AY 2020/2021

Politecnico di Milano

Acceptance Testing Document

Luca Pirovano Nicolò Sonnino

Professor
Matteo Rossi

Version 1.0
February 11, 2021

Contents

1 Tested Project

- **Authors**
  Robert Medvedec
  Toma Sikora
- **Repository URL**
  Link
- **Documents considered**
  - RASD: Requirements Analysis Specification Document;
  - DD: Design Document;
  - ITD: Implementation and Testing Document;

2 Installation

For the installation phase, we followed the *Installation Instructions* section in the ITD (6.0).

We have installed the two application artifacts (CLup and CLupSM) for Android operating system (.apk file) provided by the developers in the following environments:

- Google Pixel 3A:
  - **Platform:** emulated on the Android Virtual Device (AVD) environment, with SDK Version: 29.
  - **O.S.:** Android 10 (Q).
- OnePlus 6T:
  - **Platform:** physical device with SDK 29.
  - **O.S.:** Android 10 (Q).
- NVIDIA Shield Tablet:
  - **Platform:** physical device with SDK 26.
  - **O.S.:** Lineage O.S. 15.0 (Android Oreo).

Since there were neither iOS release and web GUI we only tested the Android artifact directly on the devices listed above and also debugged the project through the Android Studio Suite.

3 Acceptance Test Cases

Since the database was deployed on the Firebase platform, we could not access to it and consequently we could not perform tests on it. At the same time, we could not have any idea about how data was stored in the DBMS and how errors were handled.

3.1 Test Scenarios

3.1.1 Store Selection

i **Goal:** choose a specific store located in Pavia.

ii **Steps to reproduce:**

- Open the app.
- Tap *SELECT A STORE* button.
- Click the first text box and select the city *Pavia, Italy*.
- Click on the second text box and select the grocery shop.
- Select one of the addresses printed in the last text area.
- Tap *SELECT STORE*.

iii **Issues:**

- Selecting a field in the form produced no UI changes visible to the user. After a code inspection, we realized that the issue was related to the system theme. In fact, we had the dark mode enabled on each device. This produced a different UI color palette, with the consequence of white text on white background. An issue screenshot is shown in figure 1.
- Tapping on *SELECT STORE* without confirming the address leads to an application (or sometimes page) crash, reporting a runtime exception.

