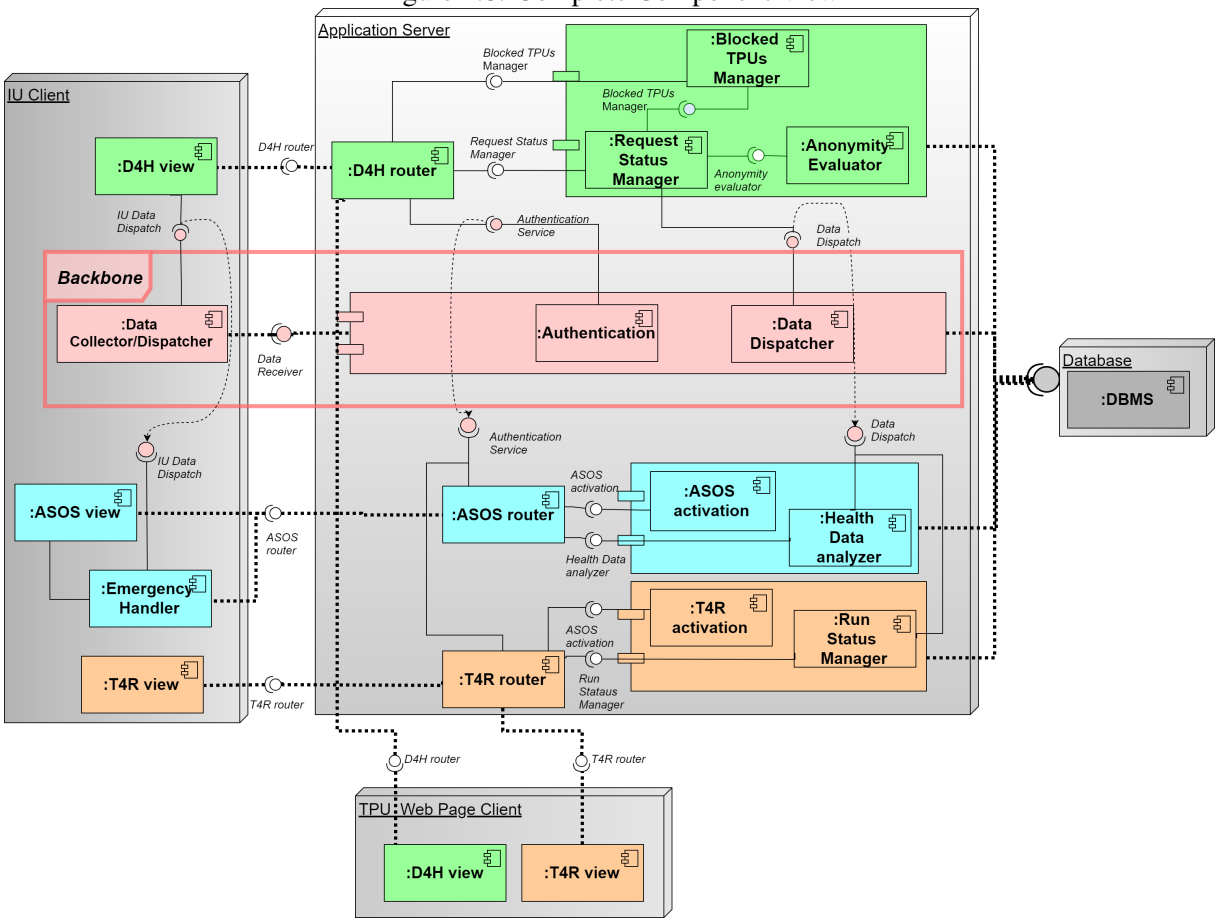
T4R router Validate the requests received from the client and dispatch them to the corresponding module or component.

T4R Activation Offers the functionality for the activation and deactivation of the T4R service.

Run Manager Provides functionality to create, cancel and enrol in runs.

2.2.5 Full system

Figure 2.8: Complete Component View

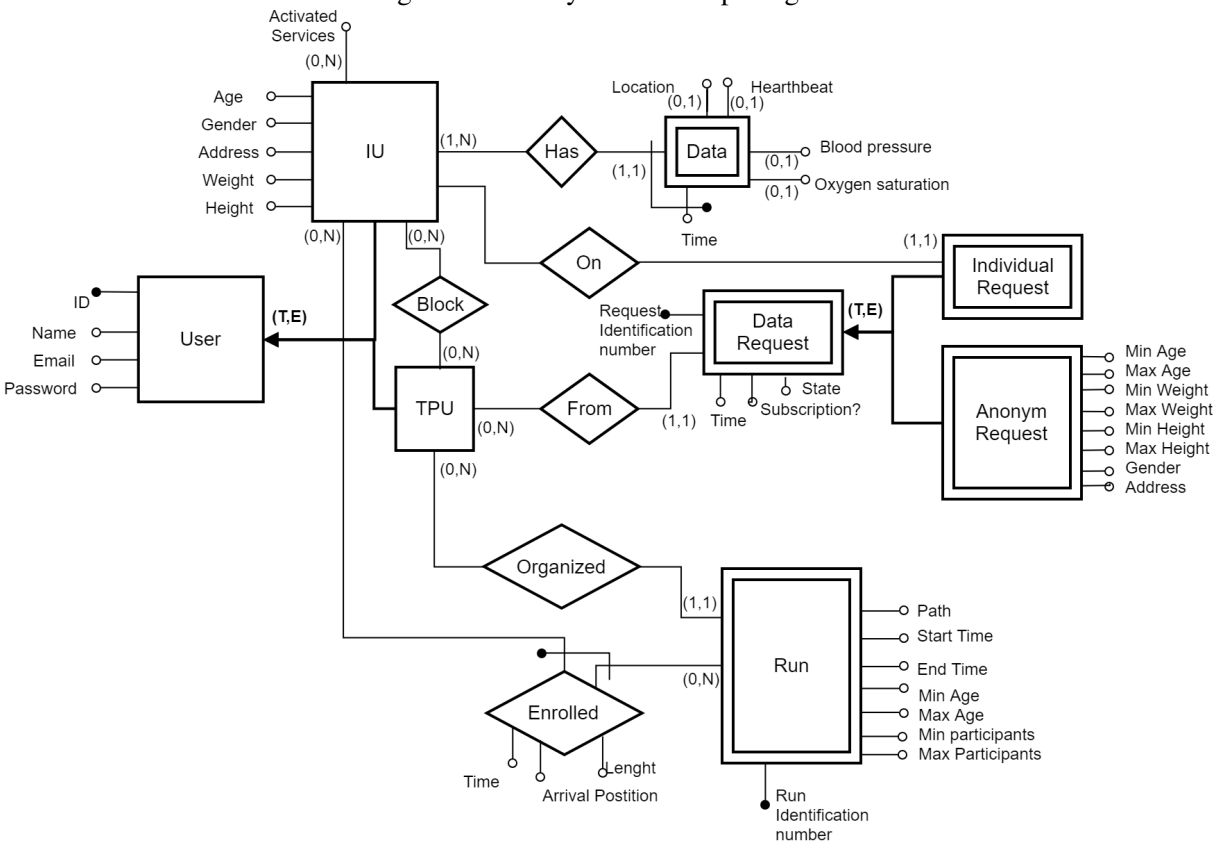


Data Managing From a more high level point of view, the backbone provides services to retrive the Individual Users live data.
This makes the red components and modules of the architecture the backbone, collecting and dispatching data, while the other subsystems can handle their unique authorization condition: D4H authorizing data dispatching based on approved requests, ASOS on the activation of the service and T4R on the enrollement in competitions.
This way all subsystem can work independently from each other.

2.2.6 Entity Relationship Diagram

The following section provides a conceptual representation of the model.

