POLITECNICO
MILANO 1863

Software Engineering 2
Design Document

Work team:
Paolo Romeo, Andrea Scotti, Francesco Staccone

AY 2018-2019

Table of contents

1. Introduction

1.1. *Purpose*

This document is thought to be an overview of the TrackMe application, in which is explained how to satisfy the several project requirements stated in the RASD. This document is principally intended for the developers and the testers, with the purpose of providing a functional description of the main architectural components, their interfaces and their interactions, along with the design patterns.

1.2. *Scope*

The TrackMe system is thought to be a Health Data Management and Run-Friendly Mobile App, useful for a quite wide range of users with different features and necessities. The software will be made up of three different modules: Data4Help, a service that allows the third parties to monitor the location and the health status of the individuals; AutomatedSOS , an optional service that monitors the health status of the users by performing a continuous evaluation of the health parameters' data acquired by Data4Help; Track4Run, a service thought for the organization of running events.
To provide the previous services in the best way, the overall system is structured in a multi-tier architecture. More specifically, the Business logic layer has the task of taking charge of the incoming requests/data, computing checks, and interacting with external third-party services through the use of interfaces. This layer is connected with the Data layer, in which are stored all the Users data (credentials, health data). The Presentation layer is build through the hybrid Client paradigm in which the client needs to perform computation for the partial analysis and filtering of incoming data.

1.3. *Definitions, acronyms, abbreviations*

1.3.1. Definitions

- Individual: a registered person who can use the TrackMe services.
- Third party: a registered entity that can use the TrackMe services.
- Client: A client is a piece of computer hardware or software that accesses a service made available by a server;
- Firewall: A network security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules;
- Server: A computer program or a device that provides functionality for other programs or devices, called "clients".

1.3.2. Acronyms

- API: Application Program Interface;
- DBMS: Database Management System;
- DD: Design Document
- GUI: Graphical User Interface;
- HTTP: Hypet Text Transfer Protocol;
- MVC: Model View Controller pattern;
- OS: Operating System;
- RASD: Requirements Analysis and Specifications Document;
- REST: REpresentational State Transfer;

1.3.3. Abbreviations

- Gn: n-goal in the RASD;
- Rn: n-functional requirement in the RASD;

1.4. *Revision history*

- 10/12/2018 Version 1
- 13/1/2019 Version 2 - Minor changes to the requirements

1.5. *Reference documents*

- RASD document;
- Mandatory project assignment;
- Slides of the lessons.

1.6. *Document structure*

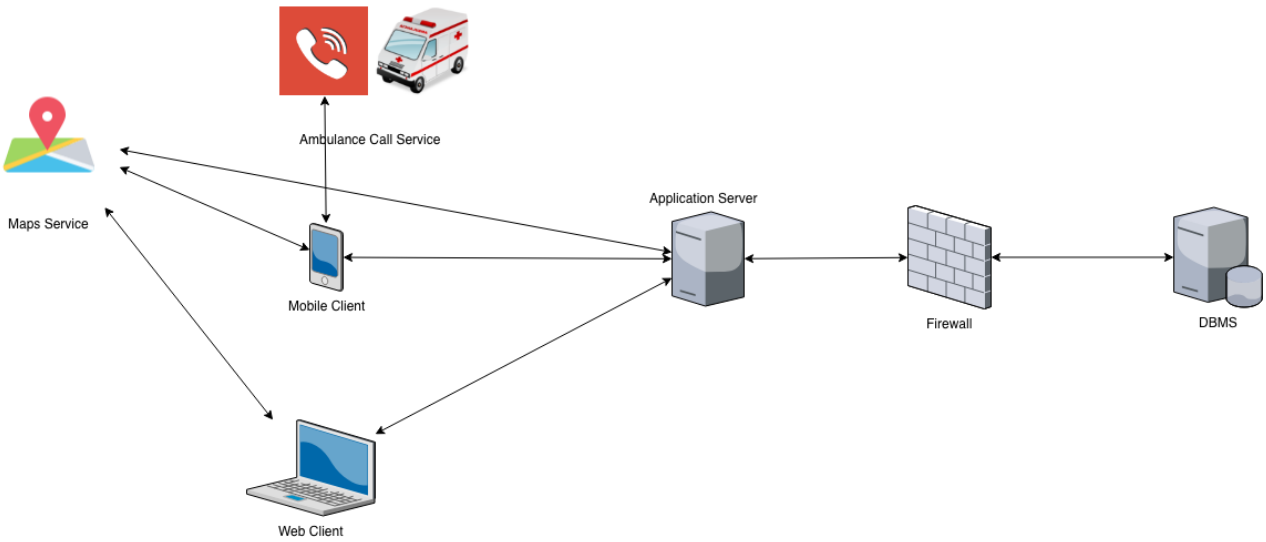
The following document is organised in this way:

- Architectural Design: this section shows the main components of the system and the connections among them. It will also focus on design choices, styles and patterns.
- User Interface Design: this section includes an improvement of the user interface given in the RASD document. It will be described through the use of UX modeling.
- Requirements Traceability: this section shows how the requirements in the RASD are mapped to the design components presented in the DD;
- Implementation, Integration and Test plan: this section shows the order in which the implementation and the integration of the subcomponents will occur and how the integration will be tested.

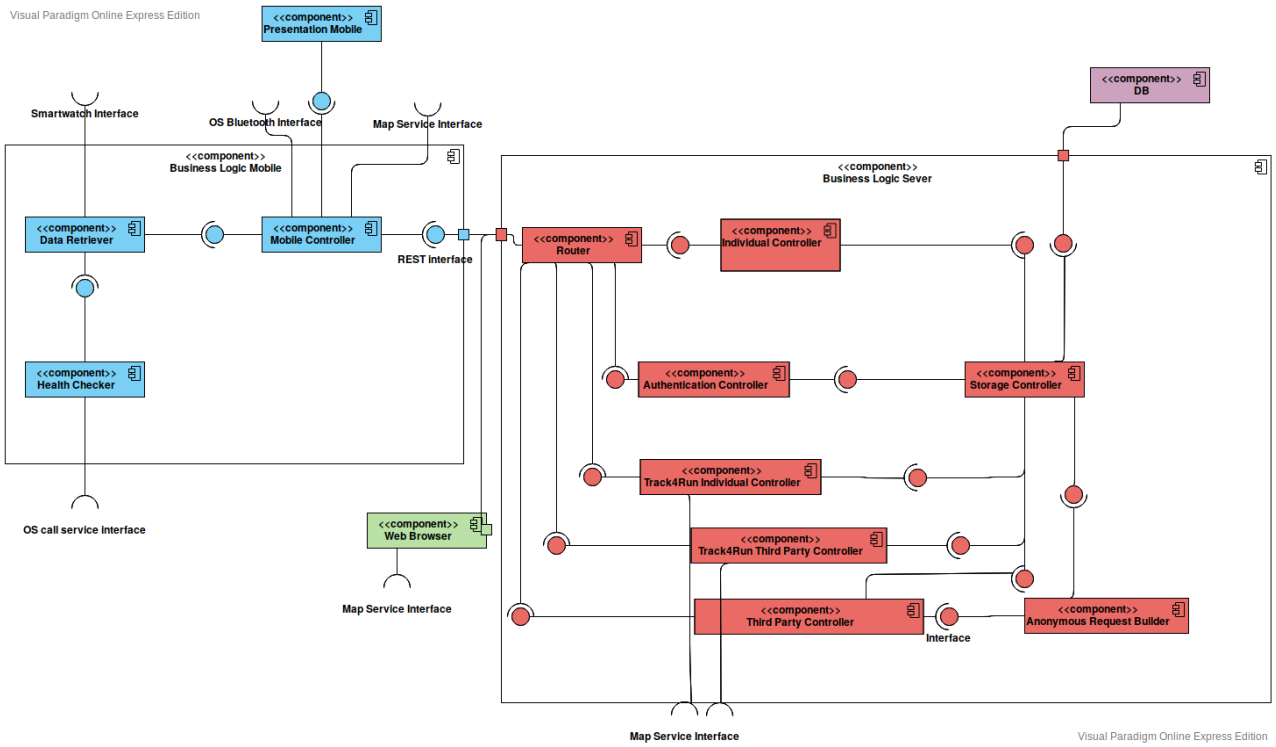
2. Architectural design

2.1. *Overview: high-level components and their interaction*

The figure below represents an high level overview of the system. Further details on the system components and their interaction will be further explained in the next sections.



2.2. *Component view*



The UML component diagram shows the internal structure of the system highlighting the individual modules and the connections among them. It models the static implementation view of a system by breaking it down into various high levels of functionality. Individual components are wired together by using an assembly connector to connect the required interface of one component with the provided interface of another component. For simplicity reasons, the interfaces between REST controllers (Individual, Third Party, Track4Run Individual and Track4Run Third Party Controller) and the Authentication Controller have not been drawn in the diagram. However they are present in the system in order to validate the successive user sessions after the log in. Below is the description of the components:

- **Mobile Controller**
  This is the component that allows the individual mobile application to access to the API of the Business Logic Server. It implements the interface dedicated to the transfer of data from client to server and it translates the individual user interactions in REST API calls. If internet connection is not available or it's too slow, older health and data are not saved locally but discarded by giving priority to new data.

- **Data Retriever**
  This component retrieves health data from smartwatches or similar devices, it filters them and forwards them to the Health Checker component, if the AutomatedSOS service is enabled, and to the Mobile Controller component.

- **Health Checker**
  This component analyzes in real time health data of the individual and checks if his or her health is fine. If it's not, it interacts with the operating system of the running application in order to make a SOS call.

- **Presentation**
  This is the component that is aimed to display the front-end content to the individual and to allow him or her to interact with the application.

- **Web Browser**
  The system allows the third party to interact with it through any web browser.

- **Router**
  It defines which controller entry points correspond to a given HTTP URL, and how parameters are to be read from the HTTP request.

- **Individual Controller**
  The main role of this component is to manage the data coming from the specific individual by invoking the proper methods of the Storage Controller. It also provides methods to accept/refuse a request of agreement and to change personal information or preferences. Furthermore it notifies the client if there are new requests or info for him.

- **Third Party Controller**
  This component provides methods that allow the third party to send invidual and anonymous requests to the system. If there are new individual agreements or new data available, it sends them to the third party.

- **Authentication Controller**
  This component provides authentication and registration processes for both the individuals and the third parties.

- **Track4Run Individual Controller**
  This component provides the methods to permit an individual to join, leave, check and watch a run. It communicates with the external service that provides maps through the Map services interface. Every time a user wants to join a run, it checks if all the constraints are satisfied (for example he can't join two overlapping runs). Finally, it keeps updated the individual about new info by sending him or her notifications.

