POLITECNICO
MILANO 1863

Software Engineering 2
A.Y. 2020-2021

CLup – Acceptance Testing Document

Vincenzo Riccio, Giancarlo Sorrentino, Emanuele Triuzzi
https://github.com/SirGian99/RiccioSorrentinoTriuzzi

Project Tested
https://github.com/AlessioBatta/BattagliaBelliniPasini

14th February, 2021

Deliverable:	ATD
Title:	Acceptance Testing Document
Authors:	Vincenzo Riccio, Giancarlo Sorrentino, Emanuele Triuzzi
Version:	1.0
Date:	14th February, 2021
Download page:	https://github.com/SirGian99/RiccioSorrentinoTriuzzi
Copyright:	

Contents

Chapter 1

Introduction

1.1 Purpose

The document purpose is to validate and test a project implemented by another group of the Software Engineering 2 course.

In particular, the authors of the analysed project are:

- Alessio Battaglia;
- Simone Bellini;
- Samuele Pasini.

Their repository is reachable through this URL:
https://github.com/AlessioBatta/BattagliaBelliniPasini.

1.2 Overview

The analysis of the project has been performed by accurately reading the documents produced by the team while paying attention to the consistency between them.

Then, we prepared a working configuration of our machines following the installation instructions included in the Implementation and Testing Document.

While doing so, it has been tracked any difficulty found.

Once the software was correctly configured, it has been properly examined by identifying which requirements it does satisfy and which does not. Moreover, the provided test cases have been run in order to evaluate the correctness of the implementation.

At the end, a full end to end test has been performed in order to test the general behaviour of the software, while looking at the use cases defined in the RASD and at the decisions taken in the DD.

1.3 Revision history

1.0 First version of the document. (14th February, 2021)

1.4 Reference documents

- Implementation and Testing assignment A.Y. 2020-2021;
- Teaching material provided by professors Matteo Rossi and Elisabetta Di Nitto;
- Requirement Analysis Specification Document, Design Document, Implementation and Testing Document of the team.

1.5 Document structure

The reference structure used for the document is an adapted version of the one suggested by professor Matteo Rossi of Politecnico of Milan.

Chapter 1

The first chapter provides an introduction to the document and introduces what has been done and evaluated in the project being tested.

Chapter 2

The second chapter describes how the installation setup has been made following the installation instructions included in the ITD.

Chapter 3

Chapter 3, which is the core of this document, focusses on acceptance tests: the software is tested with tests proposed by the group, by following test cases presented in the RASD, and by specific cases that are considered interesting. While doing so, also the performance of the tests are taken into account.

Chapter 4

The fourth chapter concludes the document summarizing and adding further comments about the work done by the group in question.

Chapter 5

Last chapter contains references to the tools and resources used to implement the system and to write the document.

Chapter 2

Installation setup

The installation guide provided in the ITD is quite detailed. We really appreciated the presence of figures describing the installation steps. However, it is very platform dependent and the part about Docker is incomplete.

For this reason, we had some troubles while preparing a Mac environment for running the whole software, while with Windows the guide is nearly perfect, with the already mentioned exception of the Docker part.

Thus, the following setup refers to the software installation on macOS.

2.1 Server installation

We started installing all the needed software to run the project. The first installed software was JDK 13. We downloaded the OpenJDK version from the Java JDK Archive website (included in the references), since the URL they provided redirects to JDK 15. We properly installed and configured it, making it available to the entire OS.

Since MySQL Server and Workbench were already installed on our machines, we only imported the CLup database dump. A little remark on that part of the guide: they do not specify a version to install, and we found some collation problems with MySQL 5.x.

Then we downloaded and installed Eclipse and all the plugins and dependencies mentioned in the ITD. The Enterprise version of Eclipse already includes Maven, so there was no need to configure it.

After opening the project, we edited the configuration file as stated in the guide in order to set the correct parameters allowing the connection to the MySQL DB instance.

However, in order to use the provided `.jar` artifact, the steps described in the ITD are not totally right. In fact, they suggest editing the `application.properties` file located in the jar's same folder as done in Eclipse, but it is not possible to directly execute the jar artifact without adding their same user to MySQL.

