

Politecnico di Milano
AA 2018/2019



POLITECNICO
MILANO 1863

Software Engineering 2

TrackMe

Acceptance Test Deliverable

Stefano Pecchia
921201

Edoardo Peretti
921286

Deliverable:	ATD
Title:	Acceptance Test Deliverable
Authors:	Stefano Pecchia and Edoardo Peretti
Version:	1.0
Date:	January 20, 2019
Download:	https://github.com/Speck1996/PecchiaPeretti
Copyright:	Copyright © 2019, S. Pecchia and E. Peretti – All rights reserved

Contents

1	Installation	3
2	Introduction	4
2.1	Introduction and Scope	4
2.2	Acronyms	4
2.3	Document structure	4
3	Testing	6
3.1	Introduction	6
3.2	System testing	6
3.3	Major Issues	6
3.4	Minor issues	7
3.5	Suggestions	9
4	Other Aspects	10
4.1	Quality of Code	10
4.2	Github Repository	10
4.3	Data simulation	10
4.4	Sample database	10
5	Effort spent	11
6	References	12

1 Introduction

1.1 Introduction and Scope

In this document the process of testing and validation activity of a Data4Help project made by an external team is presented. In particular the project analyzed is the one that can be found at <https://github.com/Framonti/MontiNappi> made by:

Fabio Nappi: fabio.nappi@mail.polimi.it

Francesco Monti: francesco4.monti@mail.polimi.it

In order to have a better experience in analyzing the code behind the deliverable provided, the following documents that can be found in the link provided above have been read:

RASD : this document was useful to get the overall functionalities that the team application should have provided.

DD : in this document a general description of the architecture adopted to build Data4Help is presented. In particular the part of the Architectural Design was very important to have an overall view of the source code.

ITD : this document was used to check what are the features implemented, as well as to have a little more specific view of the source code and testing done with respect to the DD.

In the following chapters the problems arose when setting up the system and testing it will be presented with the addition of some advice to improve it. Indeed the scope of the testing and validation was not to break the system, but rather recognizing its main weaknesses and understand how those weaknesses could be avoided or at least limited.

1.2 Acronyms

- **DD**: Design Document
- **ITD**: Implementation and testing document.
- **RASD**: Requirements Analysis and Specifications Document.
- **TPC**: Third Party Customer.

1.3 Document structure

The ADT document is structured in the following way:

Chapter 1 presents the main focuses of the document.

Chapter 2 describes the issues found during the deployment of the system.

Chapter 3 analyzes the system behavior, verifying that the requirement implemented were successfully achieved and describing the issues or bugs found during the testing process.

Chapter 4 covers other aspects such as the quality of code, expressing some appreciations on the functionalities and criticizing some general aspects of the repository delivered.

Chapter 5 contains the tables showing the effort spent by each testing team member, in testing and writing the document.

Chapter 6 contains the references of the document.

2 Installation

During the Android apk installation we didn't encounter any problem, it can be easily installed both on a real device and on an emulator. Sadly, this was not the case for the server. Following the instructions provided in the ITD we were not able to deploy the application. When trying to import as a project in NetBeans the files extracted from the war, the IDE complains that we are trying to import compiled .class files and then fails to deploy the project. In any case, we didn't find in the war archive any glassfish-resource.xml that should automatically create and configure the Connections Pool as stated in the ITD.

In order to deploy the server we had to manually configure the JDBC Resource and Connections Pool by means of the Glassfish Administration Panel (contrary to what stated in the ITD and with the MySql Connector properly installed into Glassfish). We also deployed directly the war file from the Administration Panel, not without difficulties because it seems that at the first try Glassfish will ever throw some errors. Furthermore, if Glassfish is stopped and relaunched, the application, even if it seems correctly deployed, will not working anymore and requires to be undeployed and Glassfish Server need to be relaunched before trying to redeploy the application. The previously problems were encountered both with Glassfish 4.1.2 (the suggested version) and Glassfish 5.0.