- **Track4Run Third Party Controller**
  This component provides the methods to allow a third party to organize a run. It communicates with the external service that provides maps trough the Map services interface to allow the third party to select the track for the run.

- **Storage Controller** This component provides the methods for querying and updating the Database.

- **Anonymous Request Builder** This component aggregates data in order to answer to third party group requests by assuring anonymity.

- **Database** This component represent the DBMS. It provides the interfaces to retrieve and store data. In the database there are data about users, third parties and the set of agreements among them. It also contains data about the runs.

- **External services interfaces**
  These interfaces are not implemented by the system but are provided by external services in order to run the application.
  - **Map Service Interface**
    It allows the third party to select the track and to show the position of runners in real time on the map.

  - **OS Call Service Interface**
    It allows the mobile application to make a SOS phone call in case of emergency.

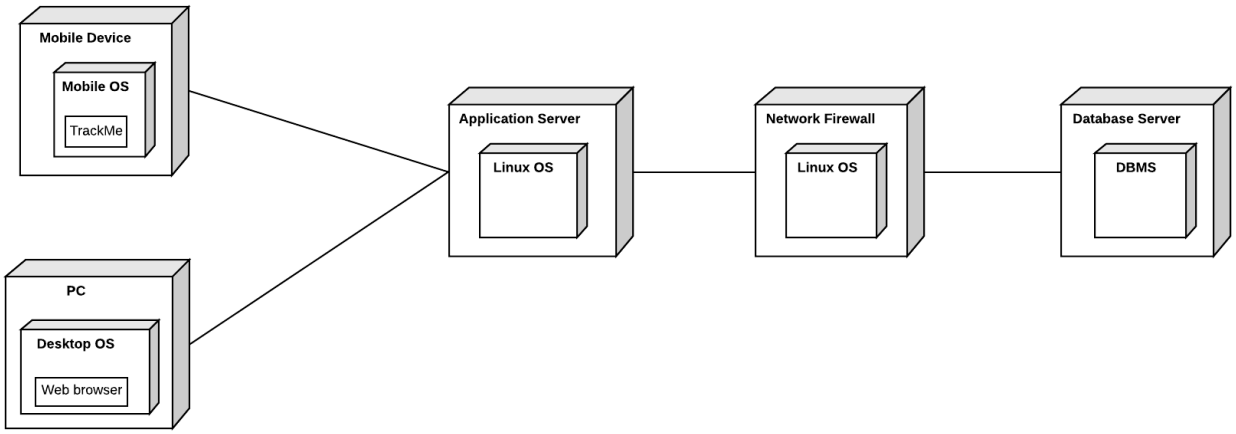
  - **OS Bluetooth Interface**
    It allows the mobile application to connect the smartphone to smartwatches or similar de-

vices.

- **Smartwatch Interface**
  It allows the mobile application to get data from smartwatches.

2.3. *Deployment view*

The figure below shows the deployment diagram of the whole system. Its main goal is to describe the distribution of components capturing the topology of the system's hardware.



As previously stated in the section 1, the system is structured in a multi-tier architecture. The specific role of each node is clarified here:

Clients
The first tier is composed by the clients machines (mobile for individuals and desktop for third parties). The individuals will be able to access TrackMe functionalities through the dedicated native application while the third parties through any web browser.

Application Server
This is the middleware level of the architecture: all the business logic of the system is contained in this server.

Network Firewall
The access to the Database is mediated by a network firewall in order to avoid unauthorised access to the data and the credentials of the user.

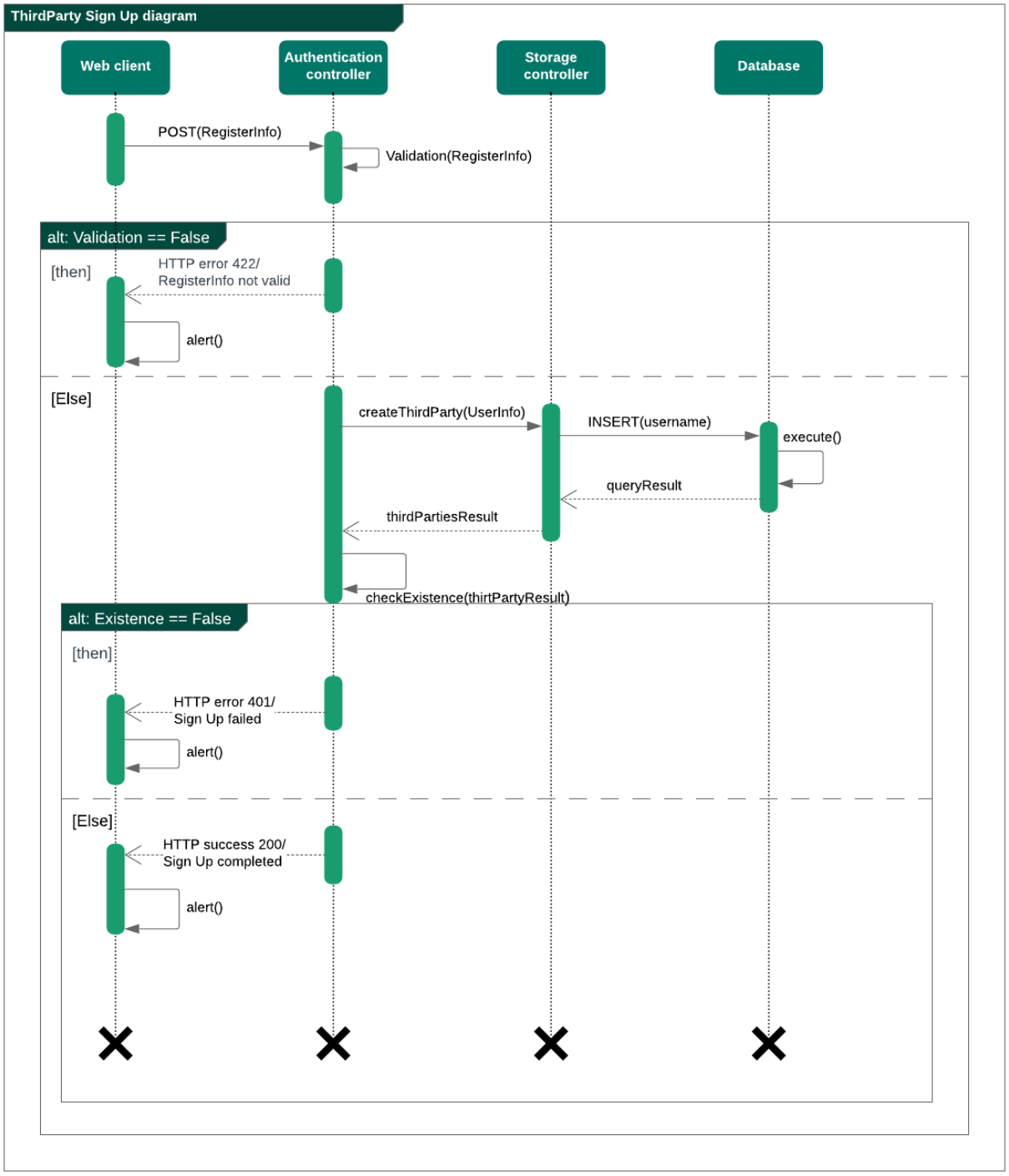
Database Server
This is the last layer of the architecture: all the data are stored in a Database Server accessed through a relational DBMS.

2.4. *Runtime view*

2.4.1. Sign up Runtime View

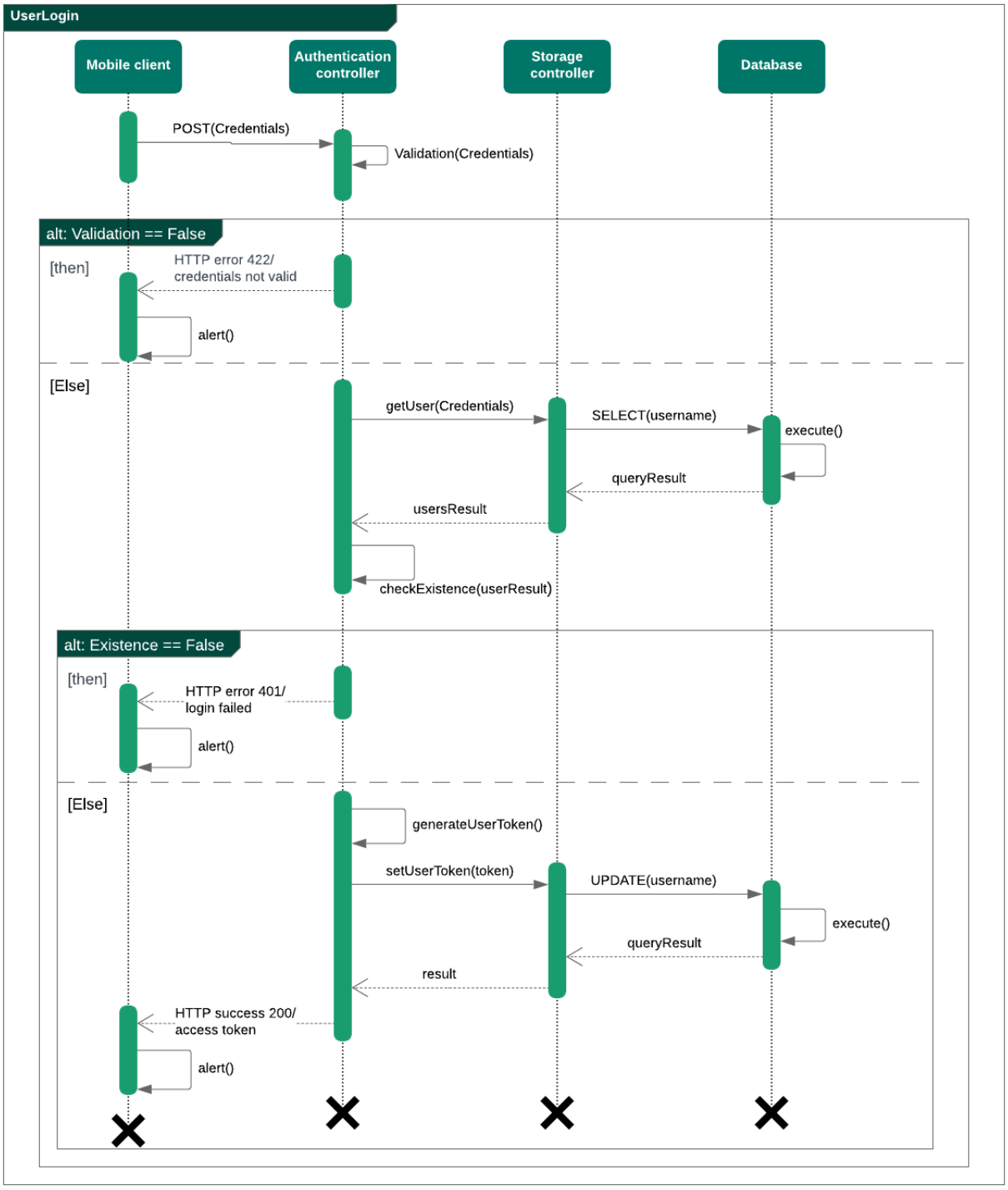
Third parties and individuals have to sign up to the application before being able to access its functionalities. The registration process is the same for both third parties and individuals, the only thing that differentiates it is the information provided by them. The third party must fill up a form with its VAT number and a password. To reach the same goal an individual must fill up a form with username, password and personal data. The information is serialized and then sent to the Application Server through an HTTP POST request. The Authorization controller

handles the request and, through the Storage interface, creates a new entry in the User table or in the ThirdParty table on the Database.



