Department of Computer Science and Engineering

Implementation & Test Deliverable Document (ITD)

TrackMe

- v1.0 -

Authors:

Avila, Diego 903988

Schiatti, Laura 904738

Virdi, Sukhpreet 904204

Source code repository

https://github.com/lauricdd/AvilaSchiattiVirdi/tree/backend/src

January 13th , 2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Purpose

This document describes in detail the processes of implementation and testing of D4H and ASOS following the specifications reported in RASD and DD documents. The purpose of this document is to provide a general overview of a running prototype implementation of TrackMe and the process of testing it.

1.2 Scope

This document represents the Integration Testing Plan Document for TrackMe Service.

Integration testing is a key activity to guarantee that all the different subsystems composing Data4Help and AutomatedSOS interoperate consistently with the requirements they are supposed to fulfil and without exhibiting unexpected behaviours. The purpose of this document is to outline, in a clear and comprehensive way, the main aspects concerning the organization of the integration testing activity for all the components that make up the system.
More precisely, the document presents:

- A list of the subsystems and their subcomponents involved in the integration activity that will have to be tested
- The criteria that must be met by the project status before integration testing of the outlined elements may begin
- A description of the integration testing approach and the rationale behind it
- The sequence in which components and subsystems will be integrated
- A description of the planned testing activities for each integration step, including their input data and the expected output
- Some performance measures that should be performed on the components to check they are fulfilling the requirements

- A list of all the tools that will have to be employed during the testing activities, together with a description of the operational environment in which the tests will be executed

.

Chapter 2

Implemented Requirements

This section describes which functional requirements were implemented in the presented TrackMe prototype which compresses D4H and ASOS systems in reference to the ones already stated in the RASD.

2.1 Data4Help

2.1.1 Data4Help Requirements

Since, regarding the implementation of Data4Help system, the requirements addressed are:

[R1] The system must allow an individual to register a new account
D4H::SignupService

[R2] The system must allow an individual to access to their account
D4H::LoginService

[R3] The system must allow an individual to accept or reject their requests of accessing personal data
D4H::D4HRequestService

[R4] The system must be able to communicate with TrackMe database in order to obtain the health status and location of an individual
D4H::SearchService

[R5] The system must allow a third party company to register a new account
D4H::SignupService

[R6] The system must allow a third party company to access to its account
D4H::LoginService

[R7] The system must be able to notify the individual that a third party company wants to access its data
D4H::D4HRequestService

[R8] The system must allow a third party company to search for an individual health status and location using his/her SSN
D4H::SearchService

[R9] The system must allow a third party company to filter data of an anonymized group of individuals by country, age, gender and blood type parameters
D4H::SearchService

[R10] The system must be able to anonymize the data of a group of individuals
D4H::APIManager

[R11] The system must allow a third party company to subscribe to an individual health status and location
D4H::SubscriptionService

[R12] The system must allow a third party company to subscribe to data of an anonymized group of individuals
D4H::SubscriptionService

2.2 AutomatedSOS

2.2.1 AutomatedSOS Requirements

With respecto to the implementation of AutomatedSOS system, the requirements that are addressed are:

[R13] The system must be able to send a request for monitoring an individual's data when he/she is older than 60 years old
ASOS::DataService

[R14] The system must be able to monitor, and compare against defined thresholds, the health status of an individual
ASOS::DataService

[R15] The system must be able to contact the health-care service associated to an individual
ASOS::DataService

2.3 Risks and Contingencies

This subsection of the Implementation Plan identifies the risks and specific actions to be taken in the event the implementation fails or needs to be altered at any point and includes the factors to be used for making the decision. The possible risk that we are addressing in our system at this point of time are as follows:

- System fails to connect to the Server/Database, and in case of out-of-range health parameters, action cannot be taken within time.
- In these kind of scenarios, as soon as the system is back on-line, latest parameters are checked and further transmitted.

2.4 Acceptance Criteria

This section of the Implementation Plan establishes the exit or acceptance criteria for transitioning the system into production. Identify the criteria that will be used to determine the acceptability of the deliverables as well as any required technical processes, methods, tools, and/ or performance benchmarks required for product acceptance.

When the system passes all the test cases and end-2-end scenarios (illustrated in chapter-6 in detail) without fail and the system is 100% on-line with all the requirements stated about are implemented and working properly and making the system stable. Based on which a go/no-go decision can be taken, firstly a beta-version of the product can be published.

Chapter 3

Adopted development frameworks

In the following chapter a list of development frameworks and technologies that are used in building D4H and ASOS systems, they were designed and implemented to be highly decoupled.

3.1 Adopted Programming Languages

The Programming languages used are:

3.1.1 Java

Java is a general-purpose computer-programming language that is concurrent, class-based, object-oriented,and specifically designed to have as few implementation dependencies as possible. This language was used for the development of D4H and ASOS backends.

Advantages

- Object-oriented: allows you to create modular programs and reusable code.
- Platform-Independent: ability to move easily from one computer system to another.
- Distributed: designed to make distributed computing easy with the networking capability that is inherently integrated into it.
- Secure: the Java language, compiler, interpreter, and runtime environment were each developed with security in mind.
- Allocation: java has the feature of Stack allocation system. It helps the data to be stored and can be restored easily.
- Multithreaded: the capability for a program to perform several tasks simultaneously within a program.

Disadvantages

- Performance: significantly slower and more memory-consuming than natively compiled languages such as C or C++.
- Look and Feel: the default look and feel of GUI applications written in Java using the Swing toolkit is very different from native applications.
- Single paradigm language: the addition of static imports in Java 5.0 the procedural paradigm is better accommodated than in earlier versions of Java.

3.1.2 Typescript

TypeScript is an open-source programming language, it is a strict syntactical superset of JavaScript, and adds optional static typing to the language. TypeScript is designed for development of large applications and transcompiles to JavaScript. As TypeScript is a superset of JavaScript, existing JavaScript programs are also valid TypeScript programs. TypeScript may be used to develop JavaScript applications for both client-side and server-side (Node.js) execution.

Advantages

- Object Oriented Programming Features
- TypeScript Does Not Need a Runtime Plugin
- Back-end Developer Feel More Comfortable With it
- It is Used in Popular Frameworks

Disadvantages

- Learning Curve
- Needs Development Tooling
- Cannot Easily be Edited by Content Management Systems (CMS)
- Short Update Cycles

3.1.3 PHP

PHP: Hypertext Preprocessor is a server-side scripting language designed for Web development. **Why we choose to use this?** PHP was used for creating the script that generates data for individuals that are initially in the database.

Advantages

- Rapid Development
- Easier Maintenance

Disadvantages

- Slower Execution
- Limited Visibility and Control

3.2 Adopted Middle-wares

To begin with, it is suggested to build Java based applications, since developers have expertise working with this language. Moreover, the proposed frameworks are:

3.2.1 Java Spark Framework

Java Spark Framework is a simple and expressive Java/Kotlin web framework DSL built for rapid development. Sparks intention is to provide an alternative for Kotlin/Java developers that want to develop their web applications as expressive as possible and with minimal boilerplate. With a clear philosophy Spark is designed not only to make you more productive, but also to make your code better under the influence of Spark's sleek, declarative and expressive syntax. It is a micro-framework for MVC applications, because pure Java web development has traditionally been very cumbersome. The configuration for this framework is minimal and with a short learning curve.