We ended the installation of the backend software by installing Docker. Here is where we found some difficulties: the installation guide is not so detailed about it and we initially found some troubles while running some containers-dependent tests.

2.2 Client installation

The mobile application required only Flutter and Android Studio in order to run it. We downloaded and installed them according to the official guides for macOS, and no problems were encountered. We configured them according to the guide they provided in the ITD, needing to adapt only the environment variables section for macOS (we followed the official Flutter installation guide for macOS, see references).

However, since Android Studio is quite demanding in terms of hardware specifications,we found some difficulties running the device simulator.

For this reason, an APK would have been really appreciated in order to test the app on a proper device instead of a slow simulator (they state that the server must run on the same machine, but with a proper configuration no problems would arise).

Chapter 3

Acceptance test

We started the acceptance test looking at the tests included in the implementation. We ran them many times to check they never fail. However, even if they suggest to never execute them near midnight, one test (`createTicketTest`) fails even at 19:00. Of course, it is just a little remark, since they offer a huge variety of tests.

3.1 Goals

G1	Limit the total number of customers in a shop to a certain amount	*Achieved*
G2	Prevent the formation of physical queues and gatherings outside	*"To prevent" is not possible, but beside that, achieved*
G3	Allow customers to enter when the number of people inside a shop is fewer than a certain amount	*Achieved*
G4	Allow the customers to enter when it is their turn	*Achieved*
G5	Block the customers' entrance when it is not their turn	*Achieved*
G6	Allow the customers to know when it is their turn	*Achieved*
G7	Allow customer to know the current situation of the queue before getting a ticket	*Achieved*
G8	Allow the user to manage booking visits to a supermarket at a certain time/day	*Achieved, but with the choice made, each customer can have at most one booking*

G9	Reduce the number of bookings without entrance	*Achieved*
G10	Let customers who do not have the application to still have an access	*Proxy not implemented*
G11	Allow the user to see if there are advantageous alternatives to their current choice	*Alternatives not implemented*

Table 3.1: Goals achieved

3.2 Requirements

R1	A customer should be allowed to sign up to use the CLup application	*Satisfied*
R2	A user should be allowed to sign in by inserting his access data (username and password)	*Satisfied*
R3	Users should be allowed to make an online booking	*Satisfied*
R4	CLup system should check if users already have a booking before giving them one	*Satisfied, but as a consequence it's not possible to let customers book more than one visit to a store at the time*
R5	CLup system should check if there are free slots before let the user makes a booking	*Satisfied*
R6	A user should be allowed to delete a previous booking	*Satisfied*
R7	If a user makes multiple bookings without going at the supermarket, the service should be temporarily deactivated	*Not implemented*
R8	Users should be allowed to take a virtual ticket	*Proxy not implemented*
R9	Customers should be allowed to take a physical ticket from the ticket generator outside the store	*Satisfied (only for app-customers)*

R10	CLup system should check if users already have a ticket before giving them one	*Satisfied*
R11	There should be a virtual lining up system for those who take a ticket	*Satisfied*
R12	A customer should be allowed to see the last called number	*Satisfied*
R13	A user should be allowed to see the approximate waiting time before getting a ticket	*Satisfied*
R14	Every shop should show the approximate waiting time at the entrance to all customers who wants to get a ticket	*Satisfied (only for app-customers)*
R15	CLup system should allow store manager to insert the parameters of his store when he adheres to the service	*Manager not implemented*
R16	The FMS should check if an entry code is associated to a booking or a ticket	*Satisfied*
R17	If the FMS status is CLOSED, the gates should be blocked for everyone	*Satisfied*
R18	If the FMS status is OPEN_FOR_BOOKINGS, the gates should be blocked for the customer with a ticket and unlocked for the one with a booking	*Satisfied*
R19	If the FMS status is OPEN, the gates should be unlocked for everyone with a valid ticket/booking	*Satisfied*
R20	For every entrance or exits, the status of the FMS should be recalculated	*Satisfied*
R21	CLup system should detect supermarkets near the chosen one with their relative queue status	*Alternatives not implemented*
R22	Store managers should be allowed to see the situation in their store, with trends, booking and queue status	*Manager not implemented*
R23	Store managers should be allowed to update their store parameters	*Manager not implemented*

Table 3.2: Requirements

The requirements which in the ITD they state to satisfy are really satisfied by the prototype. However, they do not provide any detail on the non-functional requirements satisfied (if there are).

