TrackMe: Implementation document
Software Engineer 2 - 2018/2019

Riccardo Poiani, Mattia Tibaldi, Tang-Tang Zhou
Politecnico di Milano

Version 1.0
Link to source code: TODO INSERT LINK HERE
Link to what has to be installed: TODO INSERT LINK HERE

December 20, 2018

Contents

1 Introduction

The purpose of this document is to provide all the information regarding the implementation of a viable product of the TrackMe project: in particular it regards the services of Data4Help and AutomatedSOS. It follows a brief description of the structure of the document:

- First of all, in the front page it is possible to find links to the source code and to what needs to be installed
- The second section illustrates what are the requirements that have been actually implemented, providing some useful motivations in order to understand the choices that were made
- The third one takes into consideration the frameworks adopted, recapping and introducing further comments on what was already mentioned in the Design Document. Moreover, benefits and drawbacks are better analyzed, and ulterior decisions are discussed
- The fourth chapter analyzes the structure of the source code and an UML class diagram is present to illustrate a precise structure of the written code
- The fifth section provides information on how tests were written. Coverage is here presented and system tests is presented and commented
- The final chapter helps in understanding what is necessary to do in order to install and run the software, with all the necessary prerequisites

2 Requirements and functions implemented

As already mentioned, in this section it is possible to find information regarding the requirements and the functions that are actually implemented with some motivations.

2.1 Core requirements and functions

All the core requirements from R1 to R9 have been implemented, since, as the name suggests, they are fundamental.

2.2 Data4Help requirements and functions

All the requirements related to G1, G2, G8, G9, G12 and G13 have been implemented, basically because they are considered as the main features and components of Data4Help. This is true for all the mentioned goals expects for what concerns G9, that is the blocking of third party customers, that can be considered as an additional functions, but can always be useful for engaging future users. The excluded ones are the following

- G7, that is the management of elapsing requests
- G14, that is the subscription to non existing data

The motivation that stands behind this choice is basically a constraint on development time and the fact that these were considered more as a nice feature to have, and not something really critically. Indeed, it is still possible to have a good prototype of the service, even without these features. It should be noted, however, that they do not require big efforts and can be easily integrated into the project in a second time: in particular, G7 can be considered as a periodical task to be scheduled that checks and manages the time stamp of the pending requests. For what concerns G14, instead, the discussion is a bit more complex, but it basically consists in introducing a new status for the requests and a task that operates with the requests that are in such status and that performs some checks w.r.t. to the provided dates (that are the starting date and the ending date of a request: this will be clear inspecting the source code of the entities that regard the requests).
A final note on the requirements of Data4Help is the following and it regards the type of aggregated data that third parties can request. All the filters mentioned in the design document have been developed, from a server-side point of view. For clarifying this statement, that may sound obscure, consider that every available filter, except the one that regards GPS position data, is only related to some plain input that a third party customer inserts in the application and that is sent to the system. However, for filtering on GPS position a third party should define the interested region by specifying the coordinates that specify the bounds of the interested region. Since, of course, this is totally no user friendly, the application could provide a list of possible cities and region and automatically translates it. Of course, this feature could also be deployed directly on the server.
Nevertheless, up to now, what is present is the possibility of inserting group requests specifying the GPS filters, but this has not been implemented in the mobile application for the discussed reason, and, because, at this stage, it is considered sufficient to have all the other filters.

2.3 AutomatedSOS

The goals, and the related requirements, that involves AutomatedSOS are G3 and G4. However, in this case, not many of the requirements have been implemented, also due to some external limitations: indeed, considering the Android system and hardware, it is not possible to just intercept the phone call and access the microphone (for automating the phone call) and the speakers (to retrieve the response). Therefore, in order to fully develop the requirements it would be necessary to use VoIP api, that requires some sort of payments. However, some requirements such as R12, R13, R15 and R17 have been implemented anyway, with the difference that the VoIP calls are mocked with normal phone calls and no automation is present for interacting with the emergency room operator. This has been considered enough for a viable product also because it has been chosen to give more importance to the core business of the company, that is considered to be Data4Help. Another few words should be spent on R17, that is "during phone calls, the GPS is set on high precision": this is automatically performed by the Android system when calling emergency services [1].
Finally, for what concerns the part regarding the 5 seconds the following shrewdness have been adopted: the maxitum timeout for retrieving the position takes in the worst case 1 second. After that, the more critical part is parsing a

JSON that contains the emergency numbers of the various countries, but this weights only 120kB. It is thought that this should be fast enough to satisfies the requirements of the five seconds. However, the process speed of code is a problem always related with the performance of the user device and the consumed resources of other applications, therefore being 100% sure that the task is accomplished within that time is impossible. Nevertheless, some tests are perfomed and the avarage speeds satisfies this requirement.