So why choose Spark over other frameworks? Answer is simple. The best part about Spark is that it has a very consistent API which consists of just calling static methods from within your code. Indeed, it is a framework, in the sense that you specify the code to run and it wraps it in its own functionality, but it really feels more like a library with a touch of magic happening behind the scenes. You control what route mappings you wish to declare, what code is the responsible for handling the requests and how you like to handle everything else. So far, Spark seems excellent for tiny applications or API backends.

Advantages

- Fast and lightweight
- Excellent for rapid prototyping
- Easy to setup
- Most commonly used with AngularJS
- Real micro-framework
- Uses Jetty
- Can be used inside a container or without one

Disadvantages

- Documentation could be better, it is not intended for beginners.
- Not suitable for large projects
- Small community

3.2.2 Morphia Framework

MongoDB is an open source document-oriented NoSQL database system which stores data as JSON-like documents with dynamic schemas. As it doesn't store data in tables as is done in the usual relational database setup, it doesn't map well to the JPA way of storing data. Morphia is an open source lightweight type-safe library designed to bridge the gap between the MongoDB Java driver and domain objects. It can be an alternative to SpringData if you're not using the Spring Framework to interact with MongoDB.

Advantages

- Stable
- Very good play integration
- Offers access to all Mongo driver features

- Provides more extensiblity leading you to enjoy the ability to suck most out of mongo features
- Lightweight

Disadvantages

- Does not have serializers for enumeration type

3.2.3 Lettuce Framework

Lettuce is a fully non-blocking Redis client built with netty providing Reactive, Asynchronous and Synchronous Data Access. Lettuce connects with all operational models natively supported by Redis. Partition-tolerance, Read Slaves and Transport-Level-Security provide the required foundation for highly scalable applications.

Advantages

- Standalone
- Providing synchronous, asynchronous and reactive APIs
- Partition-tolerance
- Dynamic API
- Cloud ready

3.2.4 Angular Framework

It is a TypeScript-based open-source front-end web application framework led by the Angular Team at Google and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS.

So why choose Angular over other frameworks? Answer is simple. Angular implements MVC architecture to develop web application. It provides a platform where only few attributes needs to be added to the HTML language and progress with the development will be faster. It enhances HTML which makes all difficult tasks move smoothly. Using MVC architecture helps in retrieving and inserting the data, independent of user interface. Angular makes it easy to arrange things like dynamic loading and dependencies and utilize them as required without worrying about instances and namespaces. Angular development is Robust and easy. It is has features like filters, data binding, scope management, directives, API client, form validation. These features make the web applications more straightforward and easy to detect and troubleshoot them.

Advantages

- Two way data binding
- Directives
- Dependency injection
- Community

Disadvantages

- Performance
- Steep learning curve

3.3 Additional APIs

The only external APIs to whom TrackMe environment depends on, are the health-care services to which ASOS should contact in some cases. Both D4H and T4R do not depend on any external services.

3.3.1 Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

Advantages

- Containers are small compared to VMs
- Containers uses less resources
- Fast boot
- Eliminating the “Works on My Machine” situation

Disadvantages

- Security
- Isolation
- Networking

3.3.2 MongoDB

MongoDB is a cross-platform document-oriented database program. It is issued under the Server Side Public License (SSPL) version 1, which was submitted for certification to the Open Source Initiative but later withdrawn in lieu of SSPL version 2[6]. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemata.

Advantages

- schema-less. If you have a flexible schema, this is ideal for a document store like MongoDB. This is difficult to implement in a performant manner in RDBMS
- ease of scale-out. Scale reads by using replica sets. Scale writes by using sharding (auto balancing). Just fire up another machine and away you go. Adding more machines = adding more RAM over which to distribute your working set.
- cost. Depends on which RDBMS of course, but MongoDB is free and can run on Linux, ideal for running on cheaper commodity kit.
- you can choose what level of consistency you want depending on the value of the data (e.g. faster performance = fire and forget inserts to MongoDB, slower performance = wait til insert has been replicated to multiple nodes before returning)

Disadvantages

- Data size in MongoDB is typically higher due to e.g. each document has field names stored it
- less flexibity with querying (e.g. no JOINs)
- no support for transactions - certain atomic operations are supported, at a single document level
- at the moment Map/Reduce (e.g. to do aggregations/data analysis) is OK, but not blisteringly fast. So if that's required, something like Hadoop may need to be added into the mix

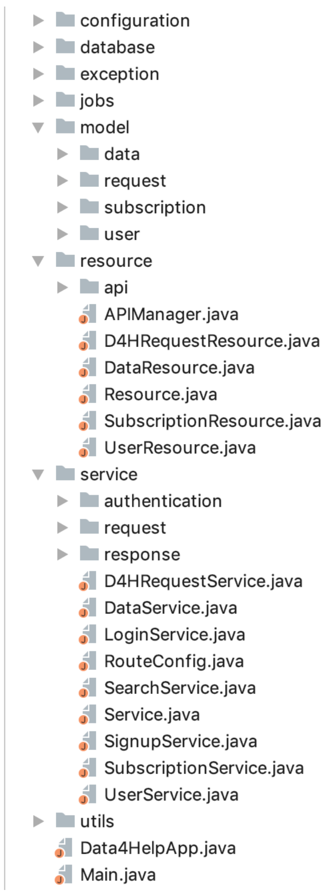
- less up to date information available/fast evolving product

Chapter 4

Structure of the source code

4.1 D4H Backend (Spark project)

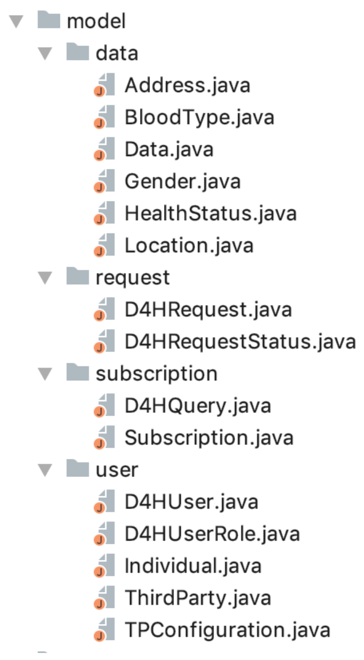
The general structure of the project is contained in src/main/java folder, it can be seen in Figure 4.10 As in any spark project there are several folders concerning general configurations of the project and also others containing the source code, in this opportunity four main ones can be highlighted: model, resource, service and jobs.



```
▶ configuration
▶ database
▶ exception
▶ jobs
▼ model
    ▶ data
    ▶ request
    ▶ subscription
    ▶ user
▼ resource
    ▶ api
      APIManager.java
      D4HRequestResource.java
      DataResource.java
      Resource.java
      SubscriptionResource.java
      UserResource.java
▼ service
    ▶ authentication
    ▶ request
    ▶ response
      D4HRequestService.java
      DataService.java
      LoginService.java
      RouteConfig.java
      SearchService.java
      Service.java
      SignupService.java
      SubscriptionService.java
      UserService.java
▶ utils
  Data4HelpApp.java
  Main.java
```