Figure 2.9: Entity Relationship Diagram



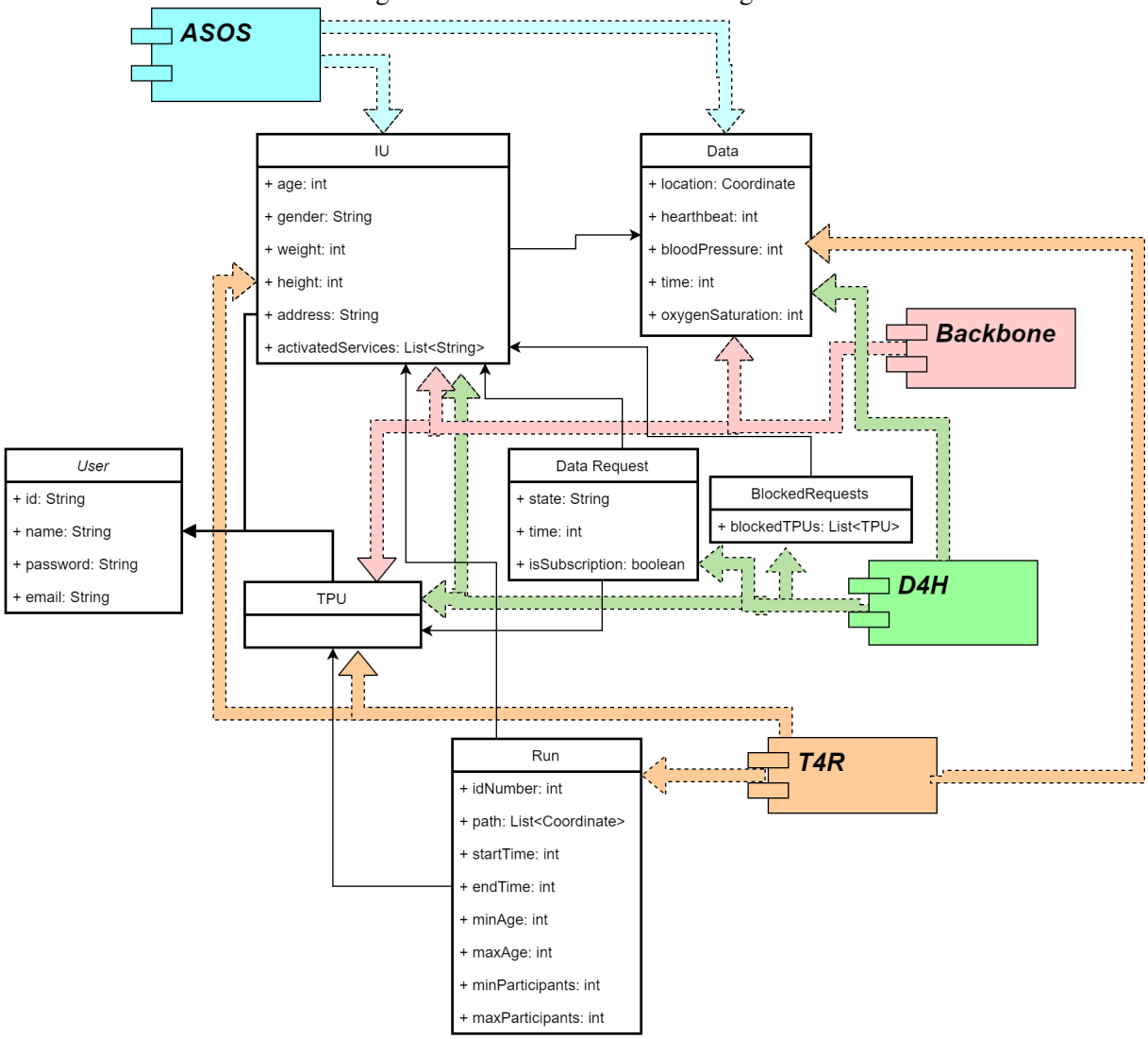
Tables

- ***User***(ID, Name, Email, Password)
- ***TPU***(ID, Name, Email, Password)
- ***IU***(ID, Name, Email, Password, Age, Gender, Address, Weight, Height)
- ***Data***(IU, Time, Location, Heartbeat, Blood pressure, Oxygen saturation)
- ***Individual Request***(Request Identification Number, IU, TPU, Time, State, Subscription?)
- ***Anonym Request***(Request Identification Number, TPU, Time, State, Subscription?, Min Age, Max Age, Min Weight, Max Weight, Min Height, Max Height, Gender, Address)
- ***Run***(Run Identification number, TPU, IU, Path, Start Time, End Time, Min Age, Max Age, Min participants, Max Participants)
- ***Run Result***(Run Identification number, IU, Lenght, Time, Arrival Position)

2.2.7 Model Interaction Diagram

The following diagram show a different representation of the model to better highlight its interaction with the application server. For each subsystem module that was connected to the DBMS in 2.2.5 is shown its relationship with the module.

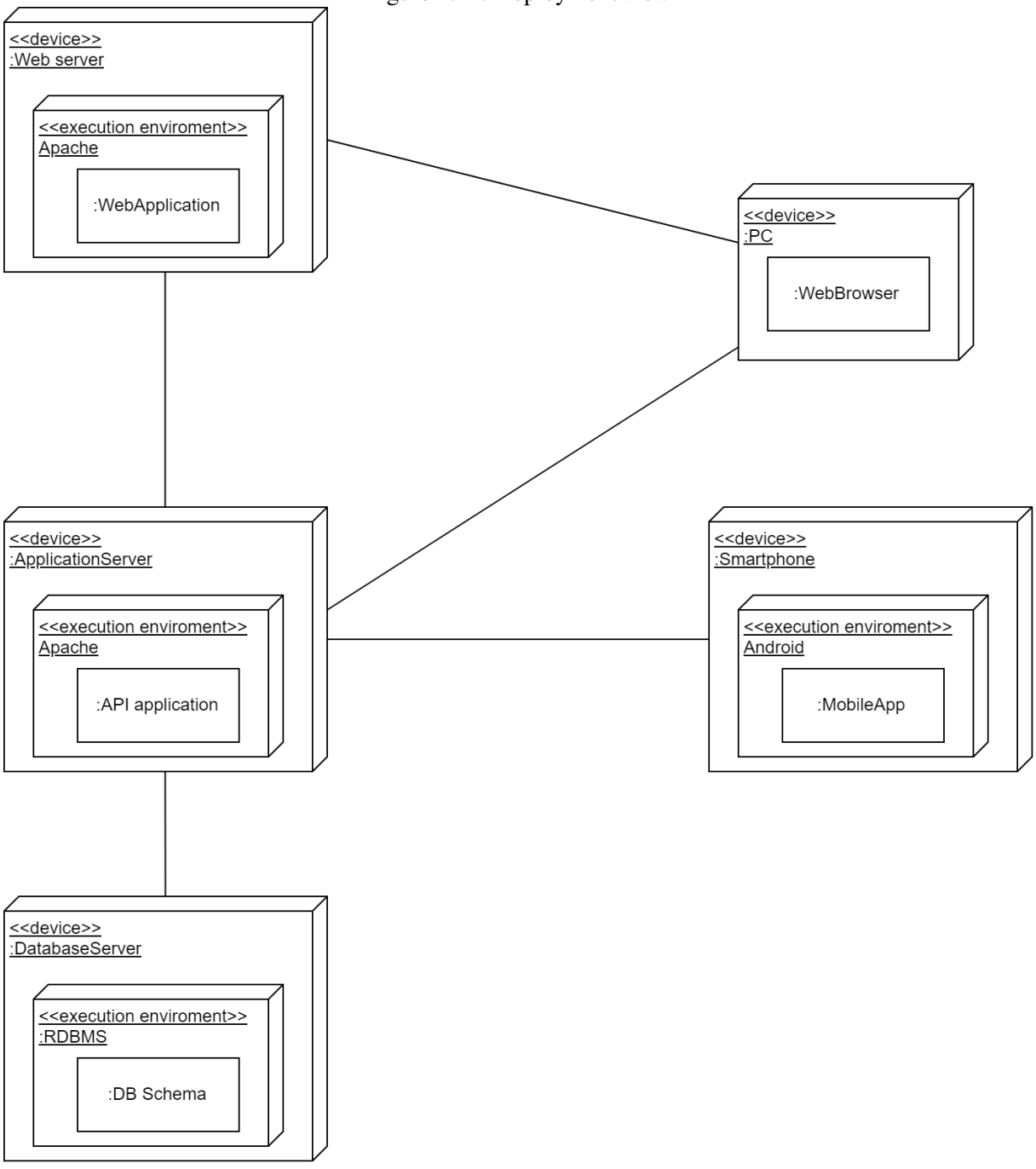
Figure 2.10: Model Interaction Diagram



2.3 Deployment view

As stated in the previous sections the system is composed by the two clients, one hosted on a web browser and the other on moblie application. They both rely on the application server while the former also interacts with the web server which host the web application. The application server provide the logic of the system and interacts with the database server which hosts the data layer of the system.