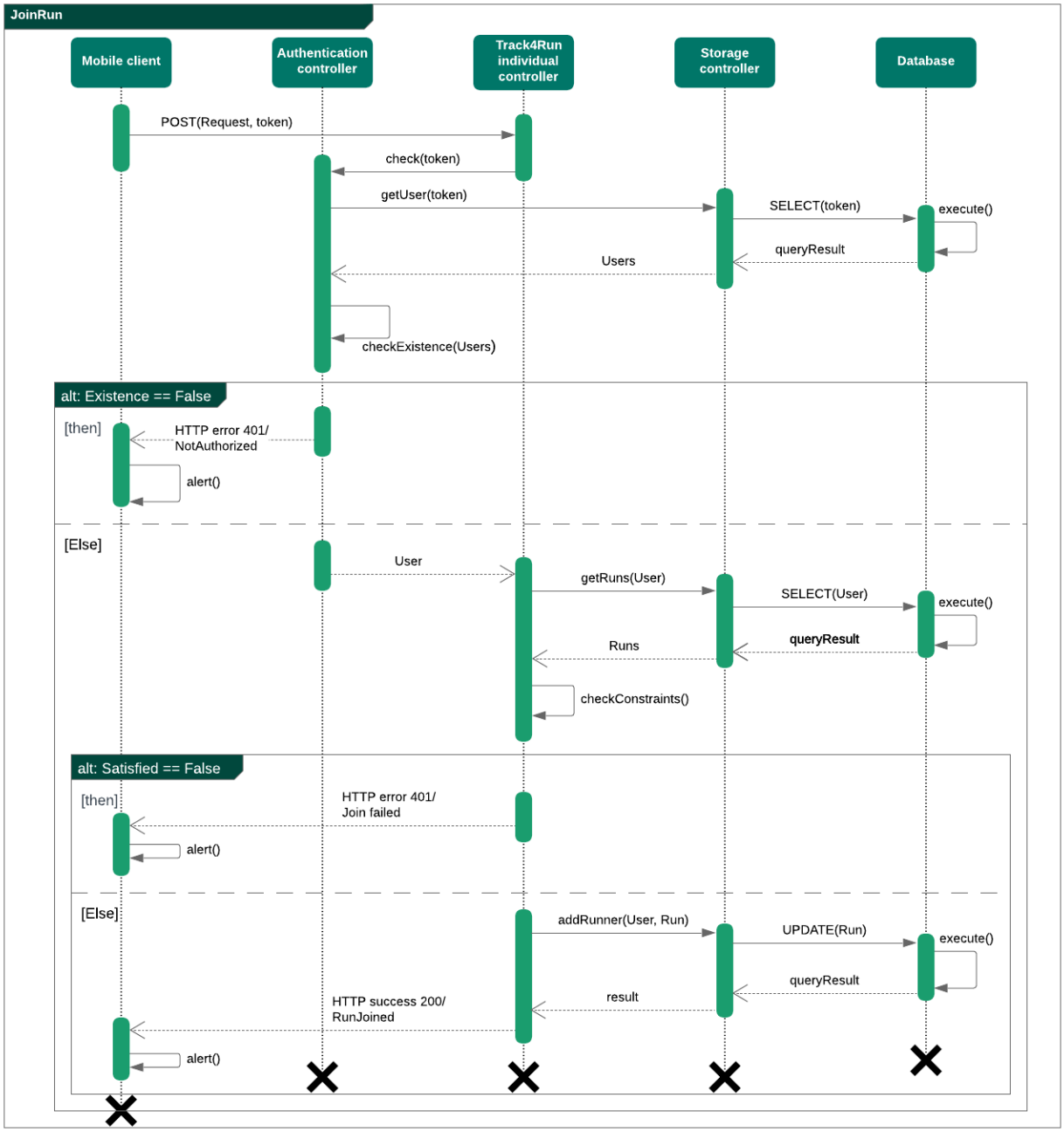
2.4.2. User Login Runtime View

Third parties and individuals have to be logged to the application before being able to access its functionalities. Both third parties and users submit the login information through and HTTP POST request. The request is handled by the Authentication Controller that validates the request and checks if the Database has an entry for the requested account. If the entity is present and the credentials are correct, the Authentication Controller generates an Access Token, sets it as the current access Token for the specified entity and returns it to the entity itself. The following sequence diagram shows how the process works in detail for the login of an individual user.



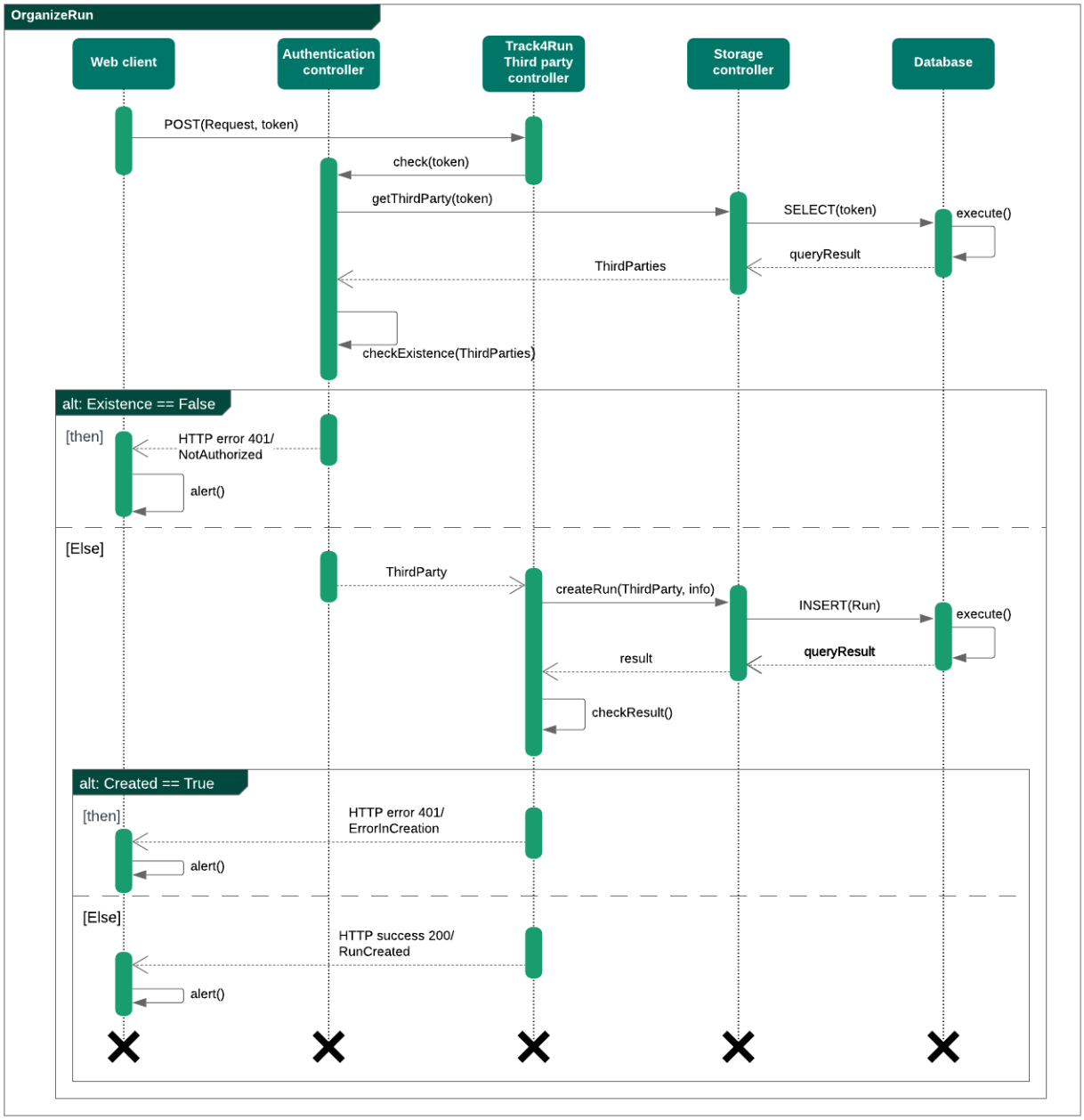
2.4.3. Join a run Runtime View

The User opens the Track4Run section of the application and, after a GET request to the Application Server he/she receives the list of all the available runs and he/she can choose to join one of them (if there is at least one displayed run). The User can send a POST request containg the Access Token and the selected run to the Track4Run Individual Controller. The Track4Run Individual Controller passes the Access Token to the Authentication Controller that checks if the Mobile Client provided a valid Token. If the Authentication succeeds the control goes back to the Track4Run Individual Controller that checks if the user has already joined an overlapping run or if the maximum number of participants was reached. If the control is satisfied the Track4Run Individual Controller asks the Storage Controller to update the tuple of the selected run in the Database. To avoid repetitions, the following sequence diagram shows the process assuming that the User already received the list of available runs.



