POLITECNICO
MILANO 1863

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/master/src

January 13th , 2019

Contents

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.

Furthermore, in this document it is explained how the code is organized, and how to prepare the environment in order to continue the work done so far. Also, it is explained how to run the systems, and the way they interact.

More precisely, the document presents:

- A list of all the implemented requirements, for Data4Help and AutomatedSOS
- The adopted programming languages and frameworks
- A description and structure of the source code
- The sequence in which components and subsystems will be integrated
- A description of the test plan, and the tests done so far

- Installation and development instructions

.

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 location, age range, gender and blood type criteria
D4H::SearchService

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

[R11] The system must subscribe a third party company to an individual health status and location when the request is approved by the individual
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 Known bugs and future work

- **No access token expiration checking in the front-end**: it is missing a way to check whether the token is no longer valid before making requests to D4H API. Then, future releases should include a checking mechanism and it corresponding redirection to the login page.
- **Search by minAge and maxAge**: currently there is a bug in the search, in which when looking for individuals between 2 ages, the maximum age is not set correctly
  query.setMinAge(minAge);
  query.setMinAge(maxAge);
- **Possible manual modification of the user role**: even if the user role is stored in localstorage variables, and therefore can be modified, the back-end has several checkings that avoid unauthorized users to access protected areas. For instance, even if the role changes from individual to third party, that user cannot search for data, an the error *"The current user is not authorized to perform the operation"* will be displayed. For the future it will be useful to implement another way of keeping track of the role of the current user.

2.5 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 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
- Does not need a runtime plugin
- It is used in popular frameworks
- Learning Curve

Disadvantages

- 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.

Advantages

- Rapid development
- Easy 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.

3.2.1 Java Spark framework

Java Spark Framework is a simple and expressive Java web framework DSL built for rapid development. Sparks intention is to provide an alternative for Java developers that want to develop their web applications as expressive as possible and with minimal boilerplate. 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.

Spark Java 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. The developers can control what route mappings to declare, what code is the responsible for handling the requests and how to handle everything else.

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 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.

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 back-end (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.

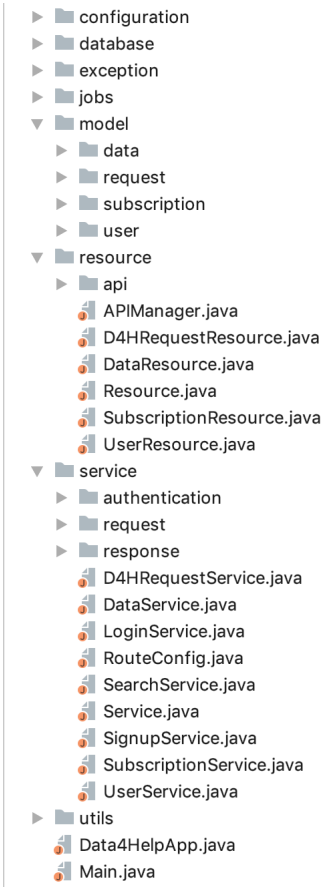

Figure 4.1: D4H Source code - General Structure

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

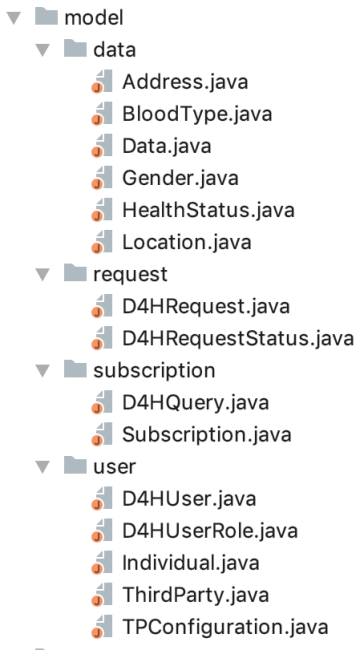

Figure 4.2: D4H Source code - Models

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

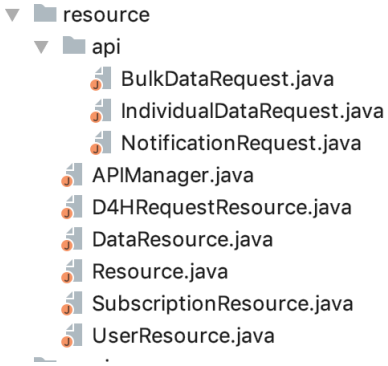

Figure 4.3: D4H Source code - Resources

- **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.