Figure 2.11: Deployment view



2.4 Runtime view

Figure 2.12: IU Registration

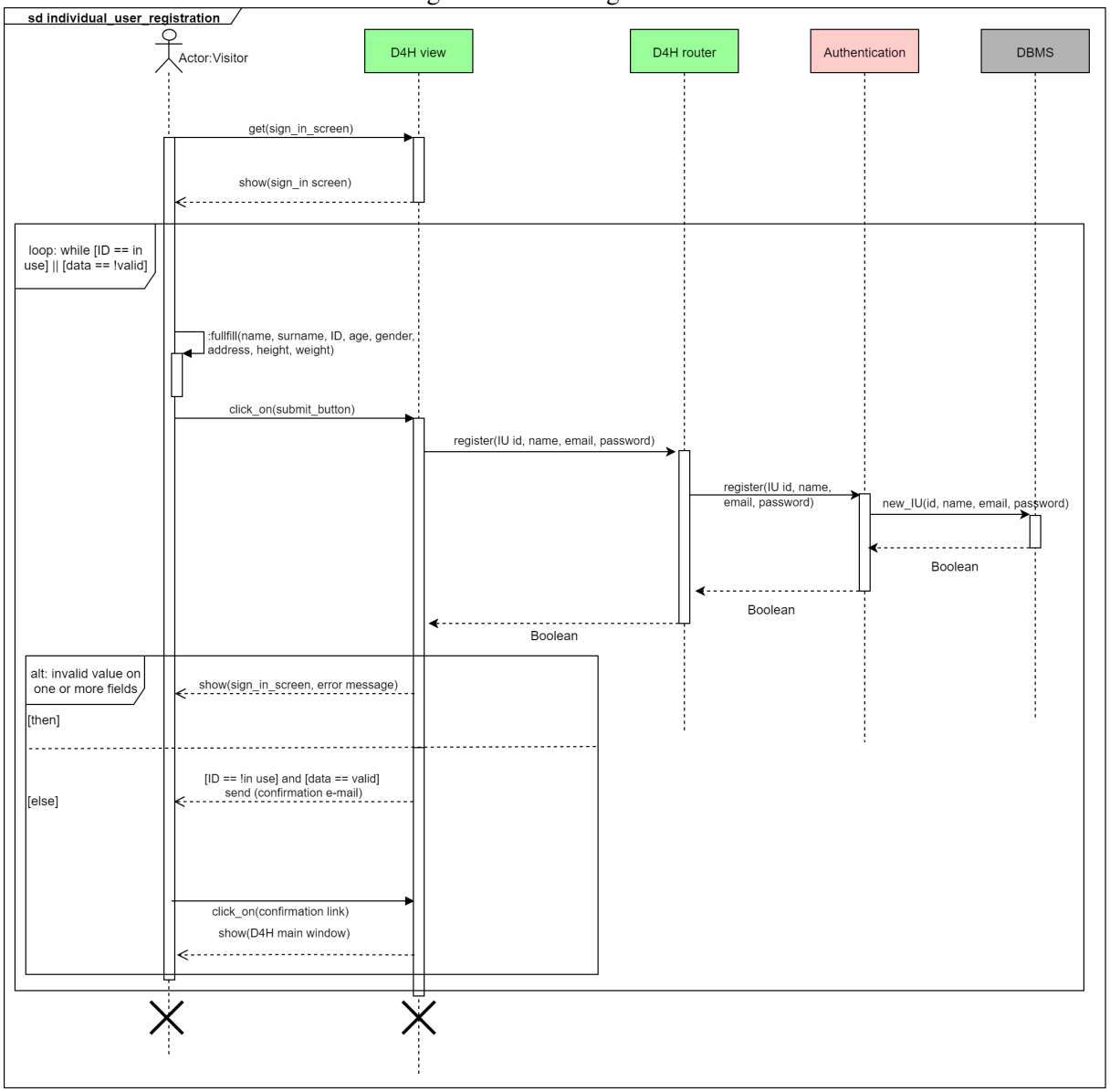


Figure 2.13: Data Requests

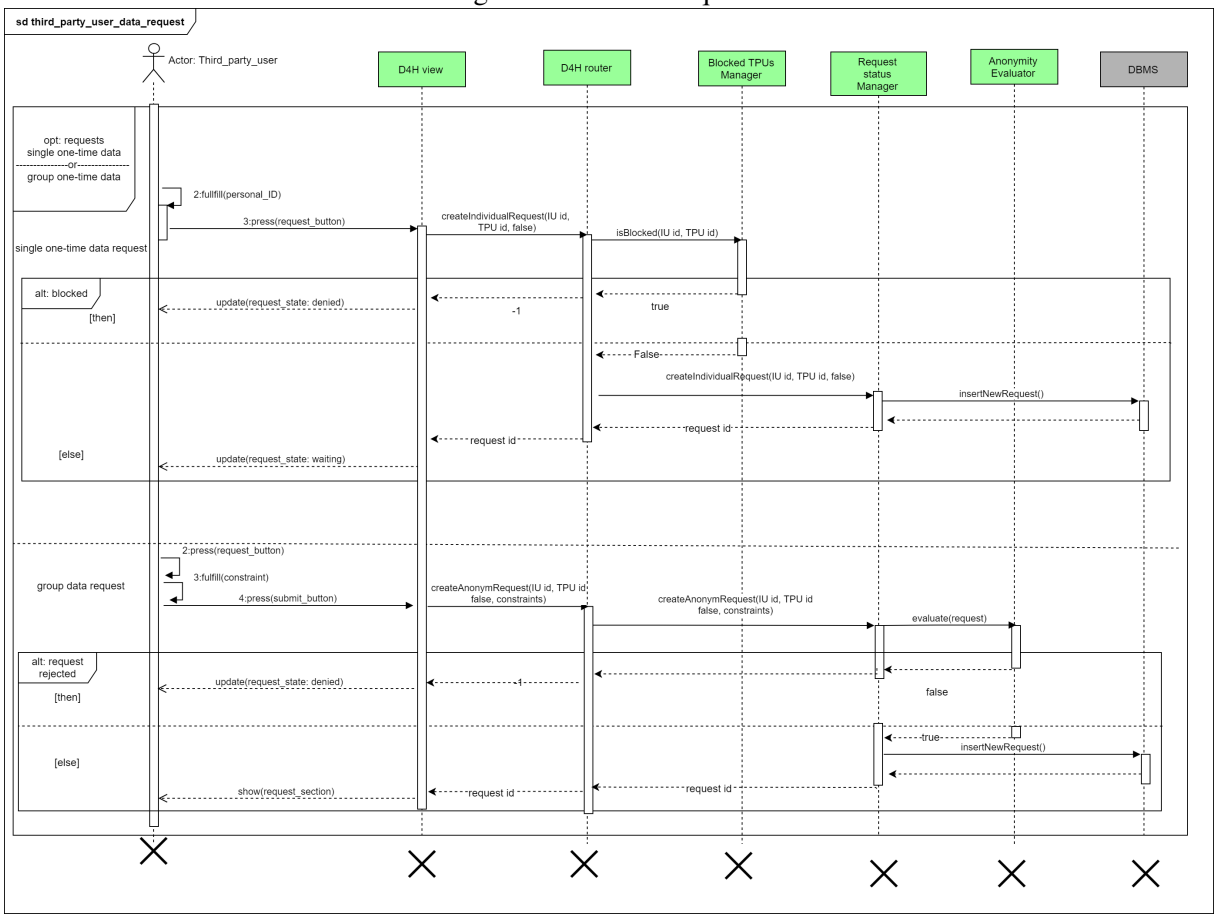


Figure 2.14: Emergency call

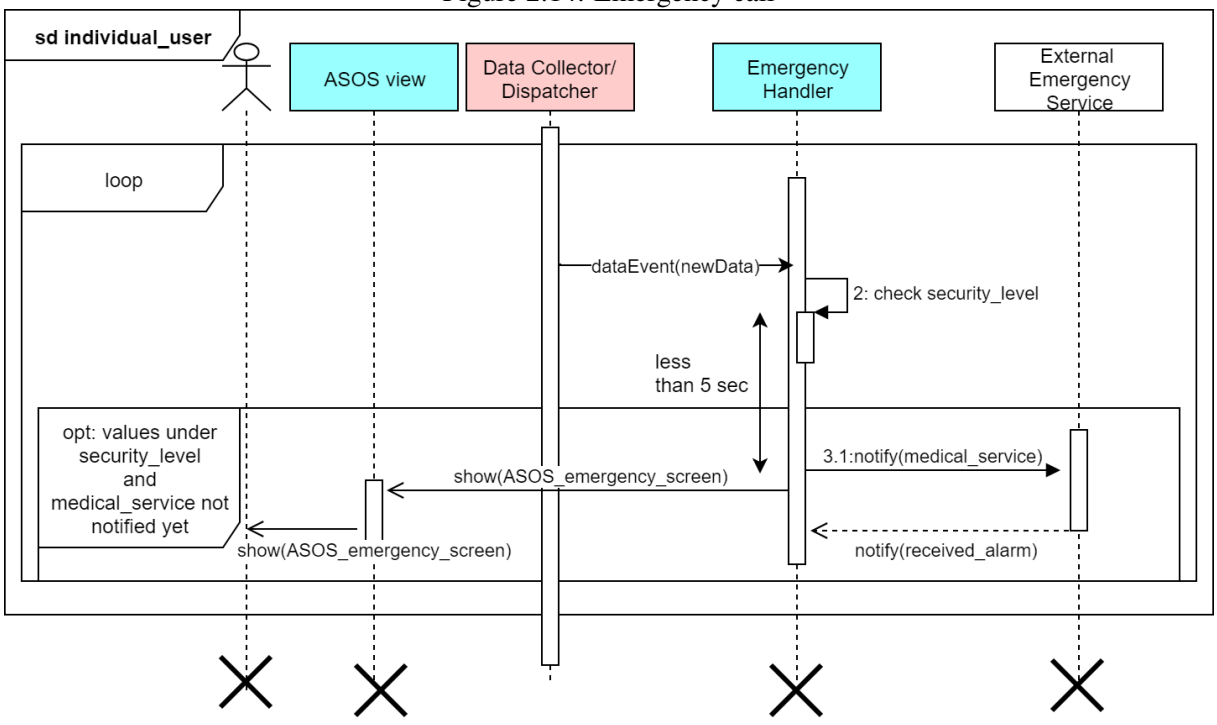
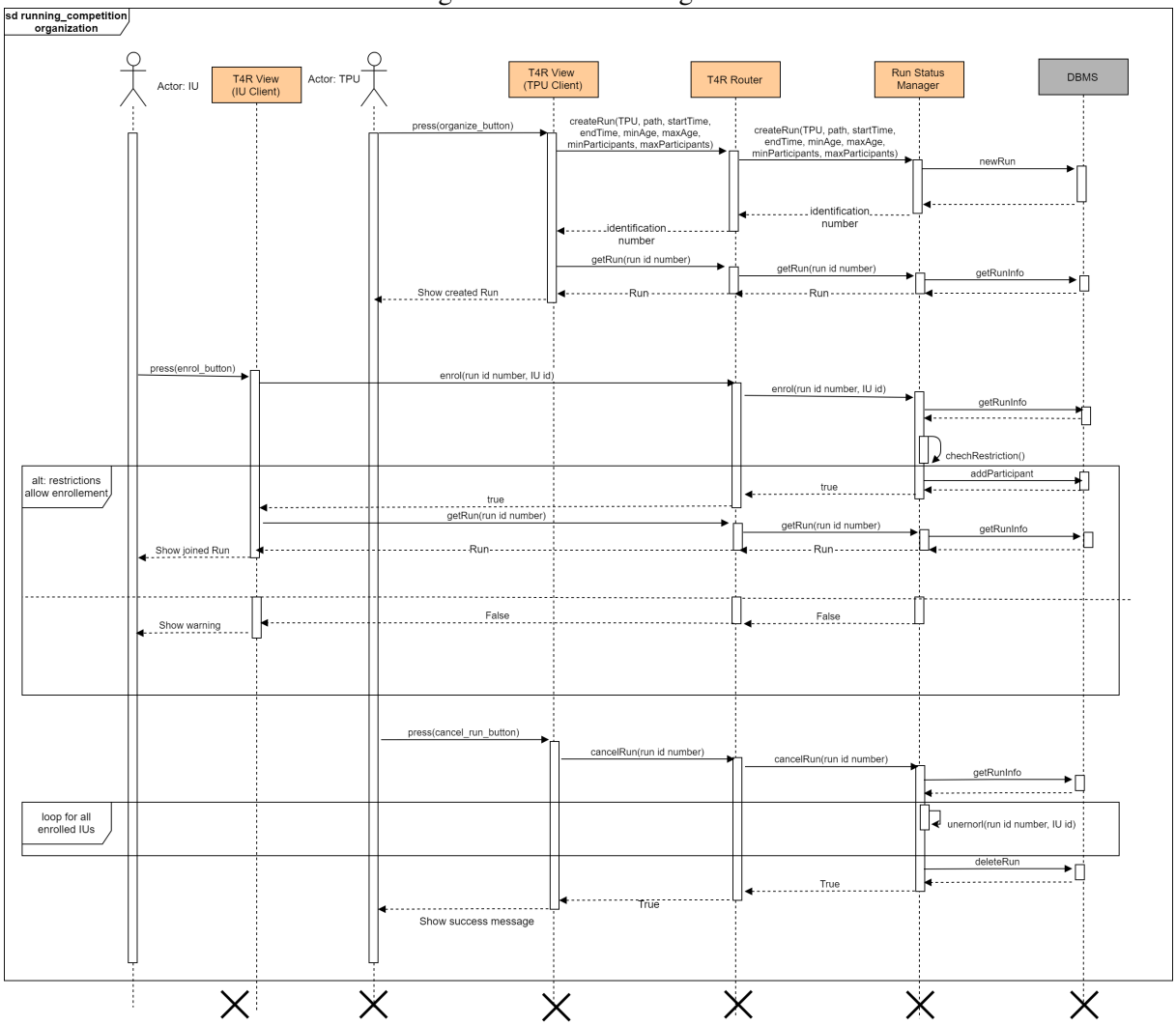


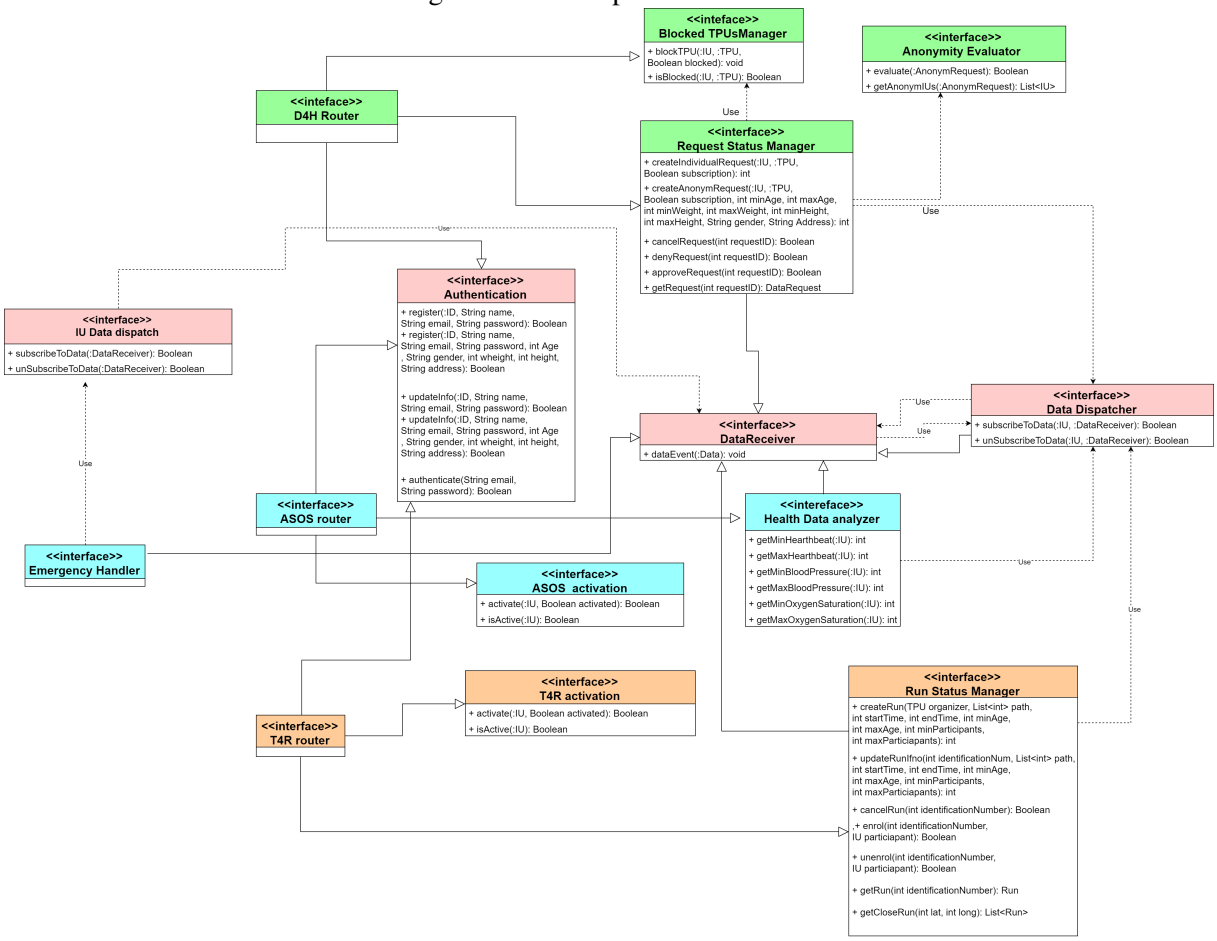
Figure 2.15: Run Management



2.5 Component interfaces

The next diagram shows the most important methods of the components interfaces which, for clarity, are named in figure 2.2.5 tracing the components names.
The routers gather oll the method required to provide the client with the corresponding subsystem services and expose the relative APIs for the clients (for the D4H router also non-human TPUs) to use.
An generic interface *Data Receiver* is extended by all the interfaces that use the *Data Dispatcher* service, to receive the updates.

Figure 2.16: Component Interfaces



2.6 Selected architectural styles and patterns

2.6.1 Client/server multi-tier

The architecture style chosen is a client/server structure with multiple tiers. The presentation layer is diveded between the two clients (IUs and TPUs clients) which are thick clients since they host a branch of the application logic to handle better and faster the system functionalities; namely, to provide the fastest possible emergency response time, the client directly handles critical conditions contacting the emergency service and the backbone handles the dispatching of the IU live data to other components on the client.
The application server hosts the logic layer, exposing API the clients to access the subsystem functionalities and, for the D4H router, to non-human TPUs which might access directly through the APIs; The application server is divided in four subsystems, each handling a piece of logic: a backbone, handling the core logic, storing data and user authorization, and providing interfaces to other subsystem to use its functionality, while the other subsystem independently handle the functionalities of the three services offered: D4H, ASOS and T4R.
The database server host the data layer and all the subsystems on the application server independtly interact with it.
This will make for a modular software, enabling a fairly independent implementation and testing of each subsystem; Morover it, alongside the tiered structure, will improve scalability and maintainability.

2.6.2 Event based paradigm

The backbone, namely the Data Dispatcher components, is an event-based subsystem that handles the dispatch of live data through the system. Live data collected by the Data Collector/Dispatcher serves as the event, broadcasted to all registered components. While introducing potential scalability problems, it simplify the addition of the other subsystem.