2.4 Non functional requirements

For what concerns the non function requirements, as mentioned in the Design document, a microservices architecture has been developed. In particular, the main implemented features of the architecture are the following:

- Communication between microservices, since this is crucial in order to have a viable product. Indeed, without this, almost no requirement could be fully accomplished
- API Gateway that performs also authentication and authorization: this is also a core feature, since it is the entry point for accessing all the Rest API that the various microservices provide
- Service registry, because otherwise, one should have set up all the network communication in a more static way, and also the management of forwarding requests from the gateway would have been more complex

The load balancer has not been implemented because, among the functions mentioned above is the last one that should be considered, since it is very complex to have one if you an API gateway and a service registry is not present.

Some basic security feature have been developed also: indeed, all the passwords are stored in the database with bcrypt and the only type of communication allowed between clients and the API gateway, is HTTPS. In order to do this, a custom SSL certificate has been generated using the java keytool [2].
The authentication have been implemented by means of an UUID random token and the APIs has been secured in such a way that a certain client can access only the method that he should access: for example, a user can't access methods that regards a third party customer, but, also a user can't access API that regards another user (e.g. a user can access only his pending request, and not the pending requests of any user)
As one can claim, using UUID random tokens is for sure not the best way of achieving authentication, however, the code has been designed in such a way that is easily possible to upgrade this to the usage of JWT just implementing a single interface. This choice has been done in order to simplify a bit the security issue and to focus more on the business core of the project.

2.5 Other comments

The database regarding the collected health and position data, has not been deployed on the cloud at this early stage, because the integration was not considered necessary. Indeed, a viable prototype can be exists even without this.

Indeed, it is more something that regards the deploy, rather than the implementation itself.
The same reasoning has been applied also for the deployment of JARs in docker containers.

3 Adopted development framework

Most of the choices that regards the frameworks were already briefly introduced and motivated in the section 2.7 of the design document (that is, other design decision). However, here frameworks and other API are recapped:

- Spring boot has been adopted since it fits well for microservices and the set up of pattern components of the architecture (e.g. service registry) can be easily integrated with the usage of Spring Cloud. The main drawback of this choice is that spring boot doesn't offer much control to the developer: this of course limit the development time, but when something goes wrong, it may take time to fix the issue
- Spring security has been used to develop the authentication and the access control of the application. It is basically the de-facto standard for securing Spring-based application.
- Spring Cloud Netflix has been used to integrate and set up the API Gateway and the service registry. In particular, Zuul has been set up as the API gateway and Eureka as the service registry. These were of a great help because it is only necessary to find the right configuration settings and then everything works as expected.
- RabbitMQ were used to set up the communications among microservices. TODO INSERT BENEFITS, DRAWBACKS, COMMENTS HERE FOR TANG TANG
- Android: the mobile application has been developed for android. Here, Butter Knife has been used in order to easily bind the layout with the activities. It also enables to automatically configure listener to onClick methods. Also, room persistence library has been used to store collected data that has not been sent to the system yet, and information on the performed emergency calls TODO FURTHER COMMENTS HERE FOR MATTIA

Moreover, not already cited APIs has been adopted, and, therefore, are listed:

- Project Lombok, that is a java library that automatically, by use of annotators, creates certain code in order to reduce the amount of boilerplate code that one write
- Jayway JsonPath for manipulating JSON
- MySQL has been adopted as a DBMS
- JPA for the management of persistence and object/relation mapping

- Google guava for the usage of immutable maps
- Jackson, that is an high performance JSON processor for Java has been used, since it the default library used by Spring boot to convert object to json and viceversa. This has been very useful in the set up of the controllers: indeed, it was possible to define POJOs as controller attributes, and the conversion between HTTP requests and Java is perfectly handled. Jackson has been used also to define views in controller method: this allows to set up that in certain controller methods only certain attributes of a POJO are used. For instance one can specify that when the user is accessing its own information, his password is not sent back, but, when registering, of course the password is needed Using different views in different methods helps in achieving what mentioned
- HATEOS is used to provide hypermedia content to the clients. This helps the client mobile application in accessing the right methods and it makes the APIs restful
- GeoNames has been used in order to find the country codes, if the Android geocoder doesn't work properly
- JaCoCo has been used to generate reports on the test coverage (they will be shown in the chapter related to testing)
- JMeter has been used to test the system as a whole // TODO TANG FURTHER COMMENT ON THE USAGE. JUST A BRIEF INTRO

