



**POLITECNICO**  
**MILANO 1863**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

---

# Implementation & Test Deliverable Document (ITD)

---

TRACKME

- v1.0 -

*Authors:*

**Avila**, Diego

**Schiatti**, Laura

**Virdi**, Sukhpreet

903988

904738

904204

SOURCE CODE REPOSITORY

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

January 13<sup>th</sup> , 2019

---

# Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Purpose . . . . .	1
1.2	Scope . . . . .	1
<b>2</b>	<b>Implemented Requirements</b>	<b>3</b>
<b>3</b>	<b>Adopted development frameworks</b>	<b>4</b>
3.1	Adopted Programming Languages . . . . .	4
3.1.1	Advantages and Disadvantages . . . . .	4
3.2	Adopted Middle-wares . . . . .	4
3.2.1	Advantages and Disadvantages . . . . .	4
3.3	Additional APIs . . . . .	4
<b>4</b>	<b>Structure of the source code</b>	<b>5</b>
<b>5</b>	<b>Integration Strategy</b>	<b>6</b>
5.1	Entry Criteria . . . . .	6
5.2	Elements to be integrated . . . . .	6
5.3	Integration testing strategy . . . . .	8
5.4	Sequence of Component/Function Integration . . . . .	8
<b>6</b>	<b>Individual Steps and Test Description</b>	<b>9</b>
6.1	Data4Help Management System . . . . .	9
6.1.1	Request Management and DBHandler . . . . .	9
6.1.2	Search Management and DBHandler . . . . .	10
6.1.3	Accept/Reject Management and DBHandler . . . . .	10
6.1.4	Login Management and DBHandler . . . . .	11
6.1.5	Signup Management and DBHandler . . . . .	11
6.1.6	Check Token (Password Retrieval) and DBHandler . . . . .	12
6.2	AutomatedSOS Management System . . . . .	13
6.2.1	Health Care Connector system and DBHandler . . . . .	13
6.3	Integration between subsystems . . . . .	13

6.3.1	Data4Help system, AutomatedSOS system . . . . .	13
<b>7</b>	<b>Required Program Stubs and Test Data</b>	<b>14</b>
7.1	Program Stubs and Drivers . . . . .	14
7.2	Test Data . . . . .	15
7.3	Test Scenario . . . . .	17
7.4	Test Cases . . . . .	17
<b>8</b>	<b>Installation instructions</b>	<b>33</b>
8.1	How to run both systems? . . . . .	33
8.1.1	How to run the tests image? . . . . .	33
8.2	How to each system independently? . . . . .	33
8.2.1	How to work on the back-end? . . . . .	33
8.2.2	How to work on the front-end? . . . . .	34
<b>9</b>	<b>Effort spent</b>	<b>35</b>
<b>10</b>	<b>References</b>	<b>36</b>

---

---

# List of Figures

---

---

# List of Tables

---

6.1	Request Management parameters . . . . .	9
6.2	Search Management parameters . . . . .	10
6.3	Accept/Reject Management parameters . . . . .	10
6.4	Login Management parameters . . . . .	11
6.5	Signup Management parameters . . . . .	11
6.6	Check Token/Update Password Management parameters . . . . .	12
6.7	Data Refresh Management parameters . . . . .	13
6.8	Data4Help, ASOS integration Management parameters . . . . .	13
7.1	Test Scenario List . . . . .	17
7.2	Test Case: Login-1A . . . . .	18
7.3	Test Case: Login-1B . . . . .	19
7.4	Test Case: Login-2A . . . . .	20
7.5	Test Case: Manage Request-1A . . . . .	21
7.6	Test Case:Manage_Request-1B . . . . .	22
7.7	Test Case:Manage Request-2A . . . . .	23
7.8	Test Case: Manage Request-2B . . . . .	24
7.9	Test Case: Manage Request-3A . . . . .	25
7.10	Test Case: Manage Request-3B . . . . .	27
7.11	Test Case: Search-1A . . . . .	29
7.12	Test Case: Search-1B . . . . .	30
7.13	Test Case:Search-2A . . . . .	31
7.14	Test Case: Search-2B . . . . .	32
9.1	Time spent by all team members . . . . .	35
9.2	Time spent by each team member . . . . .	35

# Introduction

---

## 1.1 Purpose

This document describes the plans for testing the integration of the created components. The purpose of this document is to test the interfaces between the components as described in Design Document. Every team member who cooperates in the integration tests should read this document.

## 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

.

# Implemented Requirements

---

put requirements / functions that are actually implemented in the software (with motivations for including and excluding others if applicable)...



# Adopted development frameworks

---

put references to DD...

## 3.1 Adopted Programming Languages

### 3.1.1 Advantages and Disadvantages

## 3.2 Adopted Middle-wares

### 3.2.1 Advantages and Disadvantages

## 3.3 Additional APIs

# Structure of the source code

---

put code here...

---

# 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)**.

# 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 <code>NullArgumentException</code> is raised.
A request with an id already existent in the database	An <code>InvalidArgumentValueException</code> is raised.
Formally valid arguments	An entry containing the request data is inserted into the database.
Delete Request	
Input	Effect
A NULL parameter	A <code>NullArgumentException</code> is raised.
A request with an in-existent id	An <code>InvalidArgumentValueException</code> 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 NullPointerException 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 NullPointerException 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 NullPointerException 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 <code>NullArgumentException</code> is raised.
A non-existing user	An <code>InvalidArgumentValueException</code> is raised.
An empty password	An <code>InvalidArgumentValueException</code> is raised.
A valid user and password combination, which however is not the correct one	Returns an <code>InvalidCredentialError</code> .
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 <code>NullArgumentException</code> is raised.
An empty parameter	An <code>InvalidArgumentValueException</code> is raised.
All valid user data in all fields, which however belongs to existing user	Returns an <code>InvalidCredentialError</code> .
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 NullPointerException 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 NullPointerException 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

# 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

---

# 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**

### 8.1.1 How to run the tests image?

1. After installing *Docker* and *Docker compose* and in the *src* folder, execute in a terminal: **sudo docker-compose -f —¿s**
2. This will run all the unit test cases and will exit. If the exit code is 0 and you see **Tests run: XX, Failures: 0, Errors: 0, Skipped: 0**, everything is ok.

## 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**  $\geq 8$ .
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)*

# Effort spent

Team Work	
Task	Hours
Planning Integration	8
Testing overview	4
Choosing Strategy	3
Checking document	4
<b>Total</b>	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	Integration Strategy	6
X	X	X	X	Individual Steps, Testing	5
X	X	X	X	Stubs and Test Data	10
X		X	4		
<b>Total</b>	X	<b>Total</b>	X	<b>Total</b>	26

Table 9.2: Time spent by each team member



# 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://morphismorg.github.io/morphism/>
- **Lettuce** - <https://lettuce.io/>
- **Angular** - <https://angular.io/>
- **Docker** - <https://docs.docker.com/>