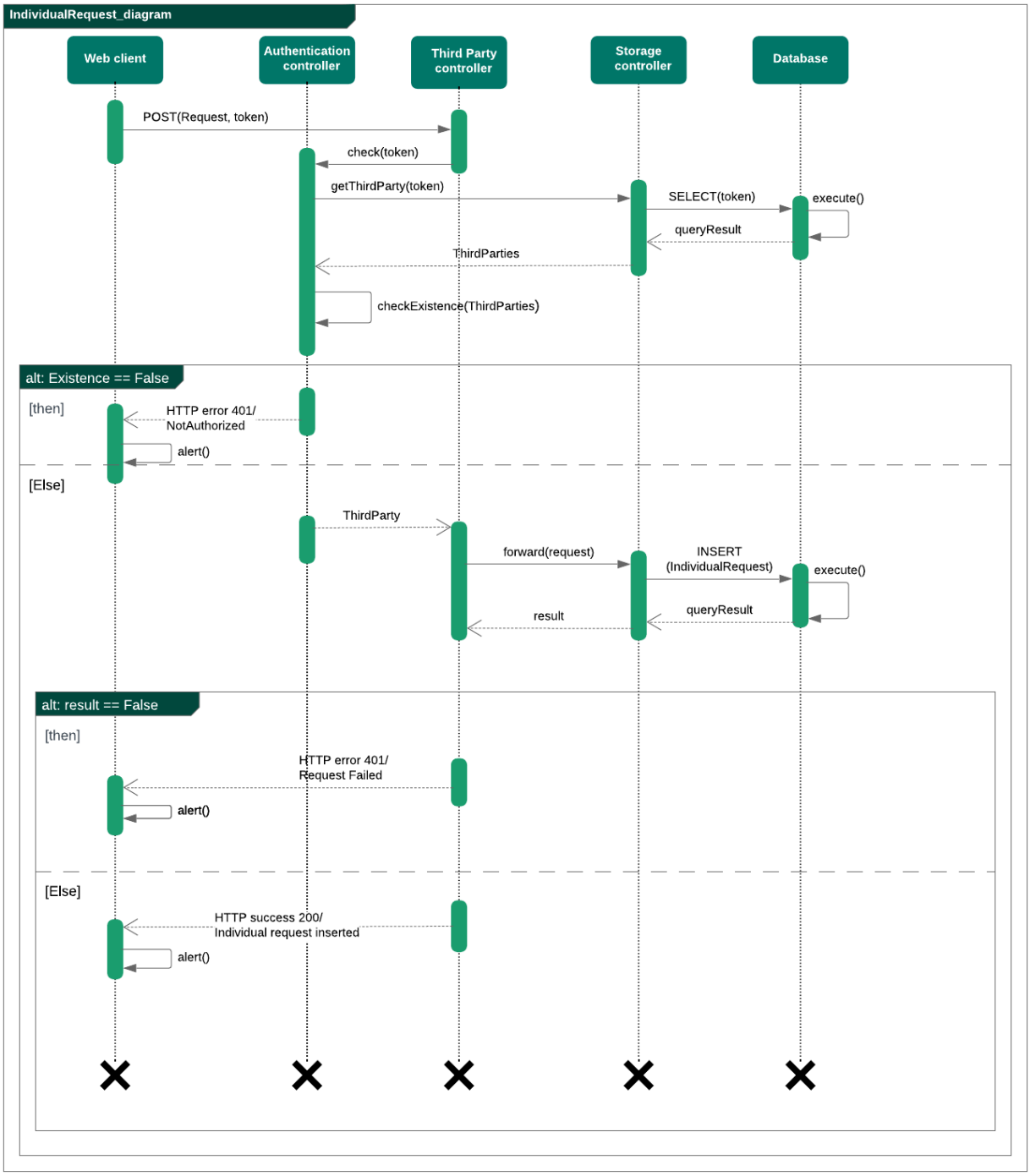
2.4.4. Organise a run Runtime View

The Third party opens the Track4Run section of the application and, after filling the form containing the information about the run it can send a POST request containg the Access Token and the filled form to the Track4Run Third Party Controller. The Track4Run Third Party Controller passes the Access Token to the Authentication Controller that checks if the Web Client provided a valid Token. If the Authentication succeeds the control goes back to the Track4Run Third Party Controller that asks the Storage Controller to insert a new tuple for the run in the Database.



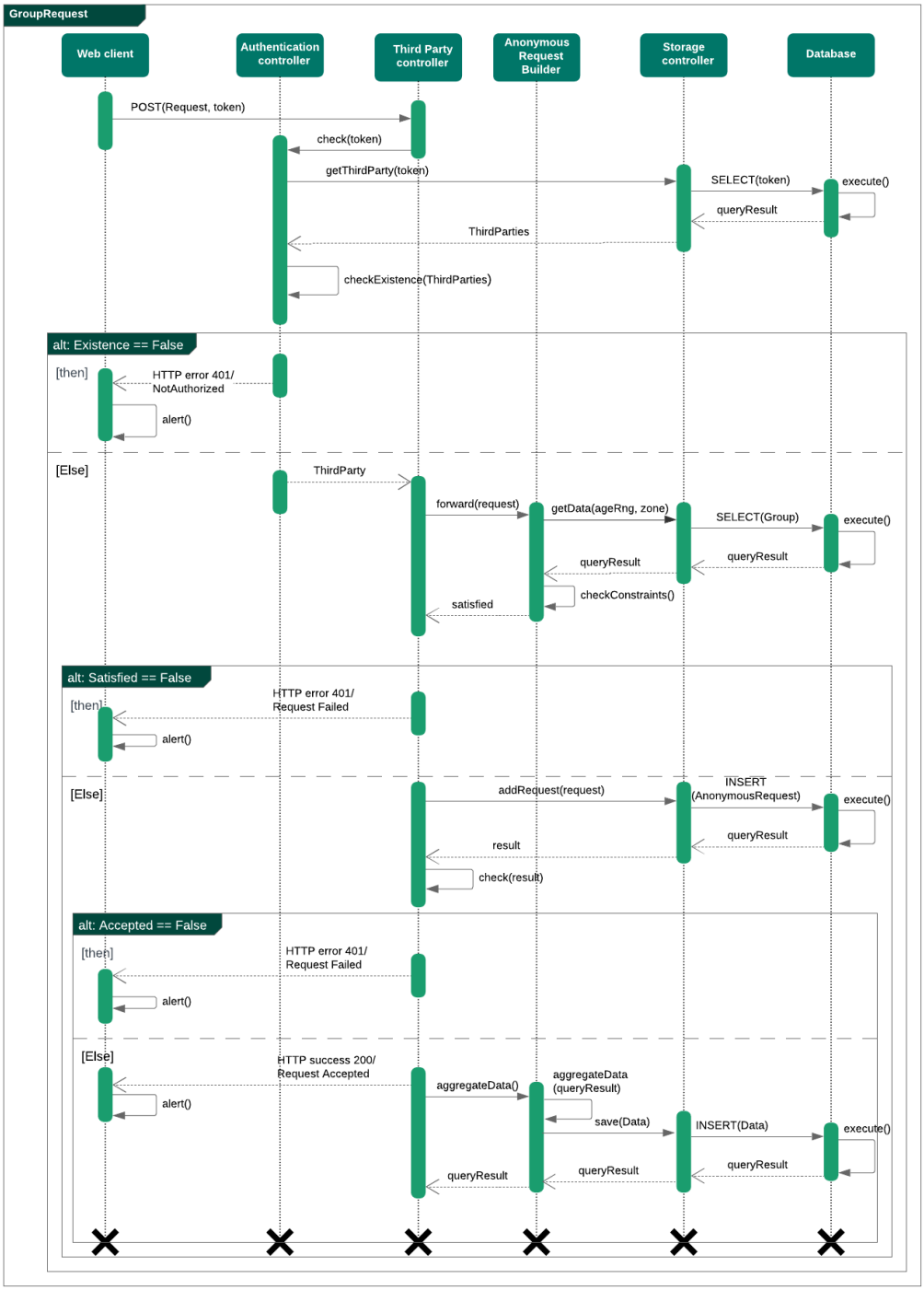
2.4.5. Individual request Runtime View

The Third party opens the Data4Help section of the application and, after filling the form containing the information about the request, it can send a POST request containg the Access Token and the filled form to the Third Party controller. The Third Party Controller passes the Access Token to the Authentication Controller that checks if the Web Client provided a valid Token. If the Authentication succeeds, the Third Party Controller forwards the request to the Storage Controller, that adds an instance of the request into the Database. Then, if the request is successfully inserted, the Third party is notified. When this process is completed and the answer to the previous request is ready, the system will notify the Third Party as soon as it will ask for notifications.



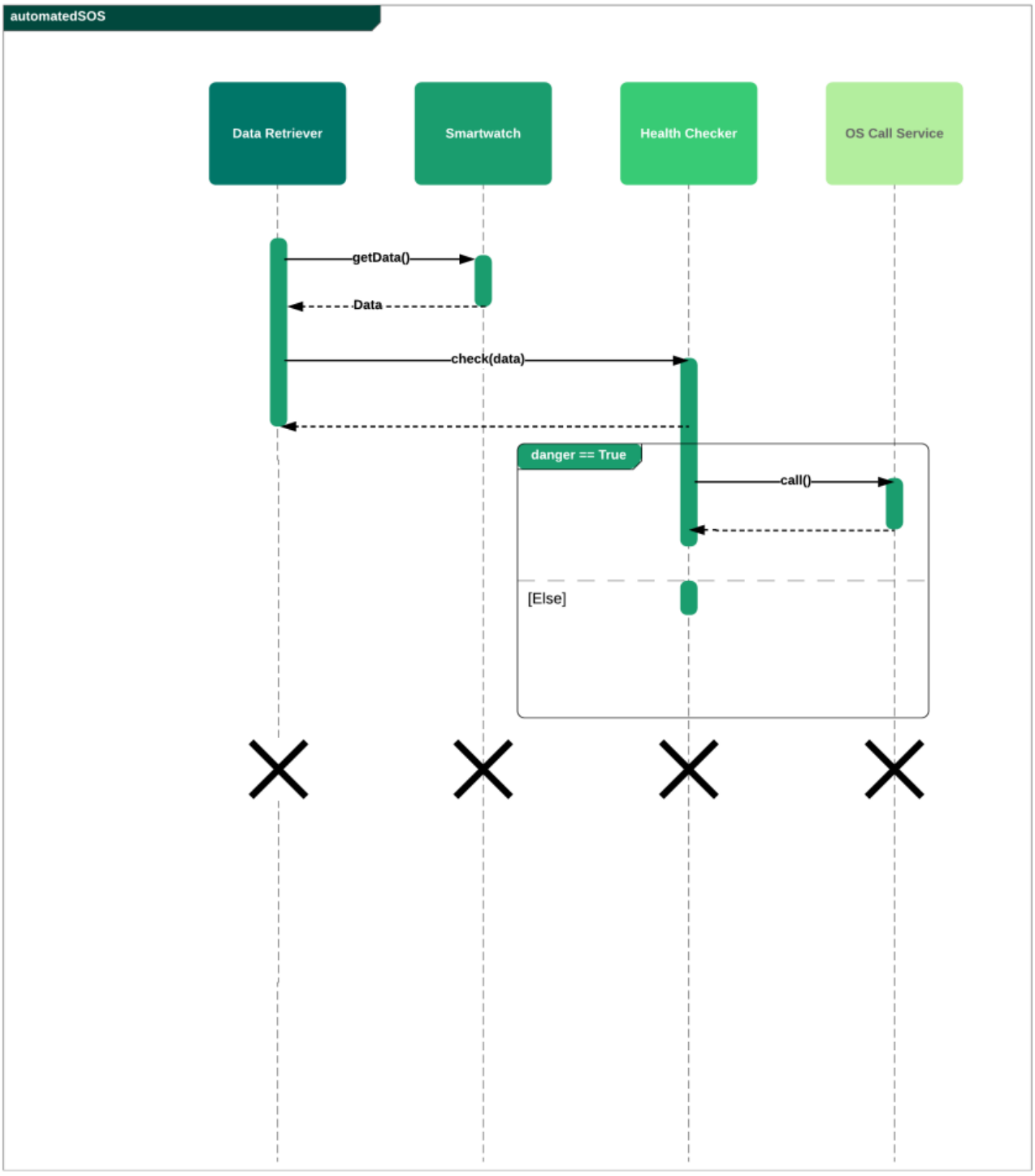
2.4.6. Group request Runtime View

The Third party opens the Data4Help section of the application and, after filling the form containing the information about the request, it can send a POST request containg the Access Token and the filled form to the Third Party controller. The Third Party Controller passes the Access Token to the Authentication Controller that checks if the Web Client provided a valid Token. If the Authentication succeeds, the Third Party Controller asks the Anonymuos Request Builder to get the data from the Database, through the Storage Controller. If the constraints on data are satisfied, the Third Party Controller asks the Storage Controller to add an instance of the request into the Database. Then, if the request is successfully inserted, the Third party is notified and the Third Party Controller communicates the Anonymous Request Builder to aggregate the received data.