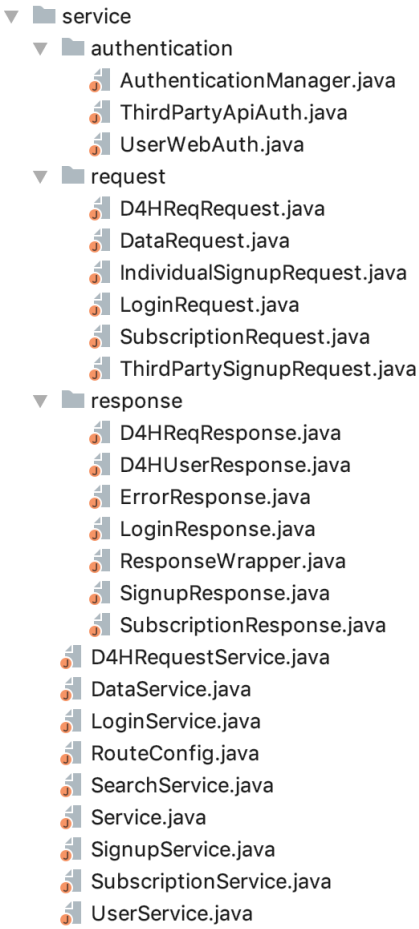

Figure 4.4: D4H Source code - Services

- **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.