2.6.3 REST and Real-Time API

By creating a RESTful system that uses REST API and using a stateless protocol which relieve the server from storing client context between requests, we increase performance, reliability, and scalability. That said, using this software architectural style only could be too restrictive since it doesn't cope well with Real-Time trasmission of data: for example within the Backbone, meaning between the component that collects the data on the client and the component on the server that receives, stores and forward it.
To publish data as fast as possible a Real-Time oriented API is used as support to handle the automated, synchronous and bi-directional communication between the server and both the clients and eventual non-human TPUs, while the REST API will be used for all other needs.

2.6.4 Model View Controller

The pattern dictates the separation the user interface components (the view components in figure 2.2.5), the data, which resides on the DBMS, and the response to user imput and logic, located mostrly on the application server and partly on the client.

2.7 Other design decisions

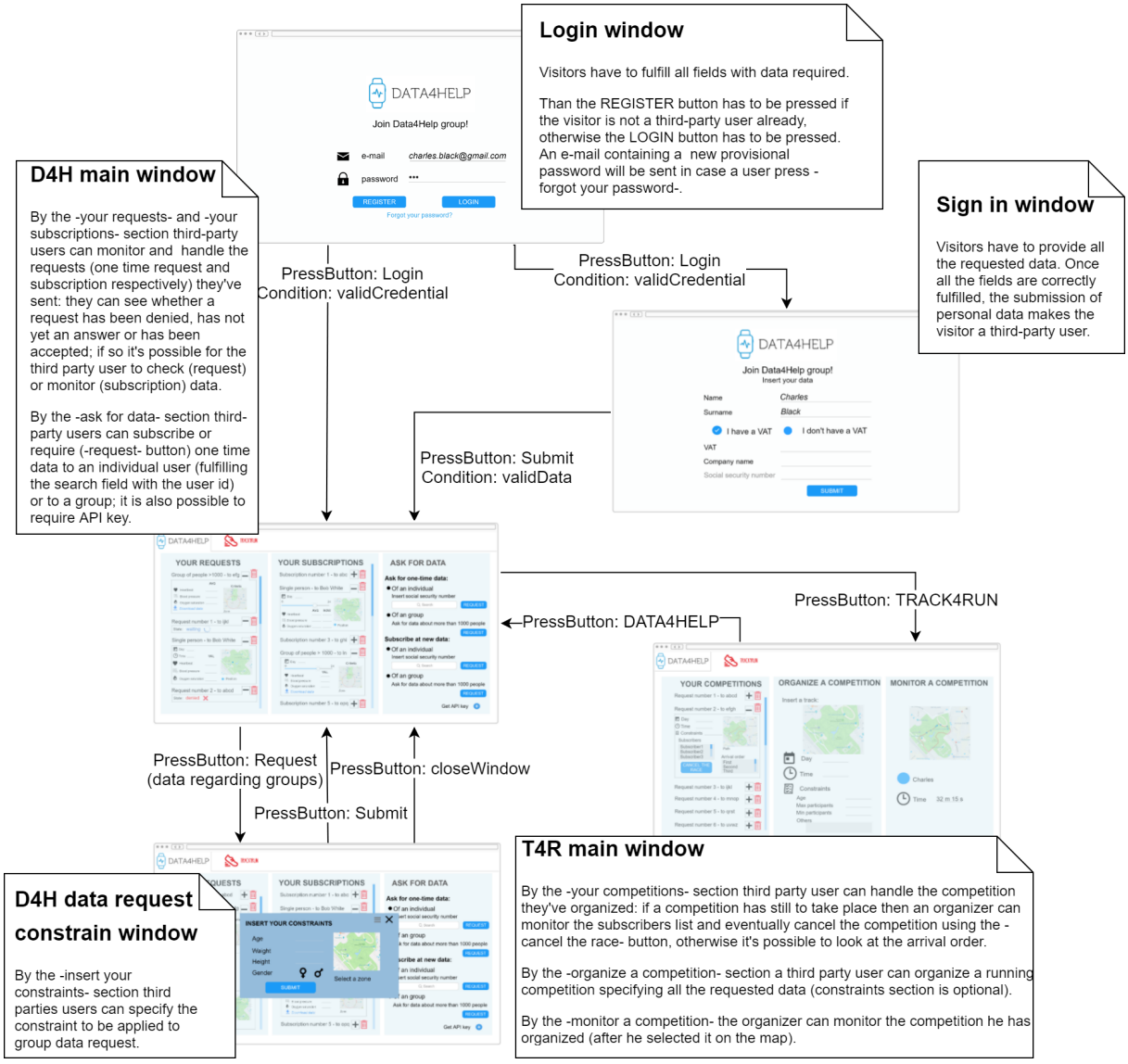
2.7.1 Emergency service

As method of contacting an ambulance there are some options: in Europe a single emergency number exist (112) and the European Emergency Number Association (EENA) handles a network of Public Safety Answering Points (PSAPs) and provides documentation, standards and regulations for automated access to 112 services. Access is possible, in its simplest form, with an automated call which relies on the IU communicating their info, or in better and more automated forms, using the Next Generation 112 (NG112) methods that are being implemented by PSAPs all over Europe.

3. User Interface Design

The user interfaces mock-ups are represented in sections 3.1.1, 3.1.2 of RASD. The following UX schemes represents a complete description of the user experience. The screen -T4R unsubscribe screen- has been added and a better description of each mock-up has been provided.

Figure 3.1: Third-party user - UX graphical representation



Third-party user The scheme above represents the main desktop screens and the way -condition and action needed- how the third-party user can move trough them.

Figure 3.2: Individual user - UX graphical representation (left part)