Figure 4.1: D4H Source code - General Structure

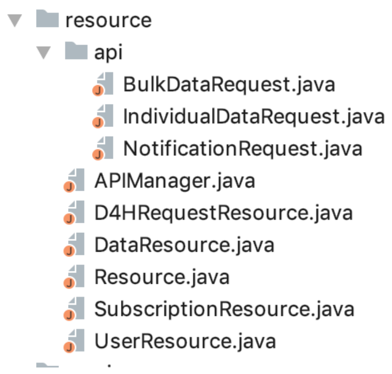
- **model**: containes the classes define all data structures, such as *Individual, ThirdParty*, etc.



```
▼ model
  ▼ data
      Address.java
      BloodType.java
      Data.java
      Gender.java
      HealthStatus.java
      Location.java
  ▼ request
      D4HRequest.java
      D4HRequestStatus.java
  ▼ subscription
      D4HQuery.java
      Subscription.java
  ▼ user
      D4HUser.java
      D4HUserRole.java
      Individual.java
      ThirdParty.java
      TPConfiguration.java
```

Figure 4.2: D4H Source code - Models

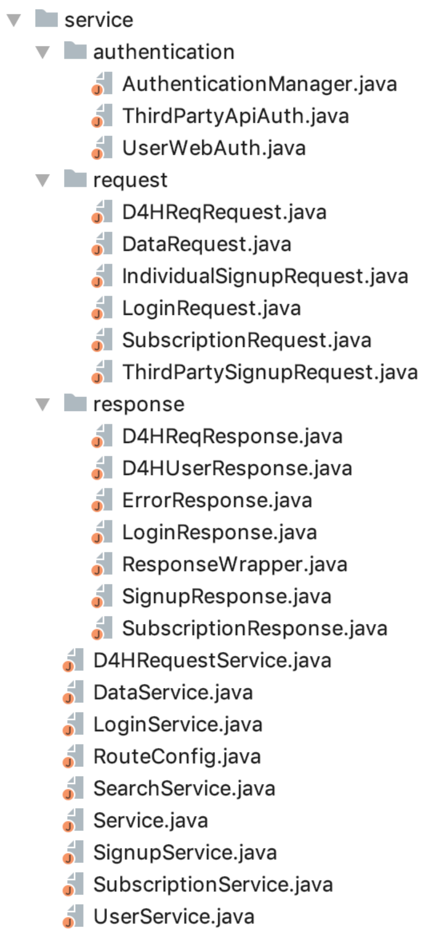
- **resource**: contains classes that directly manage the data, by interacting with the database and retrieving them to services. There are *SubscriptionResource, UserResource,* among other.



```
▼ resource
  ▼ api
      BulkDataRequest.java
      IndividualDataRequest.java
      NotificationRequest.java
    APIManager.java
    D4HRequestResource.java
    DataResource.java
    Resource.java
    SubscriptionResource.java
    UserResource.java
```

Figure 4.3: D4H Source code - Resources

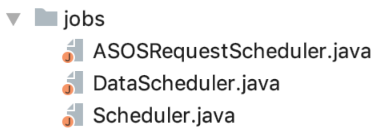
- **service**: classes in charge of controlling the application logic and expose methods and routes that can be consumed by frontend applications. D4H provides services such as *SearchService, SubscriptionService*, etc.



```
service
  authentication
    AuthenticationManager.java
    ThirdPartyApiAuth.java
    UserWebAuth.java
  request
    D4HReqRequest.java
    DataRequest.java
    IndividualSignupRequest.java
    LoginRequest.java
    SubscriptionRequest.java
    ThirdPartySignupRequest.java
  response
    D4HReqResponse.java
    D4HUserResponse.java
    ErrorResponse.java
    LoginResponse.java
    ResponseWrapper.java
    SignupResponse.java
    SubscriptionResponse.java
  D4HRequestService.java
  DataService.java
  LoginService.java
  RouteConfig.java
  SearchService.java
  Service.java
  SignupService.java
  SubscriptionService.java
  UserService.java
```

Figure 4.4: D4H Source code - Services

- **jobs**: contains scheduled async task, that repeat according to how they were configured. For instance, the DataScheduler job looks for all the subscriptions that should be executed, makes a search using the saved query, anonymizes the data, and sends it to the third party.



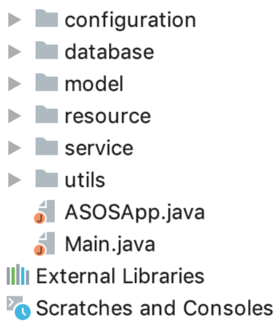
```
jobs
  ASOSRequestScheduler.java
  DataScheduler.java
  Scheduler.java
```

Figure 4.5: D4H Source code - Jobs

4.2 ASOS Backend (Spark project)

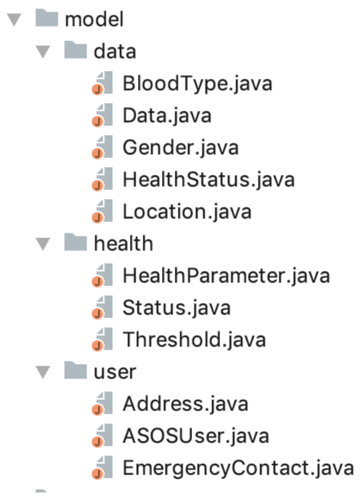
ASOS has the same structure that D4H, but in this case, since ASOS does not expose services to possible consumer apps, it has fewer classes and all of them have a specific task to complete.