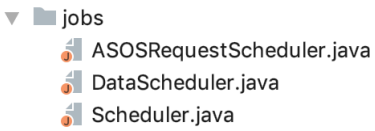

Figure 4.5: D4H Source code - Jobs

4.2 ASOS back-end (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

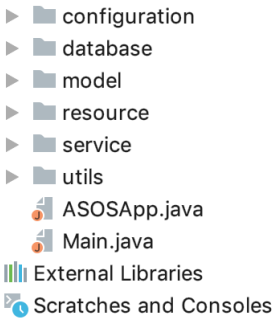

Figure 4.6: ASOS Source code - General Structure

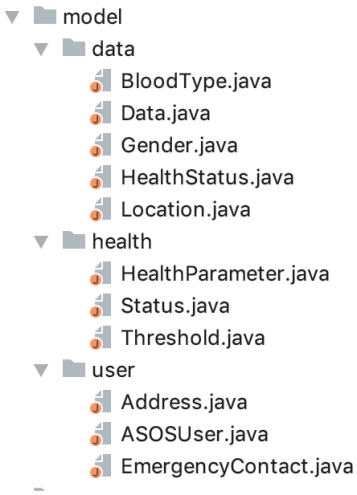

Figure 4.7: ASOS Source code - Models

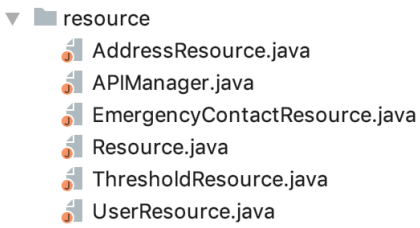

Figure 4.8: ASOS Source code - Resources

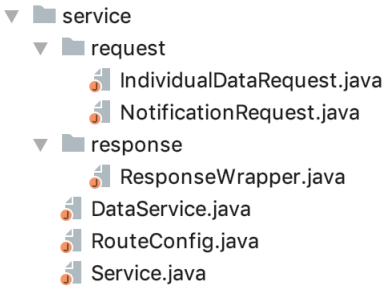

Figure 4.9: ASOS Source code - Services

4.3 D4H front-end (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.

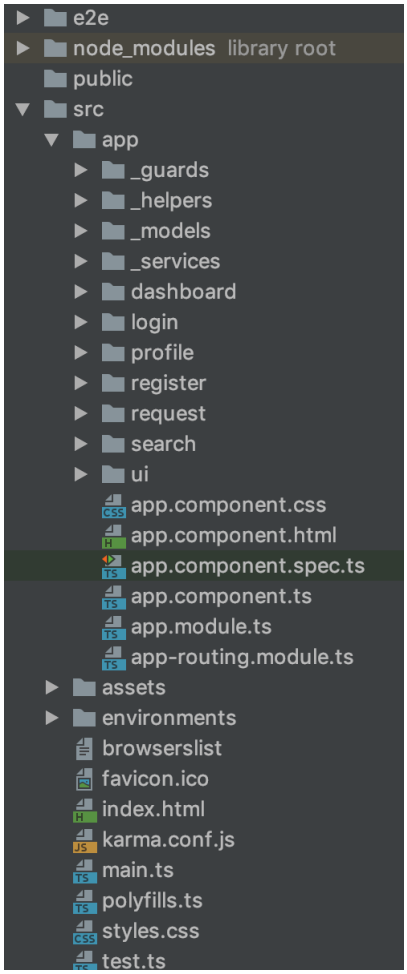

Figure 4.10: Source code: General Structure

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

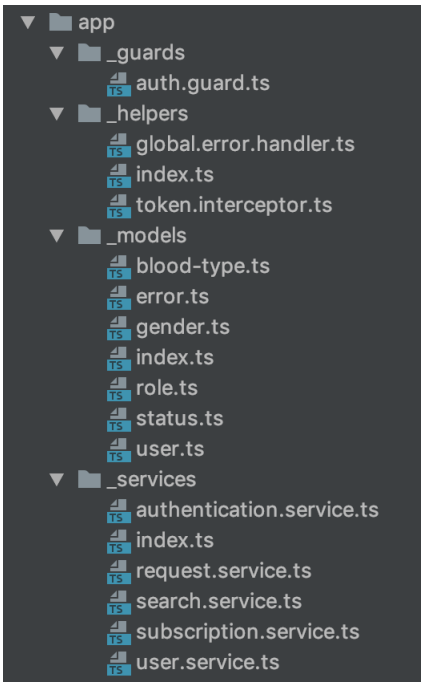

Figure 4.11: Source code: Models and Services

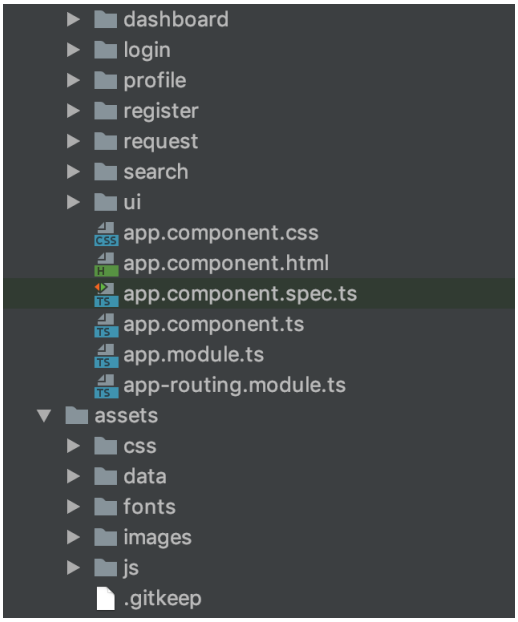

Figure 4.12: Source code: Components

– **_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 **Data4Help Backend** component
- At least **90%** for the **LoginService and RegisterService** subsystem
- At least **70%** for the **SearchManager and RequestService** subsystem
- At least **50%** for the **ASOS Backend** 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

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 **Data4Help Backend**. This involves the integration of the **Login Service and Signup Service**, **Search Service**, **Request Service**, **Subscription Service** components 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 **DataService, APIManager and DBManger** 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 **DataService** 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 **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 Service** component.
  - On the client side: the **Data4Help Web Application and Track4Run Web Application** components.