2.4.7. AutomatedSOS Runtime View

This runtime view shows the interactions among mobile client components, from the retrieving of data, from the smartwatch, to the eventual SOS call.



2.5. *Component interfaces*

2.5.1. *REST API*

REST (REpresentation State Transfer) describes an architectural style of networked systems such as this application. One of the most important REST principles for Web applications is that the interaction between the client and server is stateless between requests. Each request from the client to the server must contain all of the information necessary to understand the request. The client wouldn't notice if the server were to be restarted at any point between the requests. Additionally, stateless requests are free to be answered by any available server, which is appropriate for an environment such as cloud computing. On the server side, the application state and functionality are divided into resources. A resource is an item of interest, a conceptual identity that is exposed to the clients. Every resource is uniquely addressable using a URI (Universal Resource Identifier). All resources share a uniform interface for the transfer of state between client and server. Standard HTTP methods such as GET, PUT, POST, and DELETE are used. Hypermedia is the engine of the application state, and resource representations are interconnected by hyperlinks, as described partially in this section and section 3.

- **Authentication Controller**

  Third Party SignUp

endpoint	*/auth/thirdparty/signUp
method	POST
url params	
data params	VAT: [alphanumeric] name: [alphanumeric] password : [alphanumeric]
success response	code: 200
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 500 INTERNAL SERVER ERROR Content : {error: "Could not commit JPA transaction"} code: 409 CONFLICT Content : {error: "This user already exists"}
Notes	Allows a third party to register to the system.

Individual SignUp

endpoint	*/auth/individual/signUp
method	POST
url params	
data params	username: [alphanumeric] password : [alphanumeric] name: [text] surname: [text] latitude: [numeric] longitude: [numeric] birthDate: [date]
success response	code: 200
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 500 INTERNAL SERVER ERROR Content : {error: "Could not commit JPA transaction"} code: 409 CONFLICT Content : {error: "This user already exists"}
Notes	Allows an individual to register to the system.

Login

endpoint	*/auth
method	POST
url params	
data params	username: [alphanumeric] password : [alphanumeric]
success response	code: 200 Content : {token: [alphanumeric]}
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} Code: 401 UNAUTHORIZED Content : {error: "Bad Credentials!"}
Notes	Allows an individual or a third party to obtain an authentication Token

NOTE:

– all successive requests need to contain the token, retrieved during the login request, in order to be allowed;

– if a logged in user try to use REST API trying to be another user a excepction is launched (code: 401 UNAUTHORIZED Content : {error: "Trying to be another user!"}.

- **Third Party Controller**

Make and individual request

endpoint	*thirdparty/individualRequest
method	POST
url params	
data params	vat: [alphanumeric] fiscalcode: [alphanumeric] subscribedToNewData: [boolean]
success response	code: 200
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 409 CONFLICT Content : {error: "This user already exists"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Third Party Not Found"} code: 404 NOT FOUND Content : {error: "Individual Not Found"} Code: 401 CONFLICT Content : {error: ""This request has been already done""}
Notes	Allows the third party to do an individual request of data.

Make and anonymous request

endpoint	*thirdparty/anonymousRequest
method	POST
url params	
data params	vat: [alphanumeric] startAge: [numeric] endAge: [numeric] lat1: [float] lon1: [float] lat2: [float] lon2: [float] subscribedToNewData: [boolean]
success response	code: 200
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 500 INTERNAL SERVER ERROR Content : {error: "Could not commit JPA transaction"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Third Party Not Found"}
Notes	Allows the third party to do a group request of data.

Modify Third Party Password

endpoint	*/thirdParty/username/changePassword
method	PUT
url params	username: [alphanumeric]
data params	newPassword: [alphanumeric] oldPassword:[alphanumeric]
success response	Code: 200
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Third Party Not Found"} code: 422 UNPROCESSABLE ENTITY Content : {error: "Bad Credentials"} code: 422 UNPROCESSABLE ENTITY Content : {error: "Data are not well formed"}
Notes	Allows a third party to change its password.

Get individual requests

endpoint	*/thirdParty/{thirdParty}/individualRequests
method	GET
data params	
url params	vat: [alphanumeric]
success response	code: 200 Content : {individualRequests: List¡IndividualRequest¿}
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Third Party Not Found"}
Notes	Allows the third parties to request for all individual requests it has done.

Get individual request notifications

endpoint	*/thirdParty/{thirdParty}/notifications/individualRequests
method	GET
data params	
url params	vat: [alphanumeric]
success response	code: 200 Content : {notifications: List¡IndividualRequest¿}
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Third Party Not Found"}
Notes	Allows the third parties to request for notifications of individual requests.

Get individual request notifications counter

endpoint	*/thirdParty/{thirdParty}/notifications/countIndividualRequests
method	GET
data params	
url params	vat: [alphanumeric]
success response	code: 200 Content : {counter: [integer]}
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Third Party Not Found"}
Notes	Allows the third parties to request for the number of new notifications of individual requests.

Get past individual data

endpoint	*/thirdParty/{thirdParty}/{individual}/data
method	GET
data params	
url params	vat: [alphanumeric] fiscalCode: [alphanumeric]
success response	code: 200 Content : {individual data: List<IndividualData>}
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Third Party Not Found"} code: 404 NOT FOUND Content : {error: "Individual Not Found"} code: 404 NOT FOUND Content : {error: "The thirdParty has not the right to receive data from the individual because you never asked for it"} code: 400 BAD REQUEST Content : {error: "You can't acces this data"}
Notes	Allows the third parties to request for past data of a specific individual.

Get new individual data

endpoint	*/thirdParty/{thirdParty}/notifications/{individual}
method	GET
data params	
url params	vat: [alphanumeric] fiscalCode: [alphanumeric]
success response	code: 200 Content : {notifications: List<IndividualData>}
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Third Party Not Found"} code: 404 NOT FOUND Content : {error: "Individual Not Found"} code: 404 NOT FOUND Content : {error: "The thirdParty has not the right to receive data from the individual because you never asked for it"} code: 400 BAD REQUEST Content : {error: "You can't acces this data"}
Notes	Allows the third parties to request for new data of a specific individual.

Get anonymous requests

endpoint	*/thirdParty/{thirdParty}/anonymousRequests
method	GET
data params	
url params	vat: [alphanumeric]
success response	code: 200 Content : {anonymousRequests: List<AnonymousRequest>}
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Third Party Not Found"}
Notes	Allows the third parties to request for all anonymous requests it has done.

Get past anonymous request data

endpoint	*/thirdParty/{thirdParty}/anonymousAnswer/{anonymousRequest}
method	GET
data params	
url params	vat: [alphanumeric] anonymousRequestId: [alphanumeric]
success response	code: 200 Content : {anonymous answers: List<AnonymousAnswer>}
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Third Party Not Found"} code: 404 NOT FOUND Content : {error: "Anonymous Request Not Found"} code: 400 BAD REQUEST Content : {error: "Not your request"}
Notes	Allows the third parties to request for past data of an anonymous request.

Get new anonymous request data

endpoint	*/thirdParty/{thirdParty}/anonymousAnswer/notifications/{anonymousRequest}
method	GET
data params	
url params	vat: [alphanumeric] anonymousRequestId: [alphanumeric]
success response	code: 200 Content : {anonymous answers: List<AnonymousAnswer>}
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Third Party Not Found"} code: 404 NOT FOUND Content : {error: "Anonymous Request Not Found"} code: 400 BAD REQUEST Content : {error: "Not your request"}
Notes	Allows the third parties to request for new data of an anonymous request.

- **Individual Controller**

Answer to Request

endpoint	*individual/individualRequest/answer
method	POST
url params	
data params	vat: [alphanumeric] fiscalCode: [alphanumeric] accepted: [boolean]
success response	code: 200
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Individual Request Not Found"} code: 404 NOT FOUND Content : {error: "Individual Found"}
Notes	Allows the individual to accept or refuse an individual request.

Modify Individual Password

endpoint	*/individual/username/changePassword
method	PUT
url params	username: [alphanumeric]
data params	newPassword: [alphanumeric] oldPassword:[alphanumeric]
success response	Code: 200
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Individual Not Found"} code: 422 UNPROCESSABLE ENTITY Content : {error: "Bad Credentials"} code: 422 UNPROCESSABLE ENTITY Content : {error: "Data are not well formed"}
Notes	Allows an individual to change its password.

Modify Individual Residence

endpoint	*/individual/username/changeLocation
method	PUT
url params	username: [alphanumeric]
data params	newLatitude: [float] newLongitude:[float]
success response	Code: 200
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 422 UNPROCESSABLE ENTITY Content : {error: "Provided values are not valid"}
Notes	Allows an individual to change its password.

Get individual pending requests

endpoint	*/individual/{individual}/individualRequests
method	GET
data params	
url params	fiscalCode: [alphanumeric]
success response	code: 200 Content : {individualRequests: List¡IndividualRequest¿}
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Individual Not Found"}
Notes	Allows the individual to request for all individual requests pending for him.

Get individual accepted requests

endpoint	*/individual/{individual}/acceptedRequests
method	GET
data params	
url params	fiscalCode: [alphanumeric]
success response	code: 200 Content : {individualRequests: List<IndividualRequest>}
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Individual Not Found"}
Notes	Allows the individual to request for all individual requests that he has already accepted.