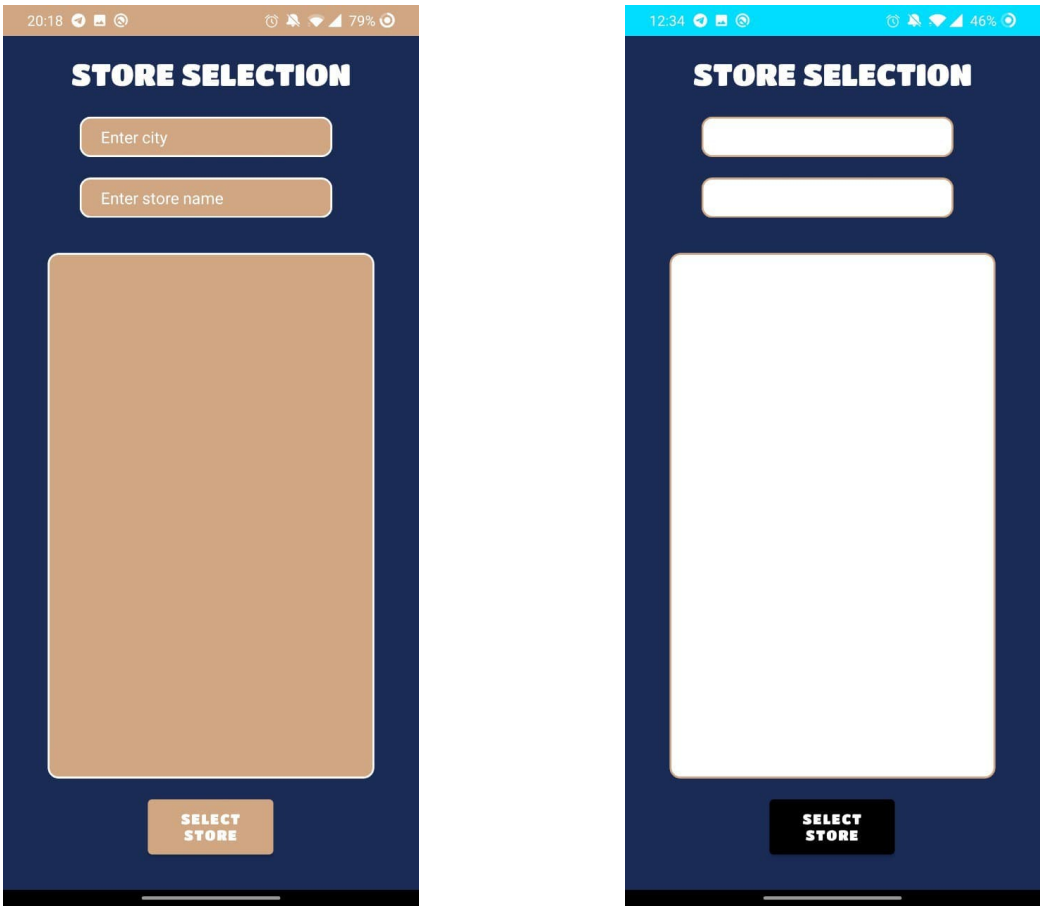

Figure 1: Store selection (expected vs. actual)

3.1.2 New Ticket Request

i **Goal:** retrieve a ticket at Minimarket located in Pavia.

ii **Steps to reproduce:**

- Select a store as described in paragraph 3.1.1
- Click on *REQUEST TICKET* button.
- Visualize the QR Code of the fresh new ticket.

iii **Test cases:**

(a) Use of *CHECK TICKET* button.

(b) Retrieve a ticket for a closed store.

iv **Test results:**

(a) If the ticket had been validated, the app returned a confirmation message. Otherwise, no actions were performed.

(b) The app correctly threw an error stating that the store was closed.

v **Issues:**

- Once a ticket has been retrieved, if the user closes the application or returns to previous page, the ticket is definitely lost and there is no way of recovering it.
  The main consequence of this behavior is that the ticket keeps its queue position, blocking all following tickets in the same slot. This issue makes the entire application useless for the affected slot.

3.1.3 Login as a Store Manager

i **Goal:** login using an existing Store Manager account.

ii **Steps to reproduce:**

- Open the application.
- Tap on *STORE MANAGER LOGIN* button.
- Insert credentials.

- Tap on *LOGIN* button.

iii **Test Cases:**

(a) Insertion of wrong credentials.

(b) Insertion of unconfirmed account credentials.

(c) Password reset (if forgotten).

(d) Tap the login button without credentials insertion.

(e) Account switching.

iv **Test results:**

(a) Correct throwing of a login error.

(b) Correct sending of confirmation email.

(c) Correct password reset flow.

(d) Correct throwing of a missing fields error.

(e) Correct logout (and successive login) of users.

3.1.4 Manage a store as Store Manager

i **Goal:** open or close an existent shop managed by our profile.

ii **Steps to reproduce:**

- Login as a store manager, as described in section 3.1.3
- Click on *OPEN STORE* or *CLOSE STORE*

iii **Test results:** the store correctly reacted to our requests, closing and opening itself. In case of closure, slots in that store became unavailable.

3.1.5 Customer Control

i **Goal:** scan tickets and monitor accesses.

ii **Steps to reproduce:**

- Login as a store manager, as described in section 3.1.3.
- Tap on *CUSTOMER CONTROL* button.

- Tap on *SCAN A TICKET* button.
- Read the QR Code using the device integrated camera and validate ticket.
- Tap on *STORE EXIT* button.

iii **Test Cases:**

(a) Scan a valid ticket.

(b) Scan an expired or lost ticket.

(c) Scan a ticket after reaching the maximum slot capacity.

(d) Scan a ticket retrieved immediately after a lost one.

iv **Test results:**

(a) The application correctly validated the ticket, giving also a confirmation message.

(b) In the case of an expired ticket, the application correctly displayed an error message. In the second case the application validated the ticket.

(c) The ticket was validated and the available slot counter went below zero, as shown in figure 2.

(d) The ticket was not validated.

v **Issues**

- Losing a ticket resulted in blocking the whole queue. Since the ticket was still valid but not scannable, every following booking was considered invalid, resulting in an unavailability of the slot.
- Scanning more tickets than the available store slots produced more accesses than the allowed ones, resulting in a negative value of the slots counter.