5.3 Integration testing strategy

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.

5.4 Sequence of component/function integration

This section has already been explained in detail in the **DD** - Design Document **(Reference section: 5.3.3 Integration order)**.

Chapter 6

Test plan

6.1 Test scenarios

In the following section a list of test scenarios is described. These scenarios were used to test D4H back-end and front-end, and ASOS back-end. The test scenarios were categorized by its importance, based on how critical are for the system.

Scenario List - D4H		
ID	**Description**	**Importance**
SC01	Validate the login functionality of the system	Medium
SC02	Validate the login functionality of the system with blank data	Medium
SC03	Validate if third party is able to request individual's data	High
SC04	Validate if third party is able to request bulk data	High
SC05	Validate the individual's response to request (Accept/Reject)	High
SC06	Validate third party is able to see the subscribed data on its dashboard	Low

Table 6.1: Test scenarios for D4H

Scenario List - ASOS		
ID	**Description**	**Importance**
SC07	Validate notification service	High
SC08	Validate data service	High
SC09	Validate threshold checking	High
SC10	Validate contacting health care service	High

Table 6.2: Test scenarios for ASOS

6.2 Test cases

The tests cases were done by hand, following the steps described below and based on the importance described in the Test Scenarios section. Not all test cases were executed,

only those with High and Medium importance.

Moreover, for testing purposes, a data generator tool has been made. It was developed using PHP as a language, and it generates and sends health data and location for all the D4H users. The testing environment was done using Docker as a virtual environment, and Docker Compose to setup the systems: Data4Help back-end and front-end, MongoDB and Redis databases, AutomatedSOS back-end, and the data generator tool.

6.2.1 Data4Help test cases

- **Login positive**
  Scenario ID: SC01
  Pre-requisite: Valid user account
  Expected output: User has logged in successfully
  Steps:

  1. Launch application
  2. Complete field Email with a valid email
  3. Complete field Password with a valid password
  4. Press Login button

  Actual output: Login successful

- **Login negative**
  Scenario ID: SC02
  Pre-requisite: Non user account
  Expected output: Error show
  Steps:

  1. Launch application
  2. Complete field Email with a non valid email
  3. Complete field Password with a non valid password
  4. Press Login button

  Actual output: The provided email and password are invalid

- **Login negative 2**
  Scenario ID: SC02
  Pre-requisite: Non user account
  Expected output: Error show
  Steps:

  1. Launch application
  2. Press Login button

  Actual output: The provided email and password are invalid

- **Send Request**
  Scenario ID: SC03
  Pre-requisite: Valid user account
  Expected output: Request sent
  Steps:

  1. Launch application
  2. Login into the application with a third party account
  3. Click on Search
  4. Fill in the SSN with a valid SSN number
  5. Click Filter button
  6. Accept the message to send the request

  Actual output: The request has been sent

- **Send Request - negative**
  Scenario ID: SC03
  Pre-requisite: Valid user account
  Expected output: Error shown
  Steps:

  1. Launch application
  2. Login into the application with a third party account
  3. Click on Search
  4. Fill in the SSN with a non valid SSN number

5. Click Filter button

Actual output: An error is shown

- **Search bulk data**
  Scenario ID: SC04
  Pre-requisite: Valid user account
  Expected output: A list with data is shown
  Steps:

  1. Launch application
  2. Login into the application with a third party account
  3. Click on Search
  4. Click on Bulk data
  5. Fill in the select a Gender
  6. Click Filter button

  Actual output: A list of data is shown

- **Search bulk data - negative 1**
  Scenario ID: SC04
  Pre-requisite: Valid user account, no data to be shown
  Expected output: no data shown
  Steps:

  1. Launch application
  2. Login into the application with a third party account
  3. Click on Search
  4. Click on Bulk data
  5. Fill in the select a Gender
  6. Click Filter button

  Actual output: An empty list is shown

- **Accept a request**
  Scenario ID: SC05
  Pre-requisite: Valid user account, request from a third party
  Expected output: Approved request
  Steps:

  1. Launch application
  2. Login into the application with an Individual account
  3. Select Approved from the status list
  4. Click on Update]

  Actual output: The page reloads and the request is approved

---

- **Reject a request**
  Scenario ID: SC05
  Pre-requisite: Valid user account, request from a third party
  Expected output: Rejected request
  Steps:

  1. Launch application
  2. Login into the application with an Individual account
  3. Select Reject from the status list
  4. Click on Update]

  Actual output: The page reloads and the request is rejected

---

6.2.2 AutomatedSOS test cases

- **User notification - accept request**
  Scenario ID: SC07
  Pre-requisite: A request from ASOS has arrived
  Expected output: New user is saved in ASOS database
  Steps:

  1. Launch application
  2. Login as Individual
  3. Accept the ASOS request

4. Check ASOS database for new user

Actual output: New user has been created

- **User notification - reject request**
  Scenario ID: SC07
  Pre-requisite: A request from ASOS has arrived
  Expected output: No new user in ASOS database
  Steps:

  1. Launch application
  2. Login as Individual
  3. Reject the ASOS request
  4. Check ASOS database for new user

  Actual output: No user has been created

- **User data**
  Scenario ID: SC08, SC09, SC10
  Pre-requisite: User has accepted the ASOS request, User exist in the ASOS database
  Expected output: ASOS sends a message to the emergency service
  Steps:

  1. Send new data to D4H with heart rate equal to 0

  Actual output: ASOS has contacted the emergency contact

Chapter 7

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.

7.1 How to run both systems?

The whole system is dockerized, so, the steps to run the whole environment are:

1. Install **Docker** and **Docker Compose**.
2. At the level of *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 D4H website using the following URL **http://0.0.0.0:4200**

7.1.1 Initial data (seeds)

D4H system comes with initial data regarding **Individuals**, **Third parties**, **Requests**, **Subscriptions** and **Addresses**. This data is stored in the database when the system starts and those individuals are saved as *D4HUsers*. Furthermore, as mentioned before, the data received from devices is simulated by means of a php script *(d4h_data_generator)*. The script generates and delivers random data regarding health status and location for all the D4H users each minute.

Then, for testing the system, those seeds are available inside the source code, in the **src/seeders** folder.

7.1.2 How to work on the back-end?

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

1. **JDK** >= 8 is needed.

2. Install **Apache Maven**.
3. Install **MongoDB** and **Redis**.
4. In the folder *src/data4help*, open a terminal and run **mvn compile**.
5. *[Optional]* To run the service:
   (a) MongoDB and Redis services must be running.
   (b) Execute the following line:
   mvn -X compile exec:java -Dexec.mainClass=avila.schiatti.virdi.Main -e.
   (c) 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

7.1.3 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 8

Effort spent

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

Table 8.1: Time spent by all team members

Individual Work					
Diego Avila		**Laura Schiatti**		**Sukhpreet Kaur**	
Task	**Hours**	**Task**	**Hours**	**Task**	**Hours**
D4H back-end dev.	50	D4H front-end dev.	30	Layout	2
ASOS back-end dev.	10	Data gen. script	8	Purpose and Scope	3
D4H & ASOS TC	5	Imp. Reqs.	3	Imp. Reqs.	6
Chap. 7	2	Seeds	10	Testing	5
		Code structure	3	Stubs and Test Data	10
				Adopted Frameworks	4
Total	67	**Total**	54	**Total**	30

Table 8.2: Time spent by each team member

Chapter 9

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/