3 Testing

3.1 Introduction

In this chapter, the system test cases performed are listed in first section. In the last two sections, all the problems found while testing the system will be presented. In more details they will be divided in two groups: Major issues and Minor issues, where the Major issues are problems that can't be fixed in a small amount of time and may lead to a general rethinking of the whole architecture of the system, while Minor issues are simple bugs that can be fixed in a small amount of time. Since the source code came with a sufficient amount of unit testing, the testing phase focused on black box testing: this technique while not being automated is one of the best for testing the general system behavior and stability.

3.2 System testing

The following test cases have been performed. If nothing is specified, the test case is to be assume positively performed.

- User and TPC signup.
- User and TPC login.
- User modifies its personal data (*with one exception for taxcode, see later*).
- TPC formulation of individual requests.
- User accept or refuse individual request.
- TPC visualization of individual data: *this task can sometimes fail causing the app to crash*.
- TPC formulation of group requests.
- TPC visualization of group data.
- TPC deletion of individual and group request.

In the application, the location of the user or the coordinates of his data is never displayed, neither in the user view nor in the TPC view. This basically misses the goals G1 and G3 as stated in the RASD.

3.3 Major Issues

The main major issue that came out from the testing phase derived from an architectural decision: the thick client with a local database and server with another persistent database. The advantage of this architecture is that the client doesn't require a permanent connection but on the other hand this architectural design is very dangerous and needs a lot of care since the changes between the two database have to be synchronized in order to avoid inconsistencies between the two. One very simple case is the following one: there are two App connected to the server and logged with the same account. One of the two changes the profile name: this change is registered by the server, but the App running on the phone where the profile was not changed doesn't update, creating inconsistency between its local DB and the server. Moreover, if the connection of the phone drops the data gathered from the disconnection to the reconnection is lost by the central server. Another case that can

	UserID	U_Name	Surname	FiscalCode	BirthDate	Gender
▶	1	pppp	aaddadada	dqwqdqwdwqq	2000-01-06	Female
*	NULL	NULL	NULL	NULL	NULL	NULL

(a) Profile stored in the server

(b) Profile stored in the phone

Figure 1: Problem of inconsistency

prove this behavior is changing the profile name, closing the app and then reopening it. For this case an example is provided below. Another major issue that was noticed while analyzing the code was the view handling: grouping together the recyclerView adapters of different fragments lead to some crashes during the test phase. Some more specific examples are provided in minor issues.

3.4 Minor issues

Here is a list of bugs that were noticed while testing the app, with the corresponding screenshots:

In the login page, the notification for unsuccessful login doesn't have text or displays raw data coming from an html document.

In the profile page of the individual's App section no check is made on the user input, meaning that an user could set a random series of number and characters for its name, surname, cellphone and so on. This bug was presented because in the Signup Form instead there was a rigid check on the TC.

In the wearable page of the individual's App, editing the wearable name caused a crash of the app. The fix for this issue is presented in picture

```
if (position >= 0)
    editGroup(position, name);
else
    editWearable(holder.mTextView.getText(), name);
holder.myCard.setName(name);
notifyItemChanged(holder.getAdapterPosition());
```

(a) Part of the code that caused the crash while editing the wearable name

```
if (fragment instanceof RequestsFragment)
    editGroup(position, name);
else if (fragment instanceof WearablesFragment)
    editWearable(holder.mTextView.getText(), name);
holder.myCard.setName(name);
notifyItemChanged(holder.getAdapterPosition());
```

(b) The fix implemented to make it work

Figure 2: Wearable edit name bug

In third parties requests page, if multiple requests are deleted, the app crashes.

The selection of countries for third parties and cities for individuals doesn't work

If a user tries to change its taxcode with a taxcode already present in the database, the application crashes.

Sometimes, when trying to visualize the data about an individual who has accepted the request, the application crashes. This happens with some specific individuals, apparently without any reason. For example, we observed this issue with the individual in Figure 5.