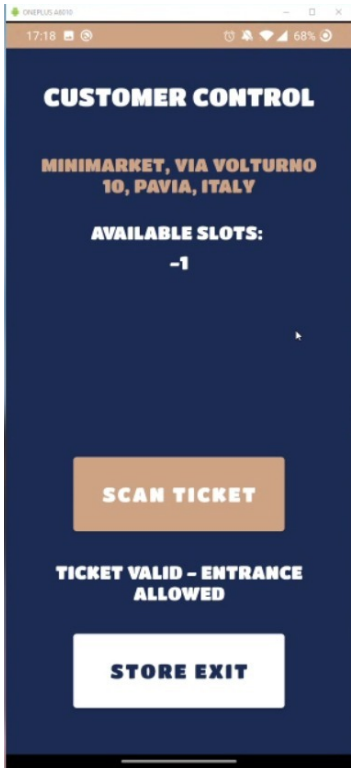

Figure 2: The counter became negative.

4 Project Inspection

4.1 Documentation Quality

4.1.1 RASD

Inside the Overall Description section (2.1.1) the figure used for describing the internal structure of the system is too technical. In fact, since the audience of this document is a possible CLup customer without any IT skills, concepts like database, API, mobile operating systems and so on are meaningless and may cause confusion for the document reader.

The choice of manually inserting each store manager into the database is a waste of resource for CLup System Administrators. Consider for example an entire medium-size city using CLup, for every store (about 50) there are a certain number of managers (realistically a minimum set of 50 people); the

system administrators will then need to manually create and maintain all these accounts, resulting in an enormous amount of work.

Finally, there is a lack of details in the use case diagrams, which are not covering all the use cases described before.

4.1.2 DD

Inside the Architectural Design Overview section (2.1) the application server is designed to communicate with the Google Maps API (in order to provide a map representation). In our opinion, linking these two components is unrealistic and resource consuming, because the map needs to be rendered directly on client machine. In a real world scenario, the client would directly contact the maps API without passing through the server.

In the deployment diagram, we could not understand why *Spring Boot - MySQL* is deployed into the Database Server. In fact, Spring is a complete Java Enterprise backend framework, which interfaces with the database server through several data access interfaces (JPA, MongoDB and so on).

Finally, the Implementation, Integration and Test Plan section is well done and very detailed. In fact, the precedence of components implementation follows a well defined logic that we think it is required in this kind of document.

4.1.3 ITD

The Adopted Development Frameworks and Languages section (3.1) is in our opinion very poor of information. In fact, the technology choices are not motivated; furthermore, there is a digression on the IDE used, which is not relevant to the section scope.

The structure of the technologies used, such as Kotlin and Firebase, is not explained. For example, we do not know the whole firebase interaction flow, Kotlin integration with Android and so on.

On the other hand, meaningless details, such as QR Code AES encryption (completely useless in our opinion), are explained and commented too much (even with random code snippet in certain points).

In the section 3.3 there is a tree description of an existent set of data in the Firebase DBMS. In our opinion this whole section is required in the Design Document instead of in the implementation one. In fact, if your project is realized in outsourcing, the assigned developer needs to know the

exact organization of the database; otherwise, the resulting product could be completely different from the designed one (in the Design Document the logical description of data is omitted).

The code structure section contains only a list of components/packages without any explanation about their utility and role. We expected at least the description of the application packages with their functionalities.

4.2 Architecture Quality

First of all, the architecture constraints and guidelines stated in the Design Documents has not been followed. In fact, the original three-tiered system became a two-tiered one, because of the total lack of an application server or any backend platform. Furthermore, the application and presentation layers both reside on the client machine.

The application interfaces directly with the Firebase DBMS system, which leads to a potential dangerous security breach; decompiling the APK (fairly easy job) lets everybody obtain firebase credential keys, google developer keys and other sensitive data.

In the current implementation is even possible for a malicious user to edit, insert or delete data from database simply making an API request.

4.3 Code Quality

During our testing phase, we faced several critical implementation issues, such as exception throwing (resulting in application crash) instead of error messages and lack of feedbacks on user actions.

The adoption of JUnit is completely pointless because it is never used as intended. In fact, unit testing means automatic model consistency test, which implies mocking up objects, runtime assertions and so on. The only type of tests made were simply some `System.out.println()` statements on several objects.

The tests are then explained with the usage of code snippets completely out of context (and also without any form of description or comment).

Finally, code commenting is not constant and, together with the total absence of code docs, make it very difficult to read.

5 Conclusions

After a complete analysis of the previous sections, we can surely assume that this implementation project was carried very hastily, often overlooking major considerations (some of them stated in the Design Document) which would have created a solid and secure system.

6 Effort Spent

Student	Time for Acceptance Testing
Luca Pirovano	10 hours
Nicolò Sonnino	10 hours