TODO MATTIA AND TANG INSERT FURTHER COMMENTS ON THE LIBRARY AND API YOU MAY HAVE USED IN ANDROID

4 Structure of the source code

4.1 Microservices

Here the code regarding the microservices architecture is explained.
The source code that meets the requirements mentioned above has been organized in the following way: for each microservices a project has been set up. Indeed, when dealing with this type of architecture, one should think of a microservice as a project that should me as much independent as possible from the others: this is the reason that stands behind the choice that has been made. Of course, in this way, it is possible to easily generate the single jars that will be deployed, when necessary, with, as mentioned in the design document, dockers. Therefore, the following projects are present: API gateway, service registry, group individual request service, individual request service and share data service. As one may notice, the one containing the set up of the API gateway also accesses all the information related with the accounts, and, therefore, authentication and authorization functions are coded here. In the following sections the structure of the single projects are analyzed.

4.1.1 API Gateway

In the figure 1, it is shown the root of the project structure. An analysis of the various elements follows below the figure.

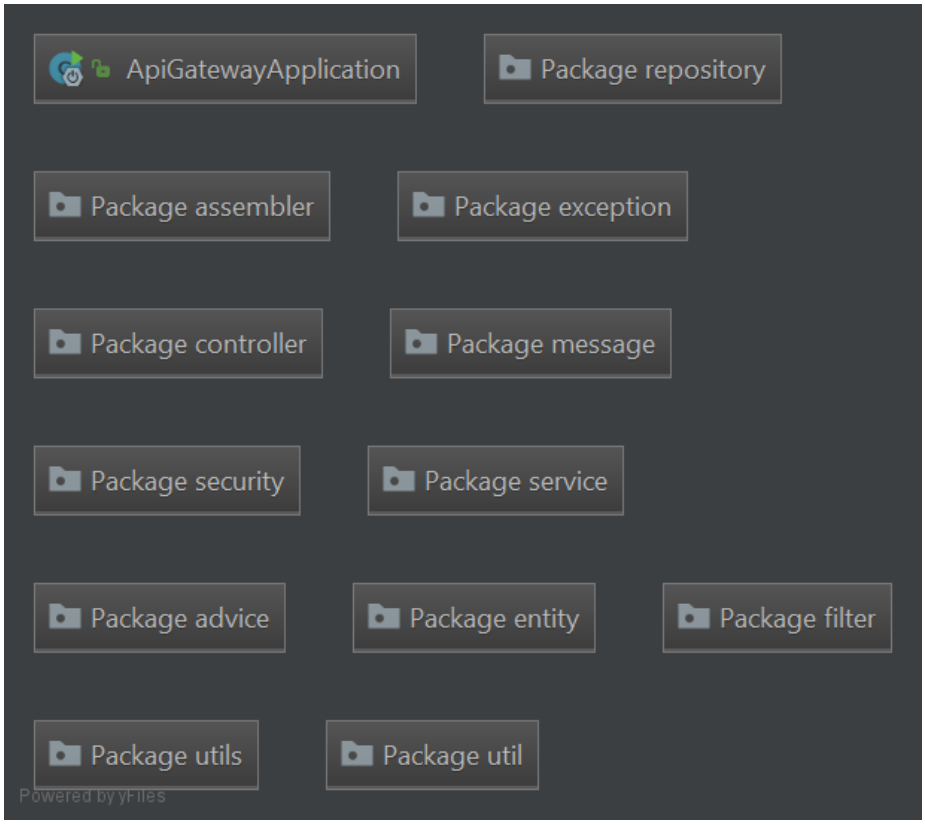

Figure 1: API gateway

- Package repository: contains the JPA repository for accessing the persistent data necessary to this service. In particular information regarding the accounts of users and third party customers are present. For what concerns the third party customers, since they can register both as private and as related with a company the following repositories are present: company details, private details, third party customers and users. An additional repository is present, and it accesses information regarding the API. Indeed, all the accessible APIs that are available are stored in a database, in order to provide access control. For better specifying this choice, one may consider the fact that it is not necessary to search on the service registry non-existing API or to forward requests that can be already classified as rejected (e.g. user A that is trying to access an API available only for third party customers)
- Package assembler: this class contains components useful to build HATEOAS resources of entities that are returned to clients, adding hypermedia contents