All the important requirements that ASOS need to fulfill are in charge of DataService and



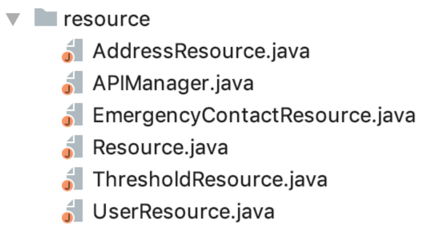
```
▶ configuration
▶ database
▶ model
▶ resource
▶ service
▶ utils
  ASOSApp.java
  Main.java
External Libraries
Scratches and Consoles
```

Figure 4.6: ASOS Source code - General Structure



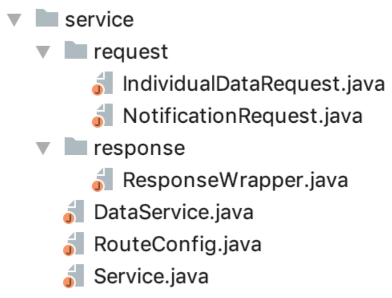
```
▼ model
  ▼ data
      BloodType.java
      Data.java
      Gender.java
      HealthStatus.java
      Location.java
  ▼ health
      HealthParameter.java
      Status.java
      Threshold.java
  ▼ user
      Address.java
      ASOSUser.java
      EmergencyContact.java
```

Figure 4.7: ASOS Source code - Models



```
▼ resource
    AddressResource.java
    APIManager.java
    EmergencyContactResource.java
    Resource.java
    ThresholdResource.java
    UserResource.java
```

Figure 4.8: ASOS Source code - Resources

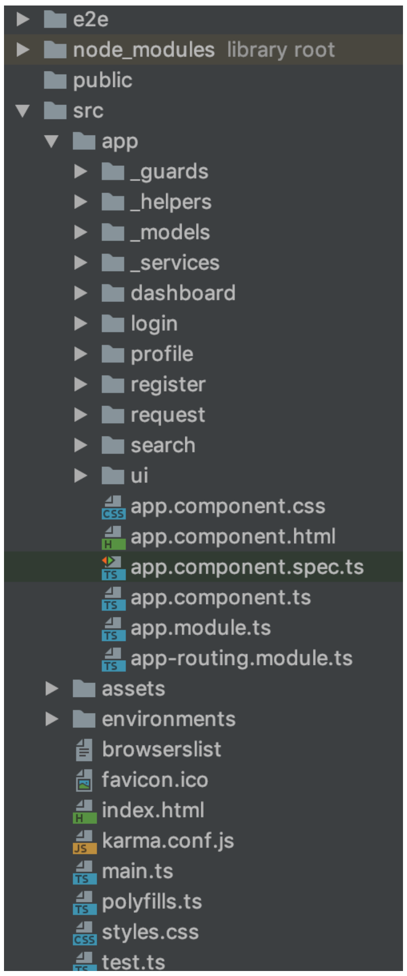


```
▼ service
  ▼ request
      IndividualDataRequest.java
      NotificationRequest.java
  ▼ response
      ResponseWrapper.java
    DataService.java
    RouteConfig.java
    Service.java
```

Figure 4.9: ASOS Source code - Services

4.3 D4H Frontend (Angular 7 project)

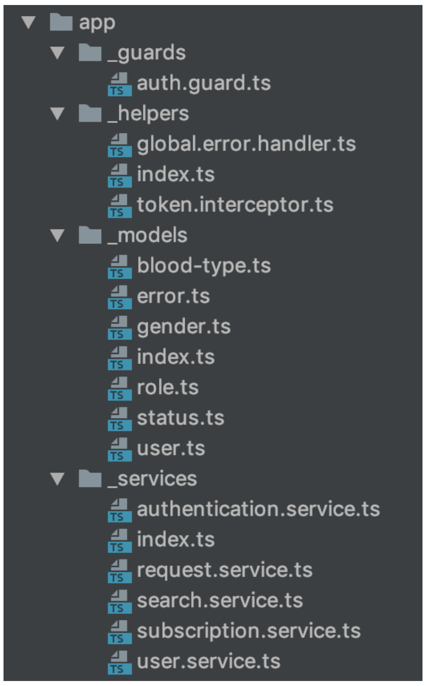
The whole project is contained in the folder src (Figure 4.10). In this folder there three main parts: app, assets and environment. Outside of them there are other configuration files.



```
▶ e2e
▶ node_modules  library root
  public
▼ src
  ▼ app
    ▶ _guards
    ▶ _helpers
    ▶ _models
    ▶ _services
    ▶ dashboard
    ▶ login
    ▶ profile
    ▶ register
    ▶ request
    ▶ search
    ▶ ui
      app.component.css
      app.component.html
      app.component.spec.ts
      app.component.ts
      app.module.ts
      app-routing.module.ts
  ▶ assets
  ▶ environments
    browserslist
    favicon.ico
    index.html
    karma.conf.js
    main.ts
    polyfills.ts
    styles.css
    test.ts
```

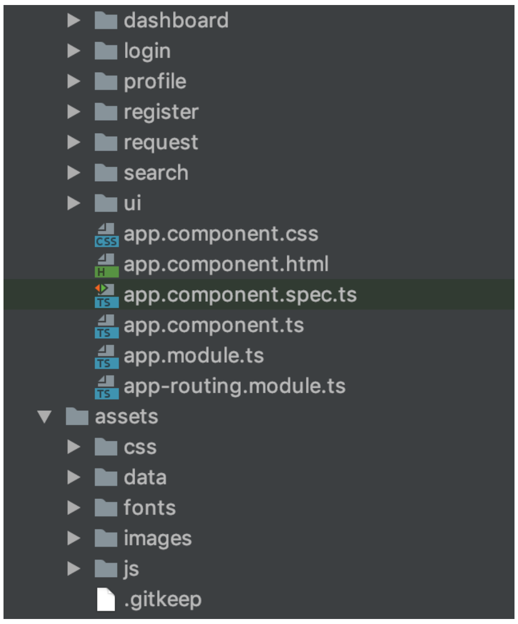
Figure 4.10: Source code: General Structure

- **app**: this folder contains the source code of the application, so we will focus on its content (Figure 4.11).



```
▼ app
  ▼ _guards
      auth.guard.ts
  ▼ _helpers
      global.error.handler.ts
      index.ts
      token.interceptor.ts
  ▼ _models
      blood-type.ts
      error.ts
      gender.ts
      index.ts
      role.ts
      status.ts
      user.ts
  ▼ _services
      authentication.service.ts
      index.ts
      request.service.ts
      search.service.ts
      subscription.service.ts
      user.service.ts
```

Figure 4.11: Source code: Models and Services



```
  ▶ dashboard
  ▶ login
  ▶ profile
  ▶ register
  ▶ request
  ▶ search
  ▶ ui
    app.component.css
    app.component.html
    app.component.spec.ts
    app.component.ts
    app.module.ts
    app-routing.module.ts
▼ assets
  ▶ css
  ▶ data
  ▶ fonts
  ▶ images
  ▶ js
    .gitkeep
```

Figure 4.12: Source code: Components

– **_guards** and **_helpers** contain some general purpose functions used by services and components.

– **_models** contain the definition of classes and enumerations regarding data structure in the app. They are highly useful to map data coming from the backend.

Angular distinguishes *components* from *services* to increase modularity and reusability. By separating a component's view-related functionality from other kinds of processing, you can make your component classes lean and efficient. Then,

  - _**services** are classes with a narrow, well-defined purpose. It should do something specific and do it well.
  - **components** they delegate certain tasks to services, such as fetching data from the server, validating user input, or logging directly to the console. In this case, dashboard, login, profile, register, request, search and ui are components, each one devoted to specific requirements.

- **assets**: contains all the asset files such as logos, images, videos, media etc.
- **environments**: is devoted to the environment based configurations, in this case contains the url to access the services exposed by the backend.