3.3 Use case tests

In order to evaluate how to prototype satisfies the goals described in the Requirement Analysis Specification Document, we performed a deep end-to-end test to check whether any functions could exhibit any problem in its correct functioning.

The following table shows which of the use cases described in the RASD are foreseen by the prototype.

All the implemented use cases are fully working and coherent with the RASD.

Sign up	*Implemented*
Sign in	*Implemented*
Logout	*Implemented*
Take a ticket	*Implemented*
Select a supermarket to line-up	*Implemented*
Ask a suggestion for a line-up	*Not implemented*
Book a visit	*Implemented*
Select slots for booking	*Implemented*
Ask suggestion for a booking	*Not implemented*
Delete a booking	*Implemented*

Table 3.3: Use cases implementation

Regarding the product functions listed in the RASD, the first three (*Virtual ticket handling, Book a visit to a supermarket and Manage the flow of customers*) are planned in the ITD and correctly implemented in the prototype. Moreover, the *State diagram for app users* (*RASD, Figure 2*) is respected too.

3.4 Unit and Integration testing

To check the correctness of the implemented code, we also run all the tests included in the project. The following table underlines the execution of the tests on our machines, and their relative time of execution.

CLupApplicationTests	*0,322 s*
BookingRequestTest	*0,207 s*
SignupRequestTest	*0,230 s*
OpeningDayServiceTest	*0,362 s*
StoreServiceTest	*0,184 s*
SlotServiceTest	*0,053 s*
TicketServiceTest	*0,184 s*
BookingServiceTest	*0,145 s*
AuthServiceTest	*0,108 s*
DayServiceTest	*3,182 s*
BookingMapperTest	*0,248 s*
TicketMapperTest	*0,257 s*
StoreMapperTest	*3,673 s*
CityRepositoryTest	*0,341 s*
StoreRepositoryTest	*0,257 s*
RegionRepositoryTest	*0,115 s*
DayRepositoryTest	*0,083 s*
SlotRepositoryTest	*0,220 s*
UserRepositoryTest	*0,159 s*
ProvinceRepositoryTest	*0,106 s*
TicketRepositoryTest	*0,250 s*
OpeningDayRepositoryTest	*0,064 s*
BookingRepositoryTest	*0,139 s*

Table 3.4: Results of the tests performed using JUnit 5

Chapter 4

Conclusions and further comments

The group did a very good job, since as said before the prototype satisfies the main requirements and it is very complete. Furthermore, the mobile application prototype follows quite perfectly the mock-ups provided through the RASD and DD and it even includes remote notifications through Firebase.

The code quality is also very good, following nice design patterns, and coherent with the architecture presented in the Design Document. Furthermore, it is also sufficiently documented.

It is glaring that the group worked with a lot of commitment and competence. Also, the group showed great disposability whenever needed in order to solve the encountered problems, in particular during the installation phase.

However, these are some last remarks regarding the implemented functionalities:

- The first remark, that regards the entire project, is the absence of a feature specified in the assignment, which is the one that allows users to specify which kind of products they intend to buy when placing a booking, in order to better handle the influx of people in the store.

- The second remark regards the "easy to use" feature of the application. The application appears to be very intuitive. In fact, as shown in the picture below, its main functionalities are handly shown just after the sign in process.