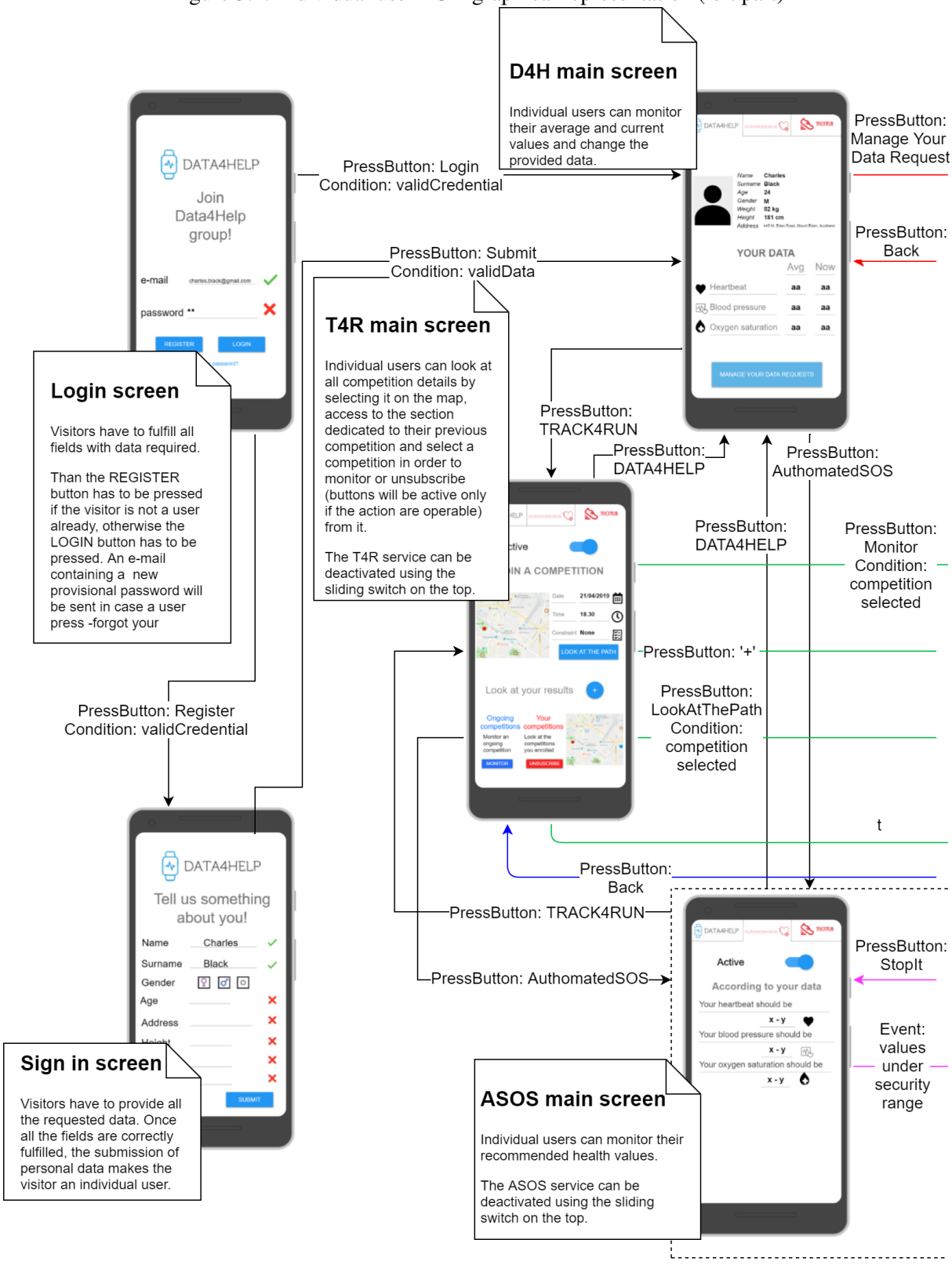
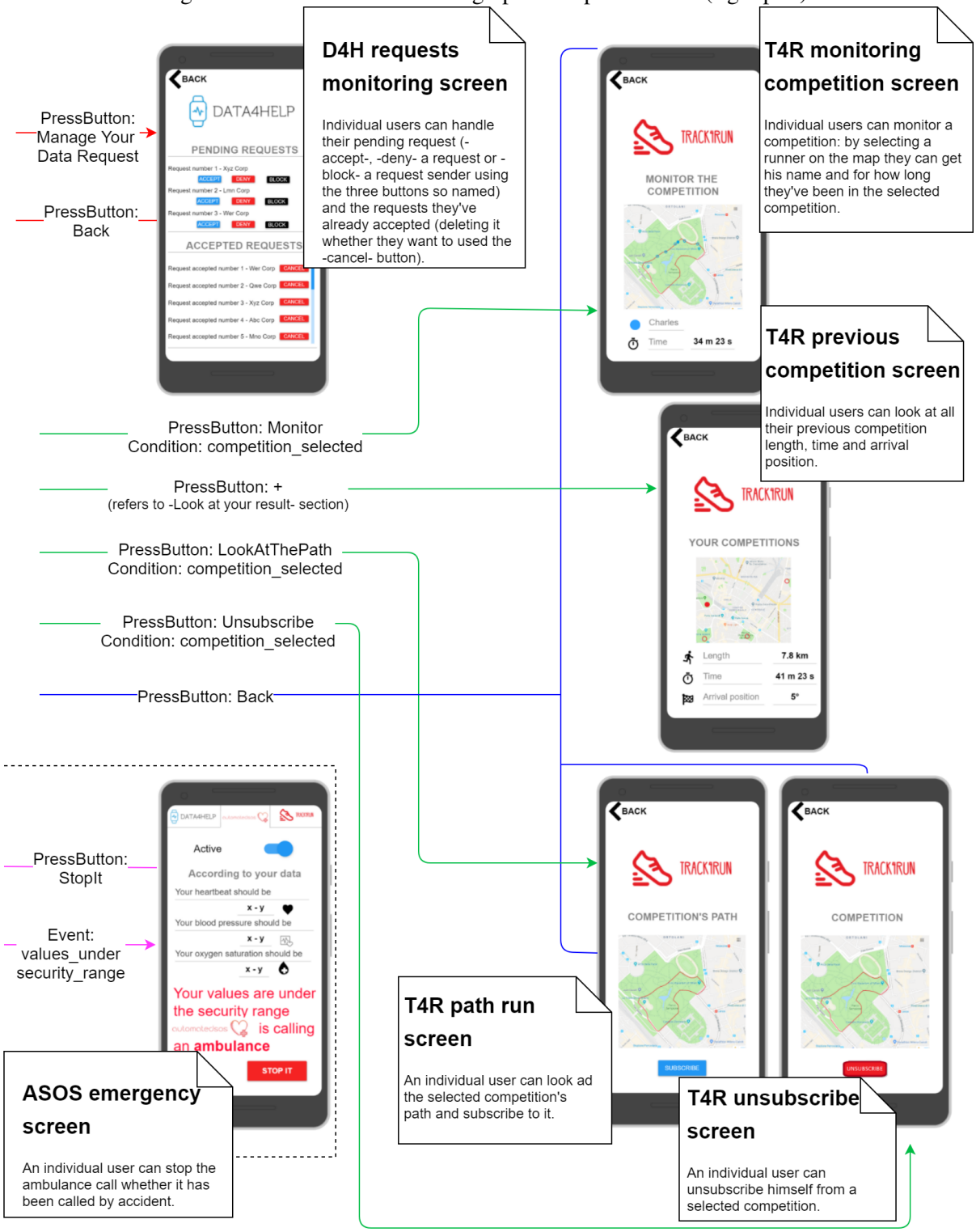


Figure 3.3: Individual user - UX graphical representation (right part)



Individual user The two schemes above represents the main mobile screens and the way -condition and action needed- how an individual user can move trough them. The scheme has been divided in two parts in order to provide a better readability.

4. Requirements Traceability

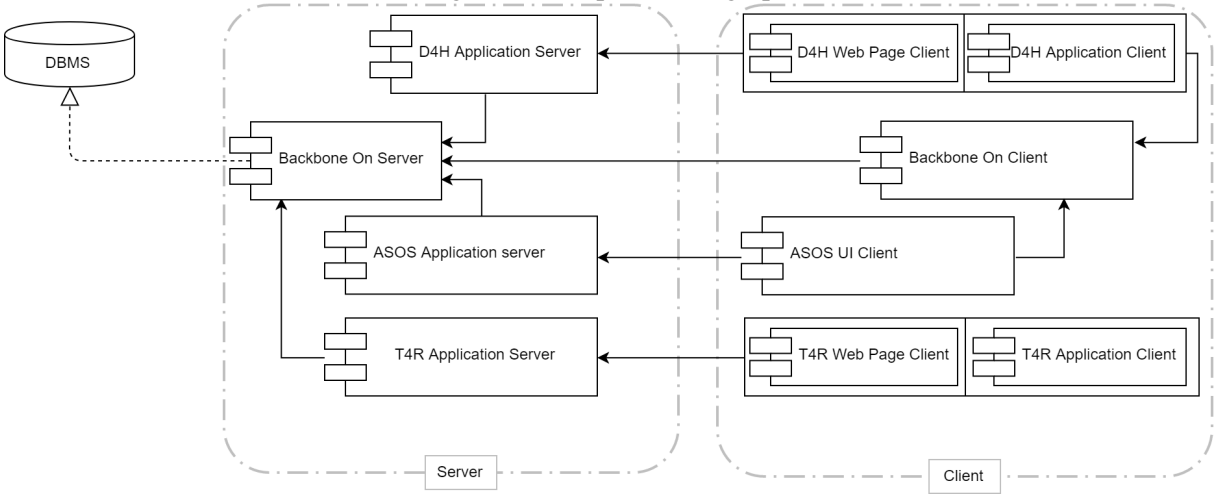
5. Implementation, Integration and Test plan

5.1 Implementation strategy

The implementation of D4H, T4R and ASOS will follow a critical-module-first approach. According to this strategy implementation should start whith the most complex, critical and connnected components. Both a bottom-up approach or a top-down one, considered as sole strategy, would be only partially effective. As a metter of fact the great advantage given to the design structure of the project, that is to say the possibility of parallelize the implementation of the three applications, whouldn't be fully exploited.
The critical-module-first approach choice is due to the structure of the system as it has been desiged. The backbone of the three application is, indeed, the same for D4H, T4R and ASOS; so it is for sure the most critical and connected module. The implementation of the three application, once all the shared components will be tested and the integration tests will be performed, could also be done in parallel.
Of course this implemetation strategy requires a sort of bottom-up approach. In fact, once the backbone of the application will be completed, the actual implementation will consists in the piecing together of systems to generate more complex systems.
About the programming language to be used, HTML5 + Java/Javascript are feasible options, but other options as Python or Android Studio are perfectly equally valid possibilities. Considering the design of the project theres no particular recommendation about programming language choice needed.
In order to provide a more exhaustive description of the implementation strategy an analysis of the dependences is provided.

5.1.1 Dependences

Figure 5.1: Dependences graph



The represented modules refer to the 2.2 section. Arrows stand for -depends on-.
The following order is not mandatory for developers. It is to be considered an advice given in order to optimize the implementation time.
1- Backbone On Server is for sure the very first module to be implemented, since many modules depend on it and it is crucial in the comunication with the DBMS.
This module is composed by the following componets:

- Authentication
- Data Dispatcher

2- Backbone On Client is the second module to be implemented in order to complete as soon as possible the applications' backbone.
This module is composed by the following componets:

- Data Collectior/Dispatcher

Once applications' backbone is completed the implementation of D4H, T4R and ASOS can run independently.
Whether a parallelized implementation is not possible a coherent with the assigment order is suggested. Here it is a possible implementation order:
3- D4H Application Server module is composed by the following componets:

- D4H Router
- Request Status Manager
- Blocked TPUs Manager
- Anonymity Evaluator

4- ASOS Application Server module is composed by the following componets:

- ASOS Router
- ASOS activation
- Health Data Analizer

5- T4R Application Server module is composed by the following componets:

- T4R Router
- T4R activation
- Run Status Manager

6- D4H Web Page Client/D4H Application Client module is composed by the following componets:

- D4H view

7- ASOS UI Client module is composed by the following componets:

- ASOS view
- Emergency Handler

8- T4R Web Page Client/T4R Application Client module is composed by the following componets:

- T4R view

5.2 Testing on components strategy

Testing on single components should be done as soon as the components itself are produced. The testing order of the single components must be coherent with the implementation order indicated at 5.1.
In testing process maight be usefoul the support of a multiplicity of software verifcation tools which allow to run large number of tests during the development and verifcation of the system. Of course the choice of verifcation tools depends on the programming language chosen, anyway some usefoul tools could be:

- **JUnit** - can be used in order to verify assertions on return values after method invocations.