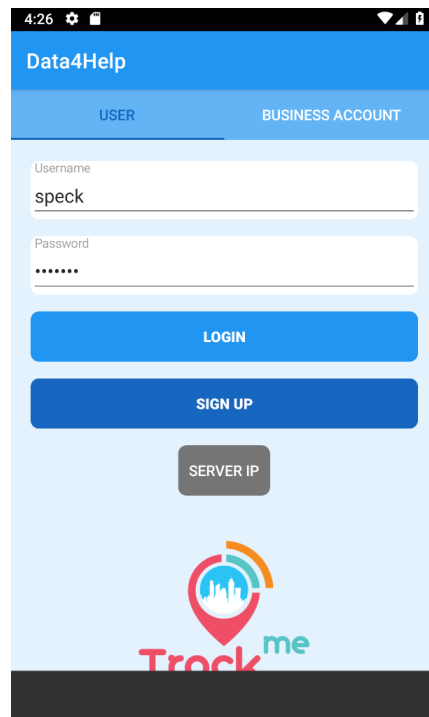
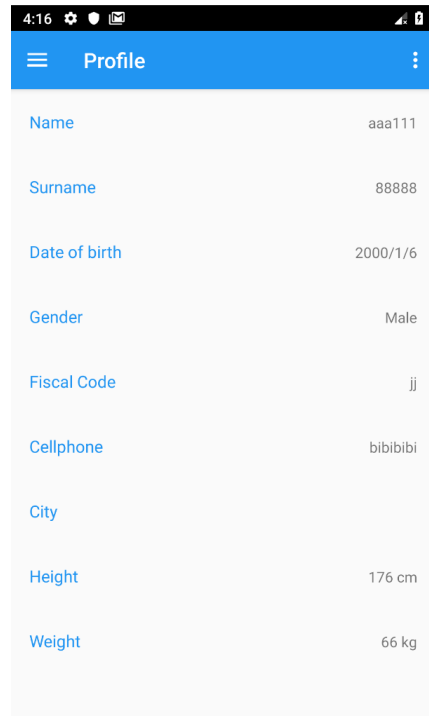


Figure 3: Grey notification bar in the login page

AccountID	Username	Password	e_Mail	AccountType
1	Ciao	okokok1	mai@mail.com	User

Figure 5: Individual who causes the crash



The screenshot shows a mobile application interface for a 'Profile' page. The status bar at the top indicates the time is 4:16. The page has a blue header with a menu icon, the title 'Profile', and a vertical ellipsis icon. Below the header, there is a list of profile fields, each with a label on the left and a value on the right. The values are: Name (aaa111), Surname (88888), Date of birth (2000/1/6), Gender (Male), Fiscal Code (jj), Cellphone (bibibibi), City (empty), Height (176 cm), and Weight (66 kg).

Field	Value
Name	aaa111
Surname	88888
Date of birth	2000/1/6
Gender	Male
Fiscal Code	jj
Cellphone	bibibibi
City	
Height	176 cm
Weight	66 kg

Figure 4: Example of profile input not checked

3.5 Suggestions

The DB could be modeled in a different way: some tables are useless, for example the userdailyactivities one that contains data which could be retrieved with simple queries. Another problem could be the fact that each eHealth data type coming from the individuals is only a column in the table userhealthstatus: the problem of doing this is poor upgradability in the case of inserting new eHealth data types, since all the previous tuples fields associated to the new data type should be set to null.

4 Other Aspects

4.1 Quality of Code

The structure of the code is well defined, although the code itself lacks some comments, making it not so easy to understand at a first glance.

The Javadoc while not being totally complete for methods and attributes, well describes the classes present in the code.

4.2 Github Repository

The organization of the Github Repository is a bit messy, a better folder managing would have been appreciated.

4.3 Data simulation

The graph representation of the simulated data is well done. The idea itself of simulating individual's data is a brilliant idea.

4.4 Sample database

The deliverable did not come along with any sample database. A small but realistic data collection would have been appreciated in order to be able to perform better and faster system black box testing.

5 Effort spent

In the following tables the time spent for each section of project testing is presented.

Stefano Pecchia

Section	Hours
Reading of provided documentation	4
Code review	4
Testing	13
Document Writing	5

Edoardo Peretti

Task	Hours
Reading of provided documentation	4
Code review	3
Testing	14
Document Writing	5

6 References

- Github Repository of the project analyzed: <https://github.com/Framonti/MontiNappi>