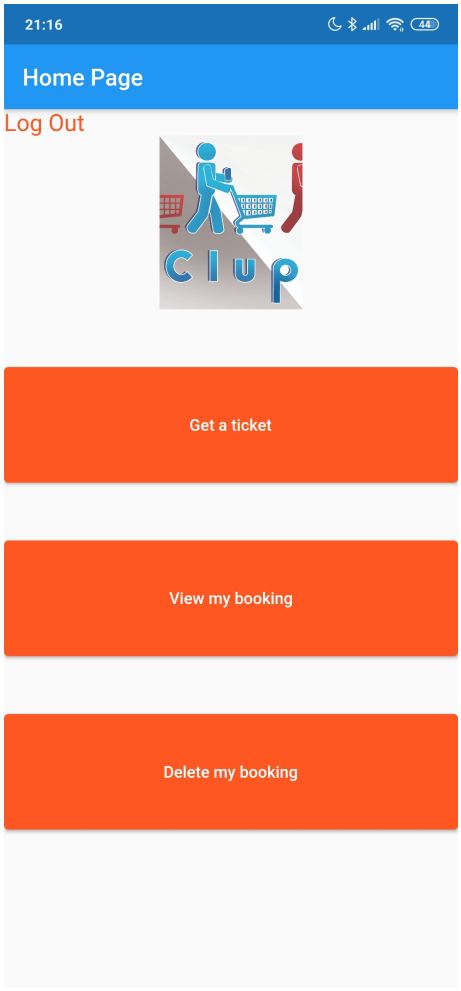

Figure 4.1: Home view

However, one remark on the application is the poor, in our opinion, handling of regions, provinces and cities. In fact, as shown in the picture below, it is quite unintuitive the selection of the previous specified fields when selecting a store.

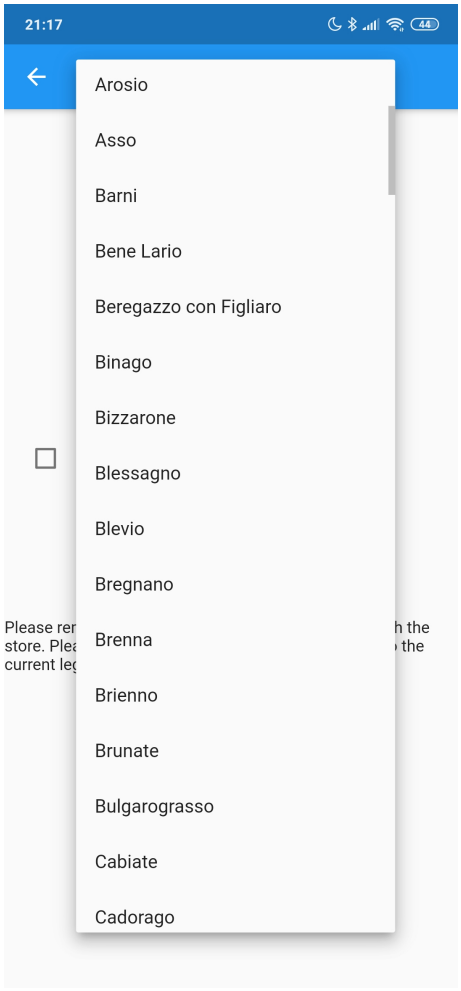

Figure 4.2: City selector

The application shows, without any logic, all the different possible values for the three fields, even if no store is available in the selected location: trying to find the available stores is indeed quite difficult among all the "useless" possibilities.

Furthermore, from a customer's point of view, it would have been great to know the working hours of a store before trying to place a booking or to get a ticket. It can be stressful finding a store among all the possible cities only to discover that it is currently closed.

- A last negative remark concerns the entity-relationship and logic shema. The proposed and implemented one is not so much scalable, since, for example, the table "day" should contain all the possible days in which a user can book a visit to a store. Taking into consideration the "test" database, it is impossible to book a visit to any store for a day after the 26/02/2021.

  To allow users to make a reservation to any store for any other day after the 26/02, those days should be manually inserted in the "day" table. This solution is quite "raw", and could have been better implemented without the need to create a "day" table.

Chapter 5

References

This chapter contains references to the tools and resources used during the writing of the document.

- *Java Development Kit*
  https://jdk.java.net/archive/
- *Eclipse*
  https://www.eclipse.org/downloads/packages/
- *Flutter installation on MacOS*
  https://flutter.dev/docs/get-started/install/macos
- *Apache Maven*
  https://maven.apache.org/guides/
- *Android Studio*
  https://developer.android.com/studio
- *Docker*
  https://www.docker.com/get-started
- *JUnit 5*
  https://junit.org/junit5/

List of Tables

List of Figures