- **Mockito** - a mocking framework for unit tests which can be used in order to perform scaffolding activity.

- **Apache JMeter** - is a Java application designed to load test functional behavior and measure performance. It can be useful to simulate a heavy load on servers, but can be used also in order to analyze overall performance. under different load types

The following table is reported in order to indicate some values that it's worth to submit. Those values are indicated to stress software and obtain a good testing coverage.
The -!(type)- stands for -submit types different from the indicated one-. The ✓stands for -submit this kind of value-.

Data	sub. model	Null	Neg.	!(int)	!(String)	!(Coord)	!(List)	!(0_1 value)	!(legal value)
age	IU	✓	✓	✓					
gender	IU	✓			✓				✓
weight	IU	✓	✓	✓					
height	IU	✓	✓	✓					
address	IU	✓			✓				✓
activatedServices	IU	✓					✓		✓
location	Data	✓				✓			✓
heartbeat	Data	✓	✓	✓					
bloodPressure	Data	✓	✓	✓					
time	Data	✓	✓	✓					✓
oxigenSaturation	Data	✓	✓	✓					
state	DataRequest	✓			✓				✓
time	DataRequest	✓	✓	✓					✓
isSubscripted	DataRequest							✓	
id	User	✓			✓				✓
name	User	✓			✓				✓
password	User	✓			✓				✓
email	User	✓			✓				✓
blockedTPUs	Blocked Requests	✓					✓		✓
idNumber	Run	✓	✓	✓					
path	Run	✓				✓	✓		✓
startTime	Run	✓	✓	✓					✓
endTime	Run	✓	✓	✓					✓
minAge	Run	✓	✓	✓					
maxAge	Run	✓	✓	✓					
minParticipants	Run	✓	✓	✓					
maxParticipants	Run	✓	✓	✓					

5.3 Integration strategy

5.3.1 Completion of components before starting testing

The integration and integration testing should start as soon as possible.Of course before starting with integration is necessary to be sure that the external services and APIs that will be used in the applications should be available and ready. In order to speed up the integration process do that, only a certain percentage of completion is actually needed. In particular the completion of components before the starting the integration should be at least:

- Backbone On Server - 90-100%
- Backbone On Client - 80-90%
- D4H Application Server - 75-85%
- ASOS Application Server - 70-80%
- T4R Application Server - 70-80%

- D4H Web Page Client/D4H Application Client - 65-75%
- ASOS UI Client - 60-70%
- T4R Web Page Client/T4R Application Client - 60-70%

According to the critical-module-first approach and the testing approach expressed at point 5.2 also component integration should happen firstly in applications backbone and just then in the three applications core. Since a parallel implementation is possible, as soon as components completion meets the required percentages integration should be performed. Whether parallel implementation is not possible, the following graphical rappresentation shows a possible integration order (obviously coherent with the inplementation plan).

Figure 5.2: 0 and 1 integration phase

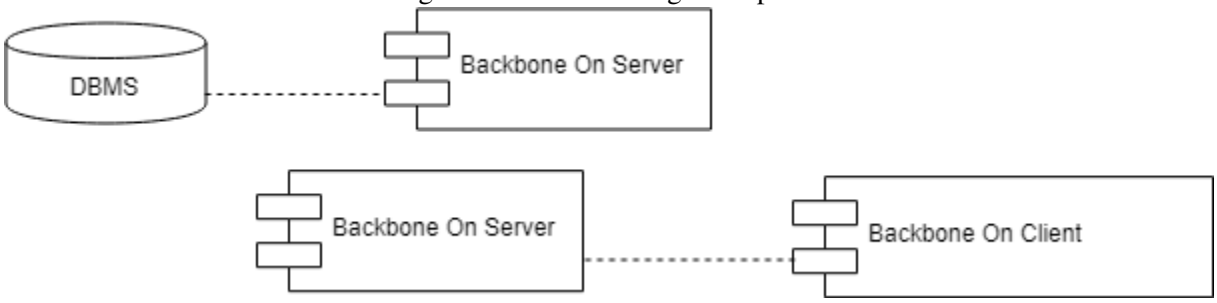
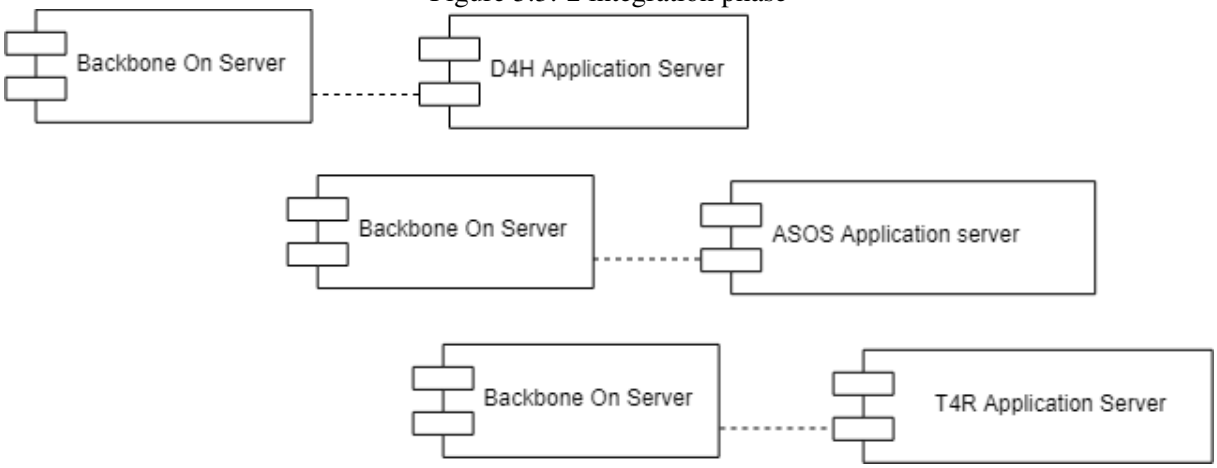


Figure 5.3: 2 integration phase



The integration order of the following modules has to depends on the implementation order. The following order is just a suggestion which doesn't consider the parallel implementation possibility.

Figure 5.4: 3 integration phase

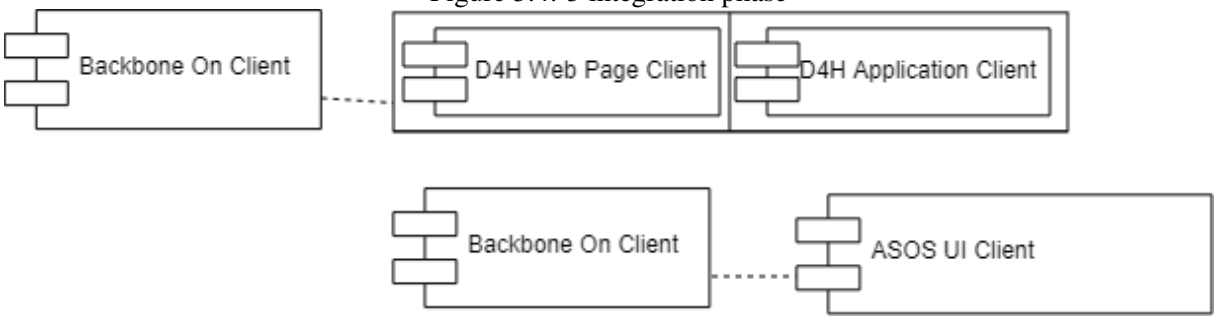
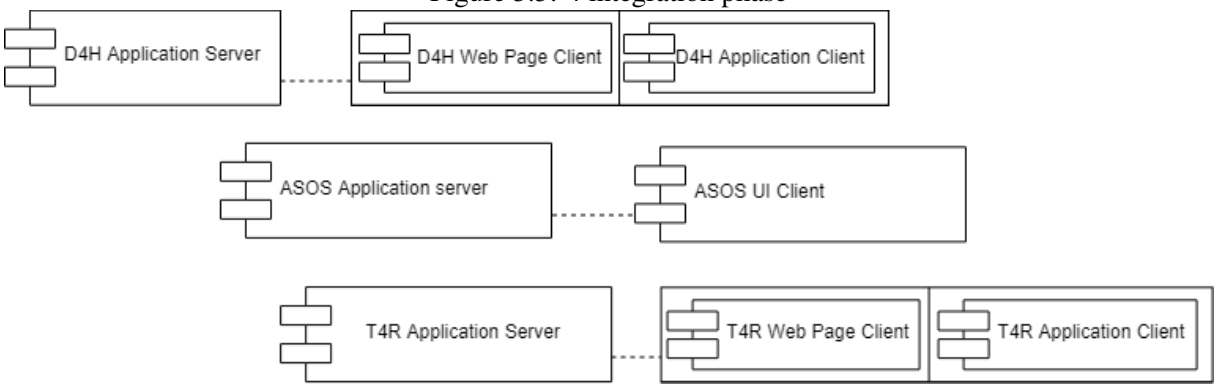


Figure 5.5: 4 integration phase



5.4 Integration test plan

The following section should represents a testing guide line: some of the most important tests that is necessary to perform in the various integration phases are listed.
Tests IDs are structured as follows: the firs two characters stand for the integration test number (tNumber), while the last two characters stand for the integration phase (iNumber).