- Package exception: this contains the custom exceptions defined during the development
- Package controller: contains the controller that defines the APIs that regards the management of the users and third party customer accounts. From here, it is possible to access the business functions that regards the account (e.g. login, logout, registration). Inside here, controller are split into secure and public controllers: the public one are accessible to everyone, while for accessing the secured ones, it is necessary to perform the log in
- Package message: it provides the functions of communicating with other microservices, by means of RabbitMQ. In particular, three subpackages are present here. The package publisher contains classes that helps in publishing the events of creation of new accounts to other other services. The package protocol defines a way of communicating the interested pieces of information, and, finally the configuration package specifies the configuration settings needed to convert object from JSON (and viceversa) during the exchange of data, and also the Rabbit queues
- Package security: contains the main features that have been introduced above regarding the security
- Package service: contains the services that implements the business functions of the account service, with all the APIs that it exposes. The interfaces present here map the component interfaces of the account service
- Package advice: this encompasses the handling of server side exceptions in order to show to clients useful messages without exposing the structure of the project and low level errors
- Package entity: here classes that are mapped to the databases are present
- Package filter: it includes filters that the API gateway applies during the management of "external" requests (i.e. requests that need to be processed by other services). In particular, it is further subdivided into three packages: pre, route and post. In this case, the pre filter that is present perform access control on the external API that is being requested, the route filter is a translation filter that adds header in order to identify the clients in the other microservices, and the post filter fixes the hypermedia content that is sent as a response to the original request
- Package util: various utility needed in the development

The routes to the other microservices available are present in the application.properties file.

4.1.2 Service registry

The project of the service registry is almost empty, no package is present, but just an application class is present, annotated with @EnableEurekaServer. The configuration of the service registry is set in the application.properties file.

4.1.3 Individual request service

Here it follows the package diagram of the individual request service project 2. As one may notice, the structure is very similar to the one explained in the api gateway project. Of course, here, the controllers provide access to the APIs that regards the individual request service. The business functions of this project are present in the service package, and the interfaces that are present, are mapped with the component interfaces of the Design document.
Another difference w.r.t. the API gateway that is worth to point out is that here the package message contains also a subpackages that defines listeners: these are charged of listening to new events that are forwarded from RabbitMQ.