Get individual request notifications

endpoint	*/individual/{individual}/notifications
method	GET
data params	
url params	fiscalCode: [alphanumeric]
success response	code: 200 Content : {notifications: List<IndividualRequest>}
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Individual Not Found"}
Notes	Allows the individual to request for all new individual requests for him (that he hasn't seen yet).

Get individual request notifications counter

endpoint	*/individual/{individual}/countNotifications
method	GET
data params	
url params	fiscalCode: [alphanumeric]
success response	code: 200 Content : {counter: [integer]}
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 NOT FOUND Content : {error: "Third Party Not Found"}
Notes	Allows an individual to request for the number of new notifications of individual requests.

Send Data

endpoint	*individual/{individual}/data
method	POST
url params	individual: [alphanumeric]
data params	individual data: { List<IndividualData>}
success response	code: 200
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 422 UNPROCESSABLE ENTITY Content : {error: "There is no data."}
Notes	Allows the individual to send data.

Send AutomatesSOS preference

endpoint	*individual/{username}/updateAutomatedSOS
method	POST
url params	username: [alphanumeric]
data params	preference: [boolean]
success response	code: 200
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 INDIVIDUAL NOT FOUND Content : {error: "Individual not found."}
Notes	Allows the individual to send his preference about automatedSOS.

Get AutomatesSOS preference

endpoint	*individual/{username}/updateAutomatedSOS
method	GET
url params	username: [alphanumeric]
data params	
success response	code: 200 Content : {preference: [boolean]}
error response	code: 400 BAD REQUEST Content : {error: "JSON parse error"} code: 401 UNAUTHORIZED Content : {error: "Bad credentials!"} code: 404 INDIVIDUAL NOT FOUND Content : {error: "Individual not found."}
Notes	Allows the individual to get his preference about automatedSOS.

- **Track4Run Individual Controller**

Get notifications

endpoint	*/individual/{id}/track4Run/notifications
method	GET
data params	
url params	accessToken: [alphanumeric]
success response	code: 200 Content : {notifications: Array of Notification}
error response	code: 401 UNAUTHORIZED Content : {error: "Individual not logged in"} code: 404 NOT FOUND Content : {error: "Individual not found."}
Notes	Allows the individual to request for notifications, such as new runs.

Get list of runs

endpoint	*/individual/{id}/track4Run/runs
method	GET
data params	
url params	accessToken: [alphanumeric]
success response	code: 200 Content : {runs: array of Run}
error response	code: 401 UNAUTHORIZED Content : {error: "Individual not logged in"} code: 404 NOT FOUND Content : {error: "Individual not found."}
Notes	Allows the individual to get the list of all the available runs, with informations about his/her partecipations.

Request to partecipate to a run

endpoint	*/individual/{id}/track4Run/request
method	POST
url params	
data params	accessToken: [alphanumeric] runId: [integer]
success response	code: 200 Content : {message: "Request received correctly."}
error response	code: 400 BAD REQUEST Content : {error: "Malformed data parameters syntax"} code: 401 UNAUTHORIZED Content : {error: "Individual not logged in"} code: 404 NOT FOUND Content : {error: "Individual not found."}
Notes	Allows the individual to send data.

Get positions of runners

endpoint	*/individual/{id}/track4run/{run}/positions
method	GET
data params	
url params	accessToken: [alphanumeric]
success response	code: 200 Content : {notifications: Array¡Position¿}
error response	code: 401 UNAUTHORIZED Content : {error: "Individual not logged in"} code: 404 NOT FOUND Content : {error: "Individual not found" or "Run not found"}
Notes	Allows the individual to request for the positions of runners during a run in order to display them on the map.

- **Track4Run Third Party Controller**

Organise a run

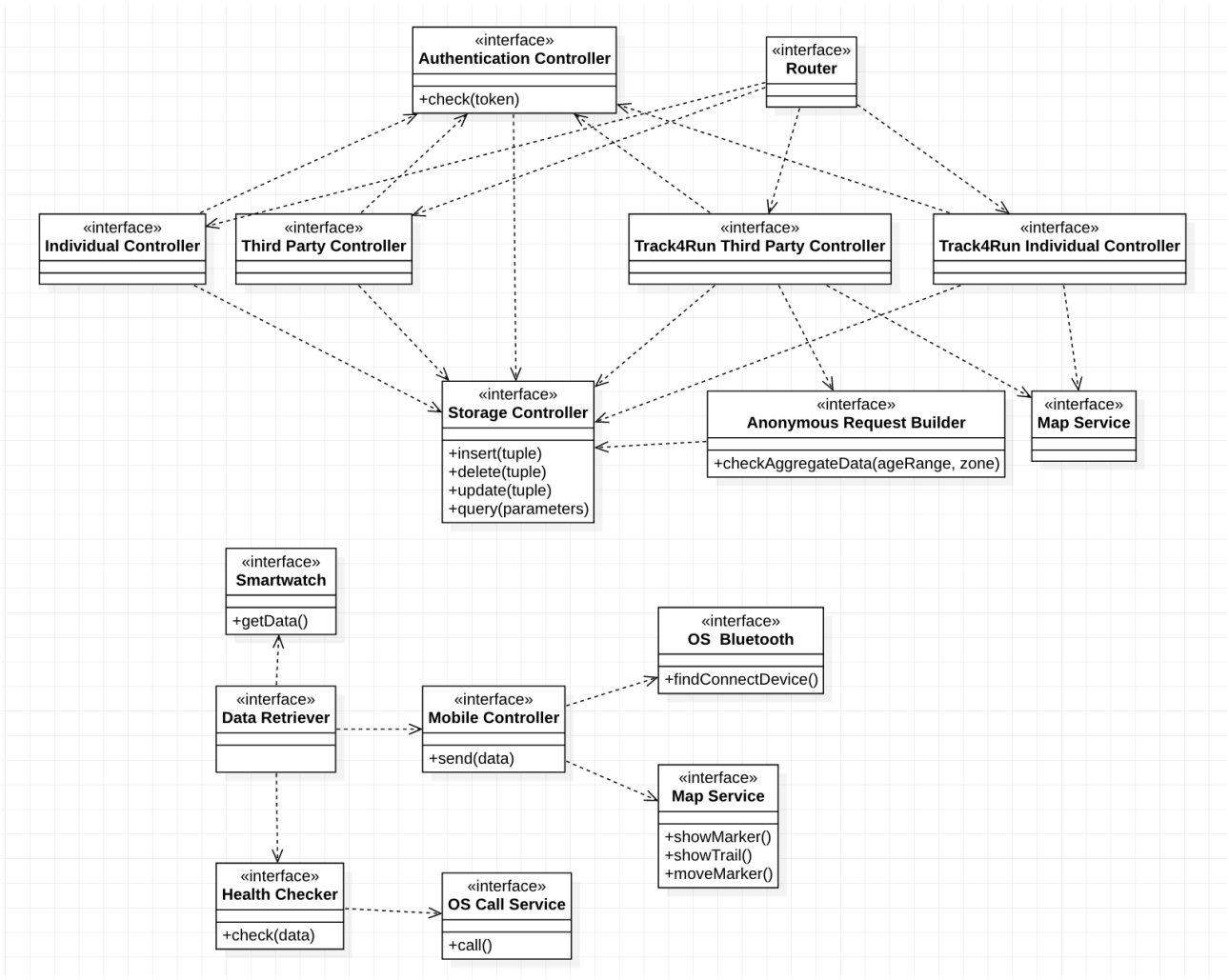
endpoint	*/thirdParty/{id}/track4Run/newRun
method	POST
url params	
data params	accessToken: [alphanumeric] runInfo: [] (implementation detail)
success response	code: 200 Content : {message: "Request received correctly."}
error response	code: 400 BAD REQUEST Content : {error: "Malformed data parameters syntax"} code: 401 UNAUTHORIZED Content : {error: "Third party not logged in"} code: 404 NOT FOUND Content : {error: "Third party not found."}
Notes	Allows the third party to organise a new run.

Get notifications

endpoint	*/thirdParty/{id}/track4Run/notifications
method	GET
data params	
url params	accessToken: [alphanumeric]
success response	code: 200 Content : {notifications: Array¡Notifications¿}
error response	code: 401 UNAUTHORIZED Content : {error: "Third party not logged in"} code: 404 NOT FOUND Content : {error: "Third part not found."}
Notes	Allows the third party to request for notifications, such as new partecipants to a run.

2.5.2. *Business Logic Interfaces*

In the following diagram, the component interfaces and the dependencies between the parts of the application are presented. Methods of interfaces are just a simplification of the real ones.



2.6. *Selected architectural styles and patterns*

2.6.1. *Multitier architecture*

TrackMe is built as a multitier architecture. More precisely it's a client/server architecture in which presentation, application processing, and data management functions are physically separated.

- Presentation tier: provides basic user interface and application access services.
- Application processing tier: contains business logic and controls application functionalities.
- Data management tier: provides the mechanism used to access and process data and holds and manages data.

This division allows each tier to be separately developed, tested, executed and reused. Other benefits of such architecture are scalability, ease of management, flexibility, and security.

- Secure: You can secure each of the three tiers separately using different methods.
- Easy to manage: You can manage each tier separately, adding or modifying each tier without affecting the other tiers.
- Scalable: If you need to add more resources, you can do it per tier, without affecting the other tiers.
- Flexible: Apart from isolated scalability, you can also expand each tier in any manner that your requirements dictate.

2.6.2. *Hybrid Client*

In our system presentation and application processing are not really totally separated, because mobile client applications have got components which work on the analysis and filtering of data. For this reason individual mobile applications are not thin clients. But they are not even considered as fat clients, because they are not thought to rely on a local store, so they are called hybrid clients. Furthermore, clients, included third parties machines, can't be considered thin clients because they have a local controller as explained in the next section.