5.4.1 0 integration phase

All the following tests wiil be generic: it is not really important the kind of data request, all that metters is submitting both valid and not valid input in order to test the server-DBMS communication. To provide a better coverage the greater possible number of different request kinds should be performed. Stub objects and oracles should be used instead of the not yet implemented components.

Test ID: t1i0	
Components involved	Data Dispatcher, Authentication, DBMS
Input specification	Data x that belonging to y stored in DBMS request
Output specification	Data x
Requirements-goals involved	R7, R8 - G4, G6
Description	To test the communication between DBMS and the Backbone On Server in dowload some data requests have to be performed. The communication should be effective and fast enough: Database Rensponse Time should not be more then 30ms (real time data) or 200ms (historical data). The autentication procedure has to be tested as well in this section.

Test ID: t2i0	
Components involved	Data Dispatcher, Authentication, DBMS
Input specification	Data x belonging to y, storing request
Output specification	Boolean: true
Requirements-goals involved	R6-G3
Description	To test the communication between DBMS and the Backbone On Server in upwload some data requests have to be performed. The communication should be effective and fast enough: Database Rensponse Time should not be more then 30ms. Whether the storage is performed correctly, the return value expected is true.

Test ID: t3i0	
Components involved	Data Dispatcher, Authentication, DBMS
Input specification	Data x that belonging to y not stored in DBMS request
Output specification	Warning
Requirements-goals involved	R8-G3, G4, G6
Description	The system has to warn the user that the requested data are not present in DBMS. The communication should be effective and fast enough: Database Rensponse Time should not be more then 30ms (real time data) or 200ms (historical data).

Test ID: t4i0	
Components involved	Data Dispatcher, Authentication, DBMS
Input specification	Several requests of data x, stored or not in DBMS, have to be performed
Output specification	Data requested
Requirements-goals involved	R8, R13.1,R13.2,R7-G4, G6
Description	A great number of requests (more than 1000) should be performed to test the system availability. This is necessary in order to be sure to have a proper backbone system for the application. For each request Database Rensponse Time should not be more then 30ms (real time data) or 200ms (historical data). According to requirements system availability must be at least 99.995%.

5.4.2 1st integration phase

The following test should be performed in order to test the behaviour of the Backbone On Client. Stub objects and oracles should be used instead of the not yet implemented components.

Test ID: t1i1	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher
Input specification	Several storing requests x belonging to y
Output specification	boolean
Requirements-goals involved	R6-G3
Description	Syncronization of client data test: if the soring request is successful a true value is returned, otherwise (false value must be returned) is necessary to check whether data are still memorized (client side).

5.4.3 2nd integration phase

All the following test will be performed in order to test the integration between modules of the Backbone and the application servers. AuthenticationService, DataReciver and DataDispatcher are the interfaces involved in this integration phase. Each of the following test should be performed using different mocks (one per applicaiton). Stub objects and oracles should be used instead of the not yet implemented components.

Test ID: t1i2	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher, D4H Application Server
Input specification	Individual user or third-party user registration data
Output specification	boolean
Requirements-goals involved	R4, R4.1, R4.2-G2
Description	By the submission of all the required data a client should be able to register successfully. In this case a true boolean value should be returned. Some attempts with incomplete or wrong data submissions should be operated in order to test the system. In this case a false boolean value should be returned.

Test ID: t2i2	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher, D4H Application Server
Input specification	Individual user or third-party user update info
Output specification	boolean
Requirements-goals involved	R4-G2
Description	Individual or third-party users should be able to update of one or more personal data. In case the process end succesfully a true boolean value should be returned. Some attempts with incomplete or wrong updates should be operated in order to test the system. In this case a false boolean value should be returned.

Test ID: t3i2	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher, D4H Application Server
Input specification	Individual user or third-party user login data
Output specification	boolean
Requirements-goals involved	R1-G1
Description	By the submission of all the required data a client should be able to register successfully. In this case a true boolean value should be returned. Some attempts with incomplete or wrong login data submissions should be operated in order to test the login system. In this case a false boolean value should be returned.

Test ID: t4i2	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher, D4H Application Server, T4R Application Server, ASOS Application Server
Input specification	Data event
Output specification	
Requirements-goals involved	R2-G1
Description	By this test is possible to check the update of all the involved application server when a data event occurs

Test ID: t5i2	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher, D4H Application Server
Input specification	IU, DataReciver
Output specification	boolean
Requirements-goals involved	R7, R13.1-G4,G6
Description	This test should verify whether third party users are able to subscribe or unsubscribe to data. Also not well formulated requests should be performed in order to test the subscription system. In this case a false boolean value should be returned.

The following tests should be performed firstly in the integration of the modules' components. This first stage doesn't depend on integration phases and can be performed as soon as the components are created according to section 5.3.1, exploiting stub objects and oracles where needed.

5.4.4 3rd and 4th integration phase

During those integration processes stub objects and oracles should be gradually substituded by implemented components. In particular the integrationd should regard client modules with the Backbone on client and client modules with the respective server modules.

Test ID: t1i3/4	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher, BlockedTPUsManager, D4H Router
Input specification	IU, TPU
Output specification	
Requirements-goals involved	R11.2-G5
Description	This test should verify whether Individual users can block correctly a data requests sender.

Test ID: t2i3/4	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher, D4H Router, Request Status Manager
Input specification	IU, TPU, subscription
Output specification	int (requestID)
Requirements-goals involved	R7, R7.1,R8-G4
Description	This test should verify whether third-party users are able to formulate properly individual data requests. If the request is correctly formulated, a univocal request ID is generated.

Test ID: t3i3/4	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher, D4H Router, Request Status Manager
Input specification	createAnonymRequest input parameters
Output specification	int (requestID)
Requirements-goals involved	R13.1, R13.2 ,R8-G6
Description	This test should verify whether third-party users are able to formulate properly group data requests. If the request is correctly formulated, a univocal request ID is generated.

Test ID: t4i3/4	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher, D4H Router, Request Status Manager, Anonymity Evaluator
Input specification	requestID
Output specification	boolean
Requirements-goals involved	R10, R11.1, R11.2, R12-G5
Description	This test should verify whether users are able to cancel requests and individual users can deny or approve a single data request. Whether the action is performed correctly a true value should be returned.

Test ID: t5i3/4	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher, ASOS view, ASOS router, T4R view, T4R router, ASOS activation, T4R activation
Input specification	IU, activated (boolean)
Output specification	boolean
Requirements-goals involved	G7, G8,G9,G10
Description	This test should verify whether individual users are able activated or deactivated the ASOS and T4R services. Whether the action is performed correctly a true value should be returned.

Test ID: t6i3/4	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher, ASOS view, ASOS router, Health Data analyzer
Input specification	IU
Output specification	int
Requirements-goals involved	G7
Description	This test should verify whether individual users can get their data parameters correctly.

Test ID: t7i3/4	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher, T4R view, T4R router, Run Status Manager
Input specification	createRun/updateRun input parameters
Output specification	int (run identificationNumber)/boolean
Requirements-goals involved	R16.1, R16.2, R17-G8
Description	This test should verify whether third-party users are able to create or update their own running competitions.

Test ID: t9i3/4	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher, T4R view, T4R router, Run Status Manager
Input specification	getCloseRun/cancelRun/getRun input parameters
Output specification	Run, boolean
Requirements-goals involved	R16.2, R17-G10, G8
Description	This test should verify whether third-party users are able to cancel a running competition they've previously organized. It should also verify whether individual users are able to select a particular set of running competition from the map.

Test ID: t10i3/4	
Components involved	Data Dispatcher, Authentication, DBMS, IU Data Dispatcher, T4R view, T4R router, Run Status Manager
Input specification	runIdentificationNumber, IU participant
Output specification	boolean
Requirements-goals involved	R18.1, R18.2-G9
Description	This test should verify whether individual users can enroll or unenroll to an organized competition.

6. Effort Spent

6.1 ARGIRO' ANNA SOFIA

DATE	DESCRIPTION OF THE TASK	HOURS SPENT
27/11/18	group work	3
2/12/18	high level overview	4
2/12/18	group work	4
8/12/18	Architecture revision, Introduction	4
8/12/18	group work	4

6.2 BATTAGLIA GABRIELE

DATE	DESCRIPTION OF THE TASK	HOURS SPENT
27/11/18	group work	3
30/11/18	component view	4
2/12/18	model diagrams	4
2/12/18	group work	4
6/12/18	Components interfaces	8
8/12/18	Deployement, sequence diagram	4
8/12/18	group work	4

6.3 CASASOLE BERNARDO

DATE	DESCRIPTION OF THE TASK	HOURS SPENT
27/11/18	group work	3
2/12/18	User interface design	4
2/12/18	group work	4
5/12/18	Implementation and testing	5
8/12/18	Implementation and testing	4
8/12/18	group work	4

7. References

7.1 Reference Documents

API

- https://en.wikipedia.org/wiki/Representational_state_transfer
- https://realtimeapi.io/hub/realtime-api-design-guide/

7.2 Software

- TeXWorks v0.6.2
- Draw.io v9.4.1
- proto.io v6.3.2.3