Chapter 5

Integration Strategy

5.1 Entry Criteria

In order for the integration testing to be possible and to produce meaningful results, there are a number of conditions on the progress of the project that have to be met.
First of all, the **Requirements Analysis and Specification Document** and the **Design Document** must have been fully written. This is a required step in order to have a complete picture of the interactions between the different components of the system and of the functionalities they offer.
Secondly, the integration process should start only when the estimated percentage of completion of every component with respect to its functionalities is:

- **100%** for the **Data4HelpWebService** component
- At least **90%** for the **LoginService and RegisterService** subsystem
- At least **70%** for the **SearchManager and RequestService** subsystem
- At least **50%** for the **ASOSService** applications

It should be noted that these percentages refer to the status of the project at the beginning of the integration testing phase and they do not represent the minimum completion percentage necessary to consider a component for integration, which must be at least **90%**. The choice of having different completion percentages for the different components has been made to reflect their order of integration and to take into account the required time to fully perform integration testing.

5.2 Elements to be integrated

In the following paragraph we're going to provide a list of all the components that need to be integrated together.
As specified in TrackMe Design Document, the system is built upon the interactions of many high-level components, each one implementing a specific set of functionalities. For the sake of modularity, each subsystem is further obtained by the combination of several lower-level components. Because of this software architecture, the integration phase will

involve the integration of components at two different levels of abstraction.

At the lowest level, we'll integrate together those components that depend strongly on one another to offer the higher level functionalities of **Data4HelpWebService**. In our specific case, this involves the integration of the **Login Service and Signup Service**, **Search Management**, **Request Management**, **Subscription and Notification Management** subcomponents in order to obtain the **Data4Help Management System** subsystem.

For what concerns the building of the **AutomatedSOS** and **Track4Run** subsystems, the integration activity is actually quite limited; in fact, they simply represent a collection of functionalities belonging to the same area which however are not dependent on one another. As a result of this, their subcomponents don't really interact with each other, and the integration phase will be limited to the task of ensuring that the set of functionalities of each subcomponent is properly exposed by the subsystem. The components involved in this phase are:

- The **Data Handler, Health Care Connector and DB Manger** subcomponents in order to obtain the **AutomatedSOS subsystem**.
- The **Login, Signup, User, Event, Notification, Data handler, Request and Authentication manager** subcomponents in order to obtain the **Track4Run Management subsystem**.

Some of these subcomponents also directly rely on higher level, atomic components: that is the case, for instance, of the dependency on the **Data Handler component**. These dependencies will be taken care of in the integration process.
Finally, we will proceed with the integration of the higher level subsystems. In particular, the integration activity will involve:

- The already existing components used to achieve specific functionalities: these are the **Health Care Service, DBMS and Notification system** components.
- Those components and subsystems specifically developed for TrackMeService, that are:
  - On the server side: the **Data4Help Management System, Authentication system, Search subsystems**, together with the **Data Handler** component.
  - On the client side: the **Data4Help Web Application and Track4Run Web Application** components.

5.3 Integration testing strategy

As already explained in the Design document, let's discuss here Integration strategy in more detail. The approach we're going to use to perform integration testing is based on a mixture of the bottom-up and critical-module-first integration strategies.

Using the bottom-up approach, we will start integrating together those components that do not depend on other components to function, or that only depend on already developed components. This strategy brings a number of important advantages. First, it allows us to perform integration tests on ”real” components that are almost fully developed and thus obtain more precise indications about how the system may react and fail in real world usage with respect to a top-down approach. Secondly, working bottom-up enables us to more closely follow the development process, which in our case is also proceeding using the bottom-up approach; by doing this we can start performing integration testing earlier in the development process as soon as the required components have been developed in order to maximize parallelism and efficiency.

Since subsystems are fairly independent from one another, the order in which they're integrated together to obtain the full system follows the critical-module-first approach. This strategy allows us to concentrate our testing efforts on the riskiest components first, that is those that represent the core functionalities of the whole system and whose malfunctioning could pose a very serious threat to the correct implementation of the entire TrackMe infrastructure. By proceeding this way, we are able to discover bugs earlier in the integration progress and take the necessary measures to correct them on time.

It should be noted that **Health Care Service, Notification System and DBMS** are commercial components that have already been developed and can thus be immediately used in a bottom-up approach without any explicit dependency.

5.4 Sequence of Component/Function Integration

This section describes the order of integration (and integration testing) of the various components and subsystems of TrackMe Service. This section has already been explained in detail in the **DD** - Design Document **(Reference section: 5.3.3 Integration order)**.

Chapter 6

Individual Steps and Test Description

In this chapter we'll provide a detailed description of the tests to be performed on each pair of components that have to be integrated.Each pair of components is described in a specific subsection, identified by the <caller;called> notation, containing the list of methods that the <caller> component invokes on the <called> component.For each method we're going to provide a brief description of the input values and the corresponding expected effects on the system.

6.1 Data4Help Management System

6.1.1 Request Management and DBHandler

Insert Request

Input	**Effect**
A NULL parameter	A NullArgumentException is raised.
A request with an id already existent in the database	An InvalidArgumentValueException is raised.
Formally valid arguments	An entry containing the request data is inserted into the database.

Delete Request

Input	**Effect**
A NULL parameter	A NullArgumentException is raised.
A request with an in-existent id	An InvalidArgumentValueException is raised.
Formally valid arguments	The entry containing the request data is deleted from the database.

Table 6.1: Request Management parameters

6.1.2 Search Management and DBHandler

Subscribe/Get Data

Input	**Effect**
A NULL parameter	A NullArgumentException is raised.
A search with an id in-existent in the database	An InvalidArgumentValueException is raised.
Formally valid arguments	The list of all valid data based on the search id.

Unsubscribe Data

Input	**Effect**
A NULL parameter	A NullArgumentException is raised.
A search with an in-existent id	An InvalidArgumentValueException is raised.
Formally valid arguments	The entry containing the search data is deleted from the requester's view.

Table 6.2: Search Management parameters

6.1.3 Accept/Reject Management and DBHandler

Update Request Queue (userid, requestid)

Input	**Effect**
A NULL parameter	A NullArgumentException is raised.
A non-existing RequestID	An InvalidArgumentValueException is raised.
A set of valid parameters to accept	The new user of the Data4Help is added to ThirdParty's view in the database.
A set of valid parameters to reject	The new user of the Data4Help is removed from request queue in the database.

Table 6.3: Accept/Reject Management parameters

6.1.4 Login Management and DBHandler

Login (userid, tokenid)

Input	**Effect**
A NULL parameter	A NullArgumentException is raised.
A non-existing user	An InvalidArgumentValueException is raised.
An empty password	An InvalidArgumentValueException is raised.
A valid user and password combination, which however is not the correct one	Returns an InvalidCredentialError.
A correct and valid user and password combination	Returns a session cookie.

Table 6.4: Login Management parameters

6.1.5 Signup Management and DBHandler

Signup (userid, tokenid)

Input	**Effect**
A NULL parameter	A NullArgumentException is raised.
An empty parameter	An InvalidArgumentValueException is raised.
All valid user data in all fields, which however belongs to existing user	Returns an InvalidCredentialError.
A correct and all valid user fields	Returns a session cookie and data inserted into the database.