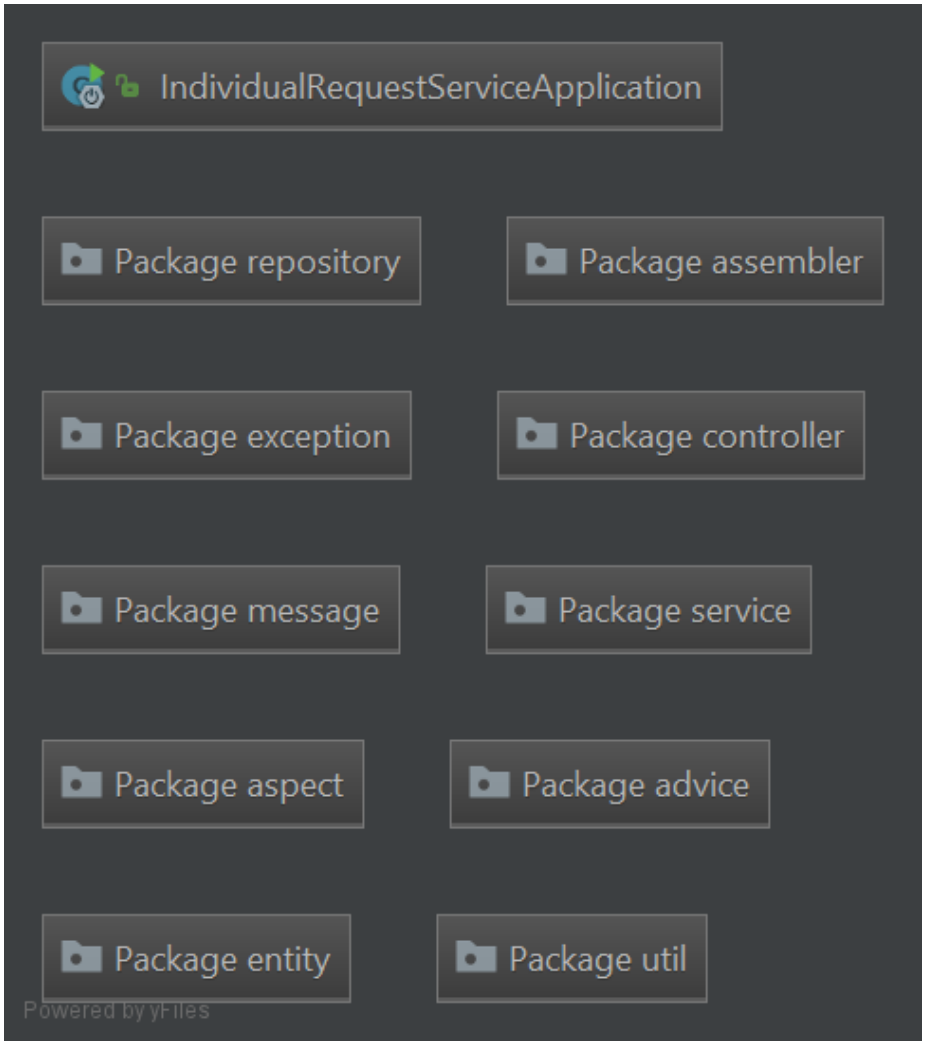

Figure 2: Individual request service

4.1.4 Group request service

In the next figure, the package diagram of the group request project is shown 3.
The same comments hold here except for the fact that the business function mapped in the component interfaces are the one regarding the group request service.

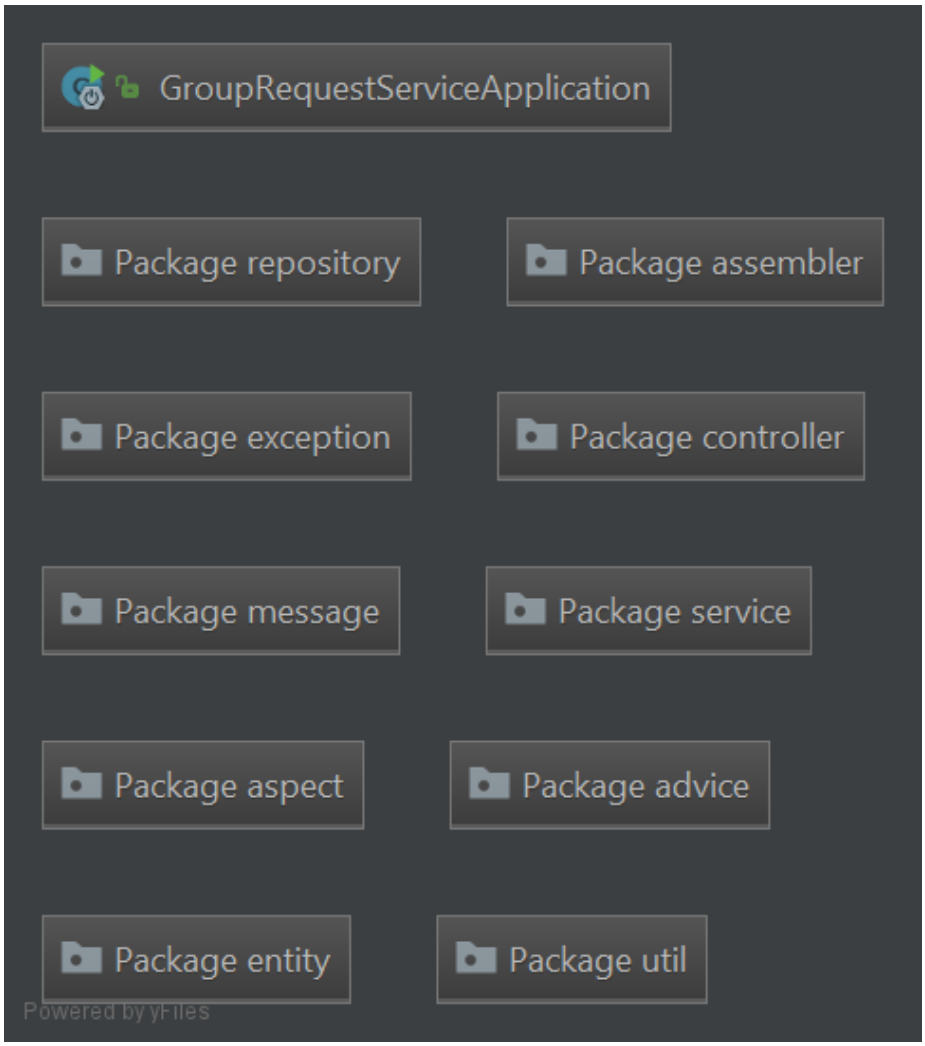

Figure 3: Group request service

4.1.5 Share data service

The structure of the share data service, as shown by the following picture, is also very similar to the previous ones 4.