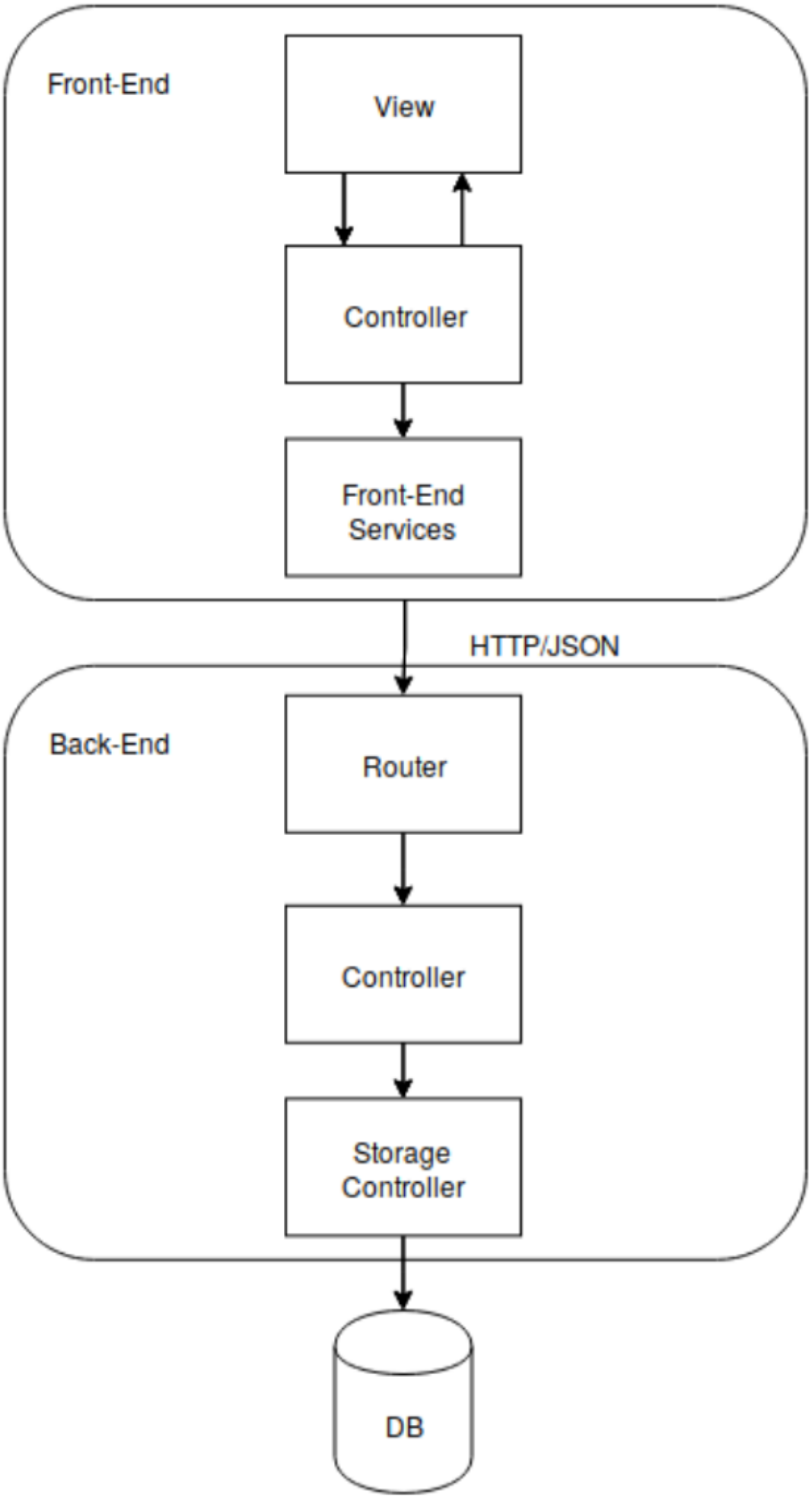
2.6.3. *MVC*

The main idea compared to other more traditional server-side architectures is to build the server as a set of stateless reusable REST services, and from an MVC perspective to take the controller out of the backend and move it into the client. The client is MVC-capable and contains all the presentation logic which is separated in a view layer, a controller layer, and a frontend services layer. After the initial application startup, only JSON data go over the wire between client and server. The frontend should only handle presentation logic, but no business logic. These are the three layers of the frontend:

- The view layer: it's intended to be displayed to the user. The presentation contains not just the content, but also the attributes for display such as HTML and CSS.
- The controller layer: it's made of functions that glue the data retrieved from the backend and the view together. The controller initialises the view model and defines how the view should react to model changes and vice-versa.
- The frontend services layer: a set of functions that allow to interact with the backend and with the controller layer.

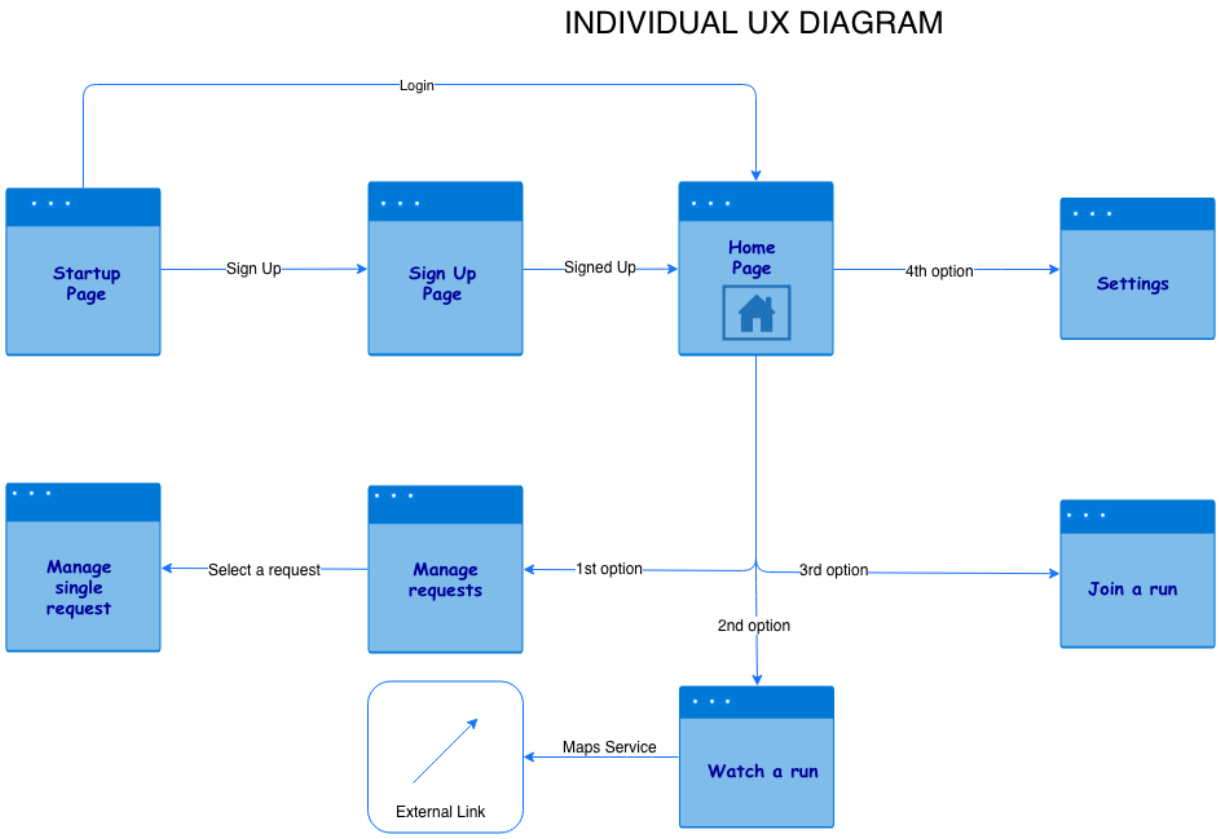
The backend is built using the usual backend layers:

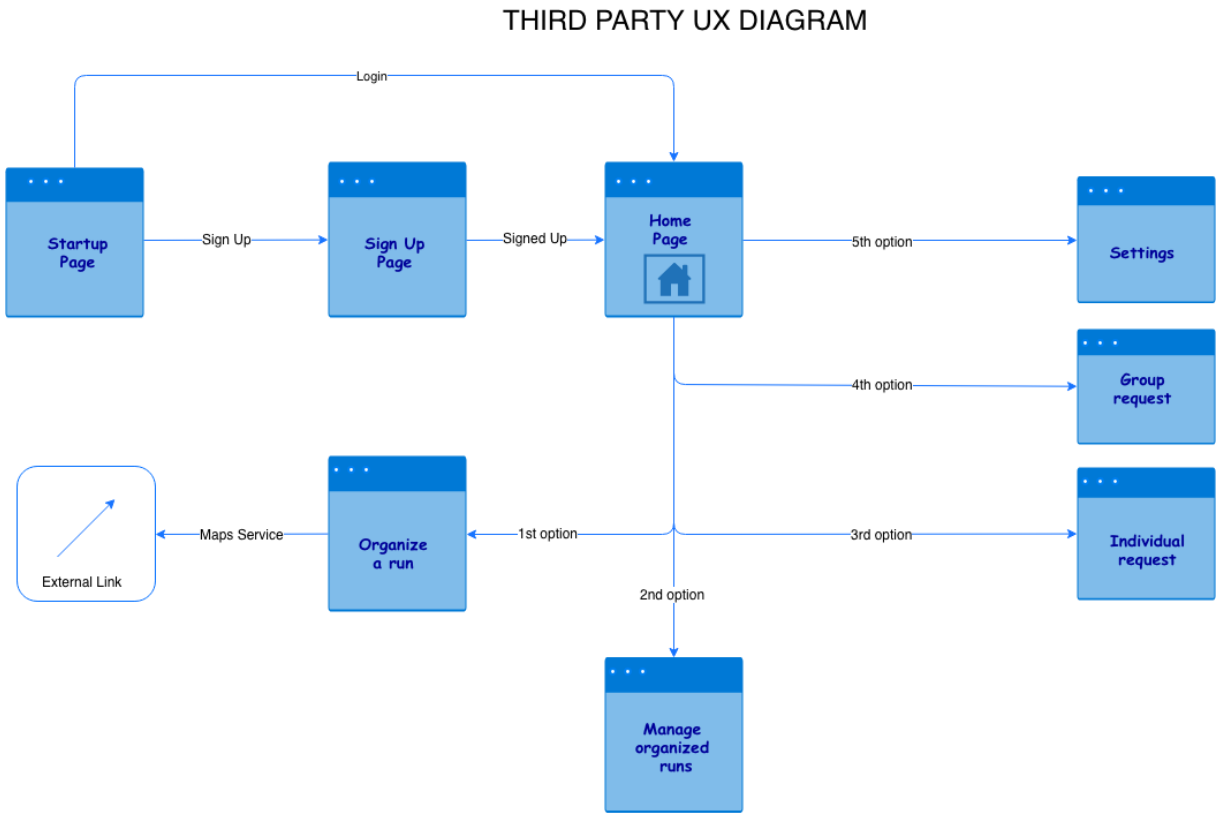
- Router layer: it defines which service entry points correspond to a given HTTP URL, and how parameters have to be read from the HTTP request.
- Controller Layer: it contains any business logic such as validations, defines the scope of business transactions. Business rules are centralized into this business logic layer that serves as an intermediary for data exchange between the front-end layer and the persistence (storage controller) layer.
- Storage controller layer: maps the database to/from in-memory domain objects.