Table 6.5: Signup Management parameters

6.1.6 Check Token (Password Retrieval) and DBHandler

CheckToken (userid, tokenid)

Input	**Effect**
A NULL parameter	A NullArgumentException is raised.
A valid user and secret-Code combination, which however is not the correct one	Returns False.
A correct and valid user and secretCode combination	Returns True.

UpdateUserPassword (userid, tokenid, newPassword)

Input	**Effect**
A NULL parameter	A NullArgumentException is raised.
A valid user and secret-Code combination, which however is not the correct one	An InvalidSecurityLevelException is raised.
A correct and valid user and secretCode combination, but an incorrectly formatted password	An InvalidArgumentFormatException is raised.
A correct and valid user and secretCode combination, and a correctly formatted password	Updates the user password in the database.

Table 6.6: Check Token/Update Password Management parameters

6.2 AutomatedSOS Management System

6.2.1 Health Care Connector system and DBHandler

DataRefresh (userid, vitalSigns, ThresholdCollection)

Input	**Effect**
A NULL parameter	A NullArgumentException is raised.
Vital Signs checked against the threshold collections and are inconsistent	An InvalidArgumentException is raised.
Vital Signs checked against the threshold collections and are consistent	Overwrite the old data with the latest update in the database until the next data fetch.

Table 6.7: Data Refresh Management parameters

6.3 Integration between subsystems

6.3.1 Data4Help system, AutomatedSOS system

EmergencyAlarm (userid, vitalSigns, ContactDetails)

Input	**Effect**
A NULL parameter	A NullArgumentException is raised.
A userId not correctly formatted	An InvalidArgumentFormatException is raised.
A userDetails whose contact details are invalid	An InvalidContactException is raised.
Vital Signs out of range	An AlarmRequest is raised and contact is sent to HealthCareService (external component) within 5 seconds.
Vital Signs in range	Overwrite the old data with the latest update until the next data fetch.

Table 6.8: Data4Help, ASOS integration Management parameters

Chapter 7

Required Program Stubs and Test Data

7.1 Program Stubs and Drivers

As we have mentioned in the Integration Testing Strategy section of this document, we are going to adopt a bottom-up approach to component integration and testing.

Because of this choice, we are going to need a number of drivers to actually perform the necessary method invocations on the components to be tested; this will be mainly accomplished in conjunction with the JUnit framework.

Here follows a list of all the drivers that will be developed as part of the integration testing phase, together with their specific role:

- **Data Access Driver** : this testing module will invoke the methods exposed by the **DB Handler** component in order to test its interaction with the **DB Manager**.
- **Request Management Driver** : this testing module will invoke the methods exposed by the **Request Management** subcomponent, including those with package - level visibility, in order to test its interaction with the **DB Handler, Notification System and the Subscription Management** components.
- **Search Management Driver** : this testing module will invoke the methods exposed by the **Subscription Management** subcomponent in order to test its interaction with the **DB Handler, Notification System and the Request Management** components.
- **Login Management Driver** : this testing module will invoke the methods exposed by the **Login Management** subcomponent in order to test its interaction with the **DB Handler and the token System** components.
- **Health Care Connector Driver** : this testing module will invoke the methods exposed by the **Health Care Connector Management** subcomponent in order

to test its interaction with the **DB Handler and the Health Care Service - external System** components.

- **Notification Management Driver** : this testing module will invoke the methods exposed by the **Notification Management** subcomponent in order to test its interaction with the **DB Handler, Request system and Search system** components.
- **Subscription Management Driver** : this testing module will invoke the methods exposed by the **Subscription Management** subcomponent in order to test its interaction with the **DB Handler and Search system** components.
- **Account Management Driver** : this testing module will invoke the methods exposed by the **Check Token Management** subcomponent in order to test its interaction with the **DB Handler, Login system and Signup system** components.

While the bottom-up approach in general doesn't require the usage of any stubs as the system is developed from the ground up, a full test of the core system isn't possible without introducing a few of them. In fact, there is a mutual dependency between the clients (which send requests) and the core system (which replies to them). Since we are developing and integrating the system from the core, we are going to introduce stubs to simulate the presence of clients until they are fully developed. In practice, the only purpose of these stubs is to write on a log that they have correctly received the messages.

7.2 Test Data

In order to be able to perform the record of tests that we have specified, we are going to need:

- A list of both valid and invalid individual or third party users to test the **Signup Management** component. The set should contain instances exhibiting the following problems:
  - Null object
  - Null fields
  - Invalid data in one or more fields
  - Tax certificate not compliant with the legal format
  - Valid data in all fields
- A list of both valid and invalid individual or third party users to test the **Login Management** component. The set should contain instances exhibiting the following problems:

  - Null object
  - Null fields
  - Invalid data in one or more fields
  - valid data but system down
  - Valid data in all fields

- A list of both valid and invalid requests to test the **Request Management** component. The set should contain instances exhibiting the following problems:
  - Null object
  - Null fields
  - Invalid data in one or more fields
  - valid data but incorrect format
  - Valid data in all fields
- A list of both valid and invalid searches to test the **Search Management** component. The set should contain instances exhibiting the following problems:
  - Null object
  - Null fields
  - Invalid data in search
  - valid data but does not exist in DB
  - Valid data
- A list of both valid and invalid searches to test the **Subscription Management** component. The set should contain instances exhibiting the following problems:
  - Null object
  - Null fields
  - valid data but does not exist in DB
  - Valid data
- A list of both valid and invalid notifications to test the **Notification Management** component. The set should contain instances exhibiting the following problems:
  - Null object
  - Null fields

  - valid data but does not exist in DB
  - Valid data

- A list of both valid and invalid notifications to test the **Health Care Service Management** component. The set should contain instances exhibiting the following problems:
  - Null object
  - Inconsistent data against threshold
  - Consistent data against threshold
  - Valid data

More specific information about the required test data can be found by analysing the inputs of all the test cases described in chapter 3.