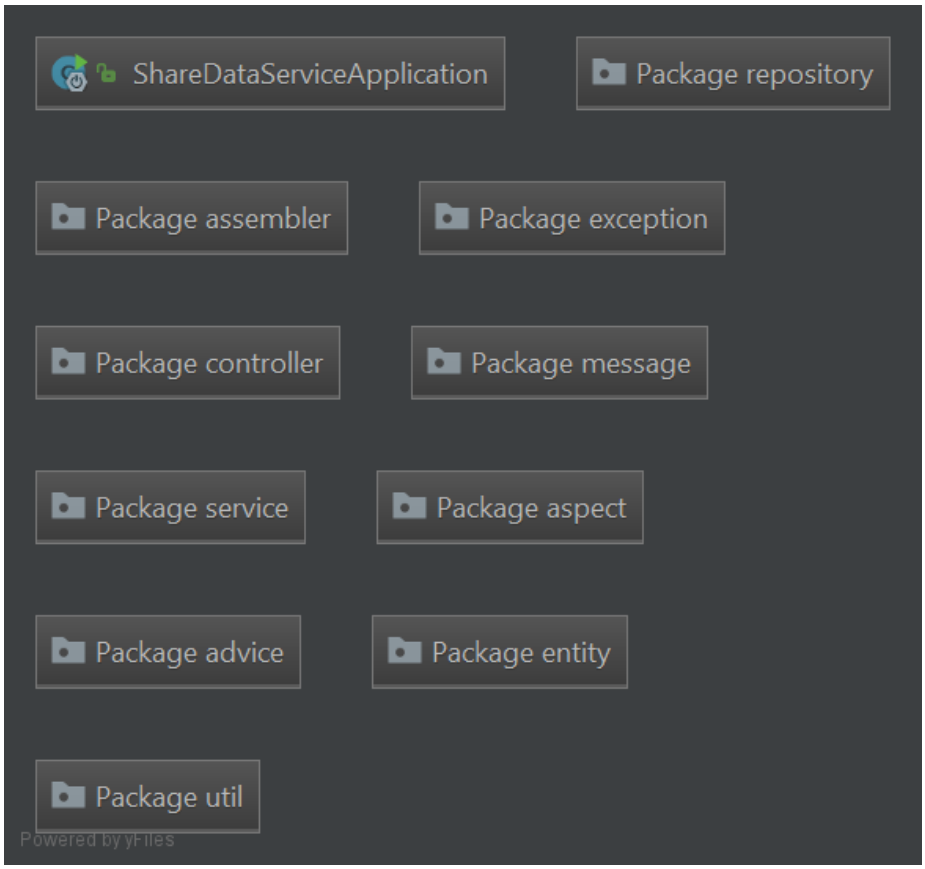

Figure 4: Share data service

However, it is important to point out the structure of the code that manages the personalized query specified by the group requests on anonymized data. // TODO TANG

4.2 Mobile code

4.2.1 Data4Help

TODO MATTIA

4.2.2 AutomatedSOS

TODO TANG TANG

5 Testing

5.1 Microservices

5.1.1 Tests of a single microservices

For managing the tests, in general, during the development, once a feature was ready it was tested. The procedure described in the Design document was followed; however, due to the fact that many spring components and classes do already many things, unit tests of single class sometimes didn't make much sense, because it would have been necessary to mock many Spring features, that are supposed to be already tested and working properly.
For the projects, the process of testing usually followed this life cycle: repositories have been tested firsts, then the services, that cover the business functions of the project under consideration, and then the controllers, for which two kinds of tests were performed: a unit and an integration test. These integration tests involves all the stacks that were developed in the projects: indeed, from here, the connection to the APIs with HTTP/ HTTPS requests are tested: this methods usually call the business functions, that modifies the databases by means of repositories. If error occurs, the classes in the advice packages come into help, and, therefore, they are tested as well.
For what concerns the tests that regards interfaces that allows to share events with other microservices, as soon as it was possible to have them working they were tested as well. However, it is worth to note that the real integration between microservices have been tested with JMeter. This is because TODO TANG.
The same things holds for the test of the service registry and for the forwarding of requests to other microservices by means of the Zuul gateway.
The approach taken for all the tests mentioned above (the ones with JMeter excluded), is white box testing. Here it follows a brief report on the coverage, with some comments. The images are taken from the report generated by JaCoCo.