Front-End
View
Controller
Front-End
Services
HTTP/JSON
Back-End
Router
Controller
Storage
Controller
DB

3. User interface design

3.1. UX Diagram





For the user interface design, we refer to section 3.1.1 of the RASD document. This section is aimed to enrich what was shown in the RASD through two User Experience diagrams. Their main goal is to explain as clear as possible the relationships between the various sections of the application (addressed to the Individuals) and the various pages of the website (addressed to the Third parties).
The two diagrams logically share the operations that allow the client to log into the application:

- Startup Page;
- Login;
- Sign Up.

Anyway, their presentation will be different due to the Mobile configuration for the Individuals and the Web based configuration for the Third Parties. The two diagrams start diverging from the principal window of the application/website, that is the Home Page. In the Individual UX Diagram, it is shown that through the Home Page the user can choose different options:

- Join a run, that allows the user to visualize the available races and select the one he wants to join;
- Manage requests, that allows the user to accept/refuse the pending requests coming from the Third parties or visualize/modify the already accepted requests;
- Open settings, that allows the user to modify his/her password, personal info, manage the connection between the smartphone and the smartwatch (or a similar device) or enable/disable the Automated-SOS service.

In the Third Party UX Diagram, the Home Page is essentially different from the other one and it allows the third party to choose the following options:

- Organize a run, that allows the third party to select che date, time and place of the event. Moreover, it is possible to select the race track thanks to the use of the Maps external service;
- Manage runs, whose goal is to allow a third party to visualize and check/modify the status of the runs organized by itself;
- Send an Individual request, through which the third party can directly ask a user for his health data, or subscribe to them;
- Send a Group request, through which the third party can ask the TrackMe system for aggregated data regarding more than 1000 users grouped by age and residence;
- Open settings, that allows the Third party to modify its password.

4. **Requirements traceability**

- **Third Party Controller**
  - R2) The system must be able to provide to the third party the health status of individuals;
  - R9) The third party is not allowed to access the users data until he/she accepts the request.
  - R11) The system is optimized to send the data received from the mobile application to the third parties as soon as possible.
  - R37) The system must allow the third party to make individual requests.
  - R36) The system must allow the third party to make group requests.
- **Anonymous Request Builder**
  - R4) The groups must be composed at least by 1000 individuals;
  - R5) The system must be able to provide to the third party the health status of individuals in an anonymous way;
  - R6) The system must be able to aggregate the data of the individuals, as requested by the third party;
  - R35) The system must be able to aggregate data based on the location of the individuals residence.
- **Data Retriever**
  - R3) The system must be able to retrieve data from the smartwatches and similar devices;
- **Storage Controller**
  - R27) The system must be able to store data retrieved from registered users.
  - R8) The system must save the preference of the user;
- **Individual Controller**
  - R1) The users must have given the consensus to the treatment of their information to the third party;
  - R28) The user must have an active subscription to stop it;
  - R29) The system must be able to allow the user to unsubscribe to the third party and to stop the transmission of his/her data.
  - R7) The system must be able to forward the requests from the third party to the user;
  - R26) The system must allow the user to enable/disable the AutomatedSOS service at any time.
  - R31) The system must allow the user to change his/her personal info.
- **Authentification Controller**
  - R13) The system must allow the individual to register to the application by selecting a password and providing his/her data, fiscal code included;

- R14) The system must allow the individual to log in to the application by providing the combination of a fiscal code and a password that matches an account;
- R15) Two different users cannot have the same username.
- R16) The system must allow the third party to register to the application, by specifying its VAT registration number, name and password;
- R17) The system must allow the third party to log in to the application by providing the combination of a VAT registration number and a password that match an account;
- R32) The system must allow the individual to change his/her password.
- R33) The system must allow the Third party to change its password.

- **Track4Run User Controller**
  - R19) The system must be able to retrieve the position of all the runners;
  - R20) The system must be able to provide the position of all the runners in the track in real time.
  - R21) The system must allow the user to check the list of available races at any time.
  - R22) The system must allow the user to join an available race only before its starting time.
  - R23) The user cannot join two different overlapping races.
  - R30) The system must avoid the registration of users after having reached of the maximum number of participants.

- **Track4Run Third Party controller**
  - R19) The system must be able to retrieve the position of all the runners;
  - R24) The system must allow the third party to organize a race by defining its track and its time.

- **Health Checker**
  - R18) When the health status values go below the threshold, the system must send an SOS within 5 seconds.
  - R25) The AutomatedSOS service must be enabled.

- **Mobile Controller**
  - R34) The system must allow the user to connect a smartwatch or a similar device to its smartphone.

5. Implementation, integration and test plan

In section 2.2 of this document there is the description of the components in which the whole system can be divided to split roles and functionalities and to simplify the development and the following testing. As it is evident from the Component Diagram, the components can be classified into the following subsystems:

- Mobile client, made up by:
  - Components: Mobile controller, Data retriever, Health checker, Presentation mobile;
  - Interfaces for external services: Map service interface, OS Bluetooth interface, Smartwatch interface, OS call service interface.
- Web client: browser and Map service interface.
- Business logic, made up by:
  - Components: Individual controller, Third party controller, Authentication controller, Track4Run Individual controller, Track4Run Third party controller, Anonymous request builder, Storage controller;
  - Interfaces for external services: Map service interface.
- External components: Map service, OS Bluetooth service, OS call service, Smartwatch service, DBMS.

A bottom-up strategy will be used to implement, integrate and test the system. Firstly single components have to be implemented and then unit tested. Later the following steps will be performed:

1. each subsystem will be implemented, integrated and tested;
2. the subsystems will be integrated and tested togheter.

Since we cannot directly manage external components, we assume they are reliable and we use the provided interfaces to test our system.

To keep the integration and testing process more clear we can split the Business logic subsystem into the following subsystems, based on their main functionalities:

- Individual subsystem: Individual controller and Storage controller;
- Authentication subsystem: Authentication controller and Storage controller;
- Thin third party subsystem: Third party controller and Storage controller;
- Full third party subsystem: Third party conroller, Anonymous request builder and Storage controller;
- Track4Run individual subsystem: Track4Run individual controller and Storage controller;
- Track4Run third party subsystem: Track4Run third party controller and Storage controller.

To reach the same goal, in the Mobile client subsystem we can distinguish:

- AutomatedSOS subsystem: Data retriever and Health checker;
- Transfer data subsystem: Data retriever and Mobile controller.

Focusing the attention on the Mobile client, after having tested all the subsystems (step 1), it is needed to test the integration of all of them. Let's call the result Mobile client subsystem.

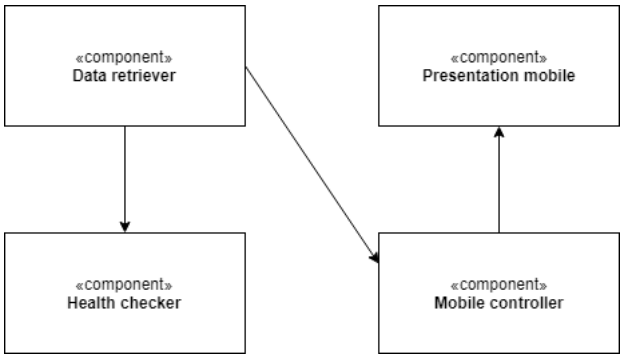


Figure 1: Mobile client subsystem

Coming to the server side, after having implemented and tested all the subsystems, it is needed to test the integration of each of them with the Mobile client subsystem and with the Web client component. More in detail:

- Mobile client subsystem and Individual subsystem;
- Mobile client subsystem and Authentication subsystem;
- Mobile client subsystem and Track4Run individual subsystem;
- Web client component and Authentication subsystem;
- Web client component and Full third party subsystem;
- Web client component and Track4Run third party subsystem.

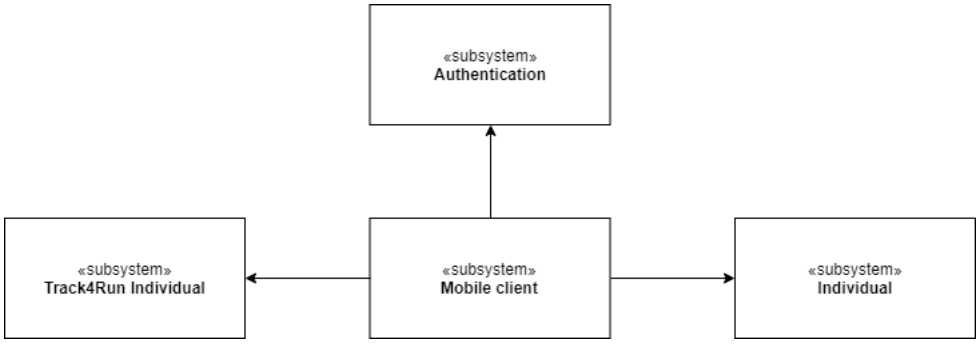


Figure 2: Mobile client tested with the other subsystems

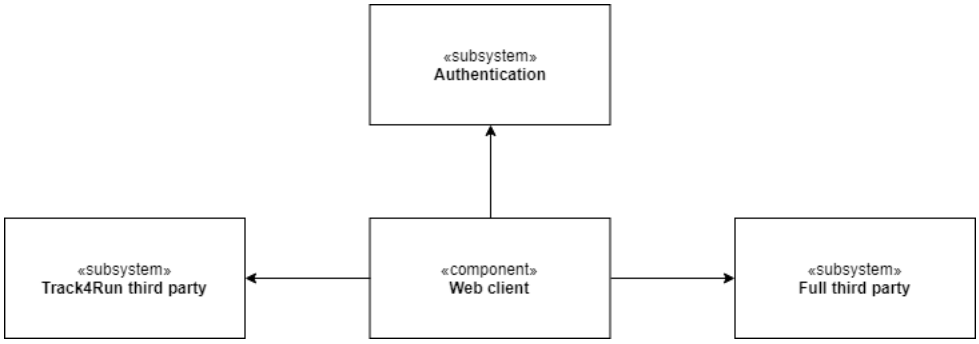


Figure 3: Web client tested with the other subsystems

After all the previous tests, a full integration test can be performed.

Figure 4: Test of the whole system

6. **Effort spent**

- **Paolo Romeo**

Description of the task	**Hours spent**
Architectural design	7
User interface design	2
Requirements traceability	2
Implementation, intergration and test plan	6
Correction of the RASD	3
Other related activities	5

- **Andrea Scotti**

Description of the task	**Hours spent**
Architectural design	8
User interface design	2
Requirements traceability	2
Implementation, intergration and test plan	6
Correction of the RASD	2
Other related activities	5

- **Francesco Staccone**

Description of the task	**Hours spent**
Architectural design	7
User interface design	4
Requirements traceability	2
Implementation, intergration and test plan	5
Correction of the RASD	3
Other related activities	4