7.3 Test Scenario

Scenario testing is a software testing activity that uses scenarios: hypothetical stories to help the tester work through a complex problem or test system. The ideal scenario test is a credible, complex, compelling or motivating story the outcome of which is easy to evaluate. The following high-level set of scenarios were considered for **Data4Help** system:

Scenario List (Data4Help Module)

SC01	Validate the login functionality of the system
SC02	Validate the login functionality of the system with blank data
SC03	Validate if third party is able to request individual's data
SC04	Validate if third party is able to request bulk data
SC05	Validate the individual's response to request (Accept/Reject)
SC06	Validate third party is able to search for the subscribed data on its dashboard
SC07	Validate Individual is able to view for the subscribers data on its dashboard

Table 7.1: Test Scenario List

7.4 Test Cases

A test case is a specification of the inputs, execution conditions, testing procedure, and expected results that define a single test to be executed to achieve a particular software testing objective. We will define a number of test cases against the test scenarios above stated to cover the Data4Help complete system. The following test cases are considered against the test scenarios:

- Login Positive Test Case

Test Scenario ID		Login-1	**Test Case ID**		Login-1A
Test Case Description		Login-Positive	**Test Priority**		High
Pre-Requisite		A valid user account	**Post-Requisite**		NA
Test Execution Steps:					
S. No.	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Launch Application	/login.html	Login Page	Login Page	Pass
2	Enter Correct Email & Password and hit Login button	Email ID: test@xyz.com; Password: ********	Login Success; User-id and access-token validated during authentication	Login Success; User-id and access-token validated during authentication	Pass

Table 7.2: Test Case: Login-1A

- Login-Negative Test Case

Test Scenario ID		Login-1	**Test Case ID**		Login-1B
Test Case Description		Login-Negative	**Test Priority**		High
Pre-Requisite		NA	**Post-Requisite**		NA
Test Execution Steps:					
S. No.	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Launch Application	/login.html	Login Page	Login Page	Pass
2	Enter invalid Email & any Password and hit Login button	Email ID: invalid@xyz.com; Password: ********	Error: ”The provided email or password are invalid”	Error: ”The provided email or password are invalid”	Pass
3	Enter valid Email & any Password and hit Login button	Email ID: valid@xyz.com; Password: ********	Error: ”The provided email or password are invalid”	Error: ”The provided email or password are invalid”	Pass

Table 7.3: Test Case: Login-1B

- Login Blank Test Case

Test Scenario ID		Login-2	**Test Case ID**		Login-2A
Test Case Description		Login-Blank	**Test Priority**		High
Pre-Requisite		NA	**Post-Requisite**		NA
Test Execution Steps:					
S. No.	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Launch Application	/login.html	Login Page	Login Page	Pass
2	Do not enter email or password and hit Login Button	No data	Error: "The provided email or password are invalid"	Error: "The provided email or password are invalid"	Pass

Table 7.4: Test Case: Login-2A

- Manage Request Individual Test Case

Test Scenario ID		Manage_Request-1	**Test Case ID**		Manage Request-1A
Test Case Description		Request Positive	**Test Priority**		High
Pre-Requisite		Valid Third party already registered & logged in to the dashboard screen	**Post-Requisite**		NA
Test Execution Steps:					
S. No.	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Launch Application	/login.html	Login Page	Login Page	Pass
2	Enter valid data in all fields and hit Login Button	Email ID: test@xyz.com; Password: ********	User is able to view the dashboard	User is able to view the dashboard	Pass
3	Select Request_type: "Specific", Filter type, & enter data and hit submit button	Request_Type: Specific; Filter_Type: SSN; Enter_Data: 123456789	User's request sent to specific individual; Request status changed to Request 'Pending'	User's request sent to specific individual; Request status changed to Request 'Pending'	Pass

Table 7.5: Test Case: Manage Request-1A

- Manage Request Individual Fail Test Case

Test Scenario ID		Manage_Request-1	**Test Case ID**		Manage Request-1B
Test Case Description		Request Fail	**Test Priority**		High
Pre-Requisite		Valid Third party already registered & logged in to the dashboard screen	**Post-Requisite**		NA
Test Execution Steps:					
S. No.	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Launch Application	/login.html	Login Page	Login Page	Pass
2	Enter valid data in all fields and hit Login Button	Email ID: test@xyz.com; Password: ********	User is able to view the dashboard	User is able to view the dashboard	Pass
3	Select Request_type: "Specific", Filter type, & enter data and hit submit button	Request_Type: Specific; Filter_Type: SSN; Enter_Data: 123456789	Error:"Session is not valid any more"	Error:"Session is not valid any more"	Pass

Table 7.6: Test Case:Manage_Request-1B

- Manage Request Bulk Positive Test Case

Test Scenario ID		Manage Request-2	**Test Case ID**		Manage Request-2A
Test Case Description		Request Bulk	**Test Priority**		High
Pre-Requisite		Valid Third party already registered & logged in to the dashboard screen	**Post-Requisite**		NA
Test Execution Steps:					
S. No.	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Launch Application	/login.html	Login Page	Login Page	Pass
2	Enter valid data in all fields and hit Login Button	Email ID: test@xyz.com; Password: ********	User is able to view the dashboard	User is able to view the dashboard	Pass
3	Select Request_type: "Bulk", Filter type, & enter data and hit submit button	Request_Type: Specific; Filter_Type: Blood Type; Enter_Data: A+	Request sent to TrackMe successfully; Request status changed to Request 'Pending'	Request sent to TrackMe successfully; Request status changed to Request 'Pending'	Pass

Table 7.7: Test Case:Manage Request-2A

- Manage Request Bulk Fail Test Case

Test Scenario ID		Manage Request-2	**Test Case ID**		Manage Request-2B
Test Case Description		Bulk Fail	**Test Priority**		High
Pre-Requisite		Valid Third party already registered & logged in to the dashboard screen	**Post-Requisite**		NA
Test Execution Steps:					
S. No.	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Launch Application	/login.html	Login Page	Login Page	Pass
2	Enter valid data in all fields and hit Login Button	Email ID: test@xyz.com; Password: ********	User is able to view the dashboard	User is able to view the dashboard	Pass
3	Select Request_type: "Bulk", Filter type, & enter data and hit submit button	Request_Type: Specific; Filter_Type: Blood Type; Enter_Data: A+	Error:"Session is not valid any more"	Error:"Session is not valid any more"	Pass

Table 7.8: Test Case: Manage Request-2B

- Manage Request Individual Accept Test Case

Test Scenario ID		Manage Request-3	**Test Case ID**		Manage Request-3A
Test Case Description		Individual accept	**Test Priority**		High
Pre-Requisite		Valid Individual already registered & logged in to the dashboard screen	**Post-Requisite**		NA
Test Execution Steps:					
S. No.	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Launch Application	/login.html	Login Page	Login Page	Pass
2	Enter valid data in all fields and hit Login Button	Email ID: test@xyz.com; Password: ********	User is able to view the dashboard	User is able to view the dashboard	Pass
3	All requests visible on dashboard	NA	User is able to view all the requests available on its dashboard screen.	User is able to view all the requests available on its dashboard screen.	Pass
4	Select Accept response drop down button and select submit	Response: Accept	Status against the request-id is changed to 'Approved'.	Status against the request-id is changed to 'Approved'.	Pass

Table 7.9: Test Case: Manage Request-3A

- Manage Request Individual Reject Test Case

Test Scenario ID		Manage Request-3	**Test Case ID**		Manage Request-3B
Test Case Description		Individual reject	**Test Priority**		High
Pre-Requisite		Valid Individual already registered & logged in to the dashboard screen	**Post-Requisite**		NA
Test Execution Steps:					
S. No.	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Launch Application	/login.html	Login Page	Login Page	Pass
2	Enter valid data in all fields and hit Login Button	Email ID: test@xyz.com; Password: ********	User is able to view the dashboard	User is able to view the dashboard	Pass
3	All requests visible on dashboard	NA	User is able to view all the requests available on its dashboard screen.	User is able to view all the requests available on its dashboard screen.	Pass
4	Select Reject from the response dropdown button and select submit	Response: Reject	Status against the request-id is changed to 'Rejected'.	Status against the request-id is changed to 'Rejected'.	Pass