First of all the coverage of the API gateway is shown.

api-gateway

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.poianitibaldizhou.trackme.apigateway.service		96%		94%	4	53	0	130	2	34	0	4
com.poianitibaldizhou.trackme.apigateway.util		89%		100%	3	9	4	23	3	7	2	5
com.poianitibaldizhou.trackme.apigateway.controller		98%		75%	1	17	1	52	0	15	0	4
com.poianitibaldizhou.trackme.apigateway.security		100%		n/a	0	21	0	60	0	21	0	5
com.poianitibaldizhou.trackme.apigateway.message.publisher		100%		n/a	0	7	0	38	0	7	0	2
com.poianitibaldizhou.trackme.apigateway.advice		100%		n/a	0	14	0	40	0	14	0	7
com.poianitibaldizhou.trackme.apigateway.message.protocol		100%		n/a	0	3	0	6	0	3	0	2
com.poianitibaldizhou.trackme.apigateway.message.configuration		100%		n/a	0	17	0	23	0	17	0	3
com.poianitibaldizhou.trackme.apigateway.assembler		100%		n/a	0	6	0	11	0	6	0	3
com.poianitibaldizhou.trackme.apigateway.exception		100%		n/a	0	5	0	10	0	5	0	5
Total	37 of 1,730	97%	3 of 46	93%	8	152	5	393	5	129	2	40

Figure 5: API gateway coverage

As one may notice, the entity and the filter packages are not present. Indeed, filters have been tested with JMeter, and therefore are not present here. Furthermore, entity classes are basically only attributes and methods that are not shown since they are auto generated with lombok.
Finally, all the other lombok auto generated methods have been excluded from the coverage.

Similar reasoning applies also to the coverage of the individual request service.

individual-request-service

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.polanitibaldizhou.trackme.individualrequestservice.message.protocol		65%		n/a	1	4	2	8	1	4	1	3
com.polanitibaldizhou.trackme.individualrequestservice.service		97%		97%	3	47	0	106	2	26	0	3
com.polanitibaldizhou.trackme.individualrequestservice.advice		93%		n/a	1	24	4	70	1	24	0	12
com.polanitibaldizhou.trackme.individualrequestservice.controller		100%		91%	2	20	0	52	0	8	0	2
com.polanitibaldizhou.trackme.individualrequestservice.assembler		100%		n/a	0	6	0	32	0	6	0	3
com.polanitibaldizhou.trackme.individualrequestservice.util		100%		n/a	0	6	0	32	0	6	0	5
com.polanitibaldizhou.trackme.individualrequestservice.message.configuration		100%		n/a	0	22	0	26	0	22	0	4
com.polanitibaldizhou.trackme.individualrequestservice.exception		100%		n/a	0	10	0	20	0	10	0	10
com.polanitibaldizhou.trackme.individualrequestservice.message.publisher		100%		n/a	0	4	0	24	0	4	0	1
com.polanitibaldizhou.trackme.individualrequestservice.message.listener		100%		100%	0	10	0	22	0	6	0	2
com.polanitibaldizhou.trackme.individualrequestservice.aspect		100%		n/a	0	3	0	10	0	3	0	1
com.polanitibaldizhou.trackme.individualrequestservice.message.protocol.enumerator		100%		n/a	0	1	0	2	0	1	0	1
Total	74 of 1.872	96%	3 of 74	95%	7	157	6	404	4	120	1	47

Figure 6: Individual request service coverage

Same things hold also for the group request service.

individual-request-service

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.polanitibaldizhou.trackme.individualrequestservice.message.protocol		65%		n/a	1	4	2	8	1	4	1	3
com.polanitibaldizhou.trackme.individualrequestservice.service		97%		97%	3	47	0	106	2	26	0	3
com.polanitibaldizhou.trackme.individualrequestservice.advice		93%		n/a	1	24	4	70	1	24	0	12
com.polanitibaldizhou.trackme.individualrequestservice.controller		100%		91%	2	20	0	52	0	8	0	2
com.polanitibaldizhou.trackme.individualrequestservice.assembler		100%		n/a	0	6	0	32	0	6	0	3
com.polanitibaldizhou.trackme.individualrequestservice.util		100%		n/a	0	6	0	32	0	6	0	5
com.polanitibaldizhou.trackme.individualrequestservice.message.configuration		100%		n/a	0	22	0	26	0	22	0	4
com.polanitibaldizhou.trackme.individualrequestservice.exception		100%		n/a	0	10	0	20	0	10	0	10
com.polanitibaldizhou.trackme.individualrequestservice.message.publisher		100%		n/a	0	4	0	24	0	4	0	1
com.polanitibaldizhou.trackme.individualrequestservice.message.listener		100%		100%	0	10	0	22	0	6	0	2
com.polanitibaldizhou.trackme.individualrequestservice.aspect		100%		n/a	0	3	0	10	0	3	0	1
com.polanitibaldizhou.trackme.individualrequestservice.message.protocol.enumerator		100%		n/a	0	1	0	2	0	1	0	1
Total	74 of 1.872	96%	3 of 74	95%	7	157	6	404	4	120	1	47

Figure 7: Group request individual service coverage

For what concerns the share data service, here follows the image of the coverage. The same comments are valid also in this situation.

share-data-service

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.polanitibaldizhou.trackme.sharedataservice.util		99%		100%	2	31	2	124	2	29	2	14
com.polanitibaldizhou.trackme.sharedataservice.service		98%		100%	1	32	0	124	1	31	0	3
com.polanitibaldizhou.trackme.sharedataservice.message.protocol.enumerator		100%		n/a	0	6	0	13	0	6	0	6
com.polanitibaldizhou.trackme.sharedataservice.repository		100%		n/a	0	7	0	51	0	7	0	1
com.polanitibaldizhou.trackme.sharedataservice.controller		100%		100%	0	14	0	40	0	8	0	2
com.polanitibaldizhou.trackme.sharedataservice.message.listener		100%		100%	0	17	0	49	0	10	0	3
com.polanitibaldizhou.trackme.sharedataservice.message.protocol		100%		75%	1	7	0	11	0	5	0	5
com.polanitibaldizhou.trackme.sharedataservice.message.configuration		100%		n/a	0	27	0	34	0	27	0	5
com.polanitibaldizhou.trackme.sharedataservice.assembler		100%		n/a	0	8	0	17	0	8	0	4
com.polanitibaldizhou.trackme.sharedataservice.advice		100%		n/a	0	8	0	24	0	8	0	4
com.polanitibaldizhou.trackme.sharedataservice.aspect		100%		n/a	0	3	0	10	0	3	0	1
com.polanitibaldizhou.trackme.sharedataservice.exception		100%		n/a	0	4	0	11	0	4	0	4
com.polanitibaldizhou.trackme.sharedataservice.message.publisher		100%		n/a	0	3	0	11	0	3	0	1
Total	12 of 3.040	99%	1 of 36	97%	4	167	2	519	3	149	2	53

Figure 8: Share data service coverage

5.1.2 JMeter

// TODO TANG TANG

5.2 Android application

// TODO TANG TANG AND MATTIA

6 Installation instruction

6.1 Microservices

First of all, let's check all the prerequisites needed to install and run the software:

- RabbitMQ is necessary, since a RabbitMQ server is needed to make the communication between microservices properly working. It is possible to download it from there: `https://www.rabbitmq.com/download.html`.
  Once the page is opened, click on the links on the right, selecting the right operative system and download the installer. When working with windows, it may be possible that the installation process throws an error that expresses the necessity of downloading the Erlang programming language. In this case, just follow `http://www.erlang.org/downloads` and download Erlang for the version of Windows that you are using

- MySQL is another critical component since it has been used as a DBMS for managing persistent data. For having this working, it is possible to follow the guide at : `https://beep.metid.polimi.it/documents/121843524/b5d81926-98f6-496f-bf45-a2a8e897228c`.
  Once that the server has been set up, it is necessary to create the databases and its admin: to do that, first of all, go to the MySQL folder and login with root. After that, write "create user 'trackmeadmin'@'%' identified by 'datecallmeeting95';".
  Now, the admin for the database has been created, and it is necessary to create all the databases used by the various microservices.
  In order to accomplish this, run "create database share_data_db;", "create database group_request_db;", "create database individual_request_db", "create database account_service_db;".
  Finally, for each database created, it is necessary to run this command: "grant all on db_name.* to 'trackmeadmin'@'%';" where db_name has to be substituted with share_data_db, group_request_db, account_service_db and individual_request_db.

6.2 Android application

TODO TANG E MATTIA

References

[1] Android emergency location service, URL https://crisisresponse.google/emergencylocationservice/how-it-works/

[2] Working with certificates and SSL, URL https://docs.oracle.com/cd/E19830-01/819-4712/ablqw/index.html