Table 7.10: Test Case: Manage Request-3B

- Search Subscribed data Test Case

Test Scenario ID	Search-1	**Test Case ID**	Search-1A
Test Case Description	requested Positive	**Test Priority**	High
Pre-Requisite	Valid Third party already registered & logged in to the dashboard screen	**Post-Requisite**	NA

Test Execution Steps:

S. No.	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Launch Application	/login.html	Login Page	Login Page	Pass
2	Enter valid data in all fields and hit Login Button	Email ID: test@xyz.com; Password: ********	User is able to view the dashboard	User is able to view the dashboard	Pass
3	All requests visible on dashboard	NA	User is able to view all the requests available on its dashboard screen.	User is able to view all the requests available on its dashboard screen.	Pass
4	Select Data from the filter available	Previous requested data	User is able to view all the subscribed data from the previous requests whose status = 'Approved'	User is able to view all the subscribed data from the previous requests whose status = 'Approved'	Pass

Table 7.11: Test Case: Search-1A

- Search Subscribed Data Fail Test Case

Test Scenario ID		Search-1	**Test Case ID**		Search-1B
Test Case Description		requested fail	**Test Priority**		High
Pre-Requisite		DB server is down after user logged in	**Post-Requisite**		NA
Test Execution Steps:					
S. No.	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Launch Application	/login.html	Login Page	Login Page	Pass
2	Enter valid data in all fields and hit Login Button	Email ID: test@xyz.com; Password: ********	User is able to view the dashboard	User is able to view the dashboard	Pass
3	All requests visible on dashboard	NA	User is able to view all the requests available on its dashboard screen.	User is able to view all the requests available on its dashboard screen.	Pass
4	Select Data from the filter available	Previous requested data	Error: ”Session is not valid any more”	Error: ”Session is not valid any more”	Pass

Table 7.12: Test Case: Search-1B

- Search Subscribers Positive test case

Test Scenario ID		Search-2	**Test Case ID**		Search-2A
Test Case Description		Subscribers Positive	**Test Priority**		High
Pre-Requisite		Valid Individual already registered & logged in to the dashboard screen	**Post-Requisite**		NA
Test Execution Steps:					
S. No.	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Launch Application	/login.html	Login Page	Login Page	Pass
2	Enter valid data in all fields and hit Login Button	Email ID: test@xyz.com; Password: ********	User is able to view the dashboard	User is able to view the dashboard	Pass
3	All requests visible on dashboard	NA	User is able to view all the requests available on its dashboard screen.	User is able to view all the requests available on its dashboard screen.	Pass
4	Select Subscriber's List from the Dashboard	Previous requested data	User is able to view all users who currently subscribed to its data from requests whose status = 'Approved'	User is able to view all users who currently subscribed to its data from requests whose status = 'Approved'	Pass

Table 7.13: Test Case:Search-2A

- Search Subscribers Fail Test Case

Test Scenario ID		Search-2	**Test Case ID**		Search-2B
Test Case Description		Subscribers Fail	**Test Priority**		High
Pre-Requisite		DB server is down after user logged in	**Post-Requisite**		NA
Test Execution Steps:					
S. No.	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Launch Application	/login.html	Login Page	Login Page	Pass
2	Enter valid data in all fields and hit Login Button	Email ID: test@xyz.com; Password: ********	User is able to view the dashboard	User is able to view the dashboard	Pass
3	All requests visible on dashboard	NA	User is able to view all the requests available on its dashboard screen.	User is able to view all the requests available on its dashboard screen.	Pass
4	Select Subscriber's List from the Dashboard	Previous requested data	Error: "Session is not valid any more"	Error: "Session is not valid any more"	Pass

Table 7.14: Test Case: Search-2B

Chapter 8

Installation instructions

The TrackMe project is composed by 3 different systems: Data4Help, AutomatedSOS, and Track4Run. We decided to develop Data4Help, which is the leading system, and AutomatedSOS.

8.1 How to run both systems?

The whole system is dockerized, just run the follow the following steps:

1. Install **Docker** and **Docker Compose**.
2. At the level of *TrackMe/src* folder, open a terminal and execute **sudo docker-compose up** (with –build to rebuild the image).
3. Open the browser and you can access to the Data4Help site using the following URL **http://0.0.0.0:4200**

8.2 How to each system independently?

8.2.1 How to work on the back-end?

The following are the steps needed to work on the back-end:

1. You will need **JDK** >= 8.
2. Install **Apache Maven**.
3. Install **MongoDB** and **Redis**.
4. In the folder *TrackMe/src/data4help/*, open a terminal and run **mvn compile**.
5. *[Optional]* To run the service:
   (a) Make sure that MongoDB and Redis services are running.
   (b) Execute the following line:
   mvn -X compile exec:java -Dexec.mainClass=avila.schiatti.virdi.Main -e.

(c) You will be able to access the site/services by accessing to **http://127.1.1.1:4567**

In order to be able to have the front end, you should follow the steps of the front-end section, without running it

8.2.2 How to work on the front-end?

In order to work on the front-end, follow this steps:

1. Install **Nodejs and NPM**.
2. Install **AngularCLI** by running the following line: **npm install -g @angular/cli**
3. Go to *src/main/resources* folder and install the package dependencies running: **npm install**
4. Build the project by running **ng build**.
5. To run the front-end, you can run: **ng serve**. You will be able to access the front-end by accessing to **http://127.1.1.1:4200**

   You won't have access to the back-end services, so probably the front-end alone is not useful)

Chapter 9

Effort spent

Team Work	
Task	**Hours**
Planning Integration	8
Testing overview	4
Choosing Strategy	3
Checking document	4
Total	19

Table 9.1: Time spent by all team members

Individual Work					
Diego Avila		**Laura Schiatti**		**Sukhpreet Kaur**	
Task	**Hours**	**Task**	**Hours**	**Task**	**Hours**
X	X	X	X	Layout	2
X	X	X	X	Purpose and Scope	3
X	X	X	X	Implemented Requirements	6
X	X	X	X	Individual Steps, Testing	5
X	X	X	X	Stubs and Test Data	10
X		X	4	Adopted Frameworks	4
Total	X	**Total**	X	**Total**	30

Table 9.2: Time spent by each team member

Chapter 10

References

- Requirement Analysis and Specification Document.pdf. Version 1.1 - 11.11.2018
- Design Document.pdf. Version 1.0 - 10.12.2018
- **Spark** - http://sparkjava.com/
- **Morphia** - http://morphiaorg.github.io/morphia/
- **Lettuce** - https://lettuce.io/
- **Angular** - https://angular.io/
- **Docker** - https://